模块默写

- 模块默写 还未加入寻址
  - 记忆点总结
    - ds18b20是两次init，单总线一次只能做一个动作DS18B20
    - PCF8591 EEPROM 写操作和读操作之间需要重新start iic

      每个发送字节的操作之后要等待Ack，读取操作之后要发送ack1
      无论读还是写，都要进行先写地址，读操作是先写再读，写操作是直接写，然后发数据一步到位
      - 读取数据操作简化流程
        - 发送要写入寄存器操作控制字节W
        - 发送要读取的地址，根据bus protocol来
        - start
        - 发送读取寄存器的操作R
        - receive
        - 根据bus protocol发一个1再stop
      - 写入数据操作简化流程
        - 发送要写入寄存器的操作来选择地址W
        - 写入地址
        - 要写入的数据
        - 结束，stop
    - ds1302

      一张表就够了
      - 设置时间要发地址和关闭写保护，然后写入。最后开启写保护。
      - 读取时间直接读
  - Seg

    先disp再trans
    注意是'1' '2'不是1 2
    译码器赋值完之后要清0
    - Seg_disp
      - 无返回值，输入是ucSeg_Code和ucSeg_pos
      - 主要是8个数码管加上两个38译码器
        - 38译码器的使能端由P2的高三位，P27 P26 P25控制
        - 消影，就是延续上次的显示
          - P0 = 0xff 数码管全部熄灭

            因为1熄灭，全1就是全熄灭

- P2使能y7c
  > 先清空高三位，再写数据
  > 写完数据后，继续清空高三位
- 110 00000->使能y6c进行位选（com号）
  - P0 传递位置信息
  - P2 使能y6c
- 111 00000->使能y7c进行段选（加引号的内容）
  - P0 传递编码信息
  - P2 使能y7c

- Seg_tran
  - 传入ucSeg_buf ucSeg_Code
  - 局部变量 temp i（对应buf任意长度） j（对应code，编码对应八个数码管）
  - switch判断本次的buf[i]，给出对应的temp
  - 在temp传给ucSeg_Code[j]之前，判断下一个buf是否为'.';如果是，则将temp&上0x7f(记住led和数码管都是0点亮)，对应的i要++
  - temp传给code[j]

- Key
  - Key_read()
    - 单独按键，一端直接接地，直接判断四个row的电平
    - 矩阵键盘 ，扫描，每次把三列的值输入1，一列输出0，这时候判断哪一行为0，是0的就是被按下为0
  - Key_proc()
    - 重点在三行按键
      - val是read返回值，按下状态就返回
      - old是上一次的val
      - down是val&(val^old)
        > 两次不一样，且本次按下，表示，下降沿
      - ~val&(val^old)但是只适用于用二进制表示的按键
        > 上升沿
    - 上升沿
      - 直接判断
        ```
        if(key_old == 7 && key_val == 0)key_up = 7;
          else key_up = 0;
        ```
    - 下降沿
      ```
      val&( val^old)
      ```
    - 长按

- 直接判断目前的val不是7不行，因为这样，每次都会进来（s7_record没清零，不过清零了也影响）

```
void Key_proc(){
    unsigned char key_val,key_down,key_up;
    static unsigned long s7_record;
    if(key_dly < key_delay) return;
    else key_dly = 0;
    key_val = key_read();
    key_down = key_val & (key_val ^ key_old);
    if(key_old == 7 && key_val == 0)key_up = 7;
    else key_up = 0;
    key_old = key_val;
    if(key_val == 7 && s7_record == 0) s7_record = ulms;
    if(ulms - s7_record > 1000){ 1 大于1s
        if(key_up == 7) count_relay = 0;
    }                          2 上升沿   直接判断目前的按键不是7也可以实现
    if(key_val != 7) s7_record = 0;
```

- 双击
  - 双击检测的代码

```
void Key_proc(){
    unsigned char key_val,key_down,key_up;
    static unsigned char key_down_old;
    static unsigned long double_record;
    static unsigned long s7_record;
    if(key_dly < key_delay) return;
    else key_dly = 0;
    key_val = key_read();
    key_down = key_val & (key_val ^ key_old);
    if(key_old == 7 && key_val == 0)key_up = 7;
    else key_up = 0;
    key_old = key_val;

    if(key_down){
        if(ulms - double_record < 250){
            if(key_down == key_down_old) ucled ^= 0xff;
        }
        key_down_old = key_down;
        double_record = ulms;
    }
```

