PIC24FJ256GA705 系列闪存编程规范

1.0 器件概述

本文档定义了 PIC24FJ256GA705 系列 16 位单片机的编程规范。本编程规范仅供为以下器件开发编程支持的人员使用：

- PIC24FJ256GA705
- PIC24FJ128GA705
- PIC24FJ64GA705
- PIC24FJ256GA704
- PIC24FJ128GA704
- PIC24FJ64GA704
- PIC24FJ256GA702
- PIC24FJ128GA702
- PIC24FJ64GA702

本文档包括以下主题：

2.0 编程概述

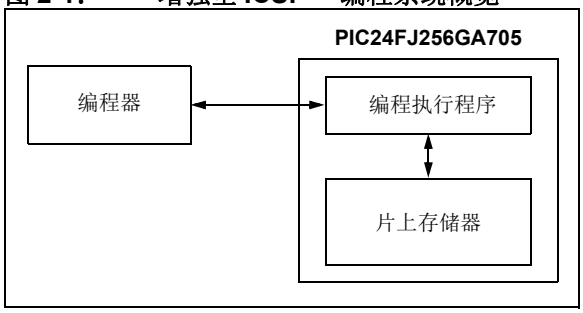
本编程规范讨论了两种对器件编程的方法：

- 在线串行编程 （In-Circuit Serial Programming™，ICSP™）
- 增强型在线串行编程

ICSP 编程方法是最直接的器件编程方法，但也是两种方法中较慢的一种。其固有的低级编程功能可对器件进行擦除、编程和校验。

增强型 ICSP 协议采用一种较快的方法，该方法利用了如图2-1中所示的编程执行程序（Programming Executive，PE）。 PE 通过一个小的命令集提供擦除、编程和校验芯片所必需的所有功能。该命令集使得编程器对 PIC24FJ256GA705 系列器件的编程无需处理低级编程协议。

图 2-1： 增强型 ICSP™ 编程系统概览



本编程规范分为两个主要部分，分别对上述两种编程方法进行了介绍。**第 3.0 节 “器件编程 ——ICSP”** 介绍了 ICSP 方法。**第 4.0 节“器件编程 —— 增强型 ICSP”** 介绍了增强型 ICSP 方法。

2.1 必需的连接

这些器件需要进行特定的连接，才可进行编程。这些连接包括电源、 VCAP、 MCLR 和一个编程引脚对（PGEDx/PGECx）。表 2-1 对这些连接进行了说明（欲知引脚说明和电源连接要求，请参见具体器件数据手册）。

2.2 电源要求

所有PIC24FJ256GA705系列器件都使用标称值为1.8V的电压为其内核数字逻辑供电。为了简化系统设计，PIC24FJ256GA705 系列中的所有器件均包含一个片上稳压器，可使器件内核逻辑通过 VDD 工作。

稳压器通过其他 VDD 引脚为内核供电。必须在 VCAP 引脚上连接一个低 ESR 的电容 （如陶瓷电容或钽电容）（见表 2-1 和图 2-2）。这有助于维持稳压器的稳定性。**第 9.0 节 “交流 / 直流特性和时序要求”** 中列出了内核电压和电容的规范。

图 2-2： 片上稳压器连接

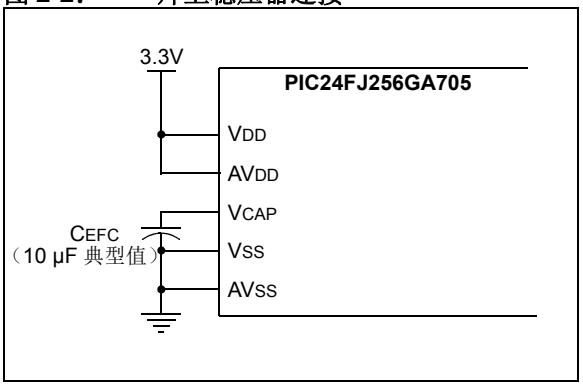


表 2-1： 编程期间使用的引脚

引脚名称	引脚类型	引脚说明
MCLR	I	编程使能
VDD 和 AVDD(1)	P	电源 (1)
VSS 和 AVSS(1)	P	地 (1)
VCAP	P	片上稳压器滤波电容
PGECx	I	编程引脚对：串行时钟
PGEDx	I/O	编程引脚对：串行数据

图注： I = 输入 O = 输出 P = 电源

注 1： 所有电源和地引脚都必须连接，包括 AVDD 和 AVSS。

2.3 引脚图

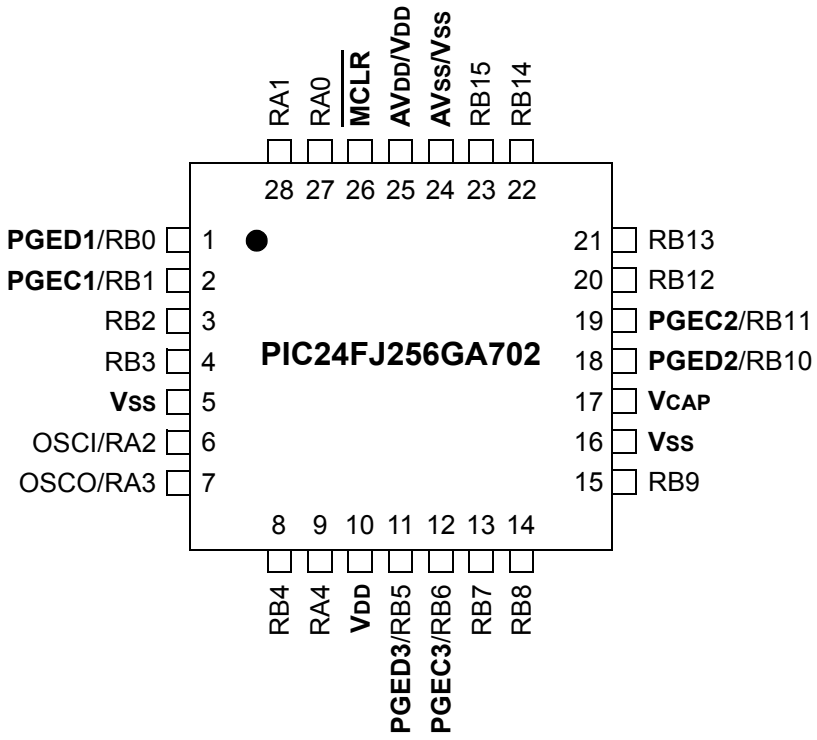
下面各图给出了 PIC24FJ256GA705 系列的引脚图。表 2-1 列出了进行编程所需的引脚。各图中以粗体文字来指示这些引脚。关于完整的引脚说明，请参见相应器件的数据手册。

2.3.1 PGECx 和 PGEDx 引脚对

PIC24FJ256GA705 系列中的所有器件均具有 3 个独立的编程引脚对，标记为 PGEC1/PGED1、PGEC2/PGED2 和 PGEC3/PGED3。 ICSP 或增强型 ICSP 可以使用这些引脚对中的任意一对来进行器件编程。与电源和地引脚不同，对器件进行编程并不需要连接全部 3 个引脚对。但是，编程方法必须使用同一引脚对的两个引脚。

