Copyright

beta版本, 是为竞赛准备的版本, 相较于 'lib' 可能会有缺省或精简的部分, 模块封装耦合度更高

目录

项目架构

- main
  - STARTUP.A51
  - main.c
  - system.h
  - system.c: 如果功能复杂, 可以将功能部分丢进来, 减少 'main.c' 的体积
- header:
  - __type.h__.h: 定义 'u8', 'u16'
  - __config__.h: 定义所有引脚宏和申明延迟函数
  - __function.h__.h: 申明所有模块需要用到的函数
- include:
  - *.h: 一些基类模块定义头更方便 eg.'i2c'
- source:
  - *.c: 所有模块的实现

项目框架

main.c

```c
#include "system.h"

void IO_Init()
{
}

void INT_Init(void)
{
}

void main(void)
{
    // 初始化
    while (1)
    {
    }
}

// 中断函数
```

system.c

```c
#include "__config__.h"
#include "__function__.h"
// 函数实现
```

__type.h__.h

```
#ifndef __TYPE___H
#define __TYPE___H

typedef unsigned char u8;
typedef unsigned int u16;

#endif // __TYPE___H
```

__config__.h

```
#ifndef __CONFIG___H
#define __CONFIG___H

#include <STC15.H>
#include "__type__.h"

// 定义延迟函数和引脚宏

#endif // __CONFIG___H
```

__function.h__.h

```
// 函数的申明
#ifndef __FUNCTION___H
#define __FUNCTION___H

#include "__type__.h"
// 函数申明

#endif // __FUNCTION___H
```

模块库

Delay 24MHz

```c
// delay.c
#include "__type__.h"

extern void _nop_(void);

void Delay5us(u8 t)
{ // 误差 0%
    u8 i;
    _nop_();
    _nop_();
    while (t)
    {
        i = t == 1 ? 21 : 26;
        while (--i)
            ;
        t--;
    }
}

void Delay1ms(u16 t)
{ // 平均误差不足 0.005% 越大误差越小, 几乎是无误差
    u8 i, j;
    while (t)
    {
        for (i = 24; i; --i)
            for (j = 248; j; --j)
                ;
        for (i = 8; i; --i)
            ;
        --t;
    }
}
```

Delay 11.0592MHz

```c
// delay.c
// STC15 11.0592MHz
void Delay10us(u8 t)
{ // 平均误差: 0.37%
    u8 i;
    _nop_();
    while (t)
    {
        _nop_();
        _nop_();
        _nop_();
        i = t == 1 ? 18 : 23;
        while (--i);
        --t;
    }
}

void Delay1ms(u16 t)
{ // 平均误差: 0.2%
    u8 i, j;
    while (t)
    {
        for (i = 11; i; --i)
            for (j = 250; j; --j)
                ;
        --t;
    }
}
```

I2C

需要定义 SDA 和 SCL 引脚

```
// __config__.h
#define I2C_SDA P6 ^ 6
#define I2C_SCL P6 ^ 7
```

```
// i2c.h
#ifndef I2C_H
#define I2C_H

#include "__type__.h"

void I2C_Start(void);

void I2C_Stop(void);

void I2C_Ack(bit a);

bit I2C_CheckAck(void);

void I2C_Transmit(u8 dat);

u8 I2C_Receive(void);

#endif // I2C_H
```

```c
// i2c.c
#include "i2c.h"
#include "__config__.h"

sbit SDA = I2C_SDA; // 数据
sbit SCL = I2C_SCL; // 时钟

void I2C_Start()
{
    SCL = 1;
    SDA = 1;
    Delay5us(1); // >4.7us
    SDA = 0;
    Delay5us(1); // >4us
    SCL = 0;
}

void I2C_Stop()
{
    SCL = 0;
    SDA = 0;
    SCL = 1;
    Delay5us(1); // >4us
    SDA = 1;
    Delay5us(1); // >4.7us
}

void I2C_Ack(bit a)
{
    SDA = !a;
    SCL = 1;
    Delay5us(1); // >4us
    SCL = 0;
    SDA = a;
}

bit I2C_CheckAck(void)
{
    bit na;
    SDA = 1;
    Delay5us(1); // >4us
    SCL = 1;
    Delay5us(1); // >4us
    na = SDA;
    SCL = 0;
    return !na;
}

void I2C_Transmit(u8 byte)
{
    SCL = 0;
    FOR_C(8)
```

```
    {
        SDA = (bit)(byte & 0x80);
        SCL = 1;
        Delay5us(1);
        SCL = 0;
        byte <<= 1;
        Delay5us(1);
    }
    SDA = 1;
    Delay5us(1);
}

u8 I2C_Receive(void)
{
    u8 dat;
    SDA = 1;
    FOR_C(8)
    {
        dat <<= 1;
        SCL = 0;
        Delay5us(1);
        SCL = 1;
        Delay5us(1);
        dat |= SDA;
    }
    SCL = 0;
    Delay5us(1);
    return dat;
}
```

