# SD2400 实时时钟模块 FAQ

**1. Q：为什么 SD2400 走时不准？**

A：1. 看参照时间是不是准。可用我公司的网络校时工具“About Time”对电脑进行校时。

2. 内部电池电量过低导致时钟振荡电路停止工作（可测试 TEST 脚电压确认）。

3. 查看数字调整寄存器是否已清零。

4. 环境温度高低会影响走时精度。

**2. Q：为什么 SD2400 不返回 ACK，读出来的数值都是 0xFF？**

A：1. VDD 电压偏低，小于 3.3V；

2. SDA 端口没有加上拉电阻（一般是 2-10K）；

3. SCL 和 SDA 端口定义反了；

4. IO 口被复用；

5. SD2400 的器件地址没有设置正确，其器件地址应为 0x64；

6. I2C 总线上有其它的器件和 SD2400 的器件地址冲突；

7. I2C 的通信程序的问题（单片机端口配置、时序不规范、总线过快、IO 复用等）较多见，见附录；

8. RTC 已损坏。

**3. Q：为什么我用仿真器单步调试检测不到 RTC 返回 ACK 呢？**

A：因为 RTC 的 I2C 总线有 0.5s 超时自动复位的功能。

**4. Q：为什么从 SD2400 读出来时间有跳秒的现象。如 9 秒直接跳到 16 秒，而且会大于 60？**

A：RTC 采用 BCD 码计时，BCD 码的用 0x10 表示十进制的 10 秒，用十进制来看就是会看到从 9 跳到 16（0x10=16）。

**5. Q：为什么有时候我读到的时间是乱码？**

A：1. 内部的电池电量耗完，芯片已完全掉电，再上电时读到的时间是乱码；

2. 总速度太快或通信程序的时序中有不符合规范的地方（见附录）；

3. 控制 SCL、SDA 的 IO 口被复用；

4. 芯片的 SDA 端口没有加上拉电阻。

**6. Q：为什么小时会跑到 25 点、26 点……？**

A：没有正确设置 12/24 小时制。在小时寄存器（02H）的最高位设置 12/24 小时制。

**7. Q：为什么小时显示 90 点、91 点……？**

A：读出小时数据后没有把最高位（12/24 小时控制位）屏蔽掉。

**8. Q：时钟芯片产生中断输出的时候的 INT 脚是高电平还是低电平？**

A：INT 端口是一个 N 沟道开路输出口，使用时通常接一个上拉电阻。没有中断输出的时候是高电平，产生中断后变为低电平。

**9. Q：时钟芯片的 INT 脚在不用的时候可以悬空吗？**

A：可以的。

**10. Q：SD2401 和 SD2405 是带充电电池的 RTC，充满电能用多长时间，没电多久能充满？**

A：充满电后能用 6 个月左右，电量耗完的情况下 72 小时可以充满。

**11. Q：为什么 2 月 30 日也可以设置进去，芯片不能容错吗？**

A：RTC 芯片没有容错功能，请务必保证写入的时间。设定不存在的时间或日期数据将导致计数器不能正常计数。

**12. Q：我给 RTC 设定好年、月、日以后它可以自动调整星期吗？**

A：日历对应的星期系统不能自动调整，可以通过一定的算法来实现，下面介绍一种常用的公式：

A：最常见的公式：

W = [Y-1] + [(Y-1)/4] - [(Y-1)/100] + [(Y-1)/400] + D

Y 是年份数，D 是这一天在这一年中的累积天数，也就是这一天在这一年中是第几天。

B：最好用的是蔡勒公式：

W = [C/4] - 2C + y + [y/4] + [13 * (M+1) / 5] + d - 1

C 是世纪数减一，y 是年份后两位，M 是月份，d 是日数。1 月和 2 月要按上一年的 13 月和 14 月来算，这时 C 和 y 均按上一年取值。

两个公式中的[...]均指只取计算结果的整数部分。算出来的 W 除以 7， 余数是几就是星期几。如果余数是 0，则为星期日。

**13. Q：我怎么判断时钟芯片内置的电池快没电了？**

A：在芯片 VDD 不上电的时候测试芯片 TEST 脚的电压，若小于 TEST 脚上的电压 1.0V 表明内部电池电量低；对于 SD2405，TEST 脚电压小于 2.0V 表明内部电池电量低。

**14. Q：怎么从软件上判断 RTC 是否掉电了？**

A：读 0FH 寄存器，判断该寄存器的最低位（RTCF）是否为 1，若为 1 表明 RTC 掉电。

**15. Q：为什么可以读到 RTC 的时间，而不能写时间？**

A：RTC 内置有三个保护软锁（WRTC1~WRTC3），需要把这三个锁位置 1 后才可以对寄存器进行写操作。置 1 解锁的顺序为：WRTC1->WRTC2、WRTC3；当写寄存器操作完成后，把这三个锁位置 0 对寄存器数据进行保护。置 0 上锁的顺序为 WRTC2、WRTC3-> WRTC1。具体可以参考相关官网的 Demo 例程。

