一样。

3. 例　程

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
printf("%d\n", strstr("This is a string", "str"));
}
```

4. 返回值

字符指针。如果没有字符串被找到,则返回为空。

3.57　STRTOK 函数

1. 提　要

```
#include <string.h>
/* 中级和低级系列单片机 */
char * strtok (char * s1, const char * s2)
/* 高级系列单片机 */
far char * strtok (far char * s1, const char * s2)
```

2. 描　述

多次调用 strtok()函数可以将字符串 s1 分为几个独立的部分。s1 中包含 0 或其他一些包含在字符串 s2 中的字符。这个调用返回 1 个指向分隔符的第 1 个字符的指针;如果不存在分隔符,则返回为空。分隔符将被空格所覆盖,从而使目前的分隔标记不再起作用。

调用 strtok()函数之后,应使指针 s1 为空。这样,将从后向前查找,又返回分隔符中第 1 个字符的指针;如果没找到,则返回为空。

3. 例　程

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
char * ptr;
char * buf = "This is a string of words.";

      .
```

```
char * sep_tok = ". ,?! ";
ptr = strtok(buf, sep_tok);
while(ptr ! = NULL) {
printf("%s\n", ptr);
ptr = strtok(NULL, sep_tok);
}
}
```

4. 返回值

分隔符中的第 1 个字符的指针;或者返回为空。

注意,每次调用函数时,分隔字符串 s2 可以不一样。

3.58 TAN 函数

1. 提 要

```
# include <math. h>
double tan (double f)
```

2. 描 述

tan()函数用来计算参数 f 的正切值。

3. 例 程

```
# include <math. h>
# include <stdio. h>
# define C 3. 141592/180. 0
void
main (void)
{
double i;
for(i = 0 ; i <= 180. 0 ; i += 10)
printf("tan(%3. 0f) = %f\n", i, tan(i * C));
}
```

4. 参 阅

sin(), cos(), asin(), acos(), atan(), atan2()。

5. 返回值

f 的正切值。

3.59　TIME 函数

1. 提　要

#include <time.h>

time_t time (time_t * t)

2. 描　述

函数需要目标系统提供当前时间。函数没有给出。这个函数需由用户实现。在运行时，函数以 s 为单位返回当前时间。当前时间从 1970 年 1 月 1 日 0 点 0 分 0 秒开始有效。如果参数 t 不为空，那么这个值同样被保存到 t 所指的内存单元。

3. 例　程

```
#include <stdio.h>
#include <time.h>
void
main (void)
{
time_t clock;
time(&clock);
printf("%s", ctime(&clock));
}
```

4. 参　阅

ctime()，gmtime()，localtime()，asctime()。

5. 返回值

被执行的函数将返回从 1970 年 1 月 1 日 0 点 0 分 0 秒开始的精确到 s 的当前时间。

注意，time()函数没有被提供，用户必须采用前面提到的规范执行这一程序。

3.60　TOLOWER，TOUPPER，TOASCII 函数

1. 提　要

#include <ctype.h>

char toupper (int c)

char tolower (int c)

char toascii (int c)

2. 描 述

toupper()函数将小写字母转换为大写字母；而 tolower()函数则与之相反；toascii()函数用来保证得到 1 个 0～0177 之间的结果。如果参数不为字母表中的字母，则 toupper()函数和 tolower()函数都返回它们原来的参数值。

3. 例 程

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void
main (void)
{
char * array1 = "aBcDE";
int i;
for(i=0;i < strlen(array1); ++i) {
printf("%c", tolower(array1[i]));
}
printf("\n");
}
```

4. 参 阅

islower()，isupper()，isascii()等。

3.61 VA_START，VA_ARG，VA_END 函数

1. 提 要

```
#include <stdarg.h>
void va_start (va_list ap, parmN)
type va_arg (ap, type)
void va_end (va_list ap)
```

2. 描 述

这些函数的宏提供一个方便的路径，以传递函数参数。函数定义时，函数参数用省略号替代。这里函数参数的个数及类型在汇编时并不知道。

函数最右端的参数(用 parmN 表示)在宏汇编中起着非常重要的作用，因为它是得到更多参数的开始。函数可以取不同数量的参数。不同类型的 va_list(变量表)应事先定义，然后激活带有名为 parmN 的一系列参数的宏 va_start()。将变量初始化，从而允许调用宏 va_arg

()，得到其他的参数。

每次调用 va_arg()需有 2 个参数：一个在前面已定义，另一个参数也需要说明其类型。注意，所有的参数都将被自动加宽为整型、无符号整型及双精度型。例如，如果一个参数为字符型，则字符型自动转换为整型，函数调用的形式相当于 va_arg(ap,int)。

下面用例子说明带有一个整型变量和一些其他变量作为参数的函数。在这个例子中，函数将得到一个字符型指针；但要注意编译器并不知道，程序员对参数正确性负责。

3. 例　程

```
#include <stdio.h>
#include <stdarg.h>
void
pf (int a, ...)
{
va_list ap;
va_start(ap, a);
while(a--)
puts(va_arg(ap, char *));
va_end(ap);
}
void
main (void)
{
pf(3, "Line 1", "line 2", "line 3");
}
```

3.62　XTOI 函数

1. 提　要

```
#include <stdlib.h>
unsigned xtoi (const char * s)
```

2. 描　述

xtoi()函数扫描参数中的字符串。它跳过前面的空格，读到符号后，将用 ASCII 码表示的十六进制数转换为整型。

3. 例　程

```
#include <stdlib.h>
#include <stdio.h>
```

```
void
main (void)
{
char buf[80];
int i;
gets(buf);
i = xtoi(buf);
printf("Read %s: converted to %x\n", buf, i);
}
```

4. 参 阅

atoi()。

5. 返回值

有符号整数。如果字符串中不包含数,则返回 0。

第 4 章　PIC16F877 单片机实验板介绍

美国微芯公司推出的 CMOS 8 bit PIC 系列单片机，采用精简指令集(RISC)、哈佛总线结构、2 级流水线取指令方式，具有实用、低价、指令集小、简单易学、低功耗、高速度、体积小、功能强等特点，体现了单片机发展的一种新趋势，深受广大用户的欢迎，已逐渐成为单片机发展的新潮流。

PIC16F87X 是微芯公司的中档产品，采用 14 bit 的类 RISC 指令系统；在保持低价格的前提下，增加了 A/D 转换器、内部 EEPROM 存储器、比较输出、捕捉输入、PWM 输出(加上简单的滤波电路后，还可作为 D/A 输出)、I^2C 总线和 SPI 总线接口电路、异步串行通信(USART)接口电路、模拟电压比较器、LCD 驱动、FLASH 程序存储器等许多功能，可以方便地在线多次编程和调试，特别适用于初学者学习和产品的开发阶段使用；也可作为产品开发的终极产品。微芯公司还将 FLASH ROM 芯片做成与 OTP 芯片价格相近，以便可用 FLASH ROM 芯片代替 OTP 芯片。

微芯公司的单片机是品种最丰富的单片机系列之一，被广泛应用于各种仪器和设备中，具有如下显著的特点。

● 开发容易，周期短　PIC 采用类 RISC 指令集，指令数目少(PIC16F87X 仅 35 条指令)，且全部为单字长指令，易学易用。相对于采用 CISC(复杂指令集)结构的单片机，可节省 30%以上的开发时间、2 倍以上的程序空间。

● 高速　PIC 采用哈佛总线和类精简指令集，逐步建立了一种新的工业标准。指令的执行速度比一般的单片机要快 4～5 倍。

● 低功耗　PIC 采用 CMOS 电路，结合了诸多的节电特性，使其功耗很低；100%的静态设计可进入休眠(sleep)省电状态，而不会影响激活后的正常运行。

微芯公司的单片机是各类单片机中低功耗设计最好的产品之一。

● 低价实用　PIC 配备 OTP(one time programmable)型、EPROM 型及 FLASH ROM 型等多种形式的芯片，其 OTP 型芯片的价格很低。

PIC 还提供程序监视器(WDT)和程序可分区保密的保密位(security fuse)等功能；提供基于 Windows 98/NT/2000 的方便易用的全系列产品开发工具、大量的子程序库及应用实例，使产品开发更容易、更快捷。

为了更好地开展大学中单片机教学、实验及毕业设计环节中单片机的应用，提高 PIC16F877 开发的速度，我们采用 PIC16F877 单片机设计了一个功能齐全的通用模板。以后章节提供的大量应用程序和接口程序样例，都是在这个实验板的基础上编写的。

4.1 实验板功能介绍

在设计过程中,合理利用了 PIC16F877 单片机的几乎所有内部资源,设计了这块功能齐全的模板。下面对这些功能做简单的介绍。

4.1.1 A/D 转换功能

为了得到现场模拟信号,需要 A/D 采集。PIC16F877 单片机片内有 8 路 10 bit A/D 转换通道,在这里用了 3 路。其中 V_{IN0},V_{IN1}(标号介绍见 4.3 节)用来采集交流电压和相应的电流。由于 PIC16F877 单片机的片内 A/D 为单极性,要将交流信号经过放大、滤波,再提升电位后,才能送入单片机;因此,对于任一需要采样的交流信号,在接入实验板之前,先用信号调理电路将其转换为峰值为+2 V 的信号(此范围可调);再在实验板上加 2 V 的提升电压,就可得到幅值在 0～+4 V 之间的信号。根据信号调理电路的衰减倍数及提升电压的数值,很容易通过软件得出实际的 A/D 采样值。

通过采集这 2 路电压、电流信号,可以通过该实验板对电网的一些重要参数进行检测.如电压和电流的有效值、电网的有功和无功功率、功率因数、谐波等,还可以进行 FFT 运算。另外,用 V_{IN2}采集直流信号。由于直流信号种类多、量值范围宽,所以采用增益可调的同相比例放大器对直流信号进行初步处理后,再接入单片机的 A/D 输入通道。由于产生直流信号的方式很多,所以没有设计具体的电路。可以搭建一些简单的电路,配合该实验板实现多种信号(如温度、压力等)的检测。

4.1.2 键 盘

在许多应用中,需要用键盘输入数据或对程序的进程进行管理;因此在单片机的设计和调试实验中,键盘是一个不可缺少的部分。

本设计中采用单片机的 RB1,RB2,RB4,RB5 等 4 个 I/O 口和若干按键,构成一个简单的矩阵式键盘。其中 RB4 和 RB5 在引脚的电平发生变化时,产生"电平变化中断";因此,对键盘的输入扫描可采用查询方式或中断方式。键盘的详细工作原理请参考本书相关章节。

4.1.3 LED 显示

通常,需用 LED 显示单片机的工作状态、运行结果及参数等,它是人机对话的重要环节。本模板设计了 LED 显示部分。

为了节约 CPU 的资源,同时为了利用该单片机强大的 I/O 口扩展能力,采用 PIC16F877 单片机的 SPI 接口和移位寄存器芯片 74HC595(该芯片可以把串行数据转换成并行数据输出),实现 LED 的静态显示。通过级连的方式,把 8 个 74HC595 芯片连在一起,形成一个

64 bit的移位寄存器;同时每个 74HC595 芯片的并行输出连接一个 8 段 LED,这样就可以通过简单编程实现显示功能。比如要在 8 个 LED 上分别显示 1,2,3,4,5,6,7,8,可以通过 SPI 口连续发送 8 个数据的显示段码,发送完毕后,通过锁存信号线 LACK 发出一个锁存信号,便可以实现静态显示。LED 的详细工作原理请参考本书相关章节。

4.1.4　液晶显示器显示

在各种智能型仪器和便携式产品中,除了显示字符、数字外,通常还需要显示各种图形、曲线及汉字,并且实现屏幕上下左右滚动、动画、闪烁及文本特征显示等功能;为此,设计了液晶显示器(LCD)显示电路,以实现这些功能。

本模板采用 SED1520F0A 驱动器驱动的 MG_1223 液晶显示器,并且用有外部扩展接口的并行从动口 D 口复用为其数据总线。用户可查阅相关资料,对其性能进行了解;后面的章节也提供了相应的接口程序。

4.1.5　8 路开关量输入和 8 路开关量输出

在单片机应用现场,经常需要采集开关量信号,而且也常常需要用开关量输出作控制信号;因此在设计过程中,利用 PIC16F877 单片机的 SPI 串行外设口和 74HC165 芯片扩展了 8 路开关量输入通道。74HC165 是一种可将 8 bit 并行数据转换成串行数据的芯片,利用单片机的 SPI 串行外设接口的数据线和时钟控制线很容易将 74HC165 转换后的串行数据送入单片机。

在设计过程中,为了实验的需要,每一个输入引脚在留有外部接口的同时,都在实验板上接了一个单刀双掷开关。实验时可根据不同的需要,将相应的开关接到高电平或低电平上,从而模拟实际控制过程中的一些开关量输入。另外,还用一个 8 bit 的拨动开关与 D 口相连,可以把开关拨向 ON 位置,使板上的 8 个发光二极管与 D 口相连,从而进行简单的开关量输出操作和逻辑控制输出。

4.1.6　D/A 输出

工业现场中,常常需要用到 D/A 转换器输出模拟信号;因此在本模板的设计中,采用了 MAX518 和单片机的 I^2C 总线接口,扩展出 2 路 D/A 输出。值得注意的是,在扩展 D/A 输出时,用的是 I^2C 总线;开关量输入时,用的是 SPI 总线。但是这 2 个总线在 PIC16F877 单片机上的数据输入线和时钟线是相同的;因此需要在硬件设计上将 SPI 总线和 I^2C 总线加以区分,具体方式见后面的章节。

4.1.7　串行通信接口 SCI

在实际应用中,单片机和 PC 机之间经常需要进行数据交换。采用 MAX232 驱动

PIC16F877 单片机的 SCI 接口和标准 RS-232 电平接口，使单片机和 PC 机之间能方便地进行数据交换；当然，也可以方便地进行 2 个或多个单片机之间的通信。

4.1.8 主从单片机多机并行通信

PIC16F877 单片机模板可以和另一块 PIC16F877 单片机模板进行主从并行多机通信，提供中断方式和查询方式；因此把从动口 PORTD 留有外部接插头。该从动口与液晶的数据总线复用，但可以通过软件使二者互不干扰。

4.1.9 捕捉方式，PWM 方式

PIC16F877 单片机可以对外部信号的边缘进行捕捉，特别适用于转速测量、脉冲计量等应用场合。

PWM 输出方式在工程中的应用就更为广泛，在精度要求不是特别高的场合，还可以在外部接上简单的低通滤波器，利用 PWM 实现简单的 D/A 输出。该实验模板已经留有 CCP1 和 CCP2 口供用户使用。

4.1.10 多种复位方式

PIC16F877 单片机可以有多种复位方式，如上电复位、掉电锁定复位、程序监视器复位等。

4.1.11 时钟信号

PIC16F877 单片机的时钟信号由外部 4 MHz 晶振或 RC 谐振电路提供。定时器 1 由外部频率为 32.768 kHz 的晶振提供时钟，从而使得定时器 1 在单片机进入 SLEEP 状态时也能继续计数；在计数时间到时，将单片机从 SLEEP 状态唤醒，运行日历时钟程序，更新日历记录。

4.1.12 其他功能

为了方便调试，模板上配有 PIC16F877 单片机的仿真头，可实现在线调试，而不需将芯片从实验板上来回拔插，调试起来十分方便。最后在模板上放置了复位按键，在实验时可方便地进行手动复位。

另外，还留有双电源，可以将实验板上的 J1 接线头接入+5 V 电源；也可以接上跳针 J10 后，用专用的+9 V 插头电源(MPLAB_ICD 的电源)从 J6 接入供电。

4.2 实验板的硬件布局

图 4.1 是实验板元件位置图。图上给出了外部接线口位置、跳针位置、各种测量点的位置及在调试中要用到的其他硬件的位置。

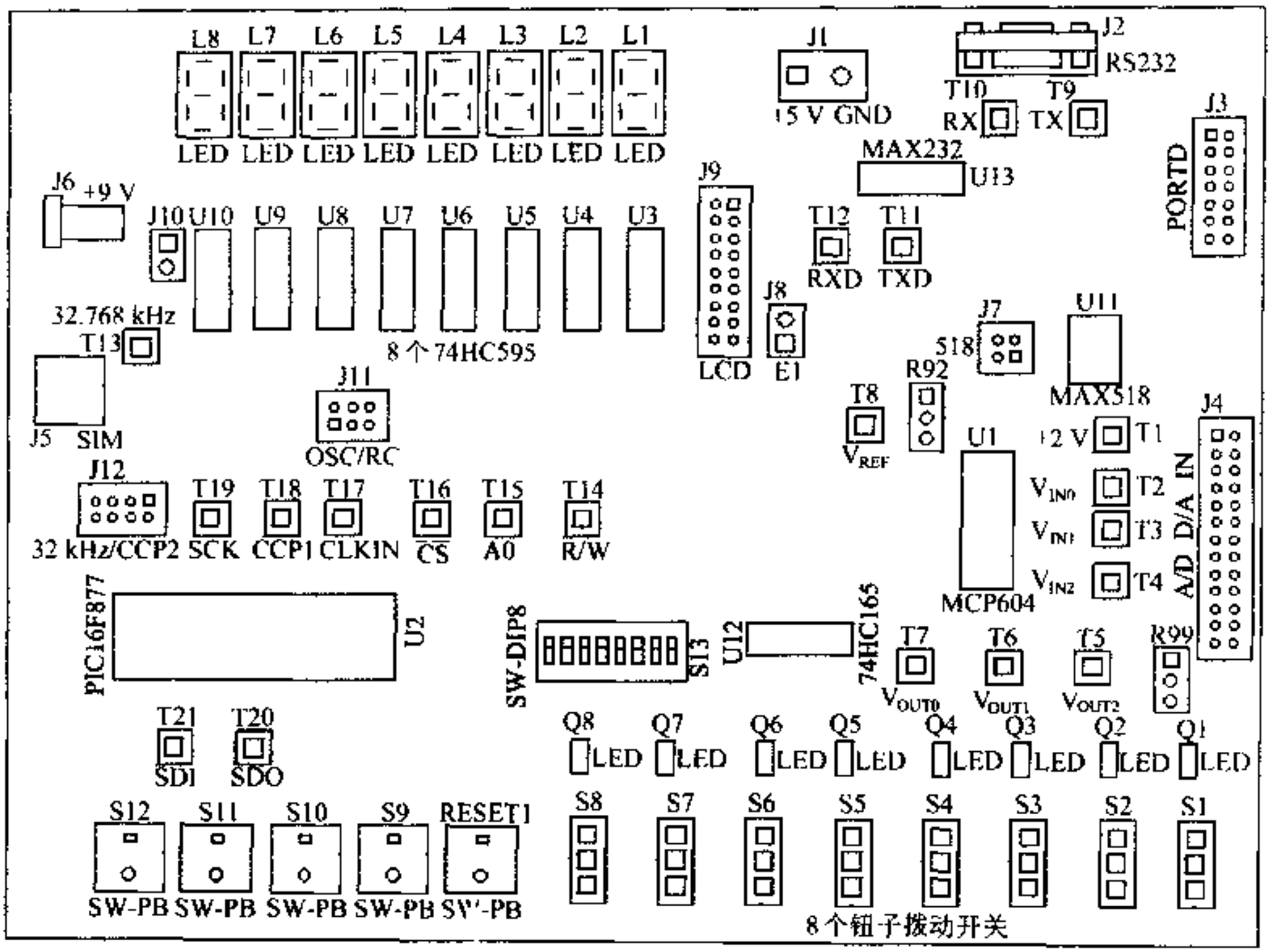


图 4.1　实验板元件位置图

4.3　测试点及主要器件介绍

表 4.1 和表 4.2 分别为实验板测试点功能介绍和主要操作器件功能介绍。

表 4.1　测试点功能介绍

编　号	文字符号	功　能
T1	+2 V	测试稳压管的稳压值
T2	V_{IN0}	外部交流模拟量 0 输入测试
T3	V_{IN1}	外部交流模拟量 1 输入测试
T4	V_{IN2}	外部直流模拟量输入测试
T5	V_{OUT2}	直流模拟量经过运放电路处理、进入单片机的测试
T6	V_{OUT1}	交流模拟量 1 经过运放电路处理、进入单片机的测试
T7	V_{OUT0}	交流模拟量 0 经过运放电路处理、进入单片机的测试
T8	V_{REF}	交流信号提升电压测试

续表 4.1

编　号	文字符号	功　能
T9	TX	通信数据电平经过 MAX232 变换后的测试
T10	RX	
T11	TXD	从单片机输出的通信数据电平的测试
T12	RXD	
T13	32.768 kHz	TMR1 振荡器电路的晶振振荡波形测试
T14	R/W	并行从动端口的$\overline{RD}$信号的测试
T15	A0	并行从动端口的$\overline{WR}$信号的测试
T16	$\overline{CS}$	并行从动端口的$\overline{CS}$信号的测试
T17	CLKIN	单片机工作晶振的振荡波形测试
T18	CCP1	CCP 模块 1 的测试
T19	SCK	SPI 或 I^2C 工作方式时的时钟测试
T20	SDO	SPI 工作方式时输出数据电平的测试
T21	SDI	SPI 工作方式时输入数据电平的测试

表 4.2　主要操作器件的功能简介

编　号	文字符号	功　能
J1	+5 V GND	外部+5V 直流工作电压输入
J2	RS233	RS-232 串行通信接口
J5	SIM	仿真器接口
J6	+9 V	外部+9 V 直流工作电压输入
S1～S 8	SW-SPDT	8 个钮子拨动开关，可在实验板上模拟 8 bit 开关量的输入
S9～S12	SW-PB	4 个自由键盘
RESET1	SW-PB	手动复位键
S13	SW-DIP8	8 bit DIP 拨码开关，拨动它可以把 PORTD 接在 8 个 LED 上

该实验板中的跳针（除 LCD 的插座 J9)引脚的序号定义均如图 4.2 所示，焊盘为方形的引脚为第 1 脚。引脚的具体意义见表 4.3。

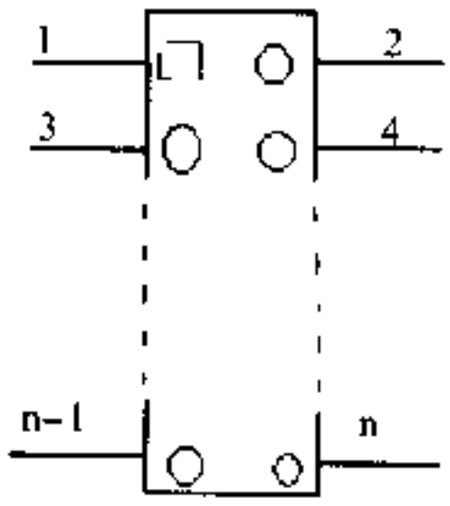


图 4.2　跳针定义

表 4.3　跳针或插座引脚设置

编　号	符号及功能	跳针引脚的功能设置							
J3	PORTD—并行从动口	引脚号	1	2	3～10	11	12	13	14
		功　能	V_{CC}	GND	数据 D0～D7	GND	RD	WR	CS
J4	A/D,D/A,IN——A/D 输入，D/A 输出，外部开关量输入	引脚号	1	2	3,5,7	1	6	8	9
		功　能	D/A0	D/A1	AGND	V_{IN0}	V_{IN1}	V_{IN2}	GND
		引脚号		10～17	18,19,25,26		20	21	22
		功　能		IN0～IN7	GND		CCP2	CCP1	B5
		引脚号		23	24				
		功　能		T1CKI	INTB0				

注：V_{CC}——+5 V 电压；GND——数字地；AGND——模拟地；D0～D7——并行从动口的 8 根数据线；RD——并行从动口的读操作输入；WR——并行从动口的写操作输入；D/A0，D/A1——MAX518 的 2 路 D/A 输出；V_{IN0}，V_{IN1}，V_{IN2}　见测试点功能介绍表；IN0～IN7——外部的 8 路开关量输入；CCP2，CCP1——单片机的 2 个 CCP 模块与外部的接口；B5——与单片机 B 口的 bit5 引脚相连(可以引入变位中断信号)；T1CKI——TMR1 计数器输入；INTB0——与单片机 B 口的 bit0 引脚相连，可以引入外部中断信号。

其他跳针及可变电阻使用说明。

J9　—该插座用于液晶接口，引脚的含义详见原理图。插上液晶时，必须保证液晶的显示屏向左。

J7——在调试 I^2C 功能之前，应把该跳针的引脚 1 和 2 短接，引脚 3 和 4 短接，即接上上拉电阻。

J8——该跳针只有 2 只引脚，调试液晶显示程序之前，应短接这 2 只引脚，使液晶的 E1 信号与单片机 B 口的 bit 0 引脚相连。

J10——该跳针只有 2 只引脚。若从 J1 接入+5 V 工作电压，则需断开该跳针；若从 J6 接入+9 V 的工作电压，则需短接该跳针。

J11——该跳针引脚 3,4 短接时，单片机工作于 RC 振荡器方式；引脚 1,2 短接以及引脚 5,6 短接时，单片机工作于晶体振荡器方式。

可以根据需要选择其中一种工作方式。

J12——该跳针引脚 1,2 和 5,6 分别短接时，接入 TMR1 的日历时钟晶体振荡器电路；当 3,4 短接时，接入 TMR1 的计数器；当 7,8 短接时，接入 CCP2 模块。

由于单片机的硬件特点，当接入 TMR1 的晶体振荡器电路并且使其工作时，TMR1 计数器和 CCP2 不接入。

R92——调整该可变电阻，可以改变交流输入电压的提升参考值(建议调整后 V_{REF} = +1 V)。

R99　调整该可变电阻可以改变直流输入电压的放大倍数。

实验板如图 4.3 所示。

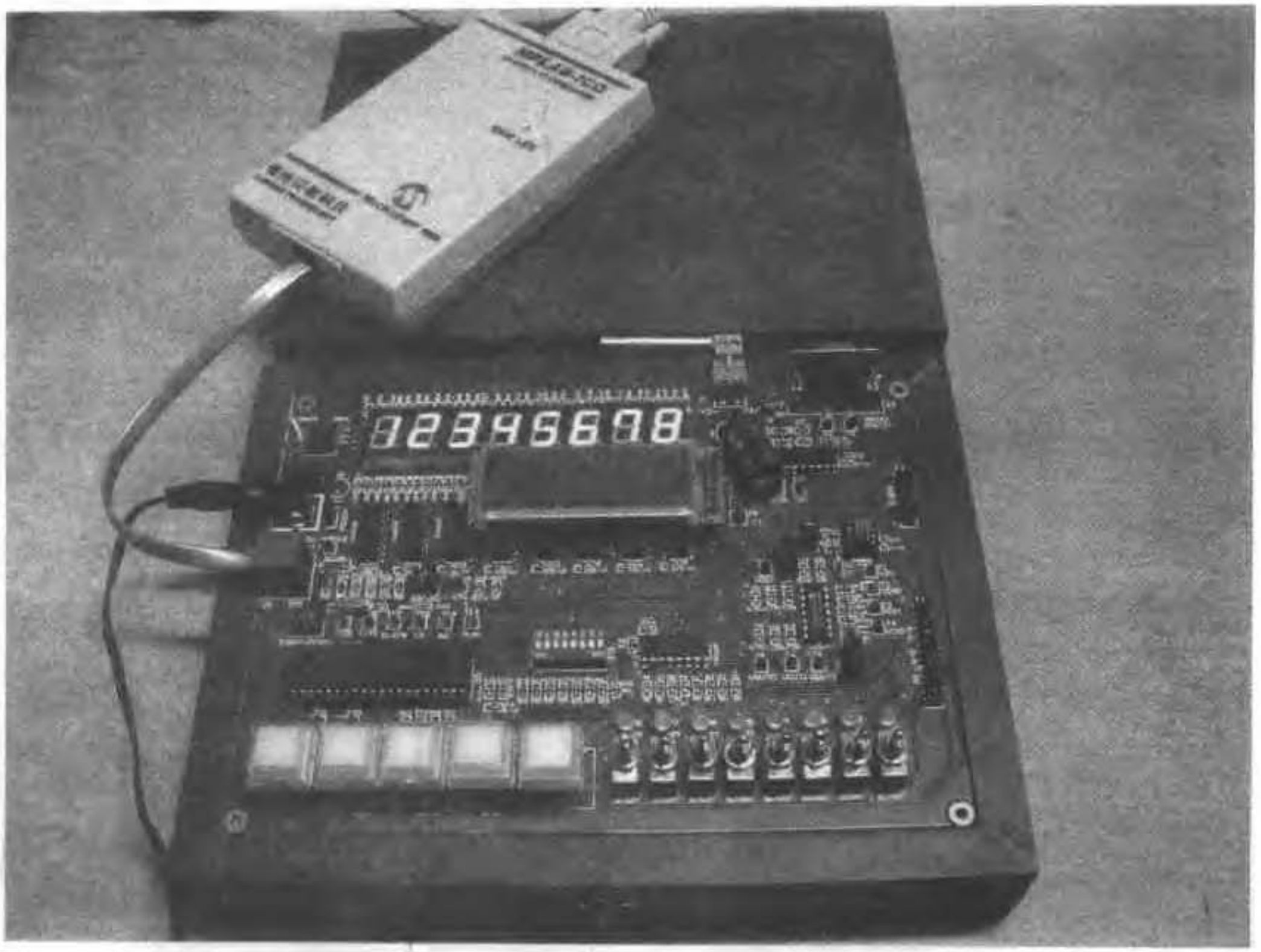


图 4.3 实验板

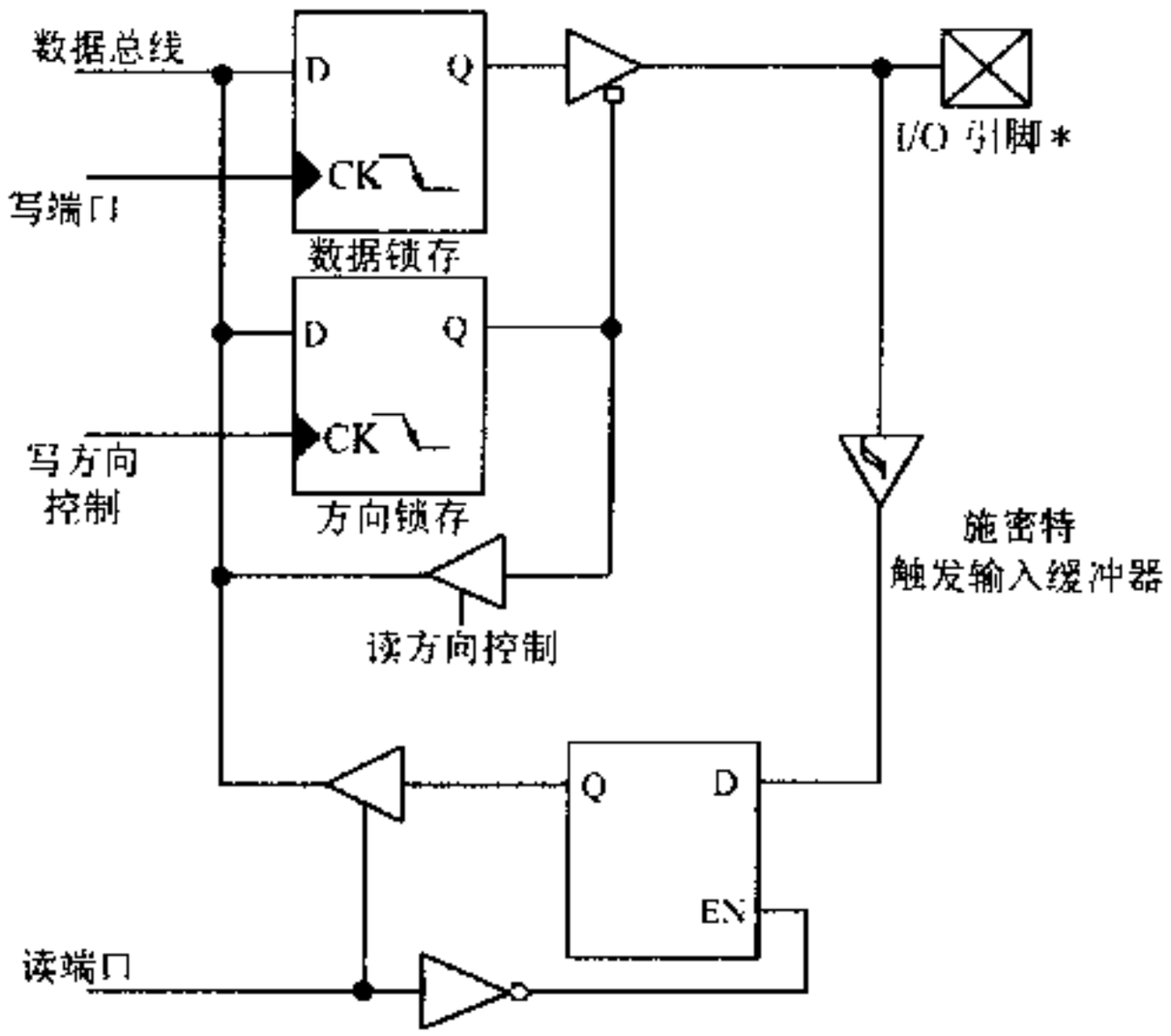
第 5 章　PIC16F877 的外围功能模块

5.1　输入/输出端口

PIC16F877 单片机包含 PORTA，PORTB，PORTC，PORTD，PORTE 等几组输入/输出(I/O)口。它们大多数具有丰富的第 2 功能，可通过设置适当的寄存器，使其工作在通用数字 I/O 口方式或第 2 功能状态下。本节主要简介 PORTD 的结构功能及其 I/O 工作方式。程序例子只需稍加修改，即可用于 PORTA，PORTB，PORTC，PORTE 端口。从例子中可以很容易地看出，所有操作都是对寄存器(或者说寄存器文件)的操作——对端口工作方式设置寄存器的操作、对端口数据方向设置寄存器的操作等，这是 PIC 单片机的一个特点。

5.1.1　PORTD 端口和 TRISD 寄存器

图 5.1 所示为 I/O 工作方式下的 PORTD 端口结构图。



* I/O 引脚有连接到 V_{DD}和 V_{SS}的保护二级管

图 5.1　I/O 工作方式下的 PORTD 端口结构图

PORTD 端口是一个带有施密特输入缓冲器的 8 bit 宽 I/O 端口，每个引脚可以分别定义成输入或输出，其方向控制寄存器是 TRISD。

当 TRISD 寄存器的相应位置 1 时，PORTD 的相应位就为输入，输出呈高阻状态；当 TRISD 寄存器的相应位清 0 时，PORTD 的相应位就为输出，输入呈高阻状态。以下为一段常见的用 C 语言编写的 PORTD 口初始化语句：

