AN2045

串行 EEPROM 与 8位PIC® 单片机的接口设计

作者： *Regine Monique Aurellano*
Microchip Technology Inc.

简介

目前对廉价和便携式非易失性存储器的需求仍保持稳定，因为对许多工业和商业技术的开发而言，长期存储器存储依然不可或缺。对于许多这样的市场，串行 EEPROM 器件仍然被认为是非易失性存储器嵌入式控制应用的经济高效的理想解决方案。尽管其他形式的非易失性存储器重新兴起，但对于要求便携性、低电流和电压操作、逐字节操作及具竞争力价格的应用和解决方案，串行 EEPROM 仍被证明是一种可行的选择。SPI 和 I^2C 同步串行协议仍然是与串行 EEPROM 器件接口的两种最流行的方式。为了与之适应，大多数 PIC® 单片机器件都内置了主同步串行端口（Master Synchronous Serial Port，MSSP）模块，为这两种协议中的同步串行操作提供了一个方便的平台。

本应用笔记旨在说明如何使用 MPLAB® X 3.10、XC8 v1.34 编译器和 MPLAB® 代码配置器 v2.25 来实现 SPI 和 I^2C 串行 EEPROM 器件的接口。 Explorer 8 开发板用作硬件开发平台。针对本应用笔记编写的固件基于 MCC 自动生成的 SPI 和 I^2C 函数代码构建，提供了关于字节读操作和写操作、缓冲区 / 页写操作、顺序读操作和写周期查询操作的参考信息。其代码已使用 MikroElectronika EEPROM 和 EEPROM2 Click™ 电路板的 EEPROM 以及 Microchip 串行 EEPROM PIM PICtail™ 包中的 SPI 和 I^2C 接插模块（Plug-In Module，PIM）进行了测试。

SPI 接口

以下三个特性最贴切地描述了 SPI 协议：同步、指定一个主器件与从器件通信并且是在主器件与从器件之间交换数据的全双工系统。 SPI 是一种使用时钟信号来同步数据传输的同步协议。除非存在时钟信号，否则不会发生数据传输。主器件提供该时钟信号并对其进行控制。所有从器件都受该主时钟控制，而不能操控它。当数据按时钟送出主器件或从器件时，同时也在送入新数据。这与该系统的全双工特性一致。片选 （Chip Select，$\overline{CS}$）信号控制与主器件通信的特定从器件，确保每次只有一个从器件接入。

对于本应用说明中的示例和波形，使用了 PIC16F1719 单片机作为主器件、装在 MikroElectronika EEPROM2 Click 电路板上的 2 Mb 串行总线 EEPROM 作为从器件。

标准 SPI 信号

SPI 协议使用四条信号线来对通信系统中的数据流进行仲裁。

- 片选 （$\overline{CS}$）
  - 该信号用于选择与主器件通信的从器件。将该信号线置为有效状态会选择器件。
- 串行时钟 （Serial Clock， SCK）
  - 它是由主器件产生的时钟信号，用于控制何时发送和读取数据。
- 串行数据输出 （Serial Data Output， SDO）
  - 它是携载送出器件的数据的信号线。
- 串行数据输入 （Serial Data Input， SDI）
  - 它是携载送入器件的数据的信号线。

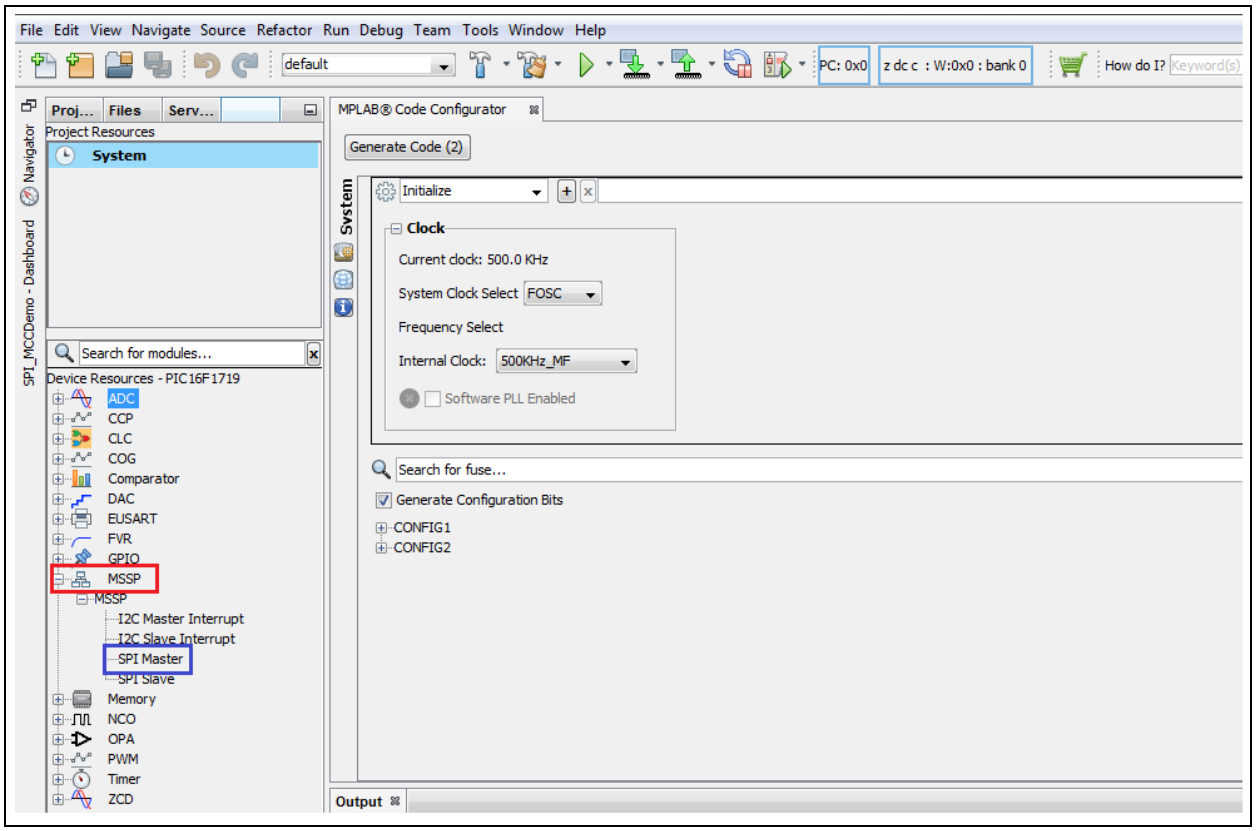
注： 主器件的 SDO 线应连接到从器件的 SDI 线。

通过 MPLAB® 代码配置器 （MCC V2.25）进行 SPI MSSP 初始化

本部分将指导用户初始化适当的寄存器，以使用大多数 PIC 器件中提供的 MSSP 模块来实现 SPI。 MPLAB 代码配置器（MPLAB Code Configurator，MCC）使这个过程变得更简便、更直观。在打开一个新项目并启动 MCC 插件后，在 Device Resources （器件资源）侧边栏中选择 MSSP 模块，它在图 1 中标记为红色。从下拉选项中，选择 SPI Master （SPI 主器件）。它在图 1 中标记为蓝色。

为了将 PIC 器件中的 MSSP 模块配置为 SPI 操作，需要正确地初始化几个寄存器。为了匹配从器件的配置，将使用 SPI 模式 （0,0）； 在该模式下， SCK 信号在空闲时为低电平，数据在时钟的下降沿发生更改，并在上升沿被视为有效。

图 1： 在 MCC 中选择 MSSP 模块



SSPx 状态寄存器 （SSPxSTAT）

SSPxSTAT 寄存器包含与 MSSP 模块关联的所有状态位。在 SPI 模式下， SMP 位决定要采样的输入部分。CKE 位决定在时钟的哪个边沿发送数据。

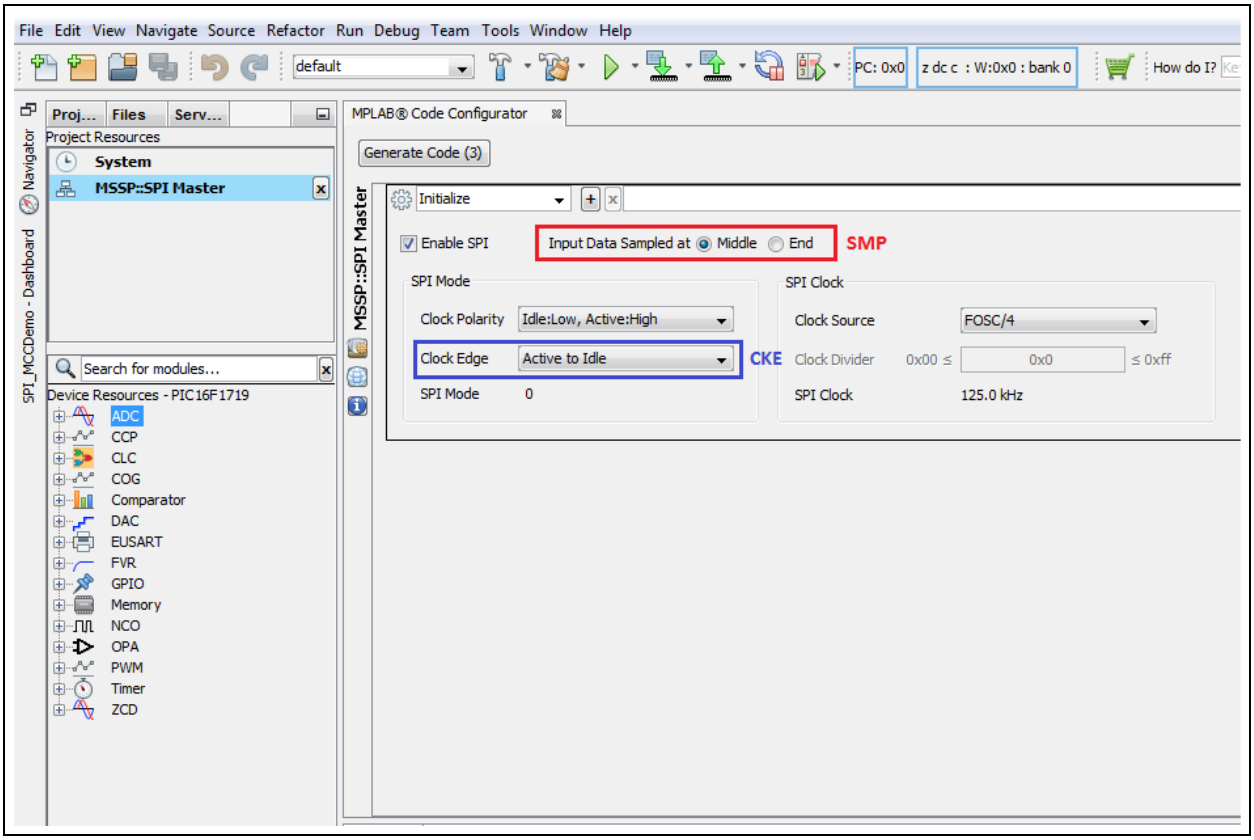
BF 位指示数据字节传输是否已经完成。在启动任何读或写操作之前，最好先确保 BF 位清零。图 2 显示了 SSPxSTAT 寄存器中的位位置，图 3 显示了 MCC 的配置屏幕中与 SSPxSTAT 寄存器中的位对应的部分。

图 2： SSPxSTAT：用于 SPI 配置的 SSPx 状态寄存器

REGISTER 30-1: SSP1STAT: SSP STATUS REGISTER

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

图 3： 在 MCC 上配置 SSPxSTAT 位



在 MCC 中单击 **Generate Code**（生成代码）按钮之后，图 3 所示的配置将在 `spi.c` 中生成以下代码行 （见例 1）。

例 1： SSPxSTAT MCC 生成的代码

```
//  BF RCinprocess_TXcomplete; SMP Sample At Middle; CKE Active to Idle;
    SSP1STAT = 0x40;
```

SSPx 控制寄存器 1 （SSPxCON1）

SSPxCON1 是 MSSP 模块的配置寄存器之一。它包含要将模块置为所需模式需要配置的几个指示位和选择位。在 SPI 模式下，需要通过将 SSPEN 位置 1 来使能串行端口。在使能时，SCK、SDO 和 SDI 引脚会配置为用于 SPI 操作。

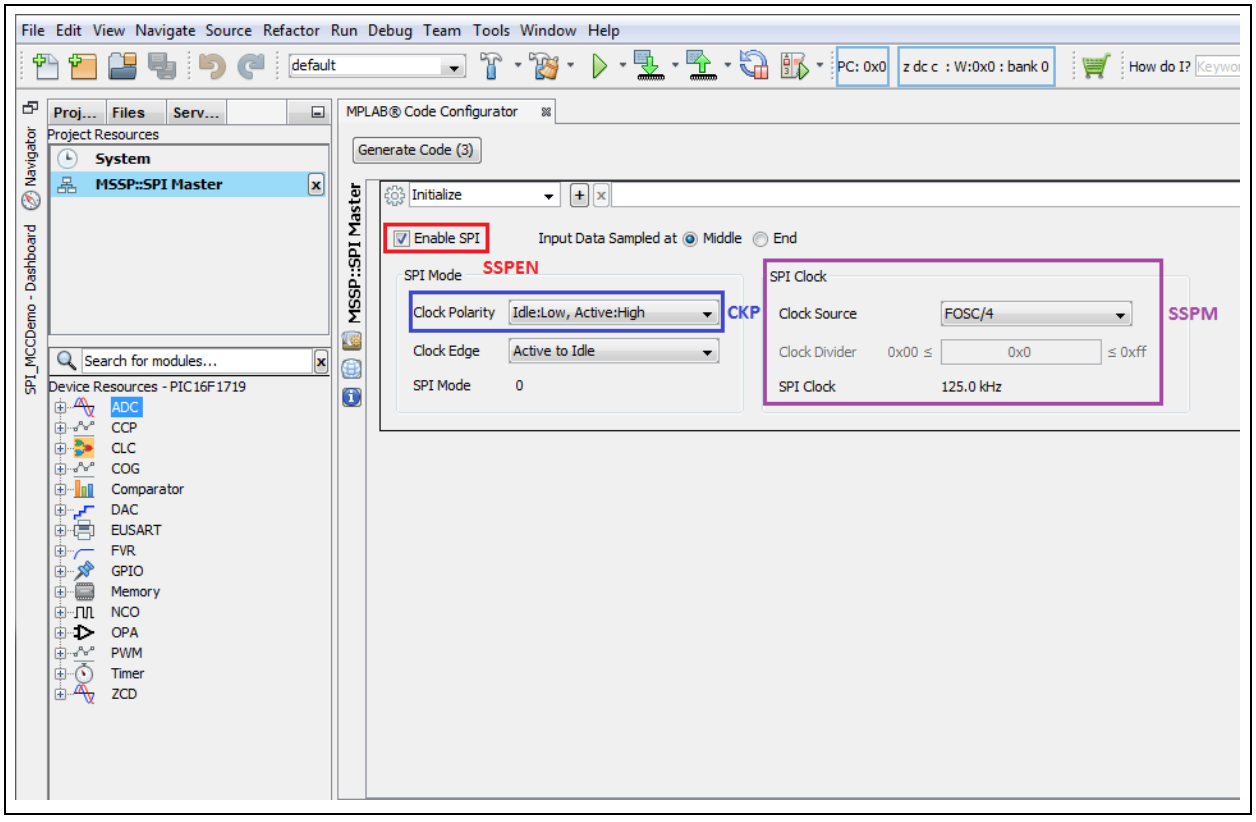
CKP 位决定时钟在空闲时为低电平还是高电平。SSPM<3:0> 位决定模块工作时所处的同步串行模式以及时钟首选项。图 4 显示了 SSPxCON1 寄存器中的位位置，图 5 显示了 MCC 的配置屏幕中与 SSPxCON1 寄存器中的位对应的部分。

图 4： SSPxCON：用于 SPI 配置的 SSPx 控制寄存器 1

REGISTER 30-2: SSP1CON1: SSP CONTROL REGISTER 1

R/C/HS-0/0	R/C/HS-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
WCOL	SSPOV(1)	SSPEN	CKP	SSPM<3:0>			
bit 7							bit 0

