TB3164

8位PIC® 单片机上的直接存储器访问

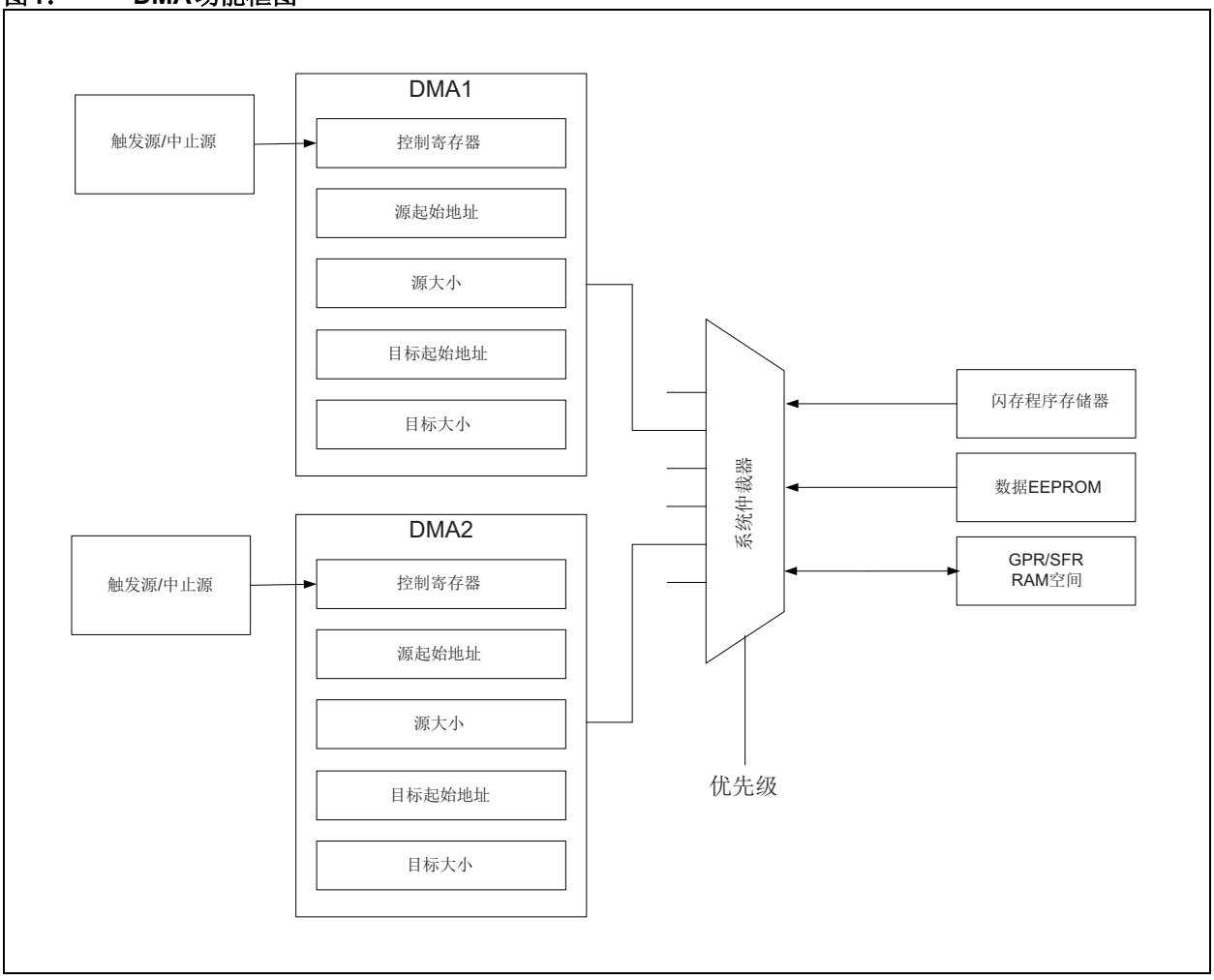
作者： *Mark Pallones*
Microchip Technology Inc.

简介

在面向数据的应用中，无需CPU干预即可在外设或不同存储区之间直接传输数据，从而显著改善延时和吞吐量。

Microchip 8位单片机中的直接存储器访问（Direct Memory Access，DMA）控制器外设可实现这种改善，允许CPU将时间用于处理其他任务，而不是等待寄存器标志或处理与数据移动相关的中断。本技术简介概述了DMA控制器的功能和特性，还提供了代码示例，以说明如何使用MPLAB® X和XC8编译器轻松配置控制器。

图1：　DMA功能框图



系统操作

在图1中，每个DMA控制器都可独立编程，以便在单个或多个地址之间移动数据，并可使用各种触发信号来启动或中止传输。它们还可以配置为一起工作，以便在没有CPU干预的情况下执行更复杂的数据传输。

DMA控制器使用与CPU相同的指令总线和数据总线在存储器之间传输数据。根据系统仲裁器中设置的优先级，DMA控制器可以通过利用未使用的CPU周期或停止CPU的方式来移动数据。

默认情况下，CPU的优先级高于DMA。在这种情况下，DMA会从CPU中挪用未使用的周期来执行读取或写入操作。由于CPU和DMA之间存在这种并发操作，处理数据的带宽增加，DMA操作可以继续而不会导致CPU停止。