- extra
  - 进入key_read ET1 = 0;出去P3=0XFF;ET1=1;关掉轮询定时器

    保持按键读取稳定
  - 高老师发的特殊按键

```
void Key_Proc(void)
{
    uint8_t ucKey_Down,ucKey_Up,ucKey_Val;
     //防抖
    if(uiKey_Dly) return; // 0-19

    ucKey_Val = Key_Read();
    ucKey_Down = ucKey_Val & (ucKey_Val ^ ucKey_Old);  //negedge
    ucKey_Up = ucKey_Val & (ucKey_Val ^ ucKey_Old);    //posedge
    ucKey_Old = ucKey_Val;

    if(ucKey_Down == 4) // s4
    {
//    if(++ucMenu == 3) ucMenu = 0;
      ucMenu = 0;
    }
//*******************************特殊的按键功能**************************
    //1.如果一个按键同时用到短按和长按功能，请使用松开按键（上升沿）来判断按键
    //以S13为例，同时有短按和长按1.5秒两种操作，首先通过ucKey_Down，记录按下按键的时刻
    //再通过ucKey_Up和松开按键时距离按下按键的时间差来判断是短按还是长按
    if(ucKey_Down == 13)
    {
       ulKey_Time = ulms;
    }
    if(ucKey_Up == 13 && (ulms-ulKey_Time)<1500) //短按 进入 界面2
    {
       ucMenu = 1;
    }
    if(ucKey_Up == 13 && (ulms-ulKey_Time)>=1500) //长按1.5s 进入界面3
    {
       ucMenu = 2;
    }
    //2.如果题目只是让你长按某个按键2s以上松开，实现类似清除数据的功能，因为有松开的要求，所以还是跟1一样方法
    //（1）以13届国赛为例，要求在湿度界面（界面1）下， 长按 S8 按键超过 1 秒后松开， 清零继电器开关次数统计
    if(ucKey_Down == 8 && ucMenu == 1)
    {
       ulKey_Time = ulms; //记录按下时间
    }
    if(ucKey_Up == 8 && (ulms-ulKey_Time)>=1000) //长按1s松开后 清零
    {
       //这里添加清零代码
       ucMenu = 2; //仅用做测试
    }
    //（2）14届省赛：时间回显子界面（界面1、子界面2）下，长按 S9 超过 2 秒后松开， 清除所有已记录的数据，触发次数重置为 0。
    if(ucKey_Down == 9 && ucMenu == 1 && ucSubMenu == 2)
    {
       ulKey_Time = ulms; //记录按下时间
    }
    if(ucKey_Up == 9 && (ulms-ulKey_Time)>=2000) //长按2s松开后 清零
    {
       //这里添加清零代码
       ucMenu = 2; //仅用做测试
    }
    //3.如果题目需要判断两个按键均处于按下状态，且状态持续一定时间后，执行某个操作
    //以14届国赛为例，要求：任意界面下，检测到S8\S9均处理按下状态，且状态持续超过2s，则回复到初始状态
    //首先要在底层驱动Key_Read()中，添加一行： case 0x0C00: Key_Vaule = 20;break;
    //此时，s8、s9同时被按下时返回键值20，然后参考单个按键的长按来实现
    //此处，未要求松开时回到初始状态，所以不需要用ucKey_Up，判断当前键值ucKey_Val；反之，如果要求松开后执行操作，此处就仍然用ucKey_Up
    if(ucKey_Down == 20)
    {
       ulKey_Time = ulms; //记录按下时间
    }
    if(ucKey_Val == 20 && (ulms-ulKey_Time)>=2000) //长按2s后 清零,没有松开，所以直接用当前键值ucKey_Val判断
    {
       //这里添加代码回到初始状态
       ucMenu = 2; //仅用做测试
    }

}
```

-

```
,
    //4.如果题目需要某个按键按下时显示界面1，松开时显示界面2，这时直接判断ucKey_Old值就可以了。
    //以13届省赛为例，要求在时间显示界面，若S17按键处于按下的状态，时间界面显示分、秒；松开S17按键则显示时、分
    //如果数据空间足够，我们可以通过一个标志位来决定显示什么
    if(ucMenu == 1)
    {
      if(ucKey_Old == 17 )
      {
         ucFlag |= 0x01; //显示分秒   1
      }
      else
         ucFlag &=~0x01; //显示时分
    }

}

void Seg_Proc(void)
{

    if(uiSeg_Dly) return;  //控制刷新时间 每500ms


    switch(ucMenu)
    {
      case 0:sprintf(pucSeg_Buf,"A      1");break;
      case 1:
             if(ucFlag&0x01)
                 sprintf(pucSeg_Buf,"U2 %02d-%02d",(unsigned int)pucRtc[1],(unsigned int)pucRtc[2]);
             else
                 sprintf(pucSeg_Buf,"U2 %02d-%02d",(unsigned int)pucRtc[0],(unsigned int)pucRtc[1]);
             break;
      case 2:sprintf(pucSeg_Buf,"H      3");break;
      default:break;
    }

    Seg_Tran(pucSeg_Buf,pucSeg_Code);
}
```