**16. Q：RTC 出厂时 TEST 脚的电压是多少？**

A：SD2401：1.75V≥Vtest≥1.18V

SD2400、SD2404：1.75V≥Vtest≥1.20V

SD2405：3.0V≥Vtest≥2.30V

**17. Q：内置可充电电池的 RTC 充不进电是什么原因？**

A：检查 Vout 脚是否有接地。Vout 脚接地是无法完成充电的，请将此脚悬空。

**18. Q：SD2400 可以过波峰焊和回流焊吗？**

A：可以过波峰焊，但是不能过回流焊。

**19. Q：SD2400 可以使用超声波清洗吗？**

A：不建议使用超声波对时钟芯片清洗。超声波洗净有时会使晶振受到共振破坏。因不能特别确使用条件（洗净机的种类、电源、时间、槽内状态等），因而不能保证超音波洗净的效果。

# 附录：程序问题举例

**1. 在切换端口方向的时候引入了 STOP 信号。**

```
/**********读取ACK信号**********/
uchar I2CWaitAck(void)  //返回为:1=有ACK,0=无ACK
{
    unsigned char ssda;
    DDRA|=0x08;                    //PA3为时钟，设置为输出
    DDRA&=0xFB;                    //设置数据口PA2(SDA)为输入
    SDA1_SET;
    SCL1_SET;
    I2CWait();
    if ( PINA&0x04 )       ssda=1;
    else                   ssda=0;
    SCL1_CLR;
    return(ssda);
}
```

这个函数开始便设置 IO 口的输入输出，本没有错。SCL 在此操作之前是低电平，但是一些单片机把 SCL 端口置为输出的时候会使 SCL 变成高电平，那么问题来了，把 SDA 置为输入的时候 SDA 也会由低电平变成高电平，这就形成了一个 STOP 信号了，总线立刻被释放掉了，之前发送的数据也清零了，当然就等不到 ACK 回应了。

SCL 线只有一个方向，一直都是输出，只要在 START 函数里设置一次就可以了。修改的方法是把 DDRA|=0x08 这条语句屏蔽掉就好了。

2. 发送 STOP 信号的时候引入 START 也可能会是 I2C 从器件工作不正常。

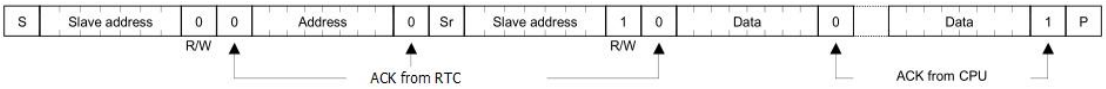
```
/********关闭I2C总线*******/
void I2CStop(void)
{
    SDA1_CLR;              //将数据端口（SDA）设为低
    SCL1_CLR;
    I2CWait();
    SCL1_SET;               //将时钟端口（SCL）设为高
    I2CWait();
    SDA1_SET;              //将数据端口（SDA）设为高
}
```

上面发送 STOP 信号的这个函数，看上去没什么错误，但是如果在调用这个函数之前 SCL 线的状态是高电平，那在这个函数开始调用的时候便会产生一个 START 信号。比较保险的做法是先拉低 SCL，再拉低 SDA，把 SDA1_CLR 和 SCL1_CLR 调换一下位置就可以了。

**3. 对 I2C 协议总线协议不熟悉导致读指定地址寄存器失败。**

```
/******I2C读一个字节******/
uchar I2CReadOneByte(uchar add)
{
  uchar dat;
  I2CStart();
  I2CSendByte(0x65);//器件地址_读
  if(!I2CWaitAck()){I2CStop(); return false;}
  I2CSendByte(add);   //设置要读的地址
  I2CWaitAck();
  dat=I2CReceiveByte();   //读数据
  I2CNoAck();
  I2CStop();
  return  dat;
}
```

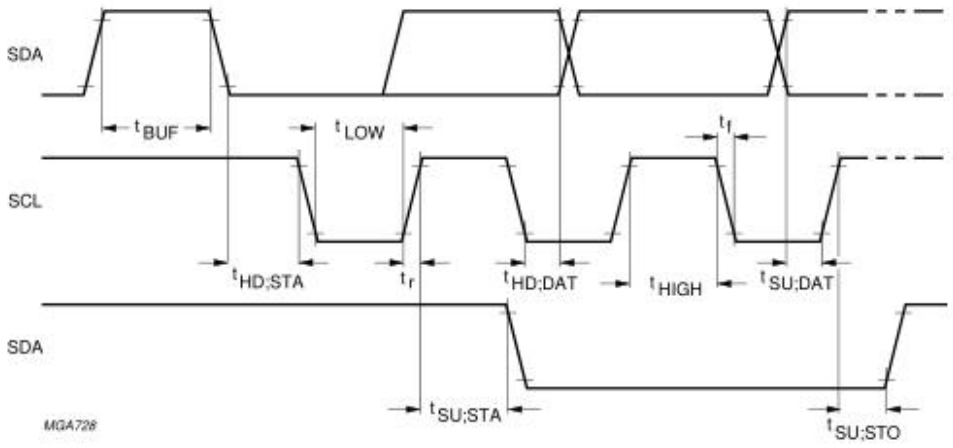
上面的程序看起来好像也没有问题，但实际上是没有了解 I2C 的传输协议，想当然的认为发送了读命令以后再发送一个地址就可以读到指定寄存器的数据了。正确的时序图如下：