DMA数据移动

气泡

如前文所述，当CPU的优先级高于DMA时，DMA会挪用未使用的CPU周期，以维持CPU和DMA之间的并发操作。在这种情况下，未使用的周期称为气泡。根据DMA将要访问的存储区，有多种气泡可供使用，这包括闪存程序存储器（Program Flash Memory，PFM）气泡、特殊功能寄存器（Special Function Register，SFR）/通用寄存器（General Purpose Register，GPR）气泡和DATA EE气泡。PFM气泡是未访问PFM空间的任何指令周期。当DMA将从某些PFM地址读取数据时，DMA将等待至有可用的PFM气泡，并使用这些气泡进行自己的PFM访问。另一方面，SFR/GPR气泡是未访问SFR/GPR存储空间的任何指令周期。当DMA将写入或读取某些SFR/GPR地址时，DMA将等待至有可用的SFR/GPR气泡，并使用这些气泡进行自己的SFR/GPR访问。最后，DATA EE气泡是未访问DATA EE存储空间的任何指令周期。当CPU未读取或写入DATA EE时，DMA可使用可用的DATA EE气泡进行自己的DATA EE访问。表1给出了一些提供气泡的指令示例（OPCODE）。

表1：　　提供气泡的指令示例

PFM气泡的有效操作码	SFR/GPR气泡的有效操作码	DATA EE气泡的有效操作码
BRA	所有PFM气泡	未读取或写入DATA EE的所有指令
BC（如果进位标志置1）	MOVLW	
BN（如果跳转）	BANKSEL	
BNC（如果进位标志未置1）	CALL及其fnop	
BNN	CLRWDT	
BNOV	GOTO及其fnop	
BNZ	PUSH和POP	
BOV	RETFIE，后跟fnop	
BZ	RETLW，后跟fnop	
RETFIE/RETURN/RETLW	ANDLW、IORLW、LFSR、MOVLB、MULLW、SUBLW和XORLW	
RCALL	TBLRD（在第一个周期期间提供1个气泡）	

事务

DMA控制器事务是指从源地址到目标地址的最小数据移动。其过程分为两步：从源地址存储器中读取值并将值存储在DMAxBUF寄存器中，然后将DMAxBUF寄存器的内容写入目标地址存储器。DMAxBUF寄存器是一个8位缓冲区，用于保存从源地址移动到目标地址的数据。传输的所有数据均通过DMAxBUF。

报文

DMA报文由一个或多个事务组成，由单个硬件触发信号或软件启动信号发起。DMA报文的大小由源报文大小（DMAxSSZ）和目标报文大小（DMAxDSZ）寄存器中编程的值确定。如果这些寄存器的值不相等，则最大值与最小值的比值决定DMA过程中有多少条报文。例如，当DMAxSSZ为2且DMAxDSZ为6时，每条报文将由两个事务组成，整个DMA过程将由三条报文组成。表2给出了基于各种DMA操作的报文大小配置值的示例。

表2：　　报文大小配置示例

DMA操作	DMAxSSZ	DMAxDSZ	备注
读取UART接收寄存器中的值并将值存储在存储器缓冲区中	1	N	N等于要存储在目标缓冲区中的字节数
读取ADC结果寄存器并将值存储在存储器缓冲区中	2	2*N	N等于要存储在存储器缓冲区中的结果数
加载存储器表中的PWM占空比值	2*N	2	N等于从存储器表中加载的占空比值的数量
通过UART发送存储器表中的数据字节	N	1	N等于存储器表中待发送的字节数

DMA寻址

存储器访问

DMA控制器将数据从一个存储器移动到另一个存储器。DMA控制器具有对PFM、数据EEPROM和SFR/GPR空间的读访问权限，以及对SFR/GPR空间的写访问权限。表3给出了支持的DMA数据事务。

表3：　　支持的DMA数据事务

读取源	写入目标
PFM	GPR
PFM	SFR
数据EE	GPR
数据EE	SFR
GPR	GPR
SFR	GPR
GPR	SFR
SFR	SFR

可分别使用源起始地址（DMAxSSA）和目标起始地址（DMAxDSA）寄存器来设置读取源和写入目标的起始地址。要选择DMAxSSA寄存器寻址哪个存储区，可设置DMA控制寄存器1（DMAxCON1）的源存储器选择位（SMR）的值。由于DMA控制器只能写入SFR/GPR存储空间，因此无需通过任何位来设置目标存储区。

寻址模式

当DMAxCON0寄存器的DMA模块使能位（EN）使能时，源指针寄存器（DMAxSPTR）和目标指针寄存器（DMAxPTR）分别指向写入DMAxSSA和DMAxDSA寄存器的地址。

但是，当DMA报文正在传输时，这些地址指针寄存器将根据DMAxCON1寄存器中的源地址模式选择位（SMODE）和目标地址模式选择位（DMODE）的值在每个事务后进行修改。实际上，将读取或写入的源地址和目标地址也会进行修改。表4和表5给出了如何根据SMODE和DMODE设置更新DMAxSPTR和DMAxDPTR。

表4：　　操作期间的DMAxSPTR值

SMODE值	DMAxSPTR值
00	DMAxSPTR
01	DMAxSPTR+1
1x	DMAxSPTR-1

表5：　　操作期间的DMAxDPTR值

DMODE值	DMAxDPTR值
00	DMAxDPTR
01	DMAxDPTR+1
1x	DMAxDPTR-1

默认情况下，SMODE和DMODE设置为00，其中地址指针在所有传输期间指向固定的源地址和目标地址，这意味着通道的每次数据传输都将向/从同一存储单元复制数据。这在从一个外设向另一个外设传输数据等情况下十分有用。对于要将数据存储在数组中的应用程序，可将DMODE配置为01或1x以在每次访问字节后增大或减小目标地址指针。类似地，要从数组中获取数据时，可将SMODE配置为01或1x以在每次访问字节后增大或减小源地址指针。最后，为了将数据从一个数组复制到另一个数组，可在每次访问字节后增大或减小源地址指针和目标地址指针。

启动DMA事务

硬件触发

当检测到DMA传输中断请求时，可启动DMA传输。传输中断请求可通过几种可用的硬件触发源触发。该触发源可使用启动中断请求源寄存器（DMAxSIRQ）进行选择。可用的触发源可能因器件而异，具体取决于器件中存在的外设。触发源可以是边沿中断或电平中断。边沿中断只能在符合给定的模块中断要求时产生一次。只要导致中断的条件为真，就会产生一个电平中断。有关可用的边沿触发中断和电平触发中断的列表，请参见具体器件的数据手册。