KEY_4x4

需要在 '__config__.h' 中定义

```
// __config__.h
#define KEY_4X4_PIN P7
#define KEY_4X4_DELAY Delay1ms(5)
```

```
// key_4x4.c
#include "__config__.h"

#define PIN KEY_4X4_PIN

u8 SwitchValue(u8 v)
{
    switch (v)
    {
    case 0x0e:
    case 0xe0: return 0x00;
    case 0x0d: return 0x01;
    case 0xd0: return 0x04;
    case 0x0b: return 0x02;
    case 0xb0: return 0x08;
    case 0x07: return 0x03;
    case 0x70: return 0x0c;
    default:
        return 0xff;
    }
}

u8 KEY_4X4_Value(void)
{
    u8 n;
    PIN = 0x0f; // 列检测
    if (PIN != 0x0f)
    {
        n = PIN;
        KEY_4X4_DELAY; // 按键消抖
        if (n == PIN)
        {
            PIN = 0xf0; // 行检测
            n = SwitchValue(n);
            return (SwitchValue(PIN) | n);
        }
    }
    return 0xff;
}
```

AT24C

```c
// 需要依赖 I2C
// at24c.c
#include "__config__.h"

#include "i2c.h"

bit AT24C_Call(u16 addr)
{
    I2C_Start();
    I2C_Transmit(0xa0);
    if (I2C_CheckAck())
    {
        I2C_Transmit(addr >> 8);
        if (I2C_CheckAck())
        {
            I2C_Transmit(addr & 0xff);
            if (I2C_CheckAck())
            {
                return 1;
            }
        }
    }
    return 0;
}

void AT24C_Read(u16 addr, u8* dat, u8 len)
{
    if (AT24C_Call(addr))
    {
        I2C_Start(); // 重新启动I2C总线
        I2C_Transmit(0xa1);
        if (I2C_CheckAck())
        {
            FOR_C(len - 1)
            {
                *dat++ = I2C_Receive();
                I2C_Ack(1);
            }
            *dat = I2C_Receive();
            I2C_Ack(0);
        }
    }
    I2C_Stop();
}

// 页写入时序 最多为 页写缓冲器的大小 字节 超出会被循环覆盖 且此函数不带延迟
void AT24C_Write(u16 addr, u8* dat, u8 len)
{
```

```
    if (AT24C_Call(addr))
    {
        FOR_C(len)
        {
            I2C_Transmit(*dat);
            dat++;
            if (!I2C_CheckAck())
                break;
        }
    }
    I2C_Stop();
    Delay1ms(10);
}
```

LCD12864

指令表 (不全, 部分几乎完全不用的指令省了)

