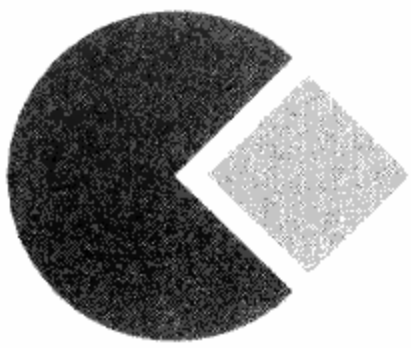

第 12 章　I²C 接口

本章目标

- 了解 I²C 总线协议
- 掌握 S3C2410/S3C2440 中 I²C 接口的使用方法

12.1　I²C 总线协议及硬件介绍

12.1.1　I²C 总线协议

1．I²C 总线的概念

I²C（Inter-Integrated Circuit，又称 IIC）总线是一种由 PHILIPS 公司开发的串行总线，用于连接微控制器及其外围设备，它具有如下特点。

- 只有两条总线线路：一条串行数据线（SDA），一条串行时钟线（SCL）。
- 每个连接到总线的器件都可以使用软件根据它的惟一的地址来识别。
- 传输数据的设备间是简单的主/从关系。
- 主机可以用作主机发送器或主机接收器。
- 它是一个真正的多主机总线，两个或多个主机同时发起数据传输时，可以通过冲突检测和仲裁来防止数据被破坏。
- 串行的 8 位双向数据传输，位速率在标准模式下可达 100kbit/s，在快速模式下可达 400kbit/s，在高速模式下可达 3.4Mbit/s。
- 片上的滤波器可以增加抗干扰功能，保证数据的完整。
- 连接到同一总线上的 IC 数量只受到总线的最大电容 400pF 的限制。

图 12.1 是一条 I²C 总线上多个设备相连的例子。

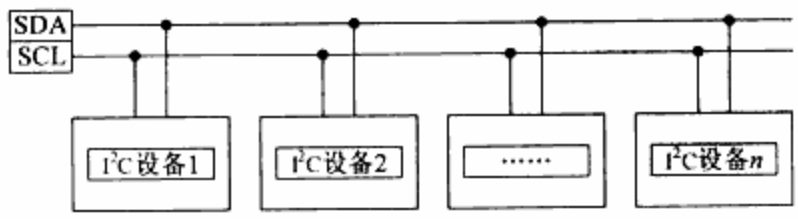

图 12.1　I²C 总线设备互连

先说明一些术语，如表 12.1 所示。

表 12.1　I²C 总线术语的定义

术　语	描　述
发送器	发送数据到总线的器件
接收器	从总线接收数据的器件
主机	发起/停止数据传输、提供时钟信号的器件
从机	被主机寻址的器件
多主机	可以有多个主机试图去控制总线，但是不会破坏数据
仲裁	当多个主机试图去控制总线时，通过仲裁可以使得只有一个主机获得总线控制权，并且它传输的信息不被破坏
同步	多个器件同步时钟信号的过程

