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

依赖

- `__type.h__.h`: 定义 `u8`, `u16`, `u32`
- `__config__.h`: 定义所有引脚宏和申明延迟函数
- `delay.c`: 根据单片机实现延迟
  - `void delay_5us(u8);`
  - `void delay_1ms(u16);`

KEY_4X4

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

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	令第一行反白
0x05	模式文字右移	令第二行反白
0x06	模式光标右移	令第三行反白
0x07	模式文字左移	令第四行反白
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 LCD_DATA P0
#define LCD_RS P2 ^ 0 // 推挽输出
#define LCD_RW P2 ^ 1 // 推挽输出
#define LCD_EN P2 ^ 2 // 推挽输出
```

```c
// lcd.c
#define DT LCD_DATA
sbit RS = LCD_RS;
sbit RW = LCD_RW;
sbit EN = LCD_EN;
```

```c
static void write(bit rs, u8 byte)
{
    EN = 0;
    RS = rs;
    RW = 0;
    DT = byte;
    EN = 1;
    EN = 0;
    delay_5us(15); // 至少72us
}
```

```c
void lcd_cmd(u8 cmd)
{
    write(0, cmd);
    if (cmd == 0x01)
    {
        delay_5us(200);
        delay_5us(105); // clear need 1.6ms
    }
}

void lcd_show(u8 dat)
{
    write(1, dat);
}
```

```c
#include <stdarg.h>
extern int vsprintf(char*, const char*, char*);
void lcd_printf(const char* format, ...)
{
    u8 s_buf[65];
    u8* s;
    va_list arg; // 定义可变参数列表数据类型的变量arg

    va_start(arg, format); // 从format开始，接收参数列表到arg变量
    vsprintf(s_buf, format, arg); // 打印格式化字符串和参数列表到字符数组中
    va_end(arg);                  // 结束变量arg

    s = s_buf;
    while (*s)
        lcd_show(*s++);
}
```

应用

```c
void lcd_init(void)
{
    lcd_cmd(0x01);
    lcd_cmd(0x0e); // 0c: 无光标; 0e: 有光标
    lcd_cmd(0x30);
}
```

```c
// 从上往下
void lcd_action(void)
{
    u8 code s1[17] = "      ABCD      ";
    u8 code s2[17] = "       07       ";
    u8 i;

    // 字从上往下, 屏幕从下往上 0x7f -> 0x70 -> 0x60
    lcd_cmd(0x01);

    lcd_cmd(0x80); // 第一行 -> 第二行 -> 屏幕外
    lcd_printf(s1);
    lcd_cmd(0xa8); // 屏幕外 -> 屏幕外 -> 第三行
    lcd_printf(s1);

    lcd_cmd(0x90); // 第二行 -> 屏幕外 -> 屏幕外
    lcd_printf(s2);
    lcd_cmd(0xb8); // 屏幕外 -> 屏三外 -> 第四行
    lcd_printf(s2);

    lcd_cmd(0x34);
    lcd_cmd(0x03);

    i = 0x7f;
    for_c(16)
    {
        delay_1ms(1000 / 16);
        lcd_cmd(i--);
    }

    delay_1ms(1000);

    for_c(16)
    {
        delay_1ms(1000 / 16);
        lcd_cmd(i--);
    }

    lcd_cmd(0x30);
    lcd_cmd(0x01);
}
```

```c
// 从下往上
void lcd_action(void)
{
    u8 code s1[17] = "      ABCD      ";
    u8 code s2[17] = "       07       ";
    u8 i;

    // 字从下往上, 屏幕从上往下 0x41 -> 0x50 -> 0x60
    lcd_cmd(0x01);

    lcd_cmd(0x88); // 第三行 -> 屏幕外 -> 屏幕外
    lcd_printf(s1);
    lcd_cmd(0xa0); // 屏幕外 -> 第二行 -> 第一行
    lcd_printf(s1);

    lcd_cmd(0x98); // 第四行 -> 第三行 -> 屏幕外
    lcd_printf(s2);
    lcd_cmd(0xb0); // 屏幕外 -> 屏幕外 -> 第二行
    lcd_printf(s2);

    lcd_cmd(0x34);
    lcd_cmd(0x03);