指令	普通指令	扩展指令
0x01	清屏	待命模式
0x02	光标返回	
0x03		允许输入卷动地址
0x04	模式光标左移	第一行反白
0x05	模式文字右移	第二行反白
0x06	模式光标右移	第三行反白
0x07	模式文字左移	第四行反白
0x08	屏幕关闭	进入睡眠模式
0x0c	亮屏光标关闭	退出睡眠模式
0x0e	亮屏光标开启	
0x0f	亮屏光标闪烁	
0x10	光标左移	
0x14	光标右移	
0x18	文字左移	
0x1c	文字右移	
0x30	普通指令	普通指令
0x34	扩展指令	扩展指令
0x36		绘图模式打开
0x40-0x7f		卷动/IRAM地址
0x80	第1行	
0x90	第2行	
0x88	第3行	
0x98	第4行	
0x80-0xff		设定绘图RAM

```
// __config__.h
#define LCD12864_DATA P0
#define LCD12864_RS P2 ^ 0
#define LCD12864_RW P2 ^ 1
#define LCD12864_EN P2 ^ 2
```

```c
// lcd12864.h
#define DT LCD12864_DATA
sbit RS = LCD12864_RS;
sbit RW = LCD12864_RW;
sbit EN = LCD12864_EN;

void LCD12864_Write(bit rs, u8 byte)
{
    EN = 0;
    RS = rs;
    RW = 0;
    DT = byte;
    EN = 1;
    EN = 0;
    Delay5us(15); // 至少72us
}

void LCD12864_Cmd(u8 cmd)
{
    LCD12864_Write(0, cmd);
    if (cmd == 0x01)
    {
        Delay5us(200);
        Delay5us(105); // clear need 1.6ms
    }
}

void LCD12864_Show(u8 dat)
{
    LCD12864_Write(1, dat);
}

void LCD12864_Init(void)
{
    LCD12864_Cmd(0x01);
    LCD12864_Cmd(0x06); // 默认为 06 可以不要
    LCD12864_Cmd(0x0c);
    LCD12864_Cmd(0x30);
}

void LCD12864_String(u8* s)
{
    while (*s)
    {
        LCD12864_Show(*s++);
    }
}
```

DS18B20

```
// __config__.h
#define DS18B20_DQ P1 ^ 7

// ds18b20.c
#include "__config__.h"

/*
 * 分辨率: 最大转换时间 默认是 12位
 *   00:  9位, 93.75 ms
 *   01: 10位, 187.5 ms
 *   10: 11位,   375 ms
 *   11: 12位,   750 ms
 * 转换后需要等待转换完成后才能读取正确的温度
 *
*/

#define SKIP_ROM 0xcc
#define CONVERT_T 0x44
#define READ_SCRATCHPAD 0xbe

#define WRITE_SCRATCHPAD 0x4e // 分别写 上限 下限 分辨率(高三位((<<5)|0x1f))
#define COPY_SCRATCHPAD 0x48 // 将其保存在ROM

sbit DQ = DS18B20_DEFINE_DQ;

/**
 * @return: 0-存在; 1-不存在;
*/
bit DS18B20_Check(void)
{
    bit x;
    DQ = 1; // 复位DQ
    DQ = 0;
    Delay5us(120); // 拉低 480~960us
    DQ = 1;
    Delay5us(24); // 等待 15~60us 240us之内
    x = DQ;                    // 总线60~240us低电平
    DQ = 1;                    // 释放总线
    Delay5us(48); // 保证时序完整 480us * 2
    return x;
}

u8 DS18B20_ReadByte(void)
{
    u8 dat = 0; // 存储读数据初始化 0
    FOR_C(8)
    { // 串行读8位数据，先读低位后读高位
        DQ = 0; // 拉低
```

```
        Delay5us(1);
        DQ = 1; // 15μs内拉高释放总线
        dat >>= 1;
        if (DQ)
            dat |= 0x80;
        Delay5us(9); // 每个读时段 最少60us
    }
    return dat;
}

void DS18B20_WriteByte(u8 dat)
{
    FOR_C(8)
    { // 串行写8位数据，先写低位后写高位
        DQ = 0; // 拉低
        Delay5us(1); // 至少间隔1us 低于15us
        DQ = dat & 0x01;          // 写 '1' 在15μs内拉高
        Delay5us(9); // 写 '0' 拉低60μs 10+50
        DQ = 1;
        dat >>= 1;
    }
}

/* ================================================================================ */

void DS18B20_Convert() // 温度转换
{
    if (DS18B20_Check())
        return;
    DS18B20_WriteByte(SKIP_ROM);
    DS18B20_WriteByte(CONVERT_T);
}

float DS18B20_ReadTemp() // 温度读取
{
    int n;
    if (DS18B20_Check())
        return 0xffff;

    DS18B20_WriteByte(SKIP_ROM);
    DS18B20_WriteByte(READ_SCRATCHPAD);
    n = DS18B20_ReadByte();
    n |= ((int)DS18B20_ReadByte()) << 8;
    return n * 0.0625;
}
```

