蓝桥杯单片机模版建立

目录 Contents

- iic.c
  - 官方的参考代码
  - 我们需要补充的代码
    - AD与DA
    - EEPROM
- iic.h

Uart串口模版建立

- uart.c
- uart.h

Ul超声波模版建立

- ul.c
- ul.h

init初始化模版建立

- init.c
- init.h

main主函数书写

- main.c
  - 全部变量的归纳
  - 数据读取模块
  - Key键盘模块
  - Seg数码管模块
  - Led灯模块
  - Uart串口模块
  - 定时器0初始化（NE555模块）
  - 定时器1初始化
  - 定时器1中断（重点）
  - 串口中断（这里代码写死）
  - main主函数书写
  - 完整的main.c
- main.h

前言

小小的bb几句

这里我说明一下我个人在学习单片机的时候的一些心得体会，以及大模板的最新建立，会持续更新。

最基础的模版来自于B站Up Alice西风的个人空间-Alice西风个人主页-哔哩哔哩视频(bilibili.com)

我的B站为左-岚的个人空间-左-岚个人主页-哔哩哔哩视频（bilibili.com）

这篇文章对应着我B站的视频为 【蓝桥杯】【单片机】大模板构建喂饭教程

建议搭配着视频使用，不然可能有些地方比较迷糊。

用到的压缩包和这个pdf的百度网盘链接

```
链接：https://pan.baidu.com/s/1LhlBR7eKAB4A4sbvkHtiJw?pwd=mvin
提取码：mvin
--来自百度网盘超级会员V4的分享
```

对应的CSDN链接为【喂饭】【速成】蓝桥杯模版手把手建立-CSDN博客

对应的知乎链接为【喂饭】【速成】蓝桥杯模版手把手建立

这两个仓库可以删除后面的东西（一直删除到master，可以看到我的全部的东西）

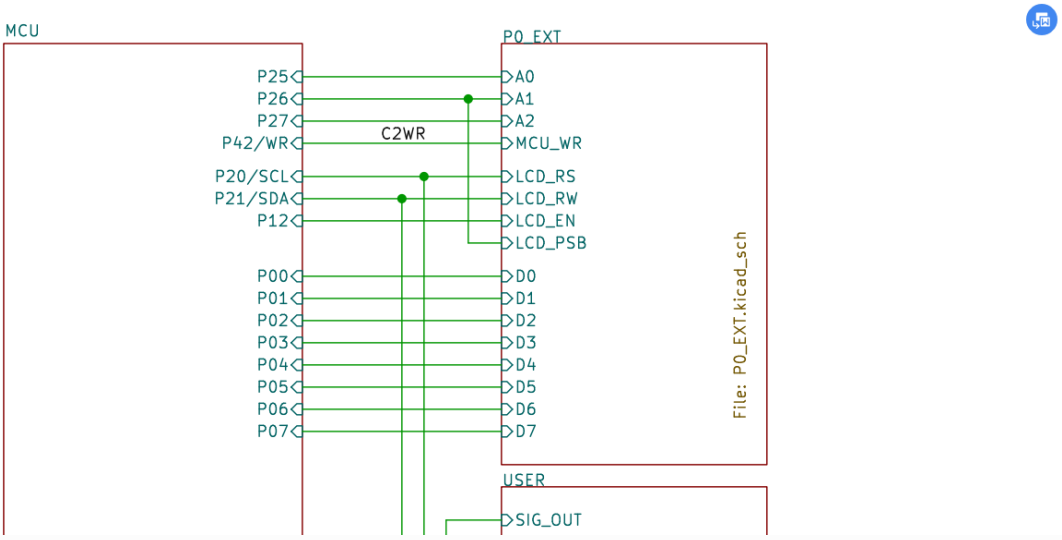
Github的仓库为 lanqiaobei_study/模板/demo_zuolan at master · zuoliangyu/lanqiaobei_study (github.com)
Gitee的仓库为 模板/demo_zuolan · zuoliangyu/蓝桥杯学习 - 码云 - 开源中国(gitee.com)

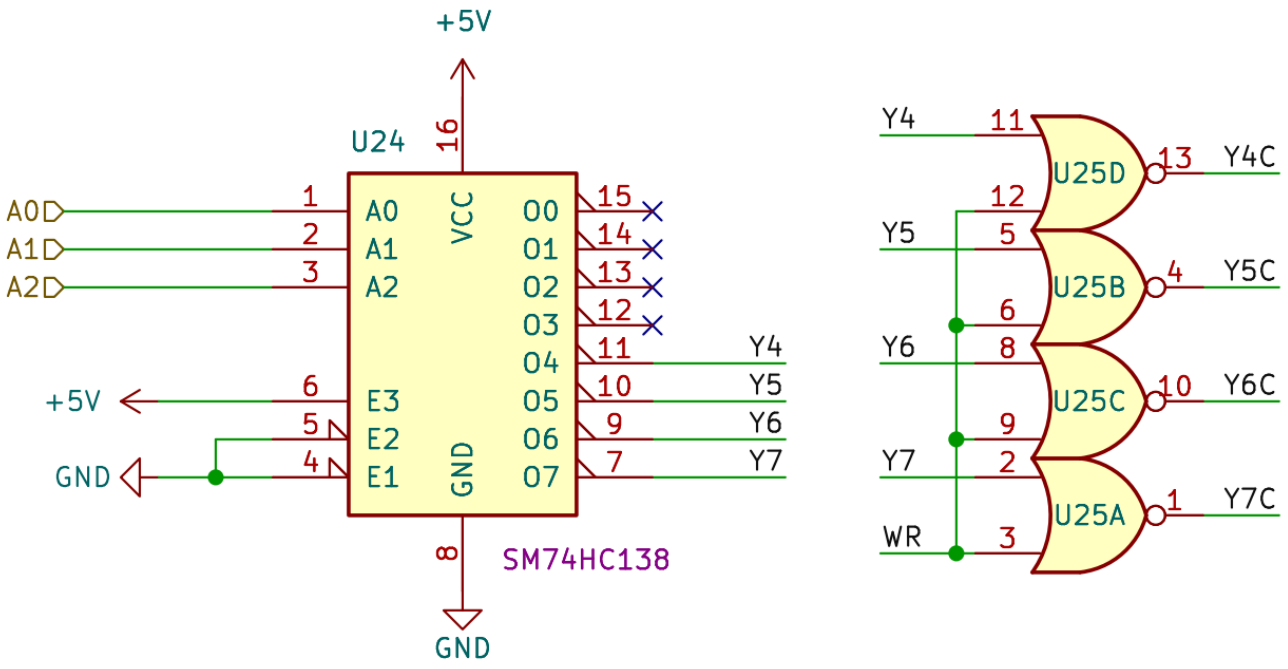
锁存器与三八译码器介绍

这里我们需要有一个基础知识，就是我们的锁存器和三八译码器介绍，这是一个比较关键的点，我们看到下面这两个图，可以看到我们的锁存器是用P25，P26，P27进行控制，即P2的高三位进行的控制，我们只需要按照如下的命令进行书写即可。

```
P2 = P2 & 0x1f | 0x?0;
P2 &= 0x1f;
```

其中的P2&=0x1f，表示我们将高三位置为0，| 0x?0的操作表明我们对高三位进行书写，比如Y4C，那么就代表我们的高三位要表示为4，即0100，我们将其放在高位，即**100**0_0000，翻译过来就是0x80





NOP延时函数

这里要注意一个问题，就是我们延时函数大部分都有NOP，包括Ds1302，IIC都使用了这个，这个函数存在于头文件intrins.h中，需要注意一下引用，一般来说，一个NOP在12MHz的单片机下是1us，其余的自己去换算一下。

LED，蜂鸣器，继电器，MOTOR模版建立

Led.c

这里虽然名称叫做Led.c，但是里面书写LED，蜂鸣器，继电器，MOTOR的代码

Led底层

我们可以看到Led的代码，我们可以注意到的是，这个地方使用了static这个关键词，因为我们需要关注内存的使用~~（这里不得不提C51的垃圾内存容量）~~，所以我们将其变为函数静态内存，可以有效节省。

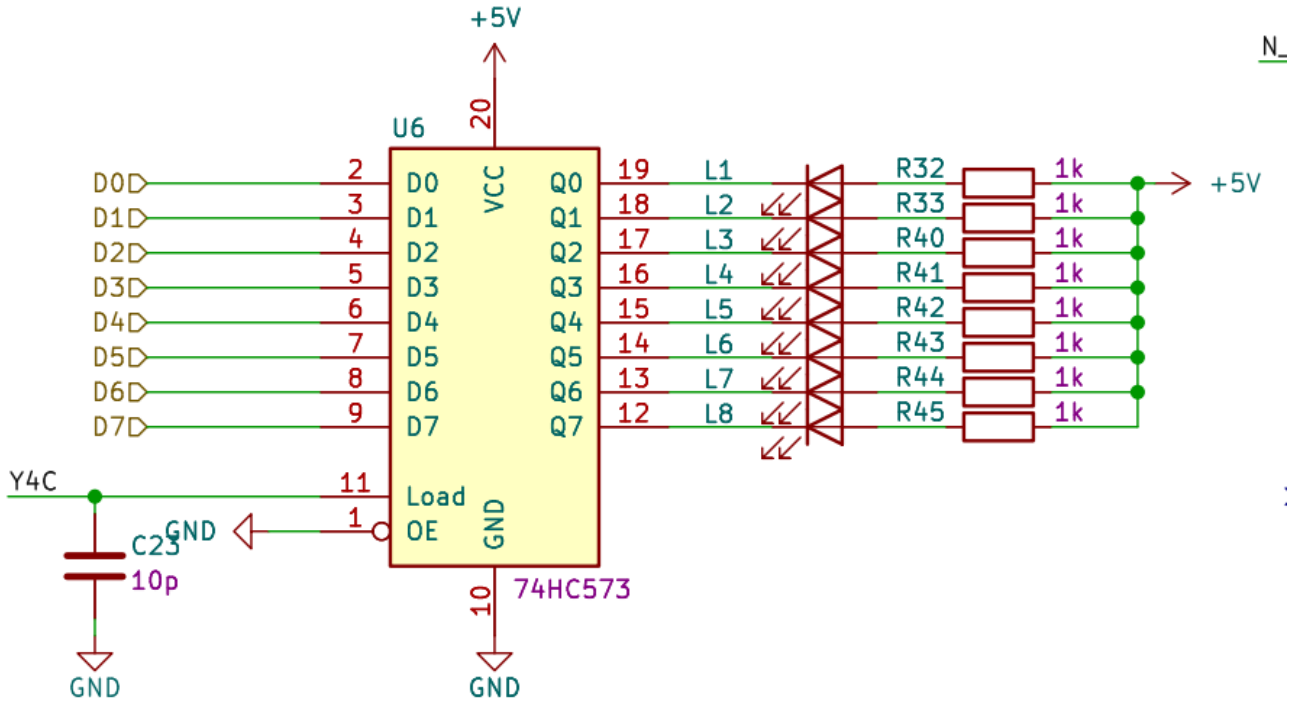
传入的参数有两个，一个是addr，代表我们需要控制的Led的地址，（0-7）对应了（L1-L8），另一个是enable，用于控制选中的Led灯的亮灭。

```
void Led_Disp(unsigned char addr, unsigned char enable)
```

我们定义了两个参数，一个是temp，一个是temp_old，你们可能会好奇，为什么我要在这里用两个变量来进行编写，明明逻辑上只需要用一个变量对P0进行赋值就行了，这里我说明一下，我们一般情况下不会一直操控锁存器，否则有概率影响数码管的显示，当我们状况出现变化（temp≠temp_old）的时候，我们才对锁存器P2的高三位进行书写。当我们需要点亮的时候，就让相应的addr变为1；当我们需要熄灭的时候就让相应的addr变为0；

> 这里你可能会有疑问，因为我们在原理图上面如果想让Led亮，那么应该给0，这里做一个说明，我们后面会直接取反，这里用1是为了符合我们的直觉~~（因为正常人就是1亮0灭）~~

```
static unsigned char temp = 0x00;
static unsigned char temp_old = 0xff;
if (enable)
    temp |= 0x01 << addr;
else
    temp &= ~(0x01 << addr);
```



我们通过查看原理图可以看到，用于控制的Load为Y4C，我们通过计算二进制可以发现结果为1000_0000，即0x80，所以我们在这里代码中的P2设置为P2 = P2 & 0x1f | 0x80;在运行后别忘记使用P2 &= 0x1f来进行高位清零，但是在我们操作P2锁存器之前，我们需要先对P0进行赋值（要记得先取反）

```
if (temp != temp_old) {
    P0 = ~temp;
    P2 = P2 & 0x1f | 0x80;
    P2 &= 0x1f;
    temp_old = temp;
}
```

完整代码如下

```
    /// @brief Led扫描
    /// @param addr 需要控制的Led的地址 (0-7)
    /// @param enable 控制该地址的Led是否点亮
    void Led_Disp(unsigned char addr, unsigned char enable) {
      static unsigned char temp = 0x00;
      static unsigned char temp_old = 0xff;
      if (enable)
        temp |= 0x01 << addr;
      else
        temp &= ~(0x01 << addr);
      if (temp != temp_old) {
        P0 = ~temp;
```

```
            P2 = P2 & 0x1f | 0x80;
            P2 &= 0x1f;
            temp_old = temp;
        }
    }
```

蜂鸣器，继电器，MOTOR底层

这里我就一起讲了，因为他们三是一个芯片的不同输出，和上方的Led类似，但是这几个需要一个全局变量（在函数外申明）

```
unsigned char temp_0 = 0x00;
unsigned char temp_old_0 = 0xff;
```

我们可以看到，BUZZ、MOTOR、RELAY为一个芯片的不同位的输出

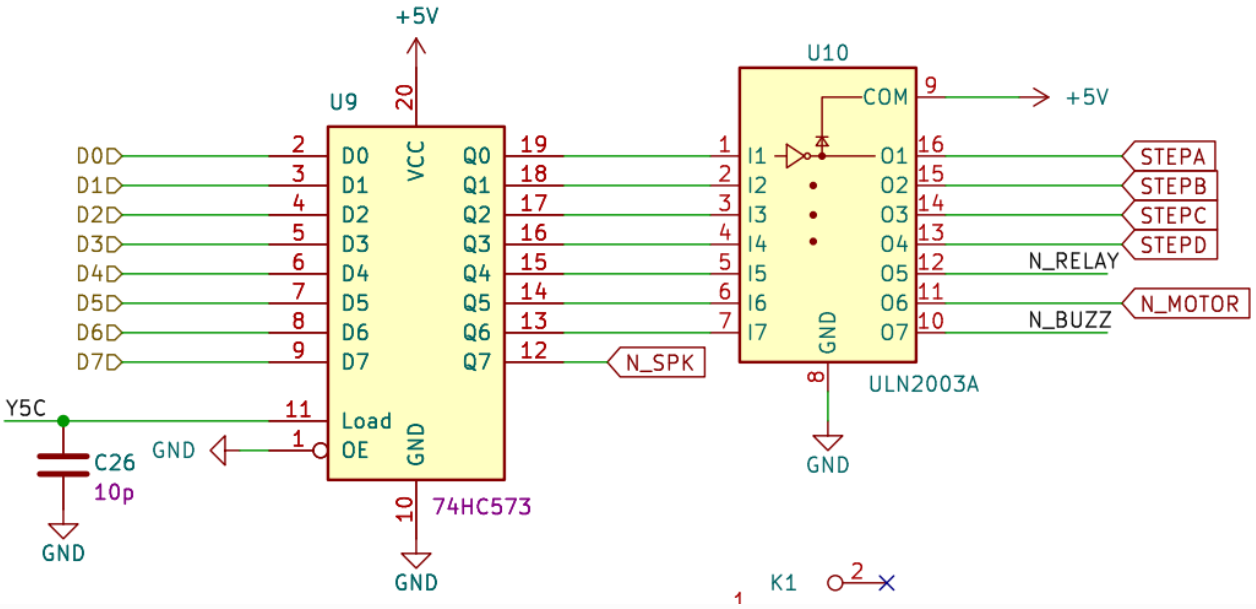
RELAY对应I5→Q4→D4→0001_0000→0x10

MOTOR对应I6→Q5→D5→0010_0000→0x20

BUZZ对应I7→Q6→D6→0100_0000→0x40

这里后我们就知道如果我们需要对应的工作，那么就要给P0赋相应的值，并且下方的Y5C也告诉我们，我们需要给锁存器为1010_0000→0xa0,即P2 = P2 & 0x1f | 0xa0;这个之后也别忘了使用P2 &= 0x1f来进行高位清零。

