

**软件学院大作业报告**

目录

[一、需求分析 1](#_Toc32665)

[1.1总体陈述 1](#_Toc17778)

[1.2需求规定 2](#_Toc11722)

[1.2.1总体需求 2](#_Toc16908)

[1.2.2功能需求 2](#_Toc27798)

[1.2.3功能分析 2](#_Toc2418)

[二、概要设计 3](#_Toc9759)

[2.1系统架构设计 3](#_Toc23623)

[2.2模块划分 4](#_Toc9078)

[三、系统实现 5](#_Toc9827)

[3.1分频模块 5](#_Toc5080)

[3.1.1设计原理 5](#_Toc10680)

[3.1.2代码 5](#_Toc19087)

[3.1.3将设计的模块电路编译成电子元件 6](#_Toc2324)

[3.2自动计时模块 6](#_Toc20760)

[3.2.1设计原理 7](#_Toc31302)

[3.2.2代码 7](#_Toc5950)

[3.2.3将设计的模块电路编译成电子元件 9](#_Toc25146)

[3.3手动调时模块 9](#_Toc6475)

[3.3.1设计原理 9](#_Toc29433)

[3.3.2代码 10](#_Toc21630)

[3.3.3将设计的模块电路编译成电子元件 12](#_Toc25271)

[3.4闹钟设置模块 12](#_Toc25695)

[3.4.1设计原理 12](#_Toc19547)

[3.4.2代码 13](#_Toc14265)

[3.4.3将设计的模块电路编译成电子元件 15](#_Toc3356)

[3.5闹钟响应模块 15](#_Toc562)

[3.5.1设计原理 15](#_Toc20517)

[3.5.2代码 15](#_Toc30686)

[3.5.3将设计的模块电路编译成电子元件 16](#_Toc16492)

[3.6整点报时模块 17](#_Toc2630)

[3.6.1设计原理 17](#_Toc23063)

[3.6.2代码 17](#_Toc23219)

[3.6.3将设计的模块电路编译成电子元件 18](#_Toc24611)

[3.7秒表计数模块 18](#_Toc29628)

[3.7.1设计原理 20](#_Toc11373)8

[3.7.2代码](#_Toc15734) 18

[3.7.3将设计的模块电路编译成电子元件](#_Toc27008) 20

[3.8调时选择模块(设置调时、分还是秒) 20](#_Toc24616)

[3.8.1设计原理 20](#_Toc11373)

[3.8.2代码 20](#_Toc15734)

[3.8.3将设计的模块电路编译成电子元件 21](#_Toc27008)

[3.9设置最终显示结果模块 22](#_Toc17887)

[3.9.1设计原理 22](#_Toc21886)

[3.9.2代码 22](#_Toc9693)

[3.9.3将设计的模块电路编译成电子元件 24](#_Toc11969)

[3.10显示模块 25](#_Toc32513)

[3.10.1设计原理 25](#_Toc686)

[3.10.2代码 25](#_Toc15055)

[3.10.3将设计的模块电路编译成电子元件 27](#_Toc10148)

[3.11按键消抖模块 27](#_Toc17772)

[3.11.1设计原理 28](#_Toc4973)

[3.11.2代码 28](#_Toc30412)

[3.11.3将设计的模块电路编译成电子元件 29](#_Toc18294)

[3.12顶层模块 30](#_Toc13459)

[3.12.1设计原理 30](#_Toc28369)

[3.12.2代码 30](#_Toc27656)

[3.12.3将设计的模块电路编译成电子元件 35](#_Toc7803)

[四、 系统测试 35](#_Toc16140)

[4.1Modelsim仿真测试 35](#_Toc21261)

[4.1.1测试自动计时、整点报时、闹钟响应功能 35](#_Toc7576)

[4.1.2秒表功能 37](#_Toc13119)

[4.2FPGA实物运行结果测试 38](#_Toc14243)

[4.2.1正常计时实物图 38](#_Toc10973)

[4.2.2手动调时实物图 39](#_Toc7857)

[4.2.3设置闹钟实物图 40](#_Toc233)

[4.2.4秒表计数实物图 41](#_Toc30549)

[五、 总结 42](#_Toc12712)

[六、参考文献 43](#_Toc14555)

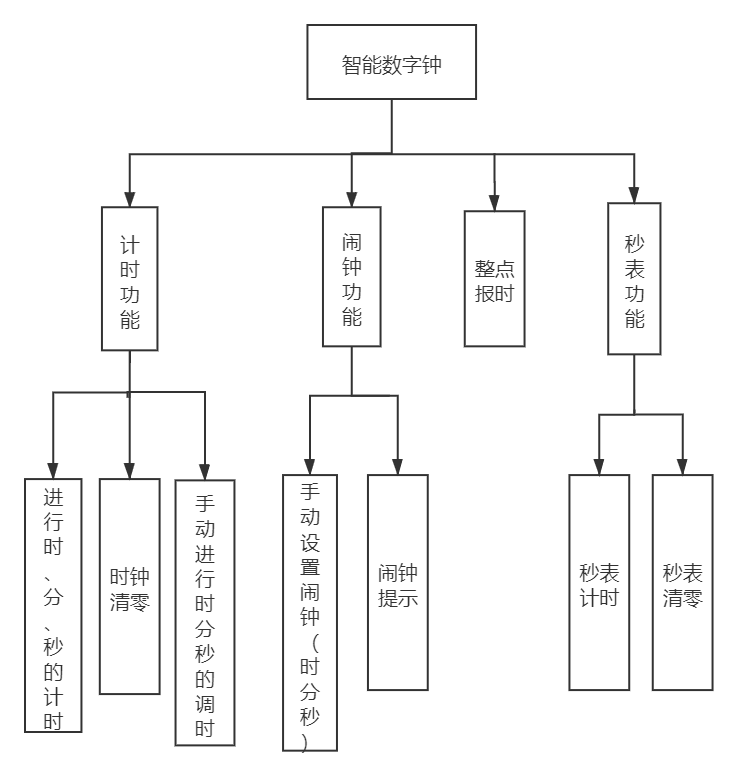
一、需求分析

1.1总体陈述

在老师所给的十个大作业选题中，我选择了智能数字钟。对于智能数字钟而言，它需要有哪些功能呢？首先是最基本的显示时间，既然有显示时间，那就应该要可以调时间。除此之外，数字钟应该还应该有闹钟，所以我又给加上了闹钟功能，既可以设置闹钟又可以在到闹钟点的时候给出闹钟提示。还有，整点报时应该也要有。最后，数字秒表的功能也是智能数字钟应该有的功能。综上所述，我设计的智能数字钟的功能如下：

* 能进行时、分、秒计时的数字钟(由于实验室的FPGA的板子只有四个数码管，实际程序中时、分用数码管显示，秒用led灯显示，一秒闪一下)
* 该数字钟能时钟清零
* 该数字钟能对时、分、秒进行手动调时以校准时间
* 该数字钟能够手动设置闹钟(也是分别对时、分、秒进行设置)
* 该数字到闹钟设置的点后能够给出提示(由于不知道实验室中的FPGA的蜂鸣器的引脚，改用led灯进行提示，闹钟的led灯会在到闹钟点的时候亮是十五秒后自动关闭，和我们手机一样)
* 该数字钟能够整点报时(在到达整点的时候进行报时，同闹钟一样，用led灯进行提示，整点报时的led灯在到达整点的时候亮灯十秒)
* 该数字钟能进行秒表功能，用户可以使用该数字钟进行秒表计时间(用四个数码管进行显示，最多9999秒，之后清零)
* 该数字钟的秒表功能也提供清零功能。(将相当于将秒表清零，用户可重新开始)

经整理后，我设计的智能数字钟整体功能图如下所示：



**1.2需求规定**

**1.2.1总体需求**

* 计时功能
* 闹钟功能
* 整点报时功能
* 秒表功能

1.2.2功能需求

* 能进行时、分、秒计时
* 能够时钟清零
* 能对时、分、秒进行手动调时
* 能够手动设置闹钟
* 能够给出闹钟提示
* 能够整点报时
* 能够进行秒表功能
* 秒表功能能够清零

1.2.3功能分析

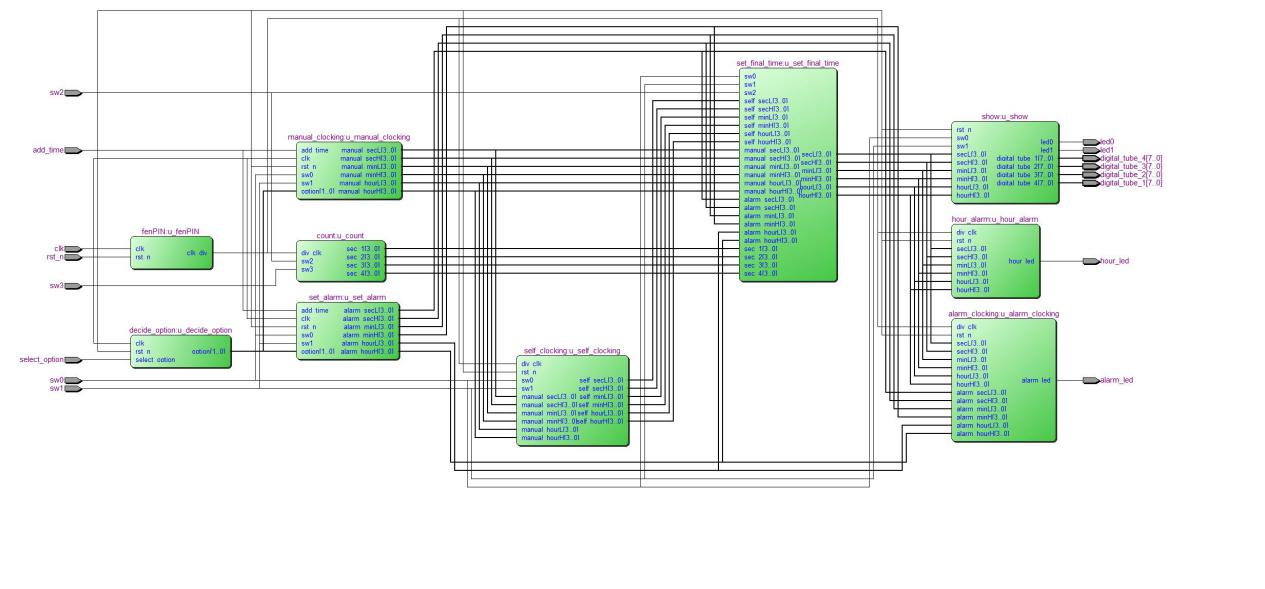
* 能进行时、分、秒计时：该程序下载到FPGA板子里后，会自动开始计时。该功能设想的是初始化时间为00:00:00。其中时、分通过四个数码管进行显示，秒通过led灯进行显示。自动计时功能不管是在用户设置闹钟还是进行秒表功能的时候都会进行后台计时。
* 能够时钟清零：只要用户按下重置按钮，系统会进行初始化，把当前的计时清为0(由于按下重置按钮相当于进行系统初始化，不建议单独用在时钟清零，因为这样也会让用户设置的闹钟也恢复初始化)
* 能对时、分、秒进行手动调时：用户可以通过将拨动开关sw2置为0，sw0置为1，sw1置为0进入到手动调时。通过按键key0设置你想调的是时、分还是秒，通过按键key1进行相应的调时，比如当用户设置调小时，每按下一次key1，小时加1，加到23后，再加就变为0。分钟和秒同理，但是由于实验室里的FPGA板子只有四个数码管，设置为led显示的秒在调整时不能很好地展示调秒的结果。(补充：在这里通过按键调时间需要进行按键消抖)
* 能够手动设置闹钟：用户可以通过将拨动开关sw2置为0，sw0置为1,sw1置为1进入闹钟设置。设置闹钟和手动调时一样的步骤。设置好后，将sw0置为1返回计时模块。
* 能够给出闹钟提示：当系统的时间到达设置的闹钟后，系统会给出提示，即将用于闹钟提示的led灯亮15秒(其他时间为灭)
* 能够整点报时：当系统时间到达整点的时候，系统会给出提示，即将用于整点报时提示的led灯亮10秒(其他时间为灭)
* 能够进行秒表功能：用户可以通过将拨动开关sw2置为1进入秒表模块，这时系统会立刻进行从0开始的秒表计时。除非用户将sw2置为0，退出该功能，或者将拨动开关sw3置为1，将秒表清零。
* 秒表功能能够清零：用户通过将拨动开关sw3置为1，可以将秒表清零(当然用户需要先将sw2置为1表示进入了秒表功能模块)。将sw3置为1后，除非用户重新将秒表置为0，否则秒表将一直为0，不再计时。