```
TRISD = 0XF0          ;TRISD 寄存器赋值，高 4 位为输入，低 4 位为输出
```

执行上面的程序语句后，PORTD 的高 4 位为输入，低 4 位为输出。

其他各口也包含 PORTX 和 TRISX 2 个寄存器(X 代表 A,B,C,D,E)。其中 TRISX 寄存器也控制 PORTX 的数据输入输出方向，控制规则同上。

5.1.2　简单应用实例

该例用于令与 PORTD 口相连的 8 个发光二极管前 4 个点亮，后 4 个熄灭。在调试程序前，应使与 PORTD 口相连的 8 bit 拨码开关拨向相应的位置。

例 5.1　PORTD 输出

```
    #include   <pic.h>
main()
{
    TRISD= 0X00;              /* TRISD 寄存器被赋值，PORTD 每一位都为输出 */
    while(1);                 /* 循环执行点亮发光二极管的语句 */
    {
        PORTD=0XF0;           /* 向 PORTD 送数据，点亮 LED(由实验模板 */
                              /* 的设计决定相应位置低时 LED 点亮)。 */
    }
}
```

5.2　利用 MSSP 模块的 SPI 方式实现与 LED 数码显示接口

在各种实验和工程应用中，常常用 LED 显示单片机的工作状态或一些运行结果、参数等。本模板中利用 MSSP 模块的 SPI 方式扩展了 LED 数码显示。

5.2.1　MSSP 模块 SPI 方式功能简介

主同步串行口 MSSP 模块是用来与其他外围串行接口或其他微控制器芯片进行通信的串行接口。这些外围设备可以是串行 EEPROM、移位寄存器、显示电路或 A/D 转换器等。MSSP 有以下 2 种工作方式：串行外围接口(SPI)和芯片间总线(I^2C)。其中 SPI 方式可允许同时同步发送和接收 8 bit 数据，其 4 种工作方式是相似的。为了完成通信任务，需用如下 3

个引脚：① 串行数据输出(SDO)；② 串行数据输入(SDI)；③ 串行时钟(SCK)。另外，当 SPI 工作在从动工作方式时，可能还需要第 4 个引脚：从动选择($\overline{SS}$)。

当对 SPI 进行初始化时，需指定几个选项，即通过对 SSPCON1 寄存器的 bit 5～bit 0 和 SSPSTAT 寄存器的 bit 7～bit 6 进行编程来实现。这些控制位可确定以下工作方式和参数：① 主控方式(SCK 为时钟输出)；② 从动方式(SCK 为时钟输入)；③ 时钟极性(SCK 为空闲状态)；④ 数据输入采样相位(数据输出时间的中间或末尾)；⑤ 时钟边沿(在 SCK 信号的上升/下降沿时输出数据)；⑥ 时钟速率(仅用于主控方式)；⑦ 从动选择方式(仅用于从动方式)。

图 5.2 是工作在 SPI 工作方式下的 MSSP 模块结构图。

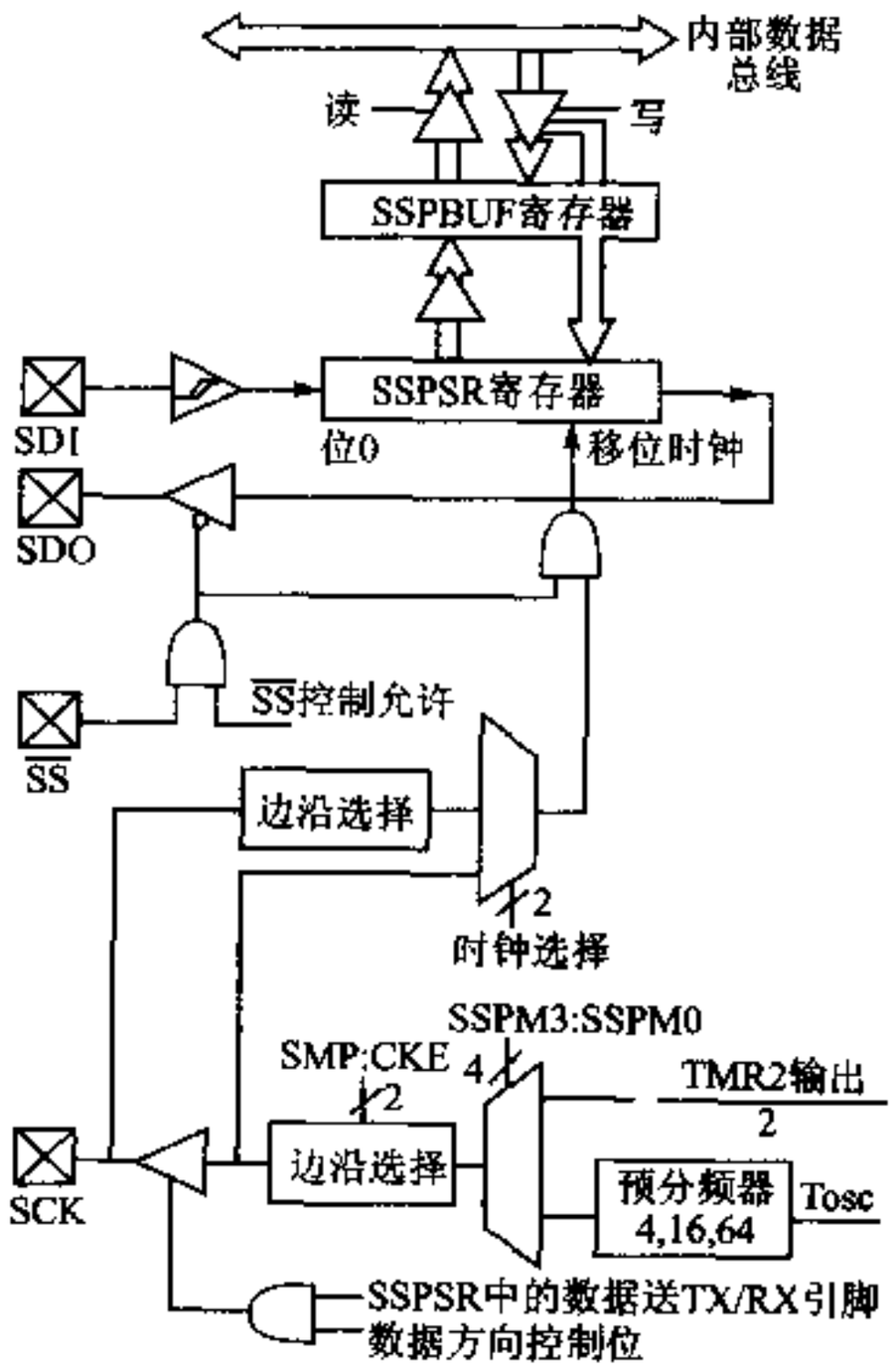


图 5.2　SPI 工作方式下的 MSSP 结构图

主同步串行口 MSSP 由一个发送/接收移位寄存器(SSPSR)和一个缓冲寄存器(SSPBUF)组成。SSPSR 寄存器用于芯片数据的移入和移出，高位在前。接收的数据准备好以后，SSPSR 才会把保存的数据写入到 SSPBUF。一旦接收到数据的第 8 位，就立即将其送入 SSPBUF 寄存器中，然后缓冲区满检测位 BF(SSPSTAT 寄存器的 bit 0)和中断标志位 SSPIF(PIR1 寄存器的 bit 3)都被置为 1。这种接收数据的双缓冲方式(SSPBUF)允许在刚接收到的数据被读取之前就开始下一个数据接收。在发送/接收数据期间，任何对 SSPBUF 的写操

作都是无效的;同时写冲突检测位 WCOL(即 SSPCON 的 bit 7)被置为 1。必须用软件将 WCOL 清 0,以判定下一个对 SSPBUF 寄存器的写操作是否成功完成。

当应用软件要接收一个有效数据时,应在下一个需要发送的数据字节写入 SSPBUF 寄存器之前,读出 SSPBUF 寄存器中的数据。缓冲区满检测位 BF 为 1 时(即 SSPSTAT 寄存器的 bit 0),表示接收到了数据并送入了 SSPBUF 寄存器中。在对 SSPBUF 寄存器进行读操作时,BF 位被自动清 0。如果 SPI 仅是一个发送器,这个数据就可能是不相关的。一般情况下,MSSP 中断用来确定发送或接收的完成时间。SSPBUF 中的数据必须读出或对其进行发送写操作。如果不想使用中断的方法,可以用软件查询的方法(可以查询 SSPIF 或 BF 2 个标志位中的任意一个),确保不会发生冲突。

下面给出与 SPI 方式密切相关的寄存器 SSPSTAT 和 SSPCON1 各位的定义。

1. 同步串行口状态寄存器 SSPSTAT(地址 94h)各位的定义

R/W 0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

R=可读位;
W=可写位;
U=未用,读出时为"0";
-n=上电复位值。

bit 7,SMP——采样位。

SPI 主控工作方式下:1 = 在输出数据末尾采样输入数据;0 = 在输出数据中间采样输入数据。

SPI 从动工作方式下:SMP 位必须被清 0。

I^2C 主、从工作方式下:1 = 关闭标准速度方式(100 kHz 和 1 MHz)的回转率控制;0 = 打开高速方式(400 kHz)的回转率控制。

bit 6,CKE——SPI 工作方式下的时钟沿选择位。

SPI 工作方式下:

CKP = 0 时,1 = 在串行时钟 SCK 的上升沿发送数据;0 = 在串行时钟 SCK 的下降沿发送数据。

CKP = 1 时,1 = 在串行时钟 SCK 的下降沿发送数据;0 = 在串行时钟 SCK 的上升沿发送数据。

I^2C 主、从工作方式下:1 = 输入电平遵照 SMBUS 规范;0 = 输入电平遵照 I^2C 规范。

bit 5,D/$\overline{A}$——数据/地址位(仅用于 I^2C 方式)。

1 = 最后接收或发送的字节是数据;0 = 最后接收或发送的字节是地址。

bit 4,P——停止位(仅用于 I^2C 方式。当 MSSP 被关闭(SSPEN=0)时,该位被清 0)。

1 = 最后检测到停止位(复位时该位为 0);0 = 最后未检测到停止位。

bit 3,S——起始位(仅用于 I^2C 方式。当 MSSP 被关闭(SSPEN=0)时,该位被清 0)。

1 = 最后检测到起始位(复位时该位为 0);0 = 最后未检测到起始位。

bit 2,R/$\overline{W}$—— 读写位信息(仅用于 I^2C 方式)。

该位记录着最后地址匹配后收到的读/写位状态信息。该位仅在地址与下一个开始位、停止位或非$\overline{ACK}$位匹配时有效。

I^2C 从动工作方式下:1 = 读;0 = 写。

I^2C 主控工作方式下:1 = 正在进行发送;0 = 不在进行发送。

它与 SEN,RSEN,PEN,RCEN 或 ACKEN 位一起表示 MSSP 处于空闲(IDLE)状态。

bit 1,UA——地址更新(仅用于 10 bit I^2C 方式)。

1 = 需要更新 SSPADD 寄存器中的地址;0 = 不需要更新 SSPADD 寄存器中的地址。

bit 0,BF——缓冲区满状态位。

接收(SPI 和 I^2C 方式)时:1 = 接收写成,缓冲区(SSPBUF)满;0 = 接收未写成,缓冲区(SSPBUF)空。

发送(仅 I^2C 方式)时:1 = 数据发送正在进行(不包括$\overline{ACK}$位和停止位),缓冲区满;0 = 数据发送已完成(不包括$\overline{ACK}$位和停止位),缓冲区空。

2. 同步串行口控制寄存器 SSPCON(地址 14h)各位的定义

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W 0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7,WCOL——写冲突检测位。

主控方式下:1 = I^2C 条件无效时,向 SSPBUF 写入数据;0 = 未发生冲突。

从动方式下:1 = 正在发送前一个数据时,又有数据写入 SSPBUF 寄存器(必须用软件清 0);0 = 未发生冲突。

bit 6,SSPOV——接收溢出标志位。

SPI 工作方式下:1 = SSPBUF 中仍保持前一个数据时,又收到新的数据。在溢出时,SSPSR 中的数据将丢失。在从动方式下,为了避免产生溢出,即使在发送时,也必须读出 SSPBUF 中的数据;在主控方式下,溢出位不会被置为 1,因为每次操作都是通过对 SSPBUF 寄存器写操作进行初始化的(必须用软件清 0)。0 = 未发生接收溢出。

在 I^2C 工作方式下:1 = SSPBUF 中仍保持前一个数据时,又收到新的数据。在发送方式下,此位无效(必须用软件清 0)。0 = 未发生接收溢出。

bit 5,SSPEN——同步串行口使能位。

SPI 工作方式下(当允许时,这些引脚必须正确地设定为输入或输出引脚):1 = 允许串行口工作,并设定 SCK,SDO,SDI 及$\overline{SS}$为串行口引脚;0 = 关闭串行口,并设定这些引脚为一般 I/O 引脚。

I²C 工作方式下(当允许时,这些引脚必须正确地设定为输入或输出引脚):1 = 允许串行口工作,并设定 SDA 和 SCL 为串行口引脚;0 = 关闭串行口,并设定这些引脚为一般 I/O 引脚。

bit 4,CKP ---时钟极性选择位。

SPI 工作方式下:1 = 时钟高电平为空闲状态;0 = 时钟低电平为空闲状态。

I²C 从动工作方式下(SCK 放弃控制):1 = 时钟工作;0 = 保持时钟线为 0,以确保数据建立时间。

I²C 主控工作方式下:未用。

bit 3~bit 0,SSPM3~SSPM0——同步串行口方式选择位。

0000 = SPI 主控工作方式,时钟 = $f_{osc}/4$;

0001 = SPI 主控工作方式,时钟 = $f_{osc}/16$;

0010 = SPI 主控工作方式,时钟 = $f_{osc}/64$;

0011 = SPI 主控工作方式,时钟 = TMR2 输出/2;

0100 = SPI 从动工作方式,时钟 = SCK 引脚,$\overline{SS}$引脚控制允许;

0101 = SPI 从动工作方式,时钟 = SCK 引脚,$\overline{SS}$引脚控制禁止,SS 引脚用作一般 I/O 引脚。

0110 = I²C 从动工作方式,7 bit 地址;

0111 = I²C 从动工作方式,10 bit 地址;

1000 = I²C 主控工作方式,时钟 = $f_{osc}/(4*(SSPADD+1))$;

1011 = I²C 固件控制为主控工作方式(从动工作方式空闲);

1110 = I²C 固件控制为主控工作方式,带启动位和停止位的 7 bit 地址中断允许;

1111 = I²C 固件控制为主控工作方式,带启动位和停止位的 10 bit 地址中断允许;

1001,1010,1100 及 1101= 保留未用。

下面是一段简单的 SPI 初始化例程,用于利用 SPI 工作方式输出数据的场合。

例 5.2 SPI 初始化程序

```
/* spi 初始化子程序 */
void    SPIINIT()
{
    PIR1=0;                     /* 清除 SPI 中断标志 */
    SSPCON=0x30;                /* SSPEN=1;CKP=0, fosc/4 */
    SSPSTAT=0xC0;
    TRISC=0x00;                 /* SDO 引脚为输出,SCK 引脚为输出 */
}
```

5.2.2　7 段数码显示硬件电路原理图

为了节约引脚，利用单片机强大的 I/O 口扩展能力，采用 PIC16F877 单片机 MSSP 模块的 SPI 方式和移位寄存器芯片 74HC595 实现数码管的静态显示，如图 5.3 所示。实验板上有 8 个共阳极数码管，该图只画出了其中的 2 个。

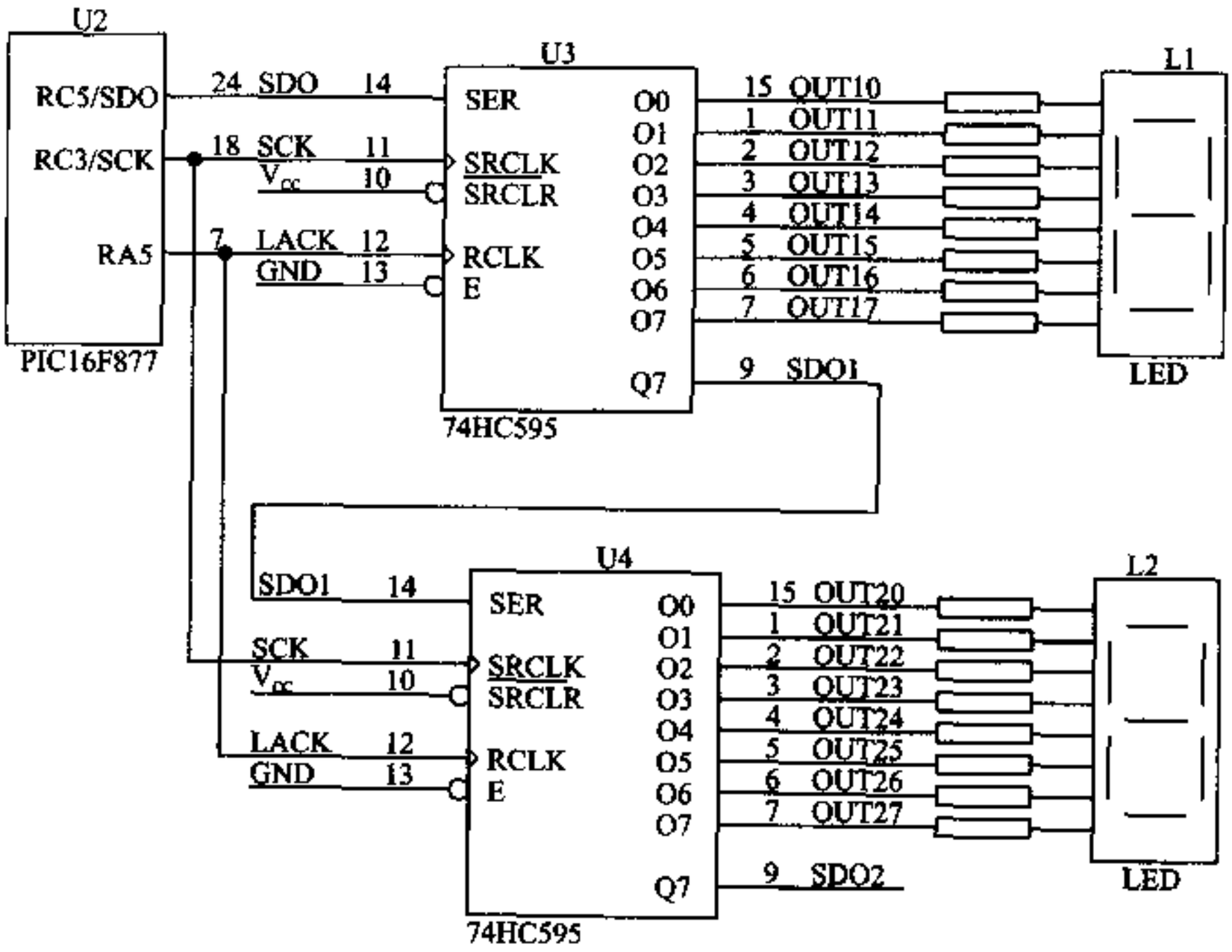


图 5.3　数码显示电路图

74HC595 是一个 8 bit 的移位寄存器，可以把串行输入的 8 bit 数据转换成并行输出。它内部含有 8 bit 串入、并出移位寄存器和 8 bit 3 态输出锁存器。寄存器的时钟输入 SRCLK 和单片机的 SCK 引脚相连，锁存信号 RCLK 与 RA5 相连。当 SRCLK 的电平从低到高跳变时，串行输入数据移入寄存器。当 RCLK 的电平从低到高跳变时，寄存器的数据置入锁存器，锁存器输出到外部器件。

图 5.3 中 SDO 为单片机的 SPI 输出；SCK 为其时钟；LACK 为锁存信号；V_{cc} 为 +5 V 电源电压；GND 接地；SDO1 为第 1 片 74HC595 的输出，接到第 2 片 74HC595 的输入；同理第 2 片的输出 SDO2 又接到第 3 片的输入……，其他引脚的连接方式一样。通过这种级连的方式，把 8 个 74HC595 芯片连在一起，同时每个 74HC595 芯片的并行输出连接 1 个 LED 数码显示器，这样就可以通过简单的程序实现静态显示功能。比如要在 8 个数码管上分别显示 1,2,3,4,5,6,7,8，可以通过 SPI 口连续发送 8 个数据的显示段码，待发送完毕后，通过 LACK 给--

个锁存信号，便可以实现信号输出锁定显示；也可以通过发送一个显示段码数据，给出一个 LACK 锁存信号，实现显示。

5.2.3 程序清单

下面给出已在实验板上调试通过的一个程序，作为编制其他程序的参考。

```
#include          <pic16F87x.h>
/* 该程序用于在 8 个 LED 上依次显示 1~8 等 8 个字符 */
static volatile int table[20]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0x88,0x83,
0xc6,0xa1,0x86,0x8e,0x7f,0xbf,0x89,0xff};
volatile      unsigned      char      data;
#define      PORTAIT(adr,bit)   ((unsigned)(&adr)*8+(bit))      /*绝对寻址位操作指令*/
static      bit PORTA_5 @ PORTAIT(PORTA,5);
/* spi 初始化子程序 */
void          SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;                          /* SSPEN=1;CKP=0 ,fosc/4 */
    SSPSTAT=0xC0;
    TRISC=0x00;                           /* SDO 引脚为输出,SCK 引脚为输出 */
}
/* 系统各输入输出口初始化子程序 */
void          initial()
{
    TRISA=0x00;                           /* A 口设置为输出 */
    INTCON=0x00;                          /* 关闭所有中断 */
    PORTA_5=0;                            /* LACK 送低电平,为锁存做准备 */
}
/* SPI 发送子程序 */
void          SPILED(int data)
{
    SSPBUF=data;                          /* 启动发送 */
    do
    {
    ;
    }while(SSPIF==0);                     /* 等待发送完毕 */
    SSPIF=0;                              /* 清除 SSPIF 标志 */
```

```
}
' * 主程序 * '
main()
{
    unsigned     l;
    initial();                          ' * 系统初始化 * /
    SPIINIT()    ;                      / * SPI 初始化 * /
     for(i=8;i>0;i    -)                / * 连续发送 8 个数据 * /
      {
        data=table[i];                  / * 通过数组的转换获得待显示的段码 * '
        SPILED(data);                   / * 发送显示段码显示 * '
      }
   PORTA_5=1;                           / * 最后给锁存信号,代表显示任务完成 * '
}
```

5.3　利用 I/O 直接扩展键盘

通常情况下,需要用键盘输入参数或对程序的进程进行管理;因此在单片机的应用设计和调试中,键盘是一个不可缺少的部分。键盘的扩展可以采用一些具有特殊功能的数字芯片,如各种移位寄存器等实现。虽然程序较为复杂,但占用单片机的接口较少,可直接用单片机的引脚作为键盘的行、列线;但对单片机 I/O 口的拉电流和灌电流特性有较高的要求。尽管扩展的键盘数量受 I/O 引脚数量的限制,但编程简单,且扩展容易。

5.3.1　键盘输入硬件电路图

本模板中的键盘扩展方法如图 5.4 所示。为了使硬件设计简单化,利用单片机的 RB1,RB2 和 RB4,RB5 进行直接扩展,为矩阵式键盘。

B5 和 B4 与单片机的变位中断输入引脚 RB5 和 RB4 相连;B1 和 B2 与单片机引脚 RB1,RB2 相连,设置为输出。对键盘的输入扫描可采用查询方式或中断方式(单片机的 RB5 和 RB4 可产生变位中断,是微芯公司专门为设计键盘中断功能使用的)。

本节主要介绍键盘工作的查询方式。矩阵式键盘的查询工作原理如下。

如图 5.4 所示,B4,B5 为列线,B1,B2 为行线。列线通过上拉电阻连接到电源上;因此当无键按下时,各列线(B4,B5)均为高电平。当行线(B1,B2)分别输出低电平时,有键按下,相应的列线 B4 或 B5 上会出现低电平。根据此原理,CPU 对整个键盘进行扫描。所谓扫描,即 CPU 不断轮流对行线置低电平,然后检查列线输入状态,确定按键情况。如图 5.4 中,在确定有键按下后,先把 B1 置为低电平,B2 置为高电平,再读入 B4,B5 的值。若 B5 为“1”,B4 为

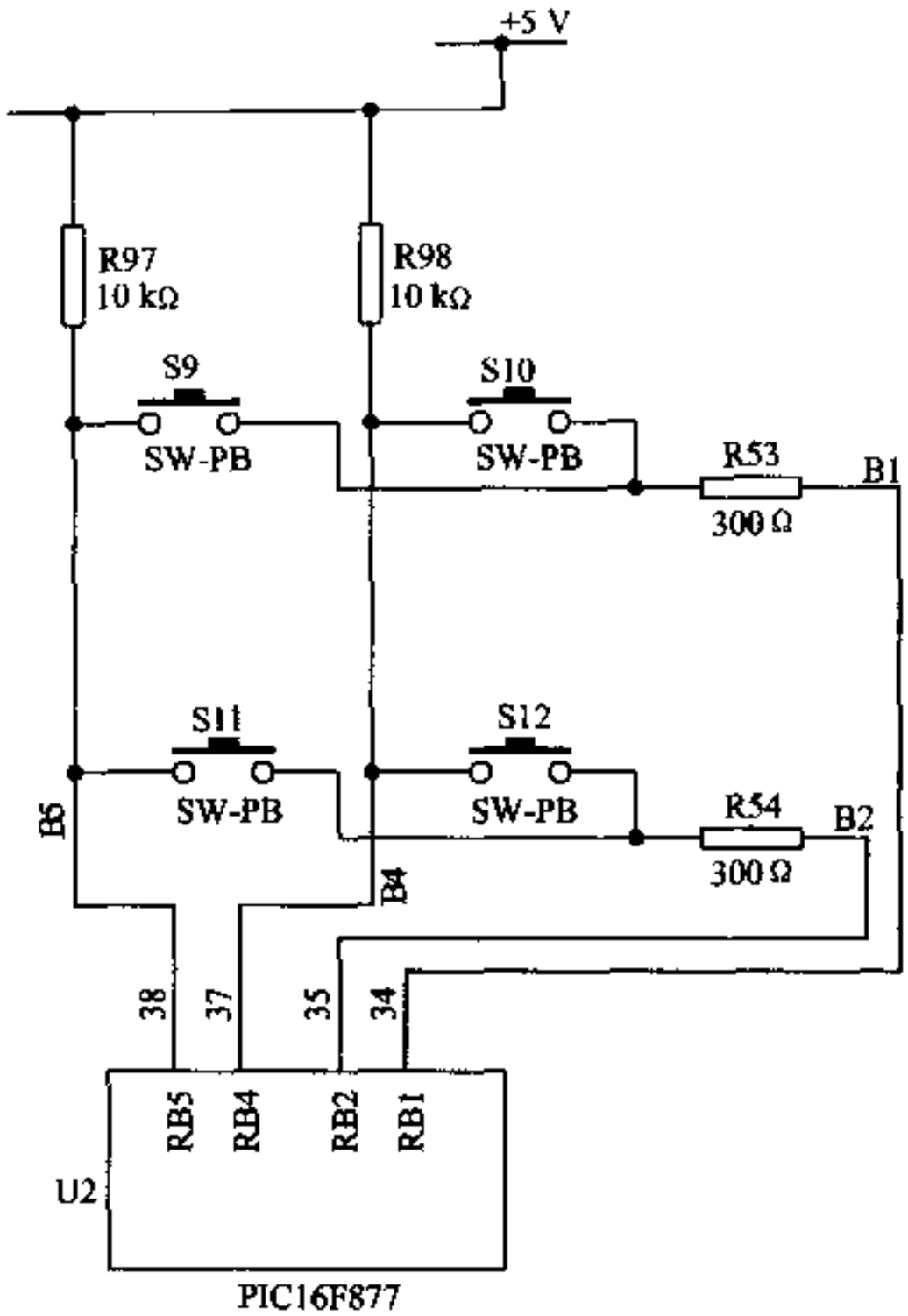


图 5.4 键盘输入电路图

"0",则 S10 键按下;若 B5 为"0",B4 为"1",则 S9 键按下;若 B4,B5 皆为"1",则证明按下的键不在该行,应进行下一行的扫描。下一行扫描时,令 B1 为高电平,B2 为低电平,判断方法同上。扫描键盘的时间很短,仅几微秒;而按键时间一次至少需要几十毫秒,所以只要有键按下,都能被扫描到。按键按下时,有一定的抖动时间;因此在编制程序时,应该用延时消除抖动,以免产生多次按键错误。

5.3.2 键盘管理程序流程图

在键服务子程序中,通过逐行逐列扫描,以确定是哪一键按下,并赋予键值寄存器 j 以不同的键值:若 S9 键按下,则 j=1;若 S11 键按下,则 j=2;若 S10 键按下,则 j=3;若 S12 键按下,则 j=4。键服务子程序返回后,经过判断 j 中所存放的值,可以知道是哪一键按下,从而做出相应的操作。若按键为干扰,则返回值键值 j = 0。在后面程序中通过简单语句判断,若 j 为 0,则证明前面的按键为干扰,面非有效按键,则不对按键进行处理。

该键盘管理流程图代表的只是一种键盘识别的方法，可根据其基本思想编制出自己习惯的程序，也可以在某些细节处做一些变化。

若在大型的控制程序中用到键盘，一般不用软件延时实现消除按键抖动；而是通过编程的灵活性，用执行某一段有效程序的时间实现消除按键抖动。

键盘管理程序流程图如图 5.5 所示。

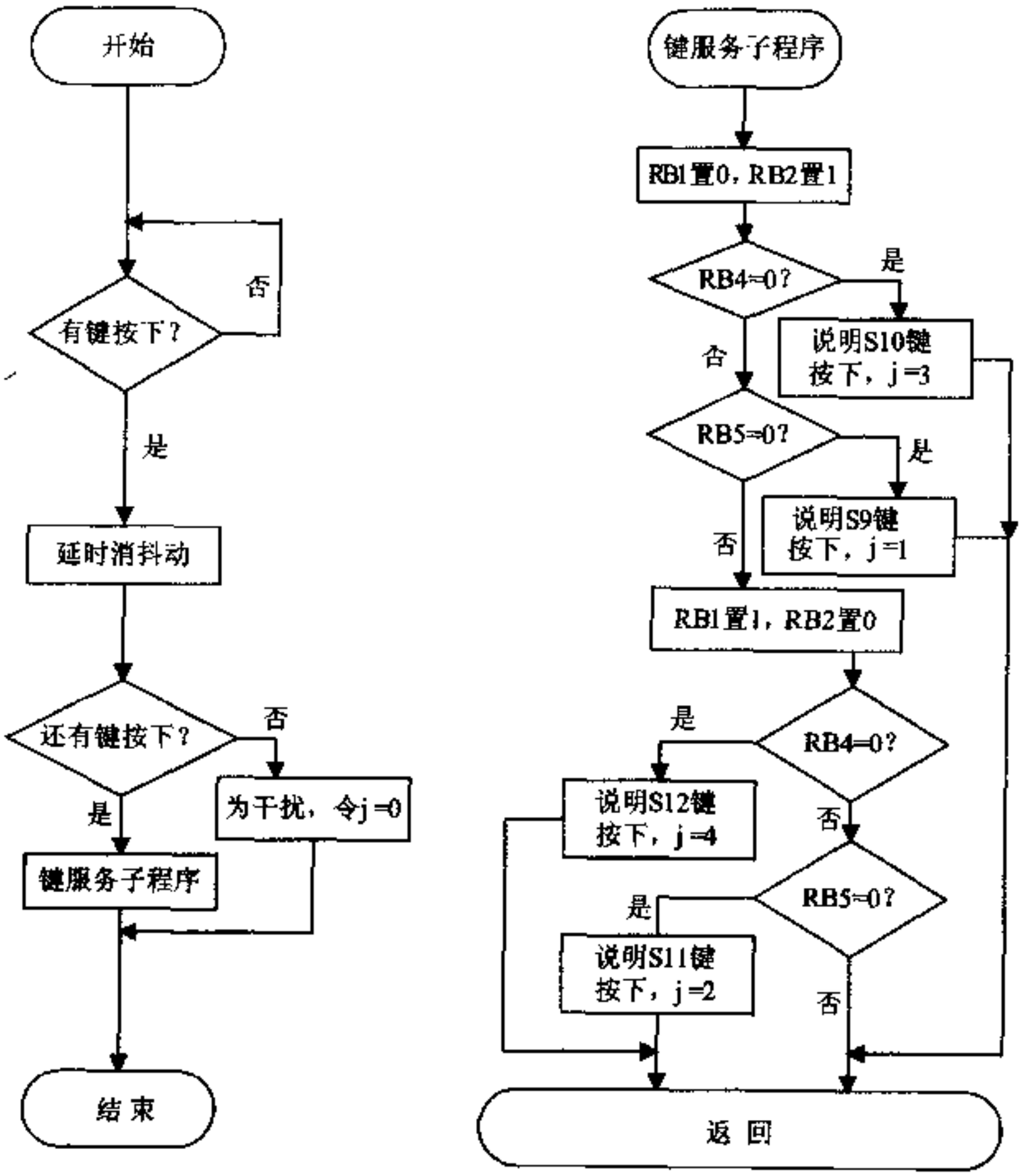


图 5.5　键盘管理程序流程图

5.3.3　程序清单

下面给出已在实验板上调试通过的程序，可作为编制其他程序的参考。

有关显示部分的 SPI 初始化，请读者参考 5.2 节。