引脚图 （28 引脚 QFN）

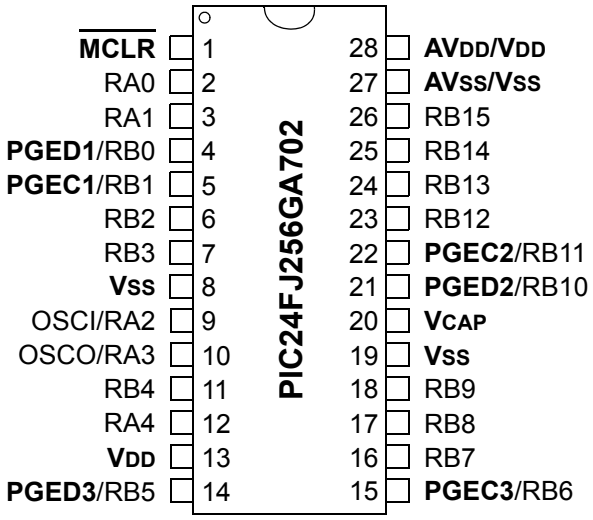
28 引脚 QFN



图注： **粗体字**表示引脚用于器件编程；随附的表中列出了与编程 / 仿真引脚相关功能的完整列表。

引脚图 （28 引脚 PDIP）

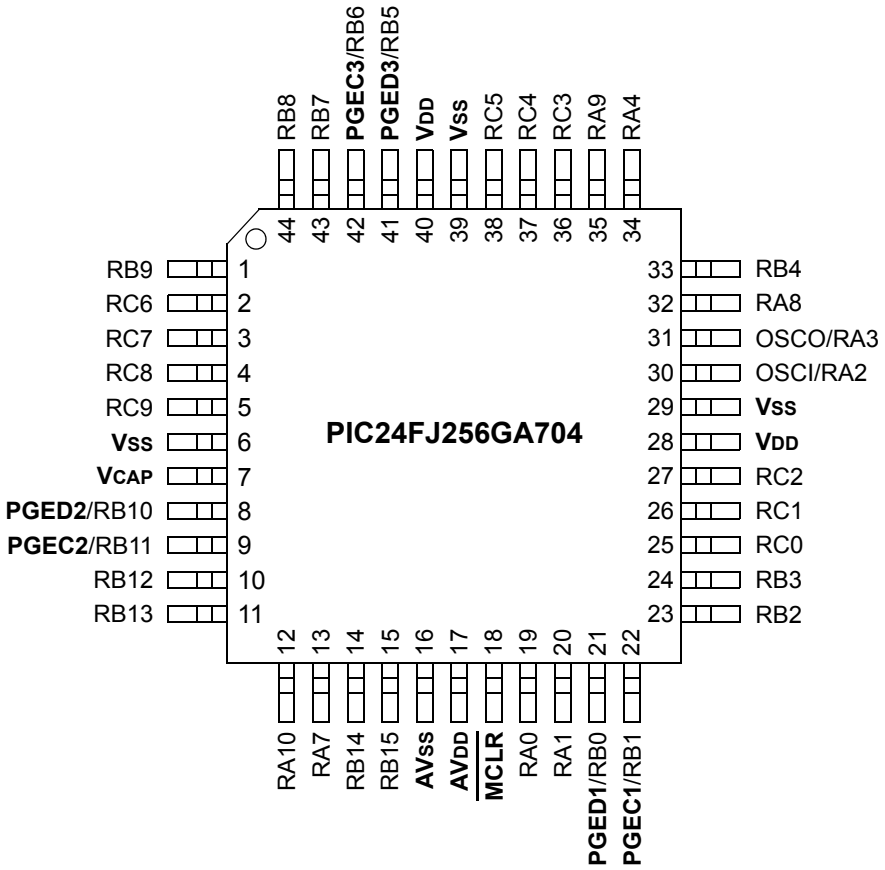
28 引脚 PDIP



图注： **粗体字**表示引脚用于器件编程；随附的表中列出了与编程 / 仿真引脚相关功能的完整列表。

引脚图 （**44** 引脚 **TQFP/QFN**）

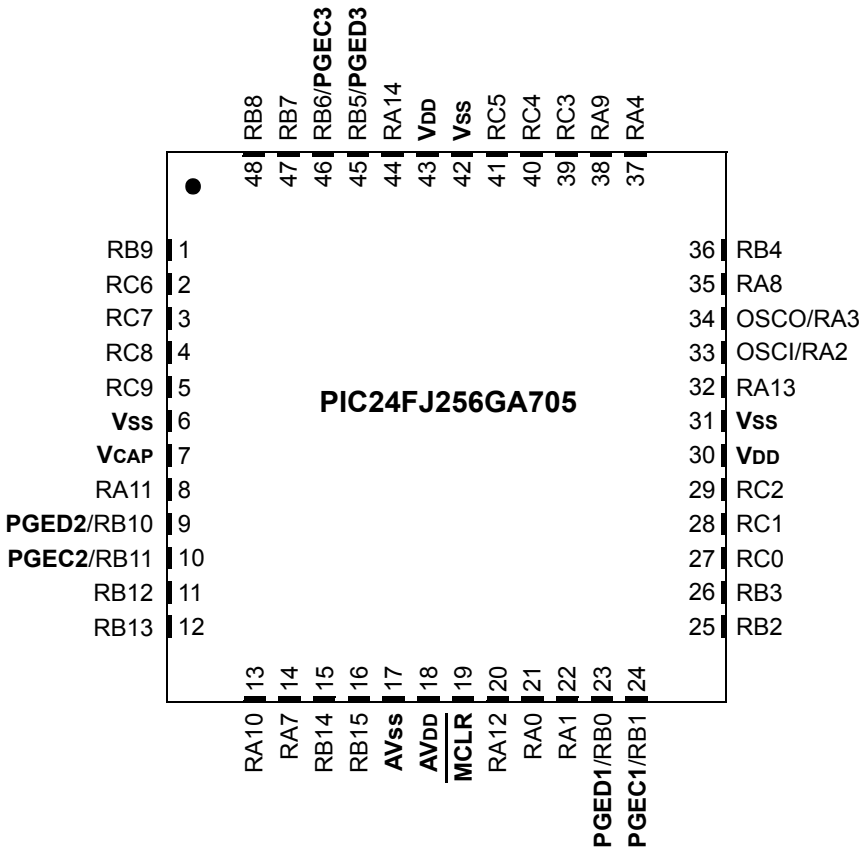
44 引脚 TQFP/QFN



图注： **粗体字**表示引脚用于器件编程；随附的表中列出了与编程 / 仿真引脚相关功能的完整列表。

引脚图 （48 引脚 TQFP/QFN）

48 引脚 TQFP/QFN



图注： **粗体字**表示引脚用于器件编程；随附的表中列出了与编程 / 仿真引脚相关功能的完整列表。

2.4 程序存储器写 / 擦除要求

为了使器件正常工作，必须严格遵守对这些器件的闪存程序存储器进行写 / 擦除的要求。规则如下：在没有擦除存储器中将要写入的任何给定字所在的存储页之前，不能对该字进行写入。因此，遵守该规则最简单的方法就是在一个写周期内写入一个编程块中的所有数据。本文档中指定的编程方法就符合这一要求。

2.5 存储器映射

程序存储空间由可字寻址的块构成。虽然它被视为 24 位宽，但将程序存储器的每个地址单元视作一个低位字和一个高位字的组合更加合理，其中高位字的高字节部分未实现。

低位字的地址始终为偶数，而高位字的地址为奇数。

程序存储器地址始终在低位字处按字对齐，并且在代码执行过程中地址将递增或递减 2。这种寻址模式也与数据存储空间寻址兼容，且为访问程序存储空间中的数据提供了可能。

地址从 0x80 0100 至 0x80 0FFE 的存储单元保留用于存储执行程序代码。该区域存储编程执行程序（PE）和调试执行程序，其中编程执行程序用于器件编程。存储器中的该区域不能用于存储用户代码。更多信息，请参见**第 6.0 节 “编程执行程序”**。

地址从 0x80 1700 至 0x80 17FE 的存储单元保留用于存储客户数据。此区域可用于存储产品信息，如序列号、系统生产日期、生产批号和其他特定于应用的信息。**第 2.6.3 节 “用户 OTP 存储区”**对它进行了介绍。

地址为 0xFF 0000 和 0xFF 0002 的两个存储单元保留用于存储器件 ID 字寄存器。编程器可使用这些位来标识要编程的器件类型。这将在**第 7.0 节 “器件 ID”**中进行介绍。即使应用了代码保护，也可正常读出器件 ID 寄存器。

图 2-3 给出了此规范中所述的器件的通用存储器映射。关于确切的存储器地址，请参见具体器件数据手册中的**“存储器构成”**章节。

表 2-2 给出了每款器件中代码存储区的大小以及擦除块的大小和数量。

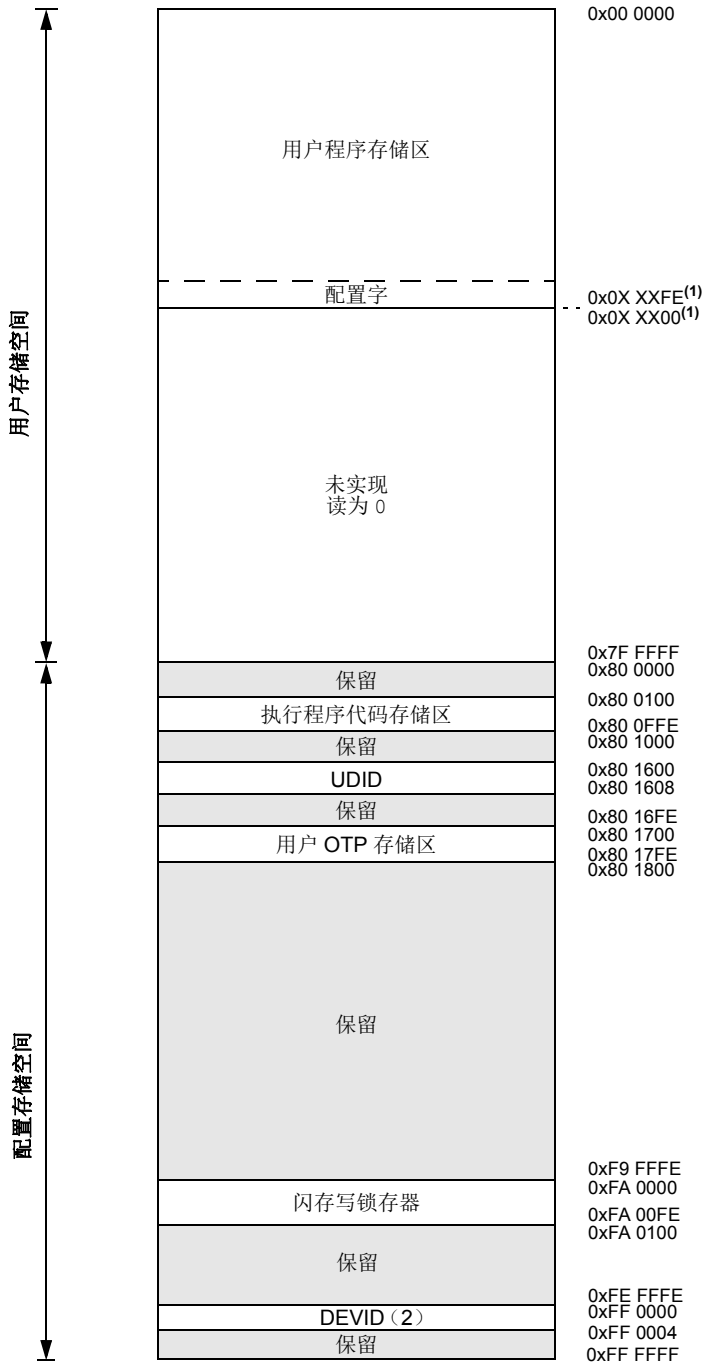
表 2-2： 程序存储器大小和边界 (2)

