学生学号	0122015710114	实验课成绩	

武漢理工大學

学 生 实 验 报 告 书

实验课程名称	单片机及嵌入式系统原理
开 课 学 院	信息工程学院
指导教师姓名	周伟
学 生 姓 名	胡姗
学生专业班级	信息 2001

2022 -- 2023 学年 第 一 学期

实验教学管理基本规范

实验是培养学生动手能力、分析解决问题能力的重要环节；实验报告是反映实验教学水平与质量的重要依据。为加强实验过程管理，改革实验成绩考核方法，改善实验教学效果，提高学生质量，特制定实验教学管理基本规范。

1、 本规范适用于理工科类专业实验课程，文、经、管、计算机类实验课程可根据具体情况参照执行或暂不执行。

2、 每门实验课程一般会包括许多实验项目，除非常简单的验证演示性实验项目可以不写实验报告外，其他实验项目均应按本格式完成实验报告。

3、 实验报告应由实验预习、实验过程、结果分析三大部分组成。每部分均在实验成绩中占一定比例。各部分成绩的观测点、考核目标、所占比例可参考附表执行。各专业也可以根据具体情况，调整考核内容和评分标准。

4、 学生必须在完成实验预习内容的前提下进行实验。教师要在实验过程中抽查学生预习情况，在学生离开实验室前，检查学生实验操作和记录情况，并在实验报告第二部分教师签字栏签名，以确保实验记录的真实性。

5、 教师应及时评阅学生的实验报告并给出各实验项目成绩，完整保存实验报告。在完成所有实验项目后，教师应按学生姓名将批改好的各实验项目实验报告装订成册，构成该实验课程总报告，按班级交课程承担单位（实验中心或实验室）保管存档。

6、 实验课程成绩按其类型采取百分制或优、良、中、及格和不及格五级评定。

附表：实验考核参考内容及标准

	观测点	考核目标	成绩组成
实验预习	1． 预习报告 2． 提问 3． 对于设计型实验，着重考查设计方案的科学性、可行性和创新性	对实验目的和基本原理的认识程度，对实验方案的设计能力	20%
实验过程	1． 是否按时参加实验 2． 对实验过程的熟悉程度 3． 对基本操作的规范程度 4． 对突发事件的应急处理能力 5． 实验原始记录的完整程度 6． 同学之间的团结协作精神	着重考查学生的实验态度、基本操作技能；严谨的治学态度、团结协作精神	30%
结果分析	1． 所分析结果是否用原始记录数据 2． 计算结果是否正确 3． 实验结果分析是否合理 4． 对于综合实验，各项内容之间是否有分析、比较与判断等	考查学生对实验数据处理和现象分析的能力；对专业知识的综合应用能力；事实求实的精神	50%

实验课程名称：　　单片机及嵌入式系统原理

实验项目名称	带日历秒表的语音报时电子钟			实验成绩	
实 验 者	胡姗	专业班级	信息 2001	组　　别	
同 组 者	无			实验日期	2022 年 12 月 13 日

第一部分：实验预习报告（包括实验目的、意义，实验基本原理与方法，主要仪器设备及耗材，实验方案与技术路线等）

一、实验目的

1、能够灵活使用 1602 液晶、数码管等显示任意字符串。

2、掌握 A/D、D/A 的基本概念和性能指标。

3、掌握定时器、中断的应用。

二、实验基本原理

我们可以把程序源代码划分为这样几个模块：DS1302 作为走时的核心自成一个模块；点阵、数码管、独立 LED 都属于 LED 的范畴，控制方式类似，也都需要动态扫描，所以把它们整体作为一个模块；液晶是另一个显示模块；按键和遥控器的驱动各自成为一个模块。对于红外所要实现的功能，我们把红外的按键代码解析出来后，把它们映射成标准键盘的键码，就跟板载按键的映射一样，这样红外和板载按键就可以很方便的共用一套应用层接口。但红外键码的映射与板载按键的映射不同，红外键码值不像矩阵按键的行列那样有规律，所以我们这里用一个二维数组来完成这个映射，二维数组每一行的第一个元素是红外遥控器的键码，第二个元素是该键要映射成的标准键码，不要的按键直接映射成 0 即可。这样，当收到一个红外键码后，在这个二维数组每行的第一个元素中查找相同值，找到后即把该行的第二个元素作为参数调用按键动作函数即可。

三、实验内容

编写一系列头文件，对所使用的引脚及变量进行宏定义，编写各个模块函数，实现时间显示，板载按键校时，闹钟，温度测量，红外遥控校时等功能。使用数码管显示时间，LCD1602 显示日期、闹钟、温度，点阵显示星期。

第二部分：实验过程记录（可加页）（包括实验原始数据记录，实验现象记录，实验过程发现的问题等）

在 main 函数中首先使能总中断，T0 定时 1ms，初始化各个模块，通电后延时 1s，刷新各个显示模块。每隔 200ms 刷新一次时间，检测一次闹钟，每隔 1s 刷新一次温度显示。

```
void main()
{
    EA = 1;
    ConfigTimer0(1);
    InitLed();
    InitDS1302();
    InitInfrared();
    InitLcd1602();
    Start18B20();
    while (!flag1s);
    flag1s = 0;
    RefreshTime();
    RefreshDate(1);
    RefreshTemp(1);
    RefreshAlarm();
    while (1)
    {
        KeyDriver();
        InfraredDriver();
        if (flag200ms)
        {
            flag200ms = 0;
            RefreshTime();
            AlarmMonitor();
            if (staSystem == E_NORMAL)
            {
                RefreshDate(0);
            }
          else if(staSystem == E_NORMAL_2)
          {
              RefreshTime2();
          }
        }
        if (flag1s)
        {
            flag1s = 0;
            if (staSystem == E_NORMAL)
            {
                RefreshTemp(0);
            }
        }
    }
}
```

当按下回车键时，系统进行状态的切换；当按下上键时，系统改变液晶的显示。

```
void KeyAction(uint8 keycode)
{
        if  ((keycode>='0') && (keycode<='9'))
        {
            InputSetNumber(keycode);
        }
        else if (keycode == 0x25)
        {
            SetLeftShift();
        }
       else if(keycode==0x26)
       {
           ShowSecondTep();
       }
        else if (keycode == 0x27)
        {
            SetRightShift();
        }
        else if (keycode == 0x0D)
        {
            SwitchSystemSta();
        }
        else if (keycode == 0x1B)
        {
            if (staSystem == E_NORMAL)
            {
                staMute = 1;
            }
            else
            {
                CancelCurSet();
            }
        }
}
```