触发源与DMA源地址和目标地址无关。DMA可使用一个外设中断作为触发源，对其他外设执行DMA操作。例如，配置为响应Timer1中断的DMA可用于将数据移入或移出通用异步收发器（Universal Asynchronous Receiver Transmitter，UART）。但是，当需要高吞吐量时，使用外设自身的中断来执行DMA操作更合适。例如，配置为将数据从UART移出的DMA可将其自身的发送中断标志（UxTXIF）用作触发源。

要启动触发源的采样，应使能DMA控制寄存器0（DMAxCON0）的传输开始中断请求位（SIRQEN）。只要SIRQEN使能且触发信号保持有效，DMA即会连续传输相同的报文。

软件启动

启动DMA传输的另一种方法是使用用户软件控制。这可以通过使能DMAxCON1寄存器中的DMA事务位（DGO）来实现。DGO位也可用于指示外部触发信号是否已被接收以及传输是否正在进行。

要了解数据是否已写入目标地址，可以监视DMAxCON0的传输进行状态位（XIP）。如果XIP位置1，则表示数据正在等待写入目标地址。如果该位清零，则表示数据已写入目标地址或未发生源读取操作。

重载

启动DMA传输后，DMAxSSZ和DMAxDSZ寄存器中初始化的对应值将被加载到源计数寄存器（DMAxSCNT）和目标计数寄存器（DMAxDCNT）中。在每个事务后，DMAxSCNT和DMAxDCNT寄存器将递减，进而指示当前DMA报文中剩余的字节数。DMAxSCNT和DMAxDCNT寄存器始终分别从DMAxSSZ和DMAxDSZ寄存器重载相应值。当DMAxSCNT从1开始递减时，它将通过DMAxSSZ加载，当DMAxDCNT从1开始递减时，它将通过DMAxDSZ加载。当DMAxSCNT或DMAxDCNT寄存器重载时，DGO位将清零以指示报文完成。

由于DMAxSCNT和DMAxDCNT始终在每次完成报文传输时进行重载，因此电平传输触发中断将允许进一步的DMA报文传输。要停止报文传输，需满足相应条件。这些条件将在下几节中讨论。

停止DMA事务

正常完成

如前文所述，将SIRQEN位置1会使能中断触发源的采样，通过该中断触发源可启动DMA传输。因此，清零该位将停止采样并禁止传输过程。

但是，如果在传输过程中SIRQEN位清零，传输过程将不会停止，DMA将等待至报文已传输。之后，不会再出现任何报文传输，因为DMA将忽略任何传输中断触发信号。

可通过软件清零SIRQEN位，也可通过将DMAxCON0寄存器的源计数器重载停止位（SSTP）和目标计数器重载停止位（DSTP）来清零该位。当SSTP位置1且DMAxSCNT寄存器重载时，SIRQEN位将清零。类似地，当DSTP位置1且DMAxDCNT寄存器重载时，SIRQEN位将清零。SSTP和DSTP是独立的，这意味着如果重载任一计数器寄存器或重载两个计数器寄存器，报文将停止。

软停止

当DGO位通过软件强制清零时，消息将停止，DMA将保持当前配置。当DGO位在读取源地址到写入目标地址的过程中清零时，DMAxBUF中的数据将不会到达其目标地址。这也称为软停止，因为一旦DGO位再次置1，操作即可再次恢复。

中止

也可使用任何中断触发源来停止DMA报文传输。这些中止触发源可通过中止中断请求源寄存器（DMAxAIRQ）进行选择。DMAxCON0寄存器的中止传输中断请求使能位（AIRQEN）置1且接收到中止中断触发信号后，DMA将通过清零DGO位来执行软停止。DMA还将清零SIRQEN位以避免溢出。AIRQEN位也会清零，以防止额外的中止信号触发虚假中止。由于DMA执行软停止，需要时，可再次将DGO位置1，以便在软停止发生后从操作停止的位置恢复操作。DMA状态信息在中止情况下不会改变。

硬停止

清零EN位也会停止报文传输，强制DMA恢复其默认配置。这也称为硬停止，因为DMA无法从操作停止的位置恢复操作。

中断

正常完成

如上一节所述，DMAxSCNT或DMAxDCNT寄存器重载时可确定DMA消息完成。发生该事件时，相应PIRx寄存器的源计数中断标志位（DMAxSCNTIF）或目标计数中断标志位（DMAxDCNTIF）也会置1。DMA报文传输完成后，这些中断标志位可用于确定何时中断CPU。

中止

当DMA因来自中止源之一的触发信号而停止数据传输时，相应PIRx寄存器的中止中断标志位（DMAxAIF）置1。每当DMA传输过程停止时，该标志即可用于中断CPU。

溢出

当DMA在当前事务完成之前收到执行新事务的请求时，相应PIRx寄存器的DMA溢出中断标志位（DMAxORIF）置1。

DMAxORIF标志置1不会导致当前DMA传输终止，这明确表明工作的通道可能无法满足正在处理的外设模块的要求。在数据传输过程中发生错误时，可使用DMAxORIF中断标志中断CPU。

用于数据传输的DMA设置

下面是要进行数据传输时DMA的一般设置步骤。

1. 将事务的适当源地址和目标地址编程到DMAxSSA和DMAxDSA寄存器中。
2. 使用SMR<1:0>位选择通过DMAxSSA寄存器寻址的源存储区。
3. 对SMODE和DMODE位进行编程，以选择寻址模式。
4. 使用要传输的字节数编程源大小DMAxSSZ和目标大小DMAxDSZ寄存器。建议采取大小寄存器互为彼此倍数的适当操作。
5. 如果用户希望在报文完成后禁止数据传输，则需要将DMAxCON0寄存器中的SSTP和DSTP位置1。
6. 如果使用硬件触发信号进行数据传输，请设置用于启动和中止DMA传输的硬件触发中断源（DMAxSIRQ和DMAxAIRQ），并将相应的中断请求使能位（SIRQEN和AIRQEN）置1。
7. 选择DMA控制器的优先级，并锁定优先级以授予存储器访问权限。
8. 使能DMA控制器（DMAxCON1bits.EN = 1）。
9. 如果使用软件控制进行数据传输，请将DGO位置1。