**二、概要设计**

**2.1系统架构设计**

为了实现程序的低耦合，我将系统分为了各个小的模块，再通过顶层模块对各个子模块进行调用，实现最终智能数字钟的功能。这样设计的好处是程序的耦合度很低，以后要进行别的项目的时候可以复用我们这个程序中的模块，避免重复工作。同时，这样的模块化也可以让程序更为简洁，可读性更好。在我们对程序进行扩展的时候，就比如我的秒表功能是最后加上去的，只需要编写秒表及时模块，分频器和显示模块都可以直接调用，很方便。在我们不想要某个功能的时候，只需要将该模块在顶层模块的调用移除就可以了，不需要其他的改动。

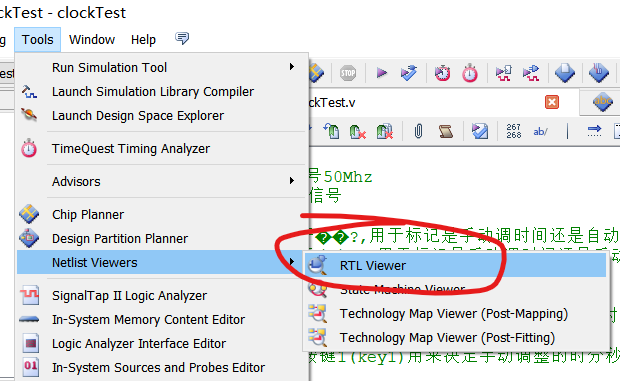
这里先给出我设计的智能数字钟的顶层模块由综合器生成的原理图，如下所示：



这个图这样看有点小，老师您可放大来看。具体的模块划分我在系统实现中会详细介绍。

附加：在Quartus II里查看综合器生成的原理图，这个问题在我答辩的时候夏老师问过我，要看我的顶层调用图。因为在写程序的时候，我自己对各个模块的划分很清楚，所以写的时候就没有想要看原理图，以至于当时夏老师问我的时候，我根本不知道该如何生成，后面我翻阅别人博客知道了答案。其实生成该原理图很简单：

在Quartus中选择Tools > Netlist viewer > RTL viewer就可以了。



2.2模块划分

对于该智能数字钟模块，我主要划分为十一个模块。分别是：

* 分频模块
* 自动计时模块
* 手动调时模块
* 闹钟设置模块
* 闹钟响应模块
* 整点报时模块
* 秒表计数模块
* 调时选择模块(设置调时、分还是秒)
* 设置最终显示结果模块(设置最终在数码管上显示的值)
* 显示模块
* 按键消抖模块

三、系统实现

**3.1分频模块**

**3.1.1设计原理**

分频器的设计目的是将FPGA中的50MHZ频率(20ns)分为1s的时钟信号。

它的实现思路是定义一个计数变量(cnt)用作计数器，在系统初始化的时候(即复位信号为0有效时)，将计数变量赋值为0，同时将分频后的输出时钟信号赋值为0。接下来，每来一个时钟信号的上升沿，计数变量(cnt)加1，当计数变量(cnt)不等于5000\_0000的时候，clk\_div等于0。当计数变量(cnt)等于5000\_0000，clk\_div等于1,同时将计数变量重新赋值为0，重新开始新一轮。

我将分频器设计为单独的一个模块，它的输入为系统时钟(clk)和系统复位信号(rst\_n),输出为分频后的信号。

3.1.2代码

module fenPIN(

input clk, //系统时钟,实验室里面的板子50Mhz，相当于20ns一次时钟信号

input rst\_n, //系统复位，低电平有效

output reg clk\_div //分频后的输出信号，1s

);

reg [25:0] cnt; //5000\_0000的二位数26位，故设置为26位寄存器

always @ (posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

cnt <= 26'd0;

clk\_div <= 1'd0;

end

else begin

if(cnt == 26'd5000\_0000) begin

clk\_div <= 1'b1;//5000\_0000次clk上升沿clk\_div置为1

cnt <= 26'd0;//cnt重新置为0

end

else begin

clk\_div <= 1'b0;//除了5000\_0000次上升沿clk\_div置为1外，其他时间置为0

cnt <= cnt + 1'b1;//每来一个clk，cnt加1

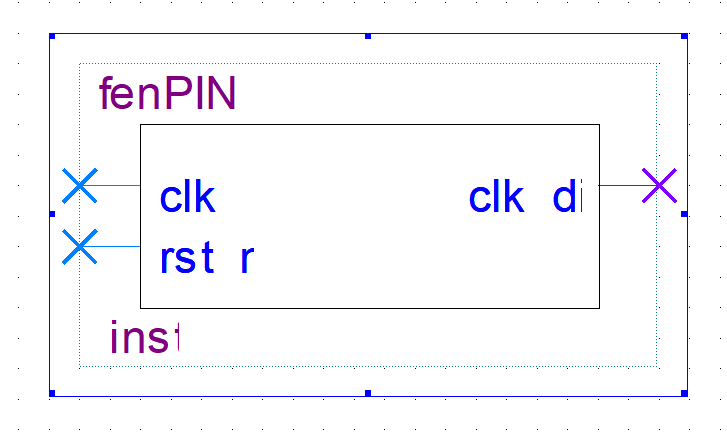
end

end

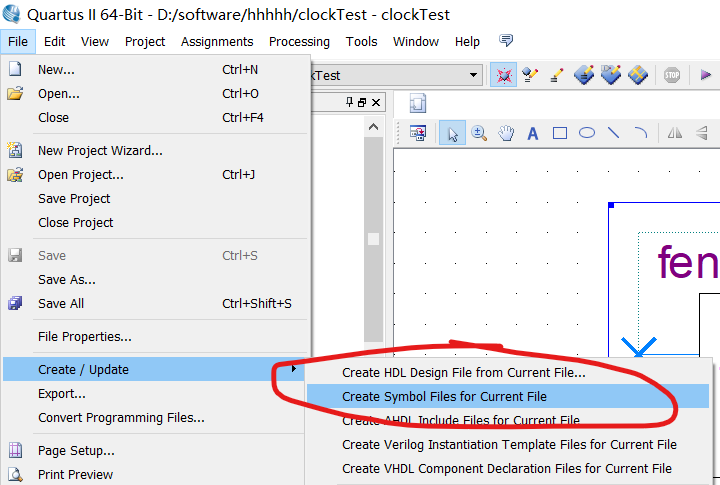
end

endmodule

**3.1.3将设计的模块电路编译成电子元件**



【说明】要在quartus中将设计的模块编译成电子元件图的步骤：将该模块编译仿真好后，选择File菜单下的Create/Uupdata子菜单下的Create [Symbol](https://www.baidu.com/s?wd=Symbol&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank) File for Current File即可。



点击即可自动生成，生成的文件一般在你项目的文件夹下。我自己生成的该文件就在我模块文件所在的文件夹的目录下。下面生成的电子元件图将不再赘述该步骤。

**3.2自动计时模块**

**3.2.1设计原理**

自动计时的目的是实现数字钟的时、分、秒的自动计时。

它的实现思路是每来一个分频后的信号的上升沿，时间加1。时间的进位就和我们普通时钟一样，60秒进一分钟，60分钟进一小时，到23:59:59的时候，清零重新开始。因为这个智能数字钟还涉及用户的手动调时，用户手动调时后，需要系统的时间继承调时的1时间，所以我设计的自动计时的模块的输入信号还有手动调时间的时间值(时分秒高低位)，如果系统处于手动调时的状态(即sw0=0,sw2=0),系统时间继承调时时间。最后将系统时间输出，由顶层模块支配调用。(因为该模块是独立运行的，当系统进入闹钟模块或者秒表模块，系统时钟仍然正常运行，只是不在数码管显示而已，当用户返回自动计时的时候，系统时间开始正常显示)

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.2.2代码**

module self\_clocking(

input div\_clk,//1s一次的分频好的时钟信号

input rst\_n,//复位信号

input sw0, //拨动开关sw0,用于标记是手动调时间还是自动计时：为1自动计时，为0手动调时

input sw1, //拨动开sw1，用于标记是手动调时间还是手动调闹钟

input wire [3:0] manual\_secL,//手动调时的秒的个位

input wire [3:0] manual\_secH,//手动调时的秒的十位

input wire [3:0] manual\_minL,//手动调时的分的个位

input wire [3:0] manual\_minH,//手动调时的分的十位

input wire [3:0] manual\_hourL,//手动调时的时的个位

input wire [3:0] manual\_hourH,//手动调时的时的十位

output reg [3:0] self\_secL,//自动计时的秒的个位

output reg [3:0] self\_secH,//自动计时的秒的十位

output reg [3:0] self\_minL,//自动计时的分的个位

output reg [3:0] self\_minH,//自动计时的分的十位

output reg [3:0] self\_hourL,//自动计时的时的个位

output reg [3:0] self\_hourH//自动计时的时的十位

);

always @ (posedge div\_clk or negedge rst\_n) begin

if(!rst\_n) begin

self\_secL <= 4'b0000;

self\_secH <= 4'b0000;

self\_minL <= 4'b0111;

self\_minH <= 4'b0101;

self\_hourL <= 4'b0011;

self\_hourH <= 4'b0010;

end

else begin

if(sw0 == 1 || (sw0==0 && sw1==1)) begin //如果sw0等于1，进入自动计时或者当设置闹钟的时候，自动计时还是继续运行

//如果到了末状态即23:59:59

if(self\_hourH==4'b0010 && self\_hourL==4'b0011 && self\_minH==4'b0101 && self\_minL==4'b1001 && self\_secH==4'b0101 && self\_secL==4'b1001) begin

self\_secL <= 4'b0000;

self\_secH <= 4'b0000;

self\_minL <= 4'b0000;

self\_minH <= 4'b0000;

self\_hourL <= 4'b0000;

self\_hourH <= 4'b0000;

end

//普通状态时计时

else begin

if(self\_secL==9) begin

self\_secL<=4'b0000;

if(self\_secH==5) begin

self\_secH<=4'b0000;

if(self\_minL==9) begin

self\_minL<=4'b0000;

if(self\_minH==5) begin

self\_minH<=4'b0000;

if(self\_hourH<2) begin

if(self\_hourL==9) begin

self\_hourL <= 4'b0000;

self\_hourH <= self\_hourH + 1'b1;

end

else

self\_hourL <= self\_hourL + 1'b1;

end

else

if(self\_hourL==3) begin

self\_hourL <= 4'b0000;

self\_hourH <= 4'b0000;

end

else

self\_hourL <= self\_hourL + 1'b1;

end

else

self\_minH<=self\_minH+1'b1;

end

else

self\_minL<=self\_minL+1'b1;

end

else

self\_secH<=self\_secH+1'b1;

end

else

self\_secL<=self\_secL+1'b1;

end

end

if(sw0==0 && sw1==1) begin//sw0==1代表进入手动模式，sw1=0代表是手动调时模式，sw1==1代表闹钟模式

//继承手动模式的结果

self\_secL <= manual\_secL;

self\_secH <= manual\_secH;

self\_minL <= manual\_minL;

self\_minH <= manual\_minH;

self\_hourL <= manual\_hourL;

self\_hourH <= manual\_hourH;

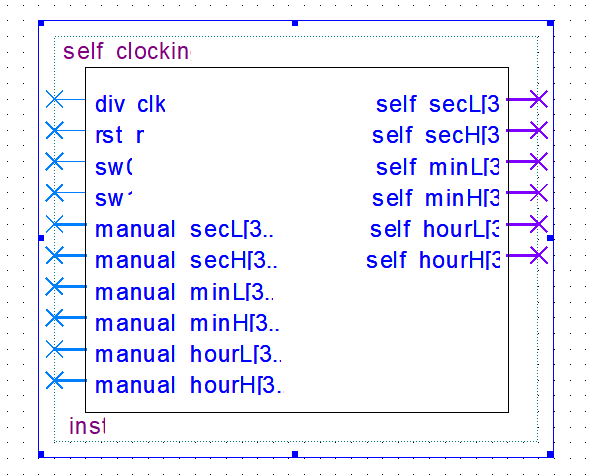
end

end

end

endmodule

**3.2.3将设计的模块电路编译成电子元件**



**3.3手动调时模块**

**3.3.1设计原理**

手动调时的目的是支持用户对该智能数字钟进行时、分、秒的调时以校准时间。

它的实现思路是将通过设置sw0=0,sw1=1进入手动调时模块，通过输入信号option实现判断是调时、分还是秒(option的设置产生在调时选择模块实现)，通过按键add\_time时间的调整，比如设置调小时，每按一下add\_time按键，小时加1，当小时加到23的时候，再加就清零重新开始。分钟和秒原理相同，只不过是满60进1清零。时间设置好之后，将设置好的时间(时分秒的高低位)输出，由顶层模块支配调用。(因为涉及按键操作，这里对按键操作进行了消抖，消抖在消抖模块进行介绍)