```
#include        <pic.h>
 * 该程序用于按下相应的键时,在第1个8段LED上显示相应的1~4的字符 */
#define     PORTAIT(adr,bit)   ((unsigned)(&adr)*8+(bit))        /* 绝对寻址位操作指令 */
static      bit  PORTA_5  @  PORTAIT(PORTA,5);
#define     PORTBIT(adr, bit)     ((unsigned)(&adr)*8+(bit))      /* 绝对寻址位操作指令 *
static      bit PORTB_5 @ PORTBIT(PORTB,5);
static      bit PORTB_4 @ PORTBIT(PORTB,4);
static      bit PORTB_1 @ PORTBIT(PORTB,1);
static      bit PORTB_2 @ PORTBIT(PORTB,2);
unsigned    int   I;
unsigned    char   j;
int   data;
 * spi初始化子程序 */
void            SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;
    SSPSTAT=0xC0;
    TRISC=0xD7;                          /* SDO引脚为输出,SCK引脚为输出 */
}
/* 系统各输入输出口初始化子程序 */
void            initial()
{
    TRISA=0xDF;
    TRISB=0XF0;                          /* 设置与键盘有关的各口的数据方向 */
    INTCON=0x00;                         /* 关闭所有中断 */
    data=0X00;                           /* 待显示的寄存器赋初值 */
    PORTB=0X00;                          /* RB1,RB2 先送低电平 */
    j=0;
}
/* 软件延时子程序 */
void            DELAY()
{
    for(i = 6553; --i  ;)
    continue;
}
 * 键扫描子程序 */
```

```
int          KEYSCAN()
{
while(1)
    {
    if ((PORTB_5==0)||(PORTB_4==0))
    break;
}                                          /* 等待有键按下 */
    DELAY();                               /* 软件延时 */
    if ((PORTB_5==0)||(PORTB_4==0))
        KEYSERVE();                        /* 如果仍有键按下,则调用键服务子程序 */
    else    j=0x00;                        /* 如果为干扰,则令返回值为 0 */
    return(j);
}
/* 键服务子程序 */
int          KEYSERVE()
{
    PORTB=0XFD      ;
    if(PORTB_5==0)     j=0X01;
    if(PORTB_4==0)     j=0X03;
    PORTB=0XFB;
    if(PORTB_5==0)     j=0X02;
    if(PORTB_4==0)     j=0X04;         /* 以上根据按下的键确定相应的键值 */
    PORTB=0X00;                        /* 恢复 PORTB 的值 */
while(1)
    {
        if((PORTB_5==1)&&(PORTB_4==1)) break;   /* 等待键盘松开 */
    }
    return(j);
}
/* spi 发送子程序 */
void              SPILED(int data)
{
    SSPBUF=data;                       /* 启动发送 */
    do
    {
        ;
    }while(SSPIF==0);                  /* 等待发送完毕
```

```
        SSPIF=0;
    }
    /* 主程序 */
    main()
    {
    static  int table[20]  {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0x88,0x83,0xc6,
    0xa1,0x86,0x8e,0x7f,0xbf,0x89,0xff};
        initial();                          /* 系统初始化 */
        SPIINIT();                          /* SPI 初始化 */
    while(1)
    {
        KEYSCAN();
        if(j! =0)                           /* 如果 j=0,证明先前的按键为干扰,则不予显示 */
        {
            data=table[j];
            PORTA_5=0;                      /* LACK 信号清 0,为锁存做准备 */
            SPILED(data);
            PORTA_5=1;                      /* 最后给锁存信号,代表显示任务完成 */
        }
    }
}
```

5.4 利用 PORTB 端口的电平变化中断实现键盘功能

5.4.1 PORTB 端口“电平变化中断”简介

PIC16F877 单片机的 PORTB 端口的 4 个引脚 RB7～RB4 上有电平变化时，可产生中断；但只有当 I/O 引脚设置为输入状态时，才能发生中断(RB7～RB4 的任何一个引脚被配置为输出，则被排除在电平变化比较引起的中断以外)。其功能的实现就是通过把 RB7～RB4 4 个引脚上的输入信号与上次读入的旧值进行比较，如果不等，则把二者异或后输出，产生变化中断，并把标志位 RBIF 置 1(INTCON bit 0)。

这个中断可以把 PIC16F877 从休眠(sleep)状态中激活。在中断服务程序中，可以用以下 2 种方法之一清除中断。

- 对 PORTB 端口进行读操作，这将结束引脚上电平变化的条件。
- 清标志位 RBIF。

按键激活以及其他仅把 PORTB 端口用作电平变化中断的操作中，“电平变化中断”这个

特性很有用。当用到“电平变化中断”这个特性时，不需轮流查询 PORTB 端口。

在这 4 个引脚上，利用 PORTB 端口的“电平变化中断”特性以及软件控制弱上拉电路的特性，可以使之非常方便地与一个矩阵键盘接口，并能用按键唤醒 CPU。这个特性对于用电池供电的应用特别有用，例如遥控器、计算器等。不用时，CPU 的大部分时间都处于低功耗的休眠状态；而一旦有键按下，即可立即唤醒 CPU 工作。

下面给出 PORTB 口“电平变化中断”的简单初始化程序。

例 5.3　PORTB 口“电平变化中断”初始化子程序

```
/* B 口"电平变化中断"初始化子程序 */
void        PORTBINT( )
{
    TRISB=0XF0;                 /* 设置相应口的输入输出方式 */
    OPTION=0x7F;                /* B 口弱上拉有效 */
    PORTB=0X00;                 /* RB1 ,RB2 先送低电平 */
    RBIE=1;                     /* B 口变位中断允许 */
    PORTB=PORTB;                /* 读 B 口的值，以锁存旧值，为变位中断创造条件 */
}
```

5.4.2　硬件电路及原理

在本模板中，对键盘的识别也可以利用 PORTB 端口的“电平变化中断”实现，电路图如图 5.3 所示。

从图 5.3 不难看出，如果程序一开始执行时就使 RB1，RB2 输出低电平，则无论哪一个键按下，都会使引脚 RB4 或 RB5 的电平从高到低发生变化，符合 PORTB 端口发生“电平变化中断”的条件，因面可以实现按键中断。

中断方式与查询方式的区别在于，无键按下时，CPU 不必像查询方式那样需不断地对键盘进行查询，这时 CPU 可以执行其他的任务或进入休眠状态。在利用中断方式时，也必须延时，以消除抖动，并通过逐行逐列扫描等步骤确定是哪一键按下。

在使用“电平变化中断”进行键输入时需特别**注意**：在中断服务程序中进行逐行逐列扫描时，要向 B1，B2 上送高低变化的电平。由于有键按下，这时可能会引起 RB4 或 RB5 引脚上的电平发生变化，使 RBIF 置 1。虽然在中断返回前禁止再次发生中断，但当程序执行中断返回语句 RETFIE 时，就又允许发生中断。由于 RBIF 已经置 1，所以又会发生一次中断，而这次中断是不合法的；因此，在中断返回前，再一次清除中断标志 RBIF，就不会出现再一次中断的问题。

5.4.3　程序清单

下面给出一个调试通过的例程，以供参考。有关显示的部分请参考前面章节。该程序中

寄存器的位都用头文件中定义的位，如 RB5 表示 PORTB 的第 5 位，而不像前面几节那样自己定义。

```
    #include     <pic.h>
/*该程序用于通过 PORTB 的"电平变化中断"进行键盘的识别。
*程序设置一个键值寄存器 j，当按下 S9 键时 j=1，按下 S11 键时
*j=2，按下 S10 键时 j=3，按下 S12 键时 j=4*/
unsigned    char        data;
unsigned    int         I;
unsigned    char        j;
const char table[20]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0x88,0x83,0xc6,
0xa1,0x86,0x8e,0x7f,0xbf,0x89,0xff};
/*B口"电平变化中断"初始化子程序*/
void        PORTBINT()
{
    TRISB=0XF0;                         /*设置相应口的输入输出方式*/
    OPTION=0x7F;
    PORTB=0X00;                         /*RB1,RB2 先送低电平*/
    RBIE=1;                             /*B口变位中断允许 */
    PORTB=PORTB;                        /*读B口的值，为变位中断创造条件*/
}
/*spi 初始化子程序*/
void        SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;
    SSPSTAT=0xC0;
    TRISC=0xD7;                         /*SDO 引脚为输出，SCK 引脚为输出*/
}
/*系统各输入输出口初始化子程序*/
void        initial()
{
    TRISA=0xDF;
    INTCON=0x00;                        /*关闭所有中断*/
    data=0X00;                          /*待显示的寄存器赋初值*/
}
/*键服务子程序*/
void        KEYSERVE()
```

```
{
    PORTB=0XFD      ;
    if(RB5==0)      j=0X01;
    if(RB4==0)      j=0X03;
    PORTB=0XFB      ;
    if(RB5==0)      j=0X02;
    if(RB4==0)      j=0X04;                 /*以上通过逐行逐列扫描,以确定是何键按下*/
    PORTB=0X00;                             /*恢复 PORTB 的值*/
}
/*软件延时子程序*/
void            DELAY()
{
    for(i == 6553; --i  ;)
    continue;
}
/*SPI 发送子程序*/
void        SPILED(int data)
{
    SSPBUF=data;                            /*启动发送*/
    do
    {
        ;
    }while(SSPIF==0);
    SSPIF=0;
}
void            IDEDIS()
{
    KEYSERVE();                             /*进行键盘的识别*/
    data=table[j];                          /*获得需要送出显示的段码*/
    RA5=0;                                  /*LACK 信号清 0,为锁存做准备*/
    SPILED(data);
    RA5=1;                                  /*最后给一个锁存信号,代表显示任务完成*/
}
/*中断服务程序*/
void    interrupt    keyint(void)
{
    DELAY();                                /*软件延时*/
```

```
        if ((RB5==0)||(RB4==0))                    /*该语句除能够确认按键是否为干扰外,*/
                                                   /*还可以屏蔽一次键松开时引起的中断*/
        KDEDIS();                                  /*键识别和显示模块*/
        PORTB=PORTB;                               /*读B口的值,改变中断发生的条件,避免键*/
                                                   /*一直按下时,连续进行键识别*/
        RBIF=0;                                    /*键扫描时可能会产生"电平变化"而使RBIF*/
                                                   /*置1,再清除一次RBIF,以避免额外中断*/
}
main()
{
    initial();                                     /*系统初始化*/
    PORTBINT();                                    /*B口变位中断初始化*/
    SPIINIT();                                     /*利用SPI显示初始化*/
    ei();                                          /*总中断允许*/
while(1)
    {
        ;
    }                                              /*等待中断*/
}
```

5.5 利用 MSSP 模块的 SPI 方式扩展并行输入端口

5.2 节中介绍的"利用 MSSP 模块的 SPI 方式实现与 LED 数码显示接口",实际上是把 MSSP 模块当作一个发送器,利用这种方式可以把数据发送到相应的器件或其他的单片机。SPI 也可以当作接收器,接收来自外部的数据。本节的扩展示例就是利用 74HC165 寄存器与 PIC16F877 的 SPI 方式配合,接收来自外部的数据。

5.5.1 并行输入扩展硬件电路

主同步串行口 MSSP 模块是用来与其他外围串行接口或其他微控制器芯片进行通信的串行接口。这些外围设备可以是串行 EEPROM、移位寄存器、显示器或 A/D 转换器等。MSSP 有以下 2 种工作方式:① 串行外围接口(SPI);② 芯片间总线(I^2C)。

其中 SPI 方式可允许同时同步发送和接收 8 bit 数据,其 4 种工作方式都相似。为了完成通信任务,需用如下 3 个引脚:① 串行数据输出(SDO);② 串行数据输入(SDI);③ 串行时钟(SCK)。

其他有关 SPI 的详细介绍请参考前面章节。

本例主要介绍利用其中的 SPI 方式和 74HC165 芯片实现扩展并行输入。硬件电路如图

5.6 所示。为了能直观地检验系统运行是否正确，把输入的 8 bit 开关量用与 PORTD 口相连的 8 bit LED 显示(8 bit 拨码开关 S13 应拨向正确的位置，即把 LED 与 PORTD 口接通)。

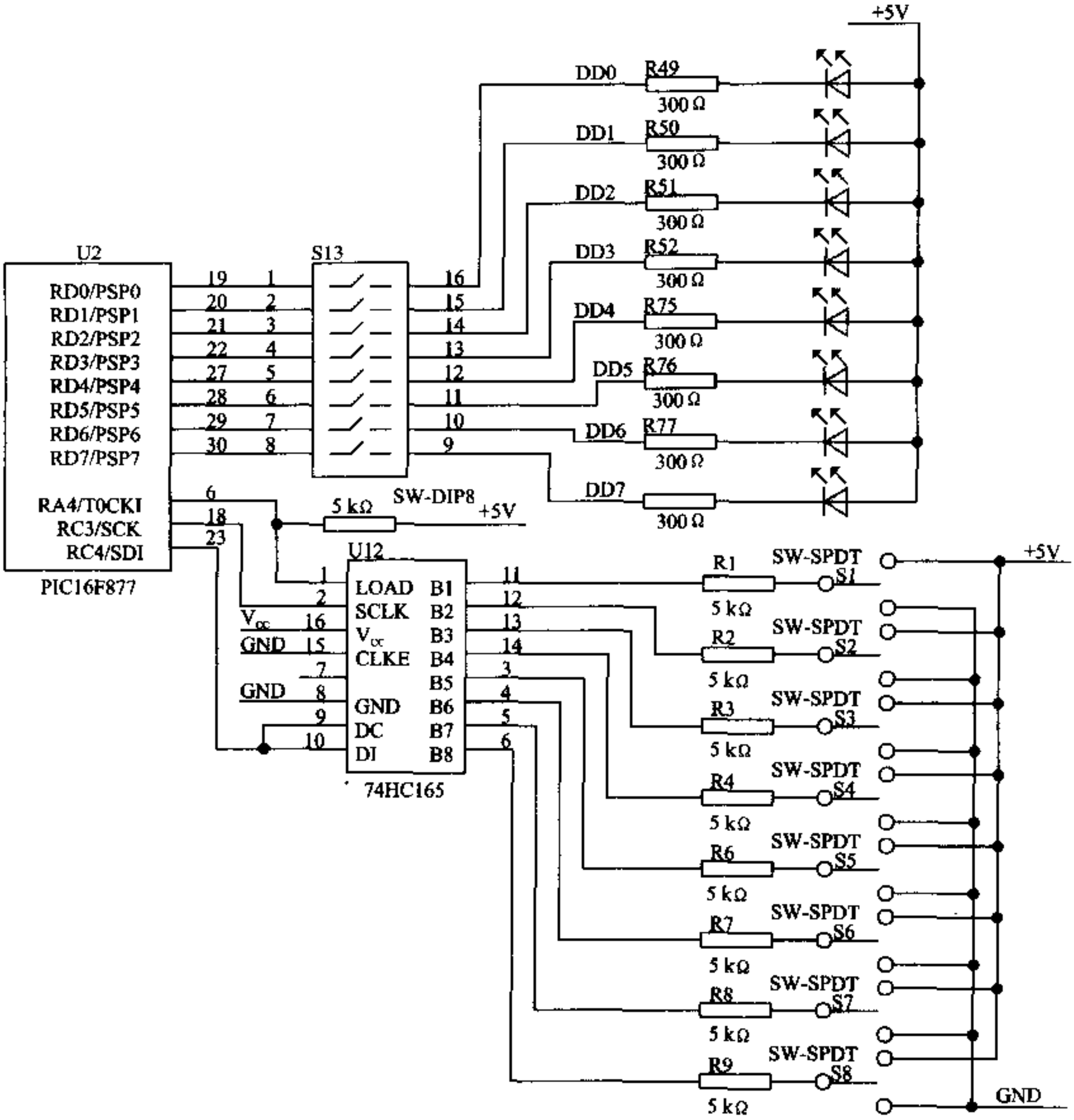


图 5.6　输入输出电路图

图中 74HC165 是一个 8 bit 移位寄存器(并行输入，串行输出)。它有 8 个并行数据输入端(B1～B8)、1 个串行数据输入端(DI)、2 个时钟输入端(SCLK，CLKE)、1 个控制端(LOAD)

及2个反相数据输出。SCLK和CLKE分别作为时钟和时钟禁止,两者的功能可以互换。当两者中有一个为高电平时,另一个时钟功能就被禁止;只有当其中一个为低电平时,另一个时钟功能才被允许。本例中CLKE始终接地,故时钟功能一直允许。该器件具有并行和串行置数功能。并行置数是异步的,当引脚LOAD为低电平时,并行数据锁存到器件内,与时钟的状态无关;接着引脚LOAD置高电平,则74HC165移位置数使能(LOAD为高电平时,芯片才能串行工作),则前面置入的数据由DC引脚串行输出。

还需特别注意,RA4为集电极开路输出,内部没有上拉电阻,要使其输出TTL电平,必须外接一个上拉电阻。PORTD口接在8个LED的阴极,故当相应的位输出高电平时,LED熄灭;输出低电平时,LED点亮。

5.5.2 程序清单

下面给出一个调试通过的例程,可作为参考。调试该程序把模板J7上的短路跳针拔下,以免产生冲突。

```
#include    <pic16F87x.h>
volatile    unsigned    char    data;
/*spi初始化子程序*/
 void    SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;                        /* SSPEN=1;CKP=0,fosc/4 */
    SSPSTAT=0xC0;
    TRISC=0x10;                         /*SDI引脚为输入,SCK引脚为输出*/
}
/*系统各输入输出口初始化子程序*/
void        initial()
{
    TRISA=0x00;
    TRISD=0x00;                         /*D口为输出方式*/
    INTCON=0x00;                        /*关闭所有中断*/
}
/*spi接收子程序*/
 int    SPIIN()
{
    RA4=0;                              /*74HC165并行置数使能,将8 bit开关量置入器件*/
                                        /*(LOAD为低电平时,8 bit并行数据置入74HC165)*/
```

```
    RA4=1;                              /*74HC165 移位置数使能(LOAD 为高电平时,芯*/
                                        /*片才能串行工作)*/
    SSPBUF=0;                           /*启动 SPI,此操作只用于清除 SSPSTAT 的
                                          *BF 位,因此 W 中的实际数据无关紧要*/
    do{
      ;
    }while(SSPIF==0);                   /*查询数据接收完毕否? */
    SSPIF=0;
    data=SSPBUF;
    return(data);                       /*返回接收到的数据*/
}
/*把 SPI 接收的数据通过 D 口显示在 8 个发光二极管上的子程序*/
void        SPIOUT(int data)
{
    PORTD=~data;
}
/*主程序*/
main( )
{
    initial();                          /*系统初始化*/
    SPIINIT();                          /*SPI 初始化*/
 while(1)
  {
    SPIIN();                            /*SPI 接收外部数据*/
    SPIOUT(data);                       /*送出数据显示*/
  }
}
```

5.6　CCP 模块的 PWM 波形产生方法

5.6.1　CCP 模块的 PWM 工作方式简介

PIC16F877 单片机内部集成 2 个 CCP(捕捉/比较/脉宽调制 PWM)模块,当它工作在 PWM 方式下时,具有 2 个脉冲宽度调制输出通道。它们可以产生宽度和周期均可由编程决定的 PWM 波形(下面主要介绍 CCP1 模块的应用,CCP2 模块的应用类似)。当 CCP1 工作在 PWM 方式下时,RC2/CCP1 引脚上可以输出分辨率为 8 bit 或 10 bit 的 PWM 波形,此时必须

将 TRISC 寄存器中的 bit 2 清 0，以设置 RC2/CCP1 引脚为输出状态。

PWM 的输出有一个时基(即周期)和一个保持为高电平的时间，PWM 的频率就是周期的倒数。PWM 的周期可通过向 TMR2 的周期寄存器 PR2 写入来设定，可由如下公式计算：

$$\text{PWM 周期} = ((\text{PR2}) + 1) * 4 * T_{OSC} * (\text{TMR2 前分频值})$$

通过控制 PR2 寄存器的值，就可以控制输出的 PWM 波形的频率。

通过写入 CCPR1L 寄存器和 CCP1CON 控制器的 bit 5～bit 4，可以得到 PWM 的高电平时间设定值，分辨率可达 10 bit：由 8 bit 的 CCPR1L 值(作为 10 bit 中的高 8 bit)和控制寄存器 CCP1CON 中的 bit 5～bit 4 两位(作为 10 bit 中的低 2 bit)组成。用以下方程可以计算 PWM 的高电平时间：

$$\text{PWM 高电平时间} = \text{CCPR1L} : \text{CCP1CON(bit 5～bit 4)} * T_{OSC} * (\text{TMR2 前分频值})$$

在 PR2 一定的情况下，通过控制 CCPR1L 寄存器和 CCP1CON 控制器的 bit 5～bit 4 的值，就可以控制输出的 PWM 波形的占空比。

通过将 CCP1CON 寄存器的 CCP1M3～CCP1M0 位设置成 1100，就可以使 CCP1 模块工作于 PWM 方式。如把 CCP 模块设置为 PWM 操作时，需按以下几步进行：

① 向 PR2 寄存器写入相应的值，以设定 PWM 周期；

② 向 CCPR1L 和控制寄存器 CCP1CON 中的 bit 5～bit 4 两位写入相应的值，以设定 PWM 高电平时间；

③ 通过对 TRISC 的 bit 2 清 0，以设定 RC2/CCP1 引脚为输出状态；

④ 设置 TMR2 的前分频值，并通过向 T2CON 写入，以使 TMR2 使能；

⑤ 设定 CCP1 模块为 PWM 操作。

● **CCP1CON 控制寄存器(地址 17h)和 CCP2CON 控制寄存器(地址 1Dh)**

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
[illegible]	[illegible]	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7～bit 6——未用，读出时为 0。

bit 5～bit 4，CCPxX～CCPxY——PWM 工作循环周期的低 2 bit。

捕捉工作方式：未用；

比较工作方式：未用；

PWM 工作方式：作为其工作循环周期的低 2 bit，高 8 bit 在 CCPRxL 中。

bit 3～bit 0，CCPxM3～ CCPxM0——CCPx 工作方式选择位。

0000 = 关闭捕捉/比较/脉宽调制模块(即 CCPx 复位)。

0100 = 捕捉工作方式，捕捉每个脉冲下降沿。

0101 = 捕捉工作方式，捕捉每个脉冲上升沿。

0110 = 捕捉工作方式，捕捉每 4 个脉冲上升沿。

0111 = 捕捉工作方式，捕捉每 16 个脉冲上升沿。

1000 = 比较工作方式，如果输出匹配使 RC2/CCPx 引脚为高电平(CCPxIF 置 1)。

1001 = 比较工作方式，如果输出匹配使 RC2/CCPx 引脚为低电平(CCPxIF 置 1)。

1010 = 比较工作方式，如果输出匹配产生软件中断(CCPxIF 置 1，CCPx 引脚不受影响)。

1011 = 比较工作方式，特殊事件触发(CCPxIF 置 1；CCP1 将 TMR1 复位；CCP2 将 TMR1 复位，并且启动模/数转换电路(如果 A/D 模块是使能的))。

11xx = 脉宽调制 PWM 工作方式。

关于捕捉/比较/脉宽调制(CCP)模块的其他资料，可参考《中等性能 PIC 系列芯片参考手册》(DS33023)。

PWM 输出经过缓冲可用来驱动直流电机，电动机的转速将正比于 PWM 波形的占空比。PWM 输出波形经过有源低通滤波器后，可以输出基本平滑的直流量，此时 CCP1 模块相当于一个 A/D 转换器。

下面给出一个 CCP 模块设置为 PWM 操作时的初始化程序。

例 5.4　CCP 模块设置为 PWM 方式时的初始化程序

```
/ * CCP1 模块的 PWM 工作方式初始化子程序 * /
void          CCP1INIT()
{
 CCPR1L=0X7F;
 CCP1CON=0X3C;                    / * 设置 CCP1 模块为 PWM 工作方式，且其工作循环
                                    * 的低 2 bit 为 11，高 8 bit 为 01111111=7F * /
 INTCON=0X00;                     / * 禁止总中断和外围中断 * /
 PR2=0XFF;                        / * 设置 PWM 的工作周期 * /
 TRISC=0XFB;                      / * 设置 CCP1 引脚为输出方式 * /
}
```

该初始化子程序设置 CCP1 模块输出分辨率为 10 bit 的 PWM 波形，且占空比为 50%。

5.6.2　硬件电路及原理

本节介绍如何通过编程使 CCP1 模块产生 PWM 波形。PWM 波形直接从单片机的 RC2/CCP1 引脚输出。如果想要改变 PWM 波形的占空比，只需在 CCP 中断服务程序中改变 CCPR1L 寄存器和 CCP1CON 控制器的 bit 5～bit 4 的值，就可改变输出波形的占空比。

5.6.3　程序清单

下面给出一个调试通过的例程，可作为编制程序的参考。

```
#include     <pic.h>
/*该程序用于使 CCP1 模块产生分辨率为 10 bit 的 PWM 波形,占空比为 50% */

/* CCP1 模块的 PWM 工作方式初始化子程序 */
void CCP1INIT()
{
 CCPR1L=0X7F;
 CCP1CON=0X3C;                    /*设置 CCP1 模块为 PWM 工作方式,且其工作
                                   *循环的低 2 bit 为 11,高 8 bit 为 01111111=7F */
INTCON=0X00;                      /*禁止总中断和外围中断*/
 PR2=0XFF;                        /*设置 PWM 的工作周期*/
 TRISC=0XFB;                      /*设置 CCP1 引脚为输出方式*/
}
/*主程序*/
main()
{
     CCP1INIT();                  /*CCP1 模块的 PWM 工作方式初始化*/
      T2CON=0X04;                 /*打开 TMR2,且使其前分频为 0,
                                   *同时开始输出 PWM 波形*/
do
     {
          ;
     }while(1);                   /*系统开始输出 PWM 波形。如果系统是
                                   *多任务的,则可以在此执行其他任务,而
                                   *不会影响 PWM 波形的产生*/
}
```

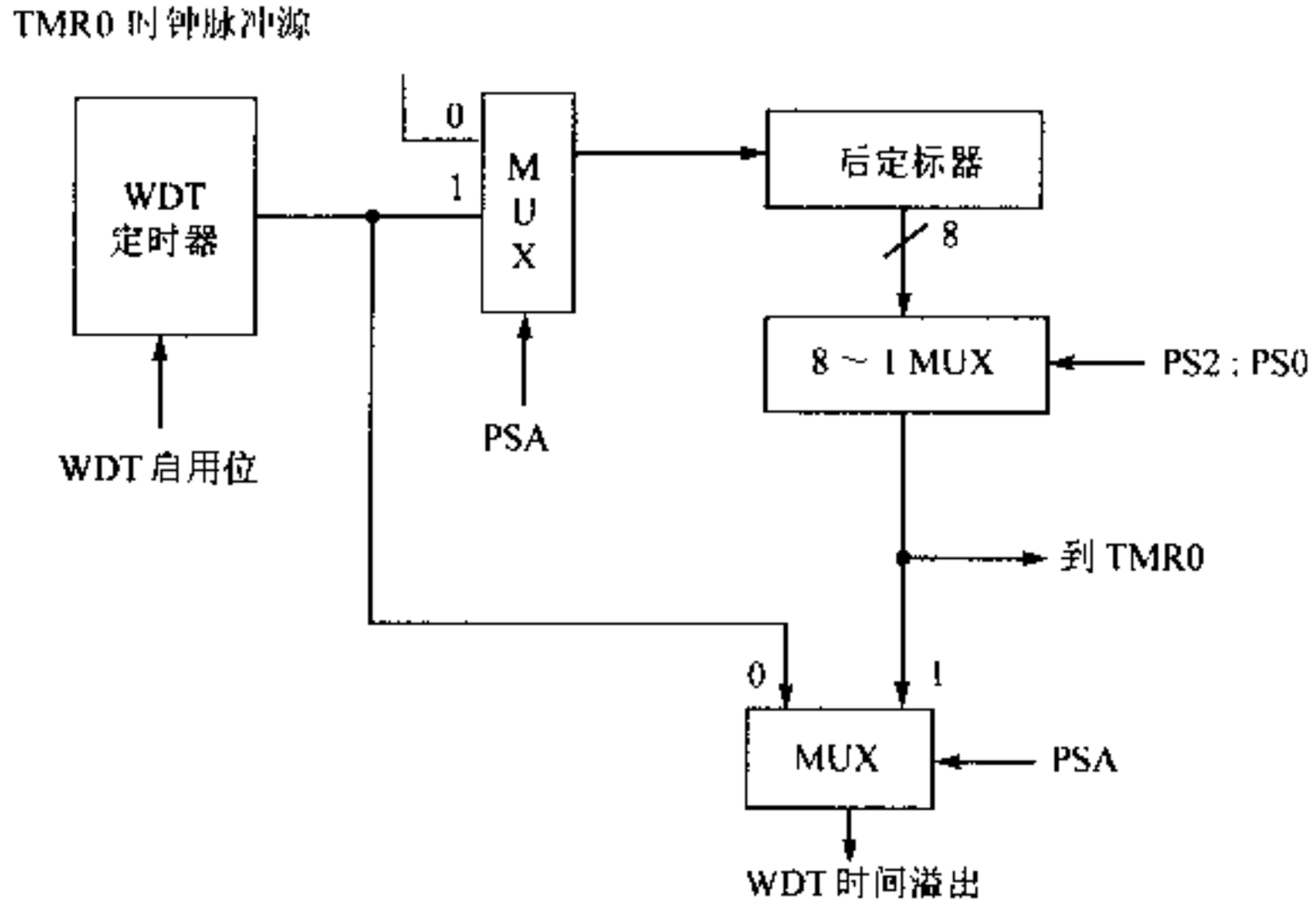
5.7 监视定时器的应用

5.7.1 监视定时器的特点及其原理

监视定时器(WDT)又名“看门狗”,是工业计算机和微控制器中常用的一种电路。在工业现场,往往由于供电电源、空间电磁干扰或其他原因,引起强烈的干扰噪声。这些干扰作用于数字器件,极易使其产生误动作,很可能引起计算机发生“程序跑飞”事故。若不进行有效处理,程序就永远不能回到正常运行状态,从而失去应有的控制功能。监视定时器(WDT)正是为了解决这类问题而产生的,在具有循环结构的程序任务中尤为有效。在正常操作期间,一次

WDT 定时时间到，将产生一次器件复位(监视定时器复位)。通过编制程序使 WDT 定时时间稍大于循环结构程序执行一遍所用的时间，并把语句 CLRWDT 放在循环程序中的某处。如果程序正常运行，就会在 WDT 定时时间到之前对 WDT 清 0，不会产生监视定时器复位；如果由于干扰使程序跑飞，则不会在 WDT 定时时间到之前执行 CLRWDT 语句，WDT 就会产生复位，从而使程序又回到正常运行状态。

一般微控制器的 WDT 电路需用专门的芯片在控制器外部自行搭建，这样 WDT 的稳定性就打了折扣；而 PIC16F877 的 WDT 电路集成在芯片内部，只需用简单的程序语句便可对其进行操作，并且稳定性极好。其结构框图如图 5.7 所示。



注：　PSA 和 PS2:PS0 是 OPTION REG 寄存器中的二进制位

图 5.7　监视定时器结构框图

监视定时器 WDT 计时脉冲由在片内独立的 RC 振荡器产生；因此它不需要任何外部器件。监视定时器的 RC 振荡器独立于接在 OSC1/CLKIN 引脚上的 RC 振荡器，意味着即使器件的 OSC1/CLKIN 引脚和 OSC2/CLKOUT 引脚上的时钟停止，例如，由于执行了“SLEEP”指令而停止，监视定时器 WDT 仍将监视定时。

在正常操作期间，一次 WDT 定时时间到，将产生一次器件复位(监视定时器复位)。如果器件处于休眠状态，一次 WDT 定时时间到，将激活器件，并使之继续进行正常操作(即监视定时器唤醒)。每经过一次监视定时器的定时时间到，STATUS 寄存器中的$\overline{TO}$位将被清 0。

对 WDTE 设置位清 0，可永久性地关闭 WDT。WDT 是系统配置寄存器 *CONFIG* 的 bit2，这个寄存器在程序存储器中映像地址为 2007H；但是这个 2007H 地址已经不在编程存储空间可寻址的范围内，仅仅在下载程序时可以选到。下载程序时，把 MPLAB-ICD 会话窗

口中 Options 菜单中的 Watchdog Timer 选为 On/Enable。WDT 前分频器(实际上是一个后分频器,只是与定时器 0 的前分频器共用,故统称为前分频器)的值可由 OPTION_REG 寄存器分配。OPTION_REG 的 bit 3 为 PSA,当 PSA=1 时,前分频器用于 WDT;当 PSA=0 时,前分频器用于 TMR0。OPTION_REG 的 bit 2 ~ bit 0 为 PS2,PS1,PS0,这 3 位用于前分频器倍率的选择,其对应情况如下:

前分频器位值	TMR0 倍率	WDT 倍率
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

注意,如果把前分频器分配给 WDT,CLRWDT 和 SLEEP 指令将同时对 WDT 和后分频器清 0,从而防止计时溢出引起芯片复位。当前分频器被分配给 WDT 且执行 CLRWDT 指令时,其计数将被清 0;但它的分配不变。

在不加分频器的情况下,WDT 的基本定时时间是 18 ms。这个定时周期还受温度、电源电压及不同器件的工艺参数等的影响。如果需要更长的定时周期,可以通过软件控制 OPTION_REG 寄存器,把预分频器配置给 WDT。这个分频器的分频情况如前所述,可以推出其最大定时周期可达 2.3 s。

5.7.2 硬件电路及原理

该应用是一个死循环模拟实际中程序跑飞的情况。要用到实验模板上的与 PORTD 口相连的 8 个发光二极管,其电路结构请参考本书相关章节。

5.7.3 应用程序

1. 程序流程图

该应用实例的程序流程图如图 5.8 所示。程序上电时,先进行初始化,使 WDT 定一段时间,并选择适当的前分频器倍率;然后清除 WDT,向 D 口送 00H,则与 D 口相连的 8 个发光二极管发光;经过一段延时后,再向 D 口送 FFH,则与 D 口相连的发光二极管熄灭,随后进入死循环。若此时无 WDT 的作用,则 8 个发光二极管不会再亮;若 WDT 起作用,则 8 个发光二极管会再亮。通过此方法,便可以方便地看出 WDT 所起的作用。(与 D 口和 8 个发光二极管的硬件连接图请参考图 5.6)。

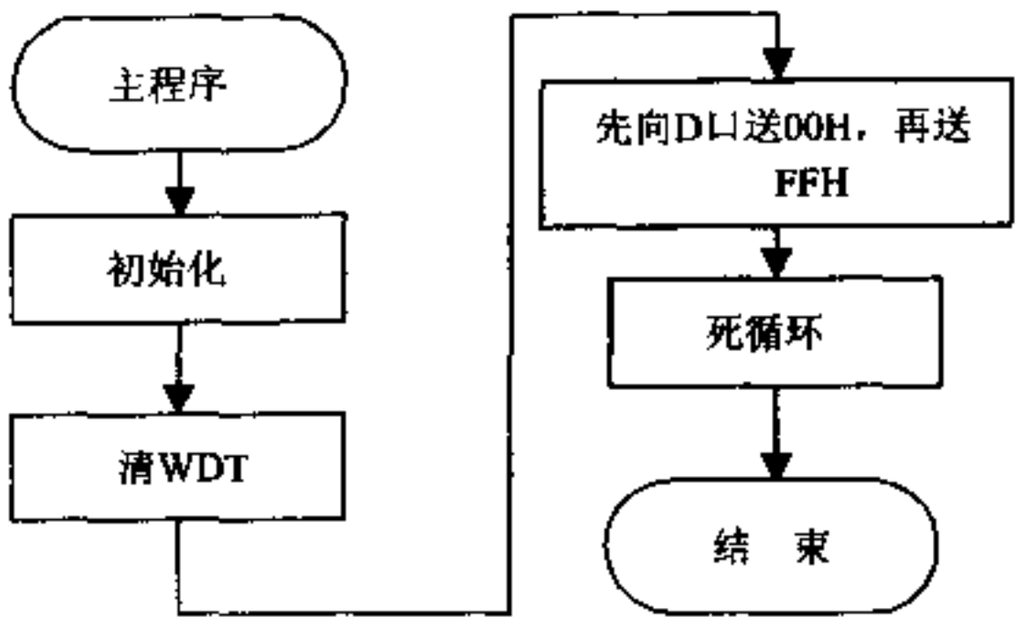


图 5.8　流程图

2. 程序清单