寄存器的地址不是用读命令发送出去的，而是用写命令发送出去的。正确的写法如下：

```
/******I2C读一个字节******/
uchar I2CReadOneByte(uchar add)
{
  uchar dat;
  I2CStart();
  I2CSendByte(0x64);//器件地址_写
  if(!I2CWaitAck()){I2CStop(); return false;}
  I2CSendByte(add);   //设置要读的地址
  I2CWaitAck();
  I2CStart();
  I2CSendByte(0x65);//器件地址_读
  I2CWaitAck();
  dat=I2CReceiveByte();   //读数据
  I2CNoAck();
  I2CStop();
  return  dat;
}
```

**4. 启动信号保持时间、数据建立或数据保持的时间太短等，可能会导致总线不能响应。**



| 符号 | 参数 | 最小值 | 典型值 | 最大值 | 单位 |
|---|---|---|---|---|---|
| $t_{HD:STA}$ | 启动信号保持时间 | 0.6 | - | - | μs |
| $t_{SU:STA}$ | 启动信号建立时间 | 0.6 | - | - | μs |
| $t_{LOW}$ | SCL 低电平时间 | 1.3 | - | - | μs |
| $t_{HIGH}$ | SCL 高电平时间 | 0.6 | - | - | μs |
| $t_{BUF}$ | 停止到启动信号之间总线空闲时间 | 1.3 | - | - | μs |
| $t_r$ | 上升时间 | - | - | 0.3 | μs |
| $t_f$ | 下降时间 | - | - | 0.3 | μs |
| $t_{SU:DAT}$ | 数据建立时间 | 100 | - | | μs |
| $t_{HD:DAT}$ | 数据保持时间 | 0 | - | 900 | μs |
| $t_{SU:STO}$ | 停止信号建立时间 | 0.6 | - | - | μs |

```
/*************MCU向RTC发送一个字节**************/
void I2CSendByte(uchar demand) //数据从高位到低位//
{
  uchar i=8;
    while(i--)
    {
      p_Clock_CLK=0;
      _nop_();
      p_Clock_SDA=(bit)(demand&0x80);
      demand<<=1;

      p_Clock_CLK=1;
      I2CWait();
    }
    p_Clock_CLK=0;
}
```

上面这个程序的问题就在于 SDA 数据建立的时间太短。对一些处理速度快的 MCU，数据是发送出去了，RTC 是响应不到的。正确的做法是在 p_Clock_CLK=1; 的前面加一个延时：

```
/*************MCU向RTC发送一个字节**************/
void I2CSendByte(uchar demand) //数据从高位到低位//
{
  uchar i=8;
    while(i--)
    {
      p_Clock_CLK=0;
      _nop_();
      p_Clock_SDA=(bit)(demand&0x80);
      demand<<=1;
      I2CWait();
      p_Clock_CLK=1;
      I2CWait();
    }
    p_Clock_CLK=0;
}
```

**5. 器件地址错误导致读写寄存器失败。**

器件代码：

| BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | R/W |

```
#define   RTC_Address     0x65//RTC器件地址

/******I2C读一个字节程序******/
uchar I2CReadOneByte(uchar DeviceAddress,uchar add)
{
  uchar dat;
  I2CStart();
  I2CSendByte(DeviceAddress);
  if(!I2CWaitAck()){I2CStop(); return false;}
    I2CSendByte(add);   //设置要读的地址
  I2CWaitAck();
  I2CStart();
  I2CSendByte(DeviceAddress+1);
  I2CWaitAck();
  dat=I2CReceiveByte();   //读数据
  I2CNoAck();
  I2CStop();
  return  dat;
}
```

上面的程序很显然是对器件地址理解上的错误和 I2C 总线操作流程不熟悉，没有搞清楚器件地址的读写操作。例如，需要读 10H 寄存器的数据时：

I2CReadOneByte(RTC_Address, 0x10);

由于宏定义里面 RTC_Address 最后一个 bit 读写位已经被置为 1，当程序运行到 I2CSendByte(DeviceAddress+1)的时候，实际发送出去的数据就发送了 I2CSendByte(0x65+1)，把 0x66 发送出去了，RTC 芯片不认识发送过来的 0x66 地址，不做出响应，自然就不会读到数据了。