代码示例

有关配置DMA控制器的代码示例，请参见**附录A：“使用PIC18F25K42的DMA代码示例”**。例A-1给出了所有后续代码示例的配置位设置。例A-2会将DMA1设置为将数据字节从GPR空间中的一个数组传输到GPR空间中的另一个数组，并使用软件启动传输。例A-3会将DMA1设置为将数据字节从DATA EE传输到GPR空间中的数组，并使用Timer1作为传输触发源。例A-4会将DMA1设置为将字符字节直接从PFM传输到UART发送寄存器，并使用自身的U1TX标志作为传输触发源。对于例A-2、例A-3和例A-4，在传输完所有数据字节后，不会传输其他数据。

只要Timer1提供的中止触发信号不置为有效，例A-5便会将DMA1设置为将字符直接从PFM连续传输到UART。例A-6会将DMA1设置为从PFM传输到UART发送寄存器。当DMA1DCNTIF置1时，程序将进入低优先级ISR以翻转RC4。当DMA1ORIF置1时，程序将进入高优先级ISR以翻转RC5并硬停止DMA传输。例A-7会将DMA1设置为将数据字节从PFM传输到GPR空间中的数组。一旦数组中存在数据，DMA2即会将数据字节直接传输到UART发送寄存器。

结论

DMA可用于在无CPU干预的情况下在存储器和外设之间传输数据。本技术简介涵盖了DMA的不同特性，这些特性有助于存储器之间灵活高效地传输数据。本文档还介绍了配置方法和代码示例，以说明如何设置DMA控制器。

附录A： 使用PIC18F25K42的DMA代码示例

软件许可协议

Microchip Technology Incorporated（“公司”）随附提供的软件旨在提供给您（该公司的客户）使用，仅限于且只能在该公司制造的产品上使用。

该软件为公司和/或其供应商所有，并受适用的版权法保护。版权所有。任何违反前述限制的使用将使其用户遭受适用法律的刑事制裁，并承担违背此许可的条款和条件的民事责任。

该软件“按现状”提供。不提供保证，无论是明示的、暗示的还是法定的保证。这些保证包括（但不限于）对出于某一特定目的应用此软件的适销性和适用性默示的保证。在任何情况下，公司都将不会对任何原因造成的特别的、偶然的或间接的损害负责。

例A-1： CONFIG位设置

```
// CONFIG1L
#pragma config FEXTOSC = OFF                // External Oscillator Selection (Oscillator not enabled)
#pragma config RSTOSC = HFINTOSC_1MHZ       // Reset Oscillator Selection (HFINTOSC with HFFRQ = 4 MHz
                                            and CDIV = 4:1)

// CONFIG1H
#pragma config CLKOUTEN = OFF               // Clock out Enable bit (CLKOUT function is disabled)
#pragma config PR1WAY = OFF                 //  (PRLOCK bit can be set and cleared repeatedly)
#pragma config CSWEN = OFF                  // Clock Switch Enable bit (The NOSC and NDIV bits cannot
                                            be changed by user software)
#pragma config FCMEN = OFF                  // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock
                                            Monitor disabled)

// CONFIG2L
#pragma config MCLRE = EXTMCLR              // MCLR Enable bit (If LVP = 0, MCLR pin is MCLR;
                                            If LVP = 1, RE3 pin function is MCLR )
#pragma config PWRTS = PWRT_OFF             // Power-up timer selection bits (PWRT is disabled)
#pragma config MVECEN = OFF                 // Multi-vector enable bit (Interrupt contoller does not
                                            use vector table to prioritze interrupts)
#pragma config IVT1WAY = OFF                // IVTLOCK bit One-way set enable bit (IVTLOCK bit can be
                                            cleared and set repeatedly)
#pragma config LPBOREN = OFF                // Low Power BOR Enable bit (ULPBOR disabled)
#pragma config BOREN = SBORDIS              // Brown-out Reset Enable bits (Brown-out Reset enabled,
                                            SBOREN bit is ignored)

// CONFIG2H
#pragma config BORV = VBOR_2P45             // (Brown-out Reset Voltage (VBOR) set to 2.45V)
#pragma config ZCD = OFF                    // ZCD Disable bit (ZCD disabled. ZCD can be enabled by
                                            setting the ZCDSEN bit of ZCDCON)
#pragma config PPS1WAY = OFF                // PPSLOCK bit One-Way Set Enable bit (PPSLOCK bit can be
                                            set and cleared repeatedly (subject to the unlock sequence)
#pragma config STVREN = OFF                 // Stack Full/Underflow Reset Enable bit
                                            (Stack full/underflow will not cause Reset)
#pragma config DEBUG = OFF                  // Debugger Enable bit (Background debugger disabled)
#pragma config XINST = OFF                  // Extended Instruction Set Enable bit (Extended
                                            Instruction Set and Indexed Addressing Mode disabled)

// CONFIG3L
#pragma config WDTCPS = WDTCPS_31           // WDT Period selection bits (Divider ratio 1:65536;
                                            software control of WDTPS)
#pragma config WDTE = OFF                   // WDT operating mode (WDT Disabled; SWDTEN is ignored)

// CONFIG3H
#pragma config WDTCWS = WDTCWS_7            // WDT Window Select bits (window always open (100%);
                                            software control; keyed access not required)
#pragma config WDTCCS = SC                  // WDT input clock selector (Software Control)

// CONFIG4L
#pragma config BBSIZE = BBSIZE_512          // Boot Block Size selection bits (Boot Block size is 512
                                            words)
#pragma config BBEN = OFF                   // Boot Block enable bit (Boot block disabled)
#pragma config SAFEN = OFF                  // Storage Area Flash enable bit (SAF disabled)
#pragma config WRTAPP = OFF                 // Application Block write protection bit
                                            (Application Block not write protected)

// CONFIG4H
#pragma config WRTB = OFF                   // Configuration Register Write Protection bit
                                            (Configuration registers (300000-30000Bh) not
                                            write-protected)
#pragma config WRTC = OFF                   // Boot Block Write Protection bit
                                            (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF                   // Data EEPROM Write Protection bit (Data EEPROM not
                                            write-protected)
#pragma config WRTSAF = OFF                 // SAF Write protection bit (SAF not Write Protected)
#pragma config LVP = OFF                    // Low Voltage Programming Enable bit (HV on MCLR/VPP must
                                            be used for programming)

// CONFIG5L
#pragma config CP = OFF                     // PFM and Data EEPROM Code Protection bit (PFM and Data
                                            EEPROM code protection disabled)
```