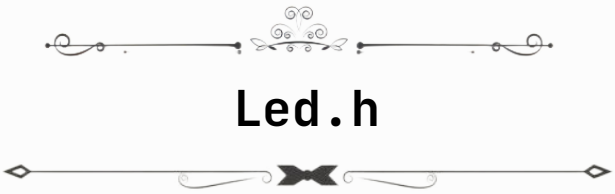
```
    if (temp_0 != temp_old_0) {
        P0 = temp_0;
        P2 = P2 & 0x1f | 0xa0;
        P2 &= 0x1f;
        temp_old_0 = temp_0;
    }
```



总体代码如下

```
unsigned char temp_0 = 0x00;
unsigned char temp_old_0 = 0xff;
/// @brief 蜂鸣器控制
/// @param enable
void Beep(bit enable) {
  if (enable)
    temp_0 |= 0x40;
  else
    temp_0 &= ~(0x40);
  if (temp_0 != temp_old_0) {
    P0 = temp_0;
    P2 = P2 & 0x1f | 0xa0;
    P2 &= 0x1f;
    temp_old_0 = temp_0;
  }
}
/// @brief 继电器控制
/// @param enable
void Relay(bit enable) {
  if (enable)
    temp_0 |= 0x10;
  else
    temp_0 &= ~(0x10);
  if (temp_0 != temp_old_0) {
    P0 = temp_0;
    P2 = P2 & 0x1f | 0xa0;
    P2 &= 0x1f;
    temp_old_0 = temp_0;
  }
```

```
}
/// @brief MOTOR控制
/// @param enable
void MOTOR(bit enable) {
  if (enable)
    temp_0 |= 0x20;
  else
    temp_0 &= ~(0x20);
  if (temp_0 != temp_old_0) {
    P0 = temp_0;
    P2 = P2 & 0x1f | 0xa0;
    P2 &= 0x1f;
    temp_old_0 = temp_0;
  }
}
```

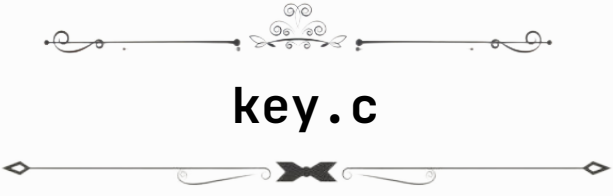


Led.h

申明代码如下

```
#include <STC15F2K60S2.H>
void Led_Disp(unsigned char addr, unsigned char enable);
void Beep(bit enable);
void Relay(bit enable);
void MOTOR(bit enable);
```

Key模版建立

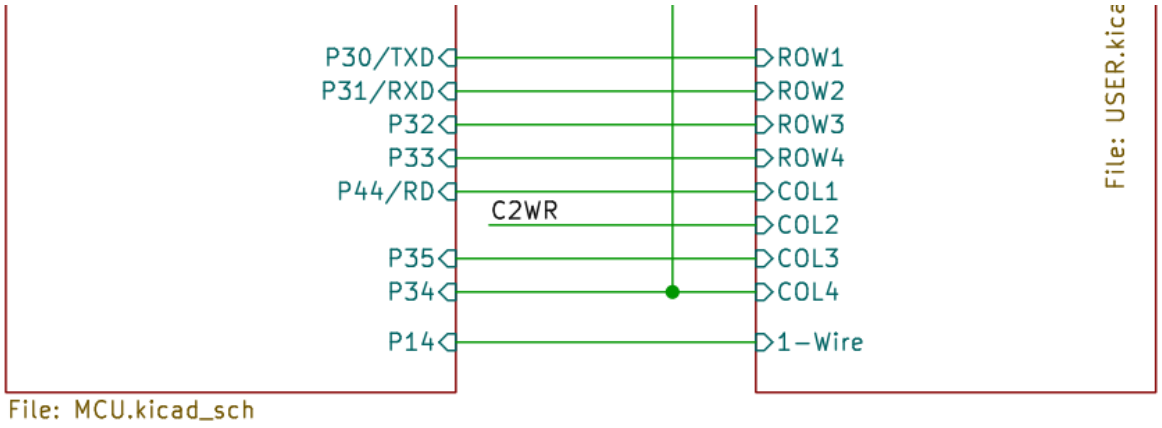


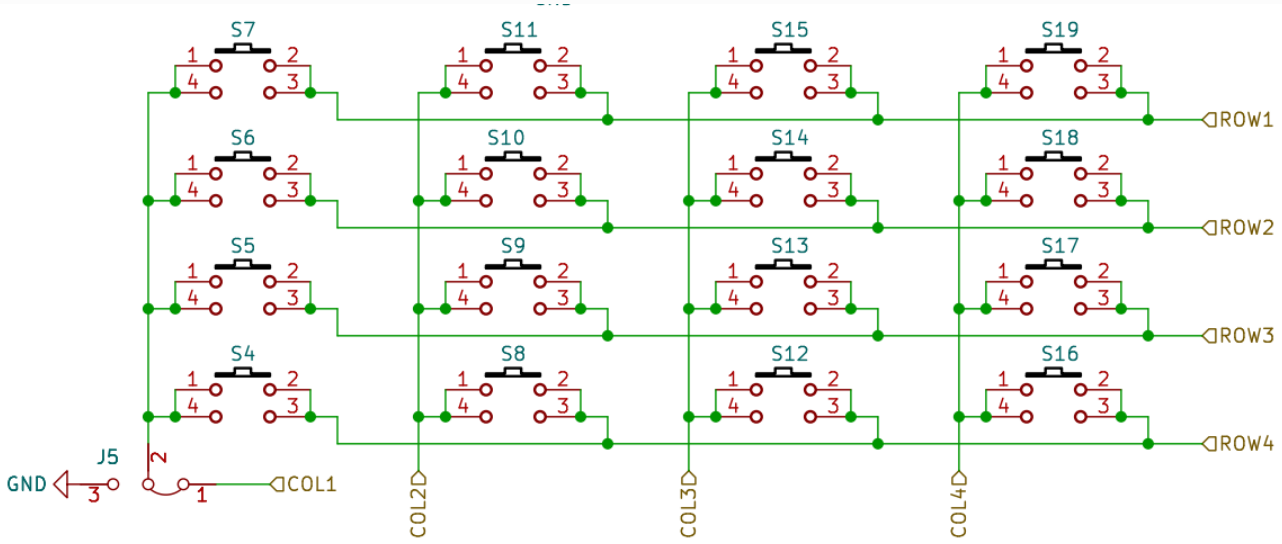
key.c

基础代码

在这里我们可以看到原理图里面的行列显示，最新版的原理图把row和col单独拿出来了，所以可能对应有点麻烦~~（其实也不麻烦）~~，如下图所示，我们直接按照一列一列进行扫描，给列低电平后判断指定行是否为低电平，如果指定行为低电平，那么我们就认为这个按键被按下。并且我们在这里不需要实现消抖等操作，我们会在main.c里面实现我们的功能，这里只实现最基础的扫描。

```
P44 = 0;P42 = 1;P35 = 1;P34 = 1;
if (P33 == 0) temp = 4;
if (P32 == 0) temp = 5;
if (P31 == 0) temp = 6;
if (P30 == 0) temp = 7;

P44 = 1;P42 = 0;P35 = 1;P34 = 1;
if (P33 == 0) temp = 8;
if (P32 == 0) temp = 9;
if (P31 == 0) temp = 10;
if (P30 == 0) temp = 11;

P44 = 1;P42 = 1;P35 = 0;P34 = 1;
if (P33 == 0) temp = 12;
if (P32 == 0) temp = 13;
if (P31 == 0) temp = 14;
if (P30 == 0) temp = 15;

P44 = 1;P42 = 1;P35 = 1;P34 = 0;
if (P33 == 0) temp = 16;
if (P32 == 0) temp = 17;
if (P31 == 0) temp = 18;
if (P30 == 0) temp = 19;
```





在我们把行列扫描写完后就可以完成工作了，但是，我们要注意一个问题，局部变量一定要注意初始化为0

```
unsigned char temp = 0;
```

注意事项

在这里要注意一个问题，我们的串口进行发送的时候会占用P30和P31口，这个是硬件的冲突，所以我们在进入这个扫描函数的时候，需要对其关掉轮询（就是在main函数里面用的什么来进行各个函数定时的定时器，叫做轮询定时器）。这里我用的是定时器1，所以我进入后就要使用ET1=0关掉，结束后用P3=0xff；ET1=1开启。

```
ET1 = 0;
----
P3 = 0xff;
ET1 = 1;
```

还有一个比较重要的问题，今年（2024）不管是省赛还是国赛都考了ne555的测量，我们ne555的测量要用到P34引脚，也就是我们的按键的最后一列，那么这里就要注意一下，这是一个硬件冲突，且无法解决，所以我们将所有P34删除，就当我们用的按键为4行3列

```
P44 = 0;P42 = 1;P35 = 1;
if (P33 == 0) temp = 4;
if (P32 == 0) temp = 5;
if (P31 == 0) temp = 6;
if (P30 == 0) temp = 7;

P44 = 1;P42 = 0;P35 = 1;
if (P33 == 0) temp = 8;
if (P32 == 0) temp = 9;
if (P31 == 0) temp = 10;
if (P30 == 0) temp = 11;

P44 = 1;P42 = 1;P35 = 0;
if (P33 == 0) temp = 12;
if (P32 == 0) temp = 13;
if (P31 == 0) temp = 14;
if (P30 == 0) temp = 15;
```

整体代码如下（没有包括ne555，如果有ne555需要把按键扫描换成上面的）

```
#include "key.h"
unsigned char Key_Read() {
  unsigned char temp = 0;
  ET1 = 0;

  P44 = 0;P42 = 1;P35 = 1;P34 = 1;
  if (P33 == 0) temp = 4;
  if (P32 == 0) temp = 5;
  if (P31 == 0) temp = 6;
  if (P30 == 0) temp = 7;

  P44 = 1;P42 = 0;P35 = 1;P34 = 1;
  if (P33 == 0) temp = 8;
```

```
    if (P32 == 0) temp = 9;
    if (P31 == 0) temp = 10;
    if (P30 == 0) temp = 11;

    P44 = 1;P42 = 1;P35 = 0;P34 = 1;
    if (P33 == 0) temp = 12;
    if (P32 == 0) temp = 13;
    if (P31 == 0) temp = 14;
    if (P30 == 0) temp = 15;

    P44 = 1;P42 = 1;P35 = 1;P34 = 0;
    if (P33 == 0) temp = 16;
    if (P32 == 0) temp = 17;
    if (P31 == 0) temp = 18;
    if (P30 == 0) temp = 19;

    P3 = 0xff;
    ET1 = 1;
    return temp;
}
```

key.h

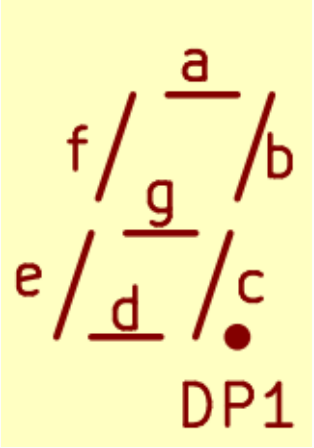
```
#include <STC15F2K60S2.H>
unsigned char Key_Read();
```

数码管模板建立

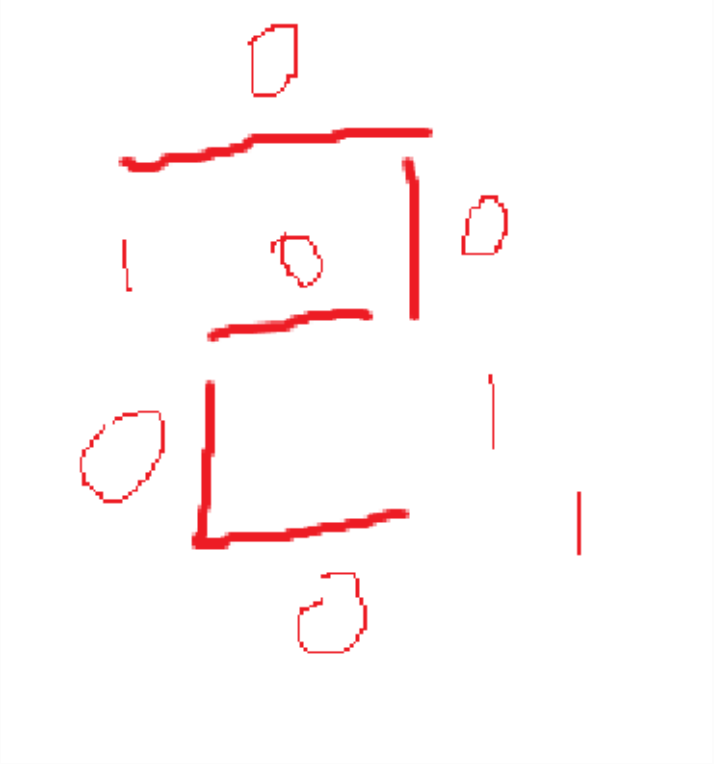
Seg.c

数码管编码推导

这里我单独开一个标题来做数码管的编码推导，因为这个很重要，我们看到下方的截图



可以观察到每一段的数码管都被编号，这里由于我们用的是共阳极数码管，即0亮1灭，下面我给一个🌰来进行说明



我们标好后按照dp→g→…→a的顺序写出，即1010_0100→0xa4，这个值在数码管的显示中就代表着“2”

所以我们就可以根据这个规律来编写一个段选表

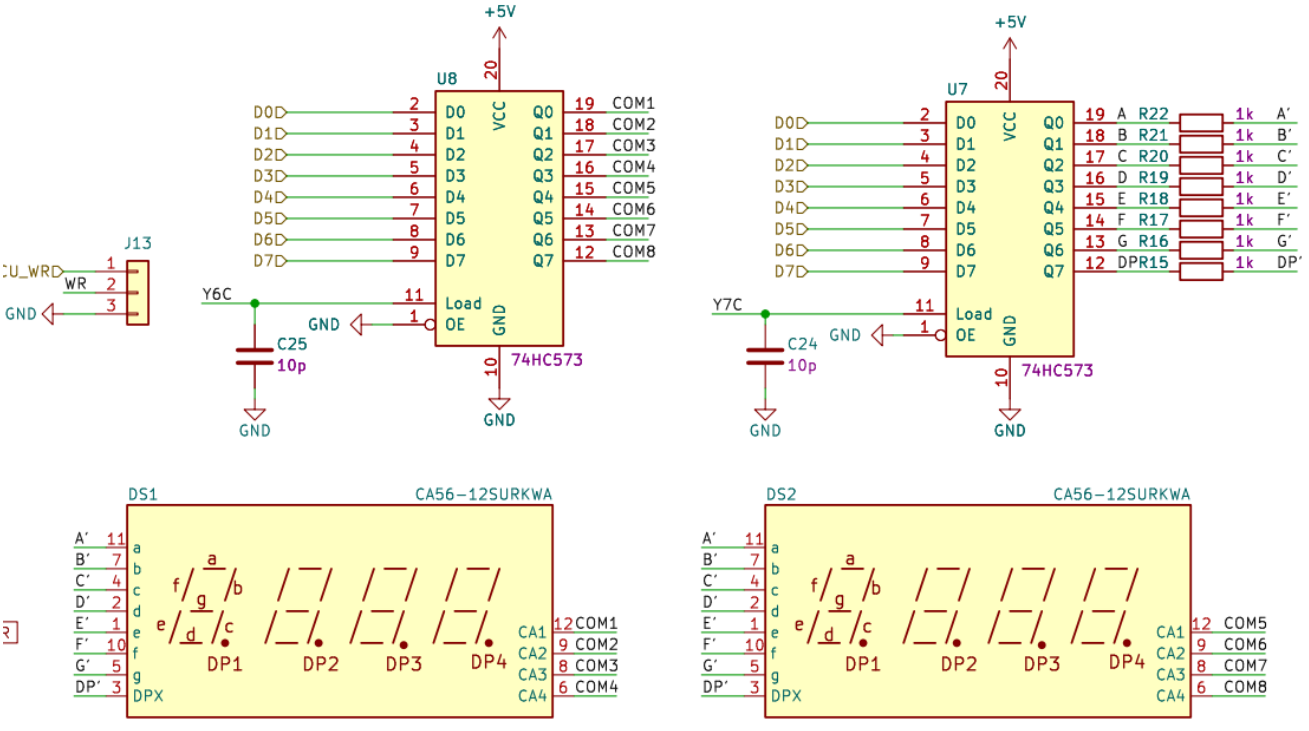
```
code unsigned char seg_dula[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99,
0x92,
                                 0x82, 0xf8, 0x80, 0x90, 0xff,
0x88};
```