图 5： 在 MCC 上配置 SSPxCON1 位



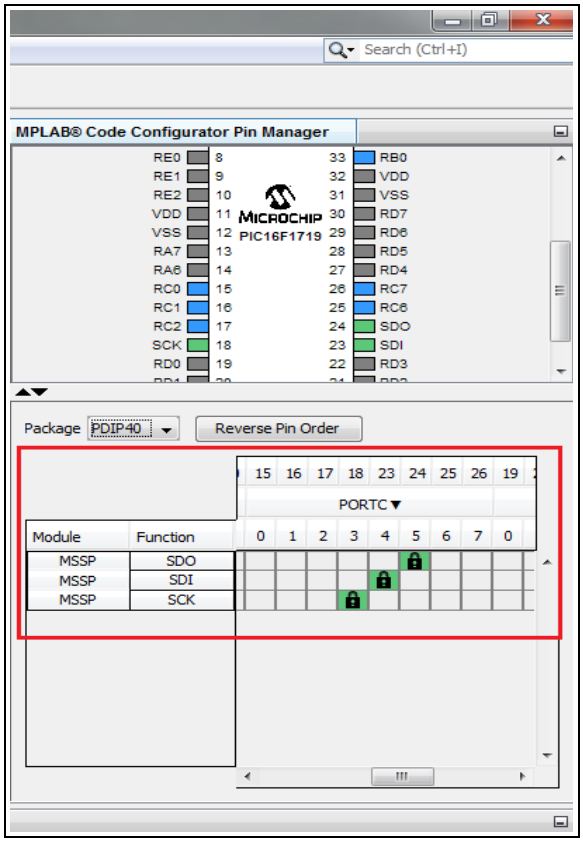
在 MCC 中单击 **Generate Code**（生成代码）按钮之后，图 5 所示的配置将在 `spi.c` 中生成以下代码行 （见例 2）。

例 2： SSPxCON1 MCC 生成的代码

```
//  SSPEN enabled; WCOL no_collision; SSPOV no_overflow; CKP Idle:Low, Active:High; SSPM FOSC/4;
    SSP1CON1 = 0x20;
```

对于具有外设引脚选择（Peripheral Pin Select，PPS）功能的器件（如 PIC16F1719），MCC 还会自动生成代码来将 SDO、SDI 和 SCK 引脚映射到在引脚管理器中选择的引脚，并相应地将它们初始化为输入或输出。对于没有 PPS 功能的其他器件，MCC 会根据需要将引脚配置为输入或输出。图 6 显示了选择作为 SDO、SDI 和 SCK 的引脚。例 3 显示了所生成的用于实现通过该器件的外设引脚选择（PPS）功能进行选择的代码片段。对于 CS 引脚，用户可以选择任意未用 I/O 引脚，并将它连接到从器件的 CS 引脚。选定的 CS 引脚在开始时应处于激活从器件所需的电平的相反电平（即，如果 CS 保持低电平会激活从器件，则它在复位后在开始时应为高电平）。这可以通过在 MCC 中在 GPIO 模块中选中相应的方框来实现，如图 7 中所示。此外，为了方便代码编写，还可以重命名该引脚。

图 6：　MSSP 引脚分配



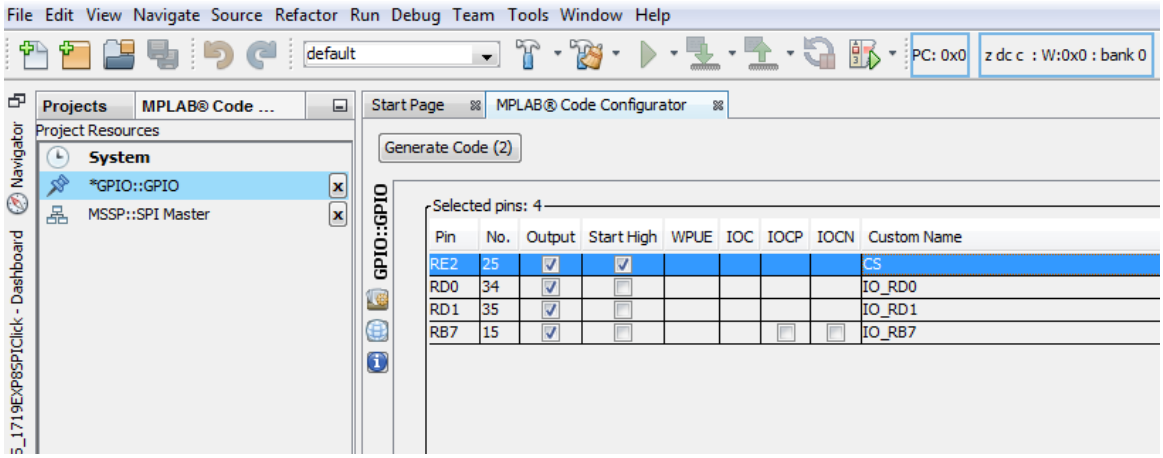
例 3：　MSSP 引脚分配代码

```
bool state = GIE;
    GIE = 0;
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00;
// unlock PPS

// RC3->MSSP:SCK
    SSPCLKPPSbits.SSPCLKPPS = 0x13;
// RC3->MSSP:SCK
    RC3PPSbits.RC3PPS = 0x10;
// RC4->MSSP:SDI
    SSPDATPPSbits.SSPDATPPS = 0x14;
// RC5->MSSP:SDO
    RC5PPSbits.RC5PPS = 0x11;

    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x01;
// lock PPS
    GIE = state;
```

图 7：　在 MCC 中设置 CS 引脚



常见的 SPI 串行 EEPROM 操作

表 1：　SPI 串行总线 EEPROM 的示例指令集

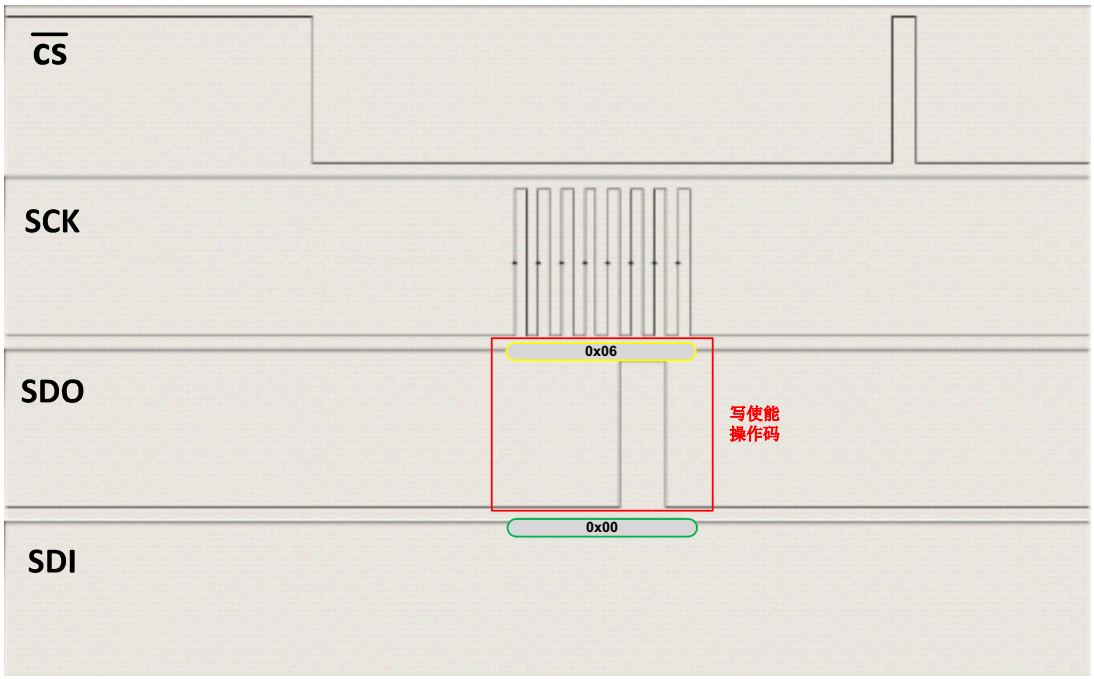
指令	说明	指令格式 / 操作码
WREN	写使能	0000 0110
WRDI	写禁止	0000 0100
RDSR	读状态寄存器	0000 0101
WRSR	写状态寄存器	0000 0001
READ	读存储器阵列	0000 0011
WRITE	写存储器阵列	0000 0010

写使能

为了开始与 EEPROM 进行交互，应激活 $\overline{CS}$ 线 —— 对于此处使用的 EEPROM，应置为低电平。这会指示从器件侦听主器件的 SCK 和 SDO 信号。 $\overline{CS}$ 线的电平跳变将结束主器件与从器件之间的所有事务。

要开始写入 EEPROM 阵列或状态寄存器，主器件必须发送 WRITE ENABLE 命令。状态寄存器的写使能（WEL）位通过发出 WRITE DISABLE 命令（WRDI）清零，如果器件掉电或写周期完成，它也会清零。图 8 给出了 WRITE ENABLE 命令的一个示例。

图 8： WRITE ENABLE 命令



对于该EEPROM，WRITE ENABLE命令操作码为0x06。关于 EEPROM 的具体命令操作码，请参见 EEPROM 的数据手册。

状态寄存器读操作

EEPROM 的状态寄存器包含一些指示 EEPROM 当前状态的位。用户可以跟踪的最重要指示位是 WEL （写使能）位和 WIP （写进行）位。如果 WEL 位置 1，则会使能对 EEPROM 的数据阵列的写操作。如果 WIP 位置 1，则说明写周期正在进行中。考虑到这一点，良好的编程做法是在尝试写或读操作之前先检查这些位，避免冲突。要读取 EEPROM 的状态寄存器，需要将 $\overline{CS}$ 线置为低电平，并发送 EEPROM 读状态寄存器（Read Status Register，RDSR）操作码（对于该EEPROM为0x05）。然后，在 EEPROM 从器件的 SDO 引脚上送出状态寄存器的内容，并在下一个后续时钟送入主器件的SDI引脚。图 9 和图 10 说明了如何使用 RDSR 命令来检查 WEL 和 WIP 位是否置 1。

图 9： 读状态寄存器命令 （WEL 位置 1）

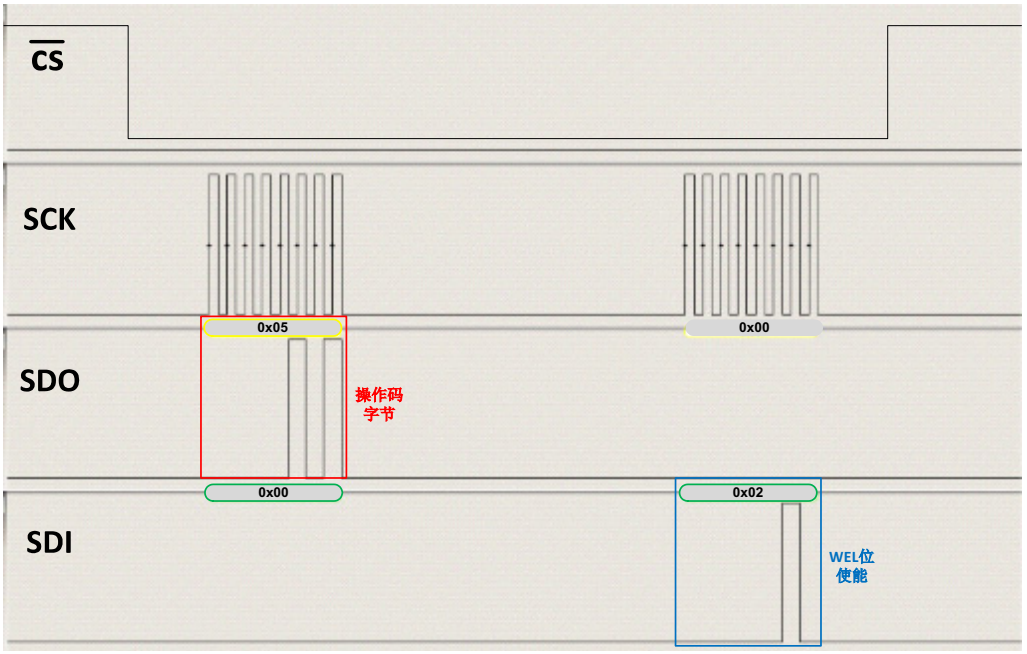
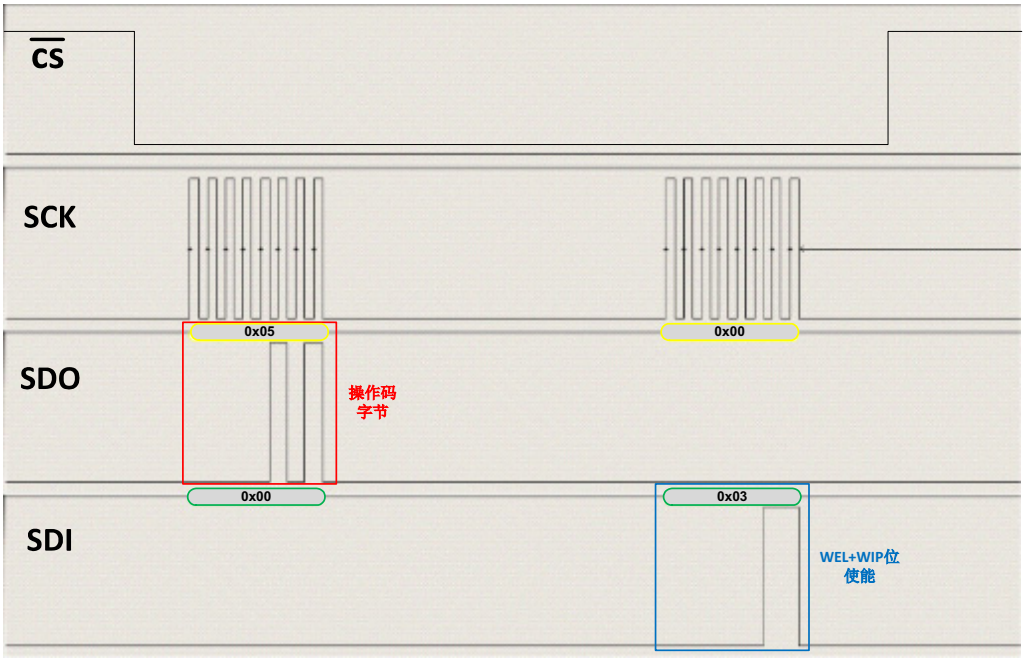


图 10： 读状态寄存器命令 （WEL+WIP 位置 1）



字节写操作和缓冲区写操作

当WEL位置1且$\overline{CS}$线置为低电平时，需要发送EEPROM写操作码（对于该 EEPROM 为 0x02），后跟目标起始地址字节，先发送高字节（Most Significant Byte，MSB）。最后送入数据字节。该命令结束时会切换 $\overline{CS}$ 线，从而启动内部写周期。

现在可以查询状态寄存器的 WIP 位，检查写操作是否已完成（稍后介绍更多信息）。

可以通过在不切换 $\overline{CS}$ 线的情况下向 EEPROM 器件连续发送数据字节来写入多个字节。但是，用户应注意器件的页大小和起始地址，避免覆盖先前存储的数据。超过所分配页大小的数据会回绕到页起始地址处，覆盖可能已在其中写入的数据。图 11 和图 12 说明了如何向 EEPROM 的阵列中写入单个字节和多个字节的数据。