    i = 0x41;
    for_c(16)
    {
        delay_1ms(1000 / 16);
        lcd_cmd(i++);
    }

    delay_1ms(1000);

    for_c(16)
    {
        delay_1ms(1000 / 16);
        lcd_cmd(i++);
    }

    lcd_cmd(0x30);
    lcd_cmd(0x01);
}
```

```c
// 从左往右
void lcd_action(void)
{
    u8 code s1[17] = "             ABCD";
    u8 code s2[17] = "              07 ";
    lcd_cmd(0x01);

    lcd_cmd(0x90);
    lcd_printf(s1);
    lcd_cmd(0x88);
    lcd_printf(s2);

    for_c(6)
    {
        delay_1ms(500);
        lcd_cmd(0x18);
    }

    lcd_cmd(0x01);
}
```

```c
// 从右往左
void lcd_action(void)
{
    u8 code s1[17] = "ABCD             ";
    u8 code s2[17] = " 07              ";
    lcd_cmd(0x01);

    lcd_cmd(0x90);
    lcd_printf(s1);
    lcd_cmd(0x88);
    lcd_printf(s2);

    for_c(6)
    {
        delay_1ms(500);
        lcd_cmd(0x1c);
    }

    lcd_cmd(0x01);
}
```

I2C

```
// __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_check_ack(void);

void i2c_transmit(u8 dat);

u8 i2c_receive(void);

#endif // I2C_H
```

```
// 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;
    delay_5us(1); // 大于 4.7us
    SDA = 0;
    delay_5us(1); // 大于 4us
    SCL = 0;
}
```

```c
void i2c_stop()
{
    SCL = 0;
    SDA = 0;
    SCL = 1;
    delay_5us(1); // 大于 4us
    SDA = 1;
    delay_5us(1); // 大于 4.7us
}
```

```c
void i2c_ack(bit a)
{
    SDA = !a;
    SCL = 1;
    delay_5us(1); // 大于 4us
    SCL = 0;
    SDA = a;
}
```

```c
bit i2c_check_ack(void)
{
    bit na;
    SDA = 1;
    delay_5us(1); // 大于 4us
    SCL = 1;
    delay_5us(1); // 大于 4us
    na = SDA;
    SCL = 0;
    return !na;
}
```

```c
void i2c_transmit(u8 byte)
{
    SCL = 0;
    for_c(8)
    {
        SDA = (bit)(byte & 0x80);
        SCL = 1;
        delay_5us(1);
        SCL = 0;
        byte <<= 1;
        delay_5us(1);
    }
    SDA = 1;
    delay_5us(1);
}
```

```c
u8 i2c_receive(void)
{
    u8 dat;
    SDA = 1;
    for_c(8)
    {
        dat <<= 1;
        SCL = 0;
        delay_5us(1);
        SCL = 1;
        delay_5us(1);
        dat |= SDA;
    }
    SCL = 0;
    delay_5us(1);
    return dat;
}
```

AT24C

```c
// at24c.c
#include "__config__.h"

#include "i2c.h" // 需要依赖 I2C
```

```c
static bit transmit_byte(u8 byte)
{
    i2c_transmit(byte);
    return i2c_check_ack();
}
```

```c
static bit check(u8 device)
{ // device: 用于在多个at24c中选择
    i2c_start();
    if (transmit_byte(0xa0 | (device & 0x0f)))
        return 1;
    i2c_stop();
    return 0;
}
```

```c
static bit inquiry(u16 addr)
{ // at24c512: 地址访问需要高低两字节
    return (check(0) && transmit_byte(addr >> 8) && transmit_byte(addr & 0xff));
}
```

```c
void at24c_read(u16 addr, u8* dat, u8 len)
{
    if (len == 0)
        return;
    if (inquiry(addr))
    {
        if (check(1)) // 重新启动I2C总线
        {
            for_c(len - 1)
            {
                *dat++ = i2c_receive();
                i2c_ack(1);
            }
            *dat = i2c_receive();
            i2c_ack(0);
        }
        i2c_stop();
    }
}
```