DS1302

```c
// __config__.h
#define DS1302_SCK P3 ^ 5
#define DS1302_SDA P3 ^ 6
#define DS1302_CE  P5 ^ 4

// ds1302.c
#include "__config__.h"

sbit SCK = DS1302_SCK; //时钟
sbit SDA = DS1302_SDA; //数据
sbit CE = DS1302_CE;   // DS1302复位

// 秒分时日月周年
u8 code addr[] = { 0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c };

void DS1302_WriteByte(u8 byte)
{
    FOR_C(8)
    {
        SCK = 0;
        SDA = byte & 0x01;
        byte >>= 1;
        SCK = 1;
    }
}

void DS1320_Write(u8 addr, u8 dat)
{
    CE = 0;
    _nop_();
    SCK = 0;
    _nop_();
    CE = 1; //启动
    _nop_();
    DS1302_WriteByte(addr); //发送地址
    DS1302_WriteByte(dat);  //发送数据
    CE = 0;                 //恢复
}

u8 DS1320_ReadBCD(u8 addr)
{
    u8 dat = 0x00;
    CE = 0;
    _nop_();
    _nop_();
    SCK = 0;
    _nop_();
    _nop_();
```

```
    CE = 1;
    _nop_();
    _nop_();

    DS1302_WriteByte(addr);

    FOR_C(8) //循环8次 读取数据
    {
        if (SDA)
            dat |= 0x80; //每次传输低字节
        SCK = 0;
        dat >>= 1; //右移一位
        _nop_();
        _nop_();
        _nop_();
        SCK = 1;
    }

    CE = 0;
    _nop_(); // 以下为DS1302复位的稳定时间
    _nop_();
    CE = 0;
    SCK = 0;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    SCK = 1;
    _nop_();
    _nop_();
    SDA = 0;
    _nop_();
    _nop_();
    SDA = 1;
    _nop_();
    _nop_();

    return (dat / 16 * 10) + (dat & 0x0f); //返回
}

//主要设置时钟芯片里的 秒分时日月周年
void DS1320_SetRTC(u8* set) //设定 日历
{
    u8 i;
    DS1320_Write(0x8E, 0x00); //写使能
    for (i = 0; i < 8; ++i)
    {
        DS1320_Write(addr[i], ((set[i] / 10 * 16) + (set[i] % 10)));
    }
    DS1320_Write(0x8E, 0x80); //写禁止
}
```

HC_SR04

```
// __config__.h
#define HC_SR04_TRIG P1 ^ 5
#define HC_SR04_ECHO P3 ^ 4

// hc_sr04.c
#include "__config__.h"

sbit TRIG = HC_SR04_TRIG;
sbit ECHO = HC_SR04_ECHO;

float HC_SR04_Result(void)
{
    AUXR &= 0x7f;           //定时器时钟12T模式
    TMOD &= 0xf0;           //设置定时器模式
    TMOD |= 0x01;           //设置定时器模式
    TH0 = TL0 = TR0 = TF0 = 0; // 最长 32768us
    TRIG = 0;
    TRIG = 1; // 触发信号
    Delay5us(2);
    TRIG = 0;
    FOR_C(65535)
        if (ECHO)
            break; // 等待开始探测
    TR0 = 1;
    while (ECHO && TF1 == 0)
        ; // 等待结束探测
    if (TF0)
    {
        TF0 = 0;
        return 0;
    }
    TR0 = 0;
    return ((((u16)TH0 << 8) | TL0) >> 1) * 0.017; // cm
}
```