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.3.2代码**

module manual\_clocking(

input rst\_n,//复位信号

input clk,//时钟信号

input sw0, //拨动开关sw1,用于标记是手动调时间还是自动计时：为1自动计时，为0手动调时��?

input sw1, //拨动开sw0，用于标记是手动调时间还是手动调闹钟

input add\_time,//按键1(key1)用来决定手动调整的时分秒的大小，每按下一次按键，加1

input wire [1:0] option,//option等于0表示调秒、等于1表示调分、等于2表示等于调时

output reg [3:0] manual\_secL,//手动调时的秒的个位

output reg [3:0] manual\_secH,//手动调时的秒的十位

output reg [3:0] manual\_minL,//手动调时的分的个位

output reg [3:0] manual\_minH,//手动调时的分的十位

output reg [3:0] manual\_hourL,//手动调时的时的个位

output reg [3:0] manual\_hourH//手动调时的时的十位

);

wire key\_value;//按键防抖的后键值

wire key\_flag;//按键防抖后的标志

key\_debounce u\_key\_debounce(

.clk(clk),

.rst\_n(rst\_n),

.key(add\_time),

.key\_value(key\_value),

.key\_flag(key\_flag)

);

always @ (posedge clk or negedge rst\_n) begin

if(!rst\_n) begin //初始位

manual\_secL <= 4'b0000;

manual\_secH <= 4'b0000;

manual\_minL <= 4'b0000;

manual\_minH <= 4'b0000;

manual\_hourL <= 4'b0000;

manual\_hourH <= 4'b0000;

end

else begin

if(key\_flag && (~key\_value)) begin

if(sw0==0 && sw1==1) begin //sw=0表示开启手动模式，sw1=1表示开始手动调时

if(option == 2'd0) begin

if(manual\_secL==9) begin

manual\_secL <= 4'b0000;

if(manual\_secH==5)

manual\_secH <= 4'b0000;

else

manual\_secH <= manual\_secH + 1'b1;

end

else

manual\_secL <= manual\_secL + 1'b1;

end

if(option == 2'd1) begin

if(manual\_minL==9) begin

manual\_minL <= 4'b0000;

if(manual\_minH==5)

manual\_minH <= 4'b0000;

else

manual\_minH <= manual\_minH + 1'b1;

end

else

manual\_minL <= manual\_minL + 1'b1;

end

if(option == 2'd2) begin

if(manual\_hourH<2) begin //如果小时的十位小于2

if(manual\_hourL==9) begin

manual\_hourL <= 4'b0000;

manual\_hourH <= manual\_hourH + 1'b1;

end

else

manual\_hourL <= manual\_hourL + 1'b1;

end

else begin

if(manual\_hourL==3) begin

manual\_hourL <= 4'b0000;

manual\_hourH <= 4'b0000;

end

else

manual\_hourL <= manual\_hourL + 1'b1;

end

end

end

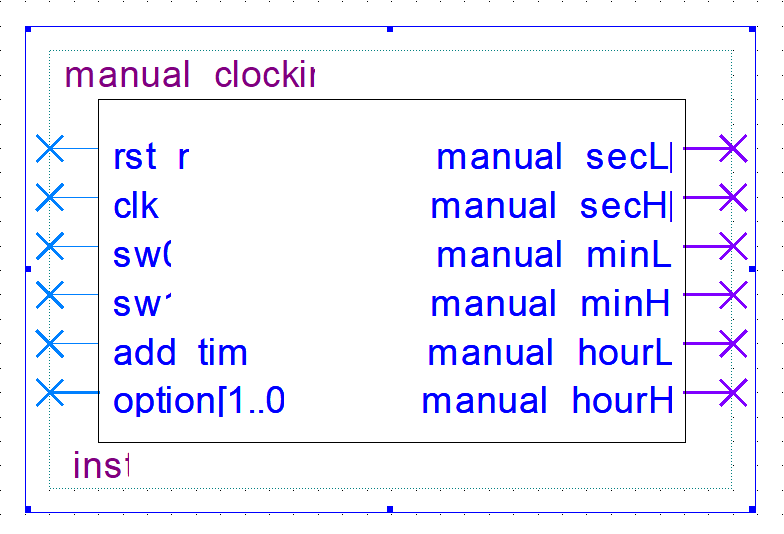
end

end

end

endmodule

**3.3.3将设计的模块电路编译成电子元件**



**3.4闹钟设置模块**

**3.4.1设计原理**

闹钟设置的目的是支持用户对该智能数字钟进行时、分、秒的闹钟设置以实现闹钟功能。

它的实现思路同手动调时差不多，将通过设置sw0=0,sw1=0进入闹钟设置模块，通过输入信号option实现判断是调时、分还是秒(option的设置产生在调时选择模块实现)，通过按键add\_time时间的调整，比如设置调小时，每按一下add\_time按键，小时加1，当小时加到23的时候，再加就清零重新开始。分钟和秒原理相同，只不过是满60进1清零。闹钟设置好之后，将设置好的闹钟时间(时分秒的高低位)输出，由顶层模块支配调用。(因为涉及按键操作，这里对按键操作进行了消抖，消抖在消抖模块进行介绍)

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.4.2代码**

/\*\*

设置闹钟模块

\*/

module set\_alarm(

input rst\_n,//复位信号

input clk,

input sw0, //拨动开关sw0,用于标记是手动调时间还是自动计时：为1自动计时，为0手动调时

input sw1, //拨动开关sw1，用于标记是手动调时间还是手动调闹钟

input add\_time,//按键1(key1)用来决定手动调整的时分秒的大小，每按下一次按键，加1

input wire [1:0] option,//option等于0表示调秒、等于1表示调分、等于2表示等于调小时

output reg [3:0] alarm\_secL, //调闹钟的秒的个位

output reg [3:0] alarm\_secH, //调闹钟的秒的十位

output reg [3:0] alarm\_minL, //调闹钟的分的个位

output reg [3:0] alarm\_minH, //调闹钟的分的十位

output reg [3:0] alarm\_hourL, //调闹钟的时的个位

output reg [3:0] alarm\_hourH //调闹钟的时的十位

);

wire key\_value;//按键防抖的后键值

wire key\_flag;//按键防抖后的标志

key\_debounce u\_key\_debounce(

.clk(clk),

.rst\_n(rst\_n),

.key(add\_time),

.key\_value(key\_value),

.key\_flag(key\_flag)

);

always @ (posedge add\_time or negedge rst\_n) begin

if(!rst\_n) begin //初始化

alarm\_secL <= 4'b0000;

alarm\_secH <= 4'b0000;

alarm\_minL <= 4'b0000;

alarm\_minH <= 4'b0000;

alarm\_hourL <= 4'b0000;

alarm\_hourH <= 4'b0000;

end

else begin

if(key\_flag && (~key\_value)) begin

if(sw0==0 && sw1==0) begin//sw=0表示开启手动模式，sw1=0表示开始设置闹��?

if(option == 2'd0) begin

if(alarm\_secL==9) begin

alarm\_secL <= 4'b0000;

if(alarm\_secH==5)

alarm\_secH <= 4'b0000;

else

alarm\_secH <= alarm\_secH + 1'b1;

end

else

alarm\_secL <= alarm\_secL + 1'b1;

end

if(option == 2'd1) begin

if(alarm\_minL==9) begin

alarm\_minL <= 4'b0000;

if(alarm\_minH==5)

alarm\_minH <= 4'b0000;

else

alarm\_minH <= alarm\_minH + 1'b1;

end

else

alarm\_minL <= alarm\_minL + 1'b1;

end

if(option == 2'd2) begin

if(alarm\_hourH<2) begin //如果小时的十位小于2

if(alarm\_hourL==9) begin

alarm\_hourL <= 4'b0000;

alarm\_hourH <= alarm\_hourH + 1'b1;

end

else

alarm\_hourL <= alarm\_hourL + 1'b1;

end

else begin

if(alarm\_hourL==3) begin

alarm\_hourL <= 4'b0000;

alarm\_hourH <= 4'b0000;

end

else

alarm\_hourL <= alarm\_hourL + 1'b1;

end

end

end

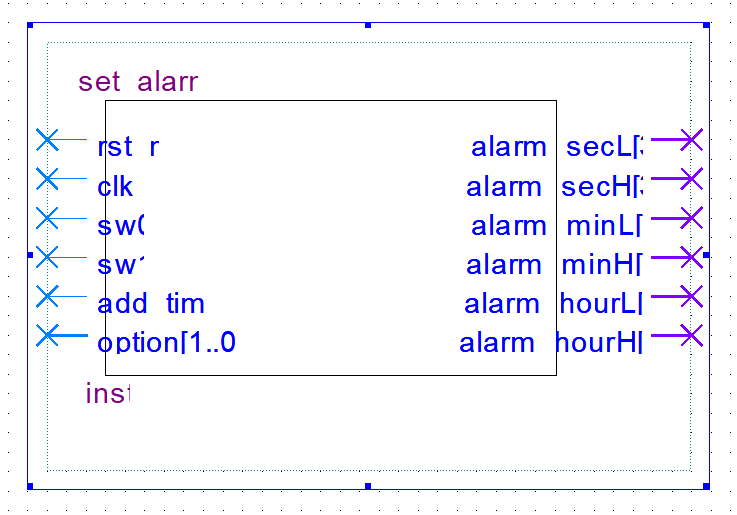
end

end

end

endmodule

**3.4.3将设计的模块电路编译成电子元件**



**3.5闹钟响应模块**

**3.5.1设计原理**

闹钟响应的目的是为了在系统时间到达闹钟设置的时间后，系统能给出提示。(由于我在夏老师给的FPGA引脚文档中并没有找到蜂鸣器的引脚，改用led灯进行提示，设计为到达闹钟设置时间后，闹钟提示的led灯亮灯15秒。亮灯十五秒的实现思路和分频器一样，也是设计一个变量(led\_ctrl)用来计数，当到达闹钟时间后，led灯置0，led\_ctrl置为0，以后每来一个分频后时钟信号，led\_ctrl加1，加到14的时候led灯置为0，如此往复就可以了)

它的实现思路很简单，只需要判断闹钟时间是否和系统时间相等就可以了，相等即触发闹钟。闹钟提示如上面所提及的思路所示。

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.5.2代码**

module alarm\_clocking(

input div\_clk,//1s一次的分频好的时钟信号

input rst\_n,//复位信号

input wire [3:0] secL,//最终的秒的个位

input wire [3:0] secH,//最终的秒的十位

input wire [3:0] minL,//最终的分的个位

input wire [3:0] minH,//最终的分的十位

input wire [3:0] hourL,//最终的时的个位

input wire [3:0] hourH,//最终的时的十位

input wire [3:0] alarm\_secL,//调闹钟的秒的个位

input wire [3:0] alarm\_secH,//调闹钟的秒的十位

input wire [3:0] alarm\_minL,//调闹钟的分的个位

input wire [3:0] alarm\_minH,//调闹钟的分的十位

input wire [3:0] alarm\_hourL,//调闹钟的时的个位

input wire [3:0] alarm\_hourH,//调闹钟的时的十位

output reg alarm\_led//设置led3为触发闹钟的标志，当到达设置的闹钟点的时，led3闪烁15秒

);

reg [4:0] led\_ctrl;

always @ (posedge div\_clk or negedge rst\_n) begin

if(!rst\_n) begin

alarm\_led<=1'b1;//设初始化为灭灯

led\_ctrl <= 1'b0;

end

else begin

//当当前时间的小时高低位、分钟高低位、秒的高位等于s设置的闹钟的时候，持续亮灯

if(secL==alarm\_secL && secH==alarm\_secH && minL==alarm\_minL && minH==alarm\_minH && hourL==alarm\_hourL && hourH==alarm\_hourH) begin

alarm\_led<=1'b0;//设置为常亮

led\_ctrl <= 1'b0;

end

else begin

if(alarm\_led == 1'b0) begin

led\_ctrl <= led\_ctrl + 1'b1;

if(led\_ctrl == 4'b1110) begin

alarm\_led<=1'b1;//设置为长灭

end

end

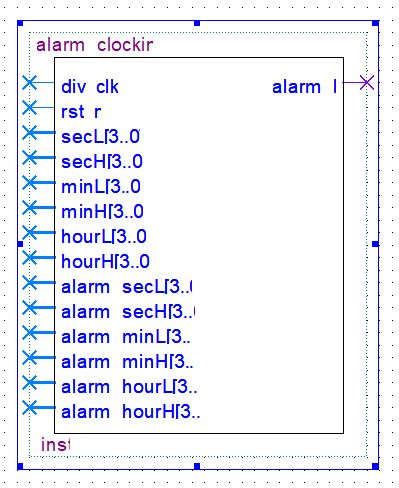
end

end

end

Endmodule

**3.5.3将设计的模块电路编译成电子元件**



**3.6整点报时模块**

**3.6.1设计原理**

整点报时的目的是为了在系统时间到达整点后后，系统能给出提示。(同闹钟一样，由于我在夏老师给的FPGA引脚文档中并没有找到蜂鸣器的引脚，改用led灯进行提示，设计为到达整点后，整点报时提示的led灯亮灯10秒。亮灯十五秒的实现思路很简单，只需要判读秒的个位小于10就可以了，当秒的个位小于10，led灯一直置为0，否则置为1)