```c
// 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
void at24c_write(u16 addr, u8* dat, u8 len)
{ // at24c512, 一页应该是 128 byte
    if (len == 0)
        return;
    if (inquiry(addr))
    {
        for_c(len)
        {
            if (!transmit_byte(*dat++))
                break;
        }
        i2c_stop();
        delay_1ms(10); // 至少 5ms
    }
}
```

DS1302

```
// __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 使能(复位)

#define WAIT _nop_, _nop_

#define CONTROL     0x8e // 控制 WP: 0x00: 写允许, 0x80: 写禁止
#define MULTI_CLOCK 0xbe
#define RD 1
#define WT 0
```

```
static void start(void)
{
    CE = 0;
    WAIT; // at least: 2v: 200ns; 5v: 50ns
    SCK = 0;
    WAIT;
    CE = 1; // 启动
    WAIT;
}
```

```
static void write_byte(u8 byte)
{
    for_c(8)
    {
        SCK = 0;
        SDA = byte & 0x01;
        byte >>= 1;
        SCK = 1; // 上升沿写入
    }
}
```

```c
static u8 read_byte(void)
{
    u8 byte = 0;
    for_c(8)
    {
        SCK = 0; // 下降沿读取
        byte >>= 1;
        if (SDA)
            byte |= 0x80; // 每次传输低字节
        SCK = 1;
    }
    return byte;
}
```

```c
// 如果不对SDA初始化，第一次读取多字节会出现混乱
void ds1302_init(void)
{
    start();
    CE = 0;
    WAIT;
    SDA = 0; // 如果不对SDA初始化，第一次读取多字节会出现混乱
}
```

```c
// 7个字节: 秒分时日月周年
// eg. {40, 9, 14, 18, 5, 6, 24}; 40秒 9分 14时 18日 5月 周六 24年
void ds1302_get(u8* get)
{
    u8 i;
    start();
    write_byte(MULTI_CLOCK | RD);
    for (i = 0; i < 7; ++i)
    { // 不要用 for_c, 因为 read_byte 用了
        get[i] = read_byte();
    }
    read_byte(); // 读取写使能

    for (i = 0; i < 7; ++i)
    {
        get[i] = (get[i] / 16 * 10) + (get[i] & 0x0f);
    }

    CE = 0;
    WAIT;
    SDA = 0;
}
```

```c
// 7个字节: 秒分时日月周年
void ds1302_set(u8* set)
{
    u8 i;
    start();
    write_byte(CONTROL | WT);
    write_byte(0x00); // 写使能
    CE = 0;

    start();
    write_byte(MULTI_CLOCK | WT);
    for (i = 0; i < 7; ++i)
    { // 不要用 for_c, 因为 write_byte 用了
        write_byte((set[i] / 10 * 16) + (set[i] % 10));
    }
    write_byte(0x80); // 写禁止
    CE = 0;
}
```

日期相关函数

```c
// 返回每月天数, 包含计算闰年
u8 month_days(u16 year, u8 month)
{
    u8 code days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (month == 0 || month > 12)
        return 0;
    if (month == 2 && ((year & 3) == 0))
        return 29;
    return days[month - 1];
}
```

```c
// 2001年1月1日 星球一 1-7[周一至周日]
u8 what_day(u16 year, u8 month, u8 day)
{
    u8 curr  = 1;
    u16 diff = year - 2001;
    diff += (diff >> 2) + (diff > 0x7fff); // %7 : 1月1日 -1

    while (curr != month)
    {
        diff += MonthDays(year, curr);
        curr++;
    } // month月1日 -1

    diff += day - 1;
    return (diff % 7) + 1;
}
```

ADC

```
// __config__.h
#define LDR 0 // P10 智慧农业 光敏电阻
#define POT 0 // P10 智能小车 电位器
```

```
// 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
#define ADC_START 0x08 // 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

// 转换速度控制位SPEED0和SPEED1，共四种状态，对应四种转换速度
// 0x00: 540 clocks; 0x20: 360 clocks; 0x40: 180 clocks; 0x90: 90 clocks;
#define ADC_SPEED 0x00
```

```
/**
 * @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 | ADC_START | ch; // 0x80 | 0x00 | 0x08 | ch
    // 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 默认 0
     * 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; // (Vo * 3.3 / 1024) (求出电压值)
}
```

PWM