改变液晶显示函数如下：当系统处于 E_NORMAL 状态时，切换为 E_NORMAL_2，并且将液晶显示改变成第一行显示时间，第二行显示日期。当系统处于 E_NORMAL_2 状态时，切换为 E_NORMAL 状态，且将液晶显示改变成显示日期、闹钟、温度。

```
void ShowSecondTep()
{
    if(staSystem == E_NORMAL)
    {
        staSystem = E_NORMAL_2;
        LcdCloseCursor();
         LcdClearScreen();
        RefreshTime2();
    }
    else if(staSystem == E_NORMAL_2)
    {
        staSystem = E_NORMAL;
         LcdCloseCursor();
         LcdClearScreen();
         RefreshTime();
         RefreshDate(1);
         RefreshTemp(1);
         RefreshAlarm();
    }
}
void RefreshTime2()
{
    unsigned char timenow[10];
    uint8 pdata str[12];
    GetRealTime(&CurTime);
        timenow[0]=((CurTime.hour>>4) & 0xF) + '0';
        timenow[1]=(CurTime.hour& 0xF) + '0';
        timenow[2]='.';
        timenow[3]=((CurTime.min>>4) & 0xF) + '0';
        timenow[4]=(CurTime.min & 0xF) + '0';
        timenow[5]='.';
        timenow[6]=((CurTime.sec>>4) & 0xF) + '0';
        timenow[7]=(CurTime.sec & 0xF) + '0';
        timenow[8]='\0';
        LcdShowStr(0,0,timenow);
        str[0] = ((CurTime.year>>12) & 0xF) + '0';   //4 位数年份
         str[1] = ((CurTime.year>>8) & 0xF) + '0';
         str[2] = ((CurTime.year>>4) & 0xF) + '0';
         str[3] = (CurTime.year & 0xF) + '0';
         str[4] = '-';                                   //分隔符
         str[5] = (CurTime.mon >> 4) + '0';     //月份
         str[6] = (CurTime.mon & 0xF) + '0';
         str[7] = '-';                                   //分隔符
         str[8] = (CurTime.day >> 4) + '0';     //日期
         str[9] = (CurTime.day & 0xF) + '0';
         str[10] = '\0';              //字符串结束符
         LcdShowStr(0, 1, str);
}
```

系统进行状态切换时，如果处于 E_NORMAL 状态，则切换成 E_SET_TIME 状态，通过板载按键或红外进行校时；如果处于 E_SET_TIME 状态，则切换成 E_SET_ALARM 状态，通过板载按键或红外进行闹钟设置；如果处于 E_SET_ALARM 状态，则切换为 E_NORMAL 状态，刷新各个模块的显示。

```
void SwitchSystemSta()
{
    if (staSystem == E_NORMAL)
    {
        staSystem = E_SET_TIME;
        SetTime.year = CurTime.year;
        SetTime.mon  = CurTime.mon;
        SetTime.day  = CurTime.day;
        SetTime.hour = CurTime.hour;
        SetTime.min  = CurTime.min;
        SetTime.sec  = CurTime.sec;
        SetTime.week = CurTime.week;
        LcdClearScreen();
        ShowSetTime();
        SetIndex = 255;
        SetRightShift();
        LcdOpenCursor();
    }
    else if (staSystem == E_SET_TIME)
    {
        staSystem = E_SET_ALARM;
        SetTime.sec = 0;
        SetRealTime(&SetTime);
        SetAlarmHour = AlarmHour;
        SetAlarmMin  = AlarmMin;
        LcdClearScreen();
        ShowSetAlarm();
        SetIndex = 255;
        SetRightShift();
    }
    else
    {
        staSystem = E_NORMAL;
        AlarmHour = SetAlarmHour;
        AlarmMin  = SetAlarmMin;
        LcdCloseCursor();
```

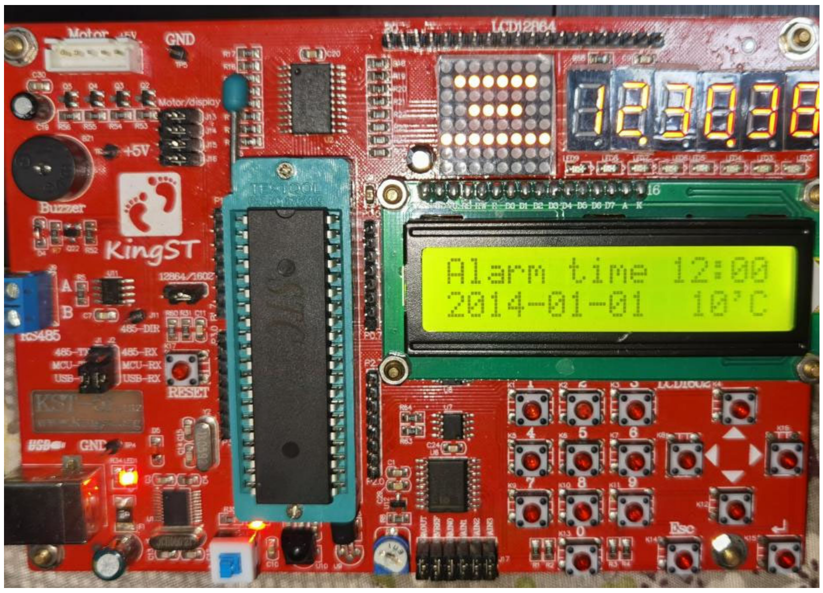
```
            LcdClearScreen();
            RefreshTime();
            RefreshDate(1);
            RefreshTemp(1);
            RefreshAlarm();
      }
}
```