ADC

```c
// 内置ADC模块 使用P1口
// adc.c
#include "__config__.h"

// 定义 ADC_CONTR 寄存器位取值
#define ADC_POWER 0x80 // ADC power control bit
#define ADC_FLAG 0x10  // ADC complete flag 模数转换结束标志位
// ADC start control bit  模数转换启动控制位
// 每当手动将其置 1 后, AD转换开始, 当AD转换结束后这个位就会自动置 0
#define ADC_START 0x08

// 转换速度控制位SPEED0和SPEED1，共四种状态，对应四种转换速度
#define ADC_SPEED_0 0x00 // 540 clocks
#define ADC_SPEED_1 0x20 // 360 clocks
#define ADC_SPEED_2 0x40 // 180 clocks
#define ADC_SPEED_3 0x60 // 90 clocks


/**
 * @param ch: 0-7 选择 P10-P17
*/
u16 ADC_Result(u8 ch)
{
    u16 Vo;
    P1ASF = (1 << ch); // 选择P1口的哪一口 这里的口和 ch 要对应才能达到选择该口
    // 开启ADC,采集 P1^ch 引脚的值
    ADC_CONTR = ADC_POWER | ADC_SPEED_0 | ADC_START | ch;
    // 这么用语句的主要原因就是不能位寻址
    // 通道选择在后3位所以直接用一个整数表示ch

    // 设置 ADC_CONTR 寄存器后需加4个CPU时钟周期的延时，才能保证值被写入 ADC_CONTR 寄存器
    _nop_();
    _nop_();
    _nop_();
    _nop_();

    while (!(ADC_CONTR & ADC_FLAG))
        ; //检测是否ADC完成

    ADC_CONTR &= ~ADC_FLAG; // 转换标志位: 手动将其置0 等待下次硬件置1

    /**
     * @param CLK_DIV: 第5位控制存储模式
     * 0: ADC_RES(高8位) + ADC_RESL(低2位)
     * 1: ADC_RESL(高2位) + ADC_RES(低8位)
    */
    Vo = ADC_RES;
    Vo = (Vo << 2) | ADC_RESL;
```

```
    return Vo;
}
```

示例程序

__type__.h

```h
#ifndef __TYPE___H
#define __TYPE___H

typedef unsigned char u8;
typedef unsigned int u16;

#endif // __TYPE___H
```

__config__.h

```h
// __config__.h
#ifndef __CONFIG___H
#define __CONFIG___H

#include <STC15.H>

#include "__type__.h"

// 很多模块可能会用到, 所以放在 config 中申明
extern void _nop_(void);
extern void Delay5us(u8 t);
extern void Delay1ms(u16 t);

// lcd12864
#define LCD12864_DATA P0
#define LCD12864_RS P2 ^ 0
#define LCD12864_RW P2 ^ 1
#define LCD12864_EN P2 ^ 2

// key_4x4
#define KEY_4X4_PIN P7
#define KEY_4X4_DELAY Delay1ms(5)

// ds18b20
#define DS18B20_DQ P1 ^ 7

// ds1302
#define DS1302_SCK P3 ^ 5
#define DS1302_SDA P3 ^ 6
#define DS1302_CE  P5 ^ 4

// hc_sr04
#define HC_SR04_TRIG P1 ^ 5
#define HC_SR04_ECHO P3 ^ 4

// i2c
#define I2C_SDA P6 ^ 6
#define I2C_SCL P6 ^ 7

// adc
#define VOLTAGE 3.3
#define LDR 0 // 光敏电阻 电位器P10

// 定义一些有用的宏函数
static u16 s_count_i;
#define FOR_C(I) for (s_count_i = (I); s_count_i; --s_count_i)

#endif // __CONFIG___H
```