2. I²C 总线的信号类型

I²C 总线在传送数据过程中共有 3 种类型信号：开始信号、结束信号和响应信号。

（1）开始信号（S）：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。

（2）结束信号（P）：SCL 为低电平时，SDA 由低电平向高电平跳变，结束传送数据。

（3）响应信号（ACK）：接收器在接收到 8 位数据后，在第 9 个时钟周期，拉低 SDA 电平。

它们的波形如图 12.2、12.3 所示。

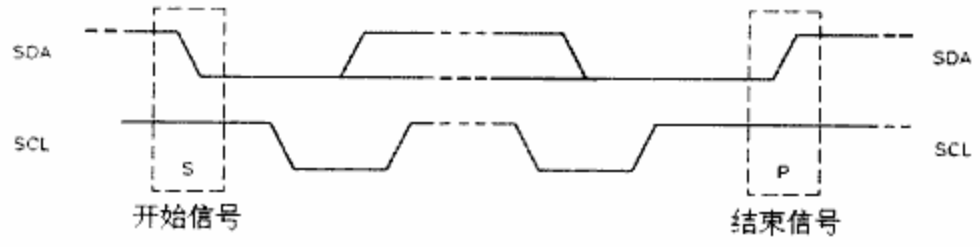

图 12.2　开始信号（S）和结束信号（P）

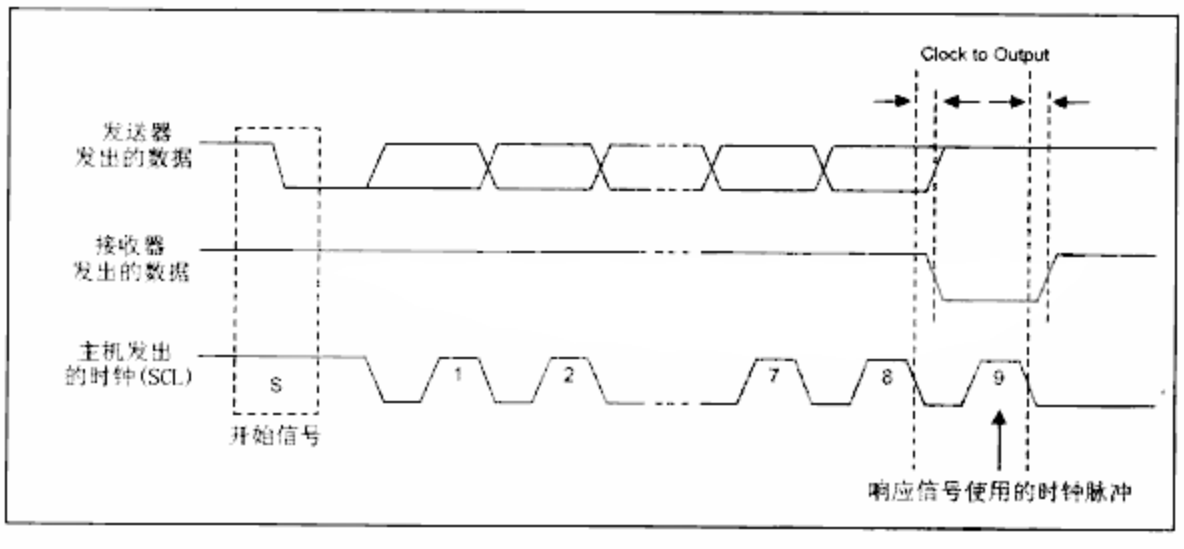

图 12.3　应答信号（ACK）

SDA 上传输的数据必须在 SCL 为高电平期间保持稳定，SDA 上的数据只能在 SCL 为低电平期间变化，如图 12.4 所示。

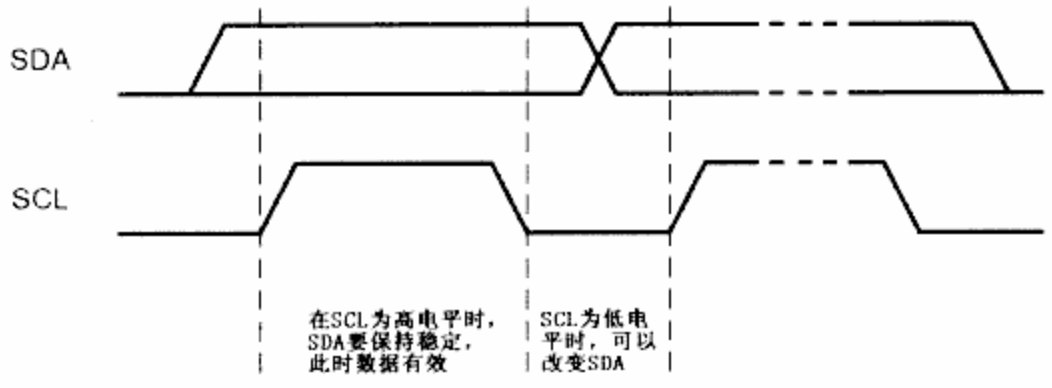

图 12.4 I²C 总线的位传输

3. I²C 总线的数据传输格式

发送到 SDA 线上的每个字节必须是 8 位的，每次传输可以发送的字节数量不受限制。每个字节后必须跟一个响应位。首先传输的是数据的最高位（MSB）。如果从机要完成一些其他功能后（例如一个内部中断服务程序）才能继续接收或发送下一个字节，从机可以拉低 SCL 迫使主机进入等待状态。当从机准备好接收下一个数据并释放 SCL 后，数据传输继续。如果主机在传输数据期间也需要完成一些其他功能（例如一个内部中断服务程序）也可以拉低 SCL 以占住总线。

启动一个传输时，主机先发出 S 信号，然后发出 8 位数据。这 8 位数据中前 7 位为从机的地址，第 8 位表示传输的方向（0 表示写操作，1 表示读操作）。被选中的从机发出响应信号。紧接着传输一系列字节及其响应位。最后，主机发出 P 信号结束本次传输。