教师签字__________

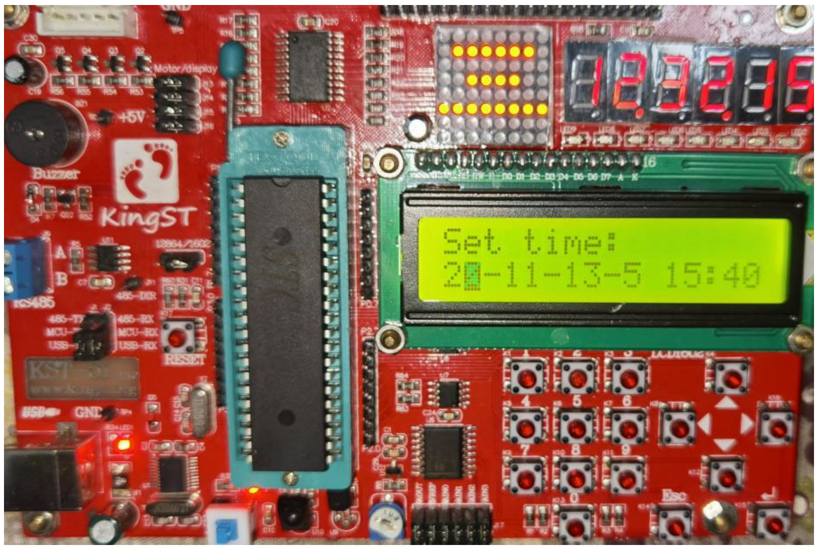
第三部分　结果与讨论（可加页）

一、实验结果分析（包括数据处理、实验现象分析、影响因素讨论、综合分析和结论等）

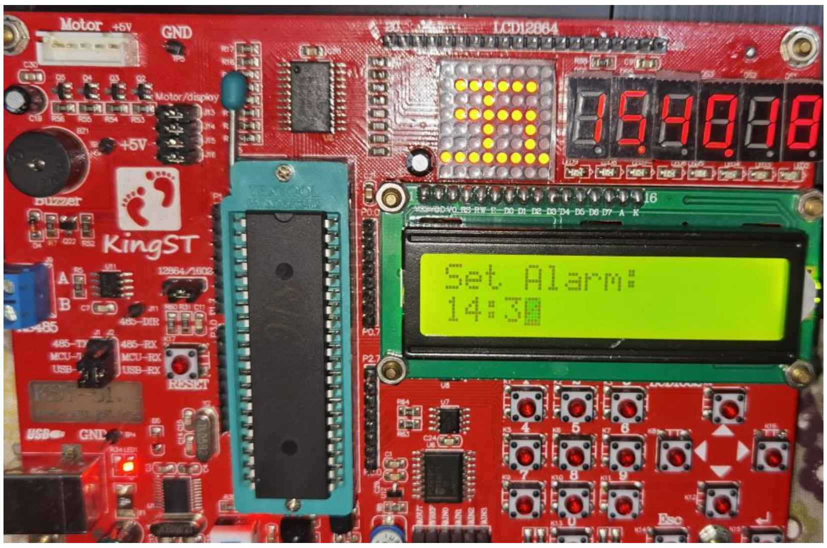
系统处于正常显示时，点阵显示星期，数码管显示时间，液晶显示闹钟、日期、温度。



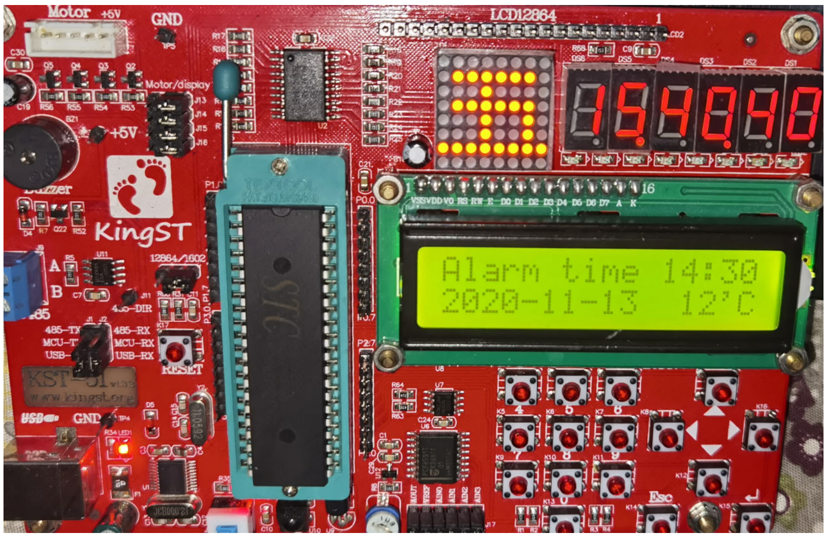
按下回车键，进入校时状态，通过板载按键或红外进行校时。



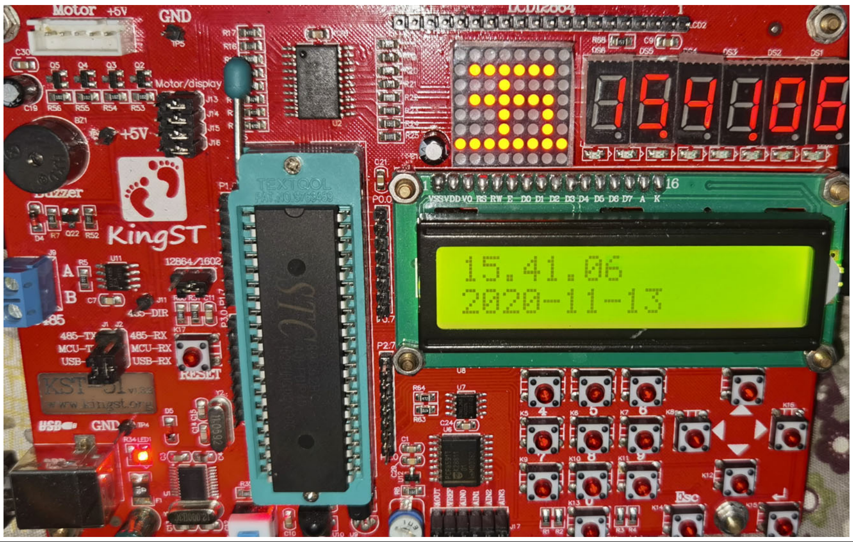
再次按下回车键，刷新校时后的显示，进入闹钟设置状态，通过板载按键或红外进行设置。



再次按下回车键，回到正常显示状态，且刷新了闹钟的时间。



正常显示状态，按下上键，切换液晶的显示，液晶开始显示时间和日期。



二、思考题

（1）请参照本实验的核心代码及硬件结构，总结运用单片机进行小型功能设计的总体流程

答：首先进行程序结构的规划，将单片机划分成一个个模块，分别实现所需要求，然后将程序进行分层，根据所需功能画出整体流程图，接着对需要的引脚、变量进行宏定义，最后分模块编写程序实现功能。

（2）请依据自身实验过程，结合自身体会，总结几点单片机开发的注意事项。请至少给出 3 条。

答：1、确定系统的复位信号是否可靠，一般在单片机的数据手册中都会提到该单片机需要的复位信号的要求。复位电平的宽度和幅度都应满足芯片的要求，并且要求保持稳定。还有特别重要的一点就是复位电平应与电源上电在同一时刻发生，即芯片一上电，复位信号就已产生。不然，由于没有经过复位，单片机中的寄存器的值为随机值，上电时就会按 PC 寄存器中的随机内容开始运行程序，这样很容易进行误操作或进入死机状态。

2、确定系统初始化是否有效，系统程序开始应延时一段时间，因为系统中的芯片以及器件从上电开始到正常工作的状态往往有一段时间，程序开始时延时一段时间，是让系统中所有器件到达正常工作状态。究竟延时多少才算合适？这取决于系统的各芯片中到达正常工作状态的时间，通常以最慢的为准。

3、上电时对系统进行检测，上电时对系统中进行检测是单片机程序中的一个良好设计。在硬件设计时也应该细细考虑将各个使用到的芯片、接口设计成容易使用软件进行测试的模式。很多有经验的单片机设计者都会在系统上电时（特别是第一次上电时）进行全面的检测，或者更进一步，将系统的运行状态中分为测试模式和正常运行模式，通过加入测试模式对系统进行详细的检测，使得系统的批量检测更为方便容易。

附录：

config.h 文件：

```
#ifndef _CONFIG_H
#define _CONFIG_H
/*  通用头文件  */
#include <reg52.h>
#include <intrins.h>
/*  数据类型定义  */
typedef   signed      char      int8;      // 8 位有符号整型数
typedef   signed      int        int16;     //16 位有符号整型数
typedef   signed      long       int32;     //32 位有符号整型数
typedef   unsigned   char       uint8;     // 8 位无符号整型数
typedef   unsigned   int        uint16;    //16 位无符号整型数
typedef   unsigned   long       uint32;    //32 位无符号整型数
/*  全局运行参数定义  */
#define SYS_MCLK      (11059200/12)   //系统主时钟频率，即振荡器频率÷12
/* IO 引脚分配定义  */
sbit KEY_IN_1    = P2^4;   //矩阵按键的扫描输入引脚 1
sbit KEY_IN_2    = P2^5;   //矩阵按键的扫描输入引脚 2
sbit KEY_IN_3    = P2^6;   //矩阵按键的扫描输入引脚 3
sbit KEY_IN_4    = P2^7;   //矩阵按键的扫描输入引脚 4
sbit KEY_OUT_1 = P2^3;    //矩阵按键的扫描输出引脚 1
sbit KEY_OUT_2 = P2^2;    //矩阵按键的扫描输出引脚 2
sbit KEY_OUT_3 = P2^1;    //矩阵按键的扫描输出引脚 3
sbit KEY_OUT_4 = P2^0;    //矩阵按键的扫描输出引脚 4
sbit ADDR0 = P1^0;    //LED 位选译码地址引脚 0
sbit ADDR1 = P1^1;    //LED 位选译码地址引脚 1
sbit ADDR2 = P1^2;    //LED 位选译码地址引脚 2
sbit ADDR3 = P1^3;    //LED 位选译码地址引脚 3
sbit ENLED = P1^4;    //LED 显示部件的总使能引脚
#define LCD1602_DB    P0      //1602 液晶数据端口
sbit LCD1602_RS = P1^0;    //1602 液晶指令/数据选择引脚
sbit LCD1602_RW = P1^1;    //1602 液晶读写引脚
sbit LCD1602_E    = P1^5;    //1602 液晶使能引脚
sbit DS1302_CE = P1^7;    //DS1302 片选引脚
sbit DS1302_CK = P3^5;    //DS1302 通信时钟引脚
sbit DS1302_IO = P3^4;    //DS1302 通信数据引脚
sbit I2C_SCL = P3^7;    //I2C 总线时钟引脚
sbit I2C_SDA = P3^6;    //I2C 总线数据引脚
sbit BUZZER = P1^6;    //蜂鸣器控制引脚
sbit IO_18B20 = P3^2;    //DS18B20 通信引脚
sbit IR_INPUT = P3^3;    //红外接收引脚
#endif
```