例A-2： SFR至SFR传输，DSTP = 1，软件启动

```
#include <xc.h>
#include <stdint.h>

uint8_t Array1[10];
uint8_t Array2[10];
uint8_t i;

void main(void)

{

    //Initialize the Arrays
    for(i=0; i<10 ; i++)
    {
        Array1[i] = i+1;
        Array2[i] = 0;
    }

    //Initialize DMA1
    DMA1SSA = (uint16_t) Array1;          //Set source start address
    DMA1DSA = (uint16_t) Array2;          //Set destination start address
    DMA1CON1bits.SMR = 0b00;              //Choose GPR space as source memory
    DMA1CON1bits.SMODE = 0b01;            //Increment source address on each transaction
    DMA1CON1bits.DMODE = 0b01;            //Increment destination address on each transaction
    DMA1SSZ = 0x0A;                       //Set 10 bytes for source size
    DMA1DSZ = 0x0A;                       //Set 10 bytes for destination size
    DMA1CON1bits.SSTP = 0b0;              //Clear source reload stop bit
    DMA1CON1bits.DSTP = 0b1;              //Set destination reload stop bit
    DMA1CON0bits.DMA1SIRQEN = 0b0;        //Source Trigger is not allowed to start DMA transfer


    // Use default priority level
    // Lock priority to grant memory access
    asm ("BANKSEL PRLOCK");
    asm ("MOVLW 0x55");
    asm ("MOVWF PRLOCK");
    asm ("MOVLW 0xAA");
    asm ("MOVWF PRLOCK");
    asm ("BSF PRLOCK, 0");

    DMA1CON0bits.DMA1EN = 1;              //Enable DMA
    DMA1CON0bits.DMA1DGO = 1;             //Start Transfer

    while (0 == PIR2bits.DMA1DCNTIF){}    //Wait after message transfer completed

    while(1);

}
```

例A-3： 数据EE至GPR传输，DSTP = 1，TIMER1触发源

```
#include <xc.h>
#include <stdint.h>


void Timer1_Init(void);

uint8_t Array1[10];
uint8_t i;


void main(void)

{

    Timer1_Init();

    //Clear Array1
    for(i=0; i<10 ; i++)
    {
        Array1[i] = 0;

    }

    DMA1SSA = 0x00;                     //Set source start address
    DMA1DSA = (uint16_t) Array1;        //Set destination start address
    DMA1CON1bits.SMR = 0b10;            //Choose DATAEE as source memory
    DMA1CON1bits.SMODE = 0b01;          //Increment source address on each transaction
    DMA1CON1bits.DMODE = 0b01;          //Increment destination address on each transaction
    DMA1SSZ = 0x0A;                     //Set 10 bytes for source size
    DMA1DSZ = 0x0A;                     //Set 10 bytes for destination size
    DMA1CON1bits.SSTP = 0b0;            //Clear source reload stop bit
    DMA1CON1bits.DSTP = 0b1;            //Set destination reload stop bit
    DMA1SIRQ = 32;                      //Choose Timer1 as Transfer Trigger Source
    DMA1CON0bits.DMA1SIRQEN = 0b1;      //Source Trigger is allowed to start DMA transfer


    // Use default priority level
    // Lock priority to grant memory access
    asm ("BANKSEL PRLOCK");
    asm ("MOVLW 0x55");
    asm ("MOVWF PRLOCK");
    asm ("MOVLW 0xAA");
    asm ("MOVWF PRLOCK");
    asm ("BSF PRLOCK, 0");

    DMA1CON0bits.DMA1EN = 1;            //Enable DMA


    T1CONbits.TMR1ON =1;                //Start Timer1


    while (0 == PIR2bits.DMA1DCNTIF){}  //Wait after message transfer completed

    while(1);
}

void Timer1_Init(void)
{

    T1CON = 0x06;                       //1:1 PreScale,NoSync,16bit mode
    T1CLK = 0x03;                       //HFINTOSC clock source

    TMR1L = 0xFF;                       //Initialize Timer Low Register
    TMR1H = 0xFF;                       //Initialize Timer High Register

    PIR4bits.TMR1IF =0;                 //Clear Timer 1 interrupt flag
}
```