它的实现思路很简单，只需要判断系统时间是否是整点就可以了，是整点即触发整点报时。整点报时提示如上面所提及的思路所示。

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.6.2代码**

module hour\_alarm(

input div\_clk,//1s一次的分频好的时钟信号

input rst\_n,//复位信号

input wire [3:0] secL,//最终的秒的个位

input wire [3:0] secH,//最终的秒的十位

input wire [3:0] minL,//最终的分的个位

input wire [3:0] minH,//最终的分的十位

input wire [3:0] hourL,//最终的时的个位

input wire [3:0] hourH,//最终的时的十位

output reg hour\_led//设置led2为整点报时的标志，led2闪烁10秒

);

always @ (posedge div\_clk or negedge rst\_n) begin

if(!rst\_n)

hour\_led <= 1'b1;//设初始化为灭灯

else begin

if(minH==5 && minL==9 && secH==5 && secL==9)

hour\_led <= 1'b0;//设置为常亮

else if(minH==0 && minL==0 && secH==0 && secL<10)

hour\_led <= 1'b0;//设置为常亮

else

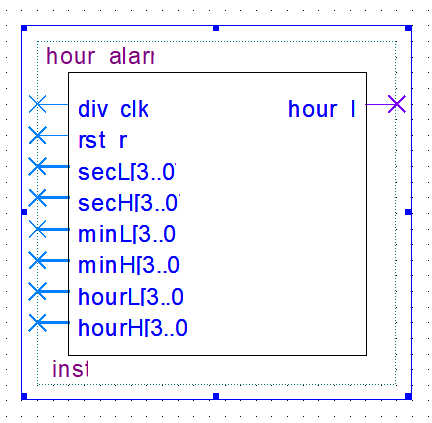
hour\_led <= 1'b1;//设置为长灭

end

end

endmodule

**3.6.3将设计的模块电路编译成电子元件**



**3.7秒表计数模块**

**3.7.1设计原理**

秒表计数的目的是实现秒表功能，即实现开始计时间后，秒从0开始计时，最多可计时9999秒(受限制于4个数码管显示)，满10000则清零。还提供手动清零的功能，以实现重新开始。

它的实现思路是将sw2置为1进入秒表计时模块，每来一个分频后的信号的上升沿，时间加1。时间的进位为逢十进一，到9999的时候，清零重新开始。同时这个秒表计数还涉及清零功能，清零功能只需要将sw3置为1就可以实现清零(注意sw3置为1后计数秒表一直为0，除非sw3重新置为1退出清零)。退出清零功能后秒表重新从0开始计数直达用户将sw2置为0退出秒表计数功能。不管清零还是秒表计数，最后都要将秒表时间输出，由顶层模块支配调用。

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.7.2代码**

module count(

input div\_clk,//1s一次的分频好的时钟信号

input sw2, //控制开始

input sw3,//控制清零

output reg [3:0] sec\_1,

output reg [3:0] sec\_2,

output reg [3:0] sec\_3,

output reg [3:0] sec\_4

);

always @ (div\_clk or sw3 or sw2) begin

if(sw3==1) begin //当sw3置零的时候，初始化为0

sec\_1 <= 4'b0000;

sec\_2 <= 4'b0000;

sec\_3 <= 4'b0000;

sec\_4 <= 4'b0000;

end

else begin

if(sw2==1) begin

if(sec\_1 == 9) begin

sec\_1 <= 4'b0000;

if(sec\_2 == 9) begin

sec\_2 <= 4'b0000;

if(sec\_3 == 9) begin

sec\_3 <= 4'b0000;

if(sec\_4 == 9) begin

sec\_4 <= 4'b0000;

end

else

sec\_4 <= sec\_4 + 1'b1;

end

else

sec\_3 <= sec\_3 + 1'b1;

end

else

sec\_2 <= sec\_2 + 1'b1;

end

else

sec\_1 <= sec\_1 + 1'b1;

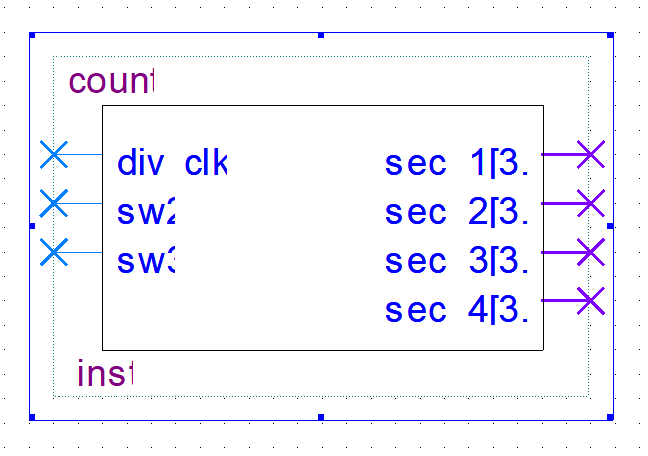
end

end

end

endmodule

**3.7.3将设计的模块电路编译成电子元件**



**3.8调时选择模块(设置调时、分还是秒)**

**3.8.1设计原理**

调时选择的目的是设置手动调时或者调闹钟的时候，是调时、分还是秒。

它的实现思路是定义一个调时选择的变量(option),通过按键0(select\_option)来控制该变量。每按一次按键0(select\_option),option加1。option变量在系统复位的时候初始化为0，它的取值范围是0，1，2。到2的时候再加就清零，重新开始。0代表调秒，1代表调分钟，2代表调小时。最后将option输出，由顶层模块支配调用。(因为涉及按键操作，这里对按键操作进行了消抖，消抖在消抖模块进行介绍)

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.8.2代码**

module decide\_option(

input rst\_n,//复位信号

input clk,//时钟信号

input select\_option,//按键0(key0)用来决定手动调时时候是调秒、分钟还是小时

output reg [1:0] option//option等于0表示调秒、等于1表示调分、等于2表示等于调时

);

wire key\_value;//按键防抖的后键值

wire key\_flag;//按键防抖后的标志

key\_debounce u\_key\_debounce(

.clk(clk),

.rst\_n(rst\_n),

.key(select\_option),

.key\_value(key\_value),

.key\_flag(key\_flag)

);

always @ (posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

option <= 2'd0;

end

else begin

if(key\_flag && (~key\_value)) begin

if(option == 2'd2)

option <= 1'b0;

else

option <= option + 1'b1;

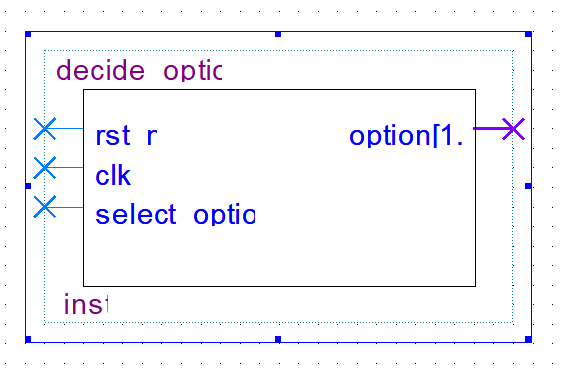
end

end

end

endmodule

**3.8.3将设计的模块电路编译成电子元件**



**3.9设置最终显示结果模块**

**3.9.1设计原理**

设置最终的显示结果的目的是通过判定系统当前进行的模块操作(比如自动计时、时钟清零、手动调时、设置闹钟、秒表计数、秒表清零等等)，将当前模块的操作数据设置为系统的最终显示结果。再传送出去，供顶层模块调用显示模块进行显示结果。

它的实现思路是同判断四个拨动开关(sw0,sw1,sw2,sw3)的值判断进行当前的状态，再将当前状态的值赋值给最终的显示结果。

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.9.2代码**

module set\_final\_time(

input sw0, //拨动开sw0,用于标记是手动调时间还是自动计时：为1自动计时，为0手动调时

input sw1, //拨动开sw1，用于标记是手动调时间还是手动调闹钟，为0手动调时间，为1手动调闹钟

input sw2,

input wire [3:0] self\_secL,//自动计时的秒的个位

input wire [3:0] self\_secH,//自动计时的秒的十位

input wire [3:0] self\_minL,//自动计时的分的个位

input wire [3:0] self\_minH,//自动计时的分的十位

input wire [3:0] self\_hourL,//自动计时的时的个位

input wire [3:0] self\_hourH,//自动计时的时的十位

input wire [3:0] manual\_secL,//手动调时的秒的个位

input wire [3:0] manual\_secH,//手动调时的秒的十位

input wire [3:0] manual\_minL,//手动调时的分的个位

input wire [3:0] manual\_minH,//手动调时的分的十位

input wire [3:0] manual\_hourL,//手动调时的时的个位

input wire [3:0] manual\_hourH,//手动调时的时的十位

input wire [3:0] alarm\_secL,//调闹钟的秒的个位

input wire [3:0] alarm\_secH,//调闹钟的秒的十位

input wire [3:0] alarm\_minL,//调闹钟的分的个位

input wire [3:0] alarm\_minH,//调闹钟的分的十位

input wire [3:0] alarm\_hourL,//调闹钟的时的个位

input wire [3:0] alarm\_hourH,//调闹钟的时的十位

input wire [3:0] sec\_1,

input wire [3:0] sec\_2,

input wire [3:0] sec\_3,

input wire [3:0] sec\_4,

output reg [3:0] secL,//最终的秒的个位

output reg [3:0] secH,//最终的秒的十位

output reg [3:0] minL,//最终的分的个位

output reg [3:0] minH,//最终的分的十位

output reg [3:0] hourL,//最终的时的个位

output reg [3:0] hourH//最终的时的十位

);

always @ (sw0 or sw1 or sw2 or sec\_1 or sec\_2 or sec\_3 or sec\_4 or self\_secL or self\_secH or self\_minL or self\_minH or self\_hourL or self\_hourH or manual\_secL or manual\_secH or manual\_minL or manual\_minH or manual\_hourL or manual\_hourH or alarm\_secL or alarm\_secH or alarm\_minL or alarm\_minH or alarm\_hourL or alarm\_hourH) begin

if(sw2==0) begin

if(sw0==1) begin

secL <= self\_secL;

secH <= self\_secH;

minL <= self\_minL;

minH <= self\_minH;

hourL <= self\_hourL;

hourH <= self\_hourH;

end

if(sw0==0 && sw1==1) begin

secL <= manual\_secL;

secH <= manual\_secH;

minL <= manual\_minL;

minH <= manual\_minH;

hourL <= manual\_hourL;

hourH <= manual\_hourH;

end

if(sw0==0 && sw1==0) begin

secL <= alarm\_secL;

secH <= alarm\_secH;

minL <= alarm\_minL;

minH <= alarm\_minH;

hourL <= alarm\_hourL;

hourH <= alarm\_hourH;

end

end

else begin

secL <= 1'b1;//灭灯

secH <= 1'b1;//灭灯

minL <= sec\_1;

minH <= sec\_2;

hourL <= sec\_3;