DS18B20.h 文件：

```
#ifndef _DS18B20_H
#define _DS18B20_H
#ifndef _DS18B20_C
#endif
bit Start18B20();
bit Get18B20Temp(int16 *temp);
#endif
```

DS1302.h 文件：

```
#ifndef _DS1302_H
#define _DS1302_H
struct sTime {  //日期时间结构
    uint16 year; //年
    uint8 mon;   //月
    uint8 day;   //日
    uint8 hour;  //时
    uint8 min;   //分
    uint8 sec;   //秒
    uint8 week;  //星期
};
#ifndef _DS1302_C
#endif
void InitDS1302();
void GetRealTime(struct sTime *time);
void SetRealTime(struct sTime *time);
#endif
```

Infrared.h 文件：

```
#ifndef _INFRARED_H
#define _INFRARED_H
```

```
#ifndef _INFRARED_C
#endif
void InitInfrared();
void InfraredDriver();
#endif
```

keyboard.h 文件：

```
#ifndef _KEY_BOARD_H
#define _KEY_BOARD_H
#ifndef _KEY_BOARD_C
#endif
void KeyScan();
void KeyDriver();
#endif
```

Lcd1602.h 文件：

```
#ifndef _LCD1602_H
#define _LCD1602_H
#ifndef _LCD1602_C
#endif
void InitLcd1602();
void LcdClearScreen();
void LcdOpenCursor();
void LcdCloseCursor();
void LcdSetCursor(uint8 x, uint8 y);
void LcdShowStr(uint8 x, uint8 y, uint8 *str);
void LcdShowChar(uint8 x, uint8 y, uint8 chr);
#endif
```

LedBuzzer.h 文件：

```
#ifndef _LED_BUZZER_H
#define _LED_BUZZER_H
struct sLedBuff {  //LED 显示缓冲区结构
    uint8 array[8];   //点阵缓冲区
    uint8 number[6];  //数码管缓冲区
};
#ifndef _LED_BUZZER_C
extern bit staBuzzer;
extern struct sLedBuff ledBuff;
```

```
#endif
void InitLed();
void ShowLedNumber(uint8 index, uint8 num, uint8 point);
void ShowLedArray(uint8 *ptr);
#endif
```

main.h 文件：

```
#ifndef _MAIN_H
#define _MAIN_H
enum eStaSystem {  //系统运行状态枚举
    E_NORMAL, E_SET_TIME, E_SET_ALARM,E_NORMAL_2
};
#ifndef _MAIN_C
extern enum eStaSystem staSystem;
#endif
void RefreshTemp(uint8 ops);
void ConfigTimer0(uint16 ms);
#endif
```

Time.h 文件：

```
#ifndef _TIME_H
#define _TIME_H
#ifndef _TIME_C
#endif
void RefreshTime();
void RefreshDate(uint8 ops);
void RefreshAlarm();
void AlarmMonitor();
void KeyAction(uint8 keycode);
void RefreshTime2();
#endif
```

DS18B20.c 文件：

```
#define  _DS18B20_C
#include "config.h"
#include "DS18B20.h"
void DelayX10us(uint8 t)
{
    do {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    } while (--t);
}
bit Get18B20Ack()
{
    bit ack;

    EA = 0;   //禁止总中断
    IO_18B20 = 0;     //产生 500us 复位脉冲
    DelayX10us(50);
    IO_18B20 = 1;
    DelayX10us(6);    //延时 60us
    ack = IO_18B20;   //读取存在脉冲
    while(!IO_18B20); //等待存在脉冲结束
    EA = 1;   //重新使能总中断

    return ack;
}
void Write18B20(uint8 dat)
{
    uint8 mask;

    EA = 0;   //禁止总中断
    for (mask=0x01; mask!=0; mask<<=1)  //低位在先，依次移出 8 个 bit
    {
        IO_18B20 = 0;         //产生 2us 低电平脉冲
        _nop_();
        _nop_();
        if ((mask&dat) == 0)  //输出该 bit 值
            IO_18B20 = 0;
        else
            IO_18B20 = 1;
        DelayX10us(6);          //延时 60us
        IO_18B20 = 1;           //拉高通信引脚
    }
    EA = 1;   //重新使能总中断
}
uint8 Read18B20()
{
    uint8 dat;
    uint8 mask;

    EA = 0;   //禁止总中断
    for (mask=0x01; mask!=0; mask<<=1)  //低位在先，依次采集 8 个 bit
    {
        IO_18B20 = 0;           //产生 2us 低电平脉冲
        _nop_();
        _nop_();
        IO_18B20 = 1;           //结束低电平脉冲，等待 18B20 输出数据
        _nop_();                //延时 2us
        _nop_();
        if (!IO_18B20)          //读取通信引脚上的值
            dat &= ~mask;
        else
            dat |= mask;
        DelayX10us(6);          //再延时 60us
    }
    EA = 1;   //重新使能总中断

    return dat;
}
bit Start18B20()
{
    bit ack;

    ack = Get18B20Ack();   //执行总线复位，并获取 18B20 应答
    if (ack == 0)          //如 18B20 正确应答，则启动一次转换
    {
        Write18B20(0xCC);  //跳过 ROM 操作
        Write18B20(0x44);  //启动一次温度转换
    }
    return ~ack;   //ack==0 表示操作成功，所以返回值对其取反
}
bit Get18B20Temp(int16 *temp)
{
    bit ack;
    uint8 LSB, MSB; //16bit 温度值的低字节和高字节

    ack = Get18B20Ack();    //执行总线复位，并获取 18B20 应答
    if (ack == 0)           //如 18B20 正确应答，则读取温度值
    {
        Write18B20(0xCC);   //跳过 ROM 操作
        Write18B20(0xBE);   //发送读命令
        LSB = Read18B20();  //读温度值的低字节
        MSB = Read18B20();  //读温度值的高字节
        *temp = ((int16)MSB << 8) + LSB;  //合成为 16bit 整型数
    }
    return ~ack;  //ack==0 表示操作应答，所以返回值为其取反值
}
```