图 12.5 是几种 I²C 总线上数据传输的格式。

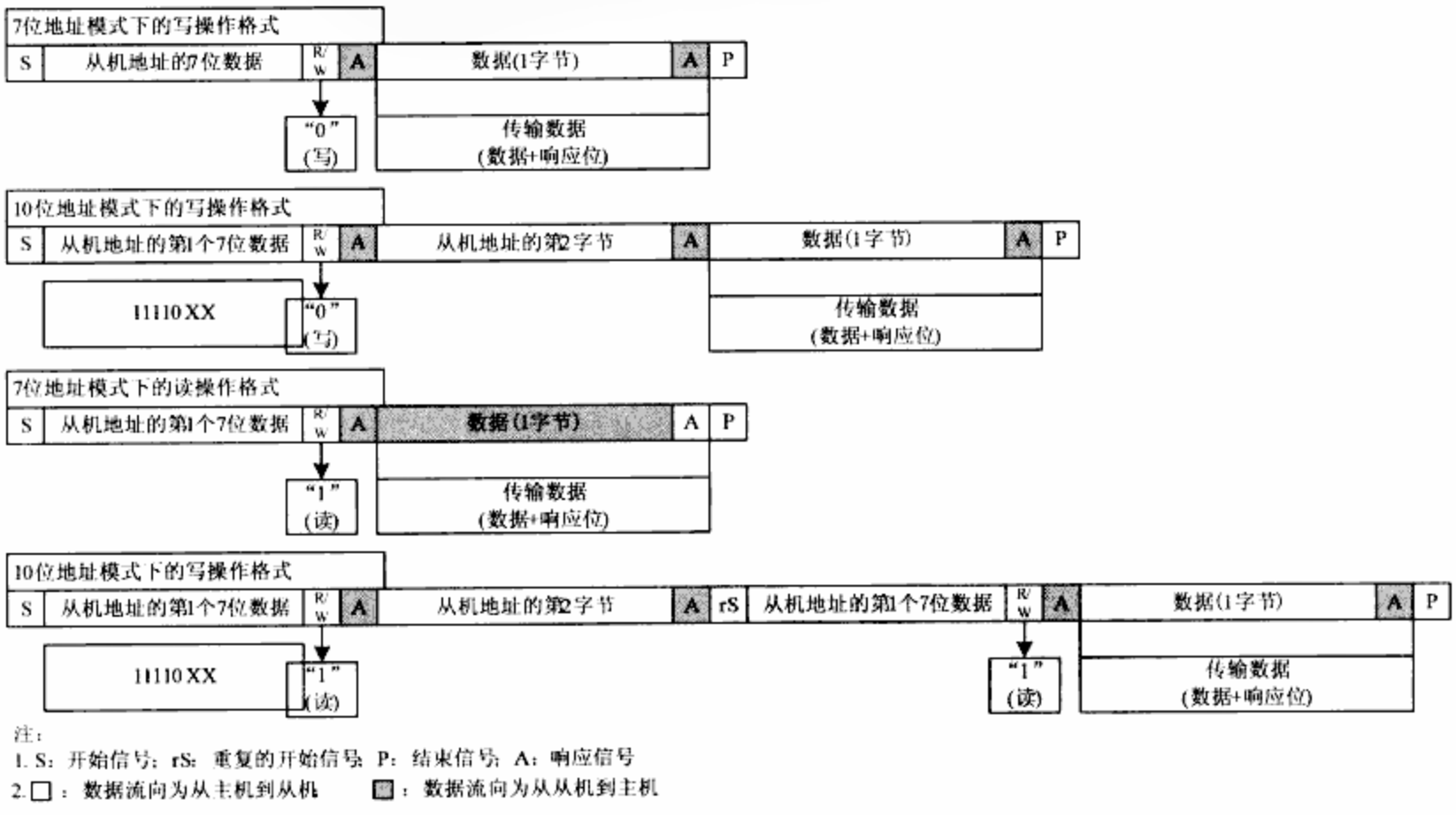

图 12.5 I²C 总线上数据传输的格式

并非每传输 8 位数据之后，都会有 ACK 信号，有以下 3 种例外。

（1）当从机不能响应从机地址时（例如它正忙于其他事而无法响应 I²C 总线的操作，或者这个地址没有对应的从机），在第 9 个 SCL 周期内 SDA 线没有被拉低，即没有 ACK 信号。这时，主机发出一个 P 信号终止传输或者重新发出一个 S 信号开始新的传输。

（2）如果从机接收器在传输过程中不能接收更多的数据时，它也不会发出 ACK 信号。这样，主机就可以意识到这点，从而发出一个 P 信号终止传输或者重新发出一个 S 信号开始新的传输。

（3）主机接收器在接收到最后一个字节后，也不会发出 ACK 信号。于是，从机发送器释放 SDA 线，以允许主机发出 P 信号结束传输。

12.1.2　S3C2410/S3C2440 I²C 总线控制器

1. S3C2410/S3C2440 I²C 总线控制器寄存器介绍

S3C2410/S3C2440 的 I²C 接口有 4 种工作模式：主机发送器、主机接收器、从机发送器、从机接收器。其内部结构如图 12.6 所示。

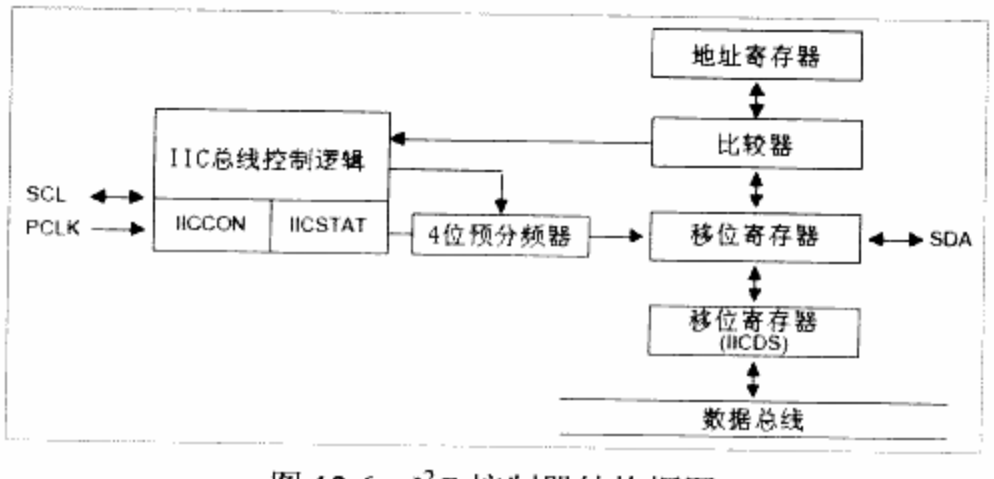

图 12.6　I²C 控制器结构框图

从图 12.6 可知，S3C2410/S3C2440 提供 4 个寄存器来完成所有的 I²C 操作。SDA 线上的数据从 IICDS 寄存器发出，或传入 IICDS 寄存器中；IICADD 寄存器中保存 S3C2410/S3C2440 当作从机时的地址；IICCON、IICSTAT 两个寄存器用来控制或标识各种状态，比如选择工作模式，发出 S 信号、P 信号，决定是否发出 ACK 信号，检测是否收到 ACK 信号。各寄存器的用法如下。

（1）IICCON 寄存器（Multi-master IIC-bus control）。

IICCON 寄存器用于控制是否发出 ACK 信号、设置发送器的时钟、开启 I²C 中断，并标识中断是否发生。它的各位含义如表 12.2 所示。

表 12.2　IICCON 寄存器的格式

功　能	位	说　明
ACK 信号使能	[7]	0 = 禁止，1 = 使能 在发送模式，此位无意义； 在接收模式，此位使能时，SDA 线在响应周期内将被拉低，即发出 ACK 信号
发送模式时钟源选择	[6]	0 = IICCLK 为 PCLK/16，1 = IICCLK 为 PCLK/512
发送/接收中断使能	[5]	0 = I²C 总线 Tx/Rx 中断禁止， 1 = I²C 总线 Tx/Rx 中断使能
中断标记	[4]	此位用来标识是否有 I²C 中断发生，读出为 0 时表示没有中断发生，读出为 1 时表示有中断发生。当此位为 1 时，SCL 线被拉低，此时所有 I²C 传输停止；如果要继续传输，需写入 0 清除它
发送模式时钟分频系数	[3:0]	发送器时钟 = IICCLK /（IICCON[3:0] + 1）

使用 IICCON 寄存器时，有如下注意事项。

① 发送模式的时钟频率由位[6]、位[3:0]联合决定。另外，当 IICCON[6]=0 时，IICCON[3:0]不能取 0 或 1。

② I²C 中断在以下 3 种情况下发生：当发出地址信息或接收到一个从机地址并且吻合时，当总线仲裁失败时，当发送/接收完一个字节的数据（包括响应位）时。

③ 基于 SDA、SCL 线上时间特性的考虑，要发送数据时，先将数据写入 IICDS 寄存器，然后再清除中断。

④ 如果 IICCON[5]=0，IICCON[4]将不能正常工作。所以，即使不使用 I²C 中断，也要将 IICCON[5]设为 1。

（2）IICSTAT 寄存器（Multi-master IIC-bus control/status）。

IICSTAT 寄存器用于选择 I²C 接口的工作模式，发出 S 信号、P 信号，使能接收/发送功能，并标识各种状态，比如总线仲裁是否成功、作为从机时是否被寻址、是否接收到 0 地址、是否接收到 ACK 信号等。