器件	程序存储器 上边界 （指令字）	写块 (1)	擦除块 (1)
PIC24FJ256GA70X	0x02 AFFE （86K）	1376	172
PIC24FJ128GA70X	0x01 5FFE （44K）	704	88
PIC24FJ64GA70X	0x00 AFFE （22K）	352	44

注 1： 1 个写块 （行） = 128 个指令字； 1 个擦除块 （页） = 1024 个指令字。

2： 为了使页大小保持为整数，存储器大小并非正好为彼此的一半。

图 2-3：　　程序存储器映射



图注：　存储区未按比例绘制。

注 1：　确切边界地址由所实现程序存储器的大小决定。详情请参见表 2-2。

2.6 配置位

2.6.1 概述

通过将这些位置 1 或清零来选择各种器件配置。具有两种不同类型的配置位：系统操作位和代码保护位。系统操作位决定系统级组件 （如振荡器和看门狗定时器）的上电设置。代码保护位防止程序存储器被读写。

表 2-3 列出了每款器件的配置寄存器地址范围。表 2-4 列出了在 PIC24FJ256GA705 系列器件中的所有配置位及其配置寄存器单元。关于具体器件的完整配置寄存器说明，请参见具体器件数据手册中的**“特殊功能”**章节。

2.6.2 代码保护配置位

这些器件实现了由 FSEC 寄存器定义的中等安全功能。引导段（Boot Segment，BS）是权限较高的段，通用段（General Segment，GS）是权限较低的段。整个用户代码存储区可以拆分为 BS 或 GS。这两个段的大小由 BSLIM<12:0> 位决定。段在用户空间内的相对位置不会改变，BS（如果存在）占用紧接在中断向量表（Interrupt Vector Table，IVT）后的存储区，GS 占用紧接在 BS 后的空间，但如果使能了备用 IVT，则 GS 占用紧接在备用 IVT 后的空间。

配置段 （Configuration Segment， CS）是用户闪存地址空间中一个很小的段 （小于一页，通常仅为一行）。它包含复位序列期间由 NVM 控制器装入的所有用户配置数据。

表 2-3： 配置字地址

配置寄存器	PIC24FJ256GA70X	PIC24FJ128GA70X	PIC24FJ64GA70X
FSEC	0x02 AF00	0x01 5F00	0x00 AF00
FBSLIM	0x02 AF10	0x01 5F10	0x00 AF10
FSIGN	0x02 AF14	0x01 5F14	0x00 AF14
FOSCSEL	0x02 AF18	0x01 5F18	0x00 AF18
FOSC	0x02 AF1C	0x01 5F1C	0x00 AF1C
FWDT	0x02 AF20	0x01 5F20	0x00 AF20
FPOR	0x02 AF24	0x01 5F24	0x00 AF24
FICD	0x02 AF28	0x01 5F28	0x00 AF28
FDEVOPT1	0x02 AF2C	0x01 5F2C	0x00 AF2C

表 2-4：　配置寄存器映射

寄存器名称	Bit 23-16	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FSEC	—	AIVTDIS	—	—	—	CSS<2:0>			CWRP	GSS<1:0>		GWRP	—	BSS<2:0>			BWRP
FBSLIM	—	—	—	—	BSLIM<12:0>												
FSIGN	—	SIGN	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
FOSCSEL	—	—	—	—	—	—	—	PLL96DIV<1:0>		IESO	PLLMODE<3:0>				FNOSC<2:0>		
FOSC	—	—	—	—	—	—	—	—	—	FCKSM<1:0>		IOL1WAY	PLLSS	SOSCSEL	OSCIOFNC	POSCMD<1:0>	
FWDT	—	—	WDTCLKS<1:0>		—	WDTCMX	—	WDTWIN<1:0>		WINDIS	FWDTEN<1:0>		WDTPRE	WDTPS<3:0>			
FPOR	—	—	—	—	—	—	—	—	—	—	—	—	—	DNVPEN	RETVRDIS	BOREN<1:0>	
FICD	—	NOBTSWP	—	—	—	—	—	—	—	BKBUG	—	JTAGEN	—	—	—	ICS<1:0>	
FDEVOPT1	—	—	—	—	—	—	—	—	—	—	—	—	DOPT<4:1>				—

图注：　**— =** 未实现，读为 1

2.6.3 用户 OTP 存储区

PIC24FJ256GA705 系列器件提供了 384 字节的可一次性编程（One-Time-Programmable，OTP）存储区，位于地址 0x80 1700 至 0x80 17FE 处。该存储区可用于持久存储的特定于应用的信息，在对器件重新编程时不会擦除这些信息。其中包括许多类型的信息，例如：

- 应用程序校验和
- 代码版本信息
- 产品信息
- 序列号
- 系统生产日期
- 生产批号

可以在任意模式下对用户 OTP 存储区进行编程，包括用户 RTSP 模式，但无法擦除该存储区。整片擦除不会清除其数据。

注： OTP 存储区位于 ECC 控制的闪存区域，但擦除操作对其无效。因此，必须严格地一次性写入 OTP。多次写入 OTP 区域可能会导致双位 ECC 错误，这将在读取 OTP 区域时强制处理器复位。如果 OTP 存储区已写入多次，则无法去除 ECC 错误。

3.0 器件编程 ——ICSP

ICSP 模式是允许您对器件存储器进行读写操作的一种特殊编程协议。 ICSP 模式是最直接的器件编程方法，通过使用 PGECx 和 PGEDx 引脚向器件串行发送控制代码和指令可实现该功能。ICSP 模式还具有以下功能：读取执行程序存储区的内容以确定编程执行程序（PE）是否存在，以及在使用增强型 ICSP 模式时将 PE 写入执行程序存储区。

在 ICSP 模式下，系统时钟总是来自 PGECx 引脚，而不管器件的振荡器配置位的设置如何。所有指令会先串行移入内部缓冲区，然后装入指令寄存器（Instruction Register，IR）并执行。不会从内部存储器执行程序取操作。一次送入 24 位指令。PGEDx 用来移入数据，PGECx 既用作串行移位时钟又用作 CPU 执行时钟。

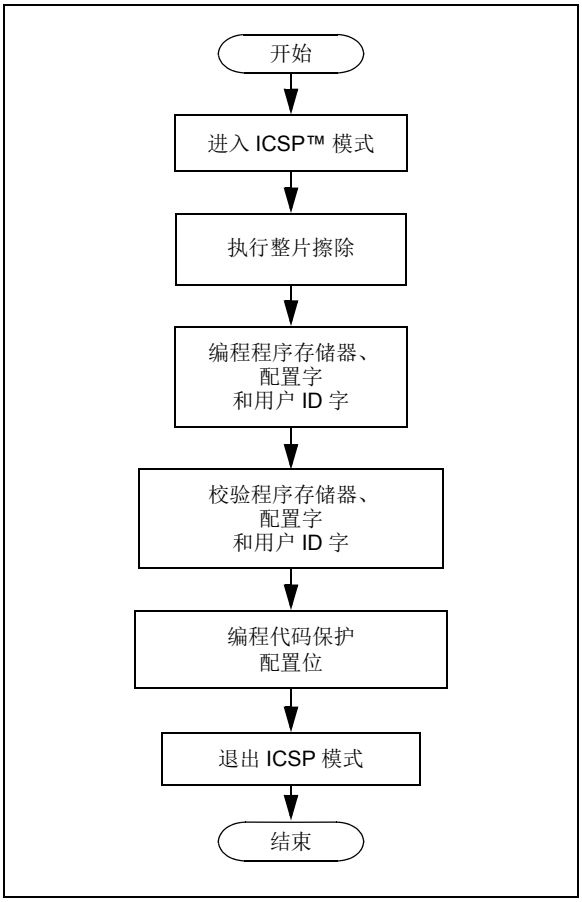
注 1： 在 ICSP 操作期间，PGECx 的工作频率不能超过 5 MHz。

2： ICSP 模式比增强型 ICSP 模式编程速度慢。

3.1 编程过程概述

图 3-1 高度概括了编程过程。进入 ICSP 模式后，首先执行的操作是对程序存储器进行整片擦除。然后，先对代码存储区编程，再对器件配置位编程。随后对代码存储区（包括配置位）进行校验以确保编程成功。最后，如果需要，可对代码保护配置位编程。