```
#include          <pic.h>
/*此程序实现"看门狗"WDT 的功能*/
unsigned          long          I;
/*系统初始化子程序*/
void              initial()
{
    OPTION = 0X0F;                    /*把前分频器分配给 WDT,且分频倍率为 1:128*
    TRISD = 0X00;                     /*D 口设为输出*/
}
/*延时子程序*/
void              DELAY()
{
 for      (i=19999;--i;)
    continue;
}
/*主程序*/
main ()
{
    initial();                        /*初始化,设定看门狗的相关寄存器*/
    PORTD = 0X00;                     /*D 口送 00H,发光二极管亮*/
    DELAY();                          /*给予一定时间的延时*/
    PORTD = 0XFF;                     /*D 口送 FFH,发光二极管灭*/
while(1)
    {
      ;
```

```
                                    /*死循环.等待看门狗溢出复位*/
```

5.8 休眠工作方式与其激活

5.8.1 PIC16F877的休眠节电工作方式

PIC16F877单片机具有休眠(sleep)工作方式。当其工作在此方式下时,主振荡器关闭、I/O端口保持休眠工作前的状态;因此使系统的功耗减小。该工作方式还具有其他优点,如在休眠工作时,仍可进行A/D转换。由于此时系统主频关闭,数字噪声的干扰大大减小,故A/D转换的精度提高。A/D转换结束中断又可以将其从休眠中唤醒,回到一般工作状态。

单片机在执行一条SLEEP指令后,便进入了休眠方式。此指令如果使能,监视定时器将被清0;但仍然保持运行,$\overline{PD}$位(STATUS寄存器的第3位)清0,$\overline{TO}$位(STATUS寄存器的第4位)置"1",主振荡器关闭。I/O端口保持执行SLEEP命令之前(驱动高、低电平或高阻态)的状态。

只要完成如下事件之一,器件便可从休眠状态中激活。

① 将外部的复位信号输入到引脚$\overline{MCLR}$。

② 监视定时器激活(如果WDT使能的话)。

③ 来自于引脚INT的中断,PORTB端口的电平变化,或者其他一些外围设备的中断。

外部$\overline{MCLR}$复位将引起器件复位;而其他事件都被认为是程序执行的继续,引起一次"激活状态"。以下外围设备的中断可使器件从休眠状态中激活。

① 并行从动端口读写操作。

② 定时器TMR1中断。定时器1此时必须工作在异步计数方式下。

③ CCP捕捉方式中断。

④ 特殊事件触发(在异步方式下,定时器1采用外部时钟)。

⑤ SSP(起始/停止)位探测中断。

⑥ 在从动方式下,SSP发送或接收中断(SPI/I^2C)。

⑦ USART RX或TX(同步从动方式)。

⑧ A/D转换(当A/D时钟源为RC时)。

⑨ EPROM写入操作完成。

其他外围设备不能产生中断将微控制器唤醒,因为他们在休眠状态下不存在片内时钟。

当执行SLEEP指令时,下一个指令(PC+1)被预先取出。为了使器件能通过某一中断事件而激活,与之对应的中断允许位必须被置位(允许)。激活态与GIE位的状态无关。如果GIE位被清0(禁止),器件将继续执行SLEEP指令后面的指令;如果GIE位被置位(允许),器

件将执行 SLEEP 指令后面的指令，然后转移到中断地址(0004h)。如果不希望执行 SLEEP 指令后面的指令，那么在 SLEEP 指令后面必须有一个 NOP 指令。

5.8.2　硬件电路及原理

该例的硬件电路很简单，只用到实验模板上的键盘和 LED 显示器件，可参考本书的相关章节。另外，该例运用按键引起的 B 口的“电平变化中断”将 PIC16F877 从休眠中激活。有关按键引起的 B 口的“电平变化中断”和显示的细节部分，也请参考本书相关章节。

5.8.3　程序清单

该例在 PIC16F877 休眠前使 8 个发光二极管的高 4 个发光，然后进入休眠工作方式；若按键引起的中断将其激活，则低 4 个发光。用 C 语言编写程序时，语句 SLEEP()相当于汇编语言中的语句“sleep”，使单片机进入休眠状态。

```
#include     <pic.h>
/*该程序实现 PIC16F877 的休眠工作方式，并由实验板上的按键产生“电平变化中断”，将其从
 *休眠状态中激活。休眠与激活的状态由与 D 口相连的 8 个 LED 显示。休眠时高 4 个
 *LED 发光，低 4 个 LED 熄灭；激活以后高 4 个 LED 熄灭，低 4 个 LED 发光。*/
unsigned     long     i;
/*系统初始化子程序*/
void initial()
{
    di();                                  /*全局中断禁止，“电平变化中断”只执行唤醒功能*/
    RBIE=1;                                /*PORTB 口电平变化中断允许*/
    RBIF=0;                                /*清除 B 口电平变化中断标志*/
    TRISB4=1;
    TRISB5=1;
    TRISB2=0;
    TRISB1=0;                              /*设置与键盘有关的各 I/O 口的输入输出方式*/
    TRISD=0X00;                            /*D 口为输出*/
    PORTB=0X00;                            /*键盘的行线送低电平，为“电平变化中断”做准备*/
    PORTB=PORTB;                           /*读 PORTB 的值，锁存旧值，也为“电平变化
                                             *中断”做准备*/
}
/*主程序*/
main ()
{
    initial();                             /*初始化*/
```

```
    PORTD=0X0F;                    /*高4个LED灯亮*/
    SLEEP();                       /*单片机开始进入休眠状态*/
    PORTD=0XF0;                    /*激活后,低4个LED灯亮*/
while(1)
    {
        ;
    }
}
```

第 6 章　模拟量输入与输出

6.1　A/D 转换的应用

许多应用中，需把模拟量转换成数字量，再送入单片机处理。PIC16F87X 系列单片机内部集成了 A/D 转换部件，并且有 8 个 A/D 输入通道，只需通过简单的编程，便可以实现单路或多路 A/D 转换的功能。另外其 A/D 转换还可在休眠状态下进行，由 A/D 转换结束中断重新激活单片机。若采用这种工作方式，在 A/D 采样和转换时间内，单片机主频关闭，干扰小，既提高了 A/D 转换的精度，又减少了功耗。A/D 转换部件有以下 4 个寄存器：

(1) A/D 结果高字节寄存器(ADRESH)：用于存放 A/D 转换的数值结果的高字节；

(2) A/D 结果低字节寄存器(ADRESL)：用于存放 A/D 转换的数值结果的低字节；

(3) A/D 控制寄存器 0(ADCON0)：用于控制 A/D 转换器的操作；

(4) A/D 控制寄存器 1(ADCON1)：用于控制选择 A/D 引脚的功能。

当 A/D 转换完成后，共 10 bit 的 A/D 转换结果放在 ADRESH 和 ADRESL 寄存器中，GO/$\overline{\text{DONE}}$位(即 ADCON0 的 bit 2)被清 0，同时 A/D 转换中断标志位 ADIF 被置 1。

1. ADCON1 寄存器各位的定义

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

R=可读位；
W=可写位；
U=不可操作位，读作 0；
-n=上电复位时的值。

bit 7，ADFM——A/D 结果格式选择位。

1=结果右移，ADRESH 寄存器的高 6 位读作“0”；

0=结果左移，ADRESL 寄存器的低 6 位读作“0”。

bit 6～4——读作“0”。

bit 3～0，PCFG3～PCFG0——A/D 转换引脚功能选择位，如表 6.1 所列。

表 6.1　A/D 转换引脚功能选择

PCFG3 PCFG0	AN7* RE2	AN6* RE1	AN5* RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}	CHAN/ REFS
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0

续表 6.1

PCFG3~PCFG0	AN7* RE2	AN6* RE1	AN5* RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}	CHAN/REFS
0001	A	A	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	7/1
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}	5/0
0011	D	D	D	A	V_{REF+}	A	A	A	RA3	V_{SS}	4/1
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}	3/0
0101	D	D	D	D	V_{REF+}	D	A	A	RA3	V_{SS}	2/1
011x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}	0/0
1000	A	A	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0
1010	D	D	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	5/1
1011	D	D	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	4/2
1100	D	D	D	A	V_{REF+}	V_{REF}	A	A	RA3	RA2	3/2
1101	D	D	D	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}	1/0
1111	D	D	D	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2	1/2

注：A 表示模拟输入；D 表示数字 I/O；* 在 28 针引脚芯片上没有这些通道。

当根据需要选择好 A/D 转换模块后，在开始转换之前，必须先选择转换通道。选中的模拟输入通道相应的方向寄存器的 TRIS 位必须被设置为输入。只有当数据采集过程完成之后，A/D 转换才能开始。下面是实现 A/D 转换的步骤。

① 设置 A/D 转换模块。

- 对模拟引脚/基准电压/数字 I/O(ADCON1)进行设置；
- 选择 A/D 输入通道(ADCON0)；
- 选择 A/D 转换时钟(ADCON0)
- 打开 A/D 转换模块(ADCON0)。

② 如需要 A/D 中断功能，设置 A/D 中断。

- 对 A/D 转换完成标志位 ADIF 清 0(ADIF=0)；
- 对 A/D 转换中断允许位 ADIE 置 1；
- 对全局中断允许位 GIE 置 1。

③ 等待模拟量采样完成。

④ 对 GO/$\overline{\text{DONE}}$置 1，启动 A/D 转换。

⑤ 等待 A/D 转换完成，可通过以下 2 种方法之一判断。

- 软件查询 GO/$\overline{\text{DONE}}$位的状态是否为 0；
- 等待 A/D 转换完成中断。

⑥ 读 A/D 结果寄存器 ADRESH 和 ADRESL。如果需要，对 A/D 转换完成中断标志

ADIF 清 0。

⑦ 如果需要进行下一次 A/D 转换，根据要求转入步骤①或步骤②。每一位 A/D 转换时间定义为 t_{AD}。从上一次转换结束到下一次转换开始至少需要等 2 t_{AD}。

2. ADCON0 寄存器各位的定义

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

R＝可读位；
W＝可写位；
U＝不可操作位，读作 0；
－n＝上电复位时的值。

bit 7～6，ADCS1～ADCS0——A/D 转换时钟选择位。
00＝$f_{OSC}/2$；
01＝$f_{OSC}/8$；
10＝$f_{OSC}/32$；
11＝f_{RC}（用 RC 振荡器驱动的时钟）。
bit 5～3，CHS2～CHS0——A/D 模拟通道选择位。
000 选择通道 0，(RA0/AN0)；
001 选择通道 1，(RA1/AN1)；
010 选择通道 2，(RA2/AN2)；
011 选择通道 3，(RA3/AN3)；
100 选择通道 4，(RA5/AN4)；
101 选择通道 5，(RE0/AN5)*；
110 选择通道 6，(RE1/AN6)*；
111 选择通道 7，(RE2/AN7)*。
bit 2，GO/$\overline{\text{DONE}}$——A/D 转换状态位。
如果 ADON＝1：
1＝A/D 转换正在进行，该位置 1，启动 A/D 转换；
0＝未进行 A/D 转换（A/D 转换完成后，该位自动清 0）。
bit 1——读作 0。
bit 0，ADON——A/D 转换允许位。
1＝打开 A/D 转换器在工作状态；
0＝关闭 A/D 转换器，且不消耗工作电流。
注：* 在 28 针引脚封装中没有这些模拟通道。

ADCON0 寄存器（地址为 1FH）用于控制 A/D 模块的操作；ADCON1 寄存器（地址为 9FH）用于对端口的引脚功能进行设置。这些端口引脚可以被设置成模拟输入（RA3 也可当

作参考电压)或数字 I/O 引脚。

在利用 PIC16F877 编制 A/D 转换程序时,需特别注意以下几点。

① 使 ADCON1 寄存器的最高位 ADFM=1,以使 A/D 转换结果右移。ADRESH 寄存器的高 6 bit 读作"0",低 2 bit 存放 10 bit 结果的高 2 bit。

② 本例中,A/D 转换之前,需通过对 ADCON1 的设置使引脚 RA2/AN2 为模拟输入通道;另外为了显示部分的需要,把引脚 RA5 设成 I/O 通道,以输出锁存信号;但是不难发现,如果只对 ADCON1 寄存器写一次控制字,不可能同时使 RA2/AN2 为模拟输入通道、RA5 为 I/O通道。因此,在 A/D 转换前,要对 ADCON1 写一个控制字,在显示锁存输出前,又要对 ADCON1 写另外一个控制字。

③ 每一位的 A/D 转换时间被定义为 t_{AD},完成一次 10 bit A/D 转换所需要的时间最小值为 12 t_{AD}。A/D 转换的时钟源用软件设置进行选择。对于 t_{AD} 可有以下 4 种选择:$2t_{OSC}$,$8t_{OSC}$,$32t_{OSC}$ 及内部 RC 振荡器。为了保证正确地进行 A/D 转换,A/D 转换时钟必须满足最小 t_{AD} 要求,即 $t_{AD} \geqslant 1.6\ \mu s$。本模板用的晶振为 4 MHz,计算可得,需把 A/D 转换的时钟源用软件设置为 $8t_{OSC}$,才满足最低要求,即 ADCS1 : ADCS0=0 : 1。下面给出一个简单的 A/D 转换的初始化程序,设置 RA2 为模拟量输入口。

例 6.1 A/D 转换初始化程序

```
//A/D转换初始化子程序
void        adinitial( )
{
    ADCON0 = 0x51;              //选择 A/D 通道为 RA2,打开 A/D 转换器
                                //在工作状态,且使 A/D 转换时钟为 8tosc
    ADCON1 = 0X80;              //转换结果右移,及 ADRESH 寄存器的高 6 bit 为"0"
                                //且把 RA2 口设置为模拟量输入方式
    PIE1 = 0X00;
    PIE2 = 0X00;
    ADIE = 1;                   //A/D 转换中断允许
    PEIE = 1;                   //外围中断允许
    TRISA2=1;                   //设置 RA2 为输入方式
}
```

6.1.1 7 段数码显示硬件电路图

该应用实例模拟一个简单的报警器,硬件电路如图 6.1 所示。图中只画出了 2 个 LED。

外界模拟电压量从单片机的 RA2/AN2 输入。当输入电压大于 2.5 V 时,系统发出报警信号,与 PORTD 相连的 8 个发光二极管闪动;当电压值恢复到 2.5 V 以下时,发光二极管停止闪动,且只有低 4 个发光。

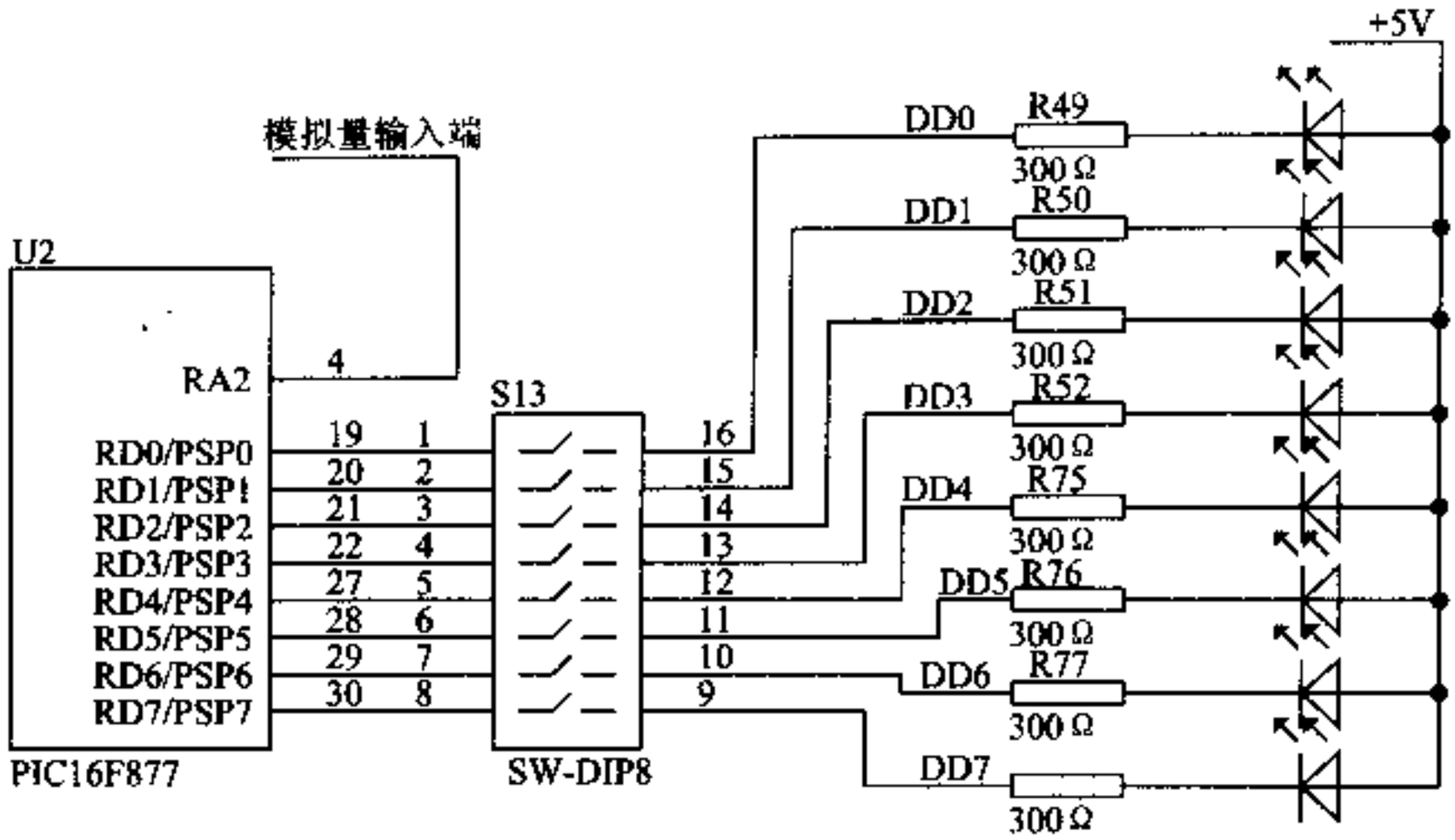


图 6.1　简易报警器电路图

6.1.2　程序清单

下面给出一个调试通过的例程，可作为编制程序的参考。该程序中用共用体的方式把A/D转换的 10 bit 结果组合在一起。有关共用体的详细资料请参考本书相关章节。

```
# include          <pic. h>
union          adres
{int          yl;
unsigned   char   adre[2];
}adresult;                                    //定义一个共用体,用于存放 A/D 转换的结果
unsigned    char      i;
unsigned    int       j;
//系统各 I/O 口初始化子程序
void           initial()
{
    TRISD=0X00;                               //D 口为输出
    i=0x00;
}
//A/D 转换初始化子程序
void           adinitial()
{
    ADCON0=0x51;                              //选择 A/D 通道为 RA2,打开 A/D 转换器
```

```
                                    //在工作状态,且使 A/D 转换时钟为 8tosc
    ADCON1=0X80;                    //转换结果右移,及 ADRESH 寄存器的高 6 bit 为"0"
                                    //且把 RA2 口设置为模拟量输入方式
    PIE1=0X00;
    PIE2=0X00;
    ADIE=1;                         //A/D 转换中断允许
    PEIE=1;                         //外围中断允许
    TRISA2=1;                       //设置 RA2 为输入方式
}
//延时子程序
void        delay()
{
    for(j=5535;--j;)    continue;
}
//报警子程序
void        alarm()
{
    i=i^0xFF;                       //通过异或方式每次把 i 的各位值取反
    PORTD=i;                        //D 口输出 i 的值
}
//中断服务程序
void        interrupt       adint(void)
{
    ADIF=0;                         //清除中断标志
    adresult.adre[0]=ADRESL;
    adresult.adre[1]=ADRESH;        //读取并存储 A/D 转换结果,A/D 转换的结果通过共
                                    //用体的形式放入了变量 y1 中
    if(adresult.y1>0x200)
    {
    alarm();                        //如果输入的模拟量大于 2.5 V(对应数字量
                                    //0x200h),则调用报警子程序
    delay();                        //调用延时子程序,使电压检测不要过于频繁
    }
    else    PORTD=0XF0;             //如果输入的模拟量小于 2.5 V,则与 D 口相连的
                                    //8 个发光二极管的低 4 个发亮,表示系统正常
    ADGO=1;                         //启动下一次 A/D 转换
}
```

```
//主程序
main()
{
    adinitial();                    //A/D转换初始化
    initial();                      //系统各 I/O 口初始化
    ei();                           //总中断允许
    ADG()=1;                        //启动 A/D转换
while(1)
    {
        ;                           //等待中断,在中断中循环检测外部电压
    }
}
```

6.2　MSSP 模块的 I²C 总线方式扩展串行 D/A 芯片

前面章节曾经提到,MSSP 模块除可以工作在串行外围接口(SPI)方式下外,还可以工作在芯片间总线(I^2C)方式下。

6.2.1　I²C 总线工作方式简介

MSSP 模块在 I^2C 方式下可实现所有的主控和从动功能(包括支持通用地址访问),以及硬件上可对启动和结束位进行检测而产生中断的功能,以判断总线(多主机方式)何时空闲。MSSP 模块可在 7 bit 寻址方式或 10 bit 寻址方式下工作。

数据传输用 2 个引脚:SCL 引脚作为时钟线,SDA 引脚作为数据线。当 I^2C 方式使能时,这 2 个引脚自动被定义好。对同步串行允许位 SSPEN(即 SSPCON 的 bit 5)置 1,可允许 SSP 模块工作。

MSSP 模块有以下 6 个寄存器用于 I^2C 操作。

① SSP 控制寄存器 1 :SSPCON1;

② SSP 控制寄存器 2:SSPCON2;

③ SSP 状态寄存器:SSPSTAT;

④ 串行接收/发送缓冲器:SSPBUF;

⑤ SSP 移位寄存器:SSPSR(不可直接访问);

⑥ SSP 地址寄存器:SSPADD。

SSPCON2 寄存器各位的定义如下。SSPCON1 寄存器和 SSPSTAT 寄存器各位的定义请参考相关章节。

● **同步串行口控制寄存器 SSPCON2(地址 91h)各位的定义**

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W 0	R/W 0	R/W 0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

bit 7,GCEN ———般调用使能位(仅用于 I^2C 从动工作方式)。

1 = 收到来自 SSPSR 寄存器的一般调用地址时中断允许;

0 = 一般调用地址关闭。

bit 6,ACKSTAT----确认状态位(仅用于 I^2C 主控工作方式)。

在主控传送方式下:

1 = 未收到来自从动方式下的确认;

0 = 收到来自从动方式下的确认。

bit 5,ACKDT----确认数据位(仅用于 I^2C 主控工作方式)。

在主控接收方式下:当用户接收并初始化确认次序后,将把值发送出去。

1 = 未确认;

0 = 已确认。

bit 4,ACKEN----确认次序使能位(仅用于 I^2C 主控工作方式)。

在主控接收方式下:

1 = 对 SDA 和 SCL 引脚初始化确认次序,并发送 ACKDT 数据位;由硬件自动清 0。

0 = 确认次序空闲。

bit 3,RCEN——接收使能位(仅用于 I^2C 主控工作方式)。

1 = 允许 I^2C 接收方式;

0 = 接收空闲。

bit 2,PEN——停止条件使能位(仅用于 I^2C 主控工作方式)。

1 = 对 SDA 和 SCL 引脚初始化停止条件;由硬件自动清 0。

0 = 停止条件空闲。

bit 1,RSEN----重复启动条件使能位(仅用于 I^2C 主控工作方式)。

1 = 对 SDA 和 SCL 引脚初始化重复启动条件;由硬件自动清 0。

0 = 重复启动条件空闲。

bit 0,SEN——起始条件使能位(仅用于 I^2C 主控工作方式)。

1 = 对 SDA 和 SCL 引脚初始化启动条件;由硬件自动清 0。

0 = 启动条件空闲。

SSPCON1 寄存器用于控制 I^2C 的操作方式。可通过设置 SSPCON1 寄存器的 bit 3 ~ bit 0选择以下几种 I^2C 方式:① I^2C 从动方式(7 bit 地址);② I^2C 从动方式(10 bit 地址);③ I^2C 主控方式。

通信速率 CLOCK=OSC/4(SSPADD+1)

在选择任何一种 I²C 工作方式前，必须设置相应的 TRIS 位，将 SCL 和 SDA 引脚设置为输入。通过将 SSPEN 位置 1，就可选择 I²C 方式。进入 I²C 方式后，SCL 和 SDA 就分别用作 I²C 方式下的时钟线和数据线。

CKE 位(即 SSPSTAT 的 bit 6～bit 7)设置 SDA 和 SCL 引脚在主控或从动方式下的电平规约。当 CKE=1 时，电平服从 SMBUS 规约；当 CKE=0 时，电平服从 I²C 方式。

SSPSTAT 寄存器用于提供数据传输的状态。这些信息包括 I²C 传输“启动(START)”和“结束(STOP)”位的检测，指出接收到的字节是数据还是地址，下一个字节是否是完整的 10 bit地址，以及数据传输是读操作还是写操作。

SSPBUF 是传输数据读写缓冲寄存器。该寄存器将数据移入或移出芯片。接收时，SSPBUF 和 SSPSR 形成一个双缓冲接收器，允许在前一个接收的数据被读取之前，接收下一个数据。当接收到一个 8 bit 数据时，它被送到 SSPBUF 寄存器，同时中断请求标志位 SSPIF 被置 1。如果 SSPBUF 中的数据被读取之前，又接收到下一个 8 bit 数据，就会发生溢出，溢出标志位 SSPOV(SSPCON 寄存器的 bit 6)被置 1，同时 SSPSR 中的数据丢失。

6.2.2　I²C 总线工作方式相关子程序

1. C 语言编写的 I²C 总线工作方式的初始化子程序

```
//I²C 初始化子程序
void     i2cint()
{
    SSPCON = 0X08;              //初始化 SSPCON 寄存器
    TRISC3 = 1;                 //设置 SCL 为输入口
    TRISC4 = 1;                 //设置 SDA 为输入口
    TRISA4 = 0;
    SSPSTAT=0X80;               //初始化 SSPSTAT 寄存器
    SSPADD=0X02;                //设定 I²C 时钟频率
    SSPCON2=0X00;               //初始化 SSPCON2 寄存器
    di();                       //关闭总中断
    SSPIF=0;                    //清 SSP 中断标志
    RA4=0;                      //关掉 74HC165 的移位时钟使能，以免 74HC165 移位
                                //数据输出与 I²C 总线的数据线发生冲突(此操作与该
                                //实验板的特殊结构有关，不是通用的)
    SSPEN=1;                    //SSP 模块使能
}
```

2. C语言编写的I^2C总线工作方式传输数据子程序

需要发送的数据在寄存器j中。