数码管代码编写

这是我们最最重要的玩意，因为这个涉及到我们的显示，我们看原理图可以看到，我们需要对数码管进行位选（选择哪一个数码管进行显示），段选（选择数码管显示的数据）。

其中ABCDEFG和DP为我们的段选，COM1-8为我们的位选

我们需要先对P0进行段选显示后，使用P2将锁存器打开（Y7C→1110_0000→0xe0）;在对P0进行位选配置后，使用P2将锁存器打开（Y6C→1100_0000→0xc0）



经过我们的分析后我们就可以知道我们需要传入的数据为wela位选，dula段选，point是否显示小数点

```
void Seg_Disp(unsigned char wela, unsigned char dula, unsigned
char point)
```

我们为了避免闪烁，所以我们需要先对数码管进行消隐，即

```
P0 = 0xff;
P2 = P2 & 0x1f | 0xe0;
P2 &= 0x1f;
```

然后我们对位置进行选择后使用P2打开相应锁存器

```
P0 = 0x01 << wela;
P2 = P2 & 0x1f | 0xc0;
P2 &= 0x1f;
```

由于我们可以知道，要显示小数点的话，我们就要让DP为0，其他不变，即&0x7f即可完成

```
P0 = seg_dula[dula];
if (point) P0 &= 0x7f;
P2 = P2 & 0x1f | 0xe0;
P2 &= 0x1f;
```

总体代码如下

```
#include "seg.h"
// 0-9 灭
//  A
code unsigned char seg_dula[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99,
0x92,
                                 0x82, 0xf8, 0x80, 0x90, 0xff,
0x88};
void Seg_Disp(unsigned char wela, unsigned char dula, unsigned
char point) {
  // 消隐
  P0 = 0xff;
  P2 = P2 & 0x1f | 0xe0;
  P2 &= 0x1f;

  // 位选
  P0 = 0x01 << wela;
  P2 = P2 & 0x1f | 0xc0;
  P2 &= 0x1f;

  // 段选
```

```
    P0 = seg_dula[dula];
    if (point) P0 &= 0x7f;
    P2 = P2 & 0x1f | 0xe0;
    P2 &= 0x1f;
}
```

Seg.h

```
#include <STC15F2K60S2.H>
void Seg_Disp(unsigned char wela, unsigned char dula, unsigned
char point);
```

Ds1302时钟模版建立

Ds1302.c

这里有官方提供的参考代码，所以我们上面的时序可以不用书写，我们直接对下方的时间读取/写入进行操作

官方的参考代码

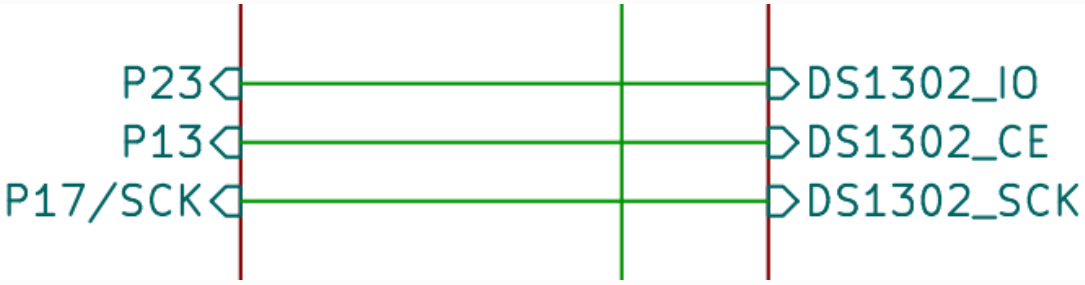
```
/*  #    DS1302代码片段说明
         1.   本文件夹中提供的驱动代码供参赛选手完成程序设计参考。
         2.
    参赛选手可以自行编写相关代码或以该代码为基础，根据所选单片机类型、运行速度
和试题
                    中对单片机时钟频率的要求，进行代码调试和修改。
```

```
*/
//
void Write_Ds1302(unsigned char temp) {
  unsigned char i;
  for (i = 0; i < 8; i++) {
    SCK = 0;
    SDA = temp & 0x01;
    temp >>= 1;
    SCK = 1;
  }
}

//
void Write_Ds1302_Byte(unsigned char address, unsigned char dat)
{
  RST = 0;
  _nop_();
  SCK = 0;
  _nop_();
  RST = 1;
  _nop_();
  Write_Ds1302(address);
  Write_Ds1302(dat);
  RST = 0;
}

//
unsigned char Read_Ds1302_Byte(unsigned char address) {
  unsigned char i, temp = 0x00;
  RST = 0;
  _nop_();
  SCK = 0;
  _nop_();
  RST = 1;
  _nop_();
  Write_Ds1302(address);
  for (i = 0; i < 8; i++) {
    SCK = 0;
    temp >>= 1;
    if (SDA) temp |= 0x80;
    SCK = 1;
  }
  RST = 0;
  _nop_();
  SCK = 0;
  _nop_();
  SCK = 1;
```

```
    _nop_();
    SDA = 0;
    _nop_();
    SDA = 1;
    _nop_();
    return (temp);
}
```

我们需要补充的代码

首先是我们需要绑定SDA，RST，SCK的引脚，在原理图中可以看到，并且要注意一个问题，我们用到了NOP函数，所以需要引入头文件intrins.h




```
#include "intrins.h"
sbit SDA = P2 ^ 3;
sbit RST = P1 ^ 3;
sbit SCK = P1 ^ 7;
```

然后就是我们的读写时钟，观察芯片手册可以看到

RTC

READ	WRITE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	RANGE
81h	80h	CH	10 Seconds			Seconds				00–59
83h	82h		10 Minutes			Minutes				00–59
85h	84h	12/$\overline{24}$	0	10 $\overline{\text{AM}}$/PM	Hour	Hour				1–12/0–23
87h	86h	0	0	10 Date		Date				1–31
89h	88h	0	0	0	10 Month	Month				1–12
8Bh	8Ah	0	0	0	0	0	Day			1–7
8Dh	8Ch	10 Year				Year				00–99
8Fh	8Eh	WP	0	0	0	0	0	0	0	—
91h	90h	TCS	TCS	TCS	TCS	DS	DS	RS	RS	—

CLOCK BURST

BFh	BEh

我们如果需要读取时钟的话，只需要读取地址0x85→小时 0x83→分钟 0x81→秒钟

我们如果需要写入时钟的话，需要先将0x8E位置的最高位WP置0，解除写保护后才能进行写入，写入的地址为0x84→小时 0x82→分钟 0x80→秒钟

所以我们的代码编写就很简单了,我们传入数据为数组的指针

```
void Set_Rtc(unsigned char *ucRtc)
void Read_Rtc(unsigned char *ucRtc)
```

这里给一个调用的示例

```
unsigned char time[3] = {11,12,13};//11:12:13
Set_Rtc(time);
Read_Rtc(time);
```

我们在写入之前需要进行解锁写保护，写完后就进行写保护

```
Write_Ds1302_Byte(0x8e, 0x00);
-----
Write_Ds1302_Byte(0x8e, 0x80);
```

我们在使用的过程中可以直接使用for循环进行读取/写入，由于我们为了方便，所以我们在这里使用十进制进行存储和读取（本质上存储和读取的都是二进制BCD码，比如0x11我们读取为11，写入为11，在里面内部使用变量进行转换）

举个🌰，我们假设要写入的时间为23:59:58，那么我们实际上要写入芯片的数据为0x23 0x59 0x58

但是我们为了方便（因为有时候需要我们是设置时钟，加减的那种，直接使用bcd码不方便，需要我们判断是否达到0xa0然后进行转换），这里我们可以在里面写一个十进制转为十进制bcd码的公式就行，以写入0x23为例，我们输入给我们Set_Rtc函数的数据为23，我们要先将其转换为0x23才能进行写入，那么我们可以观察一下，本质上就是把十位和个位分开，然后再存入即可，我们分开很简单，直接/10和%10就可以取出十位和个位，然后再使用*16来进行高位移位，将我们的十位移动到高四位去，然后|上我们的个位（低四位）就可以将其拼接完成。

23→2,3→0010,0011→0100_0011

```
temp = ucRtc[i] / 10 * 16 | ucRtc[i] % 10;
```

同理，我们读取的时候读取到的都是十进制的bcd码，那么我们将其转换后再读出

0x23→0x2,0x3→2*10+3→23

```
ucRtc[i] = temp /16 * 10 + temp & 0x0f;
```

读取和写入的代码如下

```
//写入
for (i = 0; i < 3; i++) {
    temp = ucRtc[i] / 10 *16 | ucRtc[i] % 10;
    Write_Ds1302_Byte(0x84 - 2 * i, temp);
}
//读取
for (i = 0; i < 3; i++) {
    temp = Read_Ds1302_Byte(0x85 - 2 * i);
    ucRtc[i] = temp /16 * 10 + temp & 0x0f;
}
```

需要补充的完整代码

```
// 十进制 11 → 0x11
void Set_Rtc(unsigned char *ucRtc) {
  unsigned char i;
  unsigned char temp;
  Write_Ds1302_Byte(0x8e, 0x00);
  for (i = 0; i < 3; i++) {
    temp = ucRtc[i] / 10 *16 | ucRtc[i] % 10;
    Write_Ds1302_Byte(0x84 - 2 * i, temp);
  }
  Write_Ds1302_Byte(0x8e, 0x80);
}
// 十进制 0x11 → 11
void Read_Rtc(unsigned char *ucRtc) {
  unsigned char i;
  unsigned char temp;
  for (i = 0; i < 3; i++) {
    temp = Read_Ds1302_Byte(0x85 - 2 * i);
    ucRtc[i] = temp /16 * 10 + temp & 0x0f;
```

```
  }
}
```

完整代码

```
/*  #   DS1302代码片段说明
        1.  本文件夹中提供的驱动代码供参赛选手完成程序设计参考。
        2.
   参赛选手可以自行编写相关代码或以该代码为基础，根据所选单片机类型、运行速度和试题
                中对单片机时钟频率的要求，进行代码调试和修改。
*/
#include "ds1302.h"

#include "intrins.h"
sbit SDA = P2 ^ 3;
sbit RST = P1 ^ 3;
sbit SCK = P1 ^ 7;
//
void Write_Ds1302(unsigned char temp) {
  unsigned char i;
  for (i = 0; i < 8; i++) {
    SCK = 0;
    SDA = temp & 0x01;
    temp >>= 1;
    SCK = 1;
  }
}

//
void Write_Ds1302_Byte(unsigned char address, unsigned char dat)
{
  RST = 0;
  _nop_();
  SCK = 0;
  _nop_();
  RST = 1;
  _nop_();
  Write_Ds1302(address);
  Write_Ds1302(dat);
  RST = 0;
}

//
unsigned char Read_Ds1302_Byte(unsigned char address) {
  unsigned char i, temp = 0x00;
```

```
  RST = 0;
  _nop_();
  SCK = 0;
  _nop_();
  RST = 1;
  _nop_();
  Write_Ds1302(address);
  for (i = 0; i < 8; i++) {
    SCK = 0;
    temp >>= 1;
    if (SDA) temp |= 0x80;
    SCK = 1;
  }
  RST = 0;
  _nop_();
  SCK = 0;
  _nop_();
  SCK = 1;
  _nop_();
  SDA = 0;
  _nop_();
  SDA = 1;
  _nop_();
  return (temp);
}
// 十进制 11 → 0x11
void Set_Rtc(unsigned char *ucRtc) {
  unsigned char i;
  unsigned char temp;
  Write_Ds1302_Byte(0x8e, 0x00);
  for (i = 0; i < 3; i++) {
    temp = ucRtc[i] / 10 *16 | ucRtc[i] % 10;
    Write_Ds1302_Byte(0x84 - 2 * i, temp);
  }
  Write_Ds1302_Byte(0x8e, 0x80);
}
// 十进制 0x11 → 11
void Read_Rtc(unsigned char *ucRtc) {
  unsigned char i;
  unsigned char temp;
  for (i = 0; i < 3; i++) {
    temp = Read_Ds1302_Byte(0x85 - 2 * i);
    ucRtc[i] = temp /16 * 10 + temp & 0x0f;
  }
}
```

Ds1302.h

```c
#include <STC15F2K60S2.H>
void Set_Rtc(unsigned char *ucRtc);
void Read_Rtc(unsigned char *ucRtc);
```

Onewire温度模版建立

Onewire.c

官方的参考代码

这里有官方提供的参考代码，所以我们上面的时序可以不用书写，我们直接对下方的进行温度读取操作。

```c
/*  #    单总线代码片段说明
        1.   本文件夹中提供的驱动代码供参赛选手完成程序设计参考。
        2.
    参赛选手可以自行编写相关代码或以该代码为基础，根据所选单片机类型、运行速度和试题
                    中对单片机时钟频率的要求，进行代码调试和修改。
*/
//
void Delay_OneWire(unsigned int t) {
  unsigned char i;
  while (t--) {
    for (i = 0; i < 12; i++);
```

```
  }
}

//
void Write_DS18B20(unsigned char dat) {
  unsigned char i;
  for (i = 0; i < 8; i++) {
    DQ = 0;
    DQ = dat & 0x01;
    Delay_OneWire(5);
    DQ = 1;
    dat >>= 1;
  }
  Delay_OneWire(5);
}

//
unsigned char Read_DS18B20(void) {
  unsigned char i;
  unsigned char dat;

  for (i = 0; i < 8; i++) {
    DQ = 0;
    dat >>= 1;
    DQ = 1;
    if (DQ) {
      dat |= 0x80;
    }
    Delay_OneWire(5);
  }
  return dat;
}

//
bit init_ds18b20(void) {
  bit initflag = 0;

  DQ = 1;
  Delay_OneWire(12);
  DQ = 0;
  Delay_OneWire(80);
  DQ = 1;
  Delay_OneWire(10);
  initflag = DQ;
  Delay_OneWire(5);

  return initflag;
```

```
}
```

我们需要补充的代码

首先是我们需要绑定DQ的引脚，在原理图中可以看到


```
sbit DQ = P1 ^ 4;
```

我们不需要管上方的时序的书写，只需要进行温度读取的代码书写就行，

下方是温度芯片的使用顺序（官方手册），首先我们需要运行init，然后再检查rom（可跳过），最后执行我们的功能函数，注意，一次只能使用一个功能。

DS18B20

TRANSACTION SEQUENCE

The transaction sequence for accessing the DS18B20 is as follows:

Step 1. Initialization

Step 2. ROM Command (followed by any required data exchange)

Step 3. DS18B20 Function Command (followed by any required data exchange)

It is very important to follow this sequence every time the DS18B20 is accessed, as the DS18B20 will not respond if any steps in the sequence are missing or out of order. Exceptions to this rule are the Search ROM [F0h] and Alarm Search [ECh] commands. After issuing either of these ROM commands, the master must return to Step 1 in the sequence.

我们可以在官方手册上找到需要运行的指令

DS18B20

SKIP ROM [CCh]

The master can use this command to address all devices on the bus simultaneously without sending out any ROM code information. For example, the master can make all DS18B20s on the bus perform simultaneous temperature conversions by issuing a Skip ROM command followed by a Convert T [44h] command.

Note that the Read Scratchpad [BEh] command can follow the Skip ROM command only if there is a single slave device on the bus. In this case, time is saved by allowing the master to read from the slave without sending the device's 64-bit ROM code. A Skip ROM command followed by a Read Scratchpad command will cause a data collision on the bus if there is more than one slave since multiple devices will attempt to transmit data simultaneously.

当我们发送0xcc的时候，就会跳过ROM检查，并且这里也说明了，我们发送0x44就可以开启温度转换，发送0xbe就可以开始读取温度，但是要注意一个事情，我们同样可以在我们的官方的手册上面找到一段读取的语句，这里表明我们先读取低位，再读取高位，最后将其拼接后再*精度就是我们的数据了

Figure 2. Temperature Register Format

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Table 1. Temperature/Data Relationship

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

温度的精度在文档中对应如下

OPERATION—MEASURING TEMPERATURE

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the temperature sensor is user-configurable to 9, 10, 11, or 12 bits, corresponding to increments of 0.5°C, 0.25°C, 0.125°C, and 0.0625°C, respectively. The default resolution at power-up is 12-bit. The DS18B20 powers up in a low-power idle state. To initiate a temperature measurement and A-to-D conversion, the master must issue a Convert T [44h] command. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18B20 returns to its idle state. If the DS18B20 is powered by an external supply, the master can issue "read time slots" (see the *1-Wire Bus System* section) after the Convert T command and the DS18B20 will respond by transmitting 0 while the temperature conversion is in progress and 1 when the conversion is done. If the DS18B20 is powered with parasite power, this notification technique cannot be used since the bus must be pulled high by a strong pullup during the entire temperature conversion. The bus requirements for parasite power are explained in detail in the *Powering the DS18B20* section.