图 11： 字节写命令

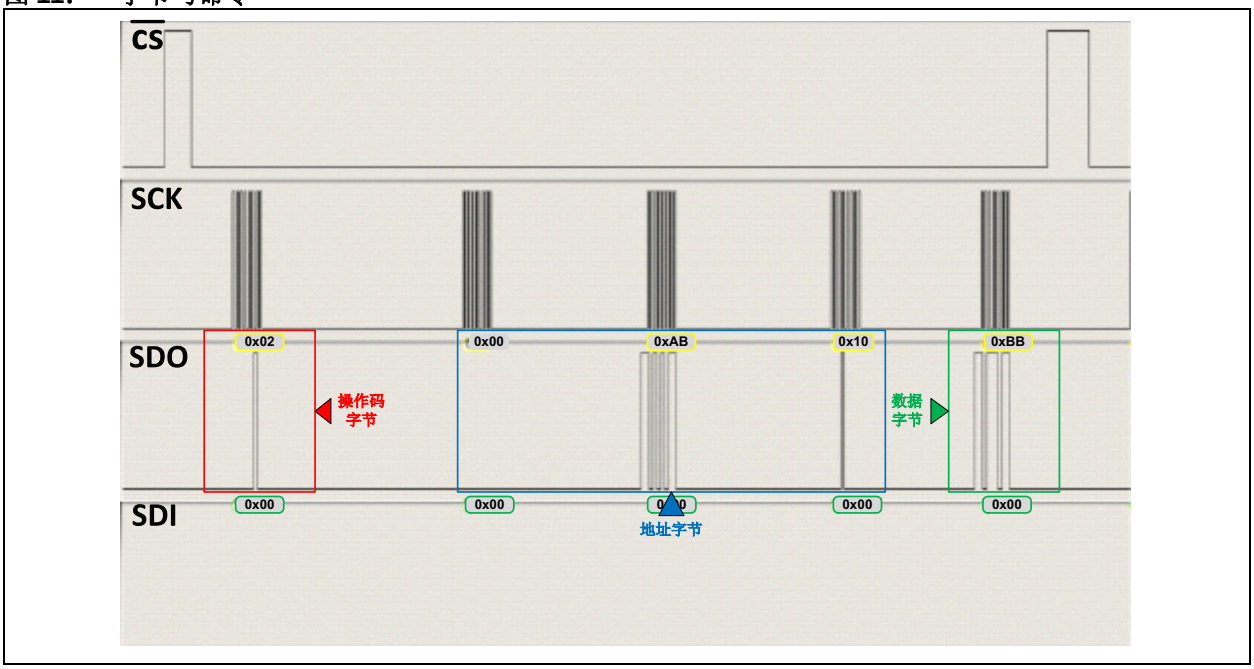
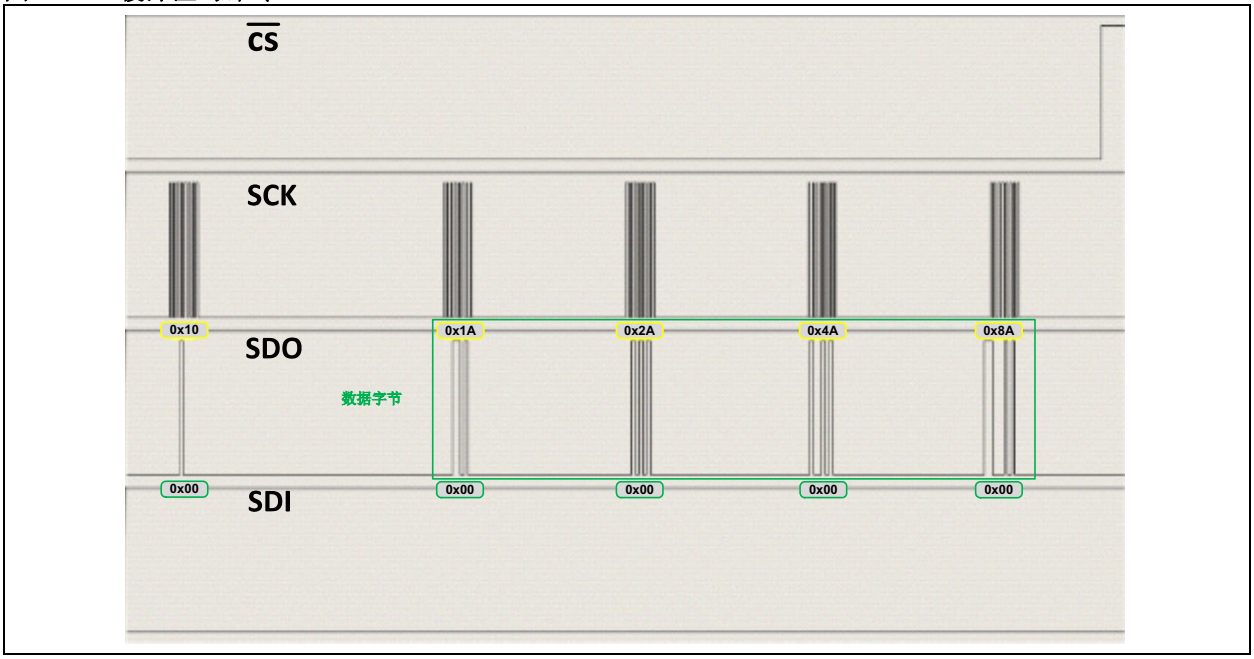


图 12： 缓冲区写命令



字节读操作和缓冲区读操作

要从 EEPROM 读取，需要将 $\overline{CS}$ 线置为低电平，并发送 EEPROM 读操作码（对于该 EEPROM 为 0x03），后跟目标起始地址字节，先发送高字节。代码通过向 SDO 线发送由零组成的伪数据来从SDI线送出数据，因为SPI是一种数据交换协议。该命令结束时会切换 $\overline{CS}$ 线，从而结束传输。多字节读操作的实现方式是，向 $\overline{CS}$ 器件连续发送伪数据字节，从而在不切换 $\overline{CS}$ 线的情况下为它提供送入数据所需的时钟周期。图 13 和图 14 说明了如何从 EEPROM 的阵列中读取单个字节和多个字节的数据。

图 13： 读字节命令

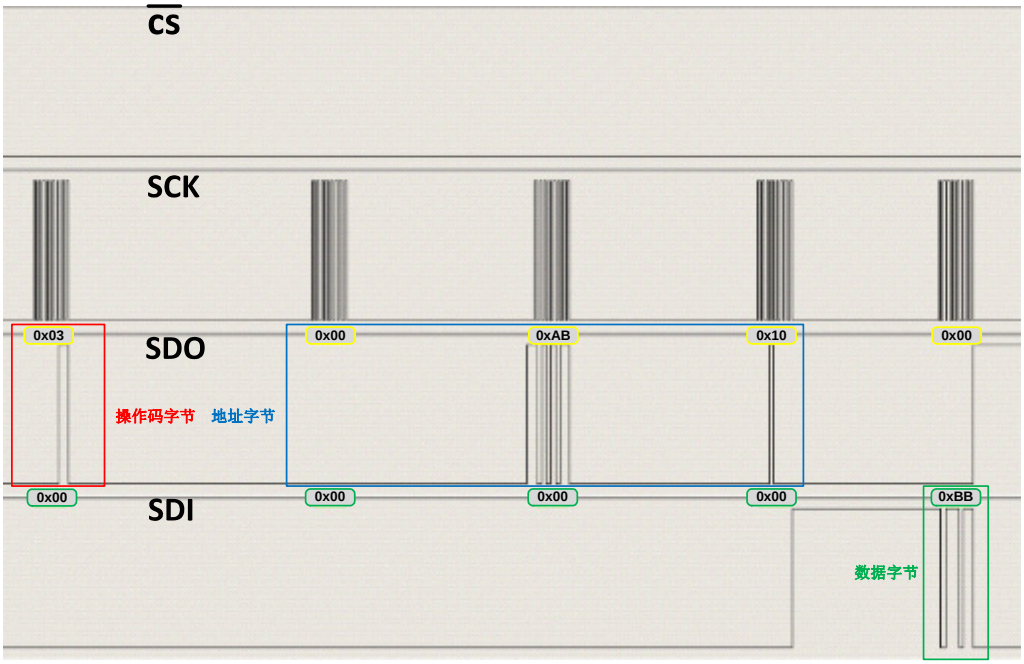
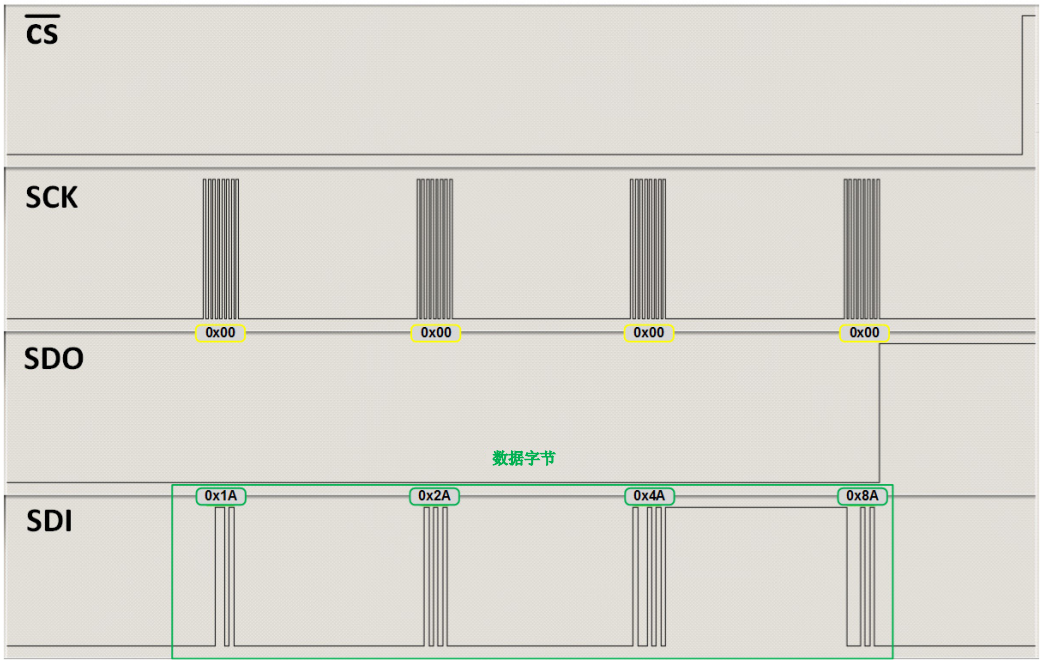


图 14： 缓冲区读命令



I²C 接口

I²C 协议具有 SPI 的前两项特性：同步并设计为一种主器件 / 从器件协议。类似于 SPI，I²C 也使用时钟信号来帮助进行数据传输。除非存在时钟信号，否则不会发生数据传输。虽然主器件仍然是提供时钟的器件，但从器件可以操控时钟，即如果它仍忙，可以通过将时钟保持为低电平来阻止进一步的数据传输 （时钟延长）。这与不允许从器件操控时钟信号的 SPI 主器件截然相反。此外， SPI 协议以全双工模式工作，允许同时从主器件和从器件发送数据。当 I²C 协议在半双工模式下工作时，允许主器件和从器件发送数据，但是不允许在同一时间发送数据。这通过“应答”系统来实现。另一项差异是 SPI 协议要求每个从器件都连接片选 （CS）线。对于 I²C，无论有多少个从器件，主器件和从器件之间都只需连接两条信号线，因为从器件地址是通过数据线而不是使用 CS 连接发送的。虽然 I²C 协议使用的引脚更少，但 SPI 协议通常速度更快。

对于本部分应用说明中的示例和波形，使用了 PIC16F1719 单片机作为主器件、装在 MikroElectronika EEPROM Click 电路板上的 8 Kb 串行 I²C 总线 EEPROM 作为从器件。

标准 I²C 信号

I²C协议仅使用两条信号线来控制通信系统中的数据流。

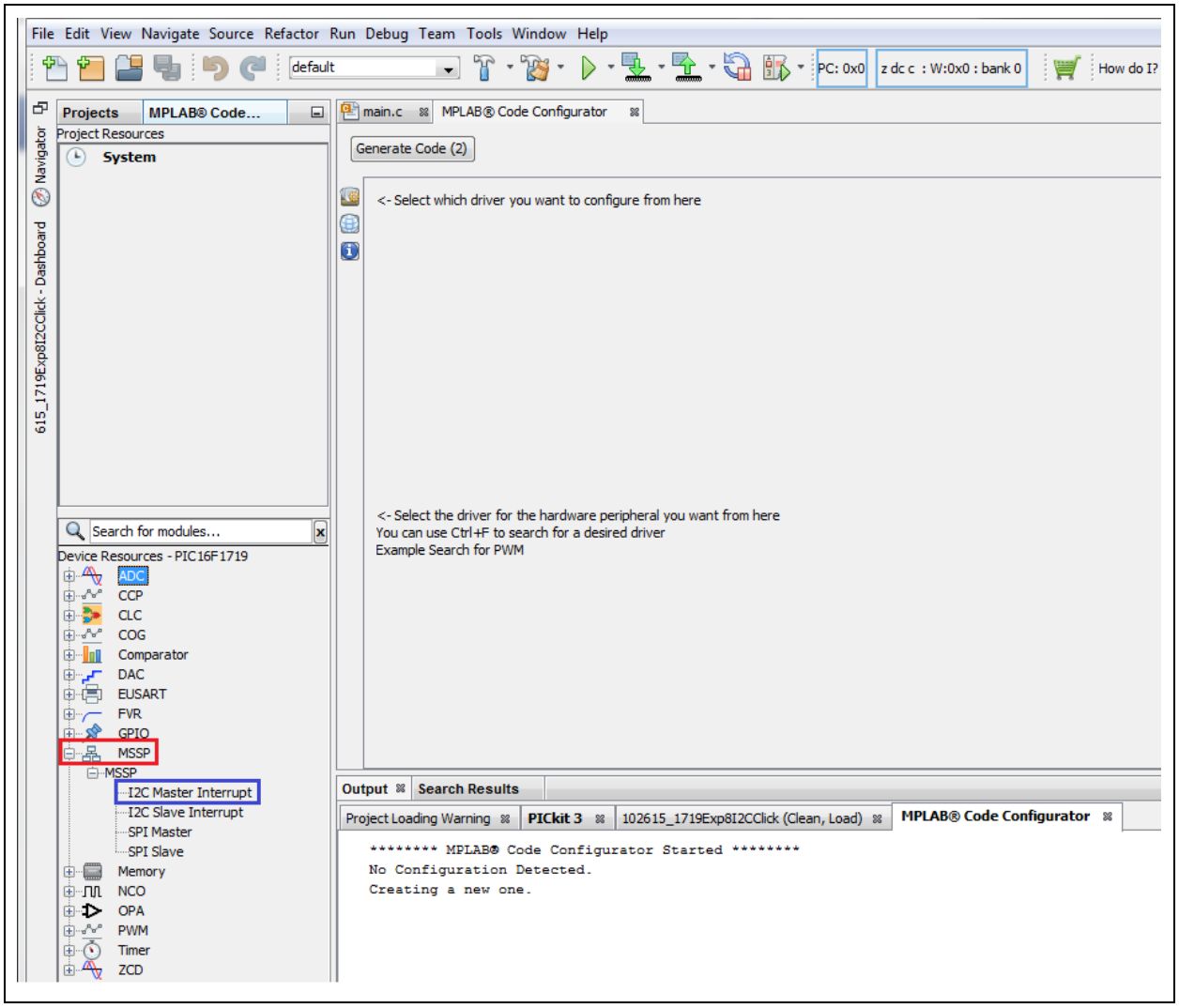
- 串行时钟线 （Serial Clock Line， SCL）
  - 它是由主器件产生的时钟信号，用于控制何时发送和接收数据。
  - 任何从器件如果由于太忙而无法接受或发送数据，则可将它保持为低电平。
- 串行数据线 （Serial Data Line， SDA）
  - 它是携载在主器件和从器件之间送入和送出的数据的信号线。

通过 MPLAB® 代码配置器 （MCC）进行 I²C MSSP 初始化

本部分将指导用户使用 MPLAB 代码配置器 （MCC）来初始化适当的寄存器，以使用大多数 PIC 器件中提供的 MSSP 模块来实现 I²C。

在打开一个新项目并启动 MCC 插件后，在 Device Resources （器件资源）侧边栏中选择 MSSP 模块，它在图 15 中标记为红色。从下拉选项中，选择 **I²C Master Interrupt** （I²C 主器件中断）作为将用作主器件的单片机，如本部分的简介中所述。它在图 15 中标记为蓝色。

图 15： 在 MCC 中选择 MSSP 模块



为了将 PIC 器件中的 MSSP 模块配置为 I²C 操作，需要正确地初始化几个寄存器。

SSPx 状态寄存器 （SSPxSTAT）

在 I^2C 主模式下，R/$\overline{W}$ 位指示是否正在发送。BF 位指示数据传输是否已经完成。图 16 显示了 SSPxSTAT 寄存器中的位位置。