```
//I²C 总线输出数据子程序
i2cout()
{
    SEN=1;                          //产生 I²C 启动信号
    for(n=0x02;--n;)      continue;//给予一定的延时,保证启动
do    {
        RSEN=1;                     //产生 I²C 重启动信号
    }while(SSPIF==0);               //如果没能启动,则反复启动,直到启动为止
    SSPIF=0;                        //SSPIF 标志清 0
    SSPBUF=0X58;                    //I²C 总线发送地址字节
do    {
        ;
    }while(SSPIF==0);               //等待地址发送完毕
    SSPIF=0;                        //SSPIF 标志清 0
    SSPBUF=0X01;                    //I²C 总线发送命令字节
do    {
        ;
    }while(SSPIF==0);               //等待命令发送完毕
    SSPIF=0;                        //SSPIF 标志清 0
    SSPBUF=j;                       //I²C 总线发送数据字节
do    {
        ;
    }while(SSPIF==0);               //等待数据发送完毕
    SSPIF=0;                        //SSPIF 标志清 0
    PEN=1;                          //产生停止条件
do {
        ;
    }while(SSPIF==0);               //等待停止条件产生
    SSPIF=0;                        //SSPIF 标志清 0
}
```

6.2.3 硬件原理电路

在单片机的过程控制和实时控制等应用中,常常需要将数字量转换成模拟量输出;因此D/A转换芯片在此类应用中扮演着很重要的角色。D/A转换芯片种类繁多,有通用廉价的、

高速高精度的、高分辨率的。本模板采用 MAX518 芯片，并通过 I^2C 总线方式与 PIC16F877 接口。

MAX518 芯片是一个带 2 线串行接口的 8 bit 电压输出 D/A 转换芯片，其引脚配置如图 6.2 所示(在模板上短接跳针 J7，可得到此电路)。

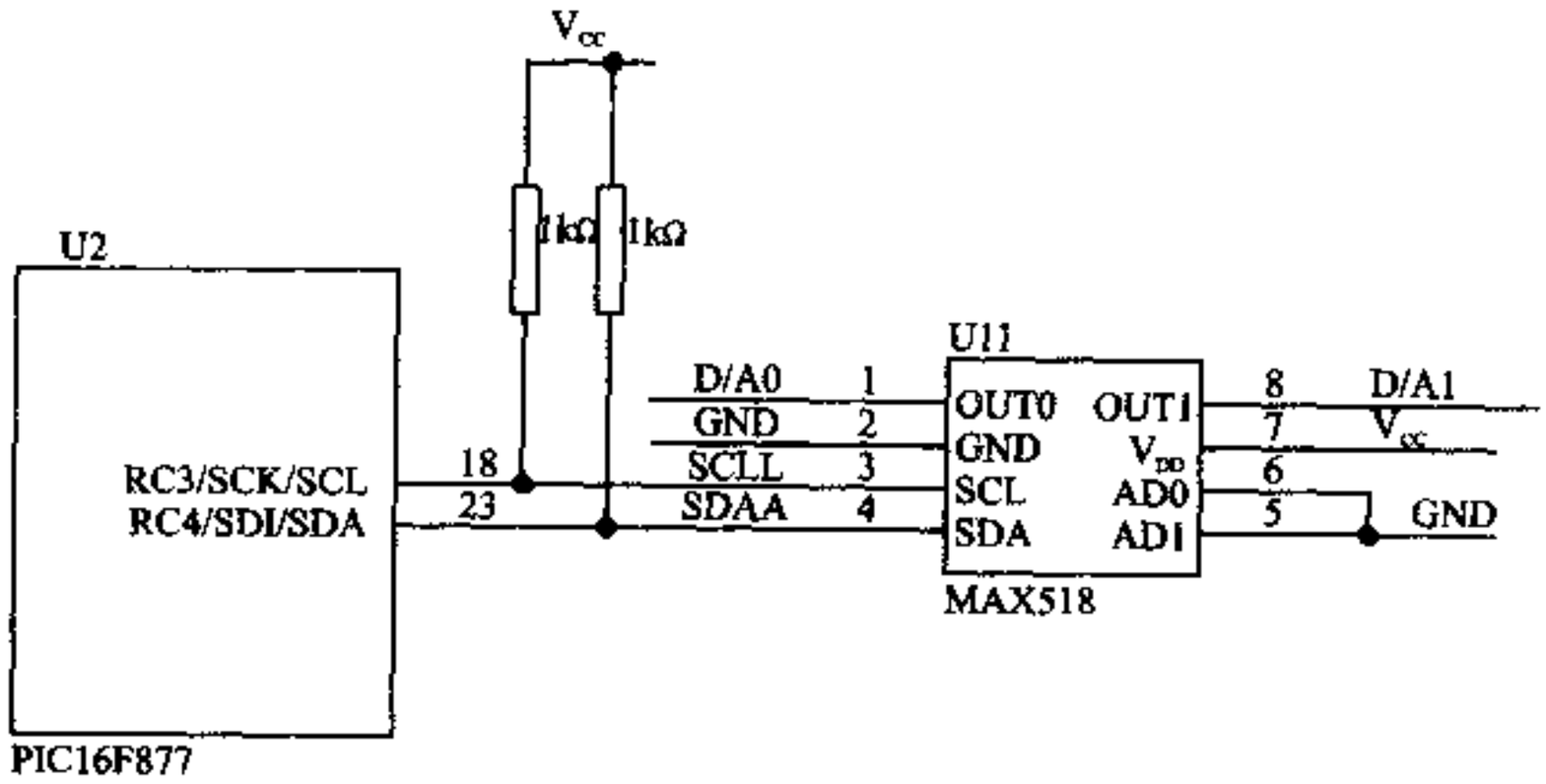


图 6.2　D/A 电路

MAX518 引脚功能如下：D/A0，D/A1 为数/模转换的电压输出；AD0，AD1 为地址输入端，用于设置器件的从动地址；SCL 为串行时钟输入；SDA 为串行数据输入。MAX518 芯片与 PIC16F877 接口时，仅需 2 个引脚。采用 I^2C 总线工作方式驱动 MAX518 芯片时，MSSP 模块在产生启动条件的基础上，连续发送 3 个字节信息，分别是地址字节、命令字节及需要转换的数字量字节。3 个字节发送完毕后，MSSP 模块产生停止条件，MAX518 才开始进行数/模转换。本例中，MAX518 自身的特点和其连接方式决定向它发送的地址字节为 58H，即只有向它发送 58H 地址时，才能选通它；命令字节可以为 00H 或 01H。当为 00H 时，选择 D/A0 通道输出模拟量；当为 01H 时，选择 D/A1 通道输出模拟量；待转换的数据量根据需要可以在 00H～FFH 间变化。当 MSSP 模块产生启动条件、地址字节发送成功、命令字节发送成功、待转换的数据量发送成功、产生停止条件时，SSPIF 都会被置 1；因此在编制程序时，可以通过查询该标志位，确定相应的步骤是否完成(当然也可以用中断的方式)。如果该步骤没有完成，继续等待查询；如果已经完成，清除 SSPIF 标志后，继续进行下一步操作。

图 6.2 中 2 个 1 kΩ 的上拉电阻是为了使 I^2C 总线空闲时，SCL 和 SDA 2 只引脚被拉成高电平。

6.2.4　程序清单

下面是一个例程。该程序利用 MAX518 进行 D/A 转换，且从 D/A0 引脚输出一个正弦波形。该例程可作为编制程序的参考。**特别注意，在调试该程序时，把模板上的钮子开关 S8**

拨向高电平，以免发生资源冲突。

```
#include      <pic.h>
//本程序将通过 PIC16F877 的 I²C 方式驱动 D/A 转换器 MAX518，使其 D/A0 通道输出
//一个连续的正弦波形(注:本程序并没对正弦波的频率进行控制)。
const char table[ ] = {0X80,0X86,0X8D, 0X93,0X99,0X9F,0XA5,0XAB,
0XB1,0XB7,0XBC,0XC2,0XC7,0XCC,0XD1,0XD6,0XDA,0XDF,0XE3,0XE7,0XEA,0XEE,0XF1,
0XF4,0XF6,0XF8,0XFA,0XFC,0XFD,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFE,0XFD,0XFB,
0XF9,0XF7,0XF5,0XF2,0XEF,0XEC,0XE9,0XE5,0XE1,0XDD,0XD8,0XD4,0XCF,0XCA,0XC5,
0XBF,0XBA,0XB4,0XAE,0XA8,0XA2,0X9C,0X96,0X90,0X89,0X83,0X80,0X79,0X72,0X6C,
0X66,0X60,0X5A,0X55,0X4E,0X48,0X43,0X3D,0X38,0X33,0X2E,0X29,0X25,0X20,0X1C,0X18,
0X15,0X11,0X0E,0X0B,0X09,0X07,0X05,0X03,0X02,0X00,0X00,0X00,0X00,0X00,0X00,0X01,
0X02,0X04,0X06,0X08,0X0A,0X0D,0X10,0X13,0X16,0X1A,0X1E,0X22,0X27,0X2B,0X30,0X35,
0X3A,0X40,0X45,0X4C,0X51,0X57,0X5D,0X63,0X69,0X6F,0X76,0X7C};
//以上的数组用于存放正弦表，在定义数组时，前面应该加上 const，
//以使数组存放于 ROM 中，而不至于占用太多的 RAM
unsigned        char    i;
unsigned        char    j;
unsigned        char    n;
//I²C 初始化子程序
void        i2cint()
{
    SSPCON = 0X08;              //初始化 SSPCON 寄存器
    TRISC3 =1;                  //设置 SCL 为输入口
    TRISC4 =1;                  //设置 SDA 为输入口
    TRISA4 = 0;
    SSPSTAT=0X80;               //初始化 SSPSTAT 寄存器
    SSPADD=0X02;                //设定 I²C 时钟频率
    SSPCON2=0X00;               //初始化 SSPCON2 寄存器
    di();                       //关闭总中断
    SSPIF=0;                    //清 SSP 中断标志
    RA4=0;                      //关掉 74HC165 的移位时钟使能，以免 74HC165
                                //移位数据输出与 I²C 总线的数据线发生冲突
    SSPEN=1;                    //SSP 模块使能
}
//I²C 总线输出数据子程序
void        i2cout()
{
```

```
    SEN=1;                          //产生 I²C 启动信号
    for(n=0x02;--n;)      continue;//给予一定的延时,保证启动
do    {
        RSEN=1;                     //产生 I²C 启动信号
    }while(SSPIF==0);               //如果没能启动,则反复启动,直到启动为止
    SSPIF=0;                        //SSPIF 标志清 0
    SSPBUF=0X58;                    //I²C 总线发送地址字节
do    {
        ;
    }while(SSPIF==0);               //等待地址发送完毕
    SSPIF=0;                        //SSPIF 标志清 0
    SSPBUF=0X01;                    //I²C 总线发送命令字节
do    {
        ;
    }while(SSPIF==0);               //等待命令发送完毕
    SSPIF=0;                        //SSPIF 标志清 0
    SSPBUF=j;                       //I²C 总线发送数据字节
do    {
        ;
    }while(SSPIF==0);               //等待数据发送完毕
    SSPIF=0;                        //SSPIF 标志清 0
    PEN=1;                          //产生停止条件
do    {
        ;
    }while(SSPIF==0);               //等待停止条件产生
    SSPIF=0;                        //SSPIF 标志清 0
}
//主程序
main ()
{
    i2cint();                       //I²C 初始化
while(1){
        for(i=0x00;i<=127;++i)
        {
            j=table[i];             //从数组中得到需要传输的数据量
            i2cout();               //利用 I²C 总线方式送出数据
        }
    }
}
```

第7章　秒　表

PIC16F877 单片机有 TMR0，TMR1，TMR2 3 个定时器/计数器，并各自带有前分频器或后分频器。合理地利用这些资源，可以实现复杂的时钟功能，如为一个系统提供精确的系统时钟等。

本章应用实例是利用 TMR0 制作一个高精度的秒表。

该应用只用到实验板上的键盘和 LED 显示，电路图请参考本书关于键盘和 LED 显示的相关章节。定时器在芯片内部。

7.1　工作原理

本例中 PIC16F877 单片机的定时器/计数器 TMR0 产生一个时基，设定为 250 μs，即通过编程使 TMR0 每 250 μs 产生一次中断。

当中断连续发生 40 次后（设置了一个中断次数寄存器 SREG），表示定时 10 000 μs＝10 ms到，10 ms 计时器 S0 加 1；当 S0 超过 9 时，100 ms 计时器 S1 加 1，同时 S0 清 0；当 S1 超过 9 时，秒计时器 S2 加 1，同时 S1 清 0。依次类推，利用 S3，S2，S1，S0 4 个寄存器，便可得到 0～99.99 s 的定时。再通过软件编程，把 S9 键设置为启动计时键、停止计时键及清 0 键（即按下 S9 时开始计时；再按下时停止计时；再按下时使 LED 清 0，等待下一次计时开始，如此循环。其他任意一键都具有此功能），并把计时时间显示在 4 个 LED 上，便得到了一个使用方便的秒表，分辨度为 10 ms＝0.01 s。

但是，考虑到 PIC16F877 单片机的特点，在编程的过程中应特别注意以下几点。

① TMR0 从 FFH 翻转到 00H 时发生中断，计数器继续从 00H 开始计数，而不能自动重装载所给的初值；因此只能在其中断服务程序中重新对其赋初值。

② 写入 TMR0 初值后的 2 个指令周期计数器不能增量，用户必须自己写入调整值。考虑到 TMR0 发生中断时要进行现场保护等问题，不能立即对 TMR0 写入初值；因此在设定初值时，必须考虑在写初值之前所用去的指令周期开销。另外 PIC16F877 单片机的中断响应是固定的 3 个指令周期；因此可以确定中断前和中断后的精确周期数，可以实现精确定时。

如本例中需要 TMR0 每 250 μs 中断一次。假设中断现场保护以及其他语句共消耗 8 个指令周期（用 C 语言编程时会自动进行中断现场保护，可以通过适当的方法查出所消耗的指令周期），以及①中所述的 2 周期滞后和固定的 3 个周期的中断响应时间，经过计算可得到 TMR0 的初值为 256－250＋2＋8＋3＝19＝13H（**注意**：TMR0 为增计数器）。

7.2 程序设计

7.2.1 程序流程图

该流程图只画出了主程序流程图和 TMR0 中断服务流程图。有关显示和键盘的细节，请参考本书的相关章节。

实际编写程序时，为了使键盘不过于灵敏，每次都等到键松开后，再进行相关操作。

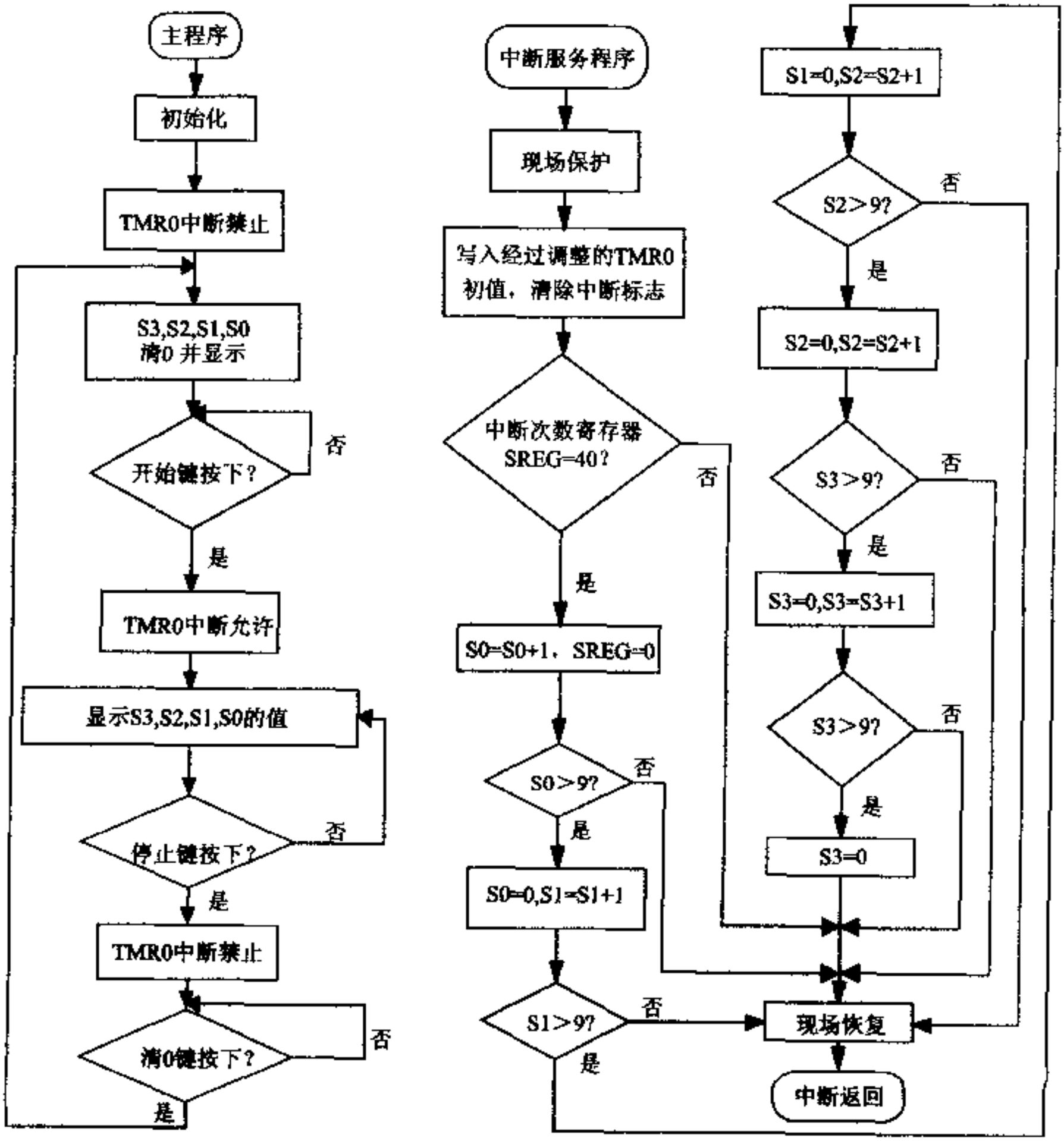


图 7.1 高精度秒表流程图

7.2.2 程序清单

该源程序已在实验板上调试通过，可直接引用，并可利用软件编程的灵活性，加以拓展，实现更为复杂的功能。

```
#include   <pic.h>
#include   <math.h>
//此程序实现计时秒表功能，时钟显示范围 00.00～99.99 s，分辨度：0.01 s
unsigned   char     s0,s1,s2,s3;
//定义 0.01 s,0.1 s,1 s,10 s 计时器
unsigned   char   s[4];
unsigned   char       k ,data ,sreg;
unsigned    int         i;
consttable[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90};
//不带小数点的显示段码表
consttable0[10]={0X40,0X79,0X24,0X30,0X19,0X12,0X02,0X78,0X00,0X10};
//带小数点的显示段码表
//TMR0 初始化子程序
void     tmint()
{
    T0CS=0;                          //TMR0 工作于定时器方式
    PSA=1;                           //TMR0 不用分频
    T0IF=0;                          //清除 TMR0 的中断标志
    T0IE=1;                          //TMR0 中断允许
}
//spi 显示初始化子程序
void     SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;
    SSPSTAT=0xC0;
//设置 SPI 的控制方式，允许 SSP 方式，并且时钟下降沿发送。与"74HC595，当其
//SCLK 从低电平到高电平跳变时，串行输入寄存器"的特点相对应
    TRISC=0xD7;                      //SDO 引脚为输出，SCK 引脚为输出
    TRISA5=0;                        //RA5 引脚置为输出，输出显示锁存信号
}
//系统其他部分初始化子程序
```

```
void    initial()
{
    TRISB1=0;
    TRISB2=0;
    TRISB4=1;
    TRISB5=1;                          //设置与键盘有关的各口的输入输出方式
    RB1=0;
    RB2=0;                             //建立键盘扫描的初始条件
}
//SPI传输数据子程序
void    SPILED(data)
{
    SSPBUF=data;                       //启动发送
    do    {
              ;
        }while(SSPIF==0);
    SSPIF=0;
}
//显示子程序,显示4bit数
void          display()
{
    RA5=0;                             //准备锁存
    for(k=4;k>0;k--)
    {
        data=s[k-1];
        if(k--=3)     data=table0[data];//第2位需要显示小数点
        else    data=table[data];
        SPILED(data);                  //发送显示段码
    }
    for(k=0;k<=4;k++)
    {
        data=0xFF;
        SPILED(data);                  //连续发送4个DARK,使显示好看一些
    }
    RA5=1;                             //最后给锁存信号,代表显示任务完成
}
//软件延时子程序
```

```
void        DELAY()
{
    for(i = 3353;  - -i  ;)     continue;
}
/*键扫描子程序
void        KEYSCAN()

while(1){
while(1)
    {
        display();                          //调用一次显示子程序
        if ((RB5==0)||(RB4==0))     break;

        DELAY();                            //若有键按下,则软件延时
        if ((RB5==0)||(RB4==0))     break;//若还有键按下,则终止循环扫描,返回
    }
*/等键松开子程序
void        keyrelax()
{
while(1){
    display();                              //调用一次显示子程序
    if ((RB5==1)&&(RB4==1))     break;
    }                                       //为防止按键过于灵敏,每次等键松开才返回
}
/*系统赋值初始化子程序
void        initzhi()
{
    s0=0x00;
    s[0]=s0;
    s1=0x00;
    s[1]=s1;
    s2=0x00;
    s[2]=s2;
    s3=0x00;
    s[3]=s3;                            //s0=s1=s2=s3=0,并放入显示缓冲数组中
    sreg=0x00;                          //tmr0 中断次数寄存器清 0
```

```
}
//中断服务程序
void          interrupt     clkint(void)
{
    TMR0=0X13;                          //对 TMR0 写入一个调整值,因为写入 TMR0 后接着的
                                        //2 个周期不能增量,中断需要 3 个周期的响应时间,
                                        //以及 C 语言自动进行现场保护要消耗周期
    T0IF=0;                             //清除中断标志
    CLRWDT();
    sreg=sreg+1;                        //中断计数器加 1
    if(sreg==40)                        //中断次数为 40 后,才对 s0,s1,s2, s3 操作
    {
        sreg=0;
        s0=s0+1;
        if(s0==10){
            s0=0      ;
            s1=s1+1;
            if(s1==10){
                s1=0      ;
                s2=s2+1;
                if(s2==10){
                    s2=0;
                    s3=s3+1;
                    if(s3==10)      s3=0      ;
                }
            }
        }
        s[0]=s0;
        s[1]=s1;
        s[2]=s2;
        s[3]=s3;
    }
    //主程序
    main()
    {
        OPTION=0XFF;
```

```
    tmint();                    //TMR0 初始化
    SPIINIT();                  //spi 显示初始化
    initial();                  //系统其他部分初始化
    di();                       //总中断禁止
    while(1)    {
        inizhi();               //系统赋值初始化
        KEYSCAN();              //键扫描,直到开始键按下
        keyrelax();             //等键松开
        ei();                   //总中断允许
        KEYSCAN();              //键扫描直到停止键按下,在键扫描时有显示
        keyrelax()    ;         //等键松开
        di();                   //总中断禁止
        KEYSCAN();              //键扫描到清 0 键按下,在键扫描时有显示
        keyrelax()    ;         //等键松开
    }
}
```

第 8 章　通用同步/异步通信的应用

计算机与外界的信息交换称为通信。通信方式有 2 种：并行通信和串行通信。本章介绍 PIC 单片机 SCI 口的串行通信方式。

许多 PIC 系列芯片中，其片内除含有同步串行口 SSP(SPI，I^2C)外，还有一个串行通信接口 SCI(serial communication interface)。这是一个通用同步/异步收发器，简称为 USART。USART 可被设置成与个人计算机等进行通信的全双工异步通信系统或半双工异步通信系统，是一种利用 RC6 和 RC7 2 个引脚作为通信线的 2 线制串行通信接口。它可以被定义为如下 3 种工作方式：

① 全双工异步方式；

② 半双工同步主控方式；

③ 半双工同步从动方式。

为了把 RC6 和 RC7 分别设置成串行通信接口的发送(TX)线和接收(RX)线，必须先把 SPEN 位(即 RCSTA 中的 bit 7)和 TRISC 中的 bit 7～bit 6 置 1。

SCI 部件含有 2 个 8 bit 的可读写状态和控制寄存器：发送状态和控制寄存器 TXSTA，接收状态和控制寄存器 RCSTA。其各位定义如下。

1. 发送状态寄存器 TXSTA(地址 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-0	R/W-0
CSRC	TX9	TXEN	SYNC		BRGH	TRMT	TX9D
bit 7							bit 0

R＝可读位；
W＝可写位；
U＝不可编程位，读作 0；
－n＝上电时的值。

bit7，CSRC——时钟源选择位。

异步方式：此位未用。

同步方式：1＝选择主控方式(由内部波特率发生器产生时钟)；0＝选择从动方式(由外部提供时钟信号)。

bit6，TX9——发送数据长度选择位。

1＝选择 9 bit 数据；0＝选择 8 bit 数据。

bit5，TXEN——发送允许位。

1＝允许发送；0＝关闭发送。

注意，在 SYNC 方式下，SREN/CREN 位比 TXEN 位优先级高。

bit4,SYNC——USART 同步/异步方式选择位。

1=选择同步方式;0=选择异步方式。

bit3,此位未用。

bit2,BRGH——高速波特率使能位。

异步方式:1=高速;0=低速。

同步方式:此位未用。

bit1,TRMT——发送移位寄存器(TSR)"空"标志位。

1=TSR 空;0=TSR 满。

bit0,TX9D——发送数据的第 9 位。

2. 接收状态寄存器 RCSTA(地址 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

R=可读;
W=可写;
U=不可编程位;
-n=上电复位时的值。

bit7,SPEN——串行口使能位。

1=允许串行口工作(把 RCT 和 RC6 设置成串行口引脚);0=禁止串行口工作。

bit6,RX9——接收数据长度选择位。

1=选择接收 9 bit 数据;0=选择接收 8 bit 数据。

bit5,SREN——单字节接收允许位。

异步方式:此位未用。

同步主控方式:1=允许接收单字节;0=禁止接收单字节。

注意,在同步从动方式下此位无用。接收完成后,该位即被清 0

bit4,CREN——连续接收选择位。

异步方式:1=允许连续接收;0=禁止连续接收。

同步方式:1=允许连续接收,直到 CREN 位被清 0 为止,CREN 位有效,则 SREN 位无效;0=禁止连续接收。

bit3,ADDEN——地址匹配检测使能位。

同步 9 bit 方式(RX9=1):

1=允许地址匹配检测,允许中断,并且当 RSR 的 D8 被置 1 时,读接收缓冲器的值;

0=禁止地址匹配检测,接收所有字节,第 9 位被看作奇偶校验位。

bit2,FERR——帧格式错标志位。

1=帧格式错(读 RCREG 寄存器可对该位刷新,并且准备接收下一个有效位);

0=无帧格式错。

bit1,OERR—— —越位溢出错标志位。

1=有溢出错,清 CREN 位可将此位清 0;

0=无溢出错。

bit0,RX9D——接收数据的第 9 位,可作奇偶校验位。

8.1　USART 的波特率发生器

USART 带有一个 8 bit 的波特率发生器 BRG,支持 USART 的同步方式和异步方式。用 SPBRG 寄存器控制一个独立的 8 bit 定时器的周期。

在异步方式下,BRGH 位(即发送状态寄存器 TXSTA 的 bit 2)也被用来控制波特率;在同步方式下,用不到 BRGH 位。表 8.1 为主控方式下(内部时钟)不同 USART 工作方式的波特率计算公式。

给出所需的波特率值和时钟频率 f_{osc},就可用表 8.1 中的公式计算出 SPBRG 寄存器中应该写入的最接近的初值,并可计算出波特率的相对误差(其中 X 为 SPBRG 寄存器的值)。

表 8.1　主控方式下的波特率计算公式

SYNC	BRGH=0(低速)	BRGH=1(高速)
0	(异步)波特率=f_{osc}/(64(X+1))	(异步)波特率=f_{osc}/(16(X+1))
1	(同步)波特率=f_{osc}/(4(X+1))	无

例 8.1　计算波特率误差

条件为:f_{osc}= 16 MHz,所要求的波特率=9 600,BRGH= 0(低速),SYNC= 0(异步方式)。

根据公式,波特率= f_{osc}/(64(X+1))

即　　9 600= 16 000 000/(64(X+1))

X= 25.042

取　　X= 25

计算　实际波特率=16 000 000/(64(25+1))=9 625

所以　相对误差=(实际波特率—期望波特率)/期望波特率=

(9 615—9 600)/9 600=

0.16%

因为在某些情况下用高速方式(BRGH= 1)下的波特率= f_{osc}/(16(X+1))可减小误差,所以用高速方式下的公式有一定的优越性,即使所需要的是低波特率。

向波特率寄存器 SPBRG 写入一个新值,会使 BRG 定时器复位(或清 0);由此可保证 BRG 不需等到定时器溢出就可输出新的波特率。

8.2 USART 的异步工作方式

在异步工作方式下，串行通信接口 USART 采用标准的不归 0(NRZ)格式(1 bit 起始位、8 bit 或 9 bit 数据位及 1 bit 停止位)，最常用的数据格式是 8 bit。片内提供的 8 bit 波特率发生器 BRG 可驱动振荡器的时钟，产生标准的波特率频率。USART 发送和接收顺序从最低位(LSB)开始。USART 的发送器和接收器在功能上是独立的；但它们所用的数据格式和波特率必须是相同的。波特率发生器可根据 BRGH 位(TXSTA 寄存器的 bit 2)的设置产生 2 种不同的移位速度：对系统时钟 16 分频和 64 分频的波特率时钟。USART 硬件不支持奇偶校验(parity)；但可以用软件实现(并可作为第 9 位数据传输)。在 CPU 处于休眠工作方式时，USART 不能用异步方式工作。

通过对 SYNC 位(TXSTA 寄存器的 bit 4)清 0，可选择 USART 异步工作方式。USART 异步工作方式由以下一些重要部件组成：波特率发生器 BRG、采样电路、异步发送器、异步接收器。

8.2.1 USART 异步发送器

图 8.1 是 USART 异步发送器的结构示意图。发送器的核心是发送移位寄存器(TSR)。

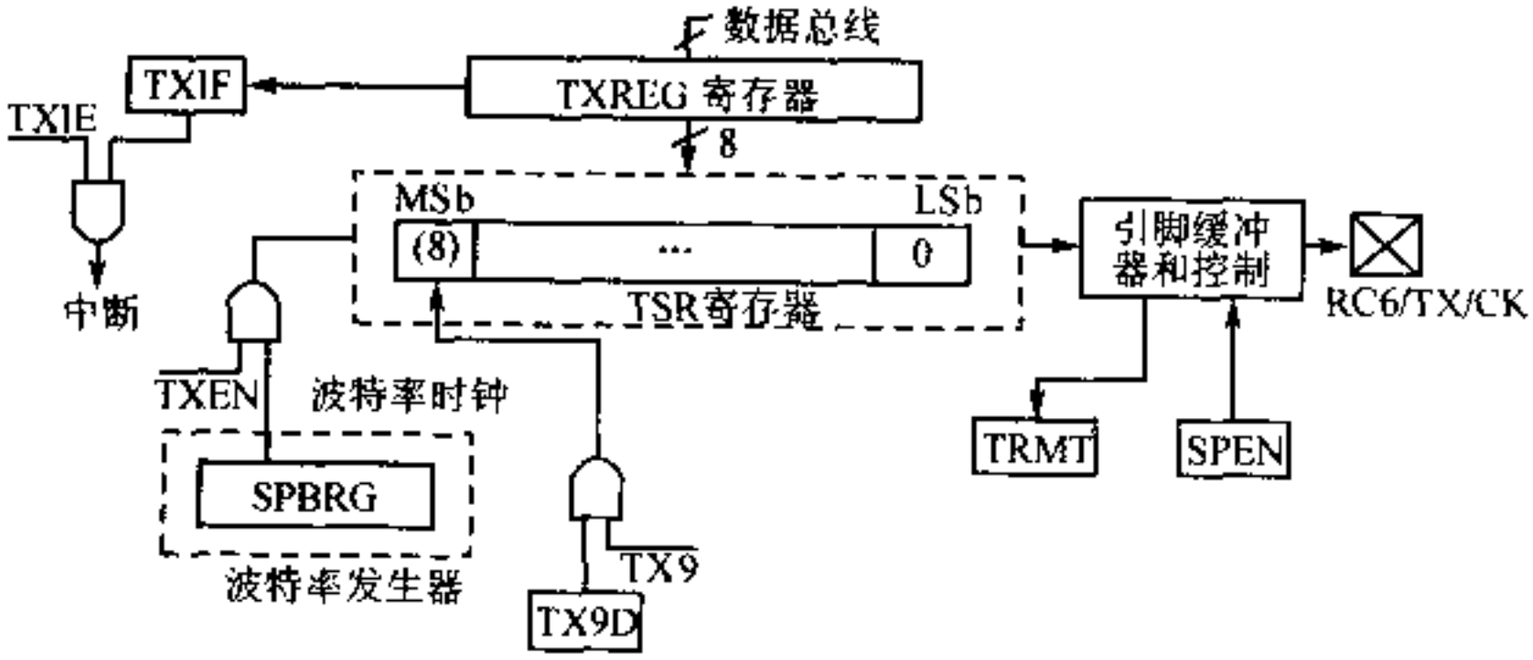


图 8.1 USART 异步发送器结构示意图

发送移位寄存器从发送缓冲器 TXREG 获得要发送的数据。TXREG 由用户软件装入数据，TSR 中前次装入的数据发送出去之后，寄存器 TXREG 中的数据就被装入 TSR。一旦把 TXREG 中的数据送入 TSR(在 1 个时钟周期里)，TXREG 就处于空闲状态，同时发送中断标志位 TXIF(PIR1 的 bit 4)被置 1。这个中断是否被 CPU 响应，可通过设置发送中断允许位 TXIE(即 PIE 的 bit 4)来决定。不管 TXIE 的状态如何，中断标志位 TXIF 都被置 1，并且 TXIF 位不能用软件清 0。只有当新的发送数据送入 TXREG 寄存器后，TXIF 位才会由硬件

复位。用 TXIE 位表示 TXREG 的状态，而用 TRMT 位(即 TXSTA 状态寄存器的 bit 1)表示 TSR 的状态。TRMT 是一个只读位，当 TSR 寄存器为空时，TRMT 位被置 1。TRMT 位与中断逻辑没有任何联系，所以如需确定 TSR 是否为空，就必须通过对这一位不断查询判断。异步发送方式的具体实现步骤见后面例程。

8.2.2　USART 异步接收器

图 8.2 是 USART 异步接收器的结构原理框图。从 RC7/RX/DT 引脚接收串行信号，并送入数据收集器。数据收集器实际上是一个以 16 倍波特率工作的高速移位寄存器，而主接收器的串行移位操作是以时钟频率 f_{osc} 的速率工作。

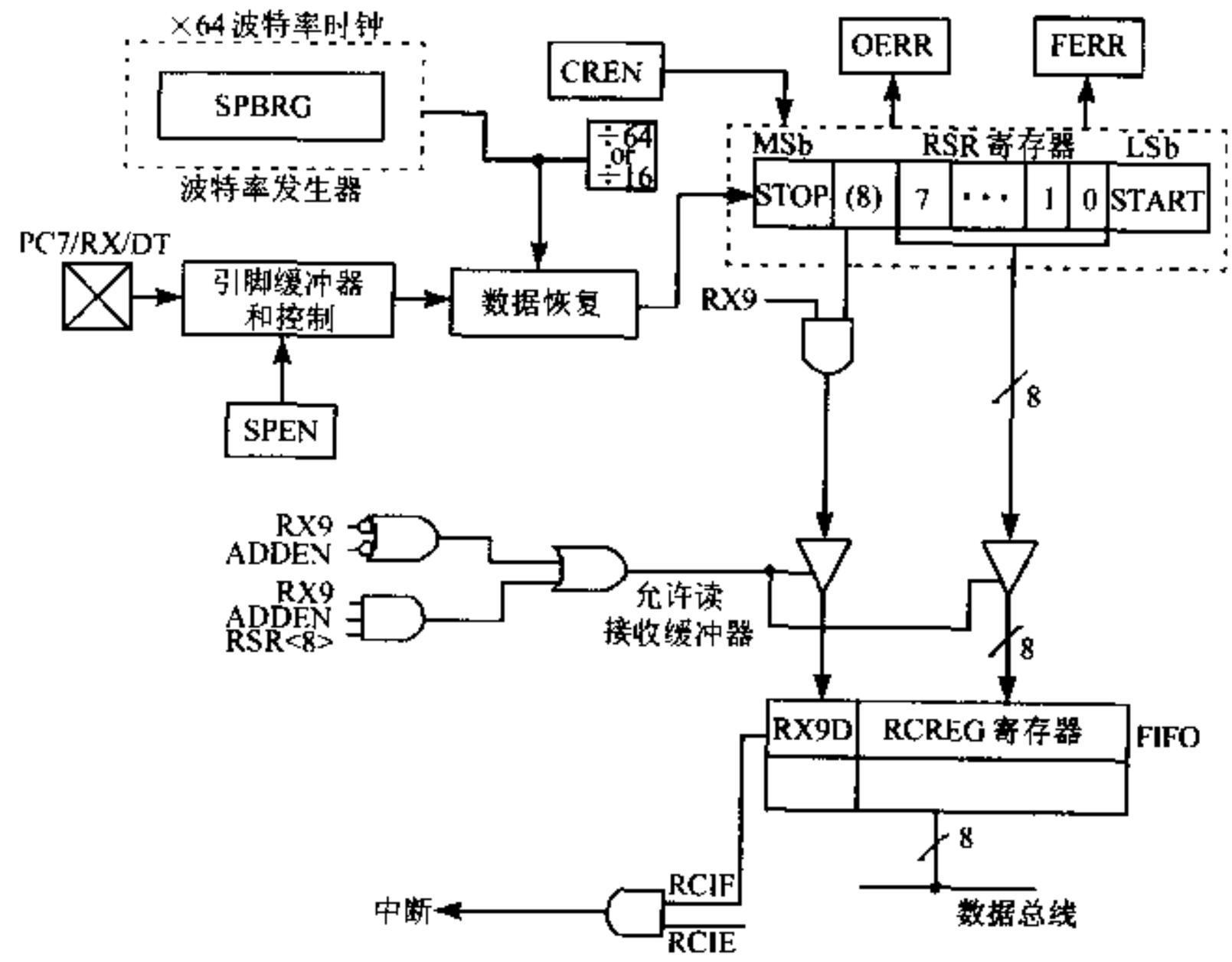


图 8.2　USART 异步接收器结构示意图

当 RCSTA 寄存器的 RX9 位被置 1 时，USART 异步接收器可以接收 9 bit 数据，且接收到的第 9 位数被放在 RCSTA 寄存器的 RX9D 位。

一旦选择异步方式，把 CREN(RCSTA 寄存器的 bit 4)位置 1，就进入接收工作状态。

8.3　USART 的同步主控方式

在同步主控方式下，数据的传输以半双工的方式进行，即发送和接收不是同时进行的，发

送数据时，接收数据被禁止，反之亦然。把 SYNC 位(TXSTA 状态寄存器中的 bit 4)置 1，就可进入同步工作方式。把 CSRC 位(TXSTA 中的 bit 7)置 1，即进入 USART 主控方式。另外还应把 SPEN 位(RCSTA 中的 bit 7)置 1，以把 RC6 和 RC7 引脚分别设定为时钟线 CK 和数据线 DT。主控方式意味着 CPU 在 CK 线上发送时钟信号。

8.3.1 USART 同步主控发送

USART 同步主控发送方式与异步发送方式过程基本相同，具体步骤见程序。

8.3.2 USART 同步主控接收

一旦选择同步方式，只要把 SREN 位(RCSTA 寄存器中的 bit 5)或 CREN 位(RCSTA 寄存器中的 bit 4)置 1，即可进入同步主控接收状态。DT 数据线上的信号在时钟的下降沿被采样。如果 SREN=1，仅接收 1B；如果 CREN=1，则可连续地接收数据，直到 CREN 被清 0 为止；如果 SREN 和 CREN 都被置 1，则 CREN 状态优先而进行连续接收。

8.4 USART 的同步从动方式

同步从动方式与主控方式不同，其移位时钟信号(CK)在 CK 引脚上由外部提供(主控方式由内部提供移位时钟)。这样就允许在 CPU 体眠(sleep)状态下发送或接收数据。把 CSRC 位(TXSTA 状态寄存器的 bit 7)清 0，即可进入从动方式。

8.4.1 USART 同步从动发送

除在休眠方式下外，同步主控发送方式和从动发送方式的操作是一样的。

如果向缓冲器 TXREG 写入 2 个要发送的数据后，再执行 SLEEP 指令，则会发生以下事件。

① 第 1 个数据将立即被送入移位寄存器 TSR 进行发送。

② 第 2 个数据仍保存在 TXREG 中。

③ 中断标志位 TXIF 将不会被置 1。

④ 当第 1 个数据已经移出 TSR，TXREG 中的第 2 个数据再送入 TSR 中时，TXIF 才会被置 1。

⑤ 如果中断允许位 TXIE 为 1，将把芯片从休眠状态下唤醒。如果总中断也是开放的，那么程序就转入中断矢量(0004h)，执行中断服务程序。

8.4.2 USART 同步从动接收

除单片机工作在休眠方式下外，同步从动接收和同步主控接收的操作基本上是一样的；另

外，在从动接收方式下，没有用到 SREN 位。

如果在执行 SLEEP 指令之前已设置好处于接收状态(CREN＝1)，那么休眠方式下仍可以接收数据。当接收到完整的数据字节后，RSR 接收移位寄存器中的数据将被送入 RCREG 寄存器中，并把接收中断标志位 RCIF 置 1。如果中断允许，即 RCIE＝1，将把单片机从休眠状态下激活；如果总中断也是开放的，那么程序就转入中断矢量(0004h)，执行中断服务程序。

8.5　单片机双机异步通信

8.5.1　单片机双机异步通信硬件连接

本例采用 PIC16F877 单片机。实现 PIC16F877 的异步串行通信的硬件接口电路可采用如图 8.3 所示的电路。图中 2 片 PIC 采用相同的晶振(本例中采用 4 MHz 晶振)，TX 为发送线，RX 为接收线。

单片机间的通信也可先用 MAX232 将各自的输出信号转换为标准的 RS-232 电平信号，利用 MAX232 芯片将输入信号转换为单片机所需的信号。这样有利于单片机间信号的稳定传输，而在软件编程上却无任何变化。

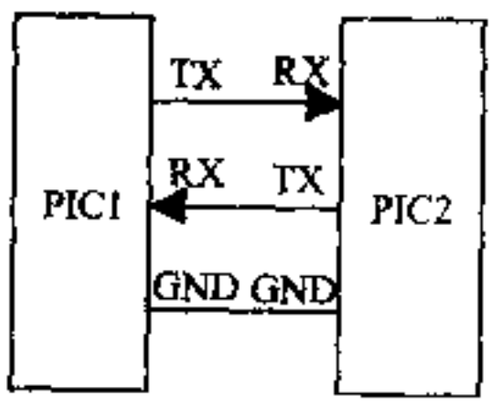


图 8.3　双机通信示意图

8.5.2　单片机双机异步通信编程

下面给出用 C 语言编制的单片机双机异步通信源程序。

规定双机通信程序任务为：先在 PIC1 中将一个数组 tran[8]赋予一定的值，再将这个数组的内容送给 PIC2 相应的接收数组 rece[8]；完成传输后，2 个 PIC 同时将这些单元中的内容显示出来。为节省篇幅，省去了与显示有关的硬件接口电路的介绍，请参考本书相关部分。为保证传输的准确性，当 PIC2 接收到 1 字节后，马上发送回一个响应字节到 PIC1，PIC1 接收到响应字节后，才再发送下一字节。这个握手的过程确保不会因为溢出错误面丢失数据。

1. 单片机 PIC1 编程(发送部分)