图 3-1： ICSP™ 编程概括流程图



3.2 进入 ICSP 模式

如图 3-2 所示，进入 ICSP 编程 / 校验模式需要三个步骤：

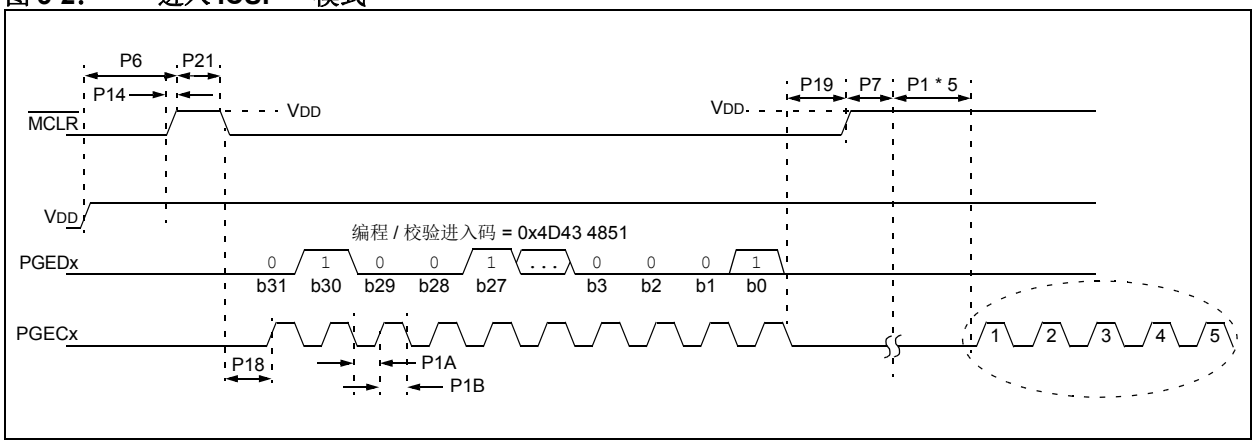
1. MCLR 短暂地驱动为高电平，然后再驱动为低电平 （P21）。
2. 将 32 位密钥序列随时钟移入到 PGEDx 引脚。必须经过至少 P18 的时间间隔才能将密钥序列移入到 PGEDx 引脚。
3. MCLR 在指定时间 P19 期间保持低电平，然后驱动为高电平。
4. 在 P7 + 5 * P1 的延时之后，必须在 PGECx 引脚上产生 5 个时钟脉冲。

注： 如果 MCLR 引脚上存在电容，进入 ICSP 模式的高电平时间可能有所不同。

该密钥序列为一个特定的 32 位形式 0100 1101 0100 0011 0100 1000 0101 0001（以其十六进制格式 0x4D43 4851 记忆更为简便）。只有在密钥序列有效时，器件才能进入 ICSP 模式。必须首先移入高 4 位的最高有效位 （Most Significant bit，MSb）。

一旦成功进入编程 / 校验模式，就能以串行方式访问程序存储器并对其编程。

图 3-2： 进入 ICSP™ 模式



3.3 ICSP 工作原理

进入 ICSP 模式后， CPU 为空闲。 CPU 执行由内部状态机控制。使用 PGECx 和 PGEDx 引脚随时钟移入用于命令 CPU 的 4 位控制代码 （见表 3-1）。

SIX 控制代码用于发送指令给 CPU 执行，而 REGOUT 控制代码用于通过 VISI 寄存器从器件中读出数据。

表 3-1： ICSP™ 模式下的 CPU 控制代码

4 位 控制代码	助记符	说明
0000	SIX	移入 24 位指令并执行。
0001	REGOUT	移出 VISI 寄存器。
0010-1111	N/A	保留。

3.3.1 SIX 串行指令执行

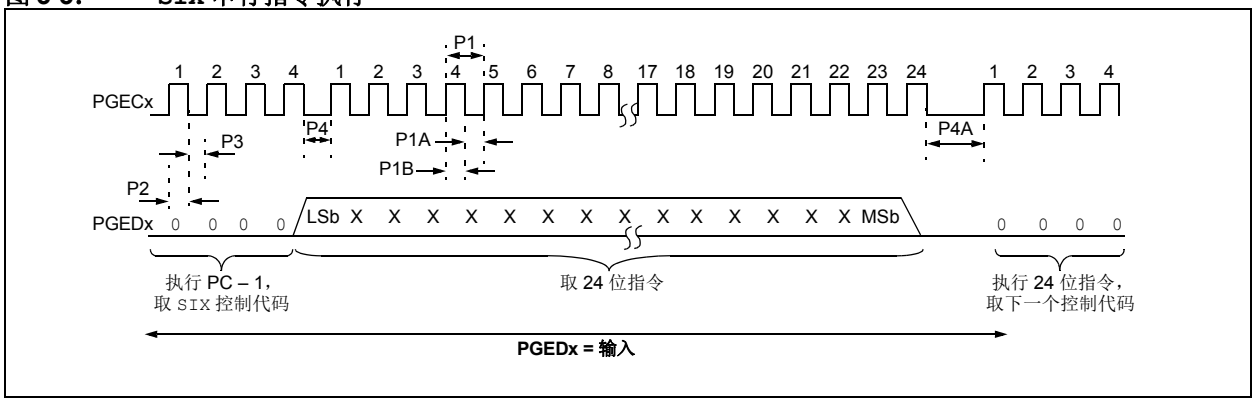
SIX控制代码允许执行汇编指令。当接收到SIX代码时，CPU 暂停 24 个时钟周期，在这段时间指令被移入内部缓冲区。一旦移入指令，状态机就允许其在接下来的 4 个时钟周期内执行。当执行接收到的指令时，状态机将同时移入下一条 4 位命令 （见图 3-3）。

注 1： 器件将在 PGECx 的上升沿锁存输入 PGEDx 的数据。对于所有数据的发送，都是先发送最低有效位（Least Significant bit， LSb）。

2： TBLRDH、TBLRDL、TBLWTH 和 TBLWTL 指令后必须跟两条 NOP 指令。

3： 在 ICSP 编程过程中， CLKO 引脚将翻转电平。如果外部逻辑连接到 CLKO，请确保此电平翻转在编程序列期间不会影响电路。

图 3-3： SIX 串行指令执行



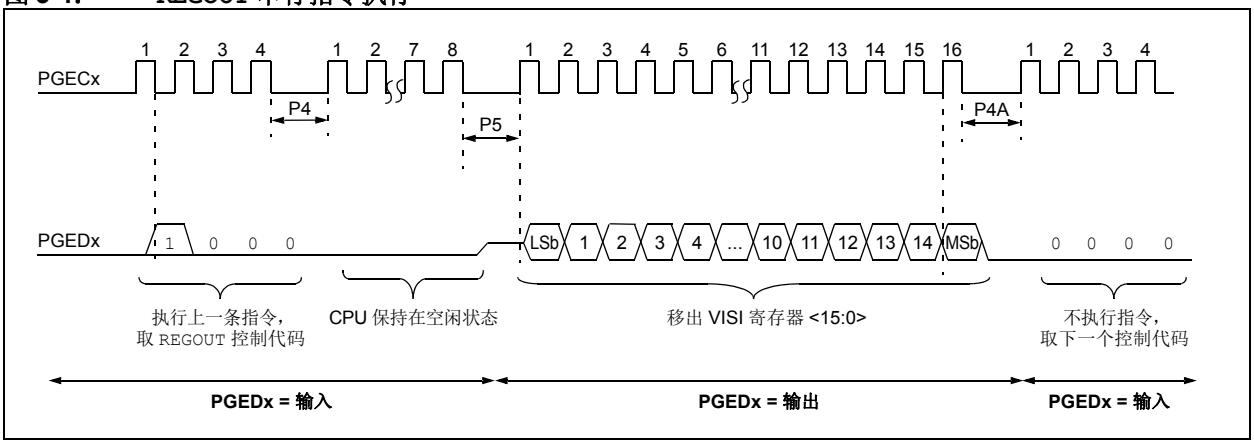
3.3.2 REGOUT 串行指令执行

REGOUT 控制代码允许在 ICSP 模式下从器件中读取数据。它用于通过 PGEDx 引脚在时钟控制下移出器件中 VISI 寄存器的内容。接收到 REGOUT 控制代码后，CPU 在 8 个周期内保持空闲。经过这 8 个周期后，还需要额外的 16 个周期才能将数据移出 （见图 3-4）。

REGOUT 代码的独特之处在于将控制代码发送到器件时， PGEDx 引脚为输入引脚。但是，一旦控制代码处理完毕，PGEDx 引脚就会变成输出引脚移出 VISI 寄存器的内容。

注： 器件将在 PGECx 上升沿在 PGEDx 线上输出数据。对于所有数据的发送，都是先发送最低有效位 （Least Significant bit， LSb）。

图 3-4： REGOUT 串行指令执行



3.4 在 ICSP 模式下对闪存编程

3.4.1 编程操作

对闪存执行的写 / 擦除操作是通过 NVMCON 寄存器控制的。编程的执行过程如下：设置 NVMCON 选择擦除操作 （表 3-2）或写操作 （表 3-3）的类型，并通过将 WR 控制位 （NVMCON<15>）置 1 启动编程。

在 ICSP 模式下，所有编程操作都采用自定时方式。当编程操作完成时硬件会清零 WR 控制位。 ICSP 编程器必须在 PGECx 引脚上提供足够的时钟脉冲，才能完成擦除或编程操作。关于擦除或写操作所需的最大时钟脉冲数的详细信息，请参见**第 9.0 节 “交流 / 直流特性和时序要求”**。

表 3-2： NVMCON 擦除操作

NVMCON 值	擦除操作
0x400E	整片擦除 （擦除用户存储区，不擦除执行程序存储区、器件 ID 或用户 OTP）。
0x4003	擦除程序存储器或执行程序存储区中的一页。