DS1302.c 文件：

```
#define  _DS1302_C
#include "config.h"
#include "DS1302.h"
void DS1302ByteWrite(uint8 dat)
{
    uint8 mask;

    for (mask=0x01; mask!=0; mask<<=1)  //低位在前，逐位移出
    {
        if ((mask&dat) != 0) //首先输出该位数据
            DS1302_IO = 1;
        else
            DS1302_IO = 0;
        DS1302_CK = 1;       //然后拉高时钟
        DS1302_CK = 0;       //再拉低时钟，完成一个位的操作
    }
    DS1302_IO = 1;           //最后确保释放 IO 引脚
}
uint8 DS1302ByteRead()
```

```
{
    uint8 mask;
    uint8 dat = 0;

    for (mask=0x01; mask!=0; mask<<=1)  //低位在前，逐位读取
    {
        if (DS1302_IO != 0)  //首先读取此时的 IO 引脚，并设置 dat 中的对应位
        {
            dat |= mask;
        }
        DS1302_CK = 1;       //然后拉高时钟
        DS1302_CK = 0;       //再拉低时钟，完成一个位的操作
    }
    return dat;              //最后返回读到的字节数据
}
void DS1302SingleWrite(uint8 reg, uint8 dat)
{
    DS1302_CE = 1;                    //使能片选信号
    DS1302ByteWrite((reg<<1)|0x80);  //发送写寄存器指令
    DS1302ByteWrite(dat);             //写入字节数据
    DS1302_CE = 0;                    //除能片选信号
}
uint8 DS1302SingleRead(uint8 reg)
{
    uint8 dat;

    DS1302_CE = 1;                    //使能片选信号
    DS1302ByteWrite((reg<<1)|0x81);  //发送读寄存器指令
    dat = DS1302ByteRead();           //读取字节数据
    DS1302_CE = 0;                    //除能片选信号

    return dat;
}
void DS1302BurstWrite(uint8 *dat)
{
    uint8 i;

    DS1302_CE = 1;
    DS1302ByteWrite(0xBE);  //发送突发写寄存器指令
    for (i=0; i<8; i++)     //连续写入 8 字节数据
    {
        DS1302ByteWrite(dat[i]);
    }
    DS1302_CE = 0;
}
void DS1302BurstRead(uint8 *dat)
{
    uint8 i;

    DS1302_CE = 1;
    DS1302ByteWrite(0xBF);  //发送突发读寄存器指令
    for (i=0; i<8; i++)     //连续读取 8 个字节
    {
        dat[i] = DS1302ByteRead();
    }
    DS1302_CE = 0;
}
void GetRealTime(struct sTime *time)
{
    uint8 buf[8];

    DS1302BurstRead(buf);
    time->year = buf[6] + 0x2000;
    time->mon  = buf[4];
    time->day  = buf[3];
    time->hour = buf[2];
    time->min  = buf[1];
    time->sec  = buf[0];
    time->week = buf[5];
}
/* 设定实时时间，时间结构体格式的设定时间转换为数组并写入 DS1302 */
void SetRealTime(struct sTime *time)
{
    uint8 buf[8];

    buf[7] = 0;
    buf[6] = time->year;
    buf[5] = time->week;
    buf[4] = time->mon;
    buf[3] = time->day;
    buf[2] = time->hour;
    buf[1] = time->min;
    buf[0] = time->sec;
    DS1302BurstWrite(buf);
}
void InitDS1302()
{
    uint8 dat;
    struct sTime code InitTime[] = {  //默认初始值：2014-01-01 12:30:00 星期3
        0x2014,0x01,0x01, 0x12,0x30,0x00, 0x03
    };

    DS1302_CE = 0;  //初始化DS1302通信引脚
    DS1302_CK = 0;
    dat = DS1302SingleRead(0);  //读取秒寄存器
    if ((dat & 0x80) != 0)      //由秒寄存器最高位CH的值判断DS1302是否已停止
    {
        DS1302SingleWrite(7, 0x00);  //撤销写保护以允许写入数据
        SetRealTime(&InitTime);      //设置DS1302为默认的初始时间
    }
}
```

Infrared.c 文件：

```
#define  _INFRARED_C
#include "config.h"
#include "Infrared.h"
#include "Time.h"

const uint8 code IrCodeMap[][2] = {
    {0x45,0x00}, {0x46,0x00}, {0x47,0x1B},
    {0x44,0x00}, {0x40,0x25}, {0x43,0x27},
    {0x07,0x00}, {0x15,0x28}, {0x09,0x26},
    {0x16, '0'}, {0x19,0x1B}, {0x0D,0x0D},
    {0x0C, '1'}, {0x18, '2'}, {0x5E, '3'},
    {0x08, '4'}, {0x1C, '5'}, {0x5A, '6'},
    {0x42, '7'}, {0x52, '8'}, {0x4A, '9'},
};
```

```
bit irflag = 0;
uint8 ircode[4];
void InfraredDriver()
{
    uint8 i;

    if (irflag)
    {
        irflag = 0;
        for (i=0; i<sizeof(IrCodeMap)/sizeof(IrCodeMap[0]); i++)
        {
            if (ircode[2] == IrCodeMap[i][0])
            {
                KeyAction(IrCodeMap[i][1]);
                break;
            }
        }
    }
}
void InitInfrared()
{
    IR_INPUT = 1;
    TMOD &= 0x0F;
    TMOD |= 0x10;
    TR1 = 0;
    ET1 = 0;
    IT1 = 1;
    EX1 = 1;
}
uint16 GetHighTime()
{
    TH1 = 0;
    TL1 = 0;
    TR1 = 1;
    while (IR_INPUT)
    {
        if (TH1 >= 0x40)
        {
            break;
        }
    }
    TR1 = 0;

    return (TH1*256 + TL1);
}
uint16 GetLowTime()
{
    TH1 = 0;
    TL1 = 0;
    TR1 = 1;
    while (!IR_INPUT)
    {
        if (TH1 >= 0x40)
        {
            break;
        }
    }
    TR1 = 0;

    return (TH1*256 + TL1);
}
void EXINT1_ISR() interrupt 2
{
    uint8 i, j;
    uint8 byt;
    uint16 time;
    time = GetLowTime();
    if ((time<7833) || (time>8755))
    {
        IE1 = 0;
        return;
    }
    time = GetHighTime();
    if ((time<3686) || (time>4608))
    {
        IE1 = 0;
        return;
    }
    for (i=0; i<4; i++)
    {
        for (j=0; j<8; j++)
        {
            time = GetLowTime();
            if ((time<313) || (time>718))
            {
                IE1 = 0;
                return;
            }
            time = GetHighTime();
            if ((time>313) && (time<718))
            {
                byt >>= 1;
            }
            else if ((time>1345) && (time<1751))
            {
                byt >>= 1;
                byt |= 0x80;
            }
            else
            {
                IE1 = 0;
                return;
            }
        }
        ircode[i] = byt;
    }
    irflag = 1;
    IE1 = 0;
}
```