图 16： SSPxSTAT：用于 I^2C 配置的 SSPx 状态寄存器

REGISTER 30-1: SSP1STAT: SSP STATUS REGISTER

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

以下代码行用于对 SSPxSTAT 寄存器进行初始化。当按下 **Generate Code** 按钮时，MCC 会自动生成该代码（见例 4）。

例 4： SSPxSTAT MCC 生成的代码

```
//  BF RCinprocess_TXcomplete; UA dontupdate;P stopbit_notdetected; S startbit_notdetected;
    R_nW write_noTX; D_nA lastbyte_address;
    SSP1STAT = 0x00;
```

SSPx 控制寄存器 1 （SSPxCON1）

在 I²C 模式下，需要通过将 SSPEN 位置 1 来使能串行端口，并将 SCL 和 SDA 配置为串行端口引脚的信号源。

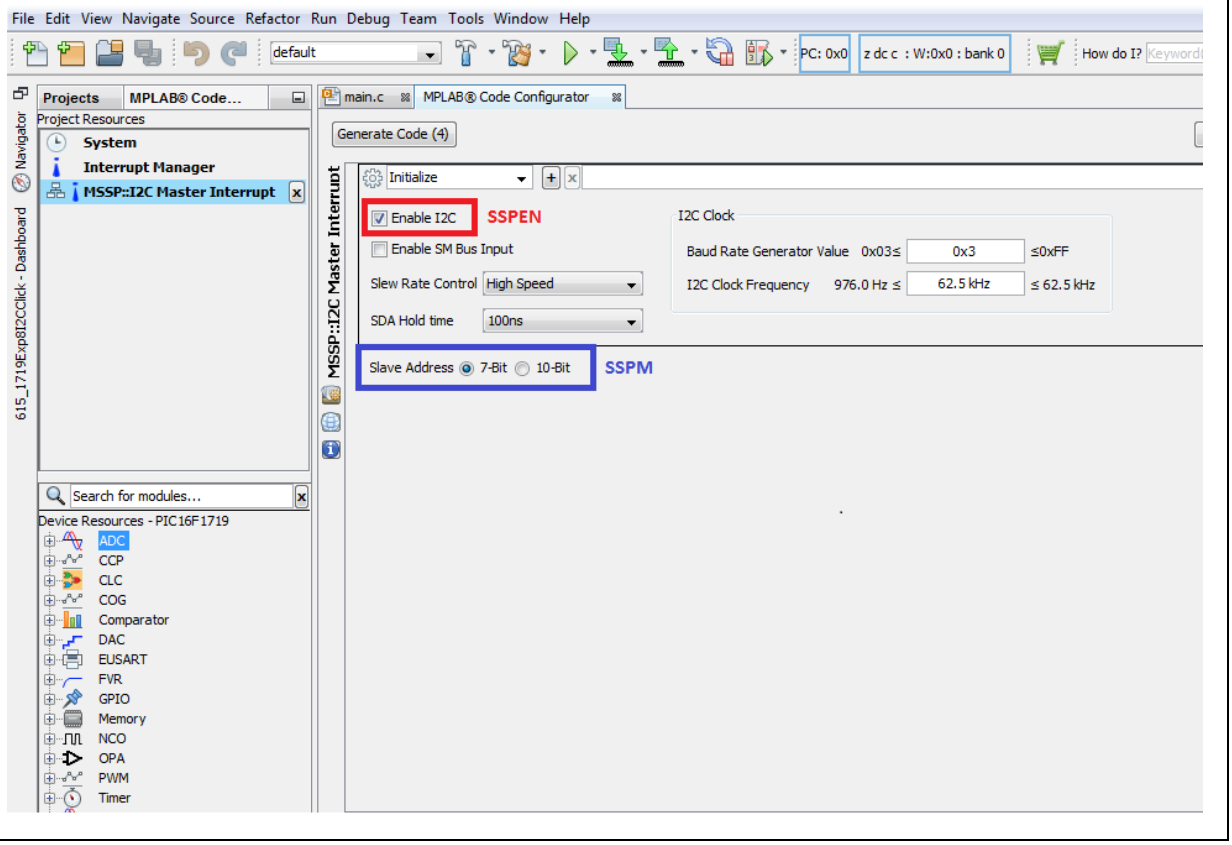
SSPM<3:0> 位决定模块工作时所处的同步串行模式，以及时钟首选项和模块处于从模式时用于从器件地址的位数。图 17 显示了 SSPxCON1 寄存器中的位位置，图 18 显示了 MCC 的配置屏幕中与 SSPxCON1 寄存器中的位对应的部分。在该示例中，I²C 串行总线 EEPROM 使用 7 位寻址。

图 17： SSPxCON1：用于 I²C 配置的 SSPx 控制寄存器 1

REGISTER 30-2: SSP1CON1: SSP CONTROL REGISTER 1

R/C/HS-0/0	R/C/HS-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
WCOL	SSPOV(1)	SSPEN	CKP	SSPM<3:0>			
bit 7							bit 0

图 18： 在 MCC 上配置 SSPxCON1 位



在 MCC 中单击 **Generate Code** 按钮之后，图 18 所示的配置将在 `i2c.c` 中生成以下代码行 （见例 5）。

例 5： SSPxCON1 MCC 生成的代码

```
// SSPEN enabled; WCOL no_collision; SSPOV no_overflow; CKP Idle:Low, Active:High; SSPM FOSC/4_SSPxADD;
   SSP1CON1 = 0x28;
```

SSPx 控制寄存器 3 （SSPxCON3）

该寄存器主要包含 I²C 功能的控制位。此处关注的是 SDAHT 位，它用于控制在 SCL 下降沿之后的 SDA 保持时间的长度。

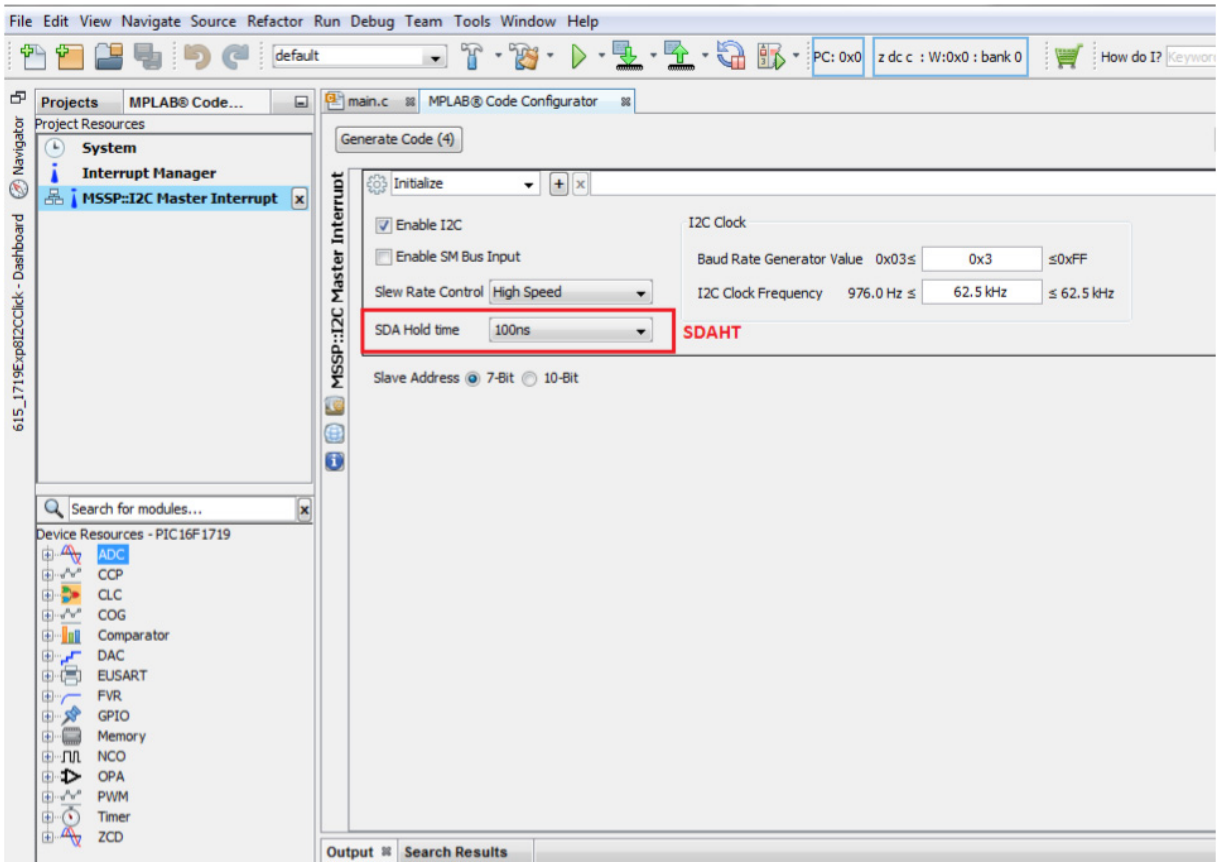
图 19 显示了 SSPxCON3 寄存器中的位位置，图 20 显示了 MCC 的配置屏幕中与 SSPxCON3 寄存器中的位对应的部分。

图 19： SSPxCON3：用于 I²C 配置的 SSPx 控制寄存器 3

REGISTER 30-4: SSP1CON3: SSP CONTROL REGISTER 3

R-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
ACKTIM(3)	PCIE	SCIE	BOEN	SDAHT	SBCDE	AHEN	DHEN
bit 7							bit 0

图 20： 在 MCC 上配置 SSPxCON3 位



在 MCC 中单击 **Generate Code** 按钮之后，图 20 所示的配置将在 `i2c.c` 中生成以下代码行 （见例 6）。

例 6： SSPxCON3 MCC 生成的代码

```
//  BOEN disabled; AHEN disabled; SBCDE disabled; SDAHT 100ns; DHEN disabled; ACKTIM ackseq;
    PCIE disabled; SCIE disabled;
    SSP1CON3 = 0x00;
```

SSPx 地址和波特率寄存器 （SSPxADD）

对于 I²C 主模式，该寄存器包含波特率时钟分频器的值（即 SCL 时钟周期）。它通过以下公式计算：
((ADD<7:0> + 1) * 4)/FOSC。

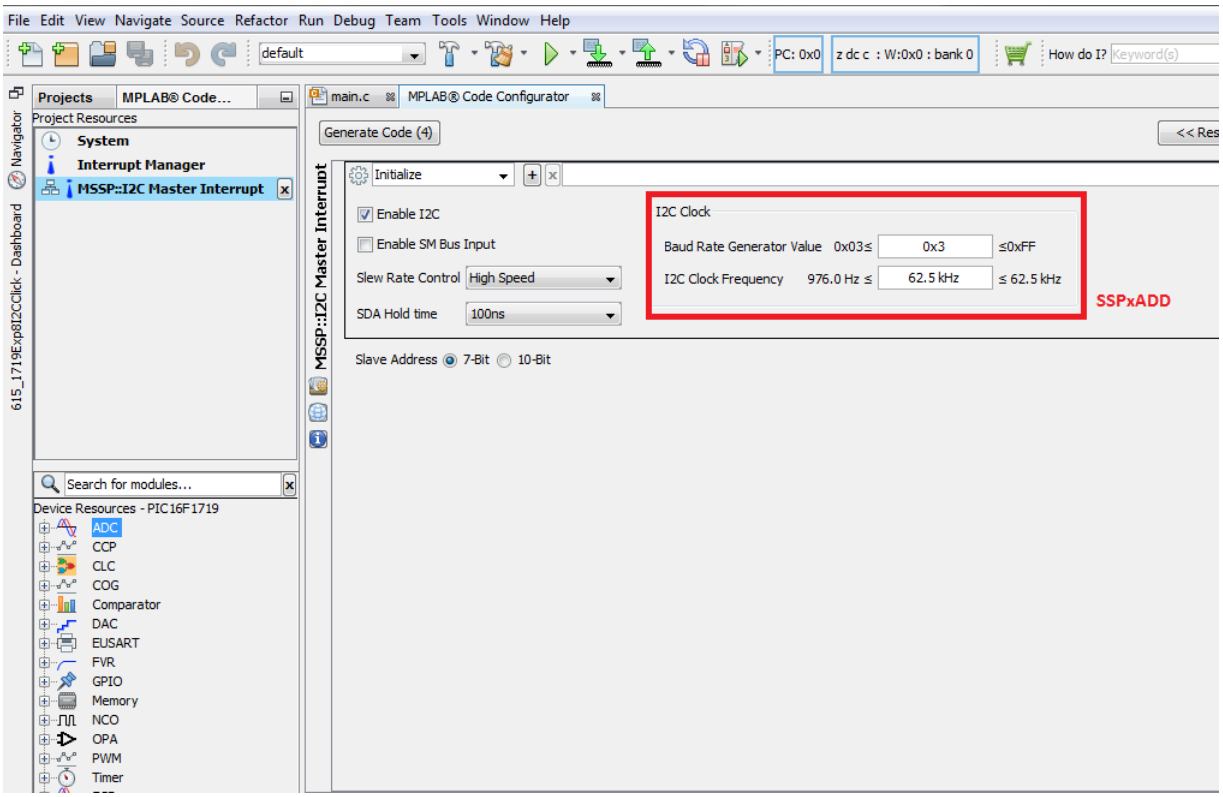
图 21 显示了 SSPxADD 寄存器中的位位置。图 22 显示了 MCC 的配置屏幕中与 SSPxADD 寄存器中的位对应的部分。

图 21： SSPxADD：SSPx 地址和波特率寄存器

REGISTER 30-6: SSP1ADD: MSSP ADDRESS AND BAUD RATE REGISTER (I²C MODE)

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
ADD<7:0>							
bit 7							bit 0

图 22： 在 MCC 上配置 SSPxADD 位



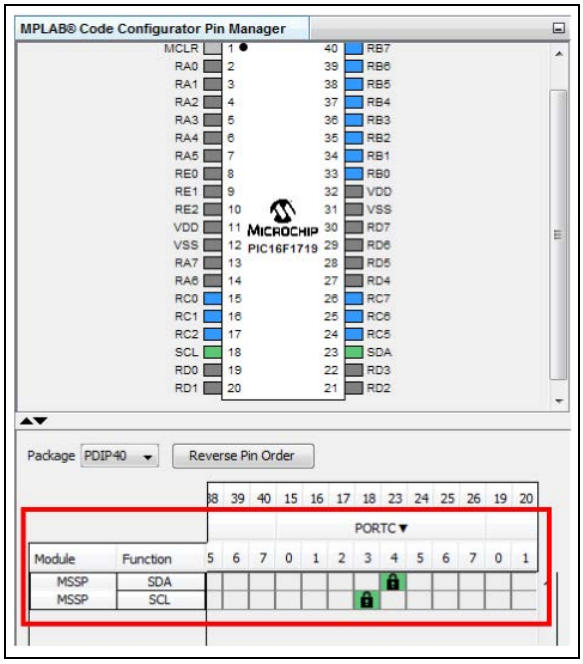
在 MCC 中单击 **Generate Code** 按钮之后，图 22 所示的配置将在 `i2c.c` 中生成以下代码行 （见例 7）。

例 7： SSPxADD MCC 生成的代码

```
//  Baud Rate Generator Value:SSP1ADD 3;
    SSP1ADD = 0x03;
```

对于具有外设引脚选择（PPS）功能的器件（如 PIC16F1719），MCC 还会自动生成代码来将 SCL 和 SDA 引脚映射到在引脚管理器中选择的引脚，并相应地将这些引脚配置为输入或输出。图 23 显示了选择用作 SCL 和 SDA 的引脚。例 8 显示了所生成的用于实现通过该器件的外设引脚选择（PPS）功能进行选择的代码片段。另外请注意，由 MCC 提供的 I^2C 功能驱动程序是基于中断的，因此用户必须允许全局和外设中断才能使其正确工作。要允许全局和外设中断，请在 MCC 生成的 `main.c` 文件中，取消注释实现 `INTERRUPT_GlobalInterruptEnable()` 和 `INTERRUPT_PeripheralInterruptEnable()` 函数的行，如图 24 中所示。