表 3-3： NVMCON 写操作

NVMCON 值	写操作
0x4001	双字编程操作。
0x4002	行编程操作。

3.4.2 启动和停止编程周期

为防止意外操作，必须将擦除/写启动序列写入NVMKEY 寄存器以允许进行任何擦除或编程操作。启动编程序列后应执行三条 NOP 指令。要启动擦除或写序列，必须完成以下步骤：

1. 将 0x55 写入 NVMKEY 寄存器。
2. 将 0xAA 写入 NVMKEY 寄存器。
3. 将 NVMCON 寄存器中的 WR 位置 1。
4. 执行 3 条 NOP 指令。

所有擦除和写周期都是自定时的。可对 WR 位进行轮询，以便为操作提供足够的 PGECx 时钟脉冲数，并确定擦除或写周期是否已经完成。

3.5 擦除程序存储器

擦除用户存储区的一般过程如图 3-5 所示。整片擦除和页擦除的过程非常相似，表 3-4 至表 3-5 介绍了这两个过程。

程序存储器最后一页的最后一行包含配置字。在编程这些字之前，必须先擦除它们。如果通过页擦除操作擦除这些字，则该页中的所有其他行也会被擦除。用户可能希望避免将该页的其余部分用于应用程序代码，或者希望确保在擦除之前复制 CS 页中的非配置数据并在之后对其进行重新编程。

注 1： 在向程序存储器写入任何数据之前必须先将其擦除。

2： 对于页擦除操作， NVMADR/NVMADRU 寄存器中还必须装入待擦除页的地址。

图 3-5： 擦除流程图

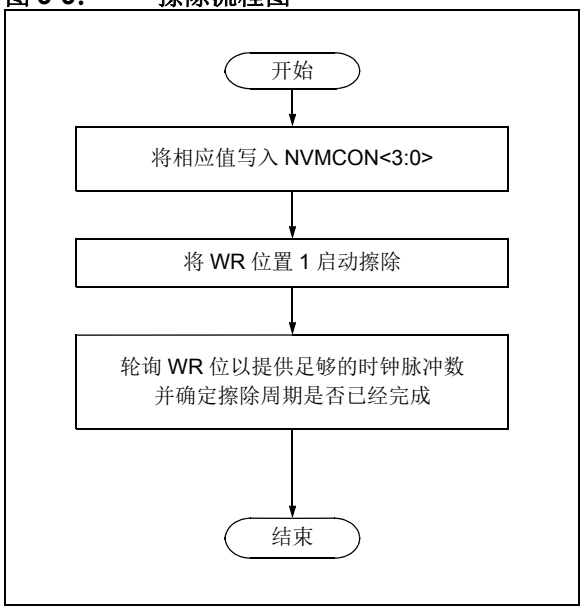


表 3-4： 整片擦除的串行指令执行

命令（二进制）	数据（十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：配置 NVMCON 寄存器以执行整片擦除。		
0000	2400E0	MOV #0x400E, W0
0000	883B00	MOV W0, NVMCON
第 3 步：将 WR 位置 1。		
0000	200550	MOV #0x55, W0
0000	883B30	MOV W0, NVMKEY
0000	200AA0	MOV #0xAA, W0
0000	883B30	MOV W0, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 4 步：重复该步骤以轮询 WR 位，直到它被硬件清零为止。		
0000	040200	GOTO 0x200
0000	000000	NOP
0000	803B02	MOV NVMCON, W2
0000	000000	NOP
0000	883C22	MOV W2, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
第 5 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

表 3-5：　页擦除的串行指令执行

命令 （二进制）	数据 （十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：设置 NVMCON 寄存器以擦除一页。		
0000	240030	MOV #0x4003, W0
0000	883B00	MOV W0, NVMCON
第 3 步：将待擦除页的地址装入 NVMADR 寄存器对。		
0000	200000	MOV #PAGE_ADDR_LO, W0
0000	883B10	MOV W0, NVMADR
0000	200000	MOV #PAGE_ADDR_HI, W0
0000	883B20	MOV W0, NVMADRU
第 4 步：将 WR 位置 1。		
0000	200550	MOV #0x55, W0
0000	883B30	MOV W0, NVMKEY
0000	200AA0	MOV #0xAA, W0
0000	883B30	MOV W0, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 5 步：重复该步骤以轮询 WR 位，直到它被硬件清零为止。		
0000	040200	GOTO 0x200
0000	000000	NOP
0000	803B02	MOV NVMCON, W2
0000	000000	NOP
0000	883C22	MOV W2, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
第 6 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

3.6 写代码存储区

对于 PIC24FJ256GA705 器件，代码存储区的写入分为三个步骤：

- 向存储器映射的写锁存器（位于地址从 0xFA 0000 至 0xFA 00FE 的配置存储空间）中写入数据；
- 设置目标地址；
- 启动存储器写序列

有两种方法可用于写入代码存储区：使用写锁存器的双字写操作或者 128 字行写操作。图 3-7 概括说明了这两种方法。

双字写操作每次对代码存储区编程两个指令字。将两个字装入写锁存器。接着，启动写序列，最后，检查 WR 位以确定序列是否完成。继续执行该过程，直到完成所有数据的编程。