keyboard.c 文件：

```
#define  _KEY_BOARD_C
#include "config.h"
#include "keyboard.h"
#include "Time.h"
```

```
const uint8 code KeyCodeMap[4][4] = {
    { '1',  '2',  '3', 0x26 },
    { '4',  '5',  '6', 0x25 },
    { '7',  '8',  '9', 0x28 },
    { '0', 0x1B, 0x0D, 0x27 }
};
uint8 pdata KeySta[4][4] = {
    {1, 1, 1, 1},  {1, 1, 1, 1},  {1, 1, 1, 1},  {1, 1, 1, 1}
};
void KeyDriver()
{
    uint8 i, j;
    static uint8 pdata backup[4][4] = {
        {1, 1, 1, 1},  {1, 1, 1, 1},  {1, 1, 1, 1},  {1, 1, 1, 1}
    };

    for (i=0; i<4; i++)
    {
        for (j=0; j<4; j++)
        {
            if (backup[i][j] != KeySta[i][j])
            {
                if (backup[i][j] != 0)
                {
                    KeyAction(KeyCodeMap[i][j]);
                }
                backup[i][j] = KeySta[i][j];
            }
        }
    }
}
void KeyScan()
{
    uint8 i;
    static uint8 keyout = 0;
    static uint8 keybuf[4][4] = {
        {0xFF, 0xFF, 0xFF, 0xFF},  {0xFF, 0xFF, 0xFF, 0xFF},
        {0xFF, 0xFF, 0xFF, 0xFF},  {0xFF, 0xFF, 0xFF, 0xFF}
    };
    keybuf[keyout][0] = (keybuf[keyout][0] << 1) | KEY_IN_1;
    keybuf[keyout][1] = (keybuf[keyout][1] << 1) | KEY_IN_2;
    keybuf[keyout][2] = (keybuf[keyout][2] << 1) | KEY_IN_3;
    keybuf[keyout][3] = (keybuf[keyout][3] << 1) | KEY_IN_4;
    for (i=0; i<4; i++)
    {
        if ((keybuf[keyout][i] & 0x0F) == 0x00)
        {
            KeySta[keyout][i] = 0;
        }
        else if ((keybuf[keyout][i] & 0x0F) == 0x0F)
        {
            KeySta[keyout][i] = 1;
        }
    }
    keyout++;
    keyout &= 0x03;
    switch (keyout)
    {
        case 0: KEY_OUT_4 = 1; KEY_OUT_1 = 0; break;
        case 1: KEY_OUT_1 = 1; KEY_OUT_2 = 0; break;
        case 2: KEY_OUT_2 = 1; KEY_OUT_3 = 0; break;
        case 3: KEY_OUT_3 = 1; KEY_OUT_4 = 0; break;
        default: break;
    }
}
```

Lcd1602.c 文件：

```
#define  _LCD1602_C
#include "config.h"
#include "Lcd1602.h"

uint8 tmpP0;
bit tmpADDR0;
bit tmpADDR1;
void LedScanPause()
{
    ENLED = 1;
    tmpP0 = P0;
    tmpADDR0 = ADDR0;
    tmpADDR1 = ADDR1;
}
void LedScanContinue()
{
    ADDR0 = tmpADDR0;
    ADDR1 = tmpADDR1;
    P0 = tmpP0;
    ENLED = 0;
}
void LcdWaitReady()
{
    uint8 sta;

    LCD1602_DB = 0xFF;
    LCD1602_RS = 0;
    LCD1602_RW = 1;
    do {
        LCD1602_E = 1;
        sta = LCD1602_DB;
        LCD1602_E = 0;
    } while (sta & 0x80);
}
void LcdWriteCmd(uint8 cmd)
{
    LedScanPause();
    LcdWaitReady();
    LCD1602_RS = 0;
    LCD1602_RW = 0;
    LCD1602_DB = cmd;
    LCD1602_E  = 1;
    LCD1602_E  = 0;
    LedScanContinue();
}
void LcdWriteDat(uint8 dat)
{
    LedScanPause();
    LcdWaitReady();
    LCD1602_RS = 1;
    LCD1602_RW = 0;
    LCD1602_DB = dat;
    LCD1602_E  = 1;
    LCD1602_E  = 0;
    LedScanContinue();
}
void LcdClearScreen()
{
    LcdWriteCmd(0x01);
}
void LcdOpenCursor()
{
    LcdWriteCmd(0x0F);
}
void LcdCloseCursor()
{
    LcdWriteCmd(0x0C);
}
void LcdSetCursor(uint8 x, uint8 y)
{
    uint8 addr;

    if (y == 0)
        addr = 0x00 + x;
    else
        addr = 0x40 + x;
    LcdWriteCmd(addr | 0x80);
}
void LcdShowStr(uint8 x, uint8 y, uint8 *str)
{
    LcdSetCursor(x, y);
    while (*str != '\0')
    {
        LcdWriteDat(*str++);
    }
}
void LcdShowChar(uint8 x, uint8 y, uint8 chr)
{
    LcdSetCursor(x, y);
    LcdWriteDat(chr);
}
void InitLcd1602()
{
    LcdWriteCmd(0x38);
    LcdWriteCmd(0x0C);
    LcdWriteCmd(0x06);
    LcdWriteCmd(0x01);
}
```

LedBuzzer.c 文件：

```
#define  _LED_BUZZER_C
#include "config.h"
#include "LedBuzzer.h"

uint8 code LedChar[] = {
    0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
    0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E
};
bit staBuzzer = 0;
struct sLedBuff ledBuff;
void InitLed()
{
    P0 = 0xFF;
    ENLED = 0;
    T2CON = 0x00;
    RCAP2H = ((65536-SYS_MCLK/1500)>>8);
    RCAP2L = (65536-SYS_MCLK/1500);
    TH2 = RCAP2H;
    TL2 = RCAP2L;
    ET2 = 1;
    PT2 = 1;
    TR2 = 1;
}
void ShowLedNumber(uint8 index, uint8 num, uint8 point)
{
    ledBuff.number[index] = LedChar[num];
    if (point != 0)
    {
        ledBuff.number[index] &= 0x7F;
    }
}
void ShowLedArray(uint8 *ptr)
{
    uint8 i;

    for (i=0; i<sizeof(ledBuff.array); i++)
    {
        ledBuff.array[i] = *ptr++;
    }
}
void InterruptTimer2() interrupt 5
{
    static uint8 i = 0;
    TF2 = 0;
    if (ENLED == 0)
    {
        P0 = 0xFF;
        P1 = (P1 & 0xF0) | i;
        P0 = *((uint8 data*)&ledBuff+i);
        if (i < (sizeof(ledBuff)-1))
            i++;
        else
            i = 0;
    }
    if (staBuzzer == 1)
        BUZZER = ~BUZZER;
    else
        BUZZER = 1;
}
```

main.c 文件：

```
#define  _MAIN_C
#include "config.h"
#include "Lcd1602.h"
#include "LedBuzzer.h"
#include "keyboard.h"
```

```
#include "DS1302.h"
#include "DS18B20.h"
#include "Infrared.h"
#include "Time.h"
#include "main.h"

bit flag1s = 0;
bit flag200ms = 0;
uint8 T0RH = 0;
uint8 T0RL = 0;
enum eStaSystem staSystem = E_NORMAL;

void main()
{
    EA = 1;
    ConfigTimer0(1);
    InitLed();
    InitDS1302();
    InitInfrared();
    InitLcd1602();
    Start18B20();
    while (!flag1s);
    flag1s = 0;
    RefreshTime();
    RefreshDate(1);
    RefreshTemp(1);
    RefreshAlarm();
    while (1)
    {
        KeyDriver();
        InfraredDriver();
        if (flag200ms)
        {
            flag200ms = 0;
            RefreshTime();
            AlarmMonitor();
            if (staSystem == E_NORMAL)
            {
                RefreshDate(0);
            }
            else if(staSystem == E_NORMAL_2)
            {
                RefreshTime2();
            }
        }
        if (flag1s)
        {
            flag1s = 0;
            if (staSystem == E_NORMAL)
            {
                RefreshTemp(0);
            }
        }
    }
}
void RefreshTemp(uint8 ops)
{
    int16 temp;
    uint8 pdata str[8];
    static int16 backup = 0;

    Get18B20Temp(&temp);
    Start18B20();
    temp >>= 4;
    if ((backup!=temp) || (ops!=0))
    {
        str[0] = (temp/10) + '0';
        str[1] = (temp%10) + '0';
        str[2] = '\'';
        str[3] = 'C';
        str[4] = '\0';
        LcdShowStr(12, 1, str);
        backup = temp;
    }
}
void ConfigTimer0(uint16 ms)
{
    uint32 tmp;
    tmp = (SYS_MCLK*ms)/1000;
    tmp = 65536 - tmp;
    tmp = tmp + 33;
    T0RH = (uint8)(tmp>>8);
    T0RL = (uint8)tmp;
    TMOD &= 0xF0;
    TMOD |= 0x01;
    TH0 = T0RH;
    TL0 = T0RL;
    ET0 = 1;
    TR0 = 1;
}
void InterruptTimer0() interrupt 1
{
    static uint8 tmr2s = 0;
    static uint8 tmr200ms = 0;

    TH0 = T0RH;
    TL0 = T0RL;
    tmr200ms++;
    if (tmr200ms >= 200)
    {
        tmr200ms = 0;
        flag200ms = 1;
        tmr2s++;
        if (tmr2s >= 5)
        {
            tmr2s = 0;
            flag1s = 1;
        }
    }
    KeyScan();
}
```