IICSTAT 寄存器的各位如表 12.3 所示。

表 12.3　IICSTAT 寄存器的格式

功　能	位	说　明
工作模式	[7:6]	0b00：从机接收器 0b01：从机发送器 0b10：主机接收器 0b11：主机发送器
忙状态位/S 信号、P 信号	[5]	读此位时 0：总线空闲，1：总线忙 写此位时 0：发出 P 信号，1：发出 S 信号。当发出 S 信号后，IICDS 寄存器中的数据将被自动发送。
串行输出使能位	[4]	0：禁止接收/发送功能，1：使能接收/发送功能
仲裁状态	[3]	0：总线仲裁成功，1：总线仲裁失败
从机地址状态	[2]	作为从机时，在检测到 S/P 信号时此位被自动清 0； 接收到的地址与 IICADD 寄存器中的值相等时，此位被置 1
0 地址状态	[1]	在检测到 S/P 信号时此位被自动清 0； 接收到的地址为 0b0000000 时，此位被置 1
最后一位的状态	[0]	0：接收到的最后一位为 0（接收到 ACK 信号）； 1：接收到的最后一位为 1（没有接收到 ACK 信号）

（3）IICADD 寄存器（Multi-master IIC-bus address）。

用到 IICADD 寄存器的位[7:1]，表示从机地址。IICADD 寄存器在串行输出使能位 IICSTAT[4]为 0 时，才可以写入；在任何时间都可以读出。

（4）IICDS 寄存器（Multi-master IIC-bus Tx/Rx data shift）。

用到 IICDS 寄存器的位[7:0]，其中保存的是要发送或已经接收的数据。IICDS 寄存器在串行输出使能位 IICSTAT[4]为 1 时，才可以写入；在任何时间都可以读出。

2. S3C2410/S3C2440 I²C 总线操作方法

启动或恢复 S3C2410/S3C2440 的 I²C 传输有以下两种方法。

（1）当 IICCON[4]即中断状态位为 0 时，通过写 IICSTAT 寄存器启动 I²C 操作。有以下两种情况。

① 在主机模式，令 IICSTAT[5:4]等于 0b11，将发出 S 信号和 IICDS 寄存器的数据（寻址），令 IICSTAT[5:4]等于 0b01，将发出 P 信号。

② 在从机模式，令 IICSTAT[4]等于 1 将等待其他主机发出 S 信号及地址信息。

（2）当 IICCON[4]即中断状态位为 1 时，表示 I²C 操作被暂停。在这期间设置好其他寄存器之后，向 IICCON[4]写入 0 即可恢复 I²C 操作。所谓“设置其他寄存器”，有以下 3 种情况。

① 对于主机模式，可以按照上面①的方法写 IICSTAT 寄存器，恢复 I²C 操作后即可发出 S 信号和 IICDS 寄存器的值（寻址），或发出 P 信号。

② 对于发送器，可以将下一个要发送的数据写入 IICDS 寄存器中，恢复 I²C 操作后即可发出这个数据。

③ 对于接收器，可以从 IICDS 寄存器中读出接收到的数据。最后向 IICCON[4]写入 0 的同时，设置 IICCON[7]以决定在接收到下一个数据后是否发出 ACK 信号。

通过中断服务程序来驱动 I²C 传输。

（1）当仲裁失败时发生中断——本次传输没有抢到总线，可以稍后继续。

（2）对于主机模式，当发出 S 信号、地址信息并经过一个 SCL 周期（对应 ACK 信号）后，发生中断——主机可在此时判断是否成功寻址到从机。

（3）对于从机模式，当接收到的地址与 IICADD 寄存器吻合时，先发出 ACK 信号，然后发生中断——从机可在此时准备后续的传输。

（4）对于发送器，当发送完一个数据并经过一个 SCL 周期（对应 ACK 信号）后，发生中断。这时可以准备下一个要发送的数据，或发出 P 信号以停止传输。

（5）对于接收器，当接收到一个数据时，先根据 IICCON[7]决定是否发出 ACK 信号后，然后发生中断。这时可以读取 IICDS 寄存器得到数据，并设置 IICCON[7]以决定接收到下一个数据后是否发出 ACK 信号。

对于 4 种工作模式，S3C2410/S3C2440 数据手册中都有它们的操作流程图。现在以主机发送器作为例子说明，它的工作流程如图 12.7 所示，其他的工作模式请读者自行查阅数据手册。