表 3-6 给出了双字写操作的 ICSP 编程示例。

装入编程锁存器的数据必须采用打包格式，如图 3-6 所示。

图 3-6： 指令字的打包格式

LSWx：指令字的低 16 位
MSBx：指令字的最高字节

注： 当传输的指令字数为奇数时，MSB2 为零，且不能发送 LSW2。

图 3-7：　代码存储区编程流程图

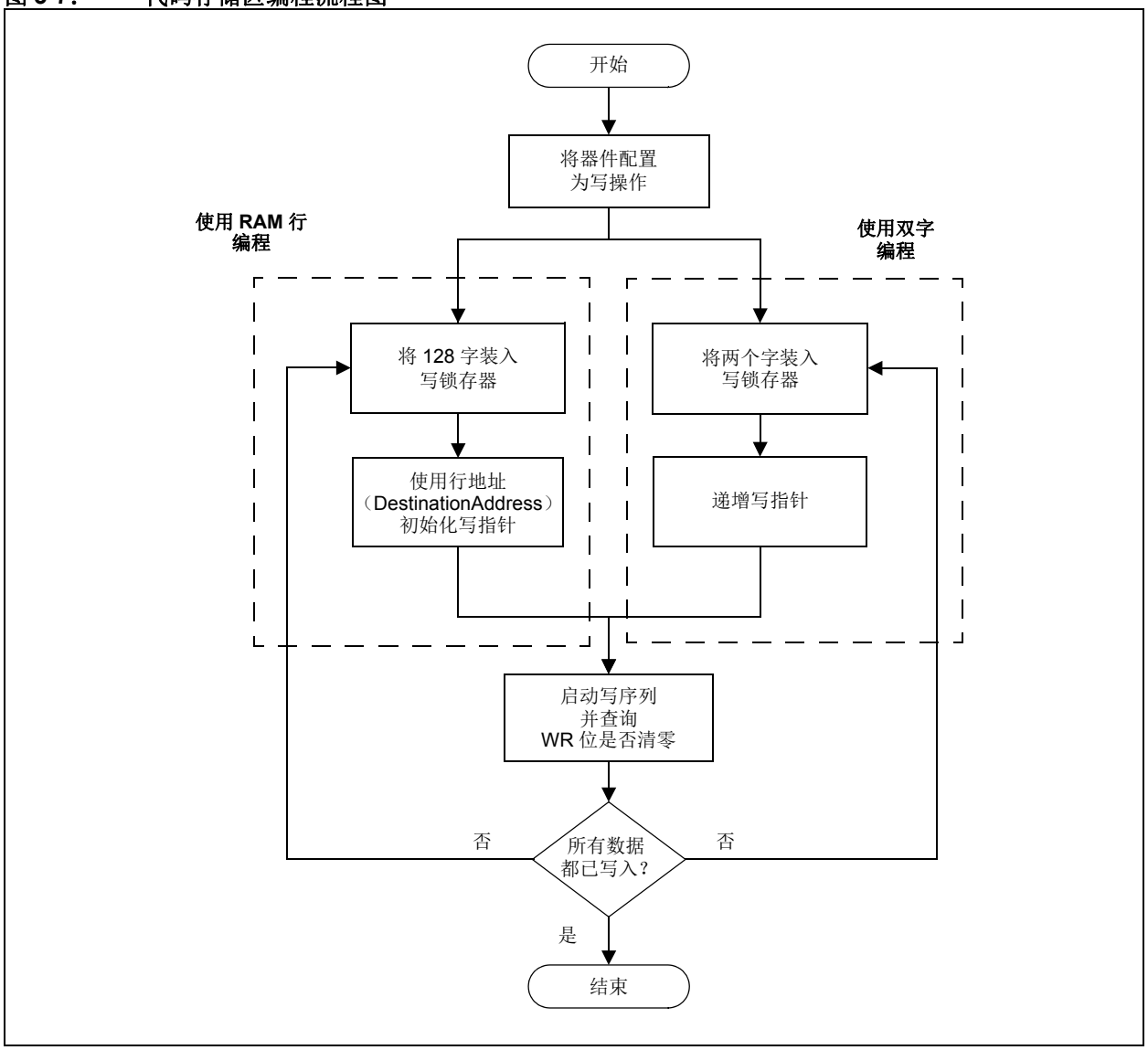


表 3-6： 代码存储区编程的串行指令执行：双字锁存器写操作

命令 （二进制）	数据 （十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：初始化 TBLPAG 寄存器以写入锁存器。		
0000	200FAC	MOV #0xFA, W12
0000	8802AC	MOV W12, TBLPAG
第 3 步：将接下来 2 个待编程的打包指令字装入 W0:W2。		
0000	2xxxx0	MOV #<LSW0>, W0
0000	2xxxx1	MOV #<MSB1:MSB0>, W1
0000	2xxxx2	MOV #<LSW1>, W2
第 4 步：设置读指针 （W6）和写指针 （W7）并装载 （下一组）写锁存器。		
0000	EB0300	CLR W6
0000	000000	NOP
0000	EB0380	CLR W7
0000	000000	NOP
0000	BB0BB6	TBLWTL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP
0000	BBDBB6	TBLWTH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BBEBB6	TBLWTH.B [W6++], [++W7]
0000	000000	NOP
0000	000000	NOP
0000	BB1BB6	TBLWTL.W [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
第 5 步：设置 NVMADRU/NVMADR 寄存器对以指向正确的地址。		
0000	2xxxx3	MOV #DestinationAddress<15:0>, W3
0000	2xxxx4	MOV #DestinationAddress<23:16>, W4
0000	883B13	MOV W3, NVMADR
0000	883B24	MOV W4, NVMADRU
第 6 步：设置 NVMCON 寄存器以编程 2 个指令字。		
0000	24001A	MOV #0x4001, W10
0000	883B0A	MOV W10, NVMCON
0000	000000	NOP
第 7 步：启动写周期。		
0000	200551	MOV #0x55, W1
0000	883B31	MOV W1, NVMKEY
0000	200AA1	MOV #0xAA, W1
0000	883B31	MOV W1, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP

表 3-6： 代码存储区编程的串行指令执行：双字锁存器写操作 （续）

命令 （二进制）	数据 （十六进制）	说明
第 8 步：等待编程操作完成并确认 WR 位清零。		
0000	803B00	MOV NVMCON, W0
0000	883C20	MOV W0, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
—	—	; Repeat until the WR bit is clear.
第 9 步：重复执行第 3 步至第 8 步，直到所有代码存储区都被编程为止。		
第 10 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

行写操作每次编程一行（128 个指令字）。首先，初始化表指针，使其指向程序锁存器，然后通过表写操作将数据写入程序锁存器。接着，使用行地址（DestinationAddress）对写指针进行初始化（NVMADRU 和 NVMADR 寄存器对）。最后，启动写序列并检查 WR 位以确定行编程是否完成。重复执行该过程，直到完成所有数据的编程。表 3-7 给出了行写操作的 ICSP 编程的详细步骤。

为了使编程时间最短，待编程数据以数据打包格式存储在 W0:W5 寄存器中（图 3-8）。所使用的格式与 PE 使用的打包格式相同。更多信息，请参见**第 6.2.2 节“数据打包格式”**。

图 3-8： W0:W5 中的打包指令字

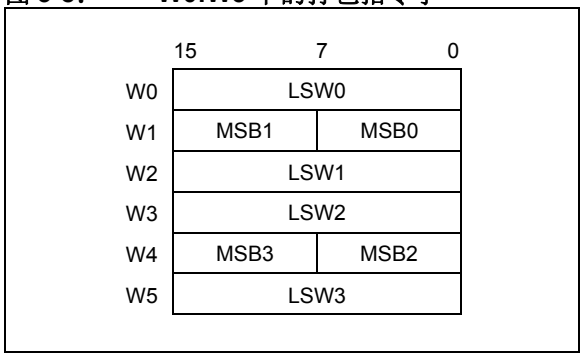


表 3-7： 代码存储区编程的串行指令执行：行写

命令（二进制）	数据（十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：设置 NVMCON 寄存器以编程 128 个指令字。		
0000	240020	MOV #0X4002, W0
0000	883B00	MOV W0, NVMCON
第 3 步：初始化 TBLPAG 寄存器以写入锁存器。		
0000	200FAC	MOV #0xFA, W12
0000	8802AC	MOV W12, TBLPAG
第 4 步：将接下来 4 个待编程的指令字装入 W0:W5。		
0000	2xxxx0	MOV #<LSW0>, W0
0000	2xxxx1	MOV #<MSB1:MSB0>, W1
0000	2xxxx2	MOV #<LSW1>, W2
0000	2xxxx3	MOV #<LSW2>, W3
0000	2xxxx4	MOV #<MSB3:MSB2>, W4
0000	2xxxx5	MOV #<LSW3>, W5
第 5 步：设置读指针（W6）并装载（下一组）写锁存器。		
0000	EB0300	CLR W6
0000	000000	NOP
0000	EB0380	CLR W7
0000	000000	NOP
0000	BB0BB6	TBLWTL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP

表 3-7： 代码存储区编程的串行指令执行：行写 （续）

命令 （二进制）	数据 （十六进制）	说明
第 5 步：设置读指针 （W6）并装载 （下一组）写锁存器。		
0000	BBDBB6	TBLWTH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BBEBB6	TBLWTH.B [W6++], [++W7]
0000	000000	NOP
0000	000000	NOP
0000	BB1BB6	TBLWTL [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BB0BB6	TBLWTL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP
0000	BBDBB6	TBLWTH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BBEBB6	TBLWTH.B [W6++], [++W7]
0000	000000	NOP
0000	000000	NOP
0000	BB1BB6	TBLWTL [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
第 6 步：将第 4 步和第 5 步重复执行 32 次，将 128 条指令装入写锁存器。		
第 7 步：设置 NVMADRU/NVMADR 寄存器对以指向正确的地址。		
0000	2xxxx3	MOV #DestinationAddress<15:0>, W3
0000	2xxxx4	MOV #DestinationAddress<23:16>, W4
0000	883B13	MOV W3, NVMADR
0000	883B24	MOV W4, NVMADRU
第 8 步：执行 WR 位解锁序列并启动写周期。		
0000	200550	MOV #0x55, W0
0000	883B30	MOV W0, NVMKEY
0000	200AA0	MOV #0xAA, W0
0000	883B30	MOV W0, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 9 步：重复该步骤以轮询 WR 位，直到它被硬件清零为止。		
0000	040200	GOTO 0x200
0000	000000	NOP
0000	803B02	MOV NVMCON, W2
0000	883C22	MOV W2, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
第 10 步：复位器件的内部程序计数器 （Program Counter， PC）。		
0000	040200	GOTO 0x200
0000	000000	NOP
第 11 步：重复执行第 3 步至第 9 步，直到所有代码存储区都被编程为止。		
第 12 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

3.7 写配置位

写配置位与写代码存储区的过程类似。表 3-8 给出了写配置位的 ICSP 编程的详细步骤。

要在编程这些配置位后更改其值，必须先擦除器件（如**第 3.5 节 “擦除程序存储器”**所述），然后再编程为期望值。请注意，只能将配置位从 1 编程为 0 来使能代码保护；无法将其从 0 编程为 1。

表 3-8： 配置字编程的串行指令执行：双字锁存器写操作

命令（二进制）	数据（十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：初始化 TBLPAG 寄存器以写入锁存器。		
0000	200FAC	MOV #0xFA, W12
0000	8802AC	MOV W12, TBLPAG
第 3 步：将接下来 2 个待编程的配置字装入 W0:W1。		
0000	2xxxx0	MOV #<Config lower word data>, W0
0000	2FFxx1	MOV #<Config upper word data>, W1
—	—	; Upper word is 0xFFFF for all Configuration Words except FBTSEQ
0000	2FFFF2	MOV #0xFFFF, W2
第 4 步：设置读指针（W6）和写指针（W7）并装载（下一组）写锁存器。		
0000	EB0300	CLR W6
0000	000000	NOP
0000	EB0380	CLR W7
0000	000000	NOP
0000	BB0BB6	TBLWTL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP
0000	BBDBB6	TBLWTH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BBEBB6	TBLWTH.B [W6++], [++W7]
0000	000000	NOP
0000	000000	NOP
0000	BB1BB6	TBLWTL.W [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
第 5 步：设置 NVMADRU/NVMADR 寄存器对以指向正确的地址。		
0000	2xxxx3	MOV #DestinationAddress<15:0>, W3
0000	2xxxx4	MOV #DestinationAddress<23:16>, W4
0000	883B13	MOV W3, NVMADR
0000	883B24	MOV W4, NVMADRU

表 3-8： 配置字编程的串行指令执行：双字锁存器写操作 （续）

命令 （二进制）	数据 （十六进制）	说明
第 6 步：设置 NVMCON 寄存器来编程 2 个指令字。		
0000	24001A	MOV #0x4001, W10
0000	883B0A	MOV W10, NVMCON
0000	000000	NOP
第 7 步：启动写周期。		
0000	200551	MOV #0x55, W1
0000	883B31	MOV W1, NVMKEY
0000	200AA1	MOV #0xAA, W1
0000	883B31	MOV W1, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 8 步：等待编程操作完成并确认 WR 位清零。		
0000	803B00	MOV NVMCON, W0
0000	883C20	MOV W0, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
—	—	; Repeat until the WR bit is clear.
第 9 步：重复执行第 3 步至第 8 步，直到所有配置字都被编程为止。		
第 10 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