精度为9→0.5;10→0.25;11→0.125;12（默认）→0.0625

我们就可以根据上面的描述来进行代码书写

```
float rd_temperature() {
    unsigned char low, high;
    //开启温度转换
    init_ds18b20();
    Write_DS18B20(0xcc);
    Write_DS18B20(0x44);
    Delay_OneWire(200);
    //开启温度读取
    init_ds18b20();
    Write_DS18B20(0xcc);
    Write_DS18B20(0xbe);
    low = Read_DS18B20();
    high = Read_DS18B20();
    return (float)(high << 8 | low) * 0.0625;
}
```

这里给一个调用示例

```
float temp = rd_temperature();
```

完整代码如下

```
/*  #    单总线代码片段说明
        1.  本文件夹中提供的驱动代码供参赛选手完成程序设计参考。
        2.
   参赛选手可以自行编写相关代码或以该代码为基础，根据所选单片机类型、运行速度和试题
                    中对单片机时钟频率的要求，进行代码调试和修改。
*/
#include "onewire.h"
sbit DQ = P1 ^ 4;
//
void Delay_OneWire(unsigned int t) {
  unsigned char i;
  while (t--) {
    for (i = 0; i < 12; i++);
  }
}

//
void Write_DS18B20(unsigned char dat) {
  unsigned char i;
  for (i = 0; i < 8; i++) {
    DQ = 0;
```

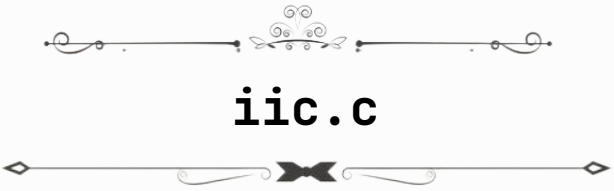
```
    DQ = dat & 0x01;
    Delay_OneWire(5);
    DQ = 1;
    dat >>= 1;
  }
  Delay_OneWire(5);
}

//
unsigned char Read_DS18B20(void) {
  unsigned char i;
  unsigned char dat;

  for (i = 0; i < 8; i++) {
    DQ = 0;
    dat >>= 1;
    DQ = 1;
    if (DQ) {
      dat |= 0x80;
    }
    Delay_OneWire(5);
  }
  return dat;
}

//
bit init_ds18b20(void) {
  bit initflag = 0;

  DQ = 1;
  Delay_OneWire(12);
  DQ = 0;
  Delay_OneWire(80);
  DQ = 1;
  Delay_OneWire(10);
  initflag = DQ;
  Delay_OneWire(5);

  return initflag;
}

float rd_temperature() {
  unsigned char low, high;
  init_ds18b20();
  Write_DS18B20(0xcc);
  Write_DS18B20(0x44);
  Delay_OneWire(200);
```

```
  init_ds18b20();
  Write_DS18B20(0xcc);
  Write_DS18B20(0xbe);
  low = Read_DS18B20();
  high = Read_DS18B20();
  return (float)(high << 8 | low) * 0.0625;
}
```

Onewire.h

```
#include <STC15F2K60S2.H>
float rd_temperature();
```

IIC（PCF8591(AD/DA)与AT24C02(EEPROM)）模版建立



iic.c

官方的参考代码

这里有官方提供的参考代码，所以我们上面的时序可以不用书写，我们直接进行AD读取与DA输出还有AT24C02的存储操作

```
/*  #    I2C代码片段说明
```

1. 本文件夹中提供的驱动代码供参赛选手完成程序设计参考。
2. 参赛选手可以自行编写相关代码或以该代码为基础，根据所选单片机类型、运行速度和试题中对单片机时钟频率的要求，进行代码调试和修改。

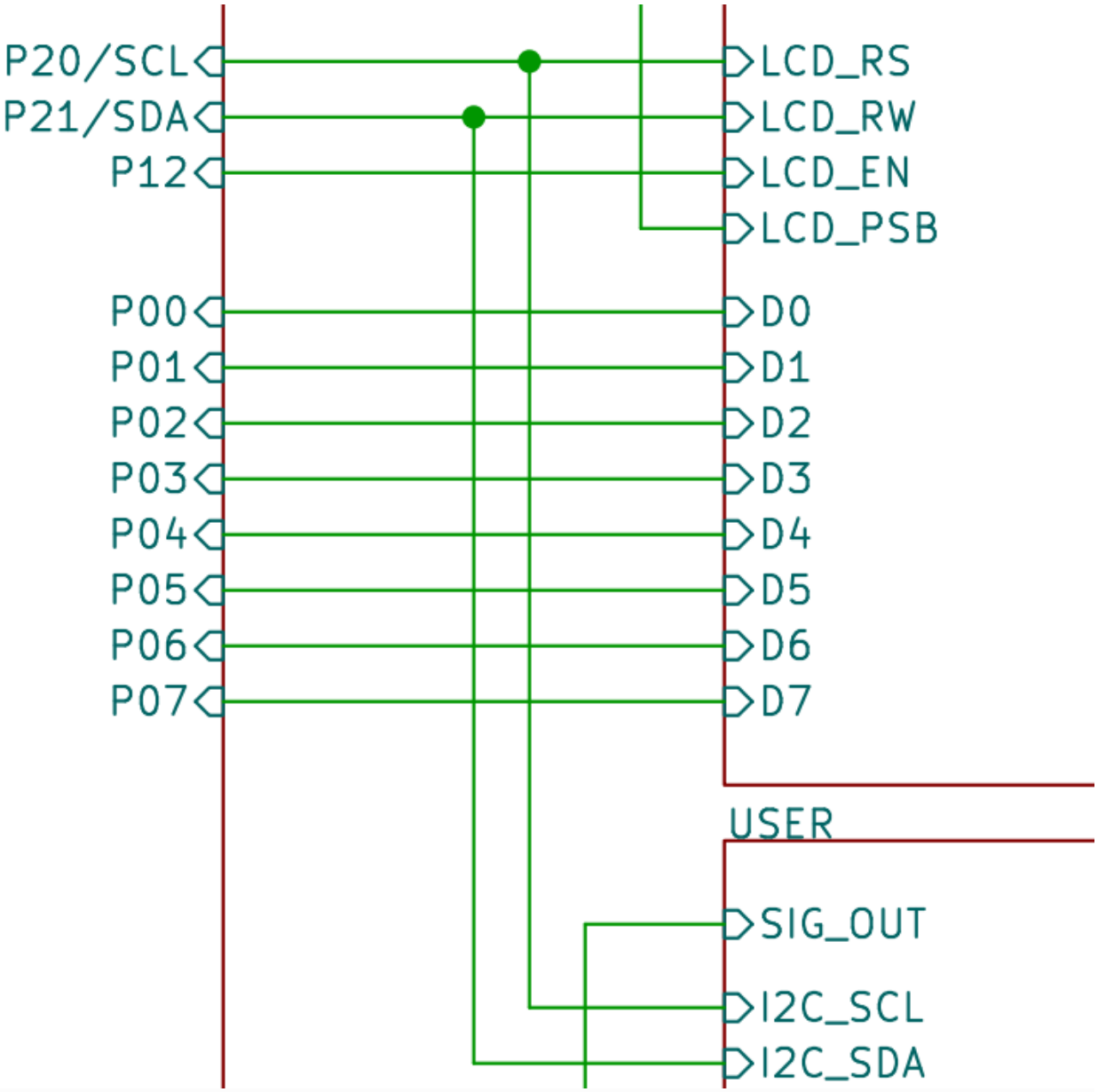
```
*/
#define DELAY_TIME 10
//
static void I2C_Delay(unsigned char n) {
  do {
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
  } while (n--);
}

//
void I2CStart(void) {
  sda = 1;
  scl = 1;
  I2C_Delay(DELAY_TIME);
  sda = 0;
  I2C_Delay(DELAY_TIME);
  scl = 0;
}

//
void I2CStop(void) {
  sda = 0;
  scl = 1;
  I2C_Delay(DELAY_TIME);
  sda = 1;
  I2C_Delay(DELAY_TIME);
}
```

```
//
void I2CSendByte(unsigned char byt) {
  unsigned char i;

  for (i = 0; i < 8; i++) {
    scl = 0;
    I2C_Delay(DELAY_TIME);
    if (byt & 0x80) {
      sda = 1;
    } else {
      sda = 0;
    }
    I2C_Delay(DELAY_TIME);
    scl = 1;
    byt ⟸ 1;
    I2C_Delay(DELAY_TIME);
  }

  scl = 0;
}

//
unsigned char I2CReceiveByte(void) {
  unsigned char da;
  unsigned char i;
  for (i = 0; i < 8; i++) {
    scl = 1;
    I2C_Delay(DELAY_TIME);
    da ⟸ 1;
    if (sda) da ⊨ 0x01;
    scl = 0;
    I2C_Delay(DELAY_TIME);
  }
  return da;
}

//
unsigned char I2CWaitAck(void) {
  unsigned char ackbit;

  scl = 1;
  I2C_Delay(DELAY_TIME);
  ackbit = sda;
  scl = 0;
  I2C_Delay(DELAY_TIME);

  return ackbit;
```

```
}

//
void I2CSendAck(unsigned char ackbit) {
  scl = 0;
  sda = ackbit;
  I2C_Delay(DELAY_TIME);
  scl = 1;
  I2C_Delay(DELAY_TIME);
  scl = 0;
  sda = 1;
  I2C_Delay(DELAY_TIME);
}
```

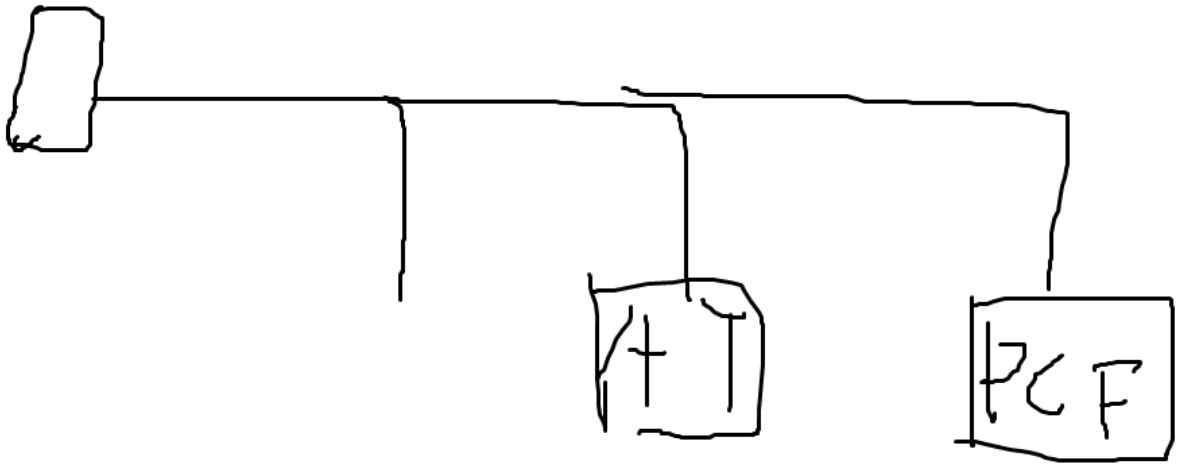
我们需要补充的代码

首先是我们需要绑定sda和scl的引脚，在原理图中可以看到，并且要注意一个问题，我们用到了NOP函数，所以需要引入头文件intrins.h




```c
#include "intrins.h"
sbit scl = P2 ^ 0;
sbit sda = P2 ^ 1;
```

我在这里需要说明一下IIC通信的方式，他是一个一个八位的数据，其中第一位是我们的发送地址，永远都是1，然后后面挂载了不同的外设，我们的蓝桥杯的板子挂了两个外设，一个是PCF8591，对应着AD/DA的功能；另一个是AT24C02，对应着EEPROM的存储功能。在我们板子上就长我们下面这个样子~~（我画的有点抽象）~~，所以我们就可以知道，如果我们要对AT24C02芯片写入数据的话，我们就需要使用地址为（1010_0000→0xa0），要读取数据的化，我们就要使用地址为（1010_0001→0xa1）；要写入DA输出的数据的话，我们就需要使用地址为（1001_0000→0x90），要读取AD采样的数据的话，我们就需要使用地址为（1001_0001->0x91）



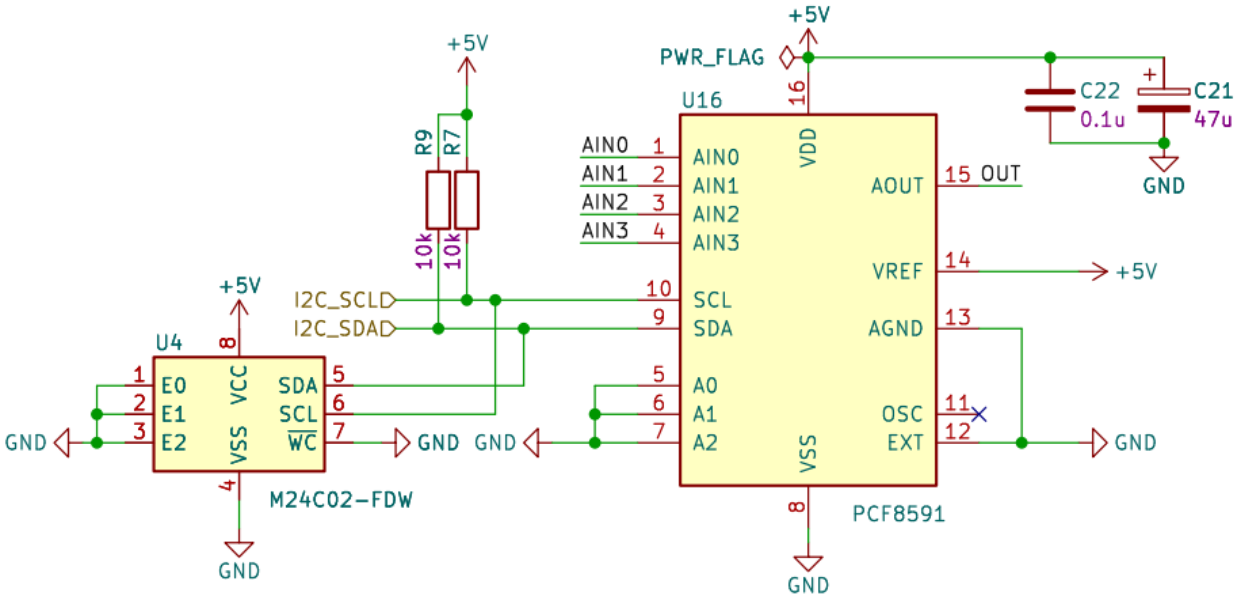
在我们使用IIC进行通信的时候，我们要遵循下面的运行逻辑

```
I2CStart();
I2CSendByte(地址);
I2CWaitAck();
I2CSendByte(功能);
I2CWaitAck();
```