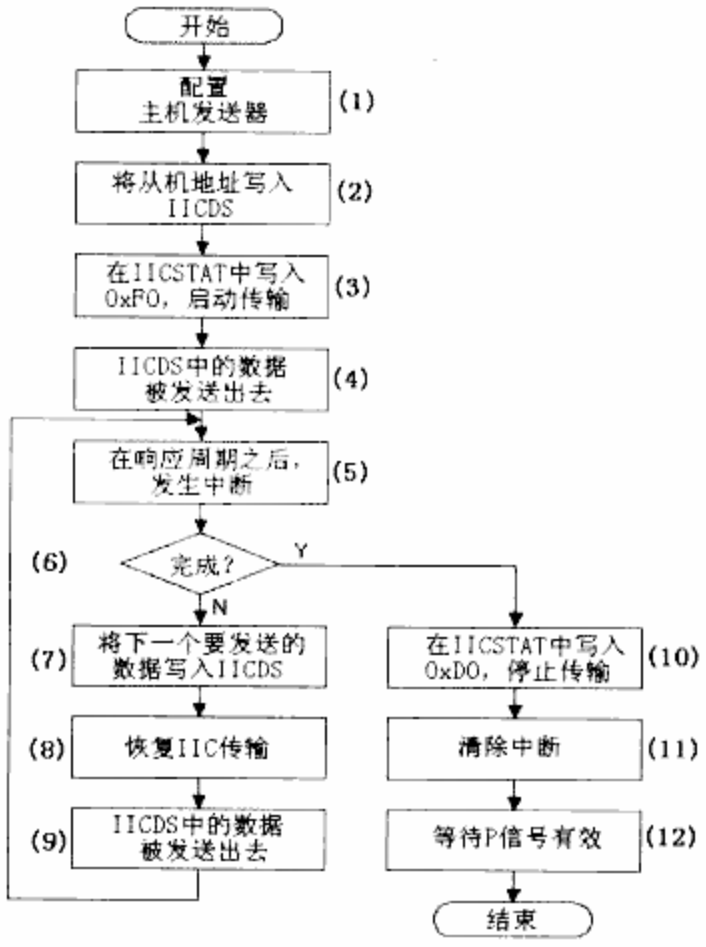

图 12.7　S3C2410/S3C2440 I²C 主机发送器的工作流程

下面结合 I²C 寄存器的用法，详细讲解图 12.7 中各步骤的含义。

（1）配置主机发送器的各类参数。

设置 GPE15、GPE14 引脚用于 SDA、SCL，设

置 IICCON 寄存器选择 I²C 发送时钟，最后，设置 IICSTAT[4]为 1，这样，后面才能写 IICDS 寄存器。

> **注意** 初始时 IICCON[4]为 0，不能将 IICSTAT 设为主机模式，否则就会立刻发出 S 信号、发送 IICDS 寄存器的值。

（2）将要寻址的从机地址写入 IICDS 寄存器。

（3）将 0xFO 写入 IICSTAT 寄存器，即设为主机发送器、使能串行输出功能、发出 S 信号。

（4）发出 S 信号后，步骤（2）中设置的 IICDS 寄存器值也将被发出，它用来寻址从机。

（5）在响应周期之后，发生中断，此时 IICCON[4]为 1，I²C 传输暂停。

（6）如果没有数据要发送，则跳到步骤（10）；否则跳到步骤（7）。

（7）将下一个要发送的数据写入 IICDS 寄存器中。

（8）往 IICCON[4]中写入 0，恢复 I²C 传输。

（9）这时 IICDS 寄存器中的值将被一位一位地发送出去。当 8 位数据发送完毕，再经过另一个 SCL 周期（对应 ACK 信号）后，中断再次发生，跳到步骤（5）。

步骤（5）～（9）不断循环直到发出了所有数据。当要停止传输时，跳到步骤（10）。

（10）将 0xFO 写入 IICSTAT 寄存器，即：设为主机发送器、使能串行输出功能、发出 P 信号。

> **注意** 这时的 P 信号并没有实际发出，只有清除了 IICCON[4]后才会发出 P 信号。

（11）清除 IICCON[4]，P 信号得以发出。

（12）等待一段时间，使得 P 信号完全发出。

12.2 I²C 总线操作实例

12.2.1 I²C 接口 RTC 芯片 M41t11 的操作方法

本书所用开发板中，通过 I²C 总线连接 RTC（实时时钟）芯片 M41t11，它使用电池供电，系统断电时也可以维持日期和时间。S3C2410/S3C2440 作为 I²C 主机向 M41t11 发送数据以设置日期和时间、读取 M41t11 以获得日期和时间。连接图如图 12.8 所示。

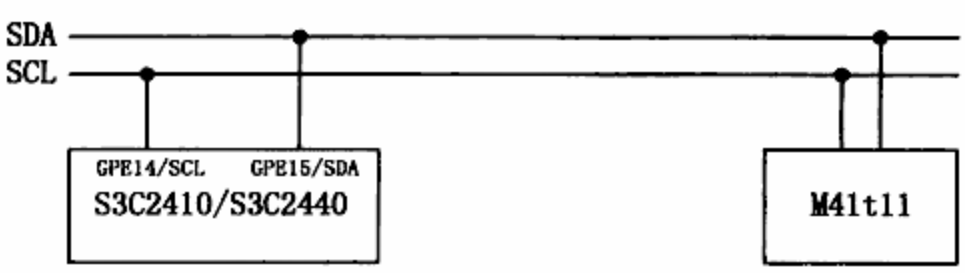

图 12.8 S3C2410/S3C2440 与 M41t11 连线图

M41t11 中有 8 个寄存器，分别对应秒、分、时、天（星期几）、日、月、年、控制寄存器，其中的数据都是以 BCD 格式保存（BCD 格式例子：0x15 表示数值 15），如表 12.4 所示。

表 12.4　M41t11 寄存器格式

地　址	数　据								功能/取值范围 BCD 编码	
	D7	D6	D5	D4	D3	D2	D1	D0		
0	ST	秒的十位数			秒的个位数				秒	00～59
1	X	分的十位数			分的个位数				分	00～59
2	CEB	CB	时的十位数		时的个位数				世纪位/时	0～1/00～23
3	X	X	X	X	X	天（星期几）			天	01～07
4	X	X	日的十位数		日的个位数				日	01～31
5	X	X	X	10 月	月的个位数				月	01～12
6	年的十位数				年的个位数				年	00～99
7	OUT	FT	S	校准					控制	

注：

①ST 为停止位，写入 1 时 M41t11 停止工作，写入 0 时开始工作。

②CEB 为“世纪使能位”，当它为 1 时，每过 100 年，CB 位就反转一次。

③“10 月”表示“月的十位数”。

④地址为 7 的寄存器有一些控制功能，本书没有使用。

除上表的 8 个寄存器（地址为 0～7）之外，M41t11 内部还有 56 字节的 RAM（地址为 8～63）。访问 M41t11 前，先设置寄存器地址，以后每次读/写操作完成后，M41t11 内部会自动将寄存器地址加 1。