3.8 读代码存储区

读代码存储区是通过执行一系列 TBLRD 指令和使用 REGOUT 命令随时钟移出数据完成的。

表 3-9 给出了读代码存储区的 ICSP 编程的详细步骤。

为了使读时间最短，将使用与PE相同的数据打包格式。关于数据打包格式的更多详细信息，请参见**第 6.2 节“编程执行程序命令”**。

3.9 读配置字

读配置字的过程与读代码存储区的过程相同，如表 3-9 所示。每次读取多个配置字中的一个。

表 3-9： 读代码存储区的串行指令执行

命令（二进制）	数据（十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：初始化写指针 （W7），使其指向 VISI 寄存器。		
0000	207847	MOV #VISI, W7
0000	000000	NOP
第 3 步：为执行 TBLRD 指令初始化 TBLPAG 寄存器和读指针 （W6）。		
0000	200xx0	MOV #<SourceAddress23:16>, W0
0000	8802A0	MOV W0, TBLPAG
0000	2xxxx6	MOV #<SourceAddress15:0>, W6
第 4 步：使用 REGOUT 命令，通过 VISI 寄存器读取并在时钟控制下移出代码存储区接下来 2 个存储单元的内容。		
0000	BA0B96	TBLRDL [W6], [W7]
0000	000000	NOP
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	BADBB6	TBLRDH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BAD3D6	TBLRDH.B [++W6], [W7--]
0000	000000	NOP
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	BA0BB6	TBLRDL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
第 5 步：复位器件的内部程序计数器。		
0000	040200	GOTO 0x200
0000	000000	NOP
第 6 步：重复第 3 步至第 5 步，直到读取完所需的所有代码存储区为止 （请注意，“复位器件的内部程序计数器”将作为第 5 步）。		

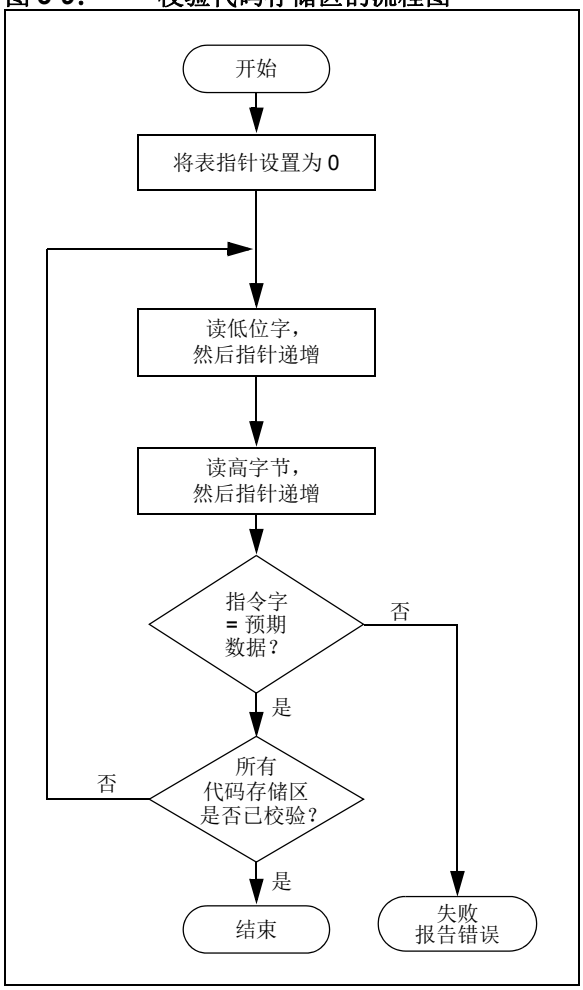
3.10 校验代码存储区和配置位

校验的步骤涉及读回代码存储空间并将读到的值与存储在编程器缓冲区中的副本作比较。使用其余代码校验配置字。

校验过程如图 3-9 所述。读取指令的低位字，然后读取高位字的低字节，并将其与编程器缓冲区中存储的指令相比较。关于读取代码存储区的实现细节，请参见**第 3.8 节 “读代码存储区”**。

> **注：** 由于配置字节包含器件代码保护位，如果要使能代码保护，应在写入代码存储区后立即对其进行校验。这是因为如果代码保护位清零后发生器件复位，将无法读取或校验器件。

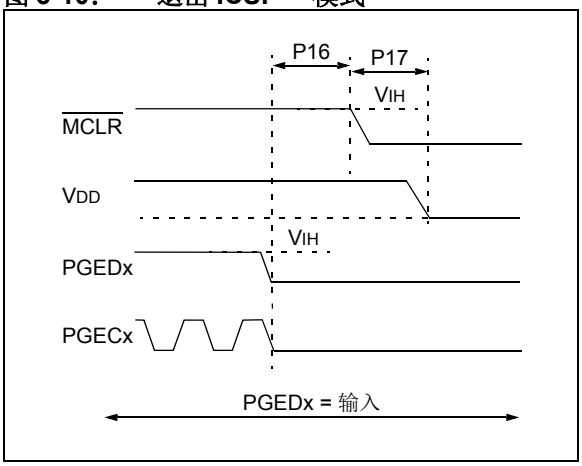
图 3-9： 校验代码存储区的流程图



3.11 退出 ICSP 模式

通过将 VIH 从 $\overline{\text{MCLR}}$ 引脚移除可退出编程 / 校验模式，如图 3-10 所示。退出操作的唯一要求是在 PGECx 和 PGEDx 引脚的最后一个时钟和编程信号之后的 P16 时间间隔后移除 VIH。

图 3-10： 退出 ICSP™ 模式



4.0 器件编程 —— 增强型 ICSP

本节讨论了如何通过增强型 ICSP 和编程执行程序（PE）对器件编程。 PE 存放在执行程序存储区（独立于代码存储区）中，当进入增强型 ICSP 编程模式时执行 PE。PE 使用一个简单的命令集和通信协议为编程器（主设备）提供了对 PIC24FJ256GA705 系列器件进行编程和校验的机制。 PE 可提供以下几项基本功能：

- 读存储器
- 擦除存储器
- 对存储器编程
- 空白检查

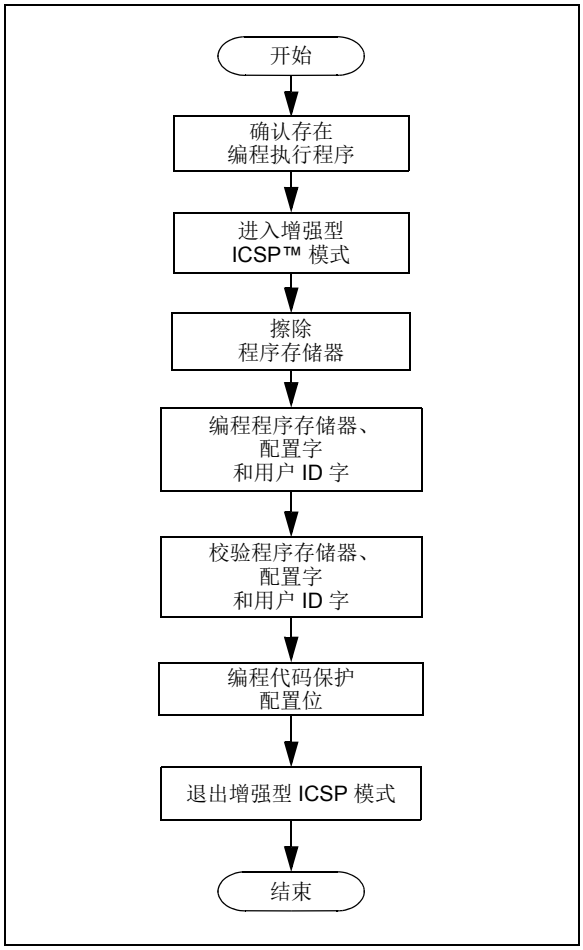
PE 执行擦除、编程和校验器件所需的低级任务。这允许编程器通过发出相应的命令和数据来对器件编程。**第 6.2 节 “编程执行程序命令”**提供了每条命令的详细说明。

注：	PE 使用器件的数据 RAM 来存放变量和执行程序。运行 PE 后，数据 RAM 中的内容就无法确定了。

4.1 编程过程概述

图 4-1 高度概括了编程过程。首先，必须确定 PE 是否存放在执行程序存储区中，然后进入增强型 ICSP 模式。接着擦除程序存储器，并编程和校验程序存储器和配置字。最后，编程代码保护配置位（如果需要），退出增强型 ICSP 模式。