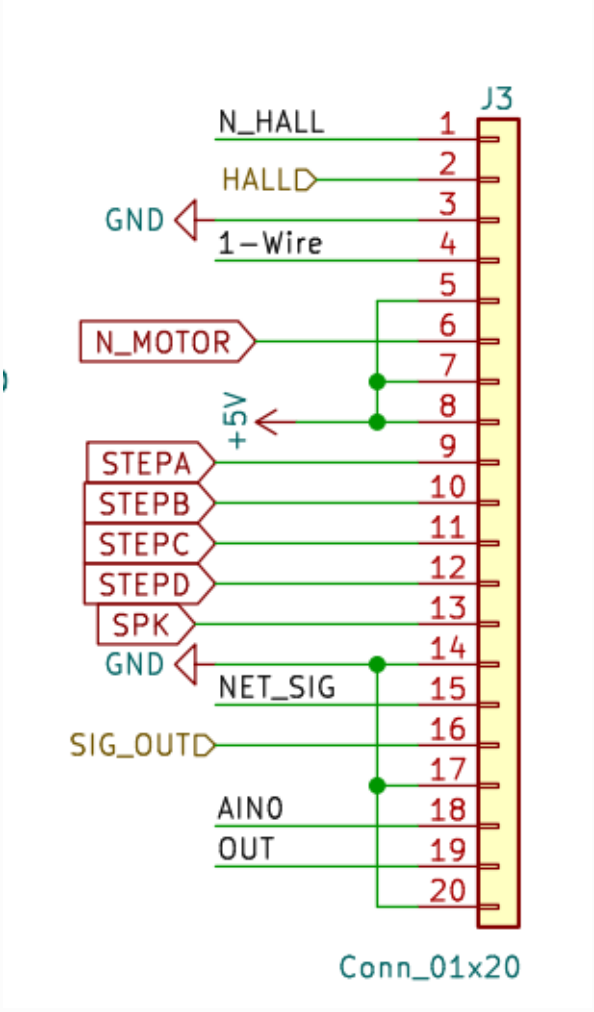
AD与DA

从上面说明的内容的我们可以知道，如果我们要对PCF8591进行操作，那么就需要对地址0x90和0x91进行操作

我们现在先说一下AD的代码，通过观察原理图我们可以发现，AD有4个通道可以进行读取



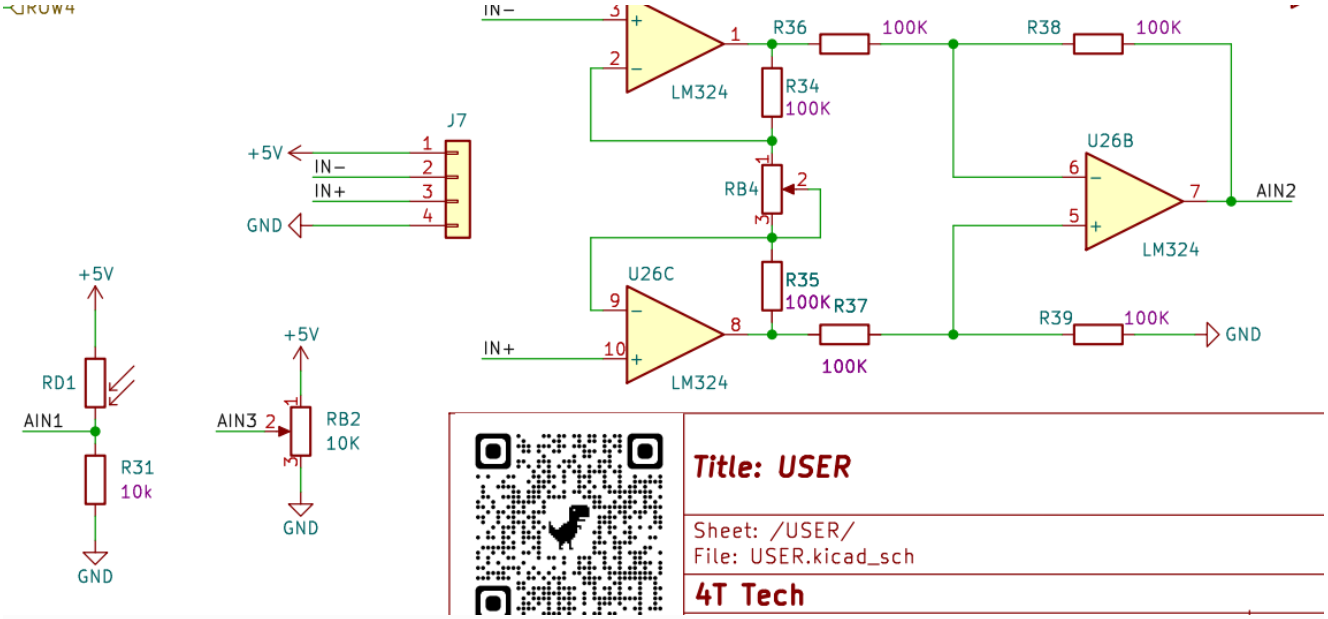
其中AIN0通道是我们的外部测量通道（由于蓝桥杯现在还没有涉及到信号发生器，所以我们可以先暂时不用管这个通道）



AIN1通道很明显就是光敏电阻的分压读取（光照越强，分压越高，采集到的结果越大），

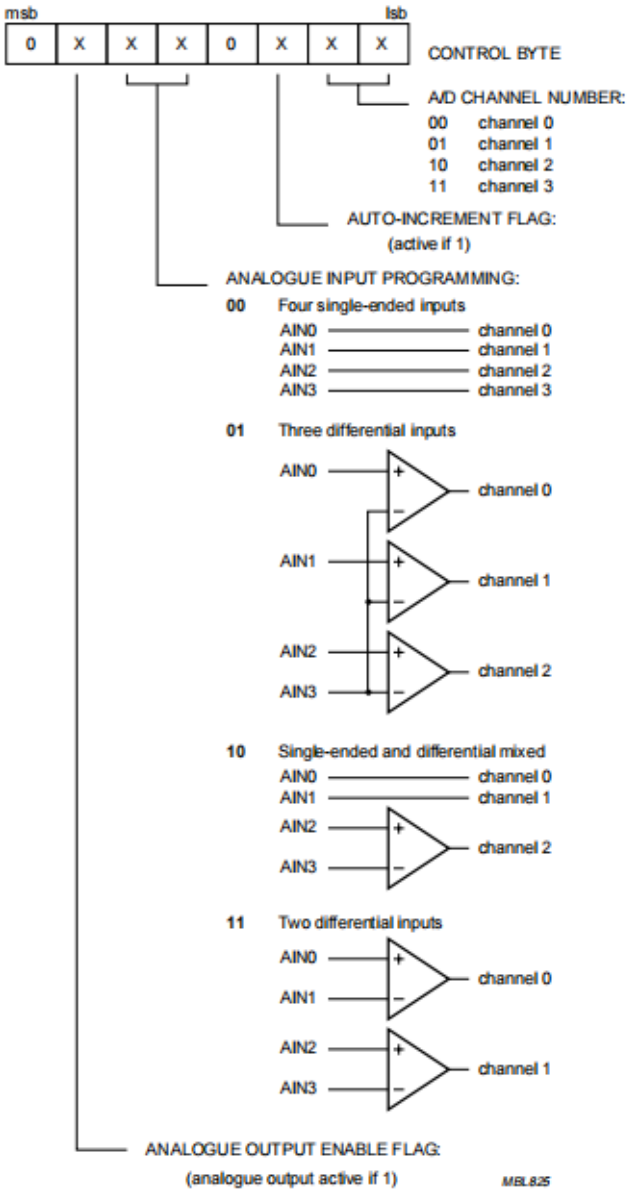
AIN2通道为差分输入（我们也是不常用到的输入，因为没有信号发生器）

AIN3通道为滑动变阻器输入（我们测量滑动变阻器的分压，这个你左右旋钮看一下就知道哪边变大了）



我们查看一下PCF8591的手册可以看到，我们读取的地址的书写方式，这里我统一给一个说明

位次	情况说明	情况
第一位	固定位	0
第二位	是否需要开启DA输出	开启DA输出（1） 不开启DA输出（0）
第三四位	选择输入通道的模式 差分输入具体要看下面的图	通道与编号对应输入（常用）（00） 通道差分输入（01,10,11）
第五位	固定位	0
第六位	自动递增标志 （用于一个一个读取AD输入口）	自我递增（没考过，因为我们一般都是单一读取）（1） 不自我递增（0）
第七八位	用于指定我们读取的通道	通道0（00），在我们正常的对应输入下为外部信号输入 通道1（01），在我们正常的对应输入下为光明电阻分压输入 通道2（10），在我们正常的对应输入下为差分信号输入 通道3，在我们正常的对应输入下为滑动变阻器分压输入（11）



我们那么这个时候常考的就是下面的形式，注意一下题目中是否需求DA输出，如果需要DA输出的话，这里使用地址的时候必须使用允许DA输出，

需要的数据	输入的地址
外部电压输入（开启DA）	0100_0000→0x40
外部电压输入（不开启DA）	0000_0000→0x00
光敏电阻分压输入（开启DA）	0100_0001→0x41
光敏电阻分压输入（不开启DA）	0000_0001→0x01
差分信号输入（开启DA）	0100_0010→0x42
差分信号输入（不开启DA）	0000_0010→0x02
滑动变阻器分压输入（开启DA）	0100_0011→0x43
滑动变阻器分压输入（不开启DA）	0000_0011→0x03

有了这样一个基础，我们就可以开始着手编写我们的AD代码了

我们只需要指定一下我们需要读取的通道的地址就行

```
unsigned char Ad_Read(unsigned char addr)
```

这里给一个调用示例，这里需要注意的是，如果你同时读取光敏电阻和滑动变阻器的值，他们会地址交换，你们可以自己去试试。

```
unsigned char temp = Ad_Read(0x01);//读取光敏电阻结果
```

首先我们需要选中我们的PCF芯片并且写入我们需要读取的地址，这里需要注意的就只有每次发送Byte后我们需要Wait一下

```
// 选择芯片为PCF
I2CStart();
I2CSendByte(0x90);
I2CWaitAck();
I2CSendByte(addr);
I2CWaitAck();
```

然后重新开启一下通话（因为我们上面是进行写操作，现在要进行读操作），读取后我们需要发送一个Ack为1，表明我们这边接收数据完成，后续不用接受了，结束后就可以关闭对话Stop了

```
I2CStart();
I2CSendByte(0x91);
I2CWaitAck();
temp = I2CReceiveByte();
I2CSendAck(1);
I2CStop();
```

AD完整代码

```
unsigned char Ad_Read(unsigned char addr) {
  unsigned char temp;
  // 选择芯片为PCF
  I2CStart();
  I2CSendByte(0x90);
  I2CWaitAck();
```

```
    I2CSendByte(addr);
    I2CWaitAck();

    I2CStart();
    I2CSendByte(0x91);
    I2CWaitAck();
    temp = I2CReceiveByte();
    I2CSendAck(1);
    I2CStop();
    return temp;
}
```

说完AD我们就要说到DA了

和AD一样，我们只需要传入我们需要写入的数据就行，但是要注意一下我们的数据为数字电压，即0-255，我们输出后的电压是模拟电压，即0-5V，所以我们是一一映射过去的

```
void Da_Write(unsigned char dat)
```

举个🌰，我们如果想要输出电压为2V,那么我们就应该

```
Da_Write(2*51);
```

如果想要输出的电压为2.5V，那么我们就应该，这里不用关心他会出现不匹配的情况，C语言底层会自动转换为unsigned char类型的,这类有啊注意

```
Da_Write(2.5*51);
```

首先我们需要选中我们的PCF芯片并且给他我们的地址信息（前面提到过如果要开启DA输出，那么我们应该写入的地址）,这里需要注意的就只有每次发送Byte后我们需要Wait一下，这里的0x41可以更换为0x40-0x43的任何一个，但是最好和题目中要求AD需要读取的通道一致（这里也要注意在你读取AD的时候加上0x4）

```
void Da_Write(unsigned char dat) {
  // 选择芯片为PCF
  I2CStart();
  I2CSendByte(0x90);
  I2CWaitAck();
  I2CSendByte(0x41);
  I2CWaitAck();
  I2CSendByte(dat);
  I2CWaitAck();
}
```

AD与

EEPROM

从上面说明的内容的我们可以知道，如果我们要对AT24C02进行操作，那么就需要对地址0xa0和0xa1进行操作

本质上我们的EEPROM写入与读取和AD/DA是一致的，但是我们需要注意一个点，因为我们通常写入/读取的不止一个数据，我们一般是一个数组写进去和读取，所以我们就需要注意这个点，我们的操作如下，传递的数据都是数组的首地址（如果存单个数据只需要使用&name就行），传入参数为需要写入/读取数组首地址名称）） str，需要写入/读取的数组地址addr，数组的大小num

```
void EEPROM_Write(unsigned char *str, unsigned char addr,
unsigned char num)
void EEPROM_Read(unsigned char *str, unsigned char addr, unsigned
char num)
```

这里给一个调用的示例，需要注意一下如果写入为unsigned int，可以不用分为高低位存储，但是需要分为高低位读取

```
unsigned char arr_input[5]={1,2,3,4,5};
unsigned char arr_output[5];
EEPROM_Write(arr_input,0,5);
EEPROM_Read(arr_output,0,5);

unsigned char input = 5;
unsigned char output;
EEPROM_Write(&input,0,1);
EEPROM_Read(&output,0,1);

unsigned int input = 5;
```

```
unsigned int output;
unsigned char high,low;
EEPROM_Write(&input,0,2);
EEPROM_Read(&high,0,1);
EEPROM_Read(&low,1,1);
output = high<<4 | low;
```

现在我来说一下实现细节

先说明一下EEPROM_Read函数，和上文的AD读取一样，我们需要先选中AT24C02，并且功能为写，然后进行位置的传输，确认我们需要读取的地址

```
I2CStart();
I2CSendByte(0xa0);
I2CWaitAck();
I2CSendByte(addr);
I2CWaitAck();
```

在我们读取的时候，和上文AD一样，我们需要重新开启一个对话进行读取

```
I2CStart();
I2CSendByte(0xa1);
I2CWaitAck();
```

但是在我们读取的时候需要注意一个点，我们使用的数组进行的数据读取，并且在我们读取的时候对指针进行++操作，当我们还没有读完数据的时候，我们发送Ack为0，代表我们需要继续读取，当我们读取数据结束后，我们发送Ack为1，表示读取结束，最后也别忘记需要进行Stop停止交流。

```
I2CStart();
I2CSendByte(0xa1);
I2CWaitAck();
while (num--) {
    *str++ = I2CReceiveByte();
    if (num)
        I2CSendAck(0);
    else
        I2CSendAck(1);
}
I2CStop();
```

在这里说明一下*str++代表的含义，这是c的一个写法，等价于下面的代码

```
unsigned char arr[5];
unsigned char i = 0;
unsigned char num = 5;
while(num--)
{
    arr[i] = I2CReceiveByte();
i++;
-----
}
```

完整的代码EEPROM_READ如下

```
void EEPROM_Read(unsigned char *str, unsigned char addr, unsigned
char num) {
  I2CStart();
  I2CSendByte(0xa0);
  I2CWaitAck();
  I2CSendByte(addr);
  I2CWaitAck();

  I2CStart();
  I2CSendByte(0xa1);
  I2CWaitAck();
  while (num--) {
    *str++ = I2CReceiveByte();
    if (num)
      I2CSendAck(0);
    else
      I2CSendAck(1);
  }
  I2CStop();
}
```

我们可以从上面看到，AD和EEPROM_READ可以进行类比；同理，我们的DA和EEPROM_Write也可以进行类比，但是EEPROM_Write有一个需要注意的点，我会在下面进行说明

我们第一步还是选择我们的芯片，选择我们需要写入的地址

```
I2CStart();
I2CSendByte(0xa0);
I2CWaitAck();
I2CSendByte(addr);
I2CWaitAck();
```

第二步就是使用一个循环进行写入，这里需要注意一个问题，就是我们每写入一个数据，都需要I2C_Delay一下，注意不是Delay，而是I2C_Delay，经过我们测试，当这个值为200 的时候效果是可以的，并且这里有个最最重要的点，在我们Stop结束后，我们需要进行4个255时间的I2C_Delay~~（其实我在群里面说过，但你们一直觉得我在讲乐子）~~

```
while (num--) {
    I2CSendByte(*str++);
    I2CWaitAck();
    I2C_Delay(200);
}
I2CStop();
I2C_Delay(255);
I2C_Delay(255);
I2C_Delay(255);
I2C_Delay(255);
```

完整的代码EEPROM_Write如下

```
void EEPROM_Write(unsigned char *str, unsigned char addr,
unsigned char num) {
  I2CStart();
  I2CSendByte(0xa0);
  I2CWaitAck();
  I2CSendByte(addr);
  I2CWaitAck();

  while (num--) {
    I2CSendByte(*str++);
    I2CWaitAck();
    I2C_Delay(200);
  }
  I2CStop();
  I2C_Delay(255);
  I2C_Delay(255);
  I2C_Delay(255);
  I2C_Delay(255);
```

```
}
```