图 23： I^2C MSSP 引脚分配



例 8： MSSP 引脚分配代码

```
bool state = GIE;
    GIE = 0;
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00;      // unlock PPS

    SSPCLKPPSbits.SSPCLKPPS = 0x13;    // RC3->MSSP:SCL

    RC3PPSbits.RC3PPS = 0x10;
//  RC3->MSSP:SCL

    SSPDATPPSbits.SSPDATPPS = 0x14;    // RC4->MSSP:SDA

    RC4PPSbits.RC4PPS = 0x11;
//  RC4->MSSP:SDA

    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x01;      // lock PPS
    GIE = state;
```

图 24：　允许中断的 main.c 文件的说明

```
void main(void) {
    // initialize the device
    SYSTEM_Initialize();

    // When using interrupts, you need to set the Global and Peripheral Interrupt Enable bits
    // Use the following macros to:

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();

    // Disable the Global Interrupts
    //INTERRUPT_GlobalInterruptDisable();

    // Disable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptDisable();
```

常见的 I²C 串行 EEPROM 操作

- 字节写操作
- 多字节写操作
- 页写操作
- 应答查询
- 写保护
- 地址读操作
- 顺序读操作

字节写操作

I²C 中的字节写操作可以拆分为以下要素：启动条件、I²C 从器件地址字节、EEPROM 地址字节、数据字节和停止条件。对于该 EEPROM，仅使用单字节的地址数据。其他 EEPROM 可能会使用多字节地址。

启动位和 I²C 从器件地址字节发送

所有 I²C 命令都必须以启动条件开始。该条件由 SCL 线为高电平时 SDA 线的由高至低跳变构成。在启动条件后，发送 I²C 从器件地址字节，它包含器件 7 位 I²C 从器件地址（对于该 EEPROM 为 0xA）和用于标识要执行的操作的读 / 写位。对于写操作，R/W 位会被拉为低电平。

在每个字节后，在第 9 个时钟周期时，EEPROM 从器件会将 SDA 线保持为低电平，以指示它已接收到先前的位。这是应答或 ACK 位。

发送 EEPROM 地址字节

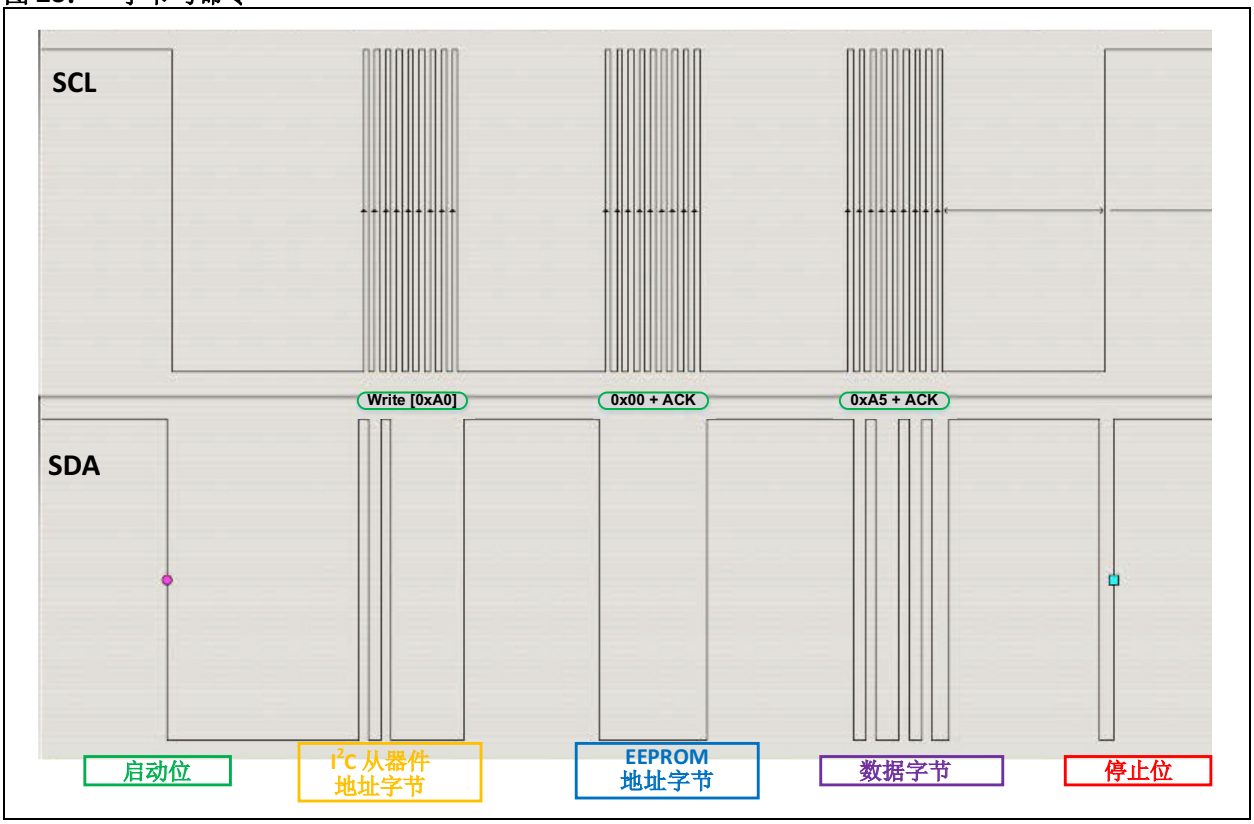
在 I²C EEPROM 从器件产生已接收到 I²C 地址的应答后，主器件应开始发送EEPROM地址字节。在EEPROM使用多字节地址的情况下，应按从高到低的顺序发送字节。对于主器件发送的每个字节，EEPROM应使用ACK位进行响应。

发送数据字节和停止位

在发送 EEPROM 地址字节和接收到 ACK 之后，即可发送数据字节。在主器件发送每个字节之后，EEPROM 应通过发送 ACK 位来进行响应。此后，主器件将产生停止条件，指示它不需要写入更多字节。停止条件通过在 SCL 线为高电平时在 SDA 线上产生由低至高的跳变来实现。

图 25 显示了从启动条件到停止条件的完整字节写操作过程。在此例中，EEPROM 仅使用单个字节进行寻址。0x00 用作要写入数据的位置的地址，0xA5 用作要发送的数据字节。

图 25： 字节写命令



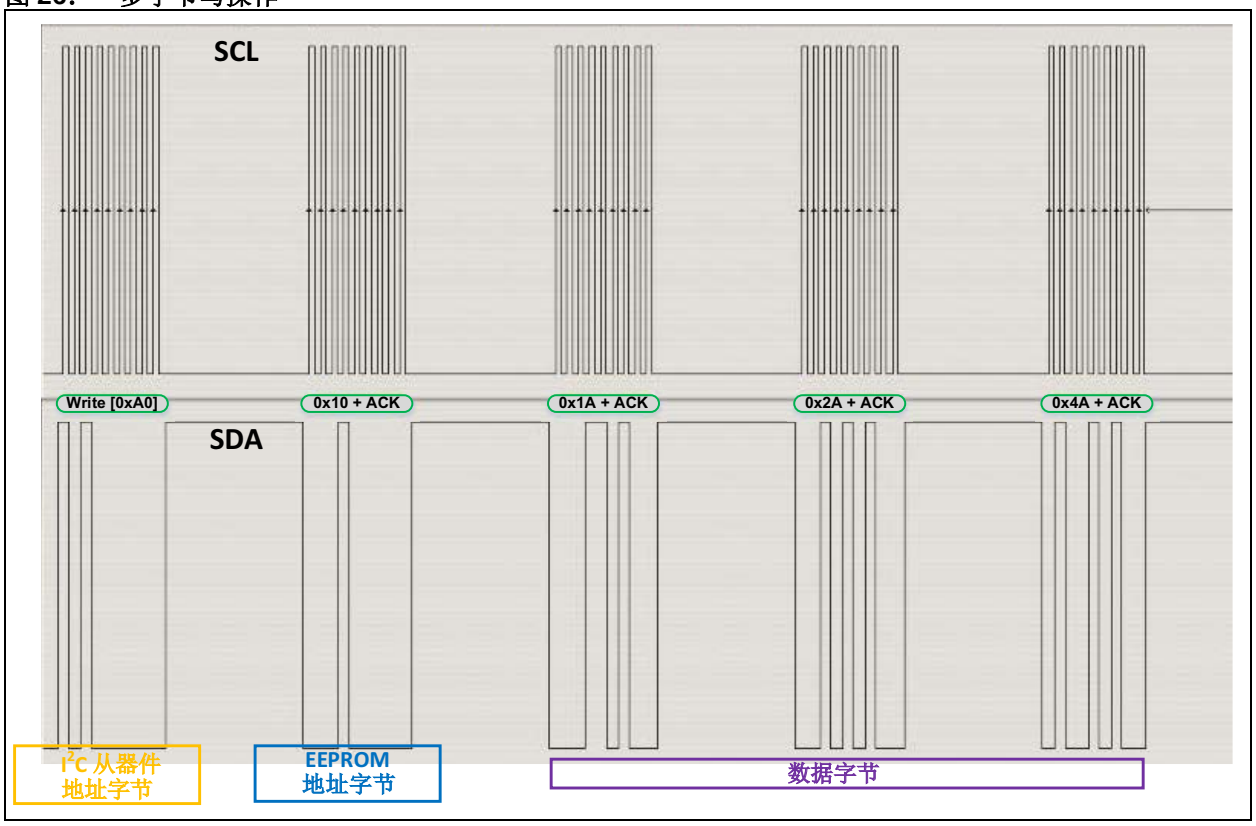
多字节写操作和页写操作

写入多个字节在开始时与字节写操作非常相似。启动位、I²C 从器件地址字节和 EEPROM 地址字节全部按相同的顺序进行发送和应答。但是，在发送和应答第一个数据字节之后，主器件不会发送停止条件，而是继续连续发送更多数据字节。该 EEPROM 最多可接受在单个写周期内写入 16 个字节。EEPROM 的页大小决定在仅发送一次 I²C 从器件地址和 EEPROM 地址字节的情况下，可以连续写入多少字节的数据。需要指出非常重要的一点是，无论实际写入多少字节，页写操作仅限于在单个物理页内写入字节。

这意味着如果写入的字节数超过页大小限制，多余的字节会回绕到页起始地址处，覆盖已在其中写入的数据。发送所有字节后，主器件会启动停止条件，从而开始内部写周期。

图 26 显示了在缓冲区写操作中发送的前 3 个字节。请注意，在 EEPROM 产生已接收到字节的应答后，主器件会立即开始发送下一个字节。

图 26： 多字节写操作



字节读操作

I²C 中的字节读操作在开始时类似于字节写操作，即，应先将 EEPROM 地址写入从器件，然后 EEPROM 从器件才能发送来自该地址的数据。先发送启动条件，然后发送 I²C 从器件地址字节，再发送 EEPROM 地址字节。在发送 EEPROM 地址字节之后，发送停止条件而不是数据字节。在 I²C 主器件使用一个新的启动条件启动读操作后，发送 LSB 被拉为高电平 （指示读操作）的 I²C 从器件地址字节。然后， I²C 主器件发送 9 个时钟脉冲，而从器件发送所需的单个字节的数据。

启动位和 I²C 从器件地址字节发送

所有 I²C 命令都必须以启动条件开始。该条件由 SCL 线为高电平时 SDA 线的由高至低跳变构成。在启动条件后，发送 I²C 从器件地址和方向字节，它包含器件 7 位 I²C 从器件地址 （对于该 EEPROM 为 0xA）和用于决定要执行的操作的读 / 写位。

要启动字节读操作，必须将目标 EEPROM 数据地址写入 EEPROM。

要实现该操作，需要将 R/W 位设置为低电平，指示 I²C 主器件将要写入从器件。在主器件写入每个字节后，从器件会将 SDA 线保持为低电平，指示已接收到前面的字节。

发送地址字节

在 EEPROM 产生已接收到 I²C 从器件地址字节的应答后，主器件应开始发送EEPROM地址字节。在EEPROM使用多字节地址的情况下，应按从高到低的顺序发送字节。在主器件发送每个字节后， EEPROM 应使用 ACK 位进行响应。

发送停止位

在读操作中发送地址字节并接收到 ACK 之后，主器件即会发送停止位。EEPROM 应通过发送 ACK 位来进行响应。

接收数据字节

产生一个新的启动条件，并再次发送 I²C 从器件地址字节，但此次 R/W 位设置为高电平，指示主器件希望执行读操作。在 I²C 从器件应答 I²C 从器件地址字节之后，I²C 主器件会发送 9 个时钟脉冲，而从器件会通过发送数据字节来进行响应。在主器件接收到数据字节后，它会在第 9 个时钟周期释放 SDA 线。这是一个无应答（NACK）条件。该条件指示主器件不再向从器件请求数据。

图 27 和图 28 显示了完整的字节读操作过程。图 27 说明了如何将要读取数据的位置的 EEPROM 地址发送到从器件。图 28 显示了读命令和从器件发送的数据字节。在此例中， EEPROM 仅使用单个字节进行寻址。0x00 用作要写入数据的位置的 EEPROM 地址，0xA5 用作要发送的数据字节。