```
// __config__.h
// 智慧农业 pwm led      P17 推挽输出
// 智能小车 pwm dc motor P16 推挽输出 最好大于50%
#define PWM_ID 7 // PWM_7: P17
#define PWM_ID 6 // PWM_6: P16
```

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

#define EAXSFR BIT_H(P_SW2, 7)
#define EAXRAM BIT_L(P_SW2, 7)

#define PWM_X(X, _) PWM##X##_
#define PWM_CR(X)   PWM_X(X, CR)
#define PWM_T1(X)   PWM_X(X, T1)
#define PWM_T2(X)   PWM_X(X, T2)
```

```
void pwm_init(void)
{
    EAXSFR; // 访问XFR
    PWM_CR(PWM_ID) = 0; // PWM7CR = 0;
    PWMCR |= (1 << (PWM_ID - 2));   // 相应的端口为PWM输出口，受PWM波形发生器控制
    PWMCFG &= ~(1 << (PWM_ID - 2)); // 0010 0000 PWM 初始低电平，不自动触发ADC
    EAXRAM; // 恢复访问XRAM
    // PWMCR &= ~0x40; // 禁止PWM计数器归零中断
    // PWMCR |= 0x80;  // 使能PWM波形发生器，PWM计数器开始计数
}
```

```
void pwm_clk(u8 ps)
{ // 设置时钟源 0-15: sys_clk / (ps + 1); 16: t2
    EAXSFR; // 访问XFR
    PWMCKS = ps & 0x1f; // PWM时钟选择
    EAXRAM; // 恢复访问XRAM
}
```

```
void pwm_set(u16 first, u16 second, u16 cycles)
{ // 设置 pwm 波(周期和两次翻转)  1~32767(0x7fff)
    EAXSFR; // 访问XFR
    PWM_T1(PWM_ID) = first;  // PWM7T1 = first;
    PWM_T2(PWM_ID) = second; // PWM7T2 = second;
    PWMC           = cycles;
    EAXRAM; // 恢复访问XRAM
}
```

```c
// 中断
void PWM_int(void) interrupt 22
{
    if (PWMIF & 0x40)
    {                   // PWM计数器归零中断标志
        PWMIF &= ~0x40; // 清除中断标志
    }
    if (PWMIF & 0x01)
    {                   // PWM2中断标志
        PWMIF &= ~0x01; // 清除中断标志
    }
    if (PWMIF & 0x02)
    {                   // PWM3中断标志
        PWMIF &= ~0x02; // 清除中断标志
    }
    if (PWMIF & 0x04)
    {                   // PWM4中断标志
        PWMIF &= ~0x04; // 清除中断标志
    }
    if (PWMIF & 0x08)
    {                   // PWM5中断标志
        PWMIF &= ~0x08; // 清除中断标志
    }
    if (PWMIF & 0x10)
    {                   // PWM6中断标志
        PWMIF &= ~0x10; // 清除中断标志
    }
    if (PWMIF & 0x20)
    {                   // PWM7中断标志
        PWMIF &= ~0x20; // 清除中断标志
    }
}

void PWMFD_int(void) interrupt 23
{
    if (PWMFDCR & 0x01)
    {                     // PWM异常检测中断标志位
        PWMFDCR &= ~0x01; // 清除中断标志
    }
}
```

应用

```c
pwm_clk(0x0b);          // clk: 2MHz
pwm_set(0, 10, 1000);   // led: 2000Hz
PWMCR |= 0x80;          // 使能PWM波形发生器, PWM计数器开始计数
PWMCR &= ~0x40;         // 禁止PWM计数器归零中断
```