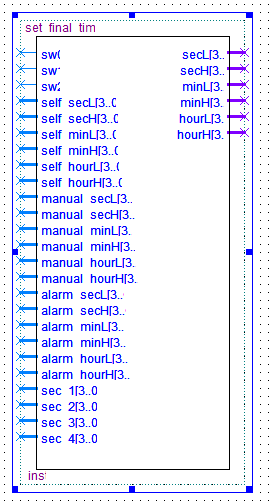
hourH <= sec\_4;

end

end

endmodule

**3.9.3将设计的模块电路编译成电子元件**



**3.10显示模块**

**3.10.1设计原理**

显示模块的目的是将系统当前模块的数据结果同数码管和led灯显示出来。

它的实现思路是将系统当前的模块数据输入进来，在数据发生变化的时候，输出改变后的数据的值。这里主要涉及到led的亮灭控制和数码管的译码。

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.10.2代码**

module show(

input sw0, //拨动开sw0,用于标记是手动调时间还是自动计时

input sw1, //拨动开sw1，用于标记是手动调时间还是手动调闹钟

input rst\_n,//复位信号

input wire [3:0] secL,//最终的秒的个位

input wire [3:0] secH,//最终的秒的十位

input wire [3:0] minL,//最终的分的个位

input wire [3:0] minH,//最终的分的十位

input wire [3:0] hourL,//最终的时的个位

input wire [3:0] hourH,//最终的时的十位

output reg [7:0] digital\_tube\_1,//输出电子数码管

output reg [7:0] digital\_tube\_2,//输出电子数码管

output reg [7:0] digital\_tube\_3,//输出电子数码管

output reg [7:0] digital\_tube\_4,//输出电子数码管

output reg led0,//led1的输出端口

output reg led1//led0的输出端口

);

always @ (secL or rst\_n) begin

if(!rst\_n) begin

led0 <= 1'b1;//led0初始化状态为灭灯

led\_ctrl\_secL <= 1'b0;//初始化为0

end

else begin

led0 <= ~led0;

end

end

always @ (secH or rst\_n) begin

if(!rst\_n) begin

led1 <= 1'b1;//led0初始化状态为灭灯

led\_ctrl\_secH <= 1'b0;//初始化为0

end

else begin

led1 <= ~led1;

end

end

always @ (minL or rst\_n) begin

case(minL)

4'b0000:digital\_tube\_1<=8'b1100\_0000;//0

4'b0001:digital\_tube\_1<=8'b1111\_1001;//1

4'b0010:digital\_tube\_1<=8'b1010\_0100;//2

4'b0011:digital\_tube\_1<=8'b1011\_0000;//3

4'b0100:digital\_tube\_1<=8'b1001\_1001;//4

4'b0101:digital\_tube\_1<=8'b1001\_0010;//5

4'b0110:digital\_tube\_1<=8'b1000\_0010;//6

4'b0111:digital\_tube\_1<=8'b1111\_1000;//7

4'b1000:digital\_tube\_1<=8'b1000\_0000;//8

4'b1001:digital\_tube\_1<=8'b1001\_0000;//9

default:digital\_tube\_1<=8'b1100\_0000;

endcase

end

always @ (minH or rst\_n) begin

case(minH)

4'b0000:digital\_tube\_2<=8'b1100\_0000;//0

4'b0001:digital\_tube\_2<=8'b1111\_1001;//1

4'b0010:digital\_tube\_2<=8'b1010\_0100;//2

4'b0011:digital\_tube\_2<=8'b1011\_0000;//3

4'b0100:digital\_tube\_2<=8'b1001\_1001;//4

4'b0101:digital\_tube\_2<=8'b1001\_0010;//5

4'b0110:digital\_tube\_2<=8'b1000\_0010;//6

4'b0111:digital\_tube\_2<=8'b1111\_1000;//7

4'b1000:digital\_tube\_2<=8'b1000\_0000;//8

4'b1001:digital\_tube\_2<=8'b1001\_0000;//9

default:digital\_tube\_2<=8'b1100\_0000;

endcase

end

always @ (hourL or rst\_n) begin

case(hourL)

4'b0000:digital\_tube\_3<=8'b1100\_0000;//0

4'b0001:digital\_tube\_3<=8'b1111\_1001;//1

4'b0010:digital\_tube\_3<=8'b1010\_0100;//2

4'b0011:digital\_tube\_3<=8'b1011\_0000;//3

4'b0100:digital\_tube\_3<=8'b1001\_1001;//4

4'b0101:digital\_tube\_3<=8'b1001\_0010;//5

4'b0110:digital\_tube\_3<=8'b1000\_0010;//6

4'b0111:digital\_tube\_3<=8'b1111\_1000;//7

4'b1000:digital\_tube\_3<=8'b1000\_0000;//8

4'b1001:digital\_tube\_3<=8'b1001\_0000;//9

default:digital\_tube\_3<=8'b1100\_0000;

endcase

end

always @ (hourH or rst\_n) begin

case(hourH)

4'b0000:digital\_tube\_4<=8'b1100\_0000;//0

4'b0001:digital\_tube\_4<=8'b1111\_1001;//1

4'b0010:digital\_tube\_4<=8'b1010\_0100;//2

4'b0011:digital\_tube\_4<=8'b1011\_0000;//3

4'b0100:digital\_tube\_4<=8'b1001\_1001;//4

4'b0101:digital\_tube\_4<=8'b1001\_0010;//5

4'b0110:digital\_tube\_4<=8'b1000\_0010;//6

4'b0111:digital\_tube\_4<=8'b1111\_1000;//7

4'b1000:digital\_tube\_4<=8'b1000\_0000;//8

4'b1001:digital\_tube\_4<=8'b1001\_0000;//9

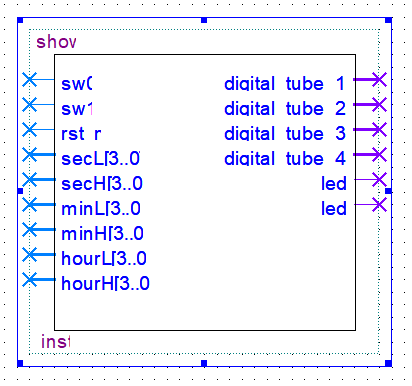
default:digital\_tube\_4<=8'b1100\_0000;

endcase

end

endmodule

**3.10.3将设计的模块电路编译成电子元件**



**3.11按键消抖模块**

**3.11.1设计原理**

按键消抖模块的目的是对按键的输入进行消抖，因为在实际按键的时候，会出现抖动，就比如我们按下按键key0，key0的值从1变为0再变为1。这个过程在我们仿真的时候时只有一个下降沿和一个上升沿，实际上，在实体按键的时候，它会出现很多小的上升沿和下降沿，而在我们程序中是捕捉按键的下降沿或者上升沿的，这个时候如果我们不对这些抖动进行消抖，则会出现我们意想不到的效果。就比如在这个程序中的手动调时的操作，不进行消抖的话，可能用户按一下按键就会增加几个或者几十个值。

它的实现思路主要有两种，一种是在按键后20ms再采取脉冲值，一般认为，在按键按下的20ms后，抖动已经结束。还有一种是某个状态保持了20ms，比如按键key0的值为1保持了20ms，我们则认为抖动已经结束。这里，我采用的是第二种方式。 我定义了一个变量存储消抖后的信号值，一个变量标志消抖是否结束。当一个信号来临的时候，如果保持了20ms不变，则将该信号赋值给消抖后的信号，并将消抖结束的值标记为1。最后，我们将消抖后的信号的值和消抖结束的标志输出。这个程序中用到消抖模块的地方主要是手动调时和设置闹钟以及调试选择模块。

该模块的输入和输入如下面代码所示(代码上已对变量做出解释，这里不再赘述)。

**3.11.2代码**

/\*\*

按键消抖模块

按键消抖模块实现思想：

如果一个信号进入之后，

保持了20ms不变，我们则认为它是消抖后稳定的信号

\*/

module key\_debounce(

input clk,

input rst\_n,

input key,

output reg key\_value,

output reg key\_flag

);

reg key\_reg;//定义寄存器寄存信号

reg [19:0] delay\_cnt;//设置延迟标志

always @ (posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

key\_reg <= 1'b1;//初始化为1

delay\_cnt <= 20'd0;

end

else begin

key\_reg <= key;//将按键的值寄存到寄存器中

if(key != key\_reg)

delay\_cnt <= 20'd1000\_000;

else begin

if(delay\_cnt >20'd0)

delay\_cnt <= delay\_cnt - 1'b1;

else

delay\_cnt <= 20'd0;

end

end

end

always @ (posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

key\_value <= 1'b1;

key\_flag <= 1'b0;

end

else begin

if(delay\_cnt == 10'd1) begin

key\_flag <= 1'b1;

key\_value <= key;

end

else begin

key\_flag <= 1'b0;

key\_value <= key\_value;

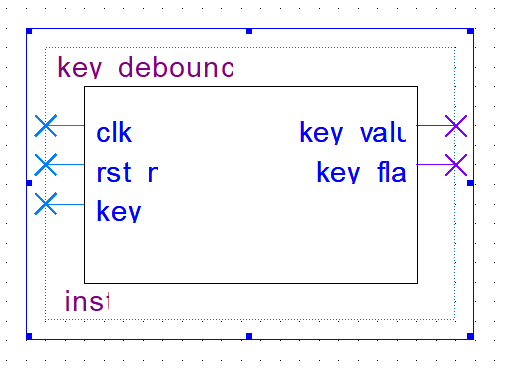
end

end

end

endmodule

**3.11.3将设计的模块电路编译成电子元件**



**3.12顶层模块**

**3.12.1设计原理**

顶层模块的目的是通过组合调用各个子模块，组成最终的智能数字钟程序。

它的实现思路是调用各个子模块就可以实现功能。但是我们需要定义关联各个子模块之间的数据传输的变量。就比如我们在分频模块分频后的时钟信号clk\_div,我们需要在顶层模块定义一个wire型的变量clk\_div，将分频后的时钟信号输入到clk\_div,之后需要用到clk\_div的子模块只需要将clk\_div作为输入信号引用到该模块就可以了。

总得来说，采用顶层模块调用子模块的这种程序设计方式不仅让程序的层次逻辑更强，程序的耦合度更低，而且在多人开发程序的时候(当然该项目不大，不需要多人开发)，也利于分工和子模块调试。

下面是顶层模块的总体代码。在代码中，我对每个模块的调用都做了注释，每个wire型的全局变量也做了解释。这里不再赘述。

**3.12.2代码**

module clockTest(

input clk,//时钟信号50Mhz

input rst\_n,//复位信号

input sw0,

input sw1,

input sw2,

input sw3,

input select\_option,

input add\_time,

output alarm\_led,

output hour\_led,

output wire [7:0] digital\_tube\_1,

output wire [7:0] digital\_tube\_2,

output wire [7:0] digital\_tube\_3,

output wire [7:0] digital\_tube\_4,

output led0,

output led1

);

wire clk\_div;

wire [3:0] manual\_secL;//手动调时的秒的个位

wire [3:0] manual\_secH;//手动调时的秒的十位

wire [3:0] manual\_minL;//手动调时的分的个位

wire [3:0] manual\_minH;//手动调时的分的十位

wire [3:0] manual\_hourL;//手动调时的时的个位

wire [3:0] manual\_hourH;//手动调时的时的十位

wire [3:0] self\_secL;//自动计时的秒的个位

wire [3:0] self\_secH;//自动计时的秒的十位

wire [3:0] self\_minL;//自动计时的分的个位

wire [3:0] self\_minH;//自动计时的分的十位

wire [3:0] self\_hourL;//自动计时的时的个位

wire [3:0] self\_hourH;//自动计时的时的十位

wire [3:0] alarm\_secL;//调闹钟的秒的个位

wire [3:0] alarm\_secH;//调闹钟的秒的十位

wire [3:0] alarm\_minL;//调闹钟的分的个位

wire [3:0] alarm\_minH;//调闹钟的分的十位

wire [3:0] alarm\_hourL;//调闹钟的时的个位

wire [3:0] alarm\_hourH;//调闹钟的时的十位

wire [3:0] secL;//最终的秒的个位

wire [3:0] secH;//最终的秒的十位

wire [3:0] minL;//最终的分的个位

wire [3:0] minH;//最终的分的十位

wire [3:0] hourL;//最终的时的个位

wire [3:0] hourH;//最终的时的十位

wire [3:0] sec\_1;//秒表的个位

wire [3:0] sec\_2;//十位

wire [3:0] sec\_3;//百位

wire [3:0] sec\_4;//千位

wire [1:0] option;

//分频模块

fenPIN u\_fenPIN(

.clk (clk),

.rst\_n (rst\_n),

.clk\_div (clk\_div)

);

//自动计时模块

self\_clocking u\_self\_clocking(

.div\_clk (clk\_div),

.rst\_n(rst\_n),

.sw0(sw0),

.sw1(sw1),

.manual\_secL(manual\_secL),

.manual\_secH(manual\_secH),

.manual\_minL(manual\_minL),

.manual\_minH(manual\_minH),

.manual\_hourL(manual\_hourL),

.manual\_hourH(manual\_hourH),

.self\_secL(self\_secL),

.self\_secH(self\_secH),

.self\_minL(self\_minL),

.self\_minH(self\_minH),

.self\_hourL(self\_hourL),

.self\_hourH(self\_hourH)

);

//手动调时模块

manual\_clocking u\_manual\_clocking(

.rst\_n(rst\_n),

.clk(clk),

.sw0(sw0),

.sw1(sw1),

.add\_time(add\_time),

.option(option),

.manual\_secL(manual\_secL),

.manual\_secH(manual\_secH),

.manual\_minL(manual\_minL),

.manual\_minH(manual\_minH),

.manual\_hourL(manual\_hourL),

.manual\_hourH(manual\_hourH)