图 27： 发送读命令的地址

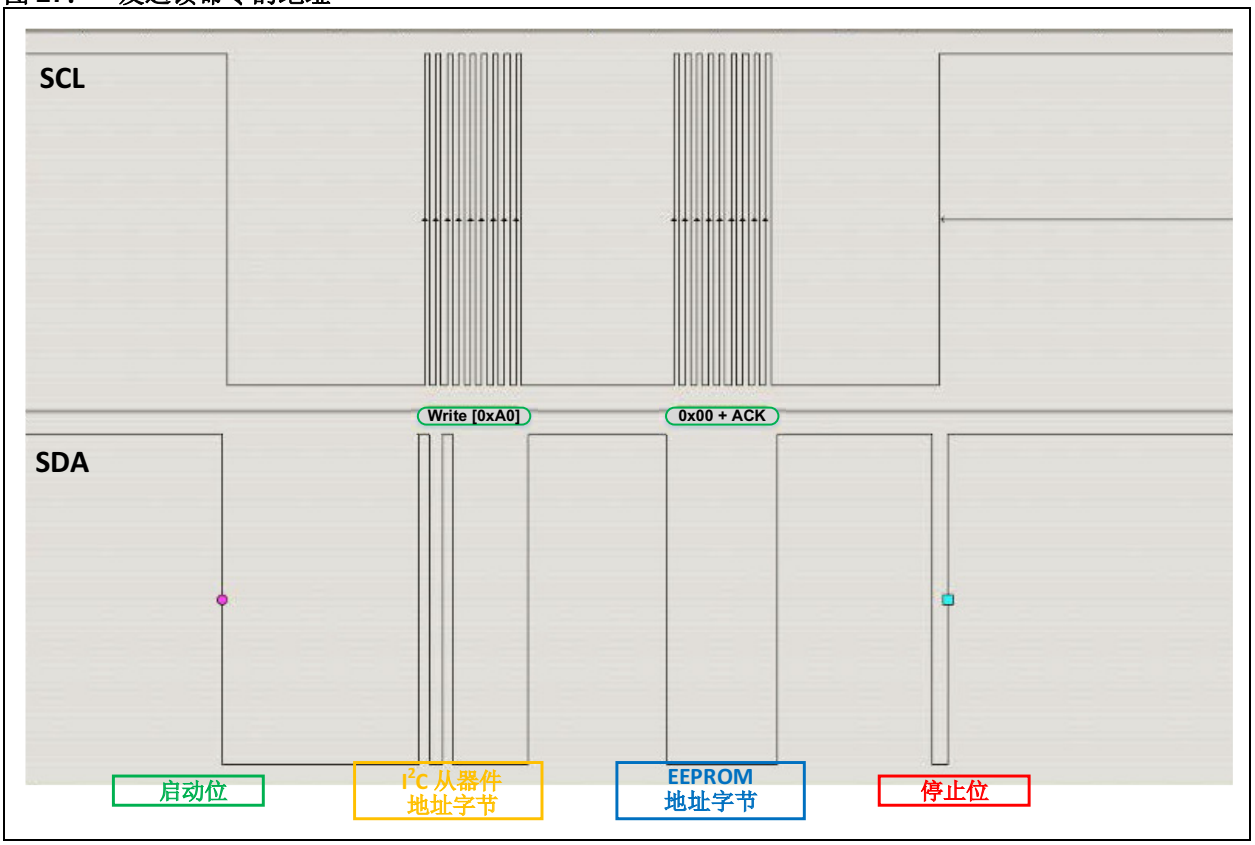
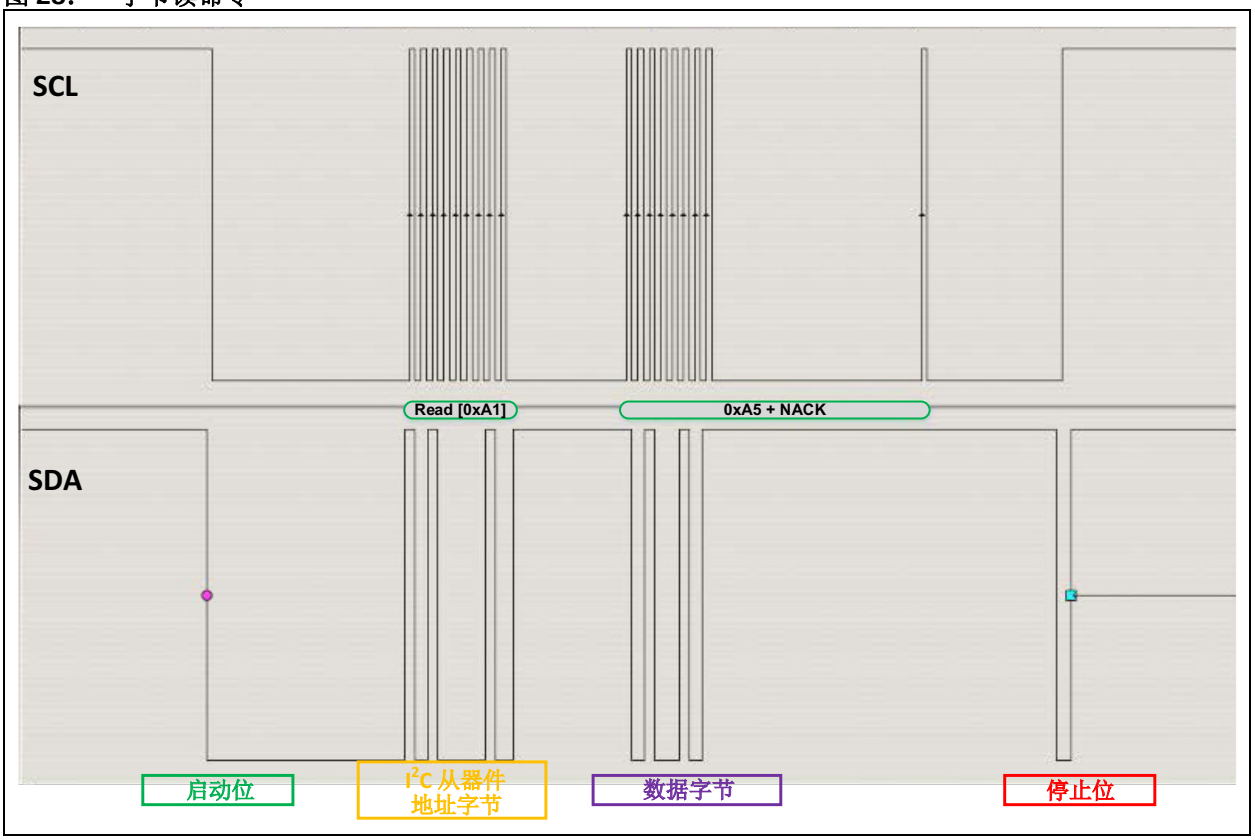


图 28： 字节读命令



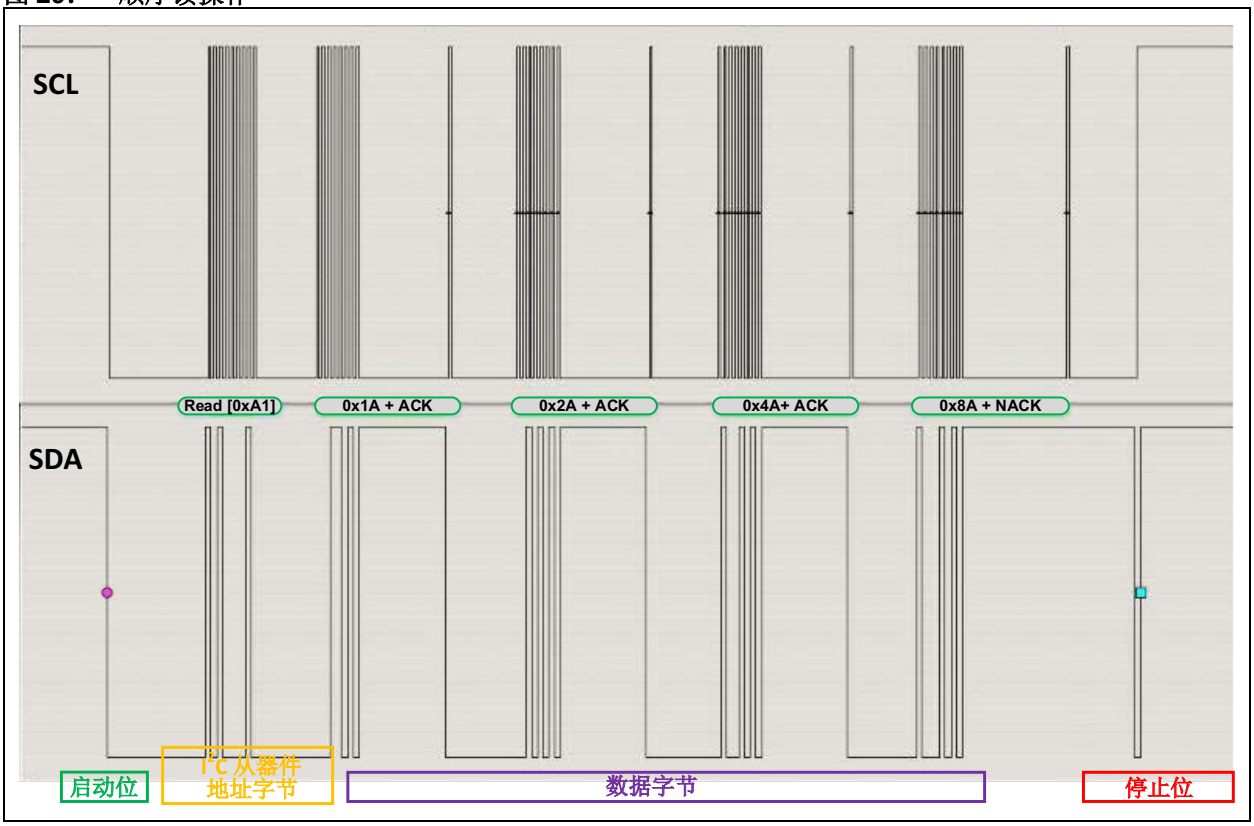
多字节 / 顺序读操作

不同于会受器件物理页大小限制的页写操作，顺序读操作可以在单个操作中读取存储器的全部内容。读取多个字节在开始时与字节读操作非常相似。启动位、I^2C 从器件地址字节、EEPROM 地址字节和停止条件全部按相同的顺序进行发送和应答。产生一个新的启动条件，并且应发送 LSB 设置为高电平的 I^2C 从器件地址字节。

但是，在发送第一个字节后，I^2C 主器件不会发送 NACK 位，而是会通过将信号线拉为低电平来发送 ACK 位，指示主器件还要请求更多数据。除最后一个字节外，主器件会在接收到其他每个字节后发送 ACK 位；对于最后一个字节，它会发送 NACK 位，指示主器件不再请求发送更多数据。在接收到所有字节后，主器件会启动停止条件，结束操作。

图 29 显示了顺序读操作。请注意，除最后一个字节之外，其他成功发送的每个字节后都跟随一个 ACK 位，最后一个字节会跟随一个 NACK 位。

图 29： 顺序读操作



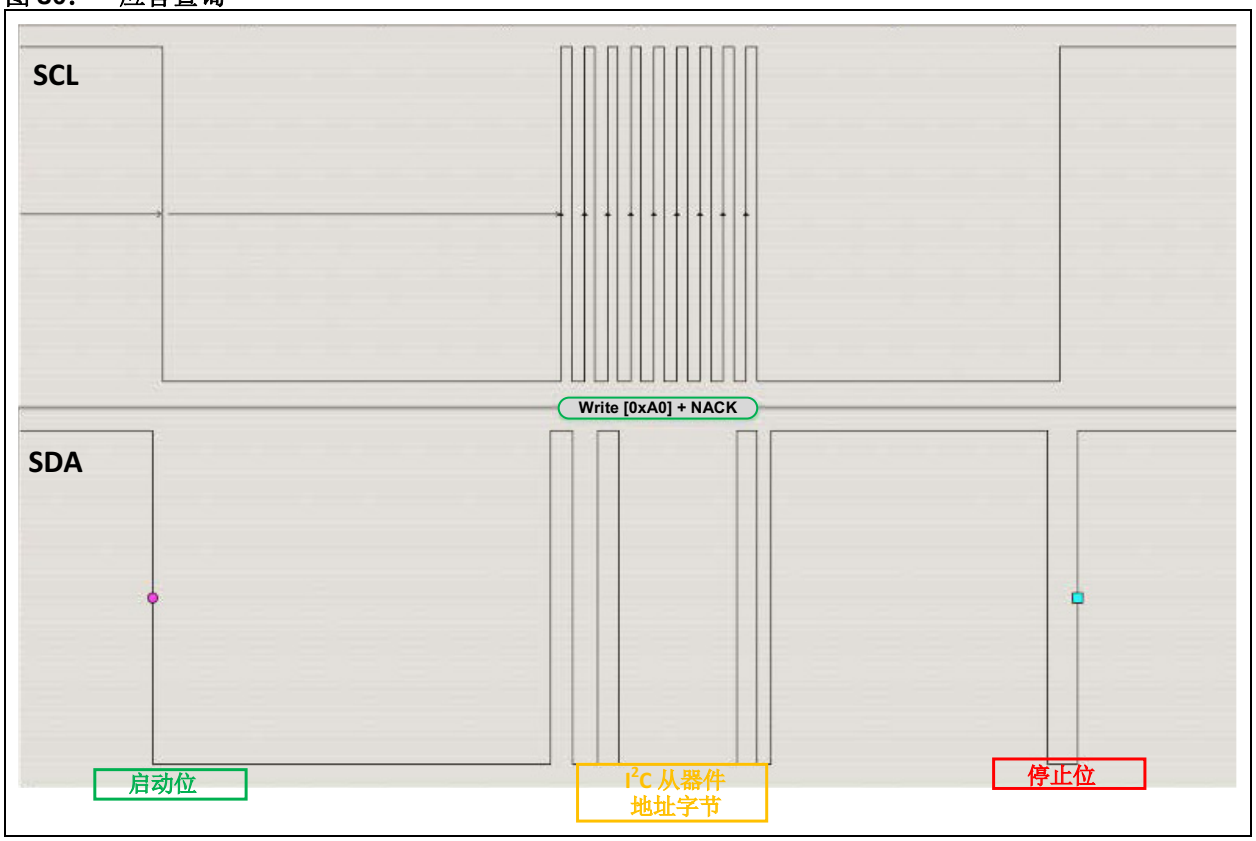
应答查询

虽然大多数 EEPROM 数据手册都规定了写周期时间，但一些写周期可能会比该周期短。因此，指定延时周期可能不太有效。建议用户使用应答查询来检查当前写周期是否已完成。

它的执行方式是，持续发送启动条件和 LSB 设置为低电平（指示写操作）的 I^2C 从器件地址字节，直到检测到应答位为止。这是因为当写周期正在进行时，EEPROM 不会应答命令。

图 30 显示了地址写操作之前的应答查询操作。

图 30： 应答查询



结论

本应用笔记说明了使用 MSSP 模块和 MCC 来实现串行 EEPROM 器件接口的简便性和有效性。本应用笔记介绍了 SPI 和 I^2C 协议中的基本操作，并给出了逐步的说明。代码是高度可移植的，只需微小的修改就可以在大多数 8 位 PIC 单片机和串行 EEPROM 器件上使用。使用所提供的代码，用户可以构建自己的 SPI 和 I^2C EEPROM 应用程序。如果需要更复杂或更解构的固件设计，用户仍然可以依靠 MCC 生成的 SPI 和 I^2C 函数，这些函数构成了所提供驱动程序文件的骨架。本文档表明：在与 MSSP 模块和 MPLAB® 代码配置器配合时，构建涉及采用 SPI 或 I^2C 的串行 EEPROM 的解决方案不再像过去一样单调乏味、耗费劳力。