例A-4： PFM至SFR传输，SSTP = 1，U1TX触发源

```
#include <xc.h>
#include <stdint.h>

void UART_Init(void);

    //Initialize 9 charters in program memory space starting at 0x000300 address const
    uint8_t table[] __at(0x000300) = {'M','i','c','r','o','c','h','i','p'};

void main(void)

{
    UART_Init();                          //Initialize UART

    DMA1SSA = 0x000300;                   //Set source start address
    DMA1DSA = (uint16_t) &U1TXB;          //Set destination start address to U1TXB
    DMA1CON1bits.SMR = 0b01;              //Choose PFM space as source memory
    DMA1CON1bits.SMODE = 0b01;            //Increment source address each transaction
    DMA1CON1bits.DMODE = 0b00;            //Remain unchanged the destination address transaction
    DMA1SSZ = 0x09;                       //Set 9 bytes for source size
    DMA1DSZ = 0x01;                       //Set 1 byte  for destination size
    DMA1CON1bits.SSTP = 0b1;              //SIRQEN is clear after sending 9 bytes
    DMA1CON1bits.DSTP = 0b0;              //No destination reload stop bit
    DMA1SIRQ = 28;                        //Choose U1TX as Transfer Trigger Source
    DMA1CON0bits.DMA1SIRQEN = 1;          //Source Trigger is allowed to start DMA transfer

    // Use default priority level
    // Lock priority to grant memory access
    asm ("BANKSEL PRLOCK");
    asm ("MOVLW 0x55");
    asm ("MOVWF PRLOCK");
    asm ("MOVLW 0xAA");
    asm ("MOVWF PRLOCK");
    asm ("BSF PRLOCK, 0");

    DMA1CON0bits.DMA1EN = 1;              //Enable DMA


    while (0 == PIR2bits.DMA1DCNTIF){}    //Wait after message transfer completed

    while(1);
}

void UART_Init(void)

{
    U1BRGH = 0x00;
    U1BRGL = 0x05;                        // Set to 9600 baud using 1Mhz HFINTOSC
    U1CON0bits.BRGS = 0;                  // BRG  normal speed mode
    U1CON0bits.MODE = 0b0000;             // Asynchronous 8-bit mode
    U1CON1bits.ON = 1;                    // Enable serial port
    U1CON0bits.TXEN = 1;                  // Enable transmit

    // TX and RX pin configuration

    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00;         // Unlock PPS
    U1RXPPS = 0b10111;                    // RC7->UART:RX
    RC6PPS = 0b010011;                    // RC6->UART:TX
    ANSELCbits.ANSELC7 = 0;               // Clear analog select bit of RX pin
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x01;         // Lock PPS
}
```

例A-5： PFM至SFR传输，SSTP = 0，TIMER1中止源

```
#include <xc.h>
#include <stdint.h>

void UART_Init(void);
void Timer1_Init(void);

    //Initialize 9 charters in program memory space starting at 0x000300 address const
    uint8_t table[] __at(0x000300) = {'M','i','c','r','o','c','h','i','p'};

void main(void)

{
    UART_Init();                            //Initialize UART
    Timer1_Init();                          //Initialize Timer1

    DMA1SSA = 0x000300;                     //Set source start address
    DMA1DSA = (uint16_t) &U1TXB;            //Set destination start address to U1TXB
    DMA1CON1bits.SMR = 0b01;                //Choose PFM space as source memory
    DMA1CON1bits.SMODE = 0b01;              //Increment source address each transaction
    DMA1CON1bits.DMODE = 0b00;              //Remain unchanged the destination address transaction
    DMA1SSZ = 0x09;                         //Set 9 bytes for source size
    DMA1DSZ = 0x01;                         //Set 1 byte  for destination size
    DMA1CON1bits.SSTP = 0b0;                //No source reload stop bit
    DMA1CON1bits.DSTP = 0b0;                //No destination reload stop bit
    DMA1SIRQ = 28;                          //Choose U1TX as Transfer Trigger Source
    DMA1AIRQ = 32;                          //choose Timer1 as Abort Trigger Source
    DMA1CON0bits.DMA1SIRQEN = 1;            //Source Trigger is allowed to start DMA transfer
    DMA1CON0bits.AIRQEN = 1;                //Abort Trigger is allowed to abort DMA transfer

    // Use default priority level
    // Lock priority to grant memory access
    asm ("BANKSEL PRLOCK");
    asm ("MOVLW 0x55");
    asm ("MOVWF PRLOCK");
    asm ("MOVLW 0xAA");
    asm ("MOVWF PRLOCK");
    asm ("BSF PRLOCK, 0");

    DMA1CON0bits.DMA1EN = 1;                //Enable DMA


    while (0 == PIR2bits.DMA1DCNTIF){}      //Wait after message transfer completed

    T1CONbits.TMR1ON =1;                    //Start Timer1

    while(1);
}

void UART_Init(void)
{
    U1BRGH = 0x00;
    U1BRGL = 0x05;                          // Set to 9600 baud using 1Mhz HFINTOSC
    U1CON0bits.BRGS = 0;                    // BRG normal speed mode
    U1CON0bits.MODE = 0b0000;               // Asynchronous 8-bit mode
    U1CON1bits.ON = 1;                      // Enable serial port
    U1CON0bits.TXEN = 1;                    // Enable transmit

    // TX and RX pin configuration
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00;           // Unlock PPS
    U1RXPPS = 0b10111;                      // RC7->UART:RX
    RC6PPS = 0b010011;                      // RC6->UART:TX
    ANSELCbits.ANSELC7 = 0;                 // Clear analog select bit of RX pin
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x01;           // Lock PPS
}


void Timer1_Init(void)
{
    T1CON = 0x36;                           //1:8 PreScale,NoSync,16bit mode
    T1CLK = 0x03;                           //HFINTOSC clock source

    TMR1L = 0xFF;                           //Initialize Timer Low Register
    TMR1H = 0XFF;                           //Initialize Timer High Register

    PIR4bits.TMR1IF =0;                     //Clear Timer 1 interrupt flag
}
```