图 4-1： 增强型 ICSP™ 编程概括流程图



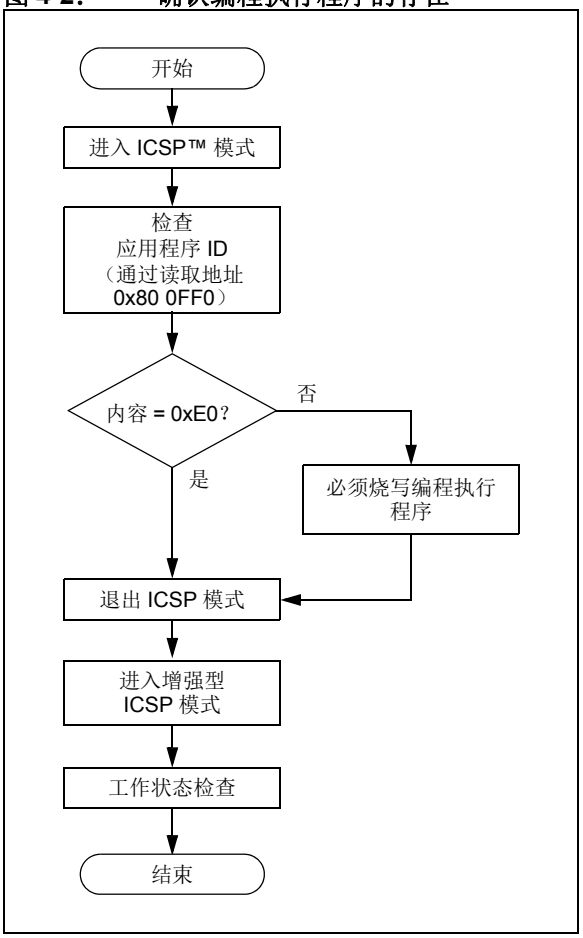
4.2 确认编程执行程序的存在

在开始编程之前，编程器必须确认 PE 存放在执行程序存储区中。图 4-2 给出了该任务的流程。

首先进入ICSP模式，然后读取存储在执行程序存储区中的唯一应用程序 ID 字。如果应用程序 ID 的值为 0xE0，表明编程执行程序位于执行程序存储区中，可以对器件编程。但是，如果应用程序 ID 字不存在，则必须通过使用**第 5.0 节 “将编程执行程序烧写到存储区”**中所述的方法将 PE 烧写到执行程序代码存储区中。

第 3.0 节 “器件编程 ——ICSP”介绍了 ICSP 编程方法。**第 4.3 节 “读应用程序 ID 字”**介绍了在 ICSP 模式下读取应用程序 ID 字的过程。

图 4-2： 确认编程执行程序的存在



4.3 读应用程序 ID 字

应用程序 ID 字存储在执行程序代码存储区的地址 0x80 0FF0 中。要读取该存储单元，必须使用 SIX 控制代码将该程序存储单元中的内容送到 VISI 寄存器。然后，必须使用 REGOUT 控制代码随时钟将 VISI 寄存器的内容移出器件。必须串行发送给器件以执行该操作的相应控制和指令代码，如表 4-1 中所示。

如果应用程序 ID 的值为 0xE0，表明编程执行程序位于执行程序存储区中，可以采用本节中所述的过程对器件编程。但是，如果应用程序 ID 为其他值，则表明 PE 不在相应的存储区中；必须在对器件编程前先将编程执行程序装入执行程序存储区中。**第 5.0 节 “将编程执行程序烧写到存储区”** 中介绍了将 PE 装入存储区的过程。

表 4-1： 读应用程序 ID 字的串行指令执行

命令（二进制）	数据（十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：为执行 TBLRD 指令初始化 TBLPAG 和读指针 （W0）。		
0000	200800	MOV #0x80, W0
0000	8802A0	MOV W0, TBLPAG
0000	20FF00	MOV #0xFF0, W0
0000	207841	MOV #VISI, W1
0000	000000	NOP
0000	BA0890	TBLRDL [W0], [W1]
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 3 步：使用 REGOUT 命令输出 VISI 寄存器的内容。		
0001	<VISI>	; Clock out the contents of the VISI register.

4.4 进入增强型 ICSP 模式

如图 4-3 所示，进入增强型 ICSP 编程 / 校验模式需要三个步骤：

1. 将 MCLR 引脚短暂地驱动为高电平，然后再驱动为低电平。
2. 将 32 位密钥序列随时钟移入到 PGEDx 引脚。
3. 然后将 MCLR 驱动为高电平并保持一段指定的时间。

加到MCLR的编程电压为VIH，对于PIC24FJ256GA705系列器件，该电压实际就是 VDD。对于 VIH 没有最短保持时间要求。在移除 VIH 后，必须经过至少 P18 时间间隔才能将密钥序列移入 PGEDx 引脚。

该密钥序列为一个特定的 32 位形式 0100 1101 0100 0011 0100 1000 0101 0000（以其十六进制格式 0x4D43 4850 记忆更为简便）。只有在密钥序列有效时，器件才能进入编程 / 校验模式。必须首先移入高 4 位的最高有效位 （MSb）。

一旦密钥序列完全移入，就必须将 VIH 加到 MCLR 并在编程 / 校验模式下始终保持该电压。必须经过至少 P19、P7 和 P1 * 5 的时间间隔才能将数据移入 PGEDx 引脚。在 P7 之前出现在 PGEDx 引脚上的信号被认定为无效。

一旦成功进入编程 / 校验模式，就能以串行方式访问程序存储器并对其编程。在编程 / 校验模式下，所有未用的 I/O 引脚都被置于高阻态。

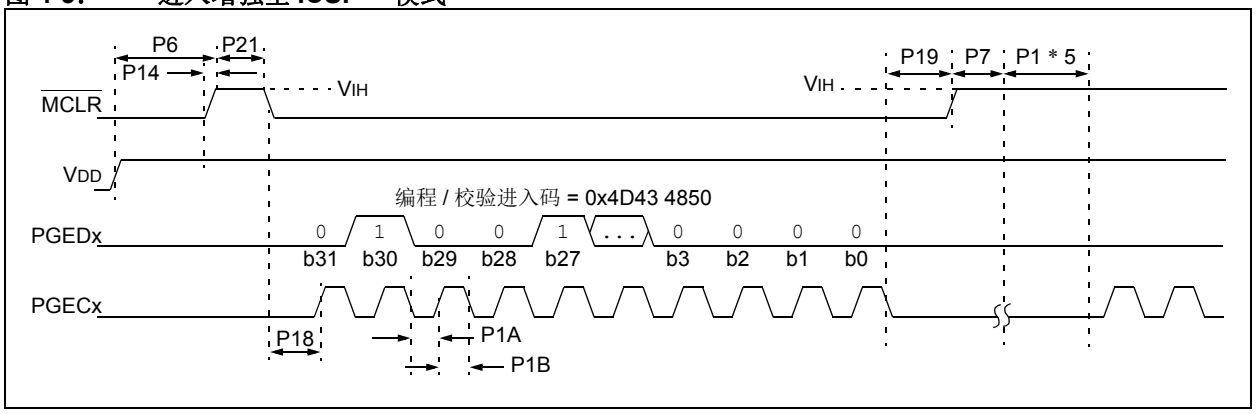
4.5 空白检查

术语“空白检查”的含义是校验器件是否已被成功擦除且没有已编程的存储单元。空白或已擦除的存储单元始终读为 1。

器件 ID 寄存器 （0xFF 0000:0xFF 0002）可被空白检查忽略，因为此区域存储了不可擦除的器件信息。另外，所有未实现的存储空间都应该被空白检查忽略。

QBLANK 命令用于空白检查。该命令检测代码存储区，确定这些存储区是否已被擦除。将返回“BLANK”（空白）或“NOT BLANK”（非空白）响应。如果确定器件非空白，则必须在尝试对芯片进行编程前先将其擦除。

图 4-3： 进入增强型 ICSP™ 模式



4.6 代码存储区编程

4.6.1 编程方法

如果要通过 PE 对代码存储区编程，有两条命令可用。PROG2W 命令将两个 24 位指令字编程到以指定地址开始的程序存储器并进行校验。另一条PROGP命令速度更快，最多可将 128 个指令字（每个 24 位）编程到以指定地址开始的程序存储区并进行校验。关于这两条命令的完整说明，请参见**第 6.0 节 “编程执行程序”**。

图 4-4 和图 4-5 显示了 PROG2W 和 PROGP 命令的编程方法。在这两个示例中，对器件的 87552 个指令字进行了编程。

> **注：** 如果需要烧写自举程序，则一定不能将其代码烧写到代码存储区的第一页中。例如，如果位于地址 0x200 处的自举程序尝试擦除第一页，则它会意外地擦除自身。请务必将自举程序烧写到第二页中 （例如， 0x400）。

图 4-4： 双字编程的流程图

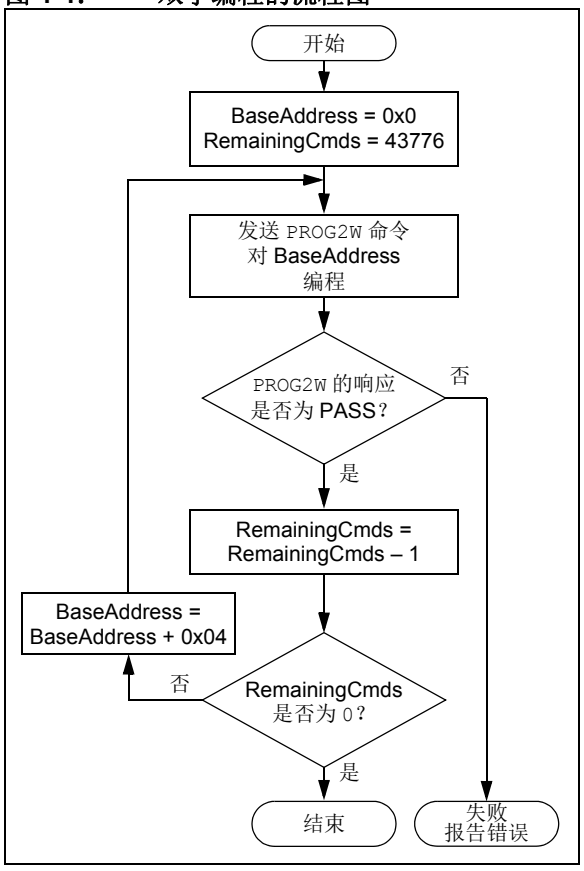