附录 A： 将 EEPROM 连接到单片机

图 A-1： SPI 连接图

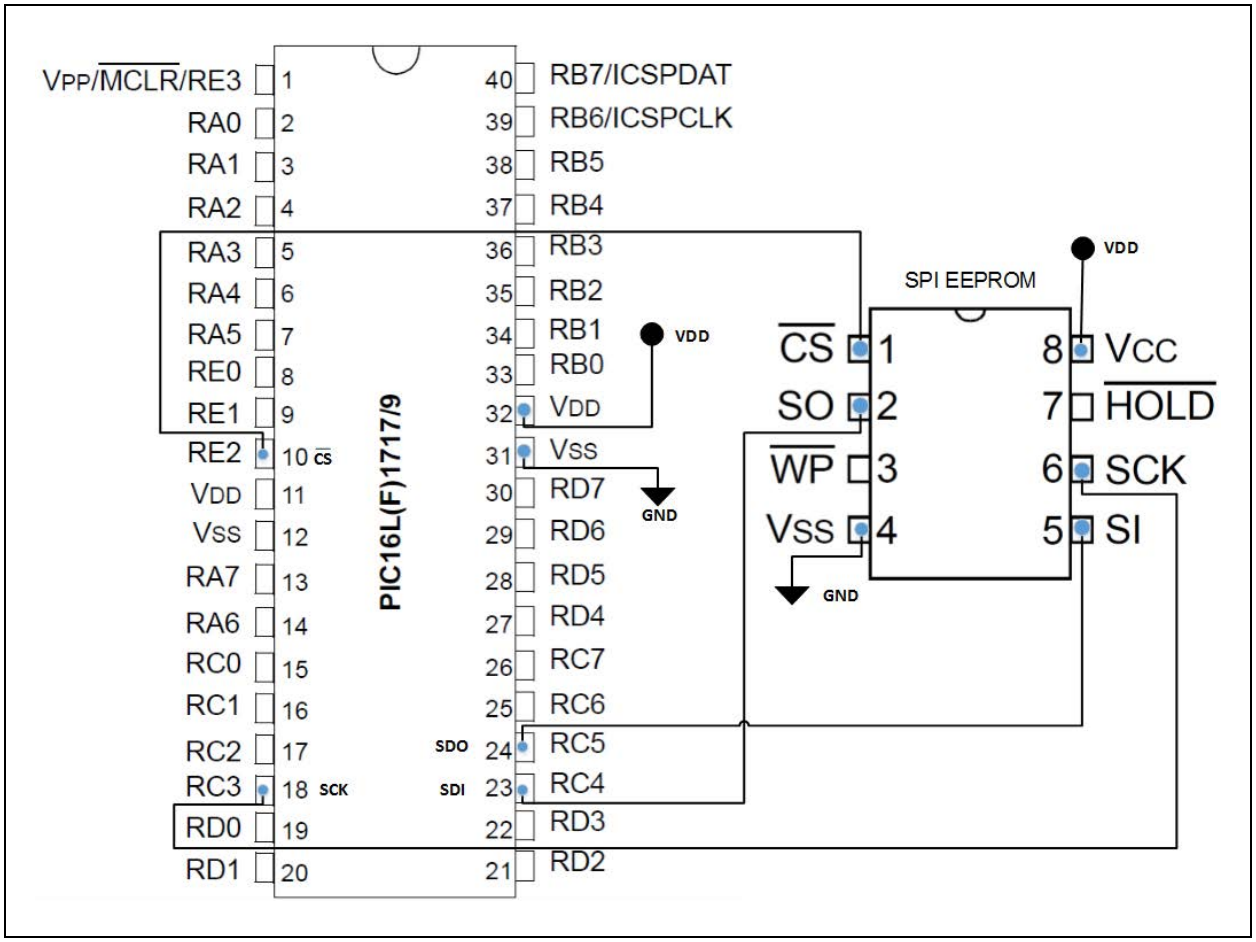
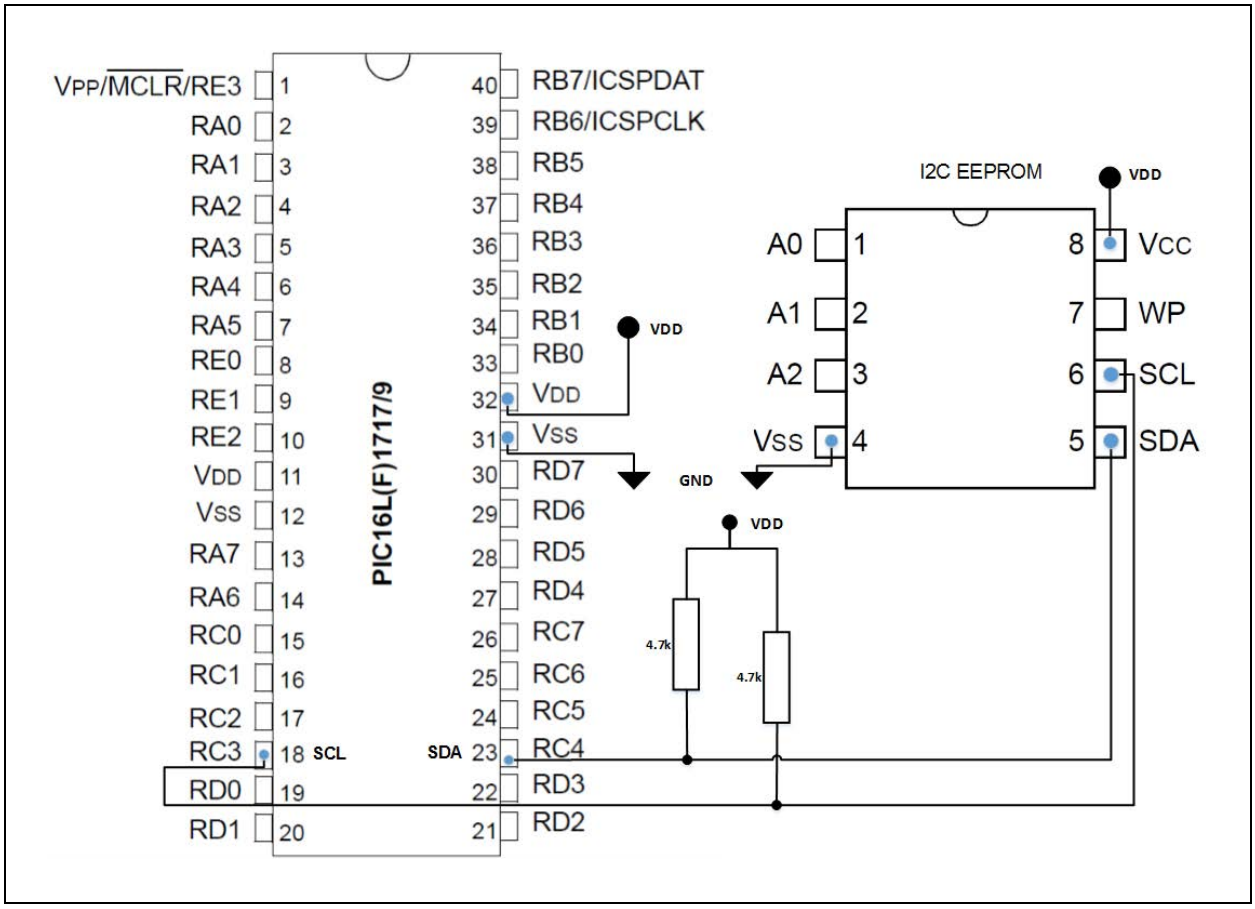


图 A-2： I²C 连接图


VPP/MCLR/RE3 1
RA0 2
RA1 3
RA2 4
RA3 5
RA4 6
RA5 7
RE0 8
RE1 9
RE2 10
VDD 11
VSS 12
RA7 13
RA6 14
RC0 15
RC1 16
RC2 17
RC3 18 SCL
RD0 19
RD1 20
40 RB7/ICSPDAT
39 RB6/ICSPCLK
38 RB5
37 RB4
36 RB3
35 RB2
34 RB1
33 RB0
32 VDD
31 VSS
30 RD7
29 RD6
28 RD5
27 RD4
26 RC7
25 RC6
24 RC5
SDA 23 RC4
22 RD3
21 RD2
PIC16L(F)1717/9
VDD
GND
VDD
4.7k
4.7k
I2C EEPROM
VDD
A0 1
A1 2
A2 3
VSS 4
8 VCC
7 WP
6 SCL
5 SDA

附录 B： 源 CLICK™ 电路板连接图

图 B-1： EEPROM CLICK™ 电路板连接图

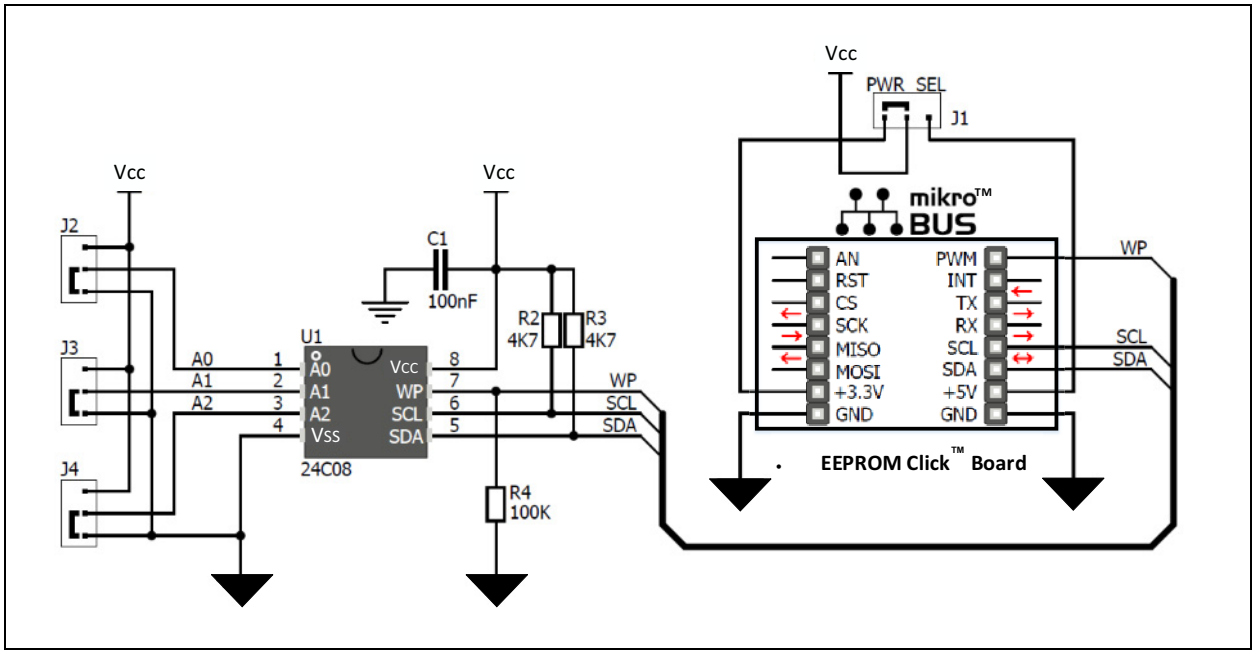
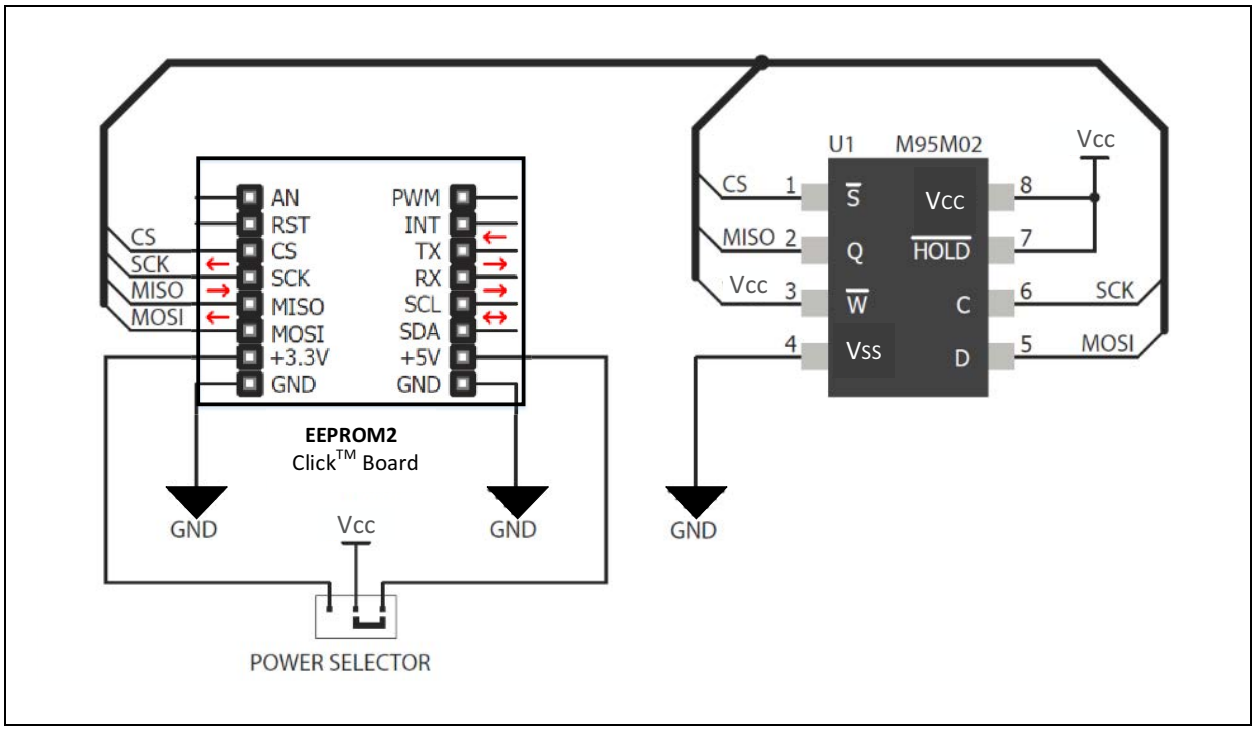


图 B-2： EEPROM2 CLICK™ 电路板连接图



附录 C： 源代码列表

例 C-1： eeprom_spi.c

```
/**
        EEPROM SPI Source File

        Company:
            Microchip Technology Inc.

        File Name:
            eeprom_spi.c

        Summary:
            This is the source file containing the EEPROM SPI functions.

        Description:
            This header file provides implementations for driver APIs for all modules selected in
            the GUI.
            Generation Information :
                Product Revision  :MPLAB® Code Configurator - v2.25.2
                Device            :PIC16F1719
                Driver Version    :2.00
            The generated drivers are tested against the following:
                Compiler          :XC8 v1.34
                MPLAB             :MPLAB X v2.35 or v3.00
    */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
```

例 C-1： eeprom_spi.c（续）

```
    */

    #include "mcc_generated_files/mcc.h"
#include "eeprom_spi.h"

void SPI_ByteWrite (uint8_t *addressBuffer, uint8_t addlen, uint8_t byteData)
{
        uint8_t check;

        //Toggle CS line to start operation
        CS_LAT = 0;

        //Send Write Enable command
        SPI_Exchange8bit(EEPROM_WREN);

        //Toggle CS line to end operation
        CS_LAT = 1;

        //Check if WEL bit is set
        while(check != 2)
            check = SPI_ReadStatusRegister();

        //Toggle CS line to start operation
        CS_LAT = 0;

        //Send Write Command
        SPI_Exchange8bit(EEPROM_WRITE_EN);
        //Send address byte/s
        SPI_Exchange8bitBuffer(addressBuffer,addlen,NULL);
        //Send data byte
        SPI_Exchange8bit(byteData);

        //Toggle CS line to end operation
        CS_LAT = 1;

}

uint8_t SPI_ByteRead (uint8_t *addressBuffer, uint8_t addlen)
{
        uint8_t readByte;

        //Toggle CS line to start operation
        CS_LAT = 0;

        //Send Read Command
        SPI_Exchange8bit(EEPROM_READ_EN);
        //Send address bytes
        SPI_Exchange8bitBuffer(addressBuffer,addlen,NULL);
        //Send Dummy data to clock out data byte from slave
        readByte = SPI_Exchange8bit(DUMMY_DATA);

        //Toggle CS line to end operation
        CS_LAT = 1;

        //return data byte read
    return(readByte);
}
```

例 C-1：　eeprom_spi.c（续）

```
uint8_t SPI_ReadStatusRegister(void)
{
        uint8_t statusByte;

        //Toggle CS line to start operation
        CS_LAT = 0;

        //Send Read Status Register Operation
        SPI_Exchange8bit(EEPROM_RDSR);
        //Send Dummy data to clock out data byte from slave
        statusByte = SPI_Exchange8bit(DUMMY_DATA);

        //Toggle CS line to end operation
        CS_LAT = 1;

    //return data byte read
        return(statusByte);
}

uint8_t SPI_WritePoll(void)
{
        uint8_t pollByte;

        //Read the Status Register
        pollByte = SPI_ReadStatusRegister();

        //Check if WEL and WIP bits are still set
        while(pollByte == 3)
        {
            pollByte = SPI_ReadStatusRegister();
        }

        //return 1 if WEL and WIP bits are cleared and the write cycle is finished
        return(1);
}

void SPI_SequentialWrite(uint8_t *addressBuffer, uint8_t addlen, uint8_t *writeBuffer,
uint8_t buflen)
{
        //Toggle CS line to begin operation
        CS_LAT = 0;

        //Send Write Enable Command
        SPI_Exchange8bit(EEPROM_WREN);

        //Toggle CS line to end operation
        CS_LAT = 1;

        //Toggle CS line to start operation
        CS_LAT = 0;

        //Send Write Command
        SPI_Exchange8bit(EEPROM_WRITE_EN);
        //Send address bytes
        SPI_Exchange8bitBuffer(addressBuffer,addlen,NULL);
        //Send data bytes to be written
        SPI_Exchange8bitBuffer(writeBuffer,buflen,NULL);
```

例 C-1： eeprom_spi.c（续）

```
        //Toggle CS line to end operation
        CS_LAT = 1;

}

uint8_t SPI_SequentialRead(uint8_t *addressBuffer,uint8_t addlen, uint8_t *readBuffer,
uint8_t buflen)
{
        //Toggle CS line to begin operation
        CS_LAT = 0;

        //Send Read Command
        SPI_Exchange8bit(EEPROM_READ_EN);
        //Send Address bytes
        SPI_Exchange8bitBuffer(addressBuffer,addlen,NULL);
        //Send dummy/NULL data to clock out data bytes from slave
    SPI_Exchange8bitBuffer(NULL,buflen,readBuffer);

        //Toggle CS line to end operation
        CS_LAT = 1;
}
```

例 C-2： eeprom_spi.h

```
/**
    EEPROM SPI Header File

    Company:
        Microchip Technology Inc.

    File Name:
        eeprom_spi.h

    Summary:
        This is the header file containing the EEPROM I2C functions.

    Description:
        This header file provides implementations for driver APIs for all modules selected in the
GUI.
        Generation Information :
            Product Revision  :MPLAB® Code Configurator - v2.25.2
            Device            :PIC16F1719
            Driver Version    :2.00
        The generated drivers are tested against the following:
            Compiler          :XC8 v1.34
            MPLAB             :MPLAB X v2.35 or v3.00
    */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).
```