Time.c 文件：

```
#define  _TIME_C
#include "config.h"
#include "DS1302.h"
```

```
#include "LedBuzzer.h"
#include "Lcd1602.h"
#include "Time.h"
#include "main.h"

uint8 code WeekMod[] = {
    0x81,0xBD,0xBD,0x81,0x81,0xBD,0xBD,0x81,
    0xFF,0xFF,0xFF,0x00,0x00,0xFF,0xFF,0xFF,
    0xFF,0xFF,0xC3,0xFF,0xFF,0x81,0xFF,0xFF,
    0xFF,0x81,0xFF,0xC3,0xFF,0x00,0xFF,0xFF,
    0x00,0x5A,0x5A,0x5A,0x18,0x7E,0x7E,0x00,
    0xFF,0x81,0xF7,0x81,0xB7,0xB7,0x00,0xFF,
    0xF7,0xE7,0xFF,0x00,0xFF,0xDB,0xBD,0x7E,
};

bit staMute = 0;
uint8 AlarmHour = 0x12;
uint8 AlarmMin  = 0x00;
struct sTime CurTime;

uint8 SetIndex = 0;
uint8 pdata SetAlarmHour;
uint8 pdata SetAlarmMin;
struct sTime pdata SetTime;
void RefreshTime()
{
    GetRealTime(&CurTime);
    ShowLedNumber(5, CurTime.hour>>4, 0);
    ShowLedNumber(4, CurTime.hour&0xF,1);
    ShowLedNumber(3, CurTime.min>>4,  0);
    ShowLedNumber(2, CurTime.min&0xF, 1);
    ShowLedNumber(1, CurTime.sec>>4,  0);
    ShowLedNumber(0, CurTime.sec&0xF, 0);
    ShowLedArray(WeekMod + CurTime.week*8);
}
void RefreshDate(uint8 ops)
{
    uint8 pdata str[12];
    static uint8 backup = 0;
    if ((backup!=CurTime.day) || (ops!=0))
    {
        str[0] = ((CurTime.year>>12) & 0xF) + '0';
        str[1] = ((CurTime.year>>8) & 0xF) + '0';
        str[2] = ((CurTime.year>>4) & 0xF) + '0';
        str[3] = (CurTime.year & 0xF) + '0';
        str[4] = '-';
        str[5] = (CurTime.mon >> 4) + '0';
        str[6] = (CurTime.mon & 0xF) + '0';
        str[7] = '-';
        str[8] = (CurTime.day >> 4) + '0';
        str[9] = (CurTime.day & 0xF) + '0';
        str[10] = '\0';
        LcdShowStr(0, 1, str);
        backup = CurTime.day;
    }
}
void RefreshAlarm()
{
    uint8 pdata str[8];

    LcdShowStr(0, 0, "Alarm time ");
    str[0] = (AlarmHour >> 4) + '0';
    str[1] = (AlarmHour & 0xF) + '0';
    str[2] = ':';
    str[3] = (AlarmMin >> 4) + '0';
    str[4] = (AlarmMin & 0xF) + '0';
    str[5] = '\0';
    LcdShowStr(11, 0, str);
}
void AlarmMonitor()
{
    if ((CurTime.hour==AlarmHour) && (CurTime.min==AlarmMin))
    {
        if (!staMute)
            staBuzzer = ~staBuzzer;
        else
            staBuzzer = 0;
    }
    else
    {
        staMute = 0;
        staBuzzer = 0;
    }
}
void ShowSetTime()
{
    uint8 pdata str[18];

    str[0]  = ((SetTime.year>>4) & 0xF) + '0';
    str[1]  = (SetTime.year & 0xF) + '0';
    str[2]  = '-';
    str[3]  = (SetTime.mon >> 4) + '0';
    str[4]  = (SetTime.mon & 0xF) + '0';
    str[5]  = '-';
    str[6]  = (SetTime.day >> 4) + '0';
    str[7]  = (SetTime.day & 0xF) + '0';
    str[8]  = '-';
    str[9]  = (SetTime.week & 0xF) + '0';
    str[10] = ' ';
    str[11] = (SetTime.hour >> 4) + '0';
    str[12] = (SetTime.hour & 0xF) + '0';
    str[13] = ':';
    str[14] = (SetTime.min >> 4) + '0';
    str[15] = (SetTime.min & 0xF) + '0';
    str[16] = '\0';
    LcdShowStr(0, 0, "Set time:");
    LcdShowStr(0, 1, str);
}
void ShowSetAlarm()
{
    uint8 pdata str[8];

    str[0] = (SetAlarmHour >> 4) + '0';
    str[1] = (SetAlarmHour & 0xF) + '0';
    str[2] = ':';
    str[3] = (SetAlarmMin >> 4) + '0';
    str[4] = (SetAlarmMin & 0xF) + '0';
    str[5] = '\0';
    LcdShowStr(0, 0, "Set Alarm:");
    LcdShowStr(0, 1, str);
}
void CancelCurSet()
{
    staSystem = E_NORMAL;
    LcdCloseCursor();
    LcdClearScreen();
    RefreshTime();
    RefreshDate(1);
    RefreshTemp(1);
    RefreshAlarm();
}
void SetRightShift()
{
    if (staSystem == E_SET_TIME)
    {
        switch (SetIndex)
        {
            case 0: SetIndex=1;  LcdSetCursor(1, 1); break;
            case 1: SetIndex=2;  LcdSetCursor(3, 1); break;
            case 2: SetIndex=3;  LcdSetCursor(4, 1); break;
            case 3: SetIndex=4;  LcdSetCursor(6, 1); break;
            case 4: SetIndex=5;  LcdSetCursor(7, 1); break;
            case 5: SetIndex=6;  LcdSetCursor(9, 1); break;
            case 6: SetIndex=7;  LcdSetCursor(11,1); break;
            case 7: SetIndex=8;  LcdSetCursor(12,1); break;
            case 8: SetIndex=9;  LcdSetCursor(14,1); break;
            case 9: SetIndex=10; LcdSetCursor(15,1); break;
            default: SetIndex=0; LcdSetCursor(0, 1); break;
        }
    }
    else if (staSystem == E_SET_ALARM)
    {
        switch (SetIndex)
        {
            case 0: SetIndex=1;  LcdSetCursor(1,1); break;
            case 1: SetIndex=2;  LcdSetCursor(3,1); break;
            case 2: SetIndex=3;  LcdSetCursor(4,1); break;
            default: SetIndex=0; LcdSetCursor(0,1); break;
        }
    }
}
void SetLeftShift()
{
    if (staSystem == E_SET_TIME)
    {
        switch (SetIndex)
        {
            case 0: SetIndex=10; LcdSetCursor(15,1); break;
            case 1: SetIndex=0;  LcdSetCursor(0, 1); break;
            case 2: SetIndex=1;  LcdSetCursor(1, 1); break;
            case 3: SetIndex=2;  LcdSetCursor(3, 1); break;
            case 4: SetIndex=3;  LcdSetCursor(4, 1); break;
            case 5: SetIndex=4;  LcdSetCursor(6, 1); break;
            case 6: SetIndex=5;  LcdSetCursor(7, 1); break;
            case 7: SetIndex=6;  LcdSetCursor(9, 1); break;
            case 8: SetIndex=7;  LcdSetCursor(11,1); break;
            case 9: SetIndex=8;  LcdSetCursor(12,1); break;
            default: SetIndex=9; LcdSetCursor(14,1); break;
        }
    }
    else if (staSystem == E_SET_ALARM)
    {
        switch (SetIndex)
        {
            case 0: SetIndex=3;  LcdSetCursor(4,1); break;