```
#include <pic.h>
/*该程序实现单片机双机异步通信功能,该程序是发送部分*/
unsigned char tran[8];                    /*定义 1 个数组存储发送数据*/
unsigned char k,data;                     /*定义通用寄存器*/
const char table[20]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0x88,0x83,0xc6,
0xa1,0x86,0x8e,0x7f,0xbf,0x89,0xff};
/*不带小数点的显示段码表*/
/*spi 显示初始化子程序*/
void    SPIINIT()
```

```
{
    PIR1=0;
    SSPCON=0x30;
    SSPSTAT=0xC0;
/*设置 SPI 的控制方式,允许 SSP 方式,并且时钟下降沿发送,与"74HC595,当其
*SCLK 从低电平到高电平跳变时,串行输入寄存器"的特点相对应*/
    TRISC=0xD7;                    /*SDO 引脚为输出,SCK 引脚为输出*/
    TRISA5=0;                      /*RA5 引脚设置为输出,以输出显示锁存信号*/
}
/*给数组赋初值子程序 */
void    fuzhi()
{
    for(k=0;k<8;k++)    {
        tran[k]=k+3;
    }
}
/*SCI 部件初始化子程序*/
void    sciint()
{
    SPBRG=0X19;                    /*将传输的波特率设为约 9 600 bps*/
    TXSTA=0X04;                    /*选择异步高速方式传输 8 bit 数据*/
    RCSTA=0X80;                    /*允许同步串行口工作*/
    TRISC6=1;
    TRISC7=1;                      /*将 RC6,RC7 设置为输入方式,对外部呈高阻状态*/
}
/*SPI 传输数据子程序*/
void    SPILED(data)
{
    SSPBUF=data;                   /*启动发送*/
    do  {
            ;
    }while(SSPIF==0);
    SSPIF=0;
}
/*显示子程序,显示 8 bit 数*/
void    display()
{
```

```
        RA5=0;                          /*准备锁存*/
        for(k=0;k<8;k++)     {
            data=tran[k];
            data=table[data];           /*查得显示的段码*/
            SPILED(data);               /*发送显示段码*/
        }
        RA5=1;                          /*最后给一个锁存信号,代表显示任务完成*/
}
/*主程序*/
main()
{
    SPIINIT();
    fuzhi();                            /*给数组赋初值*/
    sciint();                           /*SCI 部件初始化*/
    di();                               /*中断禁止*/
    TXEN=1;                             /*发送允许*/
    CREN=1;                             /*接收数据允许*/
      for(k=0;k<8;k++){
       TXREG=tran[k];                   /*发出一个字符*/
          while(1){
           if(TXIF==1) break;
       }                                /*等待写入完成*/
    while(1){
          if(RCIF==1)      break;       /*若收到响应字节,则终止等待*/
       }
       RCREG=RCREG;                     /*读响应字节,清 RCIF*/
        }
       display();                       /*显示发送的数据*/
       while(1){
            ;
       }
}
```

2. 单片机 PIC2 编程(接收部分)

```
#include <pic.h>
/*该程序实现单片机双机异步通信功能,该程序是接收部分,并把接收的数据显示在 8
*个 LED 上*/
unsigned   char      rece[8];           /*定义一个数组存储接收数据*/
```

```
unsigned  char    k,data;                    /*定义通用寄存器*/
const     char    table[20]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0x88,
0x83,0xc6,0xa1,0x86,0x8e,0x7f,0xbf,0x89,0xff};
/*不带小数点的显示段码表*/
/*spi显示初始化子程序*/
void    SPIINIT()
{
    ;详细语句见发送程序
}
/*SCI部件初始化子程序*/
void    sciint()
{
    SPBRG=0X19;                    /*波特率设置与PIC1相同,为约9 600 bps*/
    TXSTA=0X04;                    /*异步高速传输*/
    RCSTA=0X80;                    /*串行口工作使能*/
    TRISC6=1;
    TRISC7=1;                      /*将RC6,RC7设置为输入方式,对外部呈高阻状态*/
}
/*SPI传输数据子程序*/
void    SPILED(data)
{
    ;详细语句见发送程序
}
/*显示子程序,显示4 bit数*/
void    display()
{
    RA5=0;                         /*准备锁存*/
    for(k=0;k<8;k++){
        data=rece[k];
        data=table[data];          /*查得显示的段码*/
        SPILED(data);              /*发送显示段码*/
    }
    RA5=1;                         /*最后给一个锁存信号,代表显示任务完成*/
}
/*主程序*/
main()
{
```

```
SPIINIT();                          /* spi 显示初始化 */
sciint();                           /* SCI 部件初始化 */
di();                               /* 中断禁止 */
CREN=1;                             /* 接收允许 */
TXEN=1;                             /* 发送允许 */
for(k=0;k<8;k++){
while(1){
        if(RCIF==1)     break;
}                                   /* 等待接收数据 */
    rece[k]=RCREG;                  /* 读取接收数据,同时清掉 RCIF */
    TXREG=rece[k];                  /* 发送接收到的数据 */
    while(1){
        if(TXIF==1)     break;
    }                               /* 等待写入完成 */
    }
    display();                      /* 显示接收的数据 */
    while(1){
        ;
    }
}
```

8.6　单片机双机同步通信

8.6.1　单片机双机同步通信硬件连接

双机同步通信示意图如图 8.4 所示。图中 2 片 PIC 采用相同的晶振(本例中采用 4 MHz 晶振),TX 线作为时钟线,RX 线作为数据线;因而数据传输以半双工方式进行,即发送和接收不能同时进行。

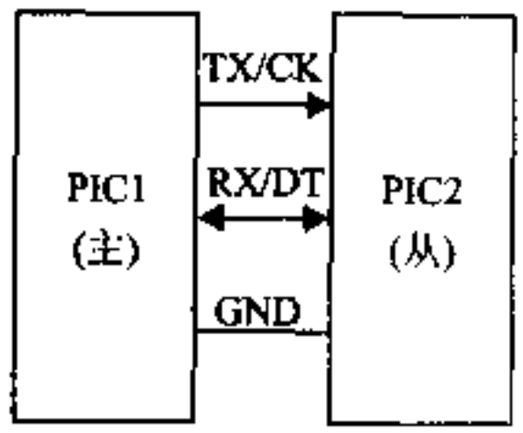


图 8.4　同步通信示意图

8.6.2　同步通信编程

同步通信数据传输的波特率由主机确定,其他和异步通信一样,主从机分别设置为 1 个开始位、8 个数据位、1 个停止位及无奇偶校验位。

与前面一样,规定程序实现下列任务:单片机 1 不断地发送 0～7 8 个数,其中“0”为同步字符;单片机 2 接收到同步字符后,开始连续接收随后的 7 个数据;接收完毕后,把同步字符和随后的 7 个数据都显示出来。同样,为节省篇幅,省去了与显示有关的硬件接口电路及其编程

部分。

单片机实现同步传输可采用查询和中断2种方式实现，下面给出了以查询方式实现的同步传输的例程。中断实现方式与查询方式基本相同。

1. 单片机 PIC1 编程(主控发送)

```
#include <pic.h>
/*该程序实现单片机双机同步通信功能,是主控发送部分。程序上电后显示
*相应的字符,表示系统正常工作。发送完毕后显示发送的数据*/
unsigned    char    tran[8];            /*定义一个数组存储发送数据*/
unsigned    char    k,data;             /*定义通用寄存器*/
const       char    table[20]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0x88,
0x83,0xc6,0xa1,0x86,0x8e,0x7f,0xbf,0x89,0xff};
/*不带小数点的显示段码表*/
/*spi显示初始化子程序*/
void    SPIINIT()
{
    ;详细程序语句请参考本章8.5节
}
/*给发送数组赋初值子程序 */
void        fuzhi()
{
    for(k=0;k<8;k++){
        tran[k]=k;
    }                                   /*发送0~7 8个数据*/
}
/*SCI部件初始化子程序*/
void        sciint()
{
    SPBRG=200 ;                         /*将传输的波特率设为约9 600 bps*/
    TXSTA=0X90;                         /*选择主控方式*/
    RCSTA=0X80;                         /*允许同步串行口工作*/
    TRISC6=1;
    TRISC7=1;                           /*将RC6,RC7设置为输入方式,对外部呈高阻状态*/
}
/*spi传输数据子程序*/
void        SPILED(data)
{
    ;详细程序语句请参考本章8.5节
```

```
}
/* 显示子程序,显示 8 bit 数 */
void          display()
{
    RA5=0;                          /* 准备锁存 */
    for(k=0;k<8;k++){
        data=tran[k];
        data=table[data];           /* 查得显示的段码 */
        SPILED(data);               /* 发送显示段码 */
    }
    RA5=1;                          /* 最后给一个锁存信号,代表显示任务完成 */
}
/* 显示子程序,显示 8 bit 数 */
void          display1()
{
    RA5=0;                          /* 准备锁存 */
    for(k=0;k<8;k++){
        data=0xf9;                  /* 显示"1"表示系统正常工作 */
        SPILED(data);               /* 发送显示段码 */
    }
    RA5=1;                          /* 最后给一个锁存信号,代表显示任务完成 */
}
/* 主程序 */
main()
{
    SPIINIT();                      /* spi 显示初始化 */
    fuzhi();                        /* 给发送数组赋发送初值 */
    sciint();                       /* SCI 部件初始化 */
    di();                           /* 中断禁止 */
    TXEN=1;                         /* 发送允许 */
    display1();                     /* 显示相应的字符,表示系统正常 */
    while(1){
    for(k=0;k<8;k++){
            TXREG=tran[k];          /* 发出一个字符 */
    while(1){
                if(TXIF==1)     break;
    }                               /* 等待上一个数据写入完成 */
```

```
        }
        display();                          * 显示发送的数据 * /
    }                                       / * 循环发送 * /
}
```

2. 单片机 PIC2 编程(从动接收)

```
#include    <pic.h>
 * 该程序实现单片机双机同步通信功能,是从动接收部分,并把接收的数据显
 * 示在 8 个 LED 上 * /
unsigned    char    rece[8];            / * 定义一个数组存储接收数据 * /
unsigned    char    k,data;             / * 定义通用寄存器 * /
unsigned    int     i;
const   char   table[20]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0x88,0x83,0xc6,
0xa1,0x86,0x8e,0x7f,0xbf,0x89,0xff};
 * 不带小数点的显示段码表 * /
/ * spi 显示初始化子程序 *
void    SPIINIT()
{
    ;详细程序语句请参考本章 8.5 节
}
/ * SCI 部件初始化子程序 * /
void    sciint()

    TXSTA =0X10     ;                   / * 选择同步从动方式 *
    RCSTA=0X90;                         / * 串行口工作使能 * /
    TRISC6 =1;
    TRISC7=1;                           / * 将 RC6,RC7 设置为输入方式,对外部呈高阻状态 *
}
/ * spi 传输数据子程序 * /
void        SPILED(data)
{
    ; / * 详细程序语句请参考本章 8.5 节 *
}
 * 显示子程序,显示 1 bit 数 * /
void    display()
{
    RA5=0;                              / * 准备锁存 * /
    for(k=0;k<8;k++){
```

```
        data=rece[k];
        data=table[data];                    /*查得显示的段码*/
        SPILED(data);                        /*发送显示段码*/
    }
    RA5=1;                                   /*最后给一个锁存信号,代表显示任务完成*/
    }
    /*主程序*/
    main()
    {
    SPIINIT();                               /*spi 显示初始化*/
    sciint();                                /*SCI 部件初始化*/
    di();                                    /*中断禁止*/
    CREN=1;                                  /*接收允许*/
    for(k=0;k<8;k++) rece[k]=0x03;
    display();                               /*显示表示系统正常运行的数据*/
    while(1)     {
        while(1){
            CREN=1;                          /*允许连续接收*/
            while(1){
                if(RCIF==1)      break;
            }                                /*等待接收数据*/
            k=0;
            rece[k]=RCREG;                   /*读取接收数据*/
            if(OERR==1) {                    /*如果有溢出错误,则处理*/
        CREN=0;
                CREN=1;
            }
    if(rece[k]==0x00) break;                 /*"0"为同步字符,只有接收到"0"时,才进行下面的接收*/
        }
        for(k=1;k<8;k++){
            while(1){
                if(RCIF==1)      break;
            }                                /*等待接收数据*/
            rece[k]=RCREG;                   /*读取接收数据*/
            if()ERR==1)      {               /*如果有溢出错误,则处理*/
                CREN=0;
                CREN=1;
```

```
            }
            rece[k]=rece[k]&0x0F;  /* 屏蔽掉高位,防止干扰 */
        }
        CREN=0;
        display();                          /* 显示接收的数据 */
        for(i=65535;--i;)continue;
        for(i=65535;--i;)continue;/* 给予一定时间的延时,再进行下一轮接收 */
    }
}
```

8.7 单片机与 PC 机通信

8.7.1 PIC 单片机与 PC 机通信的硬件连接

利用 PC 机配置的串行口,可方便地完成 IBM-PC 系列机与 PIC 单片机的数据通信。

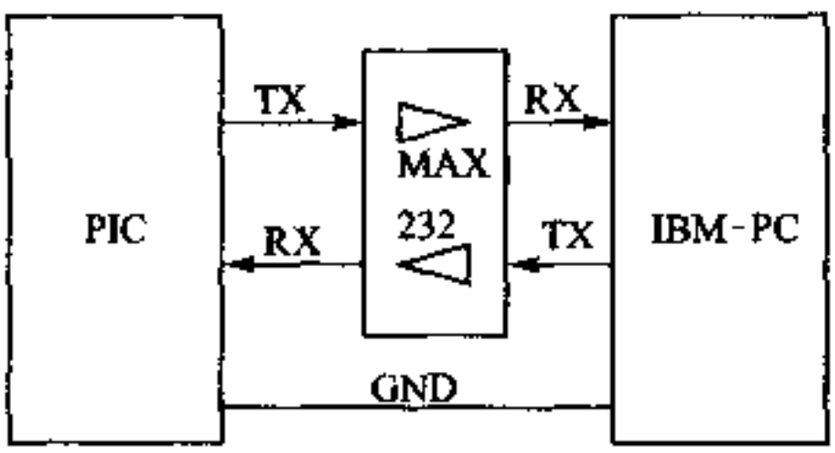


图 8.5 PIC-PC 异步通信示意图

IBM-PC 机与 PIC 单片机最简单的连接是 3 根线方式,这是进行全双工通信所必须的最少数目的线路。由于 PIC 单片机输入、输出电平为 TTL 电平,而 IBM-PC 机配置的是 RS-232C 标准串行接口,二者电气规范不一致;因此要完成 PC 机与单片机的数据通信,必须进行电平转换。图 8.5 为 IBM-PC 机与 PIC 单片机的连接示意图。图中 MAX232 将 PIC 单片机 TX 输出的 TTL 电平信号转换为 RS-232C 电平,输入到 IBM-PC 机,并将 IBM-PC 机输出的 RS-232C 电平信号转换为 TTL 电平,输出到 PIC 单片机的 RX 引脚。

8.7.2 单片机与 PC 机通信编程

假定程序实现下列功能:PC 机和单片机均可发送和接收数据,由 PC 机决定发送或接收。当 PC 机键盘输入命令"S"或"s"时,PC 机发送一组数据,单片机将接收到的数据送回 PC 机,PC 机分别显示发送和单片机回送的数据;当 PC 机键盘输入命令"R"或"r"时,PC 机接收单片机发送过来的一组数据。

要实现单片机和 PC 机间的通信,必须使单片机和 PC 机采用相同的数据传输格式。通信协议如下:波特率为 9 600 bps,8 bit 数据位,1 bit 停止位,无奇偶检验位。PC 机发送命令给单片机,命令 1 表示单片机接收数据,命令 2 表示单片机发送数据。PC 机和单片机均采用查询方式进行编程。

1. PC 机编程

PC 采用 Toubr C 进行编写，程序如下。

```
#include<stdio.h>
#define   port    0x3f8                                    /*利用串口 1 进行通信*/
int ch[15];
main ()
{
  int a;
  int i,j;
  int b[6]={88,15,38,26,20,0};
  char c;
  clrscr();
  outportb(port+3,0x80);                                   /*准备设置波特率*/
  outportb(port,0x0C);                                     /*波特率设置为 9 600 bps*/
  outportb(port+1,0x00);
  outportb(port+3,0x03);                                   /*8 bit 数据,无奇偶检验,1 bit 停止位*/
  outportb(port+1,0x00);                                   /*关中断*/
  inportb(port+5);                                         /*读一次线路状态寄存器,使其复位*/
  for(;;){
    printf("\t\tsend data or receive data: (s or r?)\n\n\n");
    c=getchar();
     switch(c) {
      case 's':
      case 'S': {
          while(! (inportb(port+5)&0x20));                 /*发送保持器满则等待*/
        outportb(port,0x01);                               /*否则发送数据 01,通知单片机准备接收*/
        for(i=0;i<6;i++){                                  /*共发送 6 个数据*/
         a=b[i];
         while(! (inportb(port+5)&0x20)) delay(100);       /*发送保持器满,等待*/
         outportb(port,a);                                 /*发送 a*/
         printf("%d\n",a);                                 /*显示 a*/
         while(! (inport(port+5)&1));                      /*接收单片机送回的数据*/
         ch[i]=inport(port);                               /*保存*/
  }
  delay(10);
  for(j=0;j<8;j++)     printf("\n%d\n",ch[j];   /*显示接收的回送数据*/
  getch();
```