完整代码如下

```
/*  #    I2C代码片段说明
        1.  本文件夹中提供的驱动代码供参赛选手完成程序设计参考。
        2.
    参赛选手可以自行编写相关代码或以该代码为基础，根据所选单片机类型、运行速度和试题
                    中对单片机时钟频率的要求，进行代码调试和修改。
*/
#include "iic.h"

#include "intrins.h"
#define DELAY_TIME 5
sbit scl = P2 ^ 0;
sbit sda = P2 ^ 1;
//
static void I2C_Delay(unsigned char n) {
  do {
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
  } while (n--);
}

//
void I2CStart(void) {
  sda = 1;
  scl = 1;
  I2C_Delay(DELAY_TIME);
  sda = 0;
  I2C_Delay(DELAY_TIME);
  scl = 0;
```

```
}

//
void I2CStop(void) {
  sda = 0;
  scl = 1;
  I2C_Delay(DELAY_TIME);
  sda = 1;
  I2C_Delay(DELAY_TIME);
}

//
void I2CSendByte(unsigned char byt) {
  unsigned char i;

  for (i = 0; i < 8; i++) {
    scl = 0;
    I2C_Delay(DELAY_TIME);
    if (byt & 0x80) {
      sda = 1;
    } else {
      sda = 0;
    }
    I2C_Delay(DELAY_TIME);
    scl = 1;
    byt <<= 1;
    I2C_Delay(DELAY_TIME);
  }

  scl = 0;
}

//
unsigned char I2CReceiveByte(void) {
  unsigned char da;
  unsigned char i;
  for (i = 0; i < 8; i++) {
    scl = 1;
    I2C_Delay(DELAY_TIME);
    da <<= 1;
    if (sda) da |= 0x01;
    scl = 0;
    I2C_Delay(DELAY_TIME);
  }
  return da;
}
```

```
//
unsigned char I2CWaitAck(void) {
  unsigned char ackbit;

  scl = 1;
  I2C_Delay(DELAY_TIME);
  ackbit = sda;
  scl = 0;
  I2C_Delay(DELAY_TIME);

  return ackbit;
}

//
void I2CSendAck(unsigned char ackbit) {
  scl = 0;
  sda = ackbit;
  I2C_Delay(DELAY_TIME);
  scl = 1;
  I2C_Delay(DELAY_TIME);
  scl = 0;
  sda = 1;
  I2C_Delay(DELAY_TIME);
}

unsigned char Ad_Read(unsigned char addr) {
  unsigned char temp;
  // 选择芯片为PCF
  I2CStart();
  I2CSendByte(0x90);
  I2CWaitAck();
  I2CSendByte(addr);
  I2CWaitAck();

  I2CStart();
  I2CSendByte(0x91);
  I2CWaitAck();
  temp = I2CReceiveByte();
  I2CSendAck(1);
  I2CStop();
  return temp;
}
// 数字电压255→5V
void Da_Write(unsigned char dat) {
  // 选择芯片为PCF
  I2CStart();
  I2CSendByte(0x90);
```

```
  I2CWaitAck();
  I2CSendByte(0x41);
  I2CWaitAck();
  I2CSendByte(dat);
  I2CWaitAck();
}
void EEPROM_Write(unsigned char *str, unsigned char addr,
unsigned char num) {
  I2CStart();
  I2CSendByte(0xa0);
  I2CWaitAck();
  I2CSendByte(addr);
  I2CWaitAck();

  while (num--) {
    I2CSendByte(*str++);
    I2CWaitAck();
    I2C_Delay(200);
  }
  I2CStop();
  I2C_Delay(255);
  I2C_Delay(255);
  I2C_Delay(255);
  I2C_Delay(255);
}
void EEPROM_Read(unsigned char *str, unsigned char addr, unsigned
char num) {
  I2CStart();
  I2CSendByte(0xa0);
  I2CWaitAck();
  I2CSendByte(addr);
  I2CWaitAck();

  I2CStart();
  I2CSendByte(0xa1);
  I2CWaitAck();
  while (num--) {
    *str++ = I2CReceiveByte();
    if (num)
      I2CSendAck(0);
    else
      I2CSendAck(1);
  }
  I2CStop();
}
```

iic.h

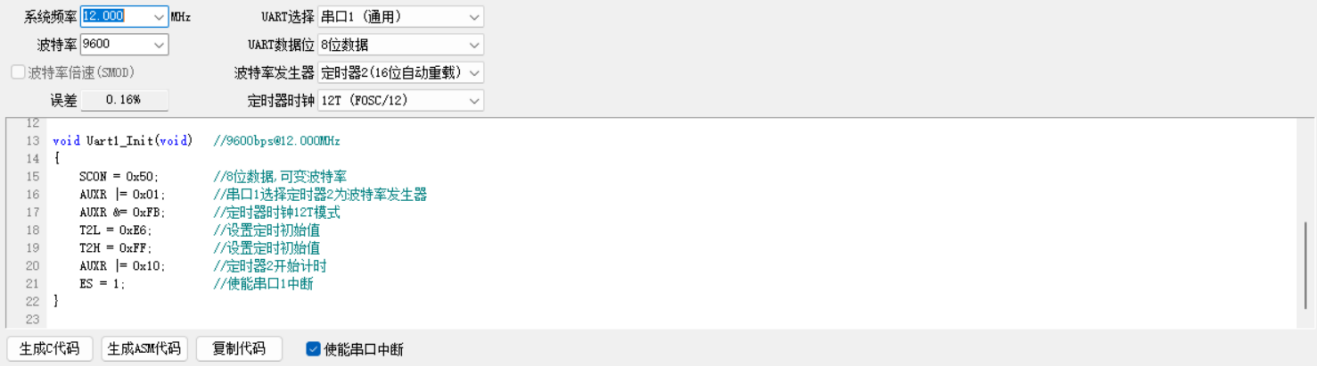
```c
#include <STC15F2K60S2.H>
unsigned char Ad_Read(unsigned char addr);
void Da_Write(unsigned char dat);
void EEPROM_Write(unsigned char *str, unsigned char addr,
unsigned char num);
void EEPROM_Read(unsigned char *str, unsigned char addr, unsigned
char num);
```

Uart串口模版建立

uart.c

在这里我们需要注意，串口官方是没有给底层的，我们需要自己手搓，在这里我给的代码为重定向代码。

首先是使用isp生成一个用定时器2计时的串口1的代码，波特率为9600（比赛的时候可能不是9600，要根据题目来看），定时器时钟选用12T，数据位为8位



我们直接把他复制到我们代码里面就行，最后别忘记加一个EA=1来开启总中断，还有些isp版本可能比较老，没有使能中断的选项，所以需要自己加上ES=1

```
void Uart1_Init(void)  // 9600bps@12.000MHz
{
  SCON = 0x50;    // 8位数据,可变波特率
  AUXR |= 0x01;   // 串口1选择定时器2为波特率发生器
  AUXR |= 0x04;   // 定时器时钟1T模式
  T2L = 0xC7;     // 设置定时初始值
  T2H = 0xFE;     // 设置定时初始值
  AUXR |= 0x10;   // 定时器2开始计时
  ES = 1;         // 使能串口1中断
  EA = 1;
}
```

我们这里不需要实现复杂的函数，只需要实现一个重定向就行，使用printf直接将数据发送到串口就行，我现在说一下步骤

我们通过查看c51里面（注意是c51，不是标准的c语言函数库，直接在keil里面右键打开），我们可以看到下方有一个putchar的一个extern的函数定义，这就是我们需要重载的函数，因为我们printf本质上就是调用putchar的

```
/*-----------------------------------------------------------------
-----------
STDIO.H

Prototypes for standard I/O functions.
Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software,
Inc.
All rights reserved.
-----------------------------------------------------------------
---------*/

#ifndef __STDIO_H__
#define __STDIO_H__

#ifndef EOF
 #define EOF -1
#endif

#ifndef NULL
 #define NULL ((void *) 0)
#endif

#ifndef _SIZE_T
 #define _SIZE_T
 typedef unsigned int size_t;
```

```
#endif

#pragma SAVE
#pragma REGPARMS
extern char _getkey (void);
extern char getchar (void);
extern char ungetchar (char);
extern char putchar (char);
extern int printf   (const char *, ...);
extern int sprintf  (char *, const char *, ...);
extern int vprintf  (const char *, char *);
extern int vsprintf (char *, const char *, char *);
extern char *gets (char *, int n);
extern int scanf (const char *, ...);
extern int sscanf (char *, const char *, ...);
extern int puts (const char *);

#pragma RESTORE

#endif
```

我们可以知道，串口在发送的时候，会将数据放进缓冲区SBUF（如果你用串口2那就是S2BUF，但是这里没涉及到，就不用管了），当我们正在发送的时候TI标志位为0，当我们发送完成后，他就会被置为0，然后我们需要将ch返回（这个是函数本身就需要返回的char）

```
extern char putchar(char ch) {
  SBUF = ch;
  while (TI == 0);
  TI = 0;
  return ch;
}
```

uart.h

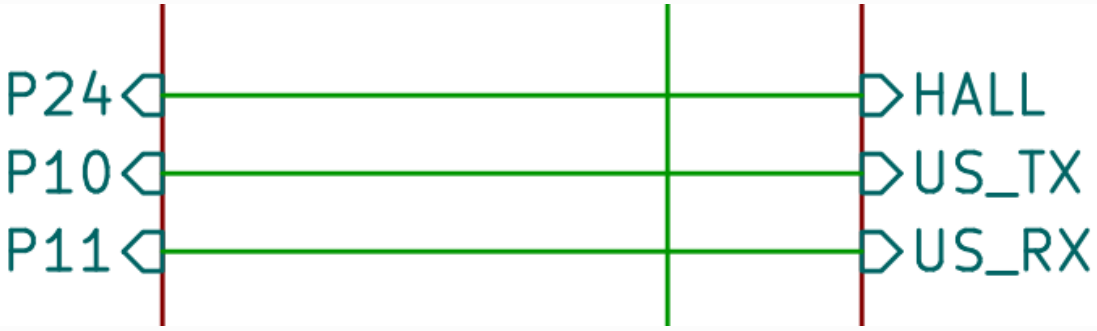
```
#include <STC15F2K60S2.H>

#include "stdio.h"
void Uart1_Init(void);
```

Ul超声波模版建立

ul.c

在这里我们需要注意，超声波官方是没有给底层的，我们需要自己手搓。我们通过查询原理图可以看到，超声波的T为P10，R为P11，所以我们需要定义一下

```
sbit Tx = P1 ^ 0;
sbit Rx = P1 ^ 1;
```

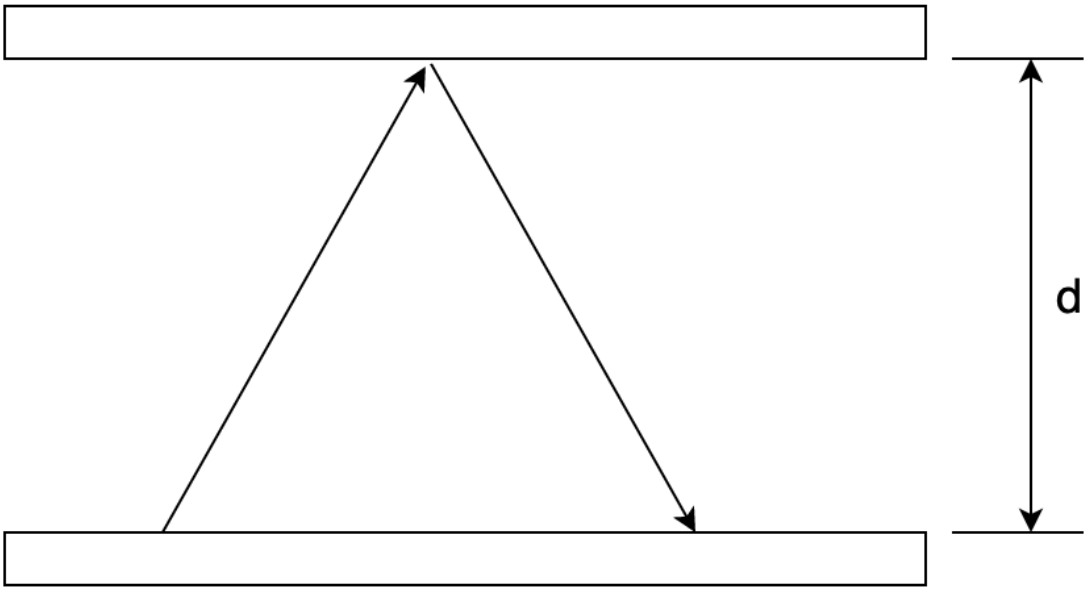


现在我们来进行逻辑的书写，我们首先要知道超声波测距的原理，我发送数据后，等待接收数据，那么我通过计算发送和接收中的间隔时间，就可以得到我们的距离，这里用数学公式可以进行计算，一般来说声速都是340m/s（也不排除题目里面和你说要对其进行变换啥的【比如第十四届国赛】，这里就按照声速为340m/s来进行说明）

$$d = \frac{vt}{2} = \frac{340m/s*t}{2} = 170m/s*t = 1.7*10^4cm/s*t$$

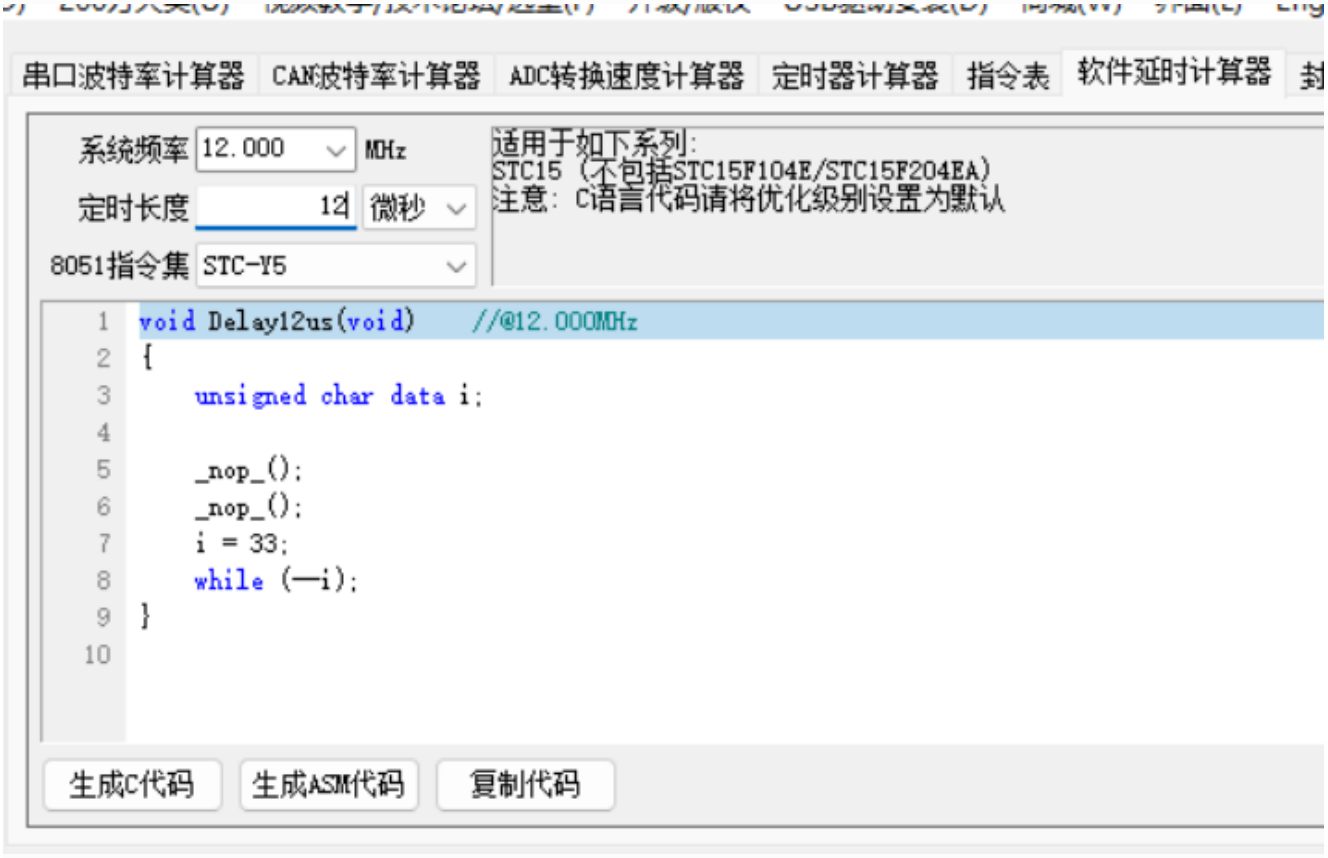
我们这里有一个单位换算需要进行注意，t的单位为us，也就是$10^{-6}s$，我们进行完整的单位换算就可以知道d的距离了（单位为cm），$d = 1.7*10^4*10^{-6}t = 0.017t$

图来自微信群友chmod-wrx绘制（我绘图太🖌🐔了）



根据我们上方的分析，我们可以知道我们首先需要进行发送，我们在这里发送8个40kHz的方波进行测量，那么周期就是$\frac{1}{40k}=25\cdot10^{-6}s=25us$，也就是我们需要$12us$将发送的电平进行反转

代码如下，我们首先使用isp生成一个12us的延时函数,这里需要注意的是要正确使用系统频率为12MHz；如果这里测量结果有问题，可以尝试将33修改为38或者其他的数据，因为软件延时不准。



```
void Delay12us(void)    //@12.000MHz
{
    unsigned char data i;

    _nop_();
    _nop_();
    i = 33;//可以修改为其他数据，比如38
    while (--i);
}
```

然后就是我们的超声波发送函数，发送8个4kHz的方波

```
void Ut_Wave_Init() {
  unsigned char i;
  for (i = 0; i < 8; i++) {
    Tx = 1;
    Delay12us();
    Tx = 0;
    Delay12us();
  }
}
```

最后是我们最重要的数据接收与转换函数的书写，在这里我们使用的pca定时器，可以节省定时器来进行其他的操作（因为ne555强制占用定时器0，串口考到就会占用一个定时器，还有一个定时器需要用来进行函数轮询，所以我们就必须用其他的定时器进行超声波的操作），其实PCA本质上还是一个定时器，所以他也存在一些定时器的参数，比如TLx→CL;THx→CH;TMOD->CMOD;TRx→CR;TFx→CF