例A-6： PFM至SFR传输，SSTP = 0，目标计数和溢出中断允许

```
#include <xc.h>
#include <stdint.h>


void IO_Init(void);
void UART_Init(void);
void Interrupt_Init(void);
void Timer0_Init(void);


    //Initialize 9 charters in program memory space starting at 0x000300 address const
    uint8_t table[] __at(0x000300) = {'M','i','c','r','o','c','h','i','p'};

void __interrupt(high_priority) highPriorityInterrupt08(void)
{

    PIR2bits.DMA1ORIF = 0;                  //Clear Over run Interrupt Flag

    LATCbits.LATC5 ^= 1;                    // Set RC5 pin Abort Interrupt indicator

    DMA1CON0bits.DMA1EN = 0;                // Hard stop DMA1 when over run interrupt is asserted


}
void __interrupt(low_priority) lowPriorityInterrupt18 (void)
{

    PIR2bits.DMA1DCNTIF =0;                 //Clear Destination Count Interrupt flag

    LATCbits.LATC4 ^= 1;                    // Set RC4 pin Destination Count Interrupt indicator

}

void main(void)

{
    Interrupt_Init();                       //Initialize Interrupt
    IO_Init();                              //Initialize IO
    UART_Init();                            //Initialize UART
    Timer0_Init();                          //Initialize Timer0


    DMA1SSA = 0x000300;                     //Set source start address
    DMA1DSA = (uint16_t) &U1TXB;            //Set destination start address to U1TXB
    DMA1CON1bits.SMR = 0b01;                //Choose PFM space as source memory
    DMA1CON1bits.SMODE = 0b01;              //Increment source address each transaction
    DMA1CON1bits.DMODE = 0b00;              //Remain unchanged the destination address transaction
    DMA1SSZ = 0x09;                         //Set 9 bytes for source size
    DMA1DSZ = 0x01;                         //Set 1 byte  for destination size
    DMA1CON1bits.SSTP = 0b0;                //No source reload stop bit
    DMA1CON1bits.DSTP = 0b0;                //No destination reload stop bit
    DMA1SIRQ = 28;                          //Change this to 31 to simulate overrun interrupt
    DMA1CON0bits.DMA1SIRQEN = 1;            //Source Trigger is allowed to start DMA transfer
    DMA1CON0bits.AIRQEN = 0;                // No Abort Trigger is allowed to abort DMA transfer


    // Use default priority level
    // Lock priority to grant memory access
    asm ("BANKSEL PRLOCK");
    asm ("MOVLW 0x55");
    asm ("MOVWF PRLOCK");
    asm ("MOVLW 0xAA");
    asm ("MOVWF PRLOCK");
    asm ("BSF PRLOCK, 0");


    DMA1CON0bits.DMA1EN = 1;                //Enable DMA
    T0CON0bits.EN =1;                       //Enable Timer0

    while (0 == PIR2bits.DMA1DCNTIF){}      //Wait after message transfer completed

    while(1);
}
```

例A-6： PFM至SFR传输，SSTP = 0，目标计数和溢出中断允许（续）

```
void UART_Init(void)
{
    U1BRGH = 0x00;
    U1BRGL = 0x05;                    // Set to 9600 baud using 1Mhz HFINTOSC
    U1CON0bits.BRGS = 0;              // BRG  normal speed mode
    U1CON0bits.MODE = 0b0000;         // Asynchronous 8-bit mode
    U1CON1bits.ON = 1;                // Enable serial port
    U1CON0bits.TXEN = 1;              // Enable transmit

    // TX and RX pin configuration
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00;     // Unlock PPS
    U1RXPPS = 0b10111;                // RC7->UART:RX
    RC6PPS = 0b010011;                // RC6->UART:TX
    ANSELCbits.ANSELC7 = 0;           // Clear analog select bit of RX pin
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x01;     // Lock PPS
}

void IO_Init(void)
{
    // Set RC4 and RC5 as digital
    // output pins and initialize as low
    ANSELCbits.ANSELC5 = 0;
    ANSELCbits.ANSELC4 =0;
    TRISCbits.TRISC5 = 0;
    TRISCbits.TRISC4 = 0;
    LATCbits.LC5 = 0;
    LATCbits.LC4 = 0;
}

void Interrupt_Init(void)
{
    INTCON0bits.GIEH = 1;             //Enable High Priority Interrupts
    INTCON0bits.GIEL = 1;             //Enable Low Priority Interrupts
    INTCON0bits.IPEN = 1;             //Enable Priority

    IPR2bits.DMA1DCNTIP = 0;          //Set Destination Count Interrupt to low priority
    IPR2bits.DMA1ORIP = 1;            //Set Overrun Interrupt to high priority

    PIR2bits.DMA1DCNTIF = 0;          //Clear Destination  Count Interrupt flag
    PIR2bits.DMA1ORIF =0;             //Clear Over run Interrupt Flag

    PIE2bits.DMA1DCNTIE = 1;          //Set Destination Count Interrupt Enable bit
    PIE2bits.DMA1ORIE =1;             //Set Destination Over Run Interrupt Enable bit
    }

void Timer0_Init(void)

{
    T0CON0 = 0x00;                    //Module Disable, Operating in 8 bit,1:1 postscale
    T0CON1 = 0x50;                    //1:32 pre-scale,synchronized to FOSC/4

    TMR0H = 0x05;                     // Initialize Timer0 High Register
    TMR0L = 0x00;                     // Initialize Timer0 Low Register

    PIR3bits.TMR0IF = 0;              //Clear Timer0 Interrupt Flag

}
```