__function__.h

```h
#ifndef __FUNCTION___H
#define __FUNCTION___H

#include "__type__.h"

// cstdio
extern int sprintf(char*, const char*, ...);

// delay
extern void Delay5us(u8 t);
extern void Delay1ms(u16 t);

// lcd12864
extern void LCD12864_Cmd(u8 cmd);
extern void LCD12864_Show(u8 cmd);
extern void LCD12864_Init(void);
extern void LCD12864_String(u8* s);

// key_4x4
extern u8 KEY_4X4_Value(void);

// ds18b20
extern void DS18B20_Convert();
extern float DS18B20_ReadTemp();

// ds1302
extern u8 DS1320_ReadBCD(u8 addr);
extern void DS1320_SetRTC(u8* set);

// hc_sr04
extern float HC_SR04_Result(void);

// at24c256
extern void AT24C_Read(u16 addr, u8* dat, u8 len);
extern void AT24C_Write(u16 addr, u8* dat, u8 len);

// adc
extern u16 ADC_Result(u8 ch);

#endif // __FUNCTION___H
```

main.c

```c
#include "__config__.h"
#include "__function__.h"

u8 getKey(void);
void resetPassword(void);
void enterPassword(void);
void updateShow(void);
void showTime(void);
void showTemperature(void);
void TempMotorRun(bit s, u16 size);

u8 num = '0';
u8 count_x1 = 0;
u8 count_t2 = 0;
u8 key;
u8 xdata buff[33];
u8 code date_time[] = {0, 0, 8, 27, 4, 6, 24};

#define PM_Z(P) (P##M0 = P##M1 = 0x00)                          // 置零
#define PPO(P, B) (P##M1 &= ~(1 << (B)), P##M0 |= (1 << (B))) // 推挽输出

void IO_Init()
{
    PM_Z(P0);
    PM_Z(P1);
    PM_Z(P2);
    PM_Z(P3);
    PM_Z(P4);
    PM_Z(P5);
    PM_Z(P6);
    PM_Z(P7);

    // LCD12864
    PPO(P2, 2);
    PPO(P2, 1);
    PPO(P2, 0);

    // 直流电机
    PPO(P1, 5); // 智慧农业
    PPO(P1, 6); // 智慧小车

    // 步进电机
    PPO(P1, 1);
    PPO(P1, 2);
    PPO(P1, 3);
    PPO(P1, 4); // 和智慧农业的红外冲突

    // 蜂鸣器
    PPO(P5, 5);
```

```
}

void INT_Init(void)
{
    // EA ES ET EX IT(外部中断方式控制)
    // 串口通信: RI TI
    // IE2: - ET4 ET3 ES4 ES3 ET2 - ES2
    // interrupt: t2: 12 t3: 19 t4: 20
    // 定时器/串口可以软件生成

    SCON = 0x50; //REN=1允许串行接受状态，串口工作模式2
    TMOD = 0x00; //定时器1为模式0（16位自动重载）
    AUXR = 0X40; //开启1T模式
    TL1 = (65535 - (24000000 / 4 / 9600)); //设置波特率重装值
    TH1 = (65535 - (24000000 / 4 / 9600)) >> 8;
    TR1 = 1; //开启定时器1
    ES = 1;  //开串口中断

    AUXR &= 0xFB; // 定时器时钟12T模式
    T2L = 0xB0;   // 设置定时初始值
    T2H = 0x3C;   // 设置定时初始值
    AUXR |= 0x10; // 定时器2开始计时

    EA = 1;
    IE2 |= 0x04; // 定时器2中断
    IT1 = 1;     // 外部中断低电平触发
    EX1 = 1;     // 外部中断允许
}

void main(void)
{
    IO_Init();
    INT_Init();
    LCD12864_Init();
    DS18B20_Convert();
    DS1320_SetRTC(date_time);


    while (1)
    {
        LCD12864_Cmd(0x80);
        sprintf(buff, "NUM:  %c", num);
        LCD12864_String(buff);
    }

    if (KEY_4X4_Value() == 0x0f)
    {
        LCD12864_Cmd(0x80);
        LCD12864_String("About To");
        LCD12864_Cmd(0x90);
        LCD12864_String("Reset Password");
        while (KEY_4X4_Value() != 0xff)
```

```
            ;
        resetPassword();
    }

    enterPassword();

    while (1)
    {
        updateShow();

        if (getKey() != 0xff)
        {
            switch (key)
            {
            case 0:
                TempMotorRun(0, 512);
                break;
            case 1:
                TempMotorRun(1, 512);
                break;
            case 2:
                P15 = !P15;
                P16 = !P16;
                break;
            default:
                break;
            }
        }
    }
}

u8 getKey(void)
{
    key = KEY_4X4_Value();