2

- Sys_init
  - 温度检测ds18b20模块，while读取，不是85之后再继续
  - EA全局中断
  - ET1定时器1中断
  - ES串口中断
  - ET0定时器1中断
  - IE2定时器2中断
- 中断的代码

- Timer
  - timer0

    NE555模块

    - stc-isp里面直接生成
    - 定时器0（通用）12mhz，12t，16位自动重载
    - `TMOD &= 0xf0` 和 `TMOD|=0x05` 十六位不自动重载
    - TL0 和 TH0全部变成0
    - 开启全局中断EA=1
    - 手册里面的定时器/计数器相关寄存器低四位是定时器0，2bit计数器，M0不自动重装

    寄存器TMOD各位的功能描述

    TMOD 地址：89H 复位值：00H

    不可位寻址

7	6	5	4	3	2	1	0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
定时器1				定时器0			

位	符号	功能
TMOD.7/	GATE	TMOD.7控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。
TMOD.3/	GATE	TMOD.3控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。
TMOD.6/	C/$\overline{T}$	TMOD.6控制定时器1用作定时器或计数器，清零则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T1/P3.5外部脉冲进行计数）
TMOD.2/	C/$\overline{T}$	TMOD.2控制定时器0用作定时器或计数器，清零则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T0/P3.4的外部脉冲进行计数）
TMOD.5/TMOD.4	M1、M0	定时器定时器/计数器1模式选择
	0 0	16位自动重装定时器，当溢出时将RL_TH1和RL_TL1存放的值自动重装入TH1和TL1中。
	0 1	16位不可重装载模式，TL1、TH1全用
	1 0	8位自动重装载定时器，当溢出时将TH1存放的值自动重装入TL1
	1 1	定时器/计数器1此时无效（停止计数）。
TMOD.1/TMOD.0	M1、M0	定时器/计数器0模式选择
	0 0	16位自动重装定时器，当溢出时将RL_TH0和RL_TL0存放的值自动重装入TH0和TL0中
	0 1	16位不可重装载模式，TL0、TH0全用
	1 0	8位自动重装载定时器，当溢出时将TH0存放的值自动重装入TL0
	1 1	不可屏蔽中断的16位自动重装定时器

  - timer1

    1ms

    - stc-isp里面直接生成
    - 12mhz，12t，16位自动重载
    - ET1=1，使能定时器1中断

      中断号为3
    - EA=1全局中断开启
  - time2
    - isp生成
- led和buzz&relay
  - led
    - 0点亮，所以要取反

        - ucled给P0
        - y4c
    - buzz and relay and motor

      注意与P0端口标号
        - P0先清零，只有两位用上
        - relay 05 buzz 07
        - y5c
        -

```
/*--------------------relay&buzz-------------------*/
void Relay_Buzz(bit relay,bit buzz){
    if(relay) P0 |= 0x10;
    if(buzz)  P0 |= 0x40;
    P2 = P2 & 0x1f | 0xa0;
    P2 = P2 & 0x1f;
}
```

- 38译码器

  对应P2的高三位，P27 P26 P25
    - y4c led
    - y5c buzz and relay
    - y6c com
    - y7c seg_code
- PCF8591

  scl sda的引脚要定义，还有delay_time（似乎不必要）
    - adc，接受模数转换结果

      写操作写的寄存器，选择读取对象
      1.写（地址000三个接地，末尾1），写入（模式，通道）
      2.读（同上，末尾0），receive
        - I2CStart()
        - 发送0x90,写操作，发送AddressByte(fig4)
        - 等待回应 I2Cwaitack()
        - 发送带有地址ControlByte(fig5)
        - 等待回应 I2Cwaitack()，等待转换完成
        - I2CStart()
        - 发送0x91,读操作，发送AddressByte(fig4)
        - 等待回应 I2Cwaitack()
        - temp 接收I2CReceiveByte()
        - 接收完成I2CSendAck(1);I2CStop();
    - dac，数模转换输出
        - iic start

- 先找表格的address byte，三个接地了，最后一位写W为0,0x90
- 等待应答
- 控制字control byte，annolog enable写1，别的不管，0x40
- 等待应答
- 发送dac的值，0-255,send函数
- iic stop