我们首先需要对定时器的CH和CL清零，为了我们计数做准备；然后将CMOD配置为0x00，十六位不重载定时器。

```
CH = CL = 0;
CMOD = 0x00;
```

在我们发送超声波的时候，我们需要将总中断关闭，在我们发送完毕后就关掉，在我们发送完成后，就打开CR开始计时，等待我们收到回应（收到回应后Rx为0）或者计数溢出（溢出后CF为1），我们就关闭CR开始读取数据

```
EA = 0;
Ut_Wave_Init();
EA = 1;

CR = 1;
while (Rx && !CF);
CR = 0;
```

如果我们是因为收到回应而结束（那么就代表我们测量到了数据），那么我们就进行时间读取，如下所示，你们可能会好奇为什么+3，这是因为我们实际测量发现，+3后在远程数据（测量结果>10的时候）误差会比较小，而近程数据（测量数据<10）满足误差范围，我们也不可能测量得到0这个数据（题目中一般都不会测到0）（所以我们才把他视为测量错误的返回结果）

```
time = CH << 8 | CL;
return (0.017 * time + 3);
```

如果我们是因为计数溢出（没有收到返回的信号，代表测量错误），我们就直接返回为0，表示错误结果

```
return 0;
```

转换的完整代码如下

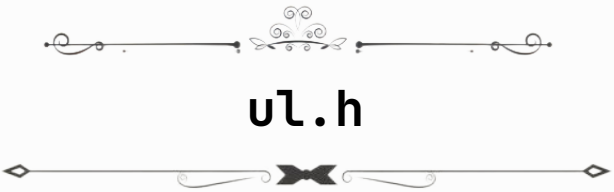
```
unsigned char Ut_Wave_Data() {
  unsigned int time;
  CH = CL = 0;
  CMOD = 0x00;

  EA = 0;
  Ut_Wave_Init();
  EA = 1;

  CR = 1;
  while (Rx && !CF);
  CR = 0;
  if (!CF) {  // us → s 10^(-6)
    // m → cm 10^2
    //  10^(-4)
    // L = V*T/2=340*time/2=170*10^(-4)*time=0.017*time
    time = CH << 8 | CL;
```

```
    return (0.017 * time + 3);
  } else {
    CF = 0;
    return 0;
  }
}
```

完整代码如下

```
#include "ul.h"

#include "intrins.h"
sbit Tx = P1 ^ 0;
sbit Rx = P1 ^ 1;
void Delay12us(void)  //@12.000MHz
{
  unsigned char data i;

  _nop_();
  _nop_();
  i = 33;  // 38
  while (--i);
}
void Ut_Wave_Init() {
  unsigned char i;
  for (i = 0; i < 8; i++) {
    Tx = 1;
    Delay12us();
    Tx = 0;
    Delay12us();
  }
}
unsigned char Ut_Wave_Data() {
  unsigned int time;
  CH = CL = 0;
  CMOD = 0x00;

  EA = 0;
  Ut_Wave_Init();
  EA = 1;

  CR = 1;
  while (Rx && !CF);
  CR = 0;
  if (!CF) {  // us -> s 10^(-6)
    // m -> cm 10^2
```

```
    //  10^(-4)
    // L = V*T/2=340*time/2=170*10^(-4)*time=0.017*time
    time = CH << 8 | CL;
    return (0.017 * time + 3);
  } else {
    CF = 0;
    return 0;
  }
}
```



ul.h

```
#include <STC15F2K60S2.H>
unsigned char Ut_Wave_Data();
```

init初始化模版建立

init.c

我们在启动开发板的时候最好对其进行初始化，关闭蜂鸣器和继电器，MOTOR和全部LED灯，避免上电情况有误

```c
#include "init.h"
void System_Init() {
  P0 = 0xff;
  P2 = P2 & 0x1f | 0x80;
  P2 &= 0x1f;

  P0 = 0x00;
  P2 = P2 & 0x1f | 0xa0;
  P2 &= 0x1f;
}
```


```c
#include <STC15F2K60S2.H>
void System_Init();
```

现在我们来进行主函数的编写，我也拆分为模块，但是最后还是会给一个整体的代码

这里我把全部的全局变量放上来

- 定义了Led的变量数组，用于控制8个灯的亮灭情况
- 定义了Seg_Pos的变量，用于控制数码管位选情况
- 定义了Seg_Buf的变量数组，用于控制每个数码管显示第数据
- 定义了Seg_Point的变量数组，用于控制某一位数码管是否显示小数点
- 定义了Uart_Buf的变量数组，用于获取串口接收的数据
- 定义了Uart_Rx_Index的变量，用于对串口接收后的数组进行索引
- 定义了Uart_flag的变量，用于判断当前是否正处于串口接收状态
- 定义了Sys_Tick的变量，用于嘀嗒计时，判断串口是否接收状态超过了10ms
- 定义了ucRtc的变量数组，用于存放时钟数据，时分秒
- 定义了int型的计时变量time_all_1s，用于全局定时
- 定义了int型的频率变量Freq，用于测频结果的存放

```
/* LED与数码管 */
unsigned char ucLed[8] = {0, 0, 0, 0, 0, 0, 0, 0};
unsigned char Seg_Pos;
unsigned char Seg_Buf[8] = {10, 10, 10, 10, 10, 10, 10, 10};
unsigned char Seg_Point[8] = {0, 0, 0, 0, 0, 0, 0, 0};

/* 串口数据*/
unsigned char Uart_Buf[10];
unsigned char Uart_Rx_Index;
bit Uart_flag;
unsigned char Sys_Tick;
/* 时间*/
unsigned char ucRtc[3] = {11, 11, 11};
unsigned int time_all_1s;

/* 数据 */
unsigned int Freq;//ne555测频
```

数据读取模块

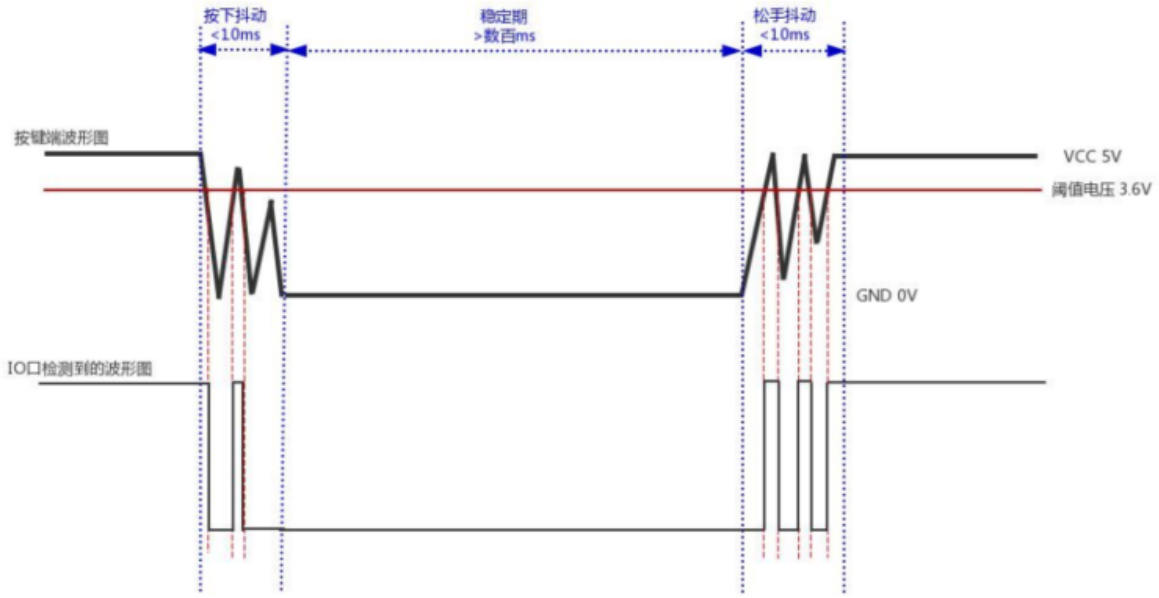
我们单独写一个模块对数据进行读取，方便我们进行不同数据的操作，这里使用%的操作来进行时间的定时，比如我们的时间读取，就每隔50ms（因为我定时器是1ms）进行一次；AD读取每隔100ms一次；温度读取每隔500ms一次（这里仅作参考，后续还是要自己搓一下试试）

```
void Data_Proc()
  if (time_all_1s % 50 == 0) {
    // 时间读取
  }
  if (time_all_1s % 100 == 0) {
    // AD读取
  }
  if (time_all_1s % 500 == 0) {
    // 温度读取
  }
}
```

Key键盘模块

这个模块是重中之重，因为用到了一个很精妙的三行代码消抖方法，这个在我以前的视频蓝桥杯单片机按键考点里面提到过，点击这行字体就可以跳转过去，我这里简单说明一下，我们的逻辑如下，Key_Val,Key_Old,Key_Down,Key_Up,这几个变量都是下面的逻辑，我们这里对其的原理进行解析

> 瞬时值获取→ 获取按下的值，获取抬起的值→更新一下值
>
> Down和Up都是瞬时值，Old比Val延后10ms，基于这个特性我们可以进行书写代码



对应的代码如下，我们将这四个变量设置为静态局部变量，可以有效节省内存空间，使用if语句那一段表明我们选择10ms跑一次代码（因为我们定时是1ms的）

```
void Key_Proc() {
  static unsigned char Key_Val, Key_Down, Key_Up, Key_Old;
  if (time_all_1s % 10) return;
  Key_Val = Key_Read();
  Key_Down = Key_Val & (Key_Old ^ Key_Val);
  Key_Up = ~Key_Val & (Key_Old ^ Key_Val);
  Key_Old = Key_Val;
}
```

如果理解不了上面的图和代码，我就拿一个🌰来进行验证，比如我们按下/抬起S4来做一下验证

键盘状态	Key_Old	Key_Val	Key_Old^Key_Val	Key_Down	Key_Up
未按下	0000	0000	0000^0000=0000	0000	0000
按下过程中(10ms)	0000	0100	0000^0100=0100	0100&0100=0100	1011&0100=0000
按下稳定(10ms后)	0100	0100	0100^0100=0000	0100&0000=0000	1011&0000=0000
抬起过程(10ms)	0100	0000	0100^0000=0100	0000&0100=0000	1111&0100=0100

我们可以从这里看出，当我们按下的时候，那一瞬间Down会是4（注意是一瞬间），其余时候都是0；同理Up也是一样，当我们抬起的瞬间Up会是4（注意是一瞬间），其余时候是0

那么我们如何在这个代码里面进行书写呢，其实很简单，我们一般判断按键都是判断的按键按下，即Down的数据，那么我们按照下面这样写就行，下面就是按下S4的一个示例

```
void Key_Proc() {
    static unsigned char Key_Val, Key_Down, Key_Up, Key_Old;
    if (time_all_1s % 10) return;
    Key_Val = Key_Read();
    Key_Down = Key_Val & (Key_Old ^ Key_Val);
    Key_Up = ~Key_Val & (Key_Old ^ Key_Val);
    Key_Old = Key_Val;
    if(Key_Down == 4)
        //执行按下S4的函数
}
```

Seg数码管模块

这个和V2模版不一样，他将数据读取放出去了，所以我们的数码管里面不过多涉及到数据读取，这里选择的是20ms进行一次数码管状态更新

```
void Seg_Proc() {
  if (time_all_1s % 20) return;
}
```

如何使用呢，其实也非常简单，我们只需要按照下方的写法即可（因为一般来说我们都有多个界面进行切换，我们定义一个变量Seg_Show_Mode来进行书写，如果有子界面的话，拿我们就再定义一个mode就行）

```
void Seg_Proc() {
    if (time_all_1s % 20) return;
    switch(Seg_Show_Mode)
    {
        case 0:
            //xx界面
            Seg_Buf[0] = xx;
            Seg_Buf[1] = xx;
            break;
        case 1:
            //xx界面
            break;
    }
}
```

Led灯模块

这里面我的话经常放的是Led，蜂鸣器，继电器，MOTOR口的相关逻辑，这里不需要设置定时进入

```
void Led_Proc() {}
```

使用示例如下

```
void Led_Proc() {
    ucLed[0] = 1;//L1点亮
    ucLed[1] = 0;//L2熄灭
    Relay(1);//打开继电器（L10亮）
    Beep(1);//蜂鸣器开启
    MOTOR(1);//MOTOR输出高电平
    Relay(0);//关闭继电器（L10灭）
    Beep(0);//蜂鸣器关闭
    MOTOR(0);//MOTOR输出低电平
}
```

Uart串口模块

我们在这里使用了串口超时解析，即10ms内没有收集到串口的信息的时候我们才进入并进行解析（后续在定时器里面会提到这个Sys_Tick），在我们解析结束后使用memset函数将接收的数组置为0（为了防止后续解析被干扰），并将索引置为0，为下一次接收解析进行准备

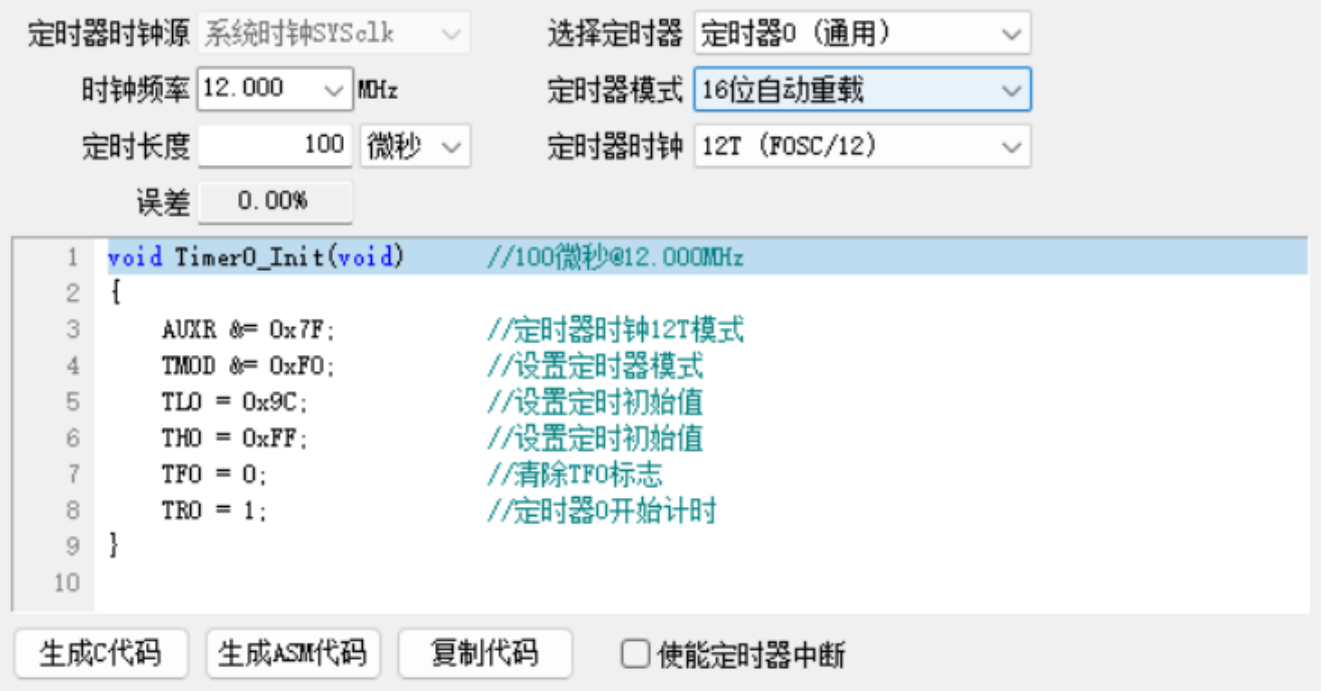
```
void Uart_Proc() {
  if (Uart_Rx_Index == 0) return;
  if (Sys_Tick ≥ 10) {
    Sys_Tick = Uart_flag = 0;

    memset(Uart_Buf, 0, Uart_Rx_Index);
    Uart_Rx_Index = 0;
  }
}
```

使用示例，这里发送小写的ok返回hello，注意一下，判断的时候用的是单引号而不是双引号

```
void Uart_Proc() {
  if (Uart_Rx_Index == 0) return;
  if (Sys_Tick ≥ 10) {
    Sys_Tick = Uart_flag = 0;
    if(Uart_Buf[0]=='o'&&Uart_Buf[1]=='k')
        printf("hello");
    memset(Uart_Buf, 0, Uart_Rx_Index);
    Uart_Rx_Index = 0;
  }
}
```