    // while (1)
    // {
    //     if ((key | KEY_4X4_Value()) == 0xff)
    //     {
    //         Delay1ms(5);
    //         if ((key | KEY_4X4_Value()) == 0xff)
    //             break;
    //     }
    // } // 消抖 (其实没必要)
    while ((key | KEY_4X4_Value()) != 0xff)
        ;
    return key;
}

void resetPassword(void)
{
    bit loop = 1;
    u8 num;
```

```
    u8 i = 0, max = 8;
    LCD12864_Cmd(0x01);
    LCD12864_Cmd(0x80);
    LCD12864_String("New Password:");
    LCD12864_Cmd(0x90);
    LCD12864_Cmd(0x0e);
    while (loop)
    {
        num = 0xff;
        if (getKey() != 0xff)
        {
            switch (key)
            {
            case 0x00:
                num = 1;
                break;
            case 0x01:
                num = 2;
                break;
            case 0x02:
                num = 3;
                break;
            case 0x04:
                num = 4;
                break;
            case 0x05:
                num = 5;
                break;
            case 0x06:
                num = 6;
                break;
            case 0x08:
                num = 7;
                break;
            case 0x09:
                num = 8;
                break;
            case 0x0a:
                num = 9;
                break;
            case 0x0d:
                num = 0;
                break;
            case 0x03: // 退格
                if (i)
                {
                    buff[i] = 0;
                    LCD12864_Cmd(0x10);
                    LCD12864_String("  ");
                    i--;
                    LCD12864_Cmd(0x10);
                }
```

```
                break;
            case 0x0f: // 确认
                loop = 0;
                AT24C_Write(0x0000, &i, 1);
                AT24C_Write(0x0001, buff, i);
                break;
            default:
                break;
            }
            if (num != 0xff && i < max)
            {
                buff[i] = num + '0';
                LCD12864_Show(buff[i]);
                LCD12864_Show(0x20);
                buff[++i] = 0;
                num = 0xff;
            }
        }
    }
    LCD12864_Cmd(0x0c);
    LCD12864_Cmd(0x01);
}

void enterPassword(void)
{
    bit loop, lock = 1;
    u8 right, num, len, i;
    u8 max = 8; // 0-7
    while (lock)
    {
        loop = 1;
        right = 0;
        i = 0;
        AT24C_Read(0x0000, &len, 1);
        AT24C_Read(0x0001, buff + 9, len);
        LCD12864_Cmd(0x01);
        LCD12864_Cmd(0x80);
        LCD12864_String("Enter Password:");
        LCD12864_Cmd(0x90);
        LCD12864_Cmd(0x0e);
        while (loop)
        {
            num = 0xff;
            if (getKey() != 0xff)
            {
                switch (key)
                {
                case 0x00:
                    num = 1;
                    break;
                case 0x01:
                    num = 2;
```

```
            break;
        case 0x02:
            num = 3;
            break;
        case 0x04:
            num = 4;
            break;
        case 0x05:
            num = 5;
            break;
        case 0x06:
            num = 6;
            break;
        case 0x08:
            num = 7;
            break;
        case 0x09:
            num = 8;
            break;
        case 0x0a:
            num = 9;
            break;
        case 0x0d:
            num = 0;
            break;
        case 0x03: // 退格
            if (i)
            {
                buff[i] = 0;
                LCD12864_Cmd(0x10);
                LCD12864_String("  ");
                if (right == i)
                {
                    right--;
                }
                i--;
                LCD12864_Cmd(0x10);
            }
            break;
        case 0x0f: // 确认
            loop = 0;
            break;
        default:
            break;
        }
        if (num != 0xff && i < max)
        {
            buff[i] = num + '0';
            LCD12864_Show(buff[i]);
            LCD12864_Show(0x20);
            if (right == i)
            {
```

```
                        right += ((buff[i] == buff[i + 9]) | (i >= len));
                    }
                    buff[++i] = 0;
                }
            }
        }
        LCD12864_Cmd(0x0c);
        LCD12864_Cmd(0x01);

        lock = (right != len);

        if (!lock)
        {
            LCD12864_String("Right Password");
        }
        else
        {
            LCD12864_String("Wrong Password");
            // buff[len + 9] = 0;
            // LCD12864_Cmd(0x90);
            // LCD12864_String(buff + 9);
        }
        LCD12864_Cmd(0x88);
        LCD12864_String("Enter Any Key...");
        while (getKey() == 0xff)
            ;
    }
    LCD12864_Cmd(0x01);
}

void updateShow(void)
{
    showTime();
    if (count_t2 > 20)
    {
        showTemperature();
        count_t2 = 0;

        sprintf(buff, "%02d", (u16)count_x1);
        LCD12864_Cmd(0x9f);
        LCD12864_String(buff);
        count_x1 = 0;

        sprintf(buff, "%4.2f", ADC_Result(LDR) * VOLTAGE / 1024);
        LCD12864_Cmd(0x86);
        LCD12864_String(buff); // 显示电压值
    }
}

void showTime(void)
{
    sprintf(
```

```
        buff,
        "20%02d-%02d-%02d",
        (u16)DS1320_ReadBCD(0x8d),
        (u16)DS1320_ReadBCD(0x89),
        (u16)DS1320_ReadBCD(0x87)
    );
    LCD12864_Cmd(0x80);
    LCD12864_String(buff);
    sprintf(
        buff,
        "%02dh %02dm %02ds",
        (u16)DS1320_ReadBCD(0x85),
        (u16)DS1320_ReadBCD(0x83),
        (u16)DS1320_ReadBCD(0x81)
    );
    LCD12864_Cmd(0x90);
    LCD12864_String(buff);
}

void showTemperature(void)
{
    EA = 0;
    sprintf(buff, "T: %6.2f C", DS18B20_ReadTemp());
    DS18B20_Convert();
    LCD12864_Cmd(0x98);
    LCD12864_String(buff);

    sprintf(buff, "D: %5.1f cm", HC_SR04_Result());
    LCD12864_Cmd(0x88);
    LCD12864_String(buff);
    EA = 1;
}

void TempMotorRun(bit s, u16 size)
{
    char i;
    u8 code spm_turn[] = {
        0x02,
        0x06,
        0x04,
        0x0c,
        0x08,
        0x09,
        0x01,
        0x03,
    };

    FOR_C(size)
    {
        if (s)
        {
            for (i = 0; i < 8; ++i)
```

```
            {
                P1 &= 0xe1;
                P1 |= spm_turn[i] << 1;
                Delay5us(255);
            }
        }
        else
        {
            for (i = 7; i >= 0; --i)
            {
                P1 &= 0xe1;
                P1 |= spm_turn[i] << 1;
                Delay5us(255);
            }
        }
    }
}

void int_x1(void) interrupt 2
{
    count_x1++;
}

void int_t2(void) interrupt 12
{
    count_t2++;
}

void int_4(void) interrupt 4 using 1
{
    if (RI)
    {
        num = '0' + SBUF;
        RI = 0;
    }
}
```