- 代码和原理图
  - 原理图
    - 0 注意这里，如果adc 与 dac同时使用要注意

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

    - 1

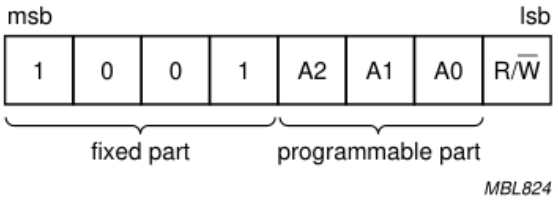
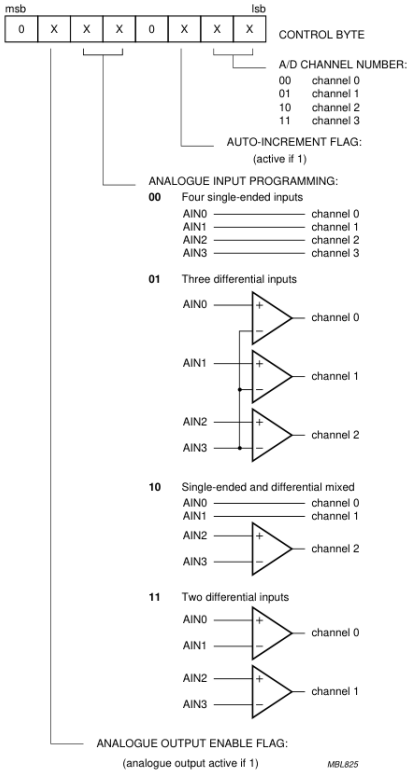


Fig.4 Address byte.

    - 2



- 3 dac与流程完全一致，但是adc，在该流程前要确定，单通道读取的通道，所以要0x90再发送通道。结束之后才是图中的start + 0x91 + a 接受，

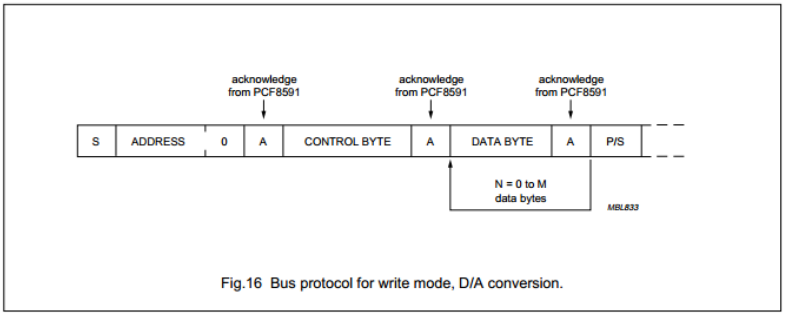


Fig.16 Bus protocol for write mode, D/A conversion.

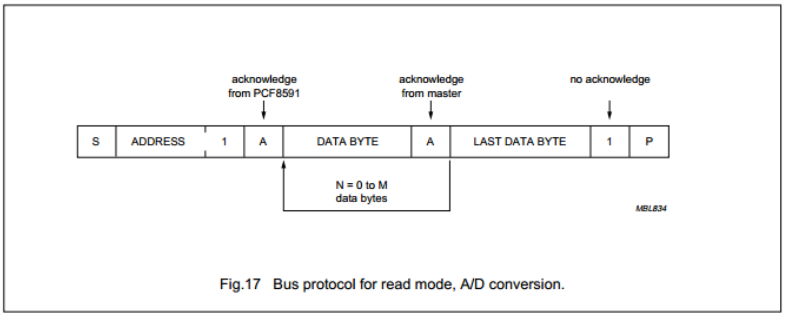


Fig.17 Bus protocol for read mode, A/D conversion.

- 代码
  - adc

```
unsigned char PCF8591_ADC(unsigned char adc_ch){
    unsigned char temp;
    I2CStart();
    I2CSendByte(0x90);//1001 0000
    I2CWaitAck();
    I2CSendByte(adc_ch);//0000 0001
    I2CWaitAck();

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

- dac

```
void PCF8591_DAC(unsigned char ucData){

    I2CStart();
    I2CSendByte(0x90);//address
    I2CWaitAck();

    I2CSendByte(0x40);//control byte
    I2CWaitAck();

    I2CSendByte(ucData);
    I2CWaitAck();

    I2CStop();

}
```

- EEPROM

  地址就是pcf8591的a0 a1 a2三个 0 0 0

  - 代码
    - 读 注意这里的if(num)

```c
//函数名：读EEPROM函数
//入口参数：读到的数据需要存储的字符串，读取的地址(务必为8的倍数)，读取的数量
//返回值：无
//函数功能：读取EERPOM的某个地址中的数据，并存放在字符串数组中。
void EEPROM_Read(unsigned char* EEPROM_String, unsigned char addr, unsigned char num)
{
    IIC_Start();//发送开启信号
    IIC_SendByte(0xA0);//选择EEPROM芯片，确定写的模式
    IIC_WaitAck();//等待EEPROM反馈