DHT11

```c
// __config__.h
#define DHT11_DQ P1 ^ 1 // 和步进电机冲突，设置为双向IO口
```

```c
// dht11.c
#include "__config__.h"

sbit DQ = DHT11_DQ;
```

```c
static bit check(void)
{ // @return: 1-存在; 0-不存在;
    bit x;
    DQ = 1; // 复位DQ
    DQ = 0;
    delay_1ms(18); // 拉低大于 18ms
    DQ = 1;
    delay_5us(8); // 20~40us, 80us之内
    x = !DQ;      // 接收响应低电平
    delay_5us(8); // DHT11总共拉低 80us
    return x;     // 后拉高 80us
}
```

```c
static bit wait_DQ(bit v, u8 t)
{ // @return: 期待 DQ 的值是 v, 如果超时未响应 则返回 0, 否则 DQ == v 返回 1
    while (DQ != v && --t)
        delay_5us(1);
    return t != 0;
}
```

```c
static bit read_byte(u8* byte)
{ // 拉低 50us, 拉高 0: 26~28us, 1: 70us; 拉低(next bit)
    u8 tmp;
    if (!wait_DQ(0, 40))
        return 0; // 等待进入低电平, 超过 200us 退出

    tmp = 0;
    for_c(8)
    {
        if (!wait_DQ(1, 20))
            return 0; // 等待50us低电平过去, 超过 100us 退出
        delay_5us(9); // 延时45us, 如果还为高则数据为1, 否则为0
        tmp <<= 1;
        if (DQ) // 数据为1时, 使dat加1来接收数据1
        {
            tmp |= 1;
            if (!wait_DQ(0, 20))
                return 0; // 等待电平拉低, 超过 100us 退出
        }
    }

    if (byte)
        *byte = tmp;
    return 1;
}
```

```c
// DHT11通电后需要先放置 2s 等待内部启动, 立即读取不会应答
bit dht11_read(float* r, float* t)
{
    u8 chkSum, r_h, t_h, r_l, t_l;

    if (!check()) return 0;

    delay_5us(16); // 拉高80us后, 开始接收数据

    if (!read_byte(&r_h))    return 0; // 接收湿度高八位
    if (!read_byte(&r_l))    return 0; // 接收湿度低八位
    if (!read_byte(&t_h))    return 0; // 接收温度高八位
    if (!read_byte(&t_l))    return 0; // 接收温度低八位
    if (!read_byte(&chkSum)) return 0; // 接收校正和

    DQ = 1; // 结束
    if ((r_h + t_h + r_l + t_l) == chkSum) // 最后一字节为校验位, 校验是否正确
    {
        if (r) *r = r_h + r_l * 0.1;
        if (t) *t = t_h + t_l * 0.1;
    }
    return 1;
}
```

STEP_MOTOR

```
// __config__.h
#define STEP_MOTOR P1, 1
```

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

#define _SET(P, B, V) (P &= ~(0x0f << (B)), P |= (V) << (B))
#define SET(P, V)     _SET(P, V)
```

```
// size = 512 转一圈
void step_motor_run(bit s, u16 size)
{ // 0: 顺时针, 1: 逆时针
    static u8 code spm_turn[] = {
        0x02, 0x06, 0x04, 0x0c, 0x08, 0x09, 0x01, 0x03,
    };
    s8 i;

    for_c(size)
    {
        if (s)
        {
            for (i = 0; i < 8; ++i)
            {
                SET(STEP_MOTOR, spm_turn[i]);
                delay_5us(255);
            }
        }
        else
        {
            for (i = 7; i >= 0; --i)
            {
                SET(STEP_MOTOR, spm_turn[i]);
                delay_5us(255);
            }
        }
    }
}
```

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
    delay_5us(2); // 拉高 10us
    TRIG = 0;
#if 0
    while (!ECHO)
        ;
    TR0 = 1;
    while (ECHO)
        ;
    TR0 = 0;
    return ((((u16)TH0 << 8) | TL0) >> 1) * 0.017; // cm
#else
    for_c(65535)
    { // 防止没有超声波传感器卡死
        if (ECHO)
        { // 等待开始探测
            TR0 = 1;
            while (ECHO && TF0 == 0)
                ; // 等待结束探测
            if (TF0 == 1)
                return TF0 = TR0 = 0;
            TF0 = TR0 = 0;
            return ((((u16)TH0 << 8) | TL0) >> 1) * 0.017; // cm
        }
    }
    return 0;
#endif
}
```

DS18B20

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

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

sbit DQ = DS18B20_DQ;
```

```c
static void reset(void)
{
    DQ = 1; // 复位DQ
    DQ = 0;
    delay_5us(120); // 拉低 480~960us
    DQ = 1;
}
```