所以读写 M41t11 分以下两个步骤。

（1）主机向 M41t11 发出要操作的寄存器起始地址（0～7）。

（2）要设置 M41t11 时，主机连续发出数据；要读取 M41t11 时，主机连续读取数据。

M41t11 的 I²C 从机地址为 0xD0。

12.2.2　程序设计

本实例将在串口上输出一个菜单，可以选择设置时间和日期，或者将它们读出来。将通过本实例验证 I²C 主机的发送、接收操作。

12.2.3　设置/读取 M41t11 的源码详解

本实例的源码在/work/hardware i2c 目录下。

文件 i2c.c 封装了 S3C2410/S3C2440 作为主机发送器、主机接收器的 4 个操作函数：i2c_init 用于初始化，i2c_write 用发起发送数据，i2c_read 用于发起读取数据，I2CIntHandle 是 I²C 中断服务程序，它用于完成后续的数据传输。

1．S3C2410/S3C2440 I²C 控制器初始化

i2c_init 函数对应于图 12.7 中步骤（1），它用来初始化 I²C，代码如下：

```
24 /*
25  * I²C 初始化
26  */
```

```
27 void i2c_init(void)
28 {
29     GPEUP  |= 0xc000;        // 禁止内部上拉
30     GPECON |= 0xa0000000;    // 选择引脚功能，GPE15:IICSDA, GPE14:IICSCL
31
32     INTMSK &= ~(BIT_IIC);
33
34     /* bit[7] = 1, 使能 ACK
35      * bit[6] = 0, IICCLK = PCLK/16
36      * bit[5] = 1, 使能中断
37      * bit[3:0] = 0xf, Tx clock = IICCLK/16
38      * PCLK = 50MHz, IICCLK = 3.125MHz, Tx Clock = 0.195MHz
39      */
40     IICCON = (1<<7) | (0<<6) | (1<<5) | (0xf);  // 0xaf
41
42     IICADD  = 0x10;      // S3C24xx slave address = [7:1]
43     IICSTAT = 0x10;      // I²C 串行输出使能(Rx/Tx)
44 }
45
```

第 29、30 行将引脚 GPE15、GPE14 的功能选择用于 I²C：IICSDA、IICSCL。

第 32 行在 INTMSK 寄存器中开启 I²C 中断，这样，以后调用 i2c_read、i2c_write 启动传输时，即可以触发中断，进而可以在中断服务程序中进一步完成后续传输。

第 40 行用于选择发送时钟，并进行一些设置：使能 ACK、使能中断。

第 42 行用于设置 S3C2410/S3C2440 作为 I²C 从机时的地址，在本实例中没有用到。

第 43 行使能 I²C 串行输出（设置 IICSTAT[4]为 1），这样，在 i2c_write、i2c_read 函数中就可以写 IICDS 寄存器了。

2. S3C2410/S3C2440 I²C 主机发送函数

初始化完成后，就可调用 i2c_read、i2c_write 读写 I²C 从机了。它们的使用方法从参数名称即可看出。这两个函数仅仅是启动 I²C 传输，然后等待，直到数据在中断服务程序中传输完毕后再返回。

i2c_write 函数的实现如下：

```
46 /*
47  * 主机发送
48  * slvAddr : 从机地址，buf : 数据存放的缓冲区，len : 数据长度
49  */
50 void i2c_write(unsigned int slvAddr, unsigned char *buf, int len)
51 {
```

```
52      g_tS3C24xx_I²C.Mode = WRDATA;    // 写操作
53      g_tS3C24xx_I²C.Pt   = 0;         // 索引值初始为 0
54      g_tS3C24xx_I²C.pData = buf;      // 保存缓冲区地址
55      g_tS3C24xx_I²C.DataCount = len; // 传输长度
56
57      IICDS   = slvAddr;
58      IICSTAT = 0xf0;          // 主机发送，启动
59
60      /* 等待直至数据传输完毕 */
61      while (g_tS3C24xx_I²C.DataCount != -1);
62  }
63
```

第 52～55 行用于设置全局变量 g_tS3C24xx_I2C，它表明当前是写操作，并保存缓冲区地址、要传送数据的长度，将缓冲区索引值初始化为 0。

第 57 将从机地址写入 IICDS 寄存器，这样，在第 58 行启动传输并发出 S 信号后，紧接着就自动发出从机地址。

第 58 行设置 IICSTAT 寄存器，将 S3C2410/S3C2440 设为主机发送器，并发出 S 信号。后续的传输工作将在中断服务程序中完成。

第 61 行等待 g_tS3C24xx_I2C.DataCount 在中断服务程序中被设为-1，这表明传输完成，于是返回。

3. S3C2410/S3C2440 I²C 主机接收函数

i2c_read 函数的实现与 i2c_write 函数类似，代码如下：

```
64  /*
65   * 主机接收
66   * slvAddr : 从机地址，buf : 数据存放的缓冲区，len : 数据长度
67   */
68  void i2c_read(unsigned int slvAddr, unsigned char *buf, int len)
69  {
70      g_tS3C24xx_I2C.Mode = RDDATA;    // 读操作
71      g_tS3C24xx_I2C.Pt   = -1;        // 索引值初始化为-1，表示第 1 个中断时不接收
数据(地址中断)
72      g_tS3C24xx_I2C.pData = buf;      // 保存缓冲区地址
73      g_tS3C24xx_I2C.DataCount = len; // 传输长度
74
75      IICDS       = slvAddr;
76      IICSTAT     = 0xb0;     // 主机接收，启动
77
```

```
78      /* 等待直至数据传输完毕 */
79      while (g_tS3C24xx_I²C.DataCount != -1);
80  }
81
```

需要注意的是第 71 行将索引值设为-1，在中断处理函数中会根据这个值决定是否从 IICDS 寄存器中读取数据。读操作时，第 1 次中断发生时表示发出了地址，这时候还不能读取数据。

4. S3C2410/S3C2440 I²C 中断服务程序

I²C 操作的主体在中断服务程序，它分 3 部分：首先是在 SRCPND、INTPND 中清除中断，后面两部分分别对应于写操作、读操作。先看清除中断的代码：

```
82  /*
83   * I²C 中断服务程序
84   * 根据剩余的数据长度选择继续传输或者结束
85   */
86  void I2CIntHandle(void)
87  {
88      unsigned int iicSt,i;
89
90      // 清中断
91      SRCPND = BIT_IIC;
92      INTPND = BIT_IIC;
93
94      iicSt  = IICSTAT;
95
96      if(iicSt & 0x8){ printf("Bus arbitration failed\n\r"); }    // 仲裁
失败
```