    IIC_SendByte(addr);//写入要读取的数据地址
    IIC_WaitAck();//等待EEPROM反馈

    IIC_Start();//发送开启信号
    IIC_SendByte(0xA1);//选择EEPROM芯片，确定读的模式
    IIC_WaitAck();//等待EEPROM反馈

    while(num--)
    {
        *EEPROM_String++ = IIC_RecByte();//将要写入的信息写入
        if(num) IIC_SendAck(0);//发送应答
            else IIC_SendAck(1);//不应答
    }

    IIC_Stop();//停止发送
}
```

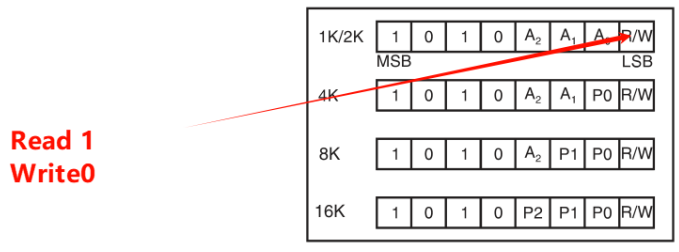
- 写 写操作结束一个stop，四个delay（255）

```c
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);
}
```

- 原理图
  - 1设备地址

Figure 7. Device Address



  - 2写

Figure 8. Byte Write

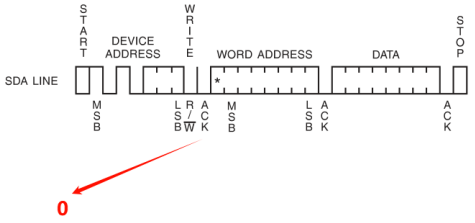
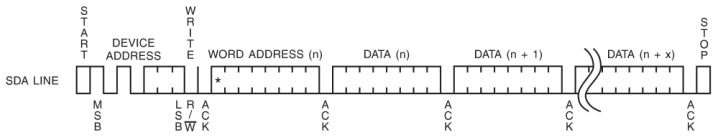


Figure 9. Page Write



(* = DON'T CARE bit for 1K)

- 3读

Figure 10. Current Address Read

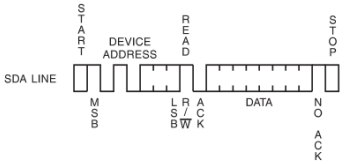
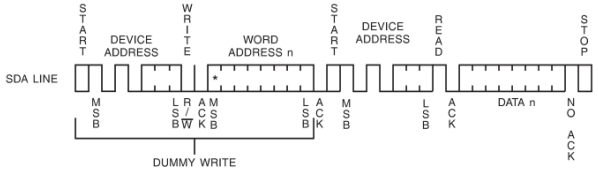
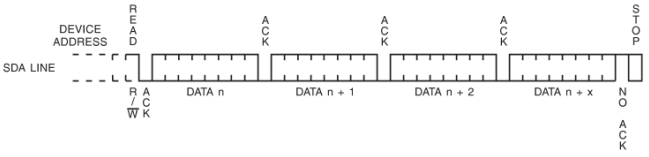


Figure 11. Random Read



(* = DON'T CARE bit for 1K)

Figure 12. Sequential Read



- DS1302

  只看table3

  - set
    - 写数据前要先关闭写保护，地址0x8e，最高位置0
    - 然后for循环，
      - 从数字转换为8位二进制
      - 从hour到Minute到Second，地址递减
    - 最后添加写保护
  - read
    - for循环
      - temp读取，地址从h到m到s递减
      - 8进制转换为十进制，位运算，或者使用*/16的方法进行转换

- 代码

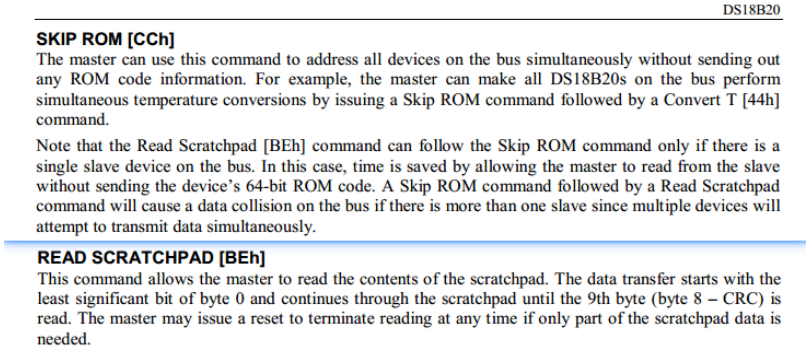
```
#include "STC15F2K60S2.H"
#include <intrins.h>
sbit SCK = P1^7;
sbit SDA = P2^3;
sbit RST = P1^3;
void Set_rtc(unsigned char * rtc){
    unsigned char i;
    unsigned char temp;
    Write_Ds1302_Byte(0x8e,0x00);
    for(i = 0;i < 3;i++)
    {
        temp = (rtc[i]/10 << 4) | (rtc[i] % 10);
        Write_Ds1302_Byte(0x84-2*i,temp);
    }
    Write_Ds1302_Byte(0x8e,0x80);
}
void Rd_rtc(unsigned char * rtc){
    unsigned char temp,i;
    for(i = 0;i < 3; i++){
        temp = Read_Ds1302_Byte(0x85 - 2*i);    nsigned
        rtc[i] = (temp>>4) * 10 + temp & 0x0f;
    }
}
```

- DS18B20

one-wire DQ = P1 ^ 4;

转换0x44和读取0xbe都在table3,skip rom(0xcc)在table3的前面

- init
- write 0xcc
- write 0x44
- init
- write 0xcc
- write 0xbe
- low = read,high = read
- return ((high << 8)|low)*0.0625 或者/16.0
- 代码和原理图
  - 原理图



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.

READ SCRATCHPAD [BEh]

This command allows the master to read the contents of the scratchpad. The data transfer starts with the least significant bit of byte 0 and continues through the scratchpad until the 9th byte (byte 8 – CRC) is read. The master may issue a reset to terminate reading at any time if only part of the scratchpad data is needed.

  - 代码