```
    break;
  }
  case r:                                          /* 接收数据 */
  case R:{
     while(! (inportb(port+5)&0x20));
      outportb(port,0x02);                         /* 发送数据 02,通知单片机发送数据 */
      for(j=0;j<9;j++)      {                      /* 共接收 9 个数据 */
       while(! (inportb(port+5)&1));
      ch[j]=inportb(port);
      }
      for(j=0;j<9;j++) printf("\n %d\n",ch[j]);
      getch();
      break;
      }
    }
  }
}
```

2. 单片机编程

单片机编程与 8.5“单片机双机异步通信”类似,在此从略。

第 9 章　PIC16F87X 在 CAN 通信中的应用

在众多通信方式中，面向工业控制的现场总线技术是目前解决工业控制现场数据通信问题的最佳方案之一。现场总线技术是 20 世纪 80 年代发展起来的一种先进的现场工业控制技术。它集数字通信、智能仪表、微机技术、网络技术于一身，从根本上突破了传统的“点对点”式的模拟信号或数字-模拟信号控制的局限性，为真正的“分散式控制，集中式管理”提供了技术保证。

PIC16F87X 系列芯片都具有 SSP(synchronous serial port)模块，可与许多外围芯片(包括串行 EEPROM、A/D、D/A 转换器、移位寄存器等)或单片机实现串行通信。可由 2 种方式实现：SPI 方式(serial peripheral interface 串行外围接口)或 I^2C 方式(inter - IC 芯片间总线)。

本章将对 PIC16F87X 系列芯片通过 SPI 方式实现 CAN 通信进行详细介绍。为了容易理解程序起见，将对 CAN 的基本内容作一简单介绍。

9.1　CAN 通信原理

CAN(controller area network)总线是德国 Bosch 公司 20 世纪 80 年代初为解决现代汽车中众多的控制与测试仪器之间的数据交换而开发的一种串行数据通信协议。其通信协议根据国际标准化组织所提供的开放系统互联模型(ISO/OSI)制定，采用了 ISO/OSI 7 层框架中的物理层和数据链路层。它是一种多主总线，网络上任何节点均可主动向其他节点发送信息，网络节点可按系统实时性要求分成不同的优先级。数据链路层采用短帧结构，每帧为 8B，易于纠错。发送期间丢失仲裁或出错的帧可自动重新发送，故障节点可自动脱离总线。CAN 总线标准支持全双工通信，传输介质采用双绞线或光纤，通信速率可高达 1 Mbps，通信距离大于 1 km。CAN 协议采用通信数据块编码，从而使得网络内的节点个数在理论上不受限制；数据块的标识码可由 11 bit(标准帧)或 29 bit(扩展帧)二进制数组成，因此可定义 2^{11} 和 2^{29} 个不同的数据块；数据段长度最多为 8B，可满足通常工业领域中控制命令、工作状态及测试数据的一般要求；同时，8B 不会占用总线时间太长，从而保证了通信的实时性；采用 CRC 检验并可提供相应的错误处理功能，保证了数据通信的可靠性。

CAN 已被愈来愈多领域所采用和推广，使得不同应用领域通信报文的标准化问题越来越突出。为此，1991 年 9 月 Philips 半导体公司制定并发布了 CAN 技术规范(Version 2.0)。该技术规范包括 A，B 2 部分。2.0A 给出了曾在 CAN 技术规范版本 1.2 中定义的 CAN 报文格式，而 2.0B 给出了标准的和扩展的 2 种报文格式。详细内容请参考 Philips 半导体公司所出

版的《CAN SPECIFICATION [Version2.0]》一书，本文只简略介绍 2.0B 版本。

9.1.1　CAN 技术规范 2.0B

1. CAN 节点的分层结构

为使设计透明和执行灵活，CAN 分为数据链层和物理层。其中数据链层包括逻辑链控制 LLC(logical link control)子层和媒体访问控制 MAC(medium access control)子层。LLC 子层的主要功能是：为数据传输和远程请求提供服务，确认由 LLC 子层接收的报文实际已被接收，并为恢复管理和通知超载提供信息。在定义目标处理时，存在许多灵活性。MAC 子层的主要功能是传输规则，亦即控制帧结构，执行仲裁，错误检测，出错标定及故障界定。MAC 子层也要确定为开始一次新的发送，总线是否开放或者是否马上开始接收。位定时特性也是 MAC 子层的一部分；但是 MAC 子层特性不存在修改的灵活性。物理层的功能是有关全部电气特性不同节点间位的实际传输。

CAN 技术规范 2.0B 定义数据链中的 MAC 子层和 LLC 子层的一部分，并描述与 CAN 通信有关的外层。物理层定义信号怎样进行发送；因此涉及位定时、位编码及同步的描述。这部分技术规范中未定义物理层的驱动器/接收器的特性，以便允许根据它们的应用，对发送媒体和信号进行优化。MAC 子层是 CAN 协议的核心，描述由 LLC 子层接收到的报文和对 LLC 子层发送的认可报文。MAC 子层可响应报文帧、仲裁、应答、错误检测及标定，由称之为故障界定的一个管理实体监控，具有永久性故障或短暂性扰动的自检测机制。LLC 子层的主要功能是报文过滤、超载通知及复原管理。

2. CAN 的帧格式和帧类型

在 CAN 技术规范 2.0B 中，存在 2 种不同的帧格式，其区别在子标识码的长度：具有 11 bit标识码的帧为标准帧，具有 29 bit 标识码的帧为扩展帧。

CAN 技术规范 2.0B 对子报文过滤进行特别描述。报文过滤以整个标识码为基准，允许将任何标识码位设置为对报文过滤是“不关心”的。屏蔽位可用于选择一组标识码，以便映像到接收缓冲器中。如果使用屏蔽寄存器，屏蔽寄存器每一位必须是可编程的，亦即它们对于报文过滤是开放或禁止的。屏蔽寄存器的长度可以是整个标识码，也可以仅是其中的一部分。

报文传送由 4 种不同类型的帧表示和控制。数据帧——携带数据由发送器至接收器；远程帧——通过总线发送，以请求发送具有相同标识码的数据帧；出错帧——由通过检测发现总线错误的任何单元发送；超载帧——用于提供当前的和后续的数据帧或远程帧之间的附加延迟。数据帧和远程帧可使用标准格式和扩展格式，它们借助帧间空间和当前帧分离。

(1) 数据帧(标准格式和扩展格式)

数据帧由 7 个不同的位场所组成，如图 9.1 所示，即帧开始、仲裁场、控制场、数据场、CRC 场、应答场(ACK)及帧结束。其中数据场长度可为 0。

① 帧开始　帧开始 SOF(start of frame)标志着数据帧和远程帧的开始，仅由一个“显性”

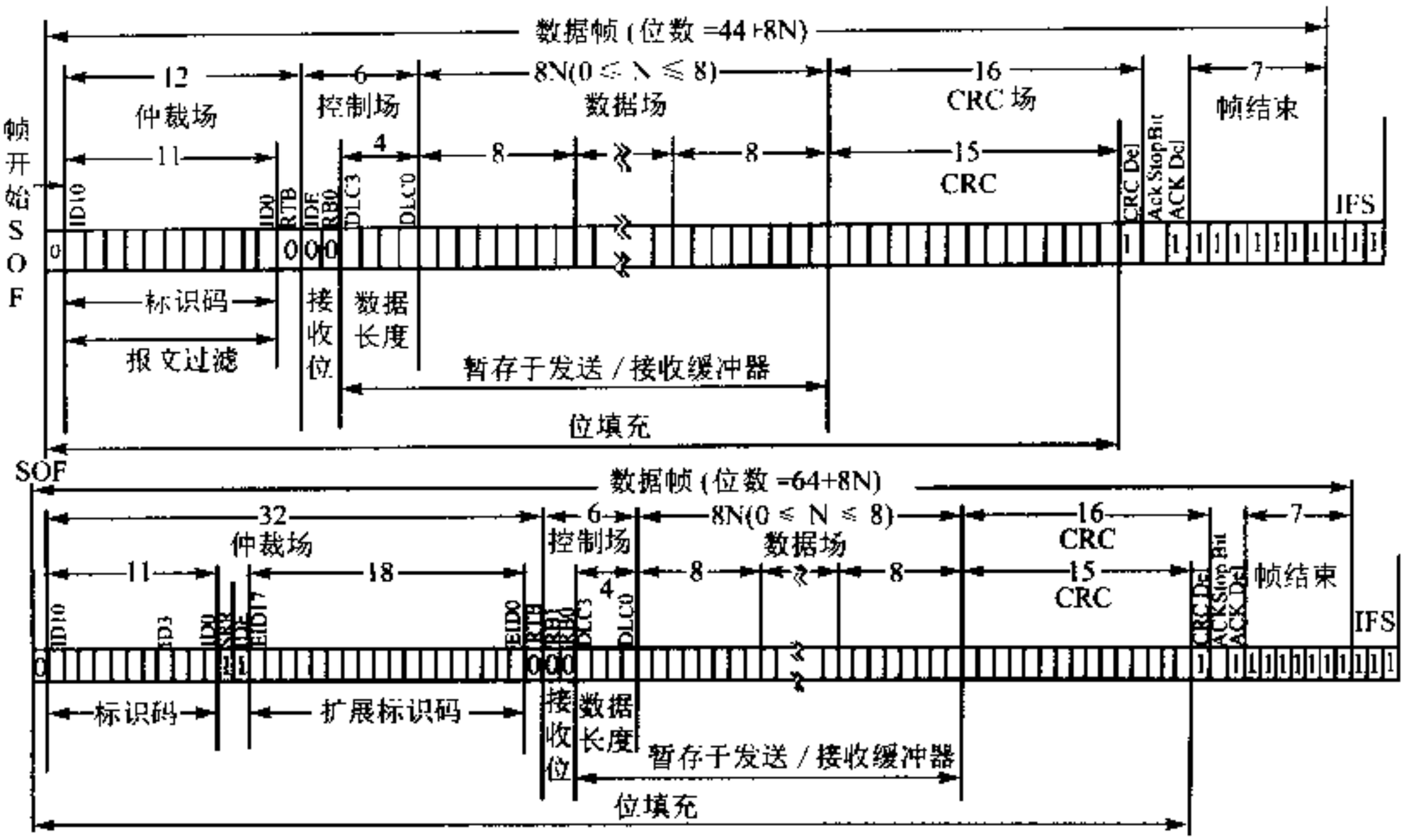


图 9.1　标准格式和扩展格式数据帧

位构成。只在总线处于空闲状态时，才允许 CAN 节点开始发送。

② 仲裁场　标准格式和扩展格式的仲裁场格式不同。在标准格式中，仲裁场由 11 bit 标识码和远程发送请求位组成，标识码位为 ID 28～ID 18；而在扩展格式中，仲裁场由29 bit标识码和替代远程请求 SRR(substitute remote request)位、扩展标识位及远程发送请求位组成，标识码位为 ID 28～ID 0。

③ 控制场　控制场由 6 bit 组成，标准格式和扩展格式不同。在标准格式中，1 帧包括数据长度码 4 bit(DLC3～DLC0)、发送“显性”电平的 IDE 位及保留位 RB0；在扩展格式中，1 帧包括数据长度码 4 bit(DLC3～DLC0)和 2 个保留位 RB1，RB0，这 2 个保留位必须发送“显性”电平；但是在全部组合中，接收器认可“显性”和“隐性”位的所有组合。

注意，CAN 中的总线数值为 2 种互补的逻辑数值之一：“显性”或“隐性”。“显性”(dominant)数值表示逻辑“0”，而“隐性”(recessive)表示逻辑“1”。当“显性”和“隐性”位同时发送时，最后的总线数值将为“显性”。

④ 数据场　数据场由数据帧中被发送的数据组成。它包括 0～8 B，由数据长度码 DLC (data lenth code)标明，1 B 包括 8 bit，最高有效位先发送。

⑤ CRC 场　包括 15 bit 的 CRC 序列和 1 bit CRC 界定符，系数为模 2 运算。其中 CRC 界定符为一个“隐性”位。

⑥ 应答场　应答场为 2 bit，包括应答间隙和应答界定符。在应答场中，发送节点送出 2

个“隐性”位，一个正确地接收到有效报文的接收器在应答间隙(送“ACK”期间)，将此信息通过传输一个“显性”位报告给发送器。所有接收到匹配CRC序列的节点，通过在应答间隙内把“显性”位写入发送器的“隐性”位来报告。应答界定符是应答场的第2位，并且必须是“隐性”位；因此，应答间隙被2个“隐性”位(CRC界定符和应答界定符)包围。

⑦ 帧结束　每个数据帧和远程帧都由7个“隐性”位组成的标志序列界定。

(2) 远程帧

激活为数据发送器的CAN节点可借助于传输一个远程帧，初始化各自源节点数据的发送。

远程帧存在标准的和扩展的2种格式。在2种情况下，均由6个不同的位场组成，如图9.2所示：帧开始、仲裁场、控制场、CRC场、应答场(ACK)及帧结束。与数据帧相反，远程帧的RTR位为“隐性”。远程帧不存在数据场。数据长度码DLC的数据值是独立的，可以被标注为允许范围0～8中的任何数据值。这一数值为对应数据帧的DLC。

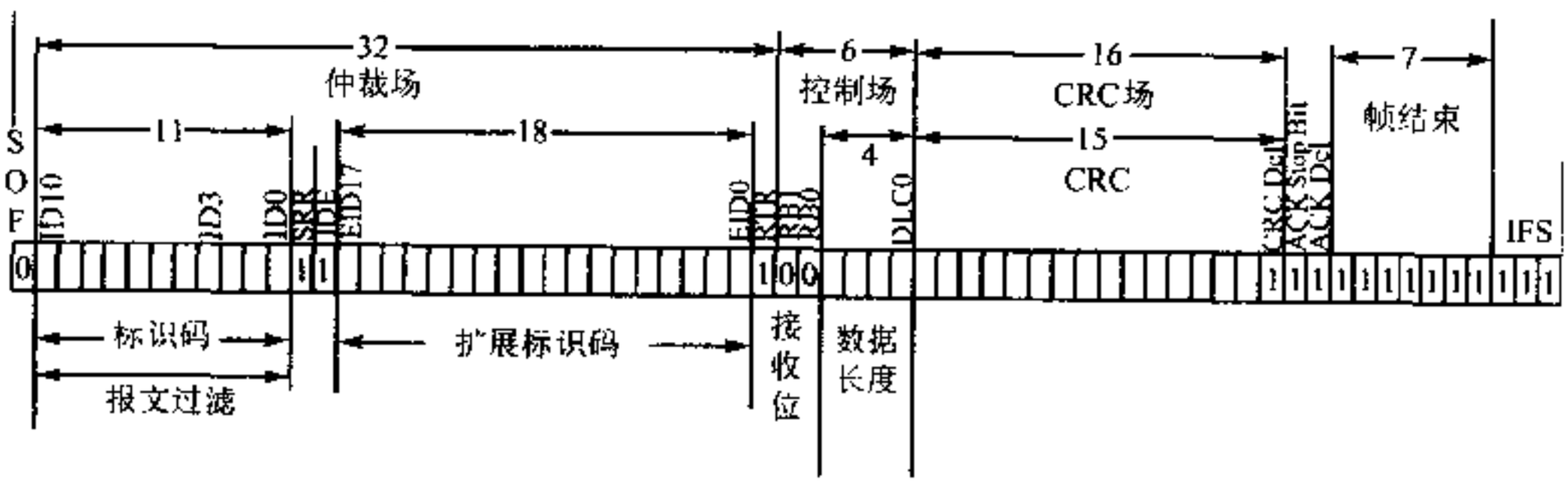


图9.2　带扩展标识的远程帧

RTR位的极性指出了所发送的是数据帧(RTR位为“显性”)，还是远程帧(RTR位为“隐性”)。

(3) 出错帧

出错帧由2个不同的场所组成。第1个是由来自各节点的错误标志叠加得到的出错标志，后随的第2个是出错界定符，如图9.3所示。

错误标志有2种形式：活动错误标志(active error flag)和认可错误标志(passive error flag)。活动错误标志由6个连续的“显性”位组成，而认可错误标志由6个连续的“隐性”位组成，除非它被来自其他CAN节点的“显性”位冲掉重写。

一旦某个CAN节点检测到出错条件的“错误激活”，就通过发送一个活动错误标志进行标注。这一出错标注形式违背了适用于帧开始至CRC界定符所有场的填充位规则，或者破坏了应答场或帧结束场的固定形式；因而其他CAN节点将检测到出错条件，并且开始由它们的部件发送出错标志。总线上被监视的“显性”位序列实际上由各个单独CAN节点发送的不同

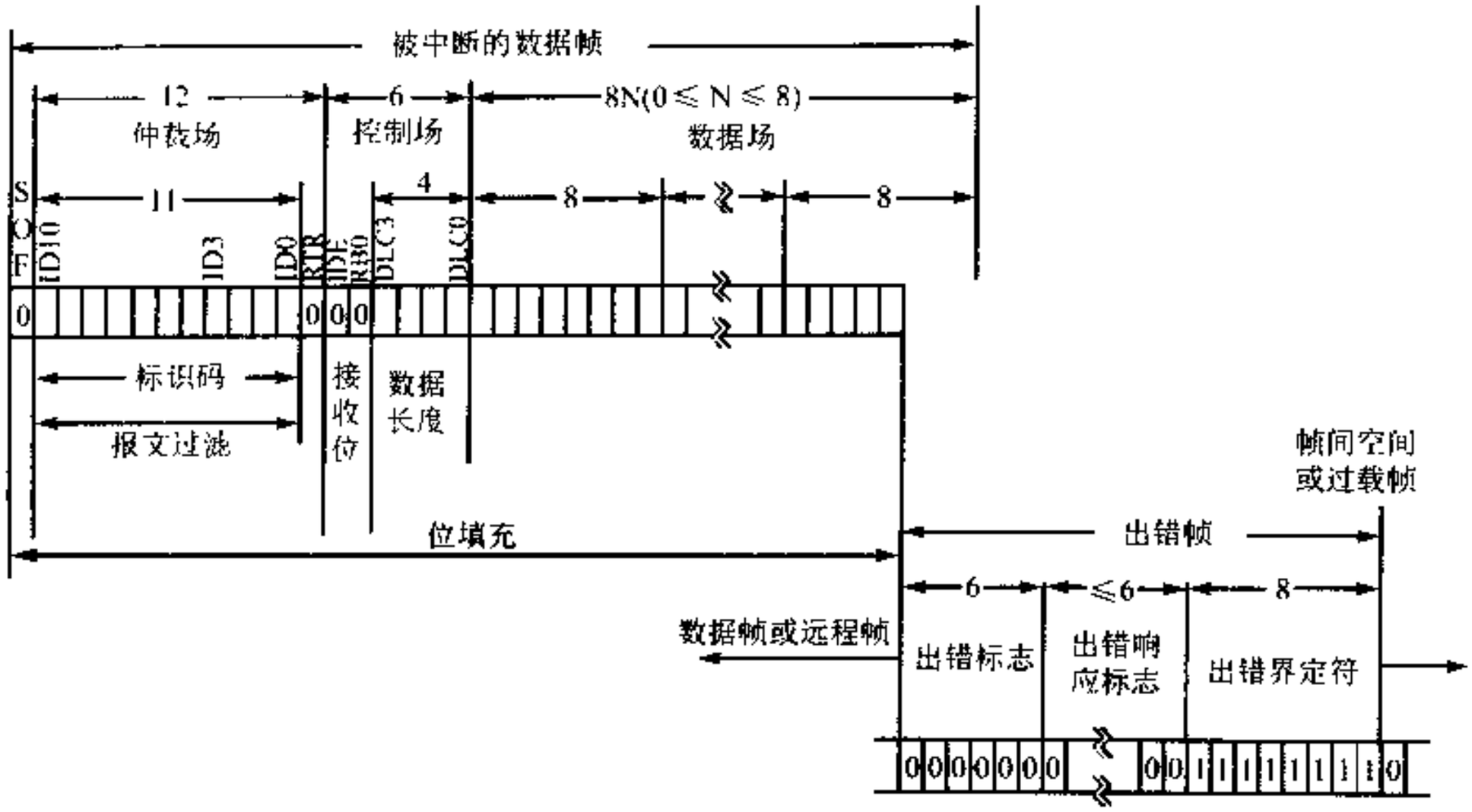


图 9.3　出错帧

出错标志叠加形成。该系列的总长度在最小值 6 bit 和最大值 12 bit 之间变化。一旦某个 CAN 节点检测到出错条件的“错误认可”，就通过发送一个认可错误标志进行标注。该“错误认可”CAN 节点自认可错误标志为起点，等待 6 个相同极性的连续位。当检测到 6 个相同位后，认可错误标志即告完成。

出错界定符包括 8 个“隐性”位。错误标志发送后，每个 CAN 节点都送出“隐性”位，并监视总线，直至检测到“隐性”位。此后，开始发送剩余的 7 个“隐性”位。

(4) 过载帧

过载帧包括 2 个位场：过载标志和过载界定符，如图 9.4 所示。

存在 2 种导致发送过载标志的过载条件：一是要求延迟下一个数据帧的内部条件，另一是在间歇场的第 1 位和第 2 位上检测到“显性”位。过载标志由 6 个“显性”位组成，全部形式对应于活动错误标志形式。过载界定符由 8 个“隐性”位组成，与出错界定符具有相同的形式。

(5) 帧间空间

数据帧、远程帧及其前面的帧，不管是何种帧(数据帧、远程帧、出错帧、过载帧)，均以称之为帧间空间的位场分隔开；但在过载帧和出错帧前面没有帧间空间，并且多个过载前面也不被帧间空间分隔。帧间空间包括间歇场和总线空闲场，其中间歇场由 3 个“隐性”位组成，而总线空闲场周期可为任意长度。

3. 位定时

在 CAN 总线通信中，通信速率的大小是由位定时设置来实现的。其中，正常位时间为正

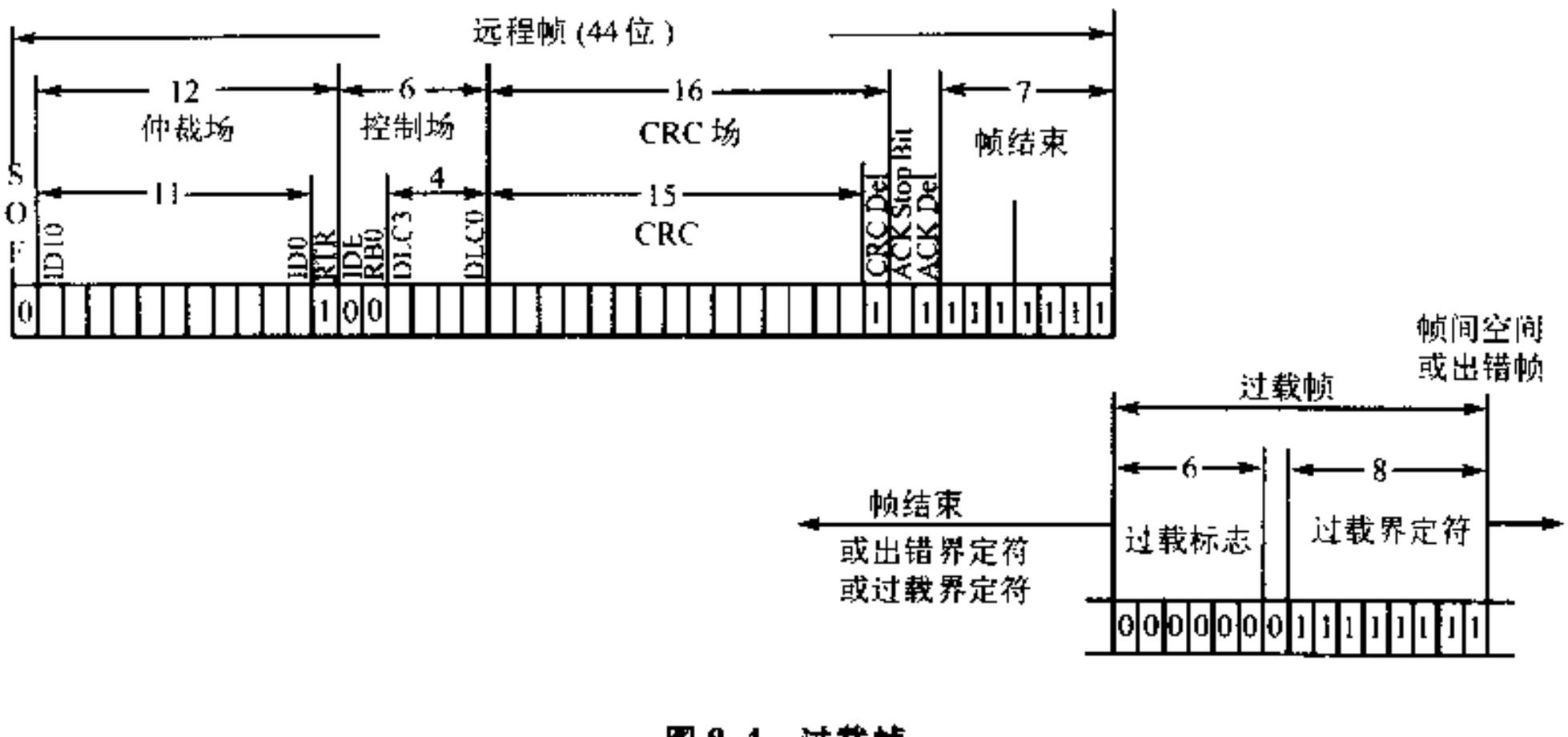


图 9.4 过载帧

常位速率的倒数。正常位时间包括几个互不重叠的时间段，如图 9.5 所示。这些时间段有同步段、传播段、相位缓冲段 1 及相位缓冲段 2。

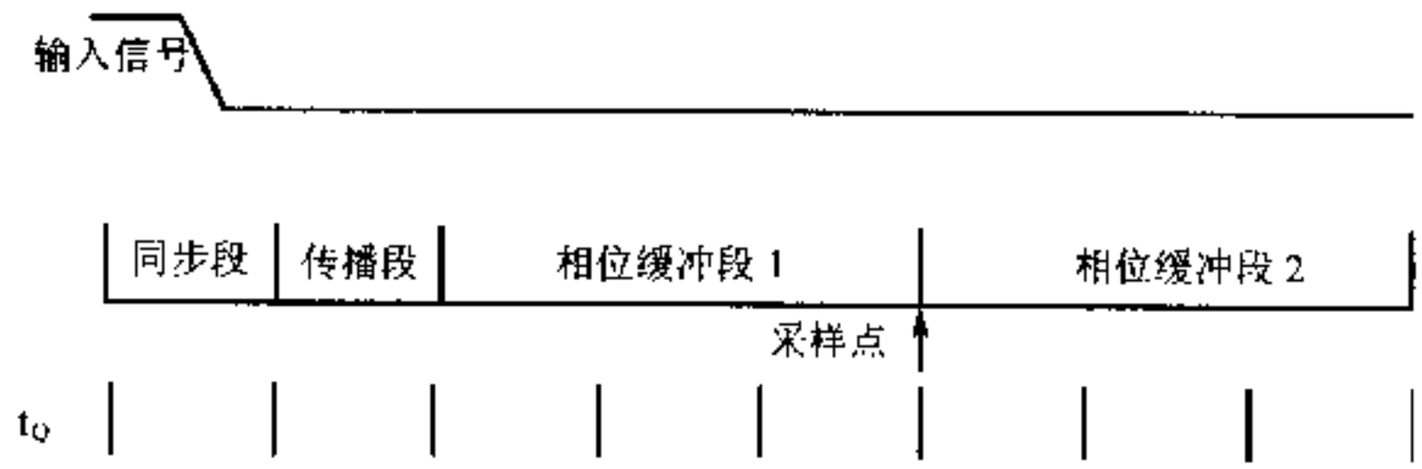


图 9.5 位时间的各段组成部分

正常位时间 $= t_Q *$（同步段＋传播段＋相位缓冲段 1＋相位缓冲段 2）

9.1.2 CAN 总线的基本工作原理图

CAN 总线的拓扑结构如图 9.6 所示。这是一个典型的串行总线的结构形式。CAN 总线以半双工的方式工作，一个节点发送信息，多个节点接收信息；但 CAN 总线的信息存取方式既不同于令牌方式的 Arcnet，也不同于主从方式的 Bit Bus，采用的是一种称为广播式的存取工作方式。与其他网络不同，在 CAN 总线的通信协议中，没有节点地址的概念，也没有任何与节点地址相关的信息存在，它支持的是基于数据的工作方式。也就是说，CAN 总线面向的是数据而不是节点；因此加入或撤消节点设备自然都不会影响网络的工作。这样的结构十分适用于控制系统要求快速、可靠、简明的特点。

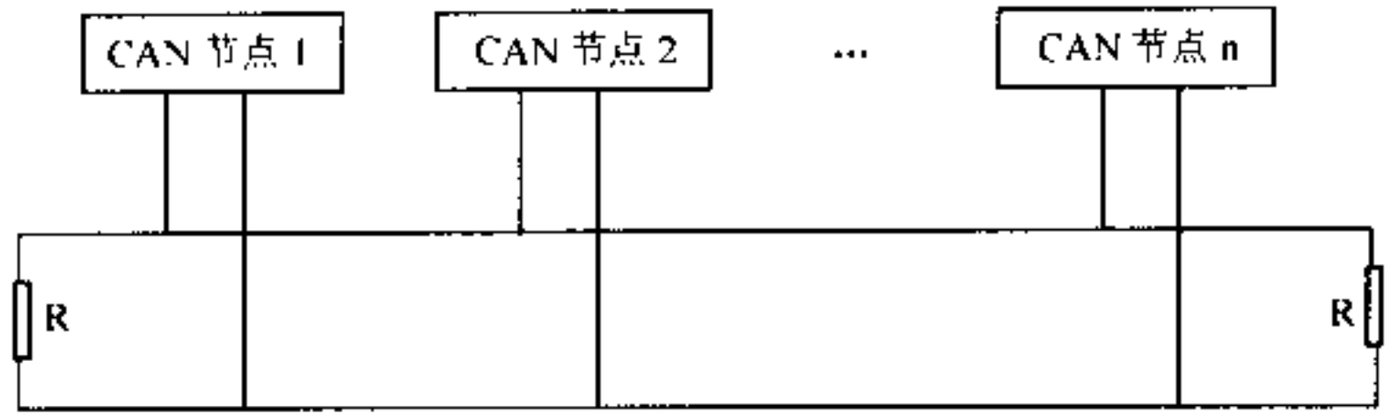


图 9.6　CAN 总线基本结构图

9.1.3　CAN 总线的基本器件

由于 CAN 总线具有通信速率高、可靠性高、连接方便及性价比高等诸多优点，推动了其应用开发的迅速发展。CAN 总线的基本器件有 INTEL 公司的 82526 和 82527 芯片，PHILIPS 公司的 82C200，82C250，8XC592 等芯片，MOTOROLA 公司的 68HC05X4 和 68HC05X16 芯片，SILIONI 公司的 SI9200 芯片，等等。在实际应用中，单独的 CAN 控制器必须与单片机组合使用，才能实现 CAN 通信；因此，许多单片机都集成有 CAN 功能模块，如 P8XC592，MOTOROLA 的 MC68HC05X 系列、196 系列等微控制器。PIC16F87X 系列芯片没有集成 CAN 功能模块，可以通过其 SPI 方式、MCP2510 芯片实现 CAN 通信。

MCP2510 是 MICROCHIP 公司生产的一种 CAN 控制器，可通过 SPI 方式与单片机接口，实现 CAN 通信(4.5 V 时，可高达 5 Mbps)。它支持 CAN 技术规范 2.0 A/B，通信速率高，可靠性高，实时性好，且连接方便；因而在工业自动化、多种控制设备、交通工具、医疗仪器、建筑及环境控制中得到广泛应用。

9.2　硬件电路

硬件设计中使用 MCP2510 芯片，下面对其作详细介绍。

9.2.1　MCP2510 引脚说明

MCP2510 有 3 种封装结构。本文介绍 18 脚的 PDIP 封装，引脚图如图 9.7 所示。其引脚的详细功能说明如表 9.1 所列。

表 9.1　MCP2510 引脚说明

名　称	引脚号	I/O/P 类型	功　能
TXCAN	1	O	向 CAN 总线发送数据引脚
RXCAN	2	I	从 CAN 总线接收数据引脚
CLKOUT	3	O	可编程时钟输出引脚

续表 9.1

名 称	引脚号	I/O/P 类型	功 能
$\overline{TX0RTS}$	4	I	发送缓冲器 0 请求发送(低电平有效)/通用数据输入(100 kbps)脚
$\overline{TX1RTS}$	5	I	发送缓冲器 1 请求发送(低电平有效)/通用数据输入(100 kbps)脚
$\overline{TX2RTS}$	6	I	发送缓冲器 2 请求发送(低电平有效)/通用数据输入(100 kbps)脚
OSC2	7	O	晶振输出脚
OSC1	8	I	晶振输入脚
V_{SS}(GND)	9	P	逻辑地
$\overline{RX1BF}$	10	O	接收缓冲器 1 中断(低电平有效)/通用数据输出脚
$\overline{RX0BF}$	11	O	接收缓冲器 0 中断(低电平有效)/通用数据输出脚
$\overline{INT}$	12	O	中断输出脚(低电平有效)
SCK	13	I	SPI 接口时钟输入脚
SI	14	I	SPI 接口数据输入脚
SO	15	O	SPI 接口数据输出脚
$\overline{CS}$	16	I	SPI 接口片选信号脚(低电平有效)
$\overline{RESET}$	17	I	MCP2510 芯片复位脚(低电平有效)
V_{DD}(V_{CC})	18	P	逻辑电源

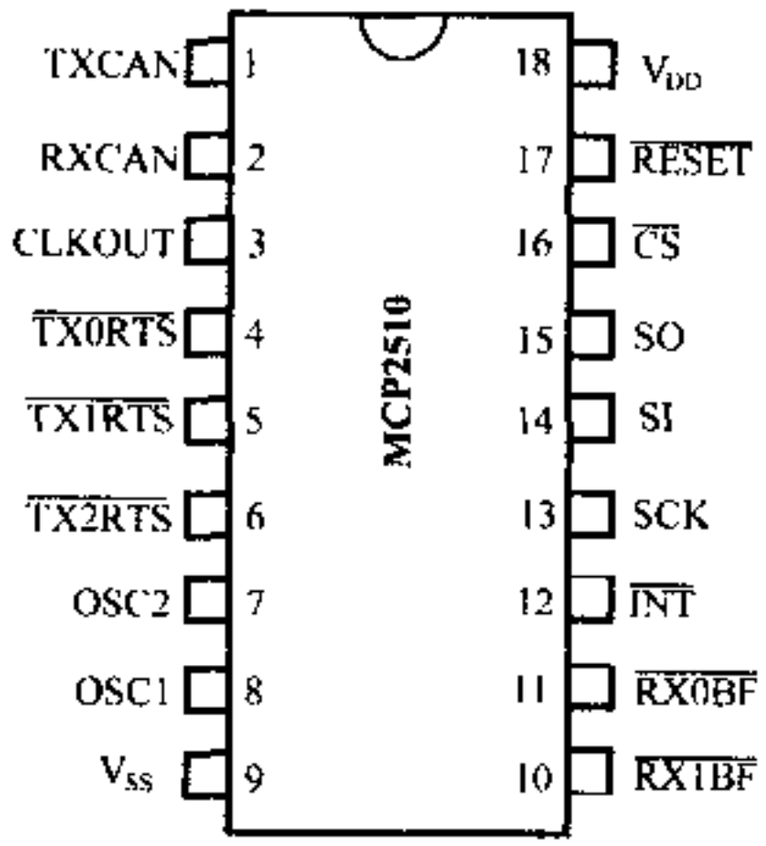


图 9.7 MCP2510 引脚图

其内部主要由 3 个模块组成:CAN 协议驱动、用于设置芯片及其操作模式的控制逻辑和静态寄存器 SRAM 及 SPI 接口模块。如图 9.8 所示。

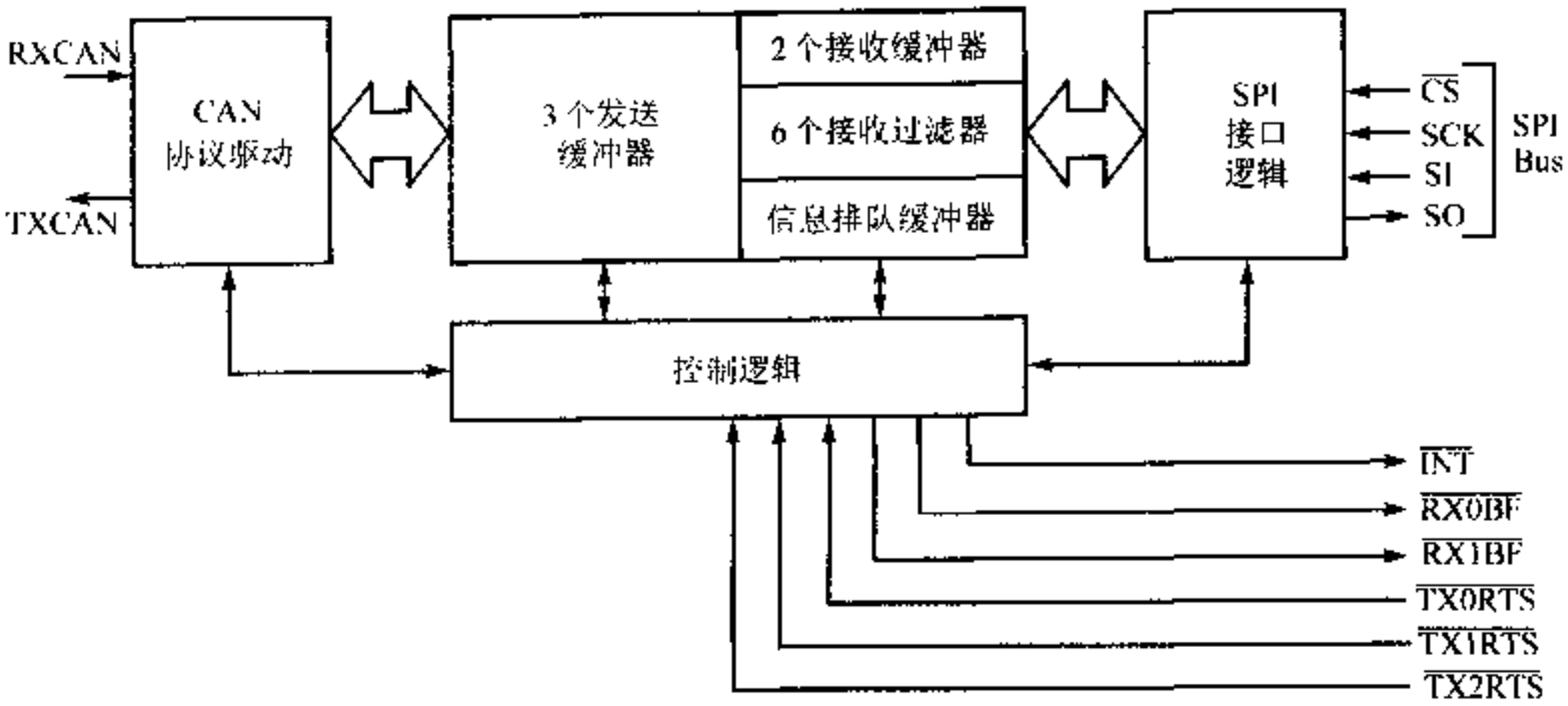


图 9.8　MCP2510 框图

9.2.2　MCP2510 芯片的几种操作模式

MCP2510 共有 5 种操作模式：配置模式(configation mode)、正常模式(normal mode)、休眠模式(sleep mode)、监听模式(listen－only mode)及自检模式(loopback mode)。

可通过 CANCTRL. REQOP 位选择某一项操作模式(或由一种操作模式进入另一种操作模式)。当芯片复位时，MCP2510 默认为配置模式。需要注意的是，当从一种操作模式转入另一种操作模式时，要等到所有的数据传输完毕后才能生效；因此运行另一种操作模式之前，要确认确实已经进入该模式(查询 CANSTAT. OPMODE 的 bit7～5 三位)。

1. 配置模式

MCP2510 在正常运行之前，必须进行初始化。只有在配置模式下，才能对其进行初始化。将 CANCTRL. REQOP 设成“100”，就可以从任何其他操作模式进入配置模式。当然，当芯片上电或复位时，自动进入该模式。当进入配置模式时，错误计数器的值被清 0；而且，只有在配置模式下，3 个配置寄存器 CNF1、CNF2、CNF3，接收过滤寄存器，屏蔽寄存器及发送请求控制寄存器 TXRTSCTRL 才能被位修改。

2. 正常模式

进入正常操作模式时，需将 CANCTRL. REQOP 设置成“000”。这是在 CAN 总线上传输数据的标准模式，亦即只有在这种模式下才能进行 CAN 信息的发送与接收。此时，MCP2510 芯片实时监视总线上的信息，并产生相应的应答位、出错帧等等。

3. 休眠模式

MCP2510 有一个内部的休眠模式，以降低芯片的功耗；但在休眠期间，SPI 接口保持有效，且可以访问芯片内的所有寄存器。

只要将 CANCTRL. REQOP 设置成“001”，就可进入休眠模式；同样要对 CANCTRL. REQOP 进行判断，是否真正进入了该模式。在内部休眠模式下，唤醒中断是保持有效的（当然要先让该中断使能），这样，还可以让单片机进入休眠模式。当 MCP2510 由总线激活时，再由该芯片唤醒单片机。需要特别注意的是，不要在传输数据的过程中让 MCP2510 进入休眠模式，因为这时数据传输就会被中止，并在总线上产生出错标志；而且，一旦从休眠模式退出时，进入休眠模式前未传输完的数据会继续被传输。在休眠模式下，MCP2510 将停止其内部晶振。

MCP2510 芯片从休眠中唤醒有 2 种方法：SPI 接口方式或 RCAN 引脚唤醒。

4. 监听模式

这种模式为 MCP2510 提供一种接收所有数据（包括错误数据）的手段。它可用作总线监视器或用于“热插拔（hot plugging）”情况下测波特率。在这种操作模式下，需将 CANCTRL. REQOP 设置成“011”。

5. 自检模式

自检模式可使 MCP2510 自发自收数据，而不需通过 CAN 总线。在系统开发和测试中常常用到自检模式，此时 CANCTRL. REQOP 设置为“010”。

在这种模式下，应答位 ACK 无效，也没有出错标志。MCP2510 芯片的接收器接收自己发送的信息就如同接收来自总线上的信息一样；但它并未实际向总线发送信息，而且，TX-CAN 引脚一直保持“隐性”状态。可以通过设置过滤器和屏蔽位，只接收某部分的信息；当将屏蔽位设置成全“0”时，则意味着接收所有的信息。

9.2.3 MCP2510 的寄存器

MCP2510 芯片共有 128 个寄存器，其地址由高 3 bit 和低 4 bit 确定，如图 9.9 所示。这样

低位地址	高位地址							
	x000 xxxx	x001 xxxx	x010 xxxx	x0011 xxxx	x100 xxxx	x101 xxxx	x110 xxxx	x111 xxxx
0000	RXF0SIDH	RXF3SIDH	RXM0SIDH	TXB0CTRL	TXB1CTRL	TXB2CTRL	RXB0CTRL	RXB1CTRL
0001	RXF0SIDL	RXF3SIDL	RXM0SIDL	TXB0SIDH	TXB1SIDH	TXB2SIDH	RXB0SIDH	RXB1SIDH
0010	RXF0EID8	RXF3EID8	RXM0EID8	TXB0SIDL	TXB1SIDL	TXB2SIDL	RXB0SIDL	RXB1SIDL
0011	RXF0EID0	RXF3EID8	RXM0EID0	TXB0EID8	TXB1EID8	TXB2EID8	RXB0EID8	RXB1EID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXB0EID0	TXB1EID0	TXB2EID0	RXB0EID0	RXB1EID0
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXB0DLC	TXB1DLC	TXB2DLC	RXB0DLC	RXB1DLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXB0D0	TXB1D0	TXB2D0	RXB0D0	RXB1D0
0111	RXF1EID0	RXF4EID0	RXM1EID0	TXB0D1	TXB1D1	TXB2D1	RXB0D1	RXB1D1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXB0D2	TXB1D2	TXB2D2	RXB0D2	RXB1D2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXB0D3	TXB1D3	TXB2D3	RXB0D3	RXB1D3
1010	RXF2EID8	RXF5EID8	CNF1	TXB0D4	TXB1D4	TXB2D4	RXB0D4	RXB1D4
1011	RXF2EID0	RXF5EID0	CANINTE	TXB0D5	TXB1D5	TXB2D5	RXB0D5	RXB1D5
1100	BFPCTRL	TEC	CANINTF	TXB0D6	TXB1D6	TXB2D6	RXB0D6	RXB1D6
1101	TXRTSCTRL	REC	EFLG	TXB0D7	TXB1D7	TXB2D7	RXB0D7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL

图 9.9 MCP2510 芯片的寄存器图

的安排考虑到了读写操作的方便性。某些专用的控制寄存器和状态寄存器可以通过 SPI 接口的位修改(bit modify)命令对各位进行修改，图 9.9 中阴影部分的寄存器即是；但需要特别注意的是，对于这些寄存器最好是用位修改命令，以免错误地置成其他不想要的状态。

图 9.10 是关于 MCP2510 芯片的控制寄存器的概要说明。

寄存器名称	地 址 (Hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	上电复位值
BFPCTRL	0C	—		B1BFS	B0BFS	B1BFE	B0BFE	B1BFM	B0BFM	- - 00 0000
TXRTSCTRL	0D	—		B2RTS	B1RTS	B0RTS	B2RTSM	B1RTSM	B0RTSM	- -xx x000
CANSTAT	xE	OPMOD2	OPMOD1	OPMOD0		ICOD2	ICOD1	ICOD0	--	100 - 000
CANCTRL	xF	REQOP2	REQOP1	REQOP0	ABAT	—	CLKEN	CLKPRE1	CLKPRE0	1110 111
TEC	1C	发送出错计数器								0000 0000
REC	1D	接收出错计数器								0000 0000
CNF3	28	--	WAKFIL		—		PHSEG22	PHSEG21	PHSEG20	-0 - - -000
CNF2	29	BTLMODE	SAM	PHSEG12	PHSEG11	PHSEG10	PRSEG2	PRSEG1	PRSEG0	0 00 0000
CNF1	2A	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	0000 0000
CANINTE	2B	MERRE	WAKIE	ERRIE	TX2IE	TX1IE	TX0IE	RX1IE	RX0IE	0000 0000
CANINTF	2C	MERRF	WAKIF	ERRIF	TX2IF	TX1IF	TX0IF	RX1IF	RX0IF	0000 0000
EFLG	2D	RX1OVR	RX0OVR	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN	0000 0000
TXB0CTRL	30	-	ABTF	MLOA	TXERR	TXREQ		TXP1	TXP0	000 0 00
TXB1CTRL	40	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	000 0-00
TXB2CTRL	50	—	ABTF	MLOA	TXERR	TXREQ	--	TXP1	TXP0	- 000 0-00
RXB0CTRL	60	-	RXM1	RXM0	--	RXRTR	BUKT	BUKT	FILHIT0	00- 0000
RXB1CTRL	70	---	RSM1	RXM0	--	RXRTR	FILHIT2	FILHIT1	FILHIT0	00 0000

图 9.10　MCP2510 控制寄存器

9.2.4　PIC16F87X 与 MCP2510 的 SPI 接口

PIC 单片机可通过 SPI 方式直接对 MCP2510 芯片进行操作(关于 PIC 单片机的 SSP 模块的详细资料，请参考北京航空航天大学出版社出版的《PIC16F87X 数据手册》一书)。

PIC16F87X 微控制器通过 SPI 接口对 MCP2510 芯片进行操作时，在时钟 SCK 的上升沿，命令和数据通过 SI 引脚送入 MCP2510，同时 MCP2510 通过 SO 引脚在 SCK 的下降沿送出数据。当然，执行任何一种操作都要使片选引脚 CS 为低电平。

MCP2510 芯片的 SPI 指令如表 9.2 所列。

表 9.2　MCP2510 的指令集(SPI 接口)

指令名	指令格式	功　能
RESET	1100 0000	将内部寄存器复位成默认状态，并置成设置(configuration)模式
READ	0000 0011	从指定地址开始的寄存器中读取数据
WRITE	0000 0010	向指定地址开始的寄存器中写入数据
RTS	1000 0nnn	请求发送指令(即置相应的 TXBnCTRL. TXREQ 位为 1)
READ STATUS	1010 0000	读取 MCP2510 的状态(包括发送接收中断标志位和各请求发送位)
BIT MODIFY	0000 0101	对指定的寄存器进行位修改

指令时序图如图 9.11～9.16 所示。

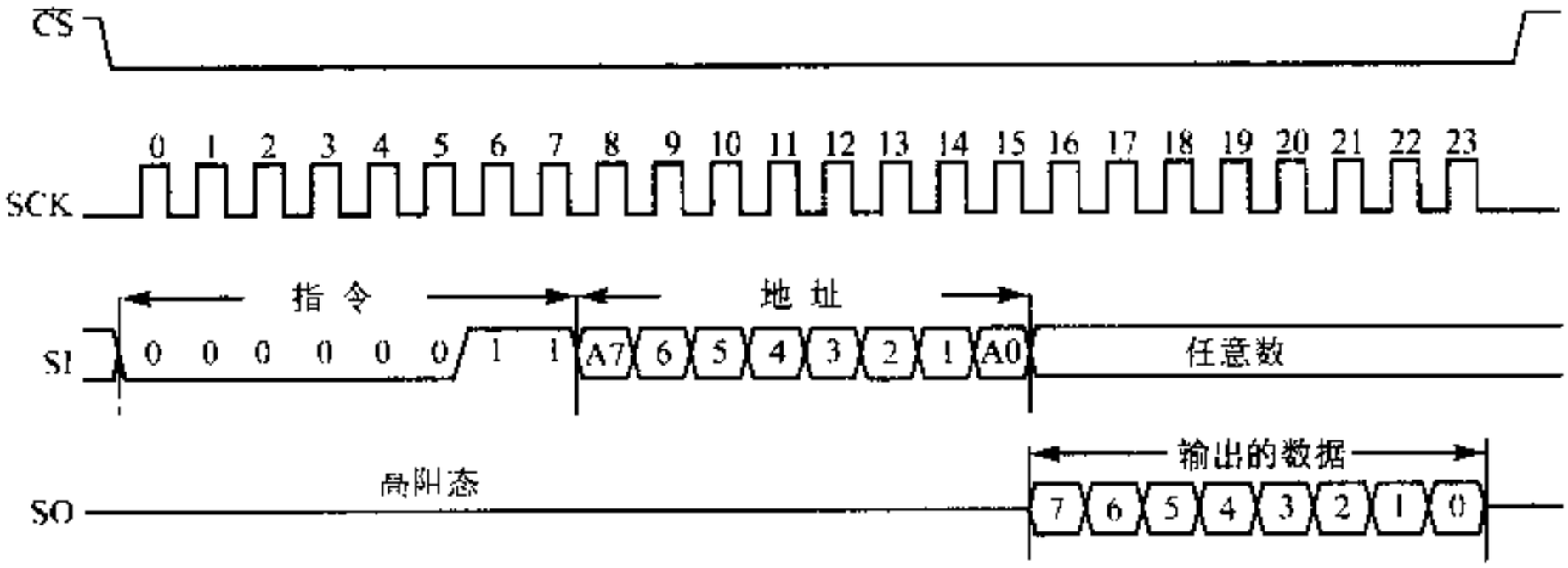


图 9.11　读指令(READ)时序

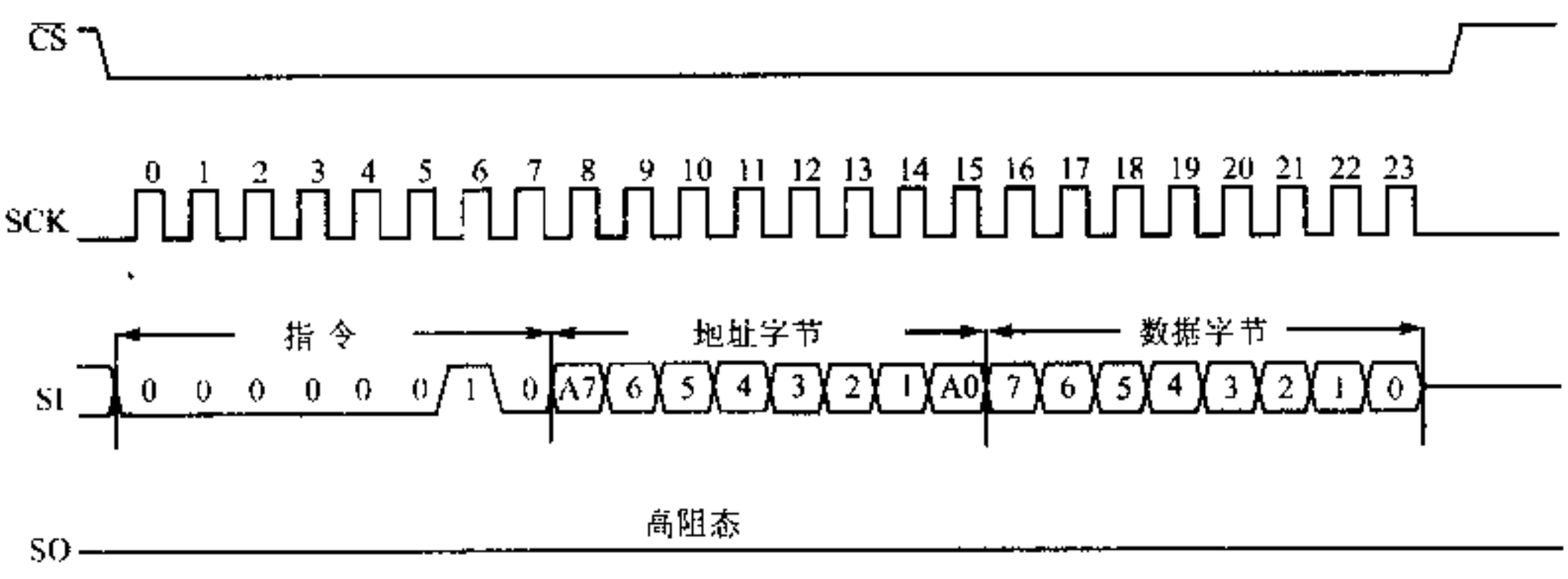


图 9.12　写指令(WRITE)时序

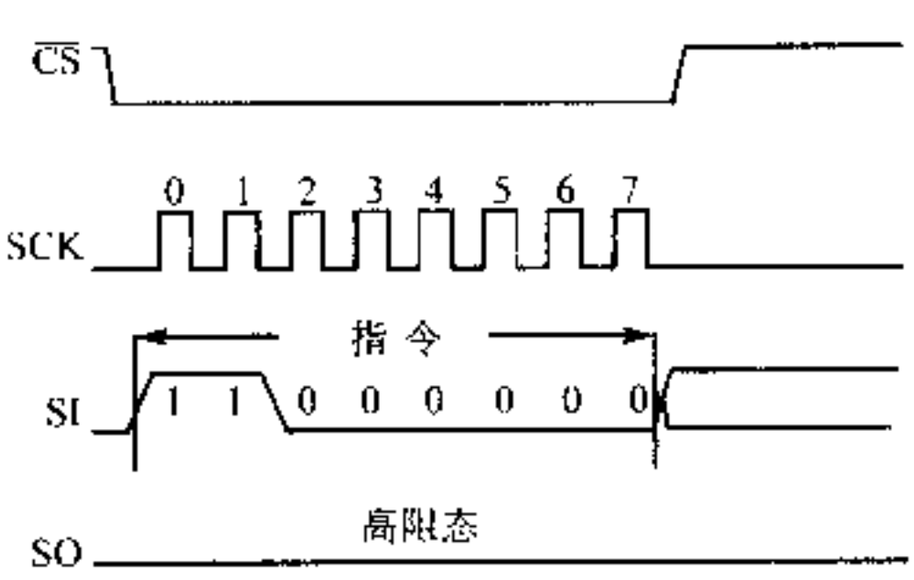


图 9.13　复位指令(RESET)时序

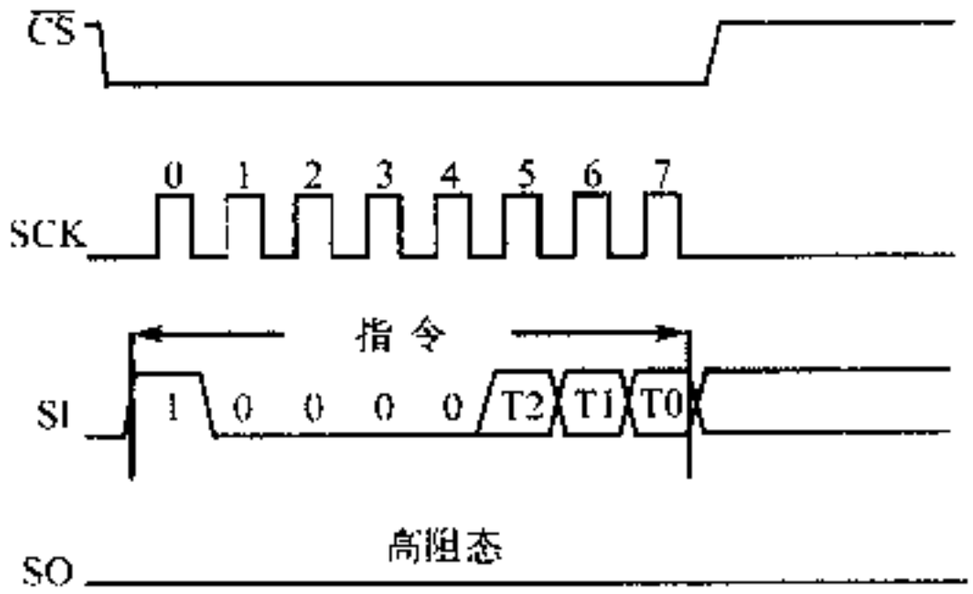


图 9.14　请求发送指令(RTS)时序

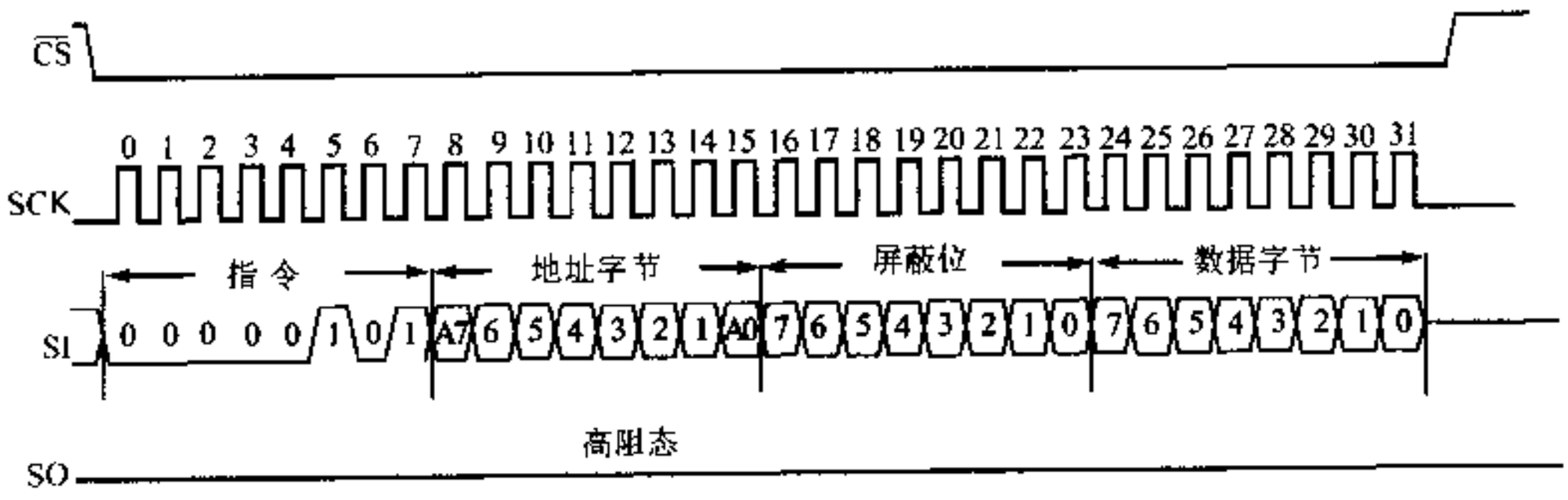


图 9.15　位修改指令(BIT MODIFY)时序

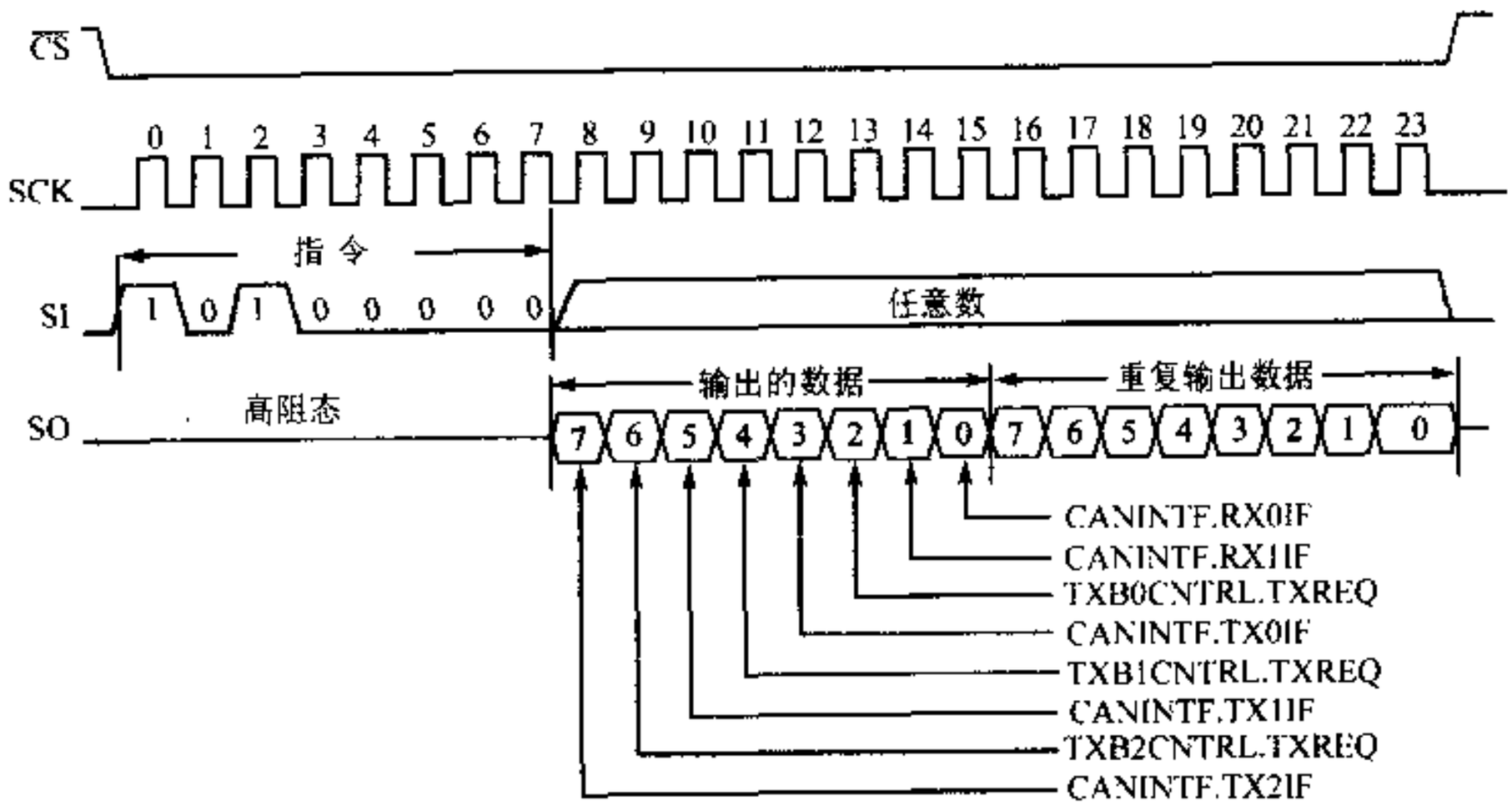


图 9.16　读取状态指令(STATUS)时序

9.2.5　PIC16F87X 与 MCP2510 的电路图

现以目前广泛应用的 PIC16F877 单片机说明实现 CAN 通信的硬件接线原理图。如图 9.17 所示：PIC16F877 的 SPI 模块接口接 MCP2510 的 SI，SO，SCK；RA4 和 RA1 分别控制 MCP2510 的芯片复位和片选(当然 RA4 可省略，因为它的作用与 RESET 指令是一样的；而在实际应用中，通常在初始化 MCP2510 时用 RESET 指令)；RB0，RB4，RB5 分别接收 MCP2510 的全局中断和缓冲器满中断。

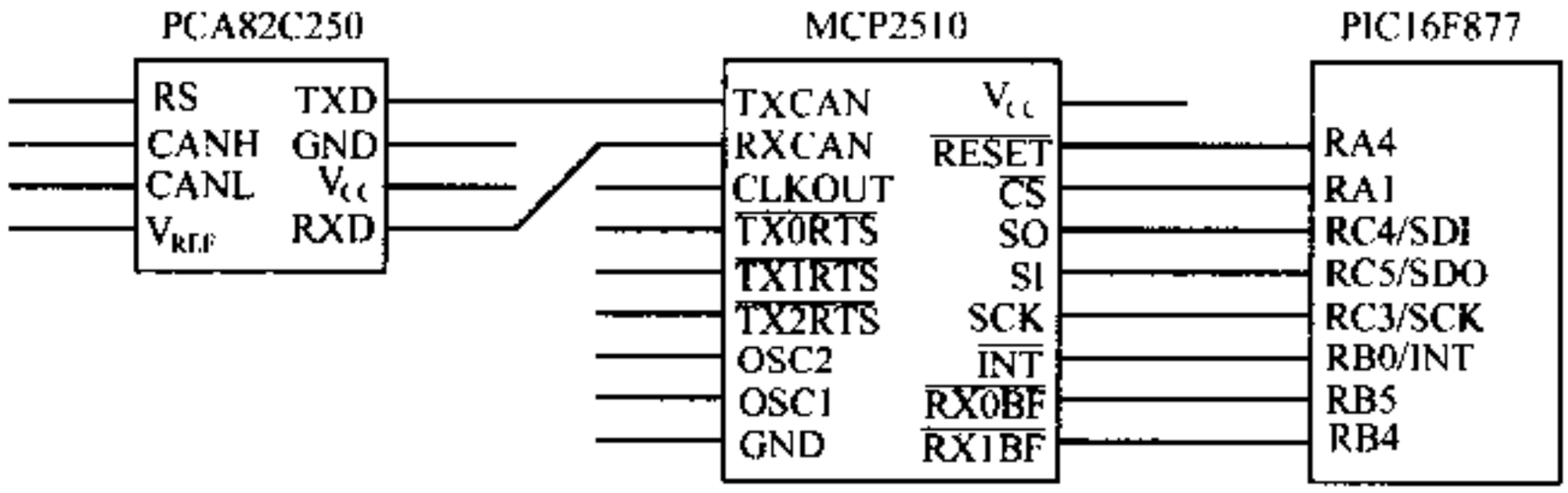


图 9.17 CAN 通信硬件简图

9.3 软件清单