定时器0初始化（NE555模块）

这个模块是最有意思的，首先我们需要注意一下，P34和singal的跳线帽要插在一起，并且按键方面需要删除P34，然后才能开始测，这里依然用isp生成一下初始化的定时器0，但是这里要注意，新版的不能生成0ms，我们就随便生成个时间，然后按照我这样勾选后复制粘贴就行

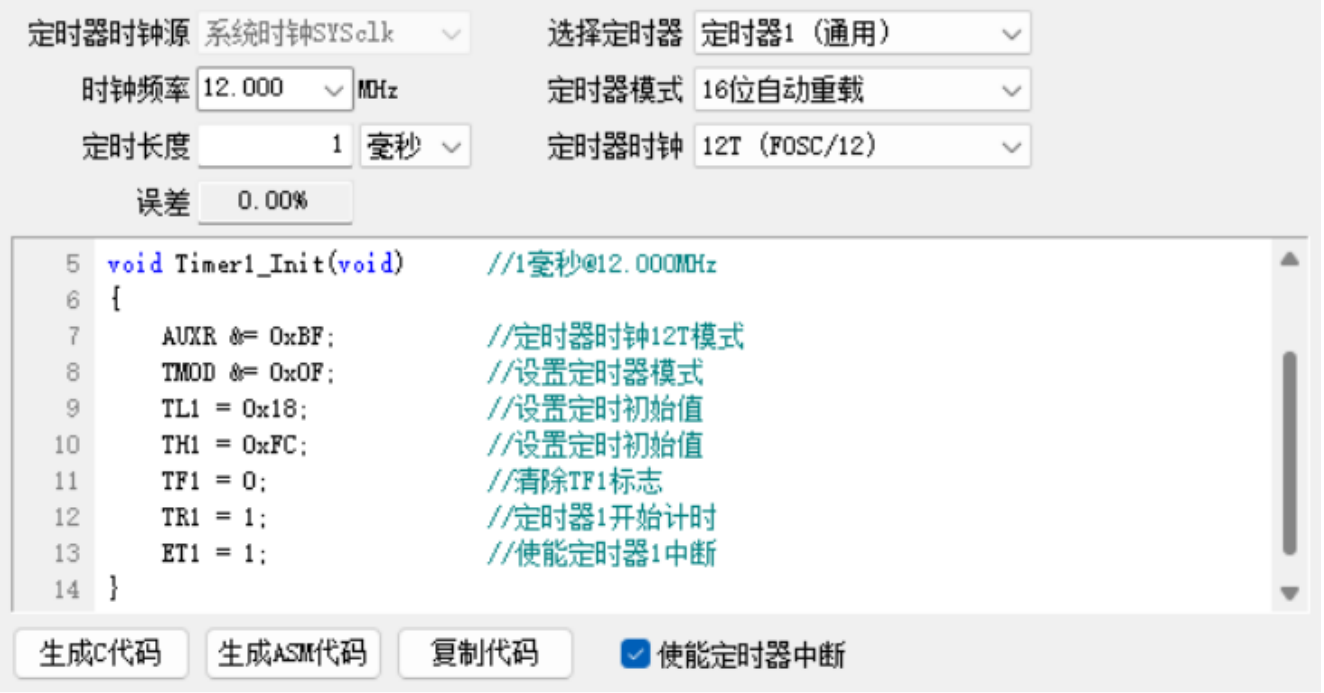


我们需要将TL0和TH0的值改为0，并且在TMOD &= 0xF0;下面对其进行配置 TMOD |= 0x05;将其变为十六位不自动重载（这里具体可以参考手册里面对于定时器的配置，我之前的视频有提到蓝桥杯单片机ne555模块，点击即可观看），这里别忘了EA开启总中断（这是我的习惯），频率的读取放在定时器1那里讲

```
void Timer0_Init(void)  // 1毫秒@12.000MHz
{
  AUXR &= 0x7F;  // 定时器时钟12T模式
  TMOD &= 0xF0;  // 设置定时器模式
  TMOD |= 0x05;
  TL0 = 0;  // 设置定时初始值
  TH0 = 0;  // 设置定时初始值
  TF0 = 0;  // 清除TF0标志
  TR0 = 1;  // 定时器0开始计时
  EA = 1;
}
```

定时器1初始化

这是我们轮询的定时器的初始化代码，我们使用isp生成1ms定时器1的代码（记得勾选上使能），老版本没有使能，所以记得加上ET1 = 1开启



当然，不要忘记EA = 1，

```c
void Timer1_Init(void)  // 1毫秒@12.000MHz
{
  AUXR &= 0xBF;  // 定时器时钟12T模式
  TMOD &= 0x0F;  // 设置定时器模式
  TL1 = 0x18;    // 设置定时初始值
  TH1 = 0xFC;    // 设置定时初始值
  TF1 = 0;       // 清除TF1标志
  TR1 = 1;       // 定时器1开始计时
  ET1 = 1;       // 使能定时器1中断
  EA = 1;
}
```

定时器1中断（重点）

我们在这里放了一堆东西，这里我做一下说明，我们每个中断都有对应的中断号（可以在官方的手册查到），这里定时器1的中断号是3

然后是我们的一个全局定时器（1s，注意time_all_1s的类型要是unsigned int，不然会溢出），在这个定时器里面我们对频率进行了读取，将TH0 << 8 | TL0的结果给了Freq（注意一下Freq的类型要是unsigned int，不然会溢出），在我们读取完数据后，将TH0和TL0进行了置0手动重载，方便下一次进行测量

接着就是一个和Sys_Tick相关的代码，当我们接收到数据的时候Uart_flag会置为1，然后将Sys_Tick置为0，也就是说我们一段时间不接受数据的话，那么Sys_Tick的值会一直累加，直到达到10ms（我们上面在Uart串口模块里面的设定），Uart_flag置为0，Sys_Tick的值置为0，为下一轮串口解析做准备

然后就是我们的一个位选的代码，因为我们数码管本质上是一位一位显示，所以我们需要写一个位选的变量来进行位选，这里使用的Seg_Pos = (++Seg_Pos) % 8将Seg_Pos这个变量在0~7中间进行循环

接着就是我们的数码管显示函数，这里不需要多说了，保持他不变就行，传入为位选、位选对应的段选显示、位选对应的小数点显示

最后就是我们的Led显示函数，这里我并没有选择和数码管同步，而是选择在中断里面写了个for循环来进行显示，因为它不涉及到扫描等操作，本质上就是可以直接一起显示。

```
void Timer1_Isr(void) interrupt 3 {
  unsigned char i;
  if (++time_all_1s == 1000) {
    time_all_1s = 0;
    Freq = TH0 << 8 | TL0;
    TH0 = TL0 = 0;
  }
  if (Uart_flag) Sys_Tick++;
  Seg_Pos = (++Seg_Pos) % 8;
  Seg_Disp(Seg_Pos, Seg_Buf[Seg_Pos], Seg_Point[Seg_Pos]);
  for (i = 0; i < 8; i++) Led_Disp(i, ucLed[i]);
}
```

串口中断（这里代码写死）

我们串口有一个接收中断，串口1的中断号查询后发现为4

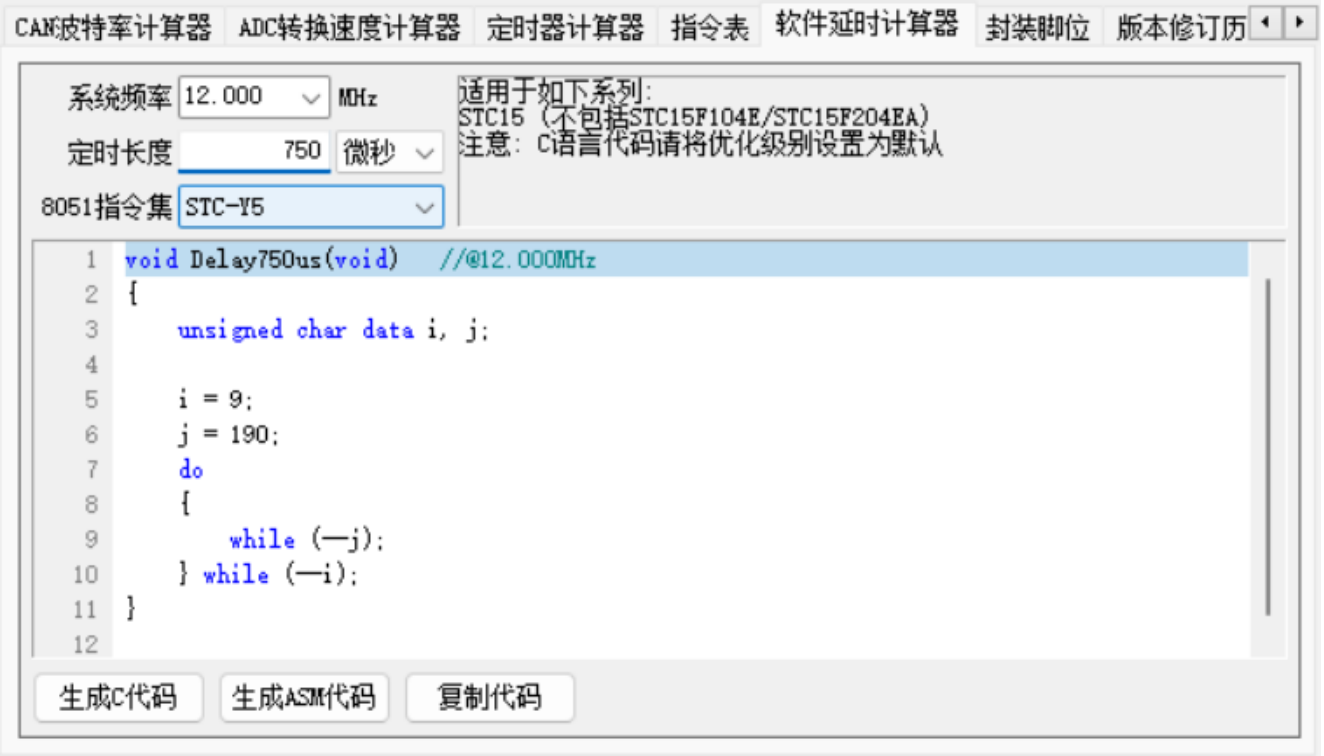
我们第一行就是判断是否RI为1，因为RI为1则代表有信息过来了，当我们收集到信息时，我们将Uart_flag置为1，Sys_Tick置为0，开始嘀嗒计时，当我们10ms没有接收到新的数据的时候，我们就认为数据接收完成，可以开始进行串口处理，后面的接收就不需要多讲了，我们是一位一位进行接收的，而读写都是用的SBUF这个缓冲区（串口1，如果是串口2则是S2BUF，但是串口2不会考，所以这里不用管），我们接收完成后，将RI置为0，为下一次接收做准备，最后一行的>10清零，则是为了避免出现传入数据溢出，导致程序异常，这里的10取决于你的字符串数组的位数，我在最上面定的是10位，所以这里写的10

```
void Uart1_Isr(void) interrupt 4 {
  if (RI) {
    Uart_flag = 1;
    Sys_Tick = 0;
    Uart_Buf[Uart_Rx_Index] = SBUF;
    Uart_Rx_Index++;
    RI = 0;
  }
  if (Uart_Rx_Index > 10) Uart_Rx_Index = 0;
}
```

main主函数书写

这是我们的一个最初的函数，没什么好说明的，就是需要进行一些初始化而已，但是顺序不要写错了，首先是系统初始化，然后是定时器0（ne555），最后是定时器1（轮询）的初始化。

你可以发现，我在这里进行了一次温度的空读取，因为温度上电后会显示85（温度转换的时间），如果在后面才给他开始温度转换的话，那么很有可能跳85，导致测量结果错误，这里的延时函数也是用isp直接生成的，注意系统时间别选错了就行



我们写完初始化之后就需要写一下循环了，51单片机里面必须将循环写死，所以我们用了while(1)，这里的顺序倒没什么特殊的要求，但是尽量按照我这个顺序就行，不会出问题的。

```
void main() {
    System_Init();
    Timer0_Init();
    Timer1_Init();
    Set_Rtc(ucRtc);
    rd_temperature();
    Delay750ms();
    while (1) {
        Data_Proc();
        Key_Proc();
        Seg_Proc();
        Uart_Proc();
        Led_Proc();
    }
}
```

完整的main.c

```
#include "main.h"
/* LED与数码管 */
```

```
unsigned char ucLed[8] = {0, 0, 0, 0, 0, 0, 0, 0};
unsigned char Seg_Pos;
unsigned char Seg_Buf[8] = {10, 10, 10, 10, 10, 10, 10, 10};
unsigned char Seg_Point[8] = {0, 0, 0, 0, 0, 0, 0, 0};

/* 串口数据*/
unsigned char Uart_Buf[10];
unsigned char Uart_Rx_Index;
bit Uart_flag;
unsigned char Sys_Tick;
/* 时间*/
unsigned char ucRtc[3] = {11, 11, 11};
unsigned int time_all_1s;

/* 数据 */
unsigned int Freq;

void Data_Proc() {
  if (time_all_1s % 50 == 0) {
    // 时间读取
  }
  if (time_all_1s % 100 == 0) {
    // AD读取
  }
  if (time_all_1s % 500 == 0) {
    // 温度读取
  }
}
/* 键盘处理*/
void Key_Proc() {
  static unsigned char Key_Val, Key_Down, Key_Up, Key_Old;
  if (time_all_1s % 10) return;
  Key_Val = Key_Read();
  Key_Down = Key_Val & (Key_Old ^ Key_Val);
  Key_Up = ~Key_Val & (Key_Old ^ Key_Val);
  Key_Old = Key_Val;
}

/* 数码管处理*/
void Seg_Proc() {
  if (time_all_1s % 20) return;
}
void Led_Proc() {}
void Uart_Proc() {
  if (Uart_Rx_Index == 0) return;
  if (Sys_Tick ⩾ 10) {
    Sys_Tick = Uart_flag = 0;
```

```
    memset(Uart_Buf, 0, Uart_Rx_Index);
    Uart_Rx_Index = 0;
  }
}
void Timer0_Init(void)  // 1毫秒@12.000MHz
{
  AUXR &= 0x7F;  // 定时器时钟12T模式
  TMOD &= 0xF0;  // 设置定时器模式
  TMOD |= 0x05;
  TL0 = 0;  // 设置定时初始值
  TH0 = 0;  // 设置定时初始值
  TF0 = 0;  // 清除TF0标志
  TR0 = 1;  // 定时器0开始计时
  EA = 1;
}

void Timer1_Init(void)  // 1毫秒@12.000MHz
{
  AUXR &= 0xBF;  // 定时器时钟12T模式
  TMOD &= 0x0F;  // 设置定时器模式
  TL1 = 0x18;    // 设置定时初始值
  TH1 = 0xFC;    // 设置定时初始值
  TF1 = 0;       // 清除TF1标志
  TR1 = 1;       // 定时器1开始计时
  ET1 = 1;       // 使能定时器1中断
  EA = 1;
}

void Timer1_Isr(void) interrupt 3 {
  unsigned char i;
  if (++time_all_1s == 1000) {
    time_all_1s = 0;
    Freq = TH0 << 8 | TL0;
    TH0 = TL0 = 0;
  }
  if (Uart_flag) Sys_Tick++;
  Seg_Pos = (++Seg_Pos) % 8;
  Seg_Disp(Seg_Pos, Seg_Buf[Seg_Pos], Seg_Point[Seg_Pos]);
  for (i = 0; i < 8; i++) Led_Disp(i, ucLed[i]);
}
void Uart1_Isr(void) interrupt 4 {
  if (RI) {
    Uart_flag = 1;
    Sys_Tick = 0;
    Uart_Buf[Uart_Rx_Index] = SBUF;
    Uart_Rx_Index++;
```

```
    RI = 0;
  }
  if (Uart_Rx_Index > 10) Uart_Rx_Index = 0;
}
void Delay750ms(void)  //@12.000MHz
{
  unsigned char data i, j, k;

  _nop_();
  _nop_();
  i = 35;
  j = 51;
  k = 182;
  do {
    do {
      while (--k);
    } while (--j);
  } while (--i);
}

void main() {
  System_Init();
  Timer0_Init();
  Timer1_Init();
  Set_Rtc(ucRtc);
  rd_temperature();
  Delay750ms();
  while (1) {
    Data_Proc();
    Key_Proc();
    Seg_Proc();
    Uart_Proc();
    Led_Proc();
  }
}
```

main.h

```
#include <STC15F2K60S2.H>
```

```
#include "ds1302.h"
#include "iic.h"
#include "init.h"
#include "intrins.h"
#include "key.h"
#include "led.h"
#include "onewire.h"
#include "seg.h"
#include "stdio.h"
#include "string.h"
#include "uart.h"
#include "ul.h"
```