```c
float Read_temperature(){
    unsigned char low,high;
    init_ds18b20();
    Write_DS18B20(0xcc);
    Write_DS18B20(0x44);
    init_ds18b20();
    Write_DS18B20(0xcc);
    Write_DS18B20(0xbe);
    low = Read_DS18B20();
    high= Read_DS18B20();
    return ((high<<8) | low) /16.0;
}
```

- UART
  - 中断号4 使能ES=1
  - init部分

    isp里设置
    - 12.0Mhz
    - 8位
    - 12T 西风模版是1T是错的
    - 定时器2 16位自动装载
  - Send
    - putchar有返回值
    - putchar输入为char
    - ES=1开启中断

```c
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 &= 0xFB;       //定时器时钟12T模式
    T2L = 0xE6;         //设置定时初始值
    T2H = 0xFF;         //设置定时初始值
    AUXR |= 0x10;       //定时器2开始计时
    ES = 1;
}
extern char putchar(char dat){
    SBUF = dat;
    while(TI == 0);//wait until data sent
    TI = 0;//clear the send flag
    return dat;
}
```

```c
if(key_down == 4){
    printf(Seg_buf);
    printf("   %4.2fC",t_cel);
}
```

  - Receive
    - 定义u8数组（长度很重要，不够的会超过sys_tick，10ms,会被清零）和索引
    - 中断处理

> 51单片机**接受完1byte（8bit）后自动将RI置为1**，存储在SBUF中；同时，进入中断（RI和TI都会触发中断，但是在putchar中TI为1后结束while立即被置为0）
>
> - 判断RI为1
> - Uart_flag写1，然后将SBUF存储在Save数组中，索引超出清零

- Uart_proc（）

> 一旦Uart_flag为1，t1定时器中sys_tick，每毫秒递增

- 如果index还是0，还没接受数据，可以直接退出
- 如果sys_tick>10，清空所有接受有关的变量

> 超时处理

- memset来自于string.h，（数组，值，长度）

```
void Uart_Proc(void)
{
    if (Uart_R_index == 0)return;
    if (Sys_Tick >= 10)
    {
        Sys_Tick = Uart_Flag = 0;
        memset(Uart_R, 0, Uart_R_index);
        Uart_R_index = 0;
    }
}
```


- 全部代码
-

```c
extern char putchar(char dat){
    SBUF = dat;
    while(TI == 0);
    TI = 0;
    return dat;
}
void Uart_proc()
{
    if(save_index == 0) return;
    if(sys_tick > 10){
        Uart_flag =  sys_tick = 0;
        memset(Save,0,save_index);
        save_index = 0;
    }
}
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 &= 0xFB;        //定时器时钟12T模式
    T2L = 0xE6;          //设置定时初始值
    T2H = 0xFF;          //设置定时初始值
    AUXR |= 0x10;        //定时器2开始计时
    ES = 1;
}
void Uart_ISR(void) interrupt 4
{
    if(RI ==  1){
        Uart_flag = 1;
        sys_tick = 0;
        Save[save_index++] = SBUF;
        RI = 0;
    }
    if(save_index >= 10) save_index = 0;
}
void T1_ISR(void) interrupt 3{
    ulms++;
    if(Uart_flag) sys_tick++;
```

变量定义