第 91、92 行用来清除 I²C 中断的代码。需要注意的是，即使清除中断后，IICCON 寄存器中的位 4（中断标志位）仍为 1，这导致 I²C 传输暂停。

第 94 读取状态寄存器 IICSTAT，发生中断时有可能是因为仲裁失败，在第 96 行对它进行处理。本程序中忽略仲裁失败（只有一个 I²C 主机，不可能发生仲裁失败）。

接下来是一个 switch 语句，分别处理写操作、读操作。先看写操作：

```
98      switch (g_tS3C24xx_I2C.Mode)
99      {
100         case WRDATA:
101         {
102             if((g_tS3C24xx_I2C.DataCount--) == 0)
```

```
103             {
104                 // 下面两行用来恢复 I²C 操作，发出 P 信号
105                 IICSTAT = 0xd0;
106                 IICCON  = 0xaf;
107                 Delay(10000);  // 等待一段时间以便 P 信号已经发出
108                 break;
109             }
110
111             IICDS = g_tS3C24xx_I2C.pData[g_tS3C24xx_I2C.Pt++];
112
113             // 将数据写入 IICDS 后，需要一段时间才能出现在 SDA 线上
114             for (i = 0; i < 10; i++);
115
116             IICCON = 0xaf;      // 恢复 I²C 传输
117             break;
118         }
119
```

g_tS3C24xx_I2C.DataCount 表示剩余等待传输的数据个数，第 102 行判断数据是否已经全部发送完毕：若是，则通过第 105、106 行发出 P 信号，停止传输。

第 105 行设置 IICSTAT 寄存器以便发出 P 信号，但是由于这时 IICCON[4]仍为 1，P 信号还没有实际发出。当第 106 行清除 IICCON[4]后，P 信号才真正发出。第 107 行等待一段时间，确保 P 信号已经发送完毕。

如果数据还没发送完毕，第 111 行从缓冲区中得到下一个要发送的数据，将它写入 IICDS 寄存器中。稍加等待之后，即可在第 116 行清除 IICCON[4]以恢复 I²C 传输，这时，IICDS 寄存器中的数据就会发送出去，这将触发下一个中断。

I²C 读操作的处理与写操作类似，代码如下：

```
120         case RDDATA:
121         {
122             if (g_tS3C24xx_I2C.Pt == -1)
123             {
124                 // 这次中断是在发送 I²C 设备地址后发生的，没有数据
125                 // 只接收一个数据时，不要发出 ACK 信号
126                 g_tS3C24xx_I2C.Pt = 0;
127                 if(g_tS3C24xx_I2C.DataCount == 1)
128                     IICCON = 0x2f;   // 恢复 I²C 传输，开始接收数据，接收到数据时不
发出 ACK
129                 else
130                     IICCON = 0xaf;   // 恢复 I²C 传输，开始接收数据
```

```
131             break;
132         }
133
134         if ((g_tS3C24xx_I2C.DataCount--) == 0)
135         {
136             g_tS3C24xx_I2C.pData[g_tS3C24xx_I2C.Pt++] = IICDS;
137
138             // 下面两行恢复 I²C 操作，发出 P 信号
139             IICSTAT = 0x90;
140             IICCON  = 0xaf;
141             Delay(10000);  // 等待一段时间以便 P 信号已经发出
142             break;
143         }
144
145     g_tS3C24xx_I2C.pData[g_tS3C24xx_I2C.Pt++] = IICDS;
146
147     // 接收最后一个数据时，不要发出 ACK 信号
148     if(g_tS3C24xx_I2C.DataCount == 0)
149         IICCON = 0x2f;   // 恢复 I²C 传输，接收到下一数据时无 ACK
150     else
151         IICCON = 0xaf;   // 恢复 I²C 传输，接收到下一数据时发出 ACK
152     break;
153 }
```

读操作比写操作多了一个步骤：第 1 次中断发生时表示发出了地址，这时候还不能读取数据，在代码中要分辨这点。对应第 122～132 行：如果 g_tS3C24xx_I2C.Pt 等于-1，表示这是第 1 次中断，然后修改 g_tS3C24xx_I2C.Pt 为 0，并设置 IICCON 寄存器恢复 I²C 传输（第 127～130 行）。

当数据传输已经开始后，每接收到一个数据就会触发一次中断。后面的代码读取数据，判断所有数据是否已经完成：如果完成则发出 P 信号，否则继续下一次传输。

第 134 行判断数据是否已经全部接收完毕：若是，先通过第 136 行将当前数据从 IICDS 寄存器中取出存入缓冲区，然后通过第 139、140 行发出 P 信号停止传输。

第 139 行设置 IICSTAT 寄存器以便发出 P 信号，但是由于这时 IICCON[4]仍为 1，P 信号还没有实际发出。第 140 行清除 IICCON[4]后，P 信号才真正发出。第 141 行等待一段时间，确保 P 信号已经发送完毕。

第 145～151 行用来启动下一个数据的接收。

第 145 行将当前数据从 IICDS 寄存器中取出存入缓冲区中。

第 148～151 行判断是否只剩下最后一个数据了：若是，就通过第 149 行中清除 IICCON[4]、IICCON[7]，这样即可恢复 I²C 传输，并使得接收到数据后，S3C2410/S3C2440 不发出 ACK 信号(这样从机即可知道数据传输完毕)；否则，在第 151 行中只要清除 IICCON[4]

以恢复 I²C 传输。

中断服务程序中，当数据传输完毕时，g_tS3C24xx_I2C.DataCount 将自减为-1，这样，i2c_read 或 i2c_write 函数即可跳出等待，直接返回。

5. RTC 芯片 M41t11 特性相关的操作

m41t11.c 文件中提供两个函数 m41t11_set_datetime、m41t11_get_datetime，前者用来设置日期与时间，后者用来读取日期与时间。它们都通过调用 i2c_read 或 i2c_write 函数来完成与 M41t11 的交互。