例 C-2： eeprom_spi.h（续）

```
You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
    */
#include "mcc_generated_files/spi.h"

#ifndef EEPROM_SPI_H
#define EEPROM_SPI_H

#ifdef  __cplusplus
extern  "C" {
#endif

#define EEPROM_READ_EN         0x03               // read data from memory
#define EEPROM_WREN            0x06               // set the write enable latch
#define EEPROM_WRITE_EN        0x02               // write data to memory array
#define EEPROM_RDSR            0x05               // read STATUS register

void SPI_ByteWrite (uint8_t *addressBuffer, uint8_t addlen, uint8_t byteData);
uint8_t SPI_ByteRead (uint8_t *addressBuffer,uint8_t addlen);
uint8_t SPI_ReadStatusRegister(void);
uint8_t SPI_WritePoll(void);
void SPI_SequentialWrite(uint8_t *addressBuffer, uint8_t addlen, uint8_t *writeBuffer,uint8_t buflen);
uint8_t SPI_SequentialRead(uint8_t *addressBuffer,uint8_t addlen, uint8_t *readBuffer, uint8_t buflen);

#ifdef  __cplusplus
}
#endif

#endif  /* EEPROM_SPI_H */
```

例 C-3：　调用 SPI 函数的示例主文件

```
/**
    Generated Main Source File

    Company:
        Microchip Technology Inc.

    File Name:
        main.c

    Summary:
        This is the main file generated using MPLAB® Code Configurator

    Description:
        This header file provides implementations for driver APIs for all modules selected in the
GUI.
        Generation Information :
            Product Revision  :MPLAB® Code Configurator - v2.25.2
            Device            :PIC16F1719
            Driver Version    :2.00
        The generated drivers are tested against the following:
            Compiler          :XC8 v1.34
            MPLAB             :MPLAB X v2.35 or v3.00
    */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
    */

#include "mcc_generated_files/mcc.h"
#include "eeprom_spi.h"

/*
                                Main application
    */

void  main(void) {
       // initialize the device
       SYSTEM_Initialize();
```

例 C-3：　调用 SPI 函数的示例主文件 （续）

```
        // When using interrupts, you need to set the Global and Peripheral Interrupt Enable bits
        // Use the following macros to:

        // Enable the Global Interrupts
        //INTERRUPT_GlobalInterruptEnable();

        // Enable the Peripheral Interrupts
        //INTERRUPT_PeripheralInterruptEnable();

        // Disable the Global Interrupts
        //INTERRUPT_GlobalInterruptDisable();

        // Disable the Peripheral Interrupts
        //INTERRUPT_PeripheralInterruptDisable();

        uint8_t    writeBuffer[] = {0x1A, 0x2A, 0x4A, 0x8A} ;
        uint8_t    readBuffer[10];
        uint8_t    addressBuffer[] = {0xAB,0x00,0x10}; // Store the address you want to access
here
        uint8_t    readByte;

        //Writes one byte to the address specified
        SPI_ByteWrite(&addressBuffer,sizeof(addressBuffer),0xA5);

        //Wait for write cycle to complete
        SPI_WritePoll();

        //Reads one byte of data from the address specified
        readByte = SPI_ByteRead(&addressBuffer,sizeof(addressBuffer));

        //Intermission
        __delay_ms(10);

        //Writes the data in writeBuffer beginning from the address specified

SPI_SequentialWrite(&addressBuffer,sizeof(addressBuffer),&writeBuffer,sizeof(writeBuffer));

        //Wait for write cycle to complete
        SPI_WritePoll();

        //Reads specified number of data bytes into the readBuffer array beginning from the
address
          indicated
        SPI_SequentialRead(&addressBuffer,sizeof(addressBuffer),&readBuffer,4);

        //Stop here
        while (1) {
            ;
            // Add your application code
        }
}
/**
    End of File
    */
```

例 C-4： eeprom_i2c.c

```
/**
    EEPROM I2C Source File

    Company:
        Microchip Technology Inc.

    File Name:
        eeprom_i2c.c

    Summary:
        This is the source file containing the EEPROM I2C functions and constants.

    Description:
        This header file provides implementations for driver APIs for all modules selected in the
GUI.
        Generation Information :
            Product Revision  :MPLAB® Code Configurator - v2.25.2
            Device            :PIC16F1719
            Driver Version    :2.00
        The generated drivers are tested against the following:
            Compiler          :XC8 v1.34
            MPLAB             :MPLAB X v2.35 or v3.00
    */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
*/
    #include "mcc_generated_files/mcc.h"
#include "eeprom_i2c.h"

uint8_t timeOut = 0;

int I2C_ByteWrite(uint8_t *dataAddress, uint8_t dataByte, uint8_t addlen)
{
        uint8_t writeBuffer[PAGE_LIMIT+3];
        uint8_t buflen;

        //Copy address bytes to the write buffer so it can be sent first
        for(int i = 0; i < addlen; i++)
```

例 C-4： eeprom_i2c.c（续）

```
{
            writeBuffer[i] = dataAddress[i];
        }

        //Check if this is an address write or a data write.
        if(dataByte != NULL)
        {
            writeBuffer[addlen] = dataByte;
            buflen = addlen+1;
        }
        else
            buflen = addlen;

        //set status to Message Pending to send the data
        I2C_MESSAGE_STATUS status = I2C_MESSAGE_PENDING;

        //This variable is the built in acknowledge polling mechanism.This counts how many
retries
          the system has already done to send the data.
        timeOut = 0;

        //While the message has not failed...
        while(status != I2C_MESSAGE_FAIL)
        {
            // Initiate a write to EEPROM
                I2C_MasterWrite(writeBuffer,buflen,SLAVE_ADDRESS,&status);

            // wait for the message to be sent or status has changed.
                while(status == I2C_MESSAGE_PENDING);
            // if transfer is complete, break the loop
                    if (status == I2C_MESSAGE_COMPLETE)
                        break;
                        // if transfer fails, break the loop
                    if (status == I2C_MESSAGE_FAIL)
                        break;
                //Max retry is set for max Ack polling.If the Acknowledge bit is not set, this
will
                  just loop again until the write command is acknowledged
                    if (timeOut == MAX_RETRY)
                        break;
                    else
                        timeOut++;
        }
                        // if the transfer failed, stop at this point
                        if (status == I2C_MESSAGE_FAIL)
                        return 1;
}

uint8_t I2C_ByteRead(uint8_t *dataAddress,uint8_t dataByte, uint8_t addlen)
{
        int check;

        //Write the address to the slave
        check = I2C_ByteWrite(dataAddress,NULL,addlen);

        //If not successful, return to function
        if(check == 1)
            return;
```

例 C-4：　eeprom_i2c.c（续）

```
        //Get ready to send data
        I2C_MESSAGE_STATUS status = I2C_MESSAGE_PENDING;
        //Set up for ACK polling
        timeOut = 0;

        //While the code has not detected message failure..
        while(status != I2C_MESSAGE_FAIL)
        {
            // Initiate a Read to EEPROM
                I2C_MasterRead(dataByte,1,SLAVE_ADDRESS,&status);

            // wait for the message to be sent or status has changed.
                while(status == I2C_MESSAGE_PENDING);

            // if transfer is complete, break the loop
                if (status == I2C_MESSAGE_COMPLETE)
                    break;

            // if transfer fails, break the loop
                if (status == I2C_MESSAGE_FAIL)
                    break;

            // check for max retry and skip this byte
                if (timeOut == MAX_RETRY)
                    break;
                else
                    timeOut++;
        }
}
int I2C_BufferWrite(uint8_t *dataAddress, uint8_t *dataBuffer, uint8_t addlen, uint8_t buflen)
{
        uint8_t writeBuffer[PAGE_LIMIT+3];
        I2C_MESSAGE_STATUS status = I2C_MESSAGE_PENDING;

        //Set Address as the bytes to be written first
        for(int i = 0; i < addlen; i++)
        {
            writeBuffer[i] = dataAddress[i];
        }

        //Ensure that the page limit is not breached so as to avoid overwriting other data
        if(buflen > PAGE_LIMIT)
            buflen = PAGE_LIMIT;

        //Copy data bytes to write buffer
        for(int j = 0; j < buflen; j++)
        {
            writeBuffer[addlen+j] = dataBuffer[j];
        }
        //Set up for ACK polling
        timeOut = 0;
        while(status != I2C_MESSAGE_FAIL)
        {
            // Initiate a write to EEPROM
                I2C_MasterWrite(writeBuffer,buflen+addlen,SLAVE_ADDRESS,&status);
```

例 C-4： eeprom_i2c.c（续）

```
            // wait for the message to be sent or status has changed.
                while(status == I2C_MESSAGE_PENDING);
            // if transfer is complete, break the loop
                if (status == I2C_MESSAGE_COMPLETE)
                    break;
                    // if transfer fails, break the loop
                if (status == I2C_MESSAGE_FAIL)
                    break;

            //check for max retry and skip this byte
                if (timeOut == MAX_RETRY)
                    break;
                else
                    timeOut++;
        }
                    // if the transfer failed, stop at this point
                    if (status == I2C_MESSAGE_FAIL)
                    return 1;

}
void I2C_BufferRead(uint8_t *dataAddress, uint8_t *dataBuffer, uint8_t addlen,uint8_t buflen)
{
        int check = 0;
        I2C_MESSAGE_STATUS status = I2C_MESSAGE_PENDING;

        //Write Address from where to read
        check = I2C_ByteWrite(dataAddress,NULL,addlen);

        //check if address write is successful
        if(check == 1)
            return;

        //Set up for ACK polling
        timeOut = 0;

        while(status != I2C_MESSAGE_FAIL){
            // Initiate a Read to EEPROM
            I2C_MasterRead(dataBuffer,buflen,SLAVE_ADDRESS,&status);

            // wait for the message to be sent or status has changed.
            while(status == I2C_MESSAGE_PENDING);

            // if transfer is complete, break the loop
            if (status == I2C_MESSAGE_COMPLETE)
                break;

            // if transfer fails, break the loop
            if (status == I2C_MESSAGE_FAIL)
                    break;

            // check for max retry and skip this byte
            if (timeOut == MAX_RETRY)
                break;
            else
                    timeOut++;
            }
}
```

例 C-5： eeprom_i2c.h

```
/**
    EEPROM I2C Header File

    Company:
        Microchip Technology Inc.

    File Name:
        eeprom_i2c.h

    Summary:
        This is the header file containing the EEPROM I2C functions and constants.

    Description:
        This header file provides implementations for driver APIs for all modules selected in the
GUI.
        Generation Information :
            Product Revision  :MPLAB® Code Configurator - v2.25.2
            Device            :PIC16F1719
            Driver Version    :2.00
        The generated drivers are tested against the following:
            Compiler          :XC8 v1.34
            MPLAB             :MPLAB X v2.35 or v3.00
    */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
    */
#ifdef __cplusplus
extern "C" {
#endif

#define MAX_RETRY          100
#define SLAVE_ADDRESS      0x50
#define PAGE_LIMIT         16         // Change as stated on the EEPROM device data sheet

int I2C_ByteWrite(uint8_t *dataAddress, uint8_t dataByte, uint8_t addlen);
uint8_t I2C_ByteRead(uint8_t *dataAddress, uint8_t dataByte,uint8_t addlen);
int I2C_BufferWrite(uint8_t *dataAddress, uint8_t *dataBuffer,  uint8_t addlen, uint8_t buflen);
void I2C_BufferRead(uint8_t *dataAddress, uint8_t *dataBuffer,  uint8_t addlen, uint8_t buflen);

#ifdef __cplusplus
}
#endif

#endif /* EEPROM_I2C_H */
```

例 C-6： 调用 I²C 函数的示例主文件

```
/**
    Generated Main Source File

    Company:9
        Microchip Technology Inc.

    File Name:
        main.c

    Summary:
        This is the main file generated using MPLAB® Code Configurator

    Description:
        This header file provides implementations for driver APIs for all modules selected in the
GUI.
        Generation Information :
            Product Revision  :MPLAB® Code Configurator - v2.25.2
            Device            :PIC16F1719
            Driver Version    :2.00
        The generated drivers are tested against the following:
            Compiler          :XC8 v1.34
            MPLAB             :MPLAB X v2.35 or v3.00
    */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
```

例 C-6： 调用 I²C 函数的示例主文件 （续）

```
*/

#include "mcc_generated_files/mcc.h"
#include "eeprom_i2c.h"

/*
                            Main application
    */
void main(void) {
        // initialize the device
        SYSTEM_Initialize();

        // When using interrupts, you need to set the Global and Peripheral Interrupt Enable bits
        // Use the following macros to:

        // Enable the Global Interrupts
        INTERRUPT_GlobalInterruptEnable();

        // Enable the Peripheral Interrupts
        INTERRUPT_PeripheralInterruptEnable();

        // Disable the Global Interrupts
        //INTERRUPT_GlobalInterruptDisable();

        // Disable the Peripheral Interrupts
        //INTERRUPT_PeripheralInterruptDisable();

        uint8_t        sourceData[] = {0x1A, 0x2A, 0x4A, 0x8A,0x1A, 0x2A, 0x4A, 0x8A,0x1A, 0x2A,
                       0x4A, 0x8A,0x1A, 0x2A, 0x4A, 0x8A};
        uint8_t        addressBuffer[] = {0xAB,0x10} ;       //Put your address here
        uint8_t        readBuffer[16];
        uint8_t        readByte;

        int r = 0;


        //Writes a byte of data to address specified
        r = I2C_ByteWrite(&addressBuffer,0x5B,sizeof(addressBuffer));

        //Reads a byte of data stored at the address specified
        I2C_ByteRead(&addressBuffer,&readByte,sizeof(addressBuffer));

        //Write a specified number of data bytes beginning at the specified address
        r = I2C_BufferWrite(&addressBuffer,&sourceData,sizeof(addressBuffer),4);

        //Reads a specified number of data bytes beginning at the specified address
        I2C_BufferRead(&addressBuffer,&readBuffer,sizeof(addressBuffer),4);

        //stop here
        while (1) {
            ; // Add your application code
        }
}
//}
/**
    End of File
    */
```

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了 《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。Microchip 的 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品严格遵守公司的质量体系流程。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949═

ISBN：978-1-5224-0799-7

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://www.microchip.com/support
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

奥斯汀 Austin, TX
Tel: 1-512-257-3370

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

克里夫兰 Cleveland
Independence, OH
Tel: 1-216-447-0464
Fax: 1-216-447-0643

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Novi, MI
Tel: 1-248-848-4000

休斯敦 Houston, TX
Tel: 1-281-894-5983

印第安纳波利斯 Indianapolis
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608

纽约 New York, NY
Tel: 1-631-435-6000

圣何塞 San Jose, CA
Tel: 1-408-735-9110

加拿大多伦多 Toronto
Tel: 905-695-1980
Fax: 905-695-2078

亚太地区

亚太总部 Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

中国 - 北京
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

中国 - 成都
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 - 重庆
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

中国 - 东莞
Tel: 86-769-8702-9880

中国 - 广州
Tel: 86-20-8755-8029

中国 - 杭州
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

中国 - 南京
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

中国 - 青岛
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 - 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 - 沈阳
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 - 深圳
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

中国 - 武汉
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 - 西安
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

中国 - 厦门
Tel: 86-592-238-8138
Fax: 86-592-238-8130

中国 - 香港特别行政区
Tel: 852-2943-5100
Fax: 852-2401-3431

亚太地区

中国 - 珠海
Tel: 86-756-321-0040
Fax: 86-756-321-0049

台湾地区 - 高雄
Tel: 886-7-213-7828

台湾地区 - 台北
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

台湾地区 - 新竹
Tel: 886-3-5778-366
Fax: 886-3-5770-955

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

印度 India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

印度 India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

印度 India - Pune
Tel: 91-20-3019-1500

日本 Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

日本 Japan - Tokyo
Tel: 81-3-6880-3770
Fax: 81-3-6880-3771

韩国 Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

韩国 Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 或
82-2-558-5934

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

菲律宾 Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

新加坡 Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

泰国 Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark-Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Dusseldorf
Tel: 49-2129-3766400

德国 Germany - Karlsruhe
Tel: 49-721-625370

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

意大利 Italy - Venice
Tel: 39-049-7625286

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

波兰 Poland - Warsaw
Tel: 48-22-3325737

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

瑞典 Sweden - Stockholm
Tel: 46-8-5090-4654

英国 UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

06/23/16