```c
u8 Save[10],save_index,sys_tick;
bit Uart_flag;
```

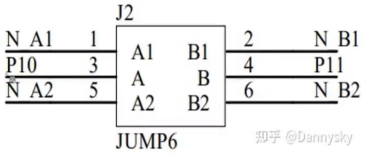
调用

```c
settime[0] = (Save[0]-48)*10 + Save[1]-48;
printf("%2d",(unsigned int)settime[0]);
```

- 接收数据转换示例
  - ‘0’的ascll码是48

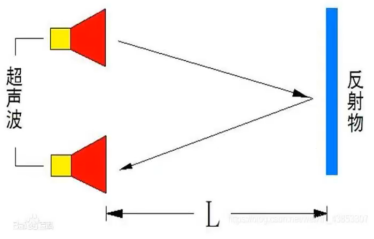
```c
settime[0] = (Save[0]-48)*10 + Save[1]-48;
printf("%2d",(unsigned int)settime[0]);
```

- UltraSound
  - 原理
    -



TX = P1^0;

RX = P1^1;



当TX = 1时为成功发送信号

当TX = 0时为发送信号失败

当RX = 0时为成功接受信号

当RX = 1时为接受信号失败

  - 代码

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

- 初始化

```
void Ut_Wave_Init() // 超声波初始化函数 产生8个40Mhz的方波信号
{
    unsigned char i;
    for (i = 0; i < 8; i++)
    {
        Tx = 1;
        Delay12us();
        Tx = 0;
        Delay12us();
    }
}
```

- 读取数据，定时器1，0.017记住

```
unsigned char Ut_Wave_Data() //超声波距离读取函数
{
  unsigned int time;//时间储存变量
  TMOD &= 0x0f;//配置定时器1计时模式
  TH1 = TL1 = 0;//复位计数值 等待超声波信号发出
  Ut_Wave_Init();//发送超声波信号
  TR1 = 1;//开始计时
  while((Rx == 1) && (TF1 == 0));//等待接受返回信号或者定时器溢出
  TR1 = 0;//停止计时
  if(TF1 == 0) //定时器没有溢出
  {
    time = TH1 << 8 | TL1;//读取当前时间
    return (time * 0.017);//返回距离值
  }
  else
  {
    TF1 = 0;//清除溢出标志位
    return 0;
  }
}
```

- 读取数据，PCA定时器，0.017记住

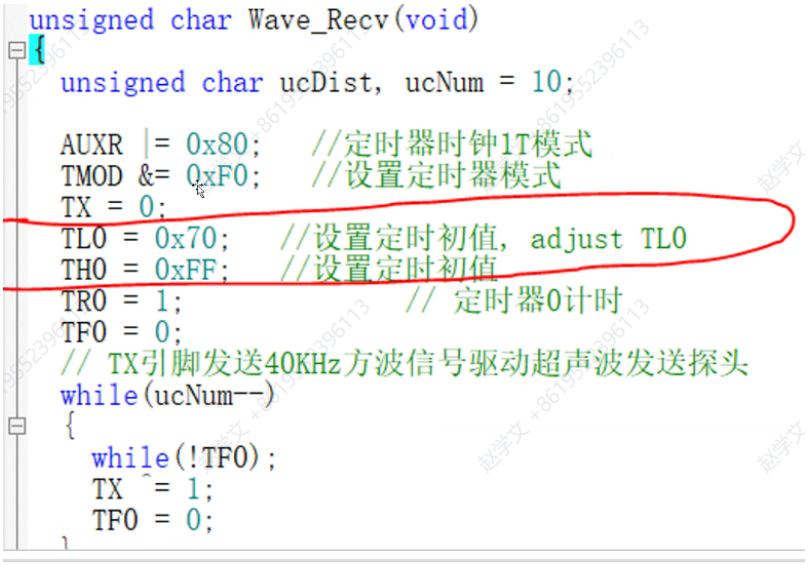
```
unsigned char Ut_Wave_Data() // 超声波距离读取函数
{
    unsigned int time; // 时间储存变量
    CMOD = 0x00;
    CH = CL = 0;    // 复位计数值 等待超声波信号发出
    Ut_Wave_Init(); // 发送超声波信号
    CR = 1;         // 开始计时
    while ((Rx == 1) && (CF == 0))
        ;        // 等待接受返回信号或者定时器溢出
    CR = 0;      // 停止计时
    if (CF == 0) // 定时器没有溢出
    {
        time = CH << 8 | CL;   // 读取当前时间
        return (time * 0.017); // 返回距离值
    }
    else
    {
        CF = 0; // 清除溢出标志位
        return 0;
    }
}
```

- 延时函数，定时器生成，用于产生方波

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

    _nop_();
    _nop_();
    i = 33;
    while (--i)
        ;
}
```