```c
/**
 * @return: 1-存在; 0-不存在;
*/
static bit check(void)
{
    bit x;
    reset();
    delay_5us(24); // 等待 15~60us 240us之内
    x  = !DQ;      // 总线60~240us低电平
    DQ = 1;        // 释放总线
    delay_5us(48); // 保证时序完整 480us * 2
    return x;
}
```

```c
static u8 read_byte(void)
{
    u8 byte = 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
        delay_5us(1);
        DQ = 1; // 15μs内拉高释放总线
        byte >>= 1;
        if (DQ)
            byte |= 0x80;
        delay_5us(9); // 每个读时段 最少60us
    }
    return byte;
}
```

```c
static void write_byte(u8 byte)
{
    for_c(8)
    {                   // 串行写8位数据，先写低位后写高位
        DQ = 0;         // 拉低
        delay_5us(1);   // 至少间隔1us 低于15us
        DQ = byte & 0x01; // 写 '1' 在15μs内拉高
        delay_5us(9);   // 写 '0' 拉低60μs 10+50
        DQ = 1;
        byte >>= 1;
    }
}
```

```c
#define SKIP_ROM     0xcc // 跳过匹配 ROM

#define CONVERT_T         0x44 // 开始温度转换
#define WRITE_SCRATCHPAD  0x4e // 写暂存器（温度上下限和精度）
#define READ_SCRATCHPAD   0xbe // 读暂存器（温度值）（温度上下限和精度）
#define COPY_SCRATCHPAD   0x48 // 存入 EEPROM（温度上下限和精度）
```

```c
void ds18b20_convert() // 温度转换
{
    if (!check())
        return;
    write_byte(SKIP_ROM); // 0xcc
    write_byte(CONVERT_T);
}
```

```c
float ds18b20_read_temp() // 温度读取
{
    s16 n;
    if (!check())
        return 0xffff;

    write_byte(SKIP_ROM); // 0xcc
    write_byte(READ_SCRATCHPAD); // 0xbe
    n = read_byte();
    n |= ((s16)read_byte()) << 8;
    reset(); // 结束复位
    return n * 0.0625;
}
```

```c
/**
 * @param upper_limit: 温度上限
 * @param lower_limit: 温度下限
 * @param accuracy: 精度(分辨率)
*/
void ds18b20_get(u8* upper_limit, u8* lower_limit, u8* accuracy)
{
    u8 tmp;
    if (!check())
        return;

    write_byte(SKIP_ROM); // 0xcc
    write_byte(READ_SCRATCHPAD); // 0xbe
    read_byte();
    read_byte();

    tmp = read_byte();
    if (upper_limit)
        *upper_limit = tmp;

    tmp = read_byte();
    if (lower_limit)
        *lower_limit = tmp;

    tmp = read_byte();
    if (accuracy)
        *accuracy = tmp >> 5;

    reset(); // 结束复位
}
```

```c
/**
 * @brief: 只是设置到 RAM, 存储需要通过 save, 写入 EEPROM
 * @param upperLimit: 温度上限
 * @param lowerLimit: 温度下限
 * @param resolution: 分辨率(精度)
 *                      00:  9位, 93.75 ms
 *                      01: 10位, 187.5 ms
 *                      10: 11位,   375 ms
 *                      11: 12位,   750 ms
*/
void ds18b20_set(u8 upperLimit, u8 lowerLimit, u8 resolution)
{
    if (!check())
        return;
    write_byte(SKIP_ROM); // 0xcc
    write_byte(WRITE_SCRATCHPAD); // 04e
    write_byte(upperLimit);       // 写高速缓存器TH高温限值
    write_byte(lowerLimit);       // 写高速缓存器TL低温限值
    write_byte(((resolution & 0x03) << 5) | 0x1f); // 精度设置
}
```

```c
void ds18b20_save(void)
{
    if (check())
        return;
    write_byte(SKIP_ROM); // 0xcc
    write_byte(COPY_SCRATCHPAD); // 0x48
    delay_5us(24); // 需要给DS18B20存储的时间
    // 经测试 最好高于80us 否则可能会造成存入失败或者数据错误
}
```