```
// ========CAN 通信程序========
#include <pic.h>
#include <pic16F87x.h>
#include <mcp2510.h>                    // MCP2510 寄存器定义
// ==========常数和变量定义==========
#define READ 0x03                       // 读 MCP2510 指令代码
#define WRITE   0x02                    // 写 MCP2510 指令代码
#define RESET   0xC0                    // 复位 MCP2510 指令代码
#define RTS   0x80                      // MCP2510 请求发送指令代码
#define STA2510 0xA0                    // 读 MCP2510 状态指令代码
#define BITMOD 0x05                     // MCP2510 位修改指令代码
int a[12];                              // SPI 发送或接收数据寄存器
int b[8];                               // 发送或接收的数据
int c[8];                               // 发送或接收的数据
int i;                                  // 临时变量
int count;                              // 发送接收计数器
int count1=0;                           // for test
int RecID_H=0;
int RecID_L=0;
int DLC=8;
void SPIINT();
void TMR1INT();
void CCP1INT();
void SPIEXCHANGE(int count);
```

```
void WAIT_SPI();
void RESET2510();
int  RD2510(int adress,int n);
void WR2510(int adress,int n);
void RTS2510(int RTSn);
int  GETS2510();
void BM2510(int adress,int mask,int data);
void SETNORMAL();
void TXCOMPLETE(int adress);
void TXMSG(int DLC);
int  RXMSG();
void INIT2510();
void INIT877();
void INITSPI();
void ACK();
void wait();
// ==========主程序==========
main(void)
{
    int i,detect=0;
    SSPIE=1;
    TMR1IE=1;
    CCP1IE=1;
    CCP2IE=1;
    PEIE=1;
    ei();                          // 开中断
    INIT877();                     // 初始化 PIC16F877 芯片
    INITSPI();                     // 初始化 SPI 接口
    INIT2510();                    // 初始化 MCP2510 芯片
flag1=0;
flag2=0;
CCP1CON=0x05;
CCP2CON=0x04;
while(1)     {
    RXMSG();
    TXMSG(8);
}
```

```
}
// ======    ===中断服务程序=======
// SPI 中断服务子程序
void SPIINT()
{
    SSPIF=0;
    a[i++]=SSPBUF;                    // 数据暂存 a[]中
    count--;
    if(count>0)  SSPBUF=a[i];         // 未发送完,继续
    else  RE2=1;                      // 否则,片选信号置高电平
    return;
}
// TMR1 中断服务子程序
void TMR1INT()
{
    TMR1IF=0;
    T1CON=0;
    if(! flag1){
        TMR1H=0xfe;                   // 512 μs 脉冲宽度
        TMR1L=0x00;
        T1CON=0x01;
        PORTD=0xff;                   // 输出所有通道
        flag1=1;
    }
    else     {
        flag1=0;
        PORTD=0;
        T1CON=0;
    }
    return;
}
// CCP1 中断服务子程序
void CCP1INT()
{
    CCP1IF=0;
    T1CON=0x01;
    return;
```

```
}
// CCP2 中断服务子程序
 void CCP2INT()
{
    CCP2IF=0;
    T1CON=0x01;
    return;
}
// 中断入口,保护现场,判中断类型
void interrupt INTS()
{
    di();
    if(TMR1IF)  TMR1INT();          // 定时器 TMR1 中断
    else if(CCP1IF)  CCP1INT();     // 电压过 0 捕捉中断 1
    else if(CCP2IF)  CCP2INT();     // 电压过 0 捕捉中断 2
    else if(SSPIF)  SPIINT();       // SPI 接口中断
    ei();
}
// ========子程序========
// 启动 SPI 传输
 void SPIEXCHANGE(count)
 int count;
{
    if(count>0) {                   // 有数据可送?
      i=0;
      RE2=0;                        // 片选位置低电平
      SSPBUF=a[i];                  // 送数
    }
    else
      ;                             // 否则,空操作,并返回
    return;
}
// 等待 SPI 传输完成
 void WAIT_SPI()
{
    do{
      ;
```

```
    }while(count>0);                    // 当 count! =0 时,等待 to add "CLRWDT"
    return;
}
// 对 MCP2510 芯片进行复位
void RESET2510()
{
    a[0]=RESET;
    count=1;
    SPIEXCHANGE(count);                 // 送复位指令
    WAIT_SPI();
    return;
}
// 读取从地址"adress"开始的寄存器中的数据,共 n 个,存放在数组 b[n]中
int RD2510(adress,n)
int     adress;
int     n;
{
    int j;
    a[0]=READ;
    a[1]=adress;
    for(j=0;j<n;j++)  a[j+2]=0;
    count=n+2;                          // 指令、地址及要得到的数据量 n
    SPIEXCHANGE(count);
    WAIT_SPI();
    for(j=0;j<n;j++)  b[j]=a[j+2];      // 数据存到数组 b[]中
    return;
}
// 向从地址"adress"开始的寄存器写入数据,共 n 个,数据存放数组 b[n]中
void WR2510(adress,n)
int     adress;
int     n;
{
    int j;
    a[0]=WRITE;
    a[1]=adress;
    for(j=0;j<n;j++) a[j+2]=b[j];
    count=n+2;                          // 指令、地址及要写入的数据量 n
```