- 例外
  - 1T模式实现12us，发送1T接受12T，不准的情况下将70增加



```c
unsigned char Wave_Recv(void)
{
  unsigned char ucDist, ucNum = 10;

  AUXR |= 0x80;    //定时器时钟1T模式
  TMOD &= 0xF0;    //设置定时器模式
  TX = 0;
  TL0 = 0x70;    //设置定时初值, adjust TL0
  TH0 = 0xFF;    //设置定时初值
  TR0 = 1;                // 定时器0计时
  TF0 = 0;
  // TX引脚发送40KHz方波信号驱动超声波发送探头
  while(ucNum--)
  {
    while(!TF0);
    TX ^= 1;
    TF0 = 0;
  }
```

- PCA 具有定时器、PWM输出、补货等功能
  -

1. PCA工作模式寄存器CMOD

PCA工作模式寄存器的格式如下：

CMOD：PCA工作模式寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	name	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF

CIDL：空闲模式下是否停止PCA计数的控制位。
当CIDL=0时，空闲模式下PCA计数器继续工作；
当CIDL=1时，空闲模式下PCA计数器停止工作。

CPS2、CPS1、CPS0：PCA计数脉冲源选择控制位。PCA计数脉冲选择如下表所示。

CPS2	CPS1	CPS0	选择CCP/PCA/PWM时钟源输入
0	0	0	0，系统时钟，SYSclk/12
0	0	1	1，系统时钟，SYSclk/2
0	1	0	2，定时器0的溢出脉冲。由于定时器0可以工作在1T模式，所以可以达到计一个时钟就溢出，从而达到最高频率CPU工作时钟SYSclk。通过改变定时器0的溢出率，可以实现可调频率的PWM输出
0	1	1	3，ECI/P1.2(或P3.4或P2.4)脚输入的外部时钟(最大速率=SYSclk/2)
1	0	0	4，系统时钟，SYSclk
1	0	1	5，系统时钟/4，SYSclk/4
1	1	0	6，系统时钟/6，SYSclk/6
1	1	1	7，系统时钟/8，SYSclk/8

例如，CPS2/CPS1/CPS0 = 1/0/0时，CCP/PCA/PWM的时钟源是SYSclk，不用定时器0，PWM的频率为SYSclk/256

如果要用系统时钟/3来作为PCA的时钟源，应选择T0的溢出作为CCP/PCA/PWM的时钟源，此时应让T0工作在1T模式，计数3个脉冲即产生溢出。用T0的溢出可对系统时钟进行1～65536级分频(T0工作在16位重装载模式)。

ECF：PCA计数溢出中断使能位。
当ECF = 0时，禁止寄存器CCON中CF位的中断；
当ECF = 1时，允许寄存器CCON中CF位的中断。

- 零碎的点
  - 中断号在第六章中断系统6.5中断寄存器
    -

1. 中断允许寄存器IE、IE2和INT_CLKO

STC15F2K60S2系列单片机CPU对中断源的开放或屏蔽，每一个中断源是否被允许中断，是由内部的中断允许寄存器IE（IE为特殊功能寄存器，它的字节地址为A8H）控制的，其格式如下：

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位，EA=1，CPU开放中断，EA=0，CPU屏蔽所有的中断申请。EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制；其次还受各中断源自己的中断允许控制位控制。

ELVD：低压检测中断允许位，ELVD=1，允许低压检测中断，ELVD=0，禁止低压检测中断。

EADC：A/D转换中断允许位，EADC=1，允许A/D转换中断，EADC=0，禁止A/D转换中断。

ES：串行口1中断允许位，ES=1，允许串行口1中断，ES=0，禁止串行口1中断。

ET1：定时/计数器T1的溢出中断允许位，ET1=1，允许T1中断，ET1=0，禁止T1中断。

EX1：外部中断1中断允许位，EX1=1，允许外部中断1中断，EX1=0，禁止外部中断1中断。

ET0：T0的溢出中断允许位，ET0=1允许T0中断，ET0=0禁止T0中断。

EX0：外部中断0中断允许位，EX0=1允许中断，EX0=0禁止中断。

- PCA寄存器

11.1 与CCP/PWM/PCA应用有关的特殊功能寄存器

- 延迟函数的新写法
  - 新写法

```
#define key_delay 15
#define seg_delay 99
/*------------ variables ------------*/
}
void Timer1_Isr(void) interrupt 3
{
    ulms++;
    if(seg_dly < seg_delay) seg_dly++;
    if(++seg_pos==8) seg_pos=0;
    Seg_disp(seg_code,seg_pos);
}
unsigned long ulms;
```