);

//设置闹钟模块

set\_alarm u\_set\_alarm(

.rst\_n(rst\_n),

.clk(clk),

.sw0(sw0),

.sw1(sw1),

.add\_time(add\_time),

.option(option),

.alarm\_secL(alarm\_secL),

.alarm\_secH(alarm\_secH),

.alarm\_minL(alarm\_minL),

.alarm\_minH(alarm\_minH),

.alarm\_hourL(alarm\_hourL),

.alarm\_hourH(alarm\_hourH)

);

//闹钟响应模块

alarm\_clocking u\_alarm\_clocking(

.div\_clk (clk\_div),

.rst\_n(rst\_n),

.secL(secL),

.secH(secH),

.minL(minL),

.minH(minH),

.hourL(hourL),

.hourH(hourH),

.alarm\_secL(alarm\_secL),

.alarm\_secH(alarm\_secH),

.alarm\_minL(alarm\_minL),

.alarm\_minH(alarm\_minH),

.alarm\_hourL(alarm\_hourL),

.alarm\_hourH(alarm\_hourH),

.alarm\_led(alarm\_led)

);

//设置调时分秒的模块

decide\_option u\_decide\_option(

.rst\_n(rst\_n),

.clk(clk),

.select\_option(select\_option),

.option(option)

);

//秒表模块

count u\_count(

.div\_clk(clk\_div),

.sw2(sw2),

.sw3(sw3),

.sec\_1(sec\_1),

.sec\_2(sec\_2),

.sec\_3(sec\_3),

.sec\_4(sec\_4)

);

//设置最终的数据模块

set\_final\_time u\_set\_final\_time(

.sw0(sw0),

.sw1(sw1),

.sw2(sw2),

.self\_secL(self\_secL),

.self\_secH(self\_secH),

.self\_minL(self\_minL),

.self\_minH(self\_minH),

.self\_hourL(self\_hourL),

.self\_hourH(self\_hourH),

.manual\_secL(manual\_secL),

.manual\_secH(manual\_secH),

.manual\_minL(manual\_minL),

.manual\_minH(manual\_minH),

.manual\_hourL(manual\_hourL),

.manual\_hourH(manual\_hourH),

.alarm\_secL(alarm\_secL),

.alarm\_secH(alarm\_secH),

.alarm\_minL(alarm\_minL),

.alarm\_minH(alarm\_minH),

.alarm\_hourL(alarm\_hourL),

.alarm\_hourH(alarm\_hourH),

.sec\_1(sec\_1),

.sec\_2(sec\_2),

.sec\_3(sec\_3),

.sec\_4(sec\_4),

.secL(secL),

.secH(secH),

.minL(minL),

.minH(minH),

.hourL(hourL),

.hourH(hourH)

);

//整点报时模块

hour\_alarm u\_hour\_alarm(

.div\_clk (clk\_div),

.rst\_n(rst\_n),

.secL(secL),

.secH(secH),

.minL(minL),

.minH(minH),

.hourL(hourL),

.hourH(hourH),

.hour\_led(hour\_led)

);

//最终的显示模块

show u\_show(

.sw0(sw0),

.sw1(sw1),

.rst\_n(rst\_n),

.secL(secL),

.secH(secH),

.minL(minL),

.minH(minH),

.hourL(hourL),

.hourH(hourH),

.digital\_tube\_1(digital\_tube\_4),

.digital\_tube\_2(digital\_tube\_3),

.digital\_tube\_3(digital\_tube\_2),

.digital\_tube\_4(digital\_tube\_1),

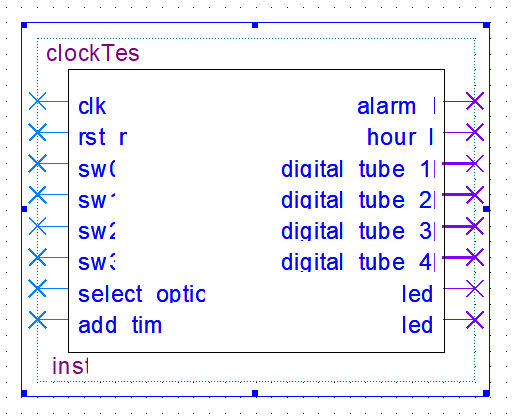
.led0(led0),

.led1(led1)

);

endmodule

**3.12.3将设计的模块电路编译成电子元件**



1. 系统测试

**4.1Modelsim仿真测试**

在完成程序设计后，我对该智能数字钟程序进行了测试，首先是在Modelsim中进行仿真，由于我设计了按键控制，所以设置闹钟模块和手动调时模块我就没有在Modelsim进行仿真，在Modelsim中只对自动计时，整点报时，闹钟响应，秒表功能进行了仿真测试。闹钟设置和手动调时我在实验室里的FPAG板子里进行了实物测试。

**4.1.1测试自动计时、整点报时、闹钟响应功能**

4.1.1.1编写激励文件

`timescale 1 ns/ 1 ns

module clockTest\_vlg\_tst();

reg add\_time;

reg clk;

reg rst\_n;

reg select\_option;

reg sw0;

reg sw1;

reg sw2;

reg sw3;

// wires

wire alarm\_led;

wire [7:0] digital\_tube\_1;

wire [7:0] digital\_tube\_2;

wire [7:0] digital\_tube\_3;

wire [7:0] digital\_tube\_4;

wire hour\_led;

wire led0;

wire led1;

clockTest i1 (

.add\_time(add\_time),

.alarm\_led(alarm\_led),

.clk(clk),

.digital\_tube\_1(digital\_tube\_1),

.digital\_tube\_2(digital\_tube\_2),

.digital\_tube\_3(digital\_tube\_3),

.digital\_tube\_4(digital\_tube\_4),

.hour\_led(hour\_led),

.led0(led0),

.led1(led1),

.rst\_n(rst\_n),

.select\_option(select\_option),

.sw0(sw0),

.sw1(sw1),

.sw2(sw2),

.sw3(sw3)

);

initial

begin

clk = 1'b0;

rst\_n = 1'b0;

add\_time = 1'b0;

#100 rst\_n = 1'b1;

sw3 = 1'b1;

sw2 = 1'b0;

sw0 = 1'b1;

sw1 = 1'b0;

select\_option = 1'b0;

#1000000 sw0 = 1'b0;

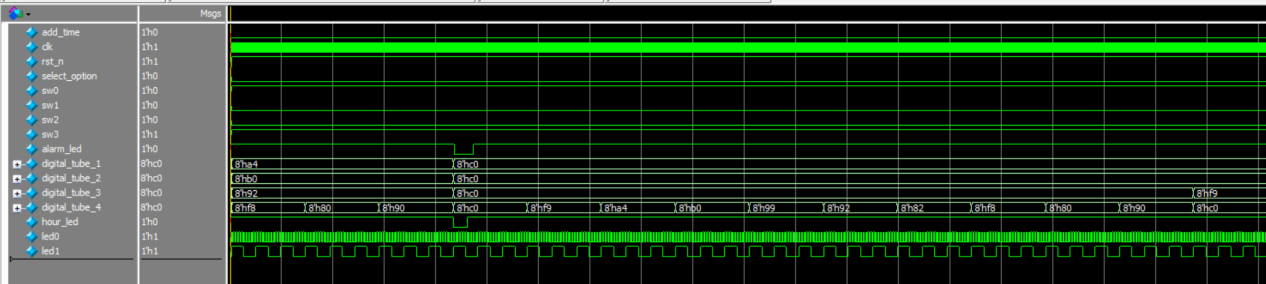
#1000000 $stop;

end

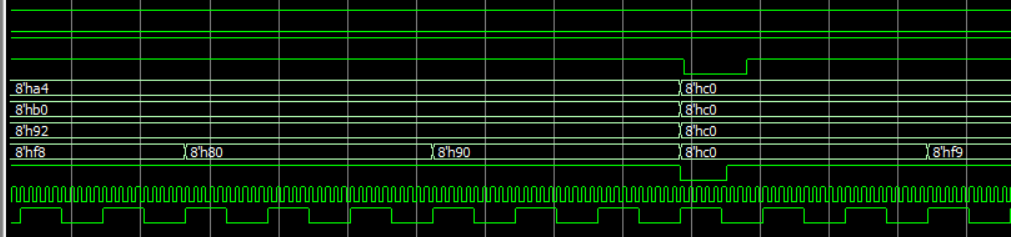
always #10 clk = ~clk;

endmodule

4.1.1.2仿真结果



细节放大图：



4.1.1.3仿真结果说明

这个仿真的时候，因为Modelsim中仿真中分频后1s的时钟信号在Modelsim中量度太大，我改为了100ns。同时我为了同时检查到23:59:59后时钟会不会清零，我将时钟的初始化的值置为23:57:00。可以看到，时钟在到达23:59:59后重新清零计时。说明这个功能没有问题。同时，在时间为00:00:00，整点报时的led灯hour\_led置零(亮灯)10个分频后的单位时间。说明整点报时无错。同时，在系统复位的时候，给闹钟赋初值为00:00:00，可以看到闹钟响应的那个led灯alarm\_led置零(亮灯)15个分频后的单位时间，说明闹钟响应功能无错。

**4.1.2秒表功能**

4.1.2.1编写激励文件

很多和上面的激励文件一样，这里只给出关键代码：

initial

begin

clk = 1'b0;

rst\_n = 1'b0;

add\_time = 1'b0;

#100 rst\_n = 1'b1;

sw3 = 1'b1;

sw2 = 1'b1;

sw0 = 1'b0;

sw1 = 1'b0;

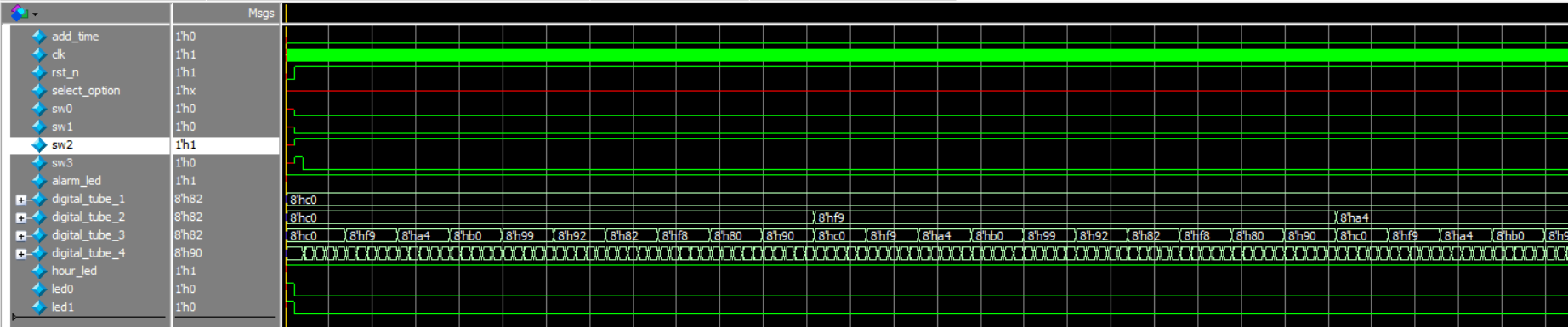
#100000 sw3 = 1'b0;

#1000000 $stop;

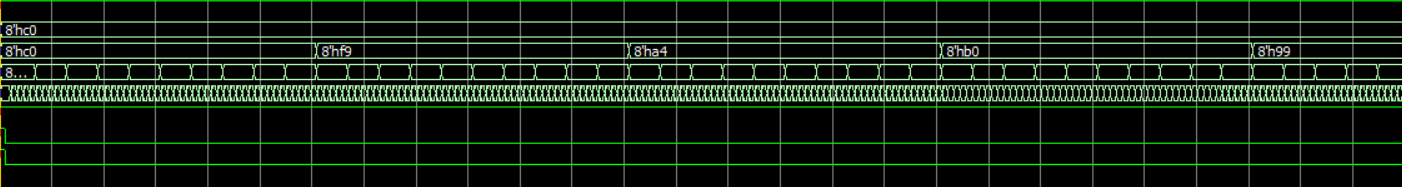
end

always #10 clk = ~clk;