前面说过，操作 M41t11 只需要两步骤：发出寄存器地址，发出数据或读取数据。m41t11_set_datetime 函数把这两个步骤合为一个 I²C 写操作，m41t11_get_datetime 函数先发起一个 I²C 写传输，再发起一个 I²C 读传输。

先看 m41t11_set_datetime 函数的代码：

```
29 /*
30  * 写 m41t11，设置日期与时间
31  */
32 int m41t11_set_datetime(struct rtc_time *dt)
33 {
34     unsigned char leap_yr;
35     struct {
36         unsigned char addr;
37         struct rtc_registers rtc;
38     } __attribute__ ((packed)) addr_and_regs;
       …… /* 设置 rtc 结构，即根据传入的参数构造各寄存器的值 */
76     i2c_write(0xD0, (unsigned char *)&addr_and_regs, sizeof(addr_and_regs));
77
78     return 0;
79 }
80
```

省略号表示的代码用来设置 addr_and_regs 结构。这个结构分为两部分：addr_and_regs.addr 表示 M41t11 寄存器地址（它被设为 0），addr_and_regs. rtc 表示 M41t11 的 8 个寄存器——秒、分、时、天（星期几）、日、月、年、控制寄存器。

根据传入参数填充好 addr_and_regs 结构之后，就可以启动 I²C 写操作了。第 38 行使用“__attribute__ ((packed))”设置这个结构为紧凑格式，使得它的大小为 9 个字节（否则大小为 12 字节）：1 个字节用来保存寄存器地址，8 个字节用来保存 8 个寄存器的值。

第 76 行发起一次 I²C 写操作，将 addr_and_regs 结构结构中的数据（寄存器地址、日期、时间）发送给 M41t11：M41t11 会把接收到的第 1 个数据当作寄存器的起始地址，随后是要写入寄存器的数据。

m41t11_get_datetime 函数的代码与 m41t11_set_datetime 函数类似，如下所示：

```
81 /*
82  * 读取 m41t11，获得日期与时间
83  */
84 int m41t11_get_datetime(struct rtc_time *dt)
85 {
86      unsigned char addr[1] = { 0 };
87      struct rtc_registers rtc;
88
89      memset(&rtc, 0, sizeof(rtc));
90
91      i2c_write(0xD0, addr, 1);
92      i2c_read(0xD0, (unsigned char *)&rtc, sizeof(rtc));
93
        ..... /* 根据读出的各寄存器的值，设置 dr 结构 */
110      return 0;
111 }
112
```

第 91 行发起一次 I²C 写传输，设置要操作的 M41t11 寄存器地址为 0。

第 92 行发起一次 I²C 读传输，读出 M41t11 各寄存器的值。

省略号对应的代码根据读出的各寄存器的值，设置 dr 结构。M41t11 中以 BCD 码表示日期与时间，需要转换为程序使用的一般二进制格式。

12.2.4 I²C 实例的连接脚本

本实例要用到第 8 章 NAND Flash 控制器的函数将代码从 NAND Flash 复制到 SDRAM 中。由于 nand 代码中用到了全局变量，而全局变量要运行于可读写的内存中，为了方便，使用连接脚本将这些初始化代码放在 Steppingstone 中。

连接脚本为 i2c.lds，内容如下：

```
01 SECTIONS {
02      . = 0x00000000;
03      .init : AT(0){ head.o init.o nand.o}
04      . = 0x30000000;
05      .text : AT(4096) { *(.text) }
06        .rodata ALIGN(4) : AT((LOADADDR(.text)+SIZEOF(.text)+3)&~(0x03))
{*(.rodata*)}
07      .data ALIGN(4)   : AT((LOADADDR(.rodata)+SIZEOF(.rodata)+3)&~(0x03))
{ *(.data) }
08      _ _bss_start = .;
09      .bss ALIGN(4)  : { *(.bss)  *(COMMON) }
```

```
10      _ _bss_end = .;
11 }
```

第2～3行将head.S、init.c和nand.c对应的代码的运行地址设为0，加载地址（存在NAND Flash上的地址）设为0。从NAND Flash启动时，这些代码被复制到Steppingstone后就可以直接运行。

第4行设置其余代码的运行地址为0x30000000；第5行将代码段的加载地址设为4096，表示代码段将存在NAND Flash地址4096处。

第6～7行的“AT(…)”设置rodata段、data段的加载地址依次位于代码段之后。“LOADADDR(…)”表示某段的加载地址，SIZEOF(…)表示它的大小。这两行的前面使用“ALIGN(4)”使得它们的运行地址为4字节对应，为了使各段之间加载地址的相对偏移值等于运行地址的相对偏移值，需要将“AT(…)”中的值也设为4字节对齐：先加上3，然后与~(0x03)进行与操作（将低两位设为0）。

12.2.5 实例测试

本程序在main函数中通过串口输出一个菜单，用于设置或读取时间，步骤如下。

（1）使用串口将开发板的COM0和PC的串口相连，打开PC的串口工具并设置其波特率为115200、8N1。

（2）在i2c目录下执行make命令生成i2c可执行程序，烧入NAND Flash后运行。

（3）在PC的串口工具上，可以看到如下菜单：

```
##### RTC Menu #####
Data format: 'year.month.day w hour:min:sec', 'w' is week day
eg: 2007.08.30 4 01:16:57
[S] Set the RTC
[R] Read the RTC
Enter your selection:
```

（4）要设置RTC，输入“s”或“S”，可以看到如下字符。

```
Enter date&time:
```

在串口中按照“year.month.day w hour:min:sec”的格式输入日期与时间，比如：“2007.08.30 4 01:16:57”，然后按回车键。

> **注意** 只能输入2000.01.1至2099.12.31之内的日期与时间；年月日与星期几必须真实存在，否则RTC芯片无法正常工作。

（5）要可读取RTC，输入“r”或“R”，即可看到当前日期与时间，串口上会输出类似下面的结果。

```
*** Now is: 2007.08.30 4 01:16:57 ***
```

（6）断电后重启，输入“R”，仍可以看到正确的时间，只要RTC芯片M41t11没被断电，它就会一直工作。实际上，常使用电池给RTC芯片供电，这使得主电源关闭后系统仍可正确计时。