            case 1: SetIndex=0;  LcdSetCursor(0,1); break;
            case 2: SetIndex=1;  LcdSetCursor(1,1); break;
            default: SetIndex=2; LcdSetCursor(3,1); break;
        }
    }
}
void InputSetNumber(uint8 ascii)
{
    uint8 num;
```

```
num = ascii - '0';
if (num <= 9)
{
    if (staSystem == E_SET_TIME)
    {
        switch (SetIndex)
        {
            case 0: SetTime.year = (SetTime.year&0xFF0F)|(num<<4);
                    LcdShowChar(0, 1, ascii);  break;
            case 1: SetTime.year = (SetTime.year&0xFFF0)|(num);
                    LcdShowChar(1, 1, ascii);  break;
            case 2: SetTime.mon = (SetTime.mon&0x0F)|(num<<4);
                    LcdShowChar(3, 1, ascii);  break;
            case 3: SetTime.mon = (SetTime.mon&0xF0)|(num);
                    LcdShowChar(4, 1, ascii);  break;
            case 4: SetTime.day = (SetTime.day&0x0F)|(num<<4);
                    LcdShowChar(6, 1, ascii);  break;
            case 5: SetTime.day = (SetTime.day&0xF0)|(num);
                    LcdShowChar(7, 1, ascii);  break;
            case 6: SetTime.week = (SetTime.week&0xF0)|(num);
                    LcdShowChar(9, 1, ascii);  break;
            case 7: SetTime.hour = (SetTime.hour&0x0F)|(num<<4);
                    LcdShowChar(11,1, ascii);  break;
            case 8: SetTime.hour = (SetTime.hour&0xF0)|(num);
                    LcdShowChar(12,1, ascii);  break;
            case 9: SetTime.min = (SetTime.min&0x0F)|(num<<4);
                    LcdShowChar(14,1, ascii);  break;
            default:SetTime.min = (SetTime.min&0xF0)|(num);
                    LcdShowChar(15,1, ascii);  break;
        }
        SetRightShift();
    }
    else if (staSystem == E_SET_ALARM)
    {
        switch (SetIndex)
        {
            case 0: SetAlarmHour = (SetAlarmHour&0x0F) | (num<<4);
                    LcdShowChar(0,1, ascii); break;
```

```
                case 1: SetAlarmHour = (SetAlarmHour&0xF0) | (num);
                        LcdShowChar(1,1, ascii); break;
                case 2: SetAlarmMin = (SetAlarmMin&0x0F) | (num<<4);
                        LcdShowChar(3,1, ascii); break;
                default:SetAlarmMin = (SetAlarmMin&0xF0) | (num);
                        LcdShowChar(4,1, ascii); break;
            }
            SetRightShift();
        }
    }
}
void SwitchSystemSta()
{
    if (staSystem == E_NORMAL)
    {
        staSystem = E_SET_TIME;
        SetTime.year = CurTime.year;
        SetTime.mon  = CurTime.mon;
        SetTime.day  = CurTime.day;
        SetTime.hour = CurTime.hour;
        SetTime.min  = CurTime.min;
        SetTime.sec  = CurTime.sec;
        SetTime.week = CurTime.week;
        LcdClearScreen();
        ShowSetTime();
        SetIndex = 255;
        SetRightShift();
        LcdOpenCursor();
    }
    else if (staSystem == E_SET_TIME)
    {
        staSystem = E_SET_ALARM;
        SetTime.sec = 0;
        SetRealTime(&SetTime);
        SetAlarmHour = AlarmHour;
        SetAlarmMin  = AlarmMin;
        LcdClearScreen();
        ShowSetAlarm();
        SetIndex = 255;
        SetRightShift();
    }
    else
    {
        staSystem = E_NORMAL;
        AlarmHour = SetAlarmHour;
        AlarmMin  = SetAlarmMin;
        LcdCloseCursor();
        LcdClearScreen();
        RefreshTime();
        RefreshDate(1);
        RefreshTemp(1);
        RefreshAlarm();
    }
}
void RefreshTime2()
{
    unsigned char timenow[10];
    uint8 pdata str[12];
    GetRealTime(&CurTime);
        timenow[0]=((CurTime.hour>>4) & 0xF) + '0';
        timenow[1]=(CurTime.hour& 0xF) + '0';
        timenow[2]='.';
        timenow[3]=((CurTime.min>>4) & 0xF) + '0';
        timenow[4]=(CurTime.min & 0xF) + '0';
        timenow[5]='.';
        timenow[6]=((CurTime.sec>>4) & 0xF) + '0';
        timenow[7]=(CurTime.sec & 0xF) + '0';
        timenow[8]='\0';
        LcdShowStr(0,0,timenow);
        str[0] = ((CurTime.year>>12) & 0xF) + '0';  //4位数年份
        str[1] = ((CurTime.year>>8) & 0xF) + '0';
        str[2] = ((CurTime.year>>4) & 0xF) + '0';
        str[3] = (CurTime.year & 0xF) + '0';
        str[4] = '-';                          //分隔符
        str[5] = (CurTime.mon >> 4) + '0';     //月份
        str[6] = (CurTime.mon & 0xF) + '0';
        str[7] = '-';                          //分隔符
        str[8] = (CurTime.day >> 4) + '0';     //日期
        str[9] = (CurTime.day & 0xF) + '0';
        str[10] = '\0';         //字符串结束符
        LcdShowStr(0, 1, str);
}
void ShowSecondTep()
{
    if(staSystem == E_NORMAL)
    {
        staSystem = E_NORMAL_2;
        LcdCloseCursor();
        LcdClearScreen();
        RefreshTime2();
    }
    else if(staSystem == E_NORMAL_2)
    {
        staSystem = E_NORMAL;
        LcdCloseCursor();
        LcdClearScreen();
        RefreshTime();
        RefreshDate(1);
        RefreshTemp(1);
        RefreshAlarm();
    }
}
void KeyAction(uint8 keycode)
{
    if  ((keycode>='0') && (keycode<='9'))
    {
        InputSetNumber(keycode);
    }
    else if (keycode == 0x25)
    {
        SetLeftShift();
    }
    else if(keycode==0x26)
    {
        ShowSecondTep();
    }
    else if (keycode == 0x27)
    {
        SetRightShift();
    }
    else if (keycode == 0x0D)
    {
        SwitchSystemSta();
    }
    else if (keycode == 0x1B)
    {
        if (staSystem == E_NORMAL)
        {
            staMute = 1;
        }
        else
        {
            CancelCurSet();
        }
    }
}
```