例A-7： 使用DMA1的PFM至GPR传输以及使用DMA2的GPR至SFR传输

```
#include <xc.h>
#include <stdint.h>

void UART_Init(void);
void Timer1_Init(void);

    //Initialize 9 charters in program memory space starting at 0x000300 address
    const uint8_t table[] __at(0x000300) = {'M','i','c','r','o','c','h','i','p'};
    // Initialize Variables
    uint8_t i;
    uint8_t Array1[9];

void main(void)
{
    // Clear Array

    for (i=0;i<9;i++)
    {
        Array1[i]=0;
    }

    UART_Init();                        //Initialize UART
    Timer1_Init();                      //Initialize Timer1

    DMA1SSA = 0x000300;                 //Set source start address
    DMA1DSA = (uint16_t) Array1;        //Set destination start address to U1TXB
    DMA1CON1bits.SMR = 0b01;            //Choose PFM space as source memory
    DMA1CON1bits.SMODE = 0b01;          //Increment source address each transaction
    DMA1CON1bits.DMODE = 0b01;          //Increment destination address each transaction
    DMA1SSZ = 0x09;                     //Set 9 bytes for source size
    DMA1DSZ = 0x09;                     //Set 9 bytes  for destination size
    DMA1CON1bits.SSTP = 0b0;            //No source reload stop bit
    DMA1CON1bits.DSTP = 0b1;            //Set destination reload stop bit
    DMA1SIRQ = 32;                      //Choose Timer1 as Transfer Trigger Source
    DMA1CON0bits.DMA1SIRQEN = 1;        //Source Trigger is allowed to start DMA transfer
    DMA1CON0bits.AIRQEN = 0;            //Abort Trigger is not allowed to abort DMA transfer

    DMA2SSA = (uint16_t) Array1;        //Set source start address
    DMA2DSA = (uint16_t) &U1TXB;        //Set destination start address to U1TXB
    DMA2CON1bits.SMR = 0b00;            //Choose GPR space as source memory
    DMA2CON1bits.SMODE = 0b01;          //Increment source address each transaction
    DMA2CON1bits.DMODE = 0b00;          //Remain unchanged the destination address transaction
    DMA2SSZ = 0x09;                     //Set 9 bytes for source size
    DMA2DSZ = 0x01;                     //Set 1 byte  for destination size
    DMA2CON1bits.SSTP = 0b1;            //Set source reload stop bit
    DMA2CON1bits.DSTP = 0b0;            //No destination reload stop bit
    DMA2SIRQ = 28;                      //Choose U1TX as Transfer Trigger Source
    DMA2AIRQ = 32;                      //choose Timer1 as Abort Trigger Source
    DMA2CON0bits.SIRQEN = 1;            //Source Trigger is allowed to start DMA transfer
    DMA2CON0bits.AIRQEN = 0;            //Abort Trigger is not allowed to abort DMA transfer

    // Use default priority level
    // Lock priority to grant memory access
    asm ("BANKSEL PRLOCK");
    asm ("MOVLW 0x55");
    asm ("MOVWF PRLOCK");
    asm ("MOVLW 0xAA");
    asm ("MOVWF PRLOCK");
    asm ("BSF PRLOCK, 0");

    DMA1CON0bits.DMA1EN = 1;            //Enable DMA1
    T1CONbits.TMR1ON =1;                //Start Timer1

    while (0 == PIR2bits.DMA1DCNTIF){}  //Wait after message available in Array1

    DMA2CON0bits.DMA2EN = 1;            //Enable DMA2

    while (0 == PIR5bits.DMA2DCNTIF){}  //Wait after message transfer completed

    while(1);
}
```

例A-7：　使用DMA1的PFM至GPR传输以及使用DMA2的GPR至SFR传输（续）

```
void UART_Init(void)

{
    U1BRGH = 0x00;
    U1BRGL = 0x05;                    // Set to 9600 baud using 1Mhz HFINTOSC
    U1CON0bits.BRGS = 0;              // BRG  normal speed mode
    U1CON0bits.MODE = 0b0000;         // Asynchronous 8-bit mode
    U1CON1bits.ON = 1;                // Enable serial port
    U1CON0bits.TXEN = 1;              // Enable transmit

    // TX and RX pin configuration
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x00;     // Unlock PPS
    U1RXPPS = 0b10111;                // RC7->UART:RX
    RC6PPS = 0b010011;                // RC6->UART:TX
    ANSELCbits.ANSELC7 = 0;           // Clear analog select bit of RX pin
    PPSLOCK = 0x55;
    PPSLOCK = 0xAA;
    PPSLOCKbits.PPSLOCKED = 0x01;     // Lock PPS
}

void Timer1_Init(void)

{
    T1CON = 0x36;                     //1:8 PreScale,NoSync,16bit mode
    T1CLK = 0x03;                     //HFINTOSC clock source

    TMR1L = 0xFF;                     //Initialize Timer Low Register
    TMR1H = 0XFF;                     //Initialize Timer High Register

    PIR4bits.TMR1IF =0;               //Clear Timer 1 interrupt flag
}
```

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：978-1-5224-2027-9

11/07/16

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

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
Tel: 1-317-536-2380

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608
Tel: 1-951-273-7800

罗利 Raleigh, NC
Tel: 1-919-844-7510

纽约 New York, NY
Tel: 1-631-435-6000

圣何塞 San Jose, CA
Tel: 1-408-735-9110
Tel: 1-408-436-4270

加拿大多伦多 Toronto
Tel: 1-905-695-1980
Fax: 1-905-695-2078

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: 86-21-3326-8000
Fax: 86-21-3326-8021

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: 886-7-213-7830

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

芬兰 Finland - Espoo
Tel: 358-9-4520-820

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

法国 France - Saint Cloud
Tel: 33-1-30-60-70-00

德国 Germany - Garching
Tel: 49-8931-9700

德国 Germany - Haan
Tel: 49-2129-3766400

德国 Germany - Heilbronn
Tel: 49-7131-67-3636

德国 Germany - Karlsruhe
Tel: 49-721-625370

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

德国 Germany - Rosenheim
Tel: 49-8031-354-560

以色列 Israel - Ra'anana
Tel: 972-9-744-7705

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

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

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

挪威 Norway - Trondheim
Tel: 47-7289-7561

波兰 Poland - Warsaw
Tel: 48-22-3325737

罗马尼亚 Romania - Bucharest
Tel: 40-21-407-87-50

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

瑞典 Sweden - Gothenberg
Tel: 46-31-704-60-40

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

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