4.1.2.2仿真结果



细节放大图：



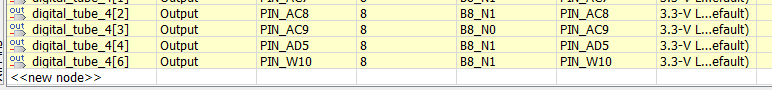
4.1.2.3仿真结果说明

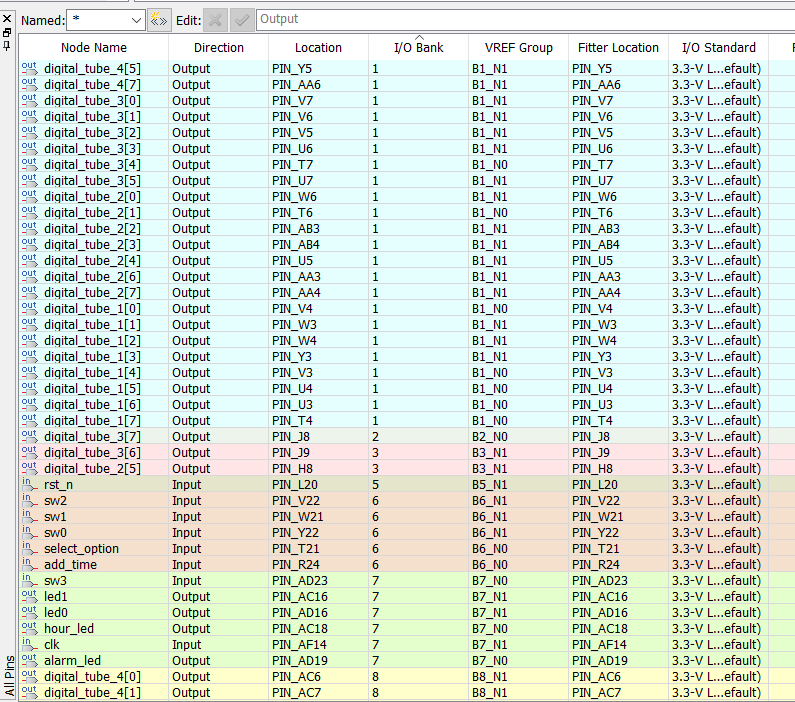
同样地，在进行这个仿真的时候我测试秒表功能是否能正常工作，可以看到，秒表的每个为位都是十进一，符合预期，说明秒表计时功能无错。

**4.2FPGA实物运行结果测试**

在将程序下载到FPGA板子里需要先配置好FPGA的引脚，配置好引脚后重新编译，再打开SignalTap II Logic Analyzer，将程序下载到fpga板子里就可以了。

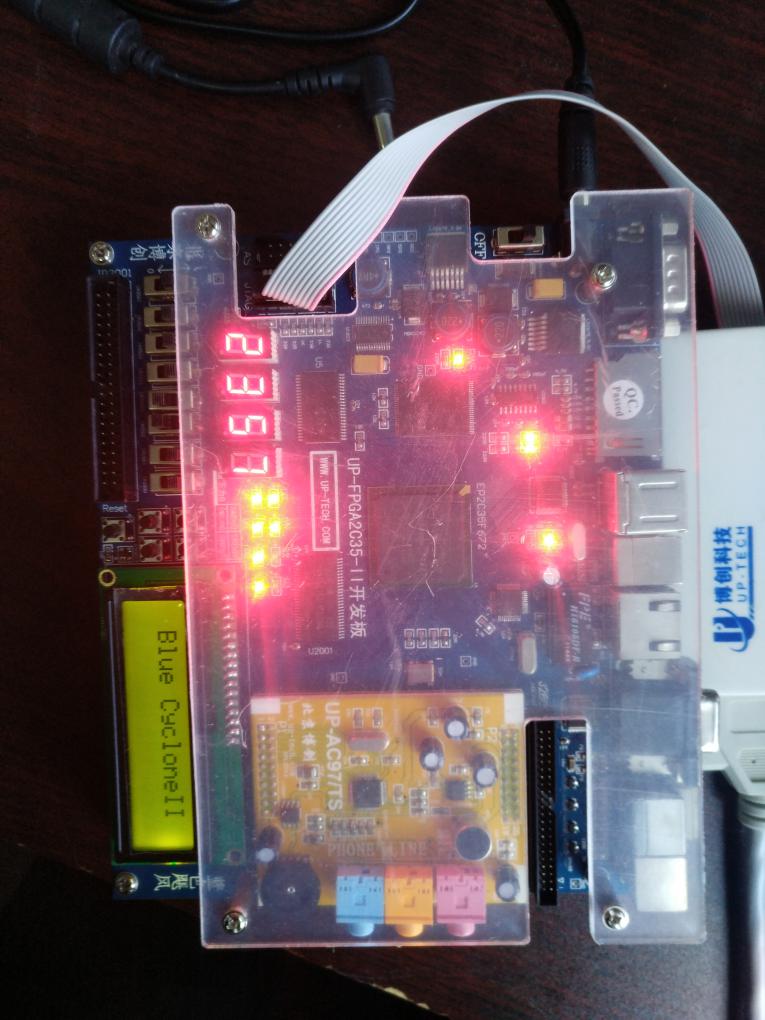
下面是我配好的这个程序的引脚图：



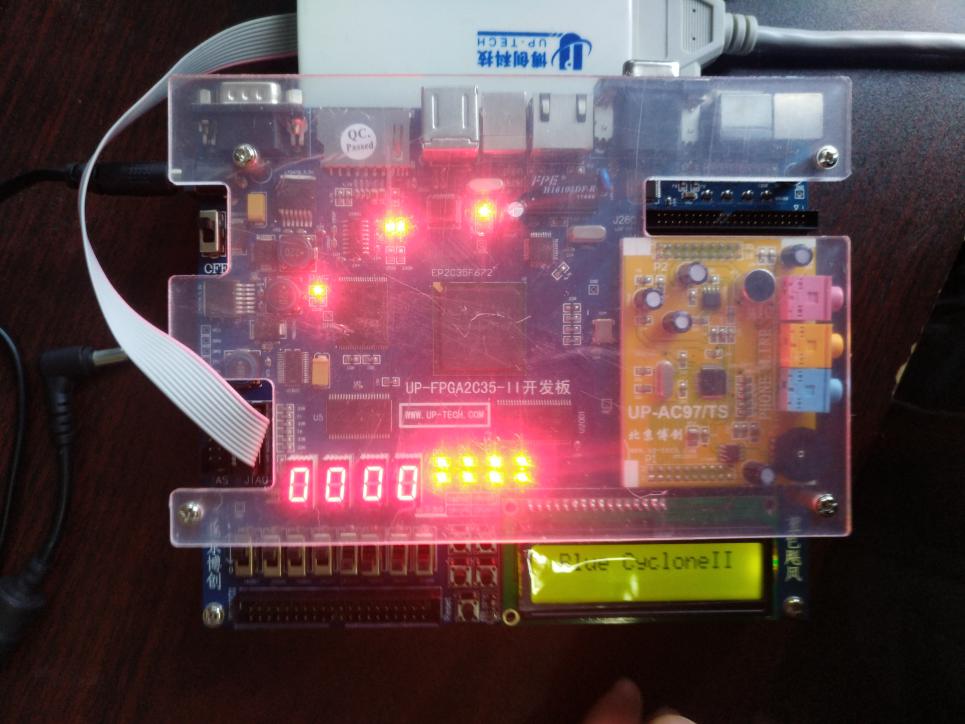


以下是实物图:

**4.2.1正常计时实物图**

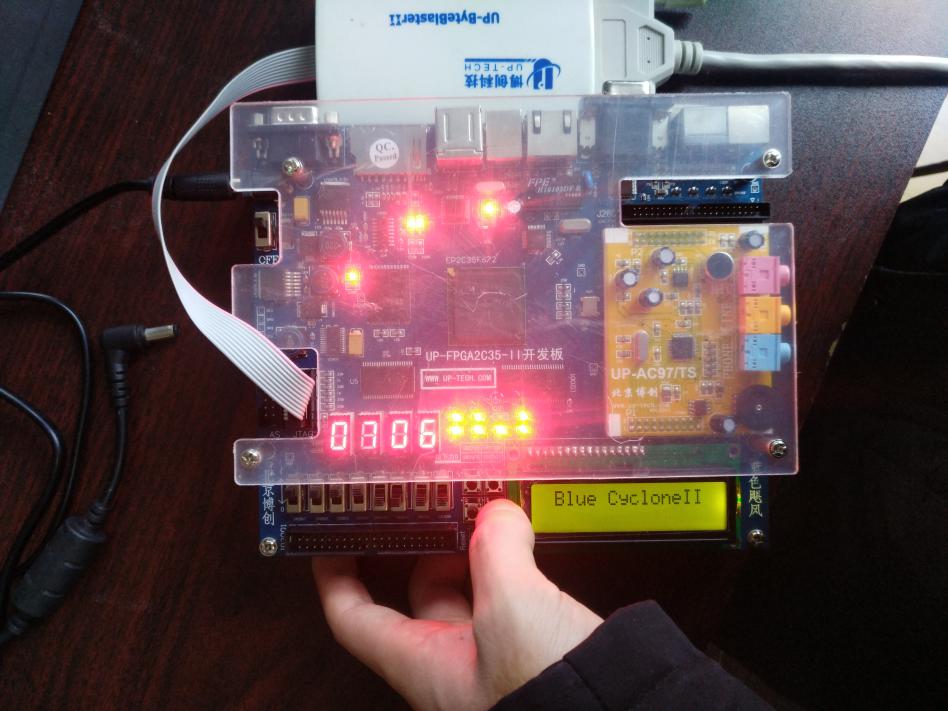
****

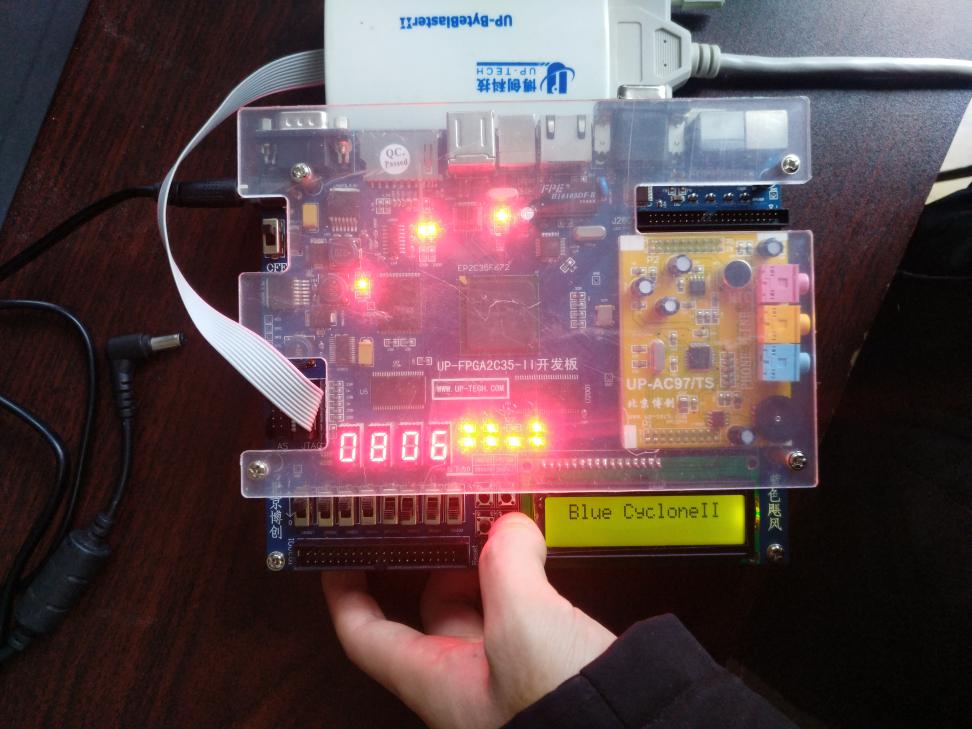
计时到23:59:59后变为00:00:00：

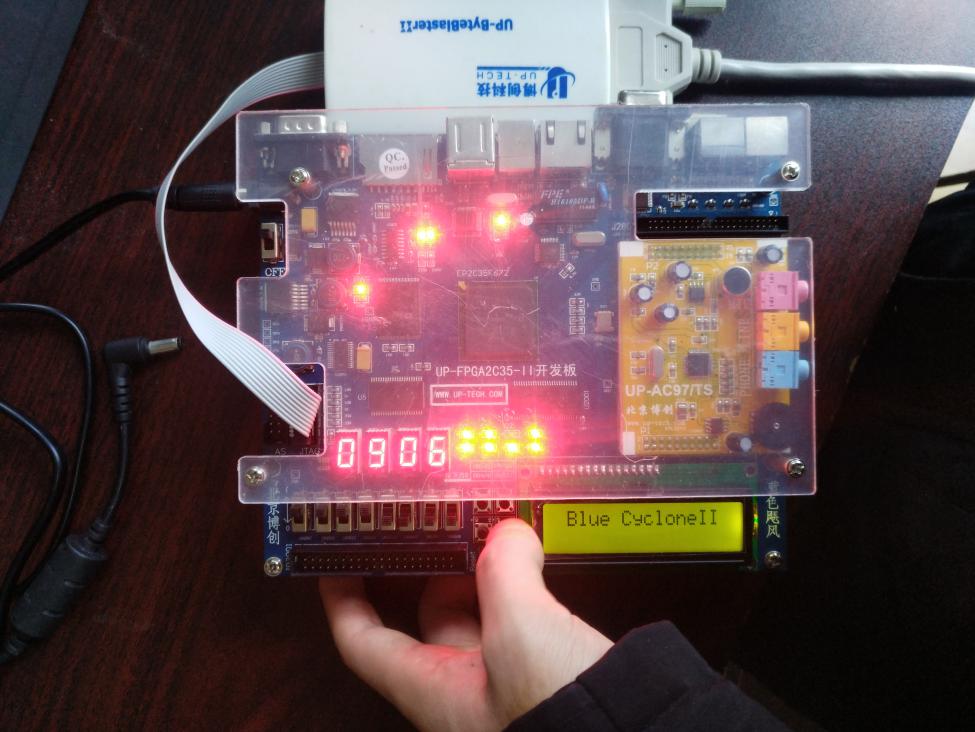
****

**4.2.2手动调时实物图**

调小时，从7点调9点。

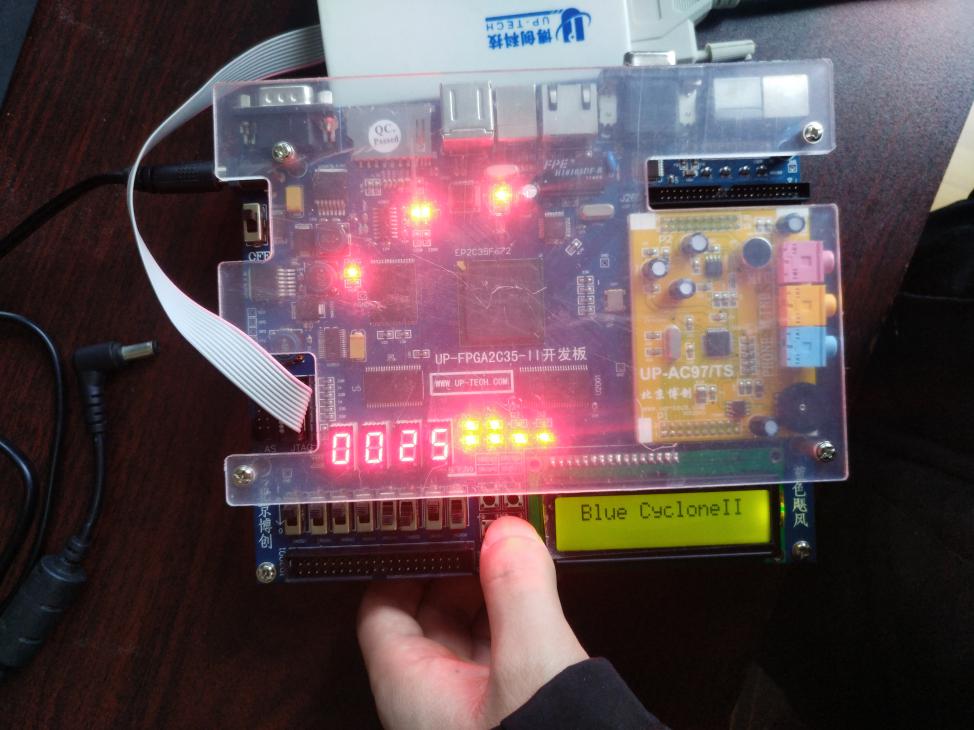
****

****

****

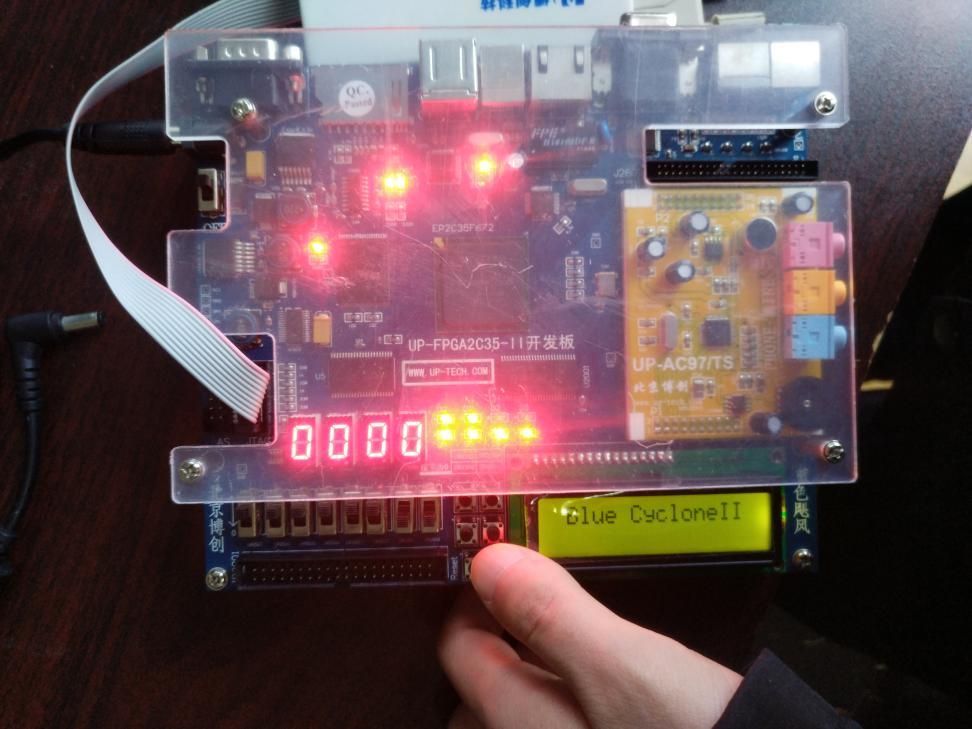
**4.2.3设置闹钟实物图**

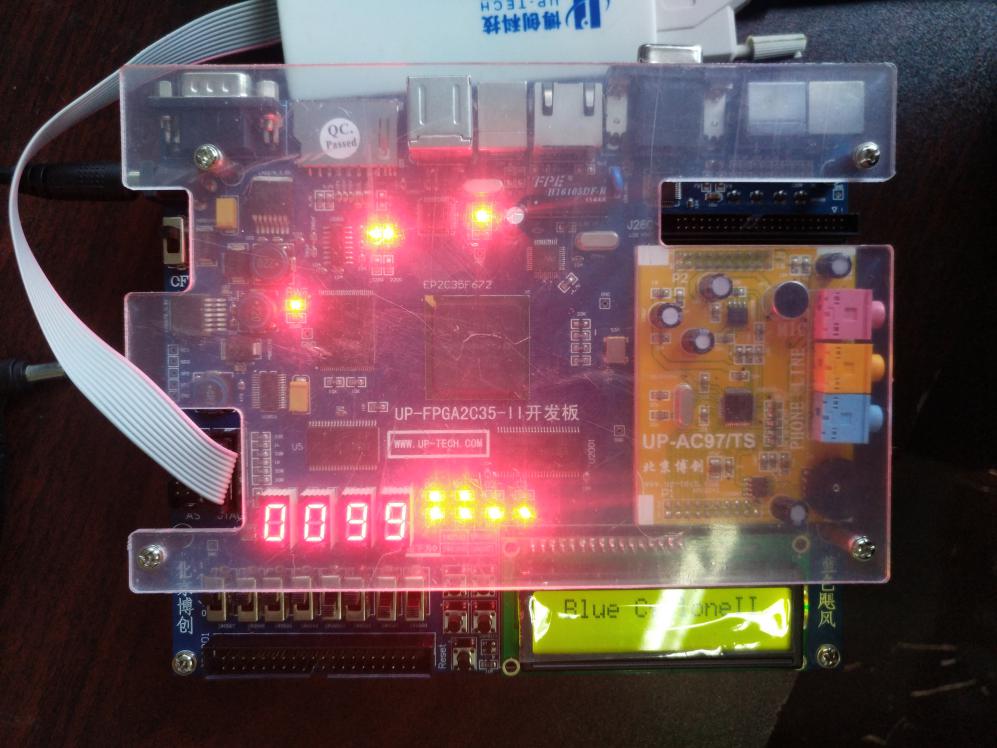
调闹钟为00:25:xx。秒这里不知道几秒，因为不是数码管显示的。

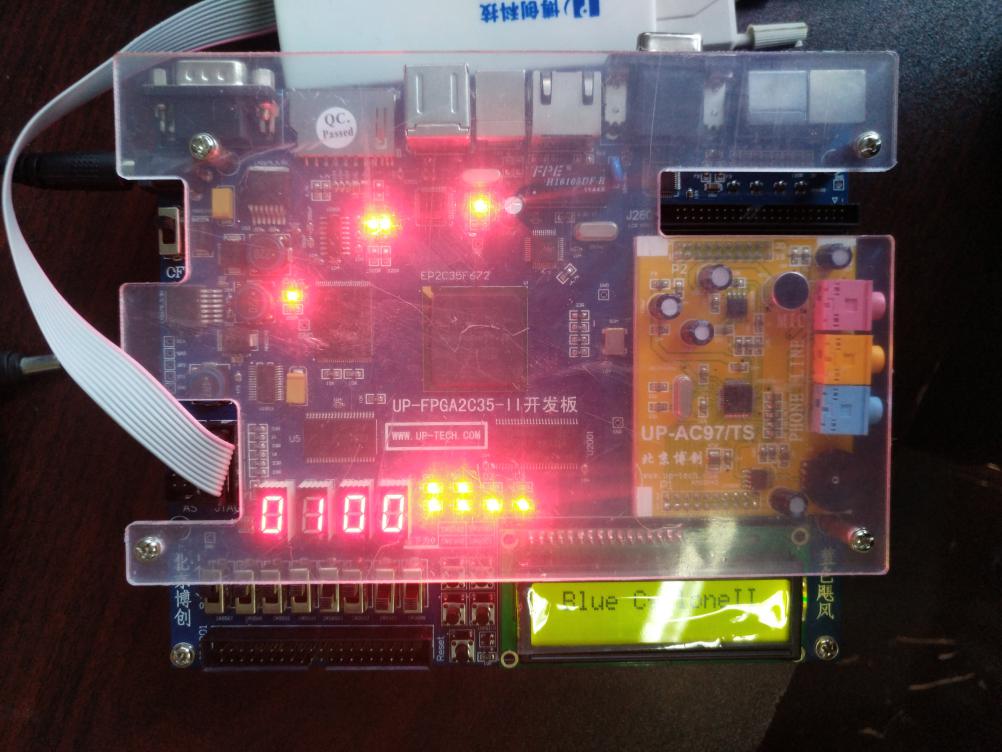


**4.2.4秒表计数实物图**

从0秒计时到101秒的示意图：







1. 总结

对这个期末大作业——智能数字钟的编写过程，我对verilog HDL语言的使用变得更为熟练。由于之前在做期中考试经历了完成整个EDA大作业的流程，期末的这个智能数字钟我主要放精力在代码的编写和调试中。为了实现程序的低耦合，我在编写这个项目的时候，将程序分为了不同的功能模块。这样的程序的可扩展性会更好(比如我在答辩的那个上午就轻松地加了一个秒表功能，只需要再编写秒表模块再在程序的顶层模块进行调用就可以了)，并且这样编写程序，程序的条理性也会更好。

在编写程序的时候也遇到了很多问题，比如刚开始顶层模块调用子模块的时候，模块之间的全局变量我刚开始设置的是reg类型，结果要设置为wire型才有用。又比如在将程序写入FPGA板子中进行调时的时候没有按键消抖，导致调时功能不能很好完成，后面我又加了按键消抖处理。再比如在进行秒表、设置闹钟的时候，要保持自动计时继续进行，我又设计了全局变量进行最后结果的显示。还有我当初写完程序后，发现我使用的判断都是if语句，我在闹钟响应模块尝试使用while语句，结果发现不行，翻阅别人博客好像是因为while不能进行综合。还出现过很多其他的问题，这里就不再赘述。

总得来说，通过这个学期的学习，我大体上掌握了使用verilog语言设计EDA项目，虽然目前的理解还是比较浅的，但是至少算是入门了。

对于我自己而言，我也不确定自己以后是否还会用到verilog HDL，但是这个学期的学习也让我对使用编程语言进行硬件编程有了基本的理解。以后要是用到，也会明白它的工作原理和基本使用方法，这门课也算是有收获了。

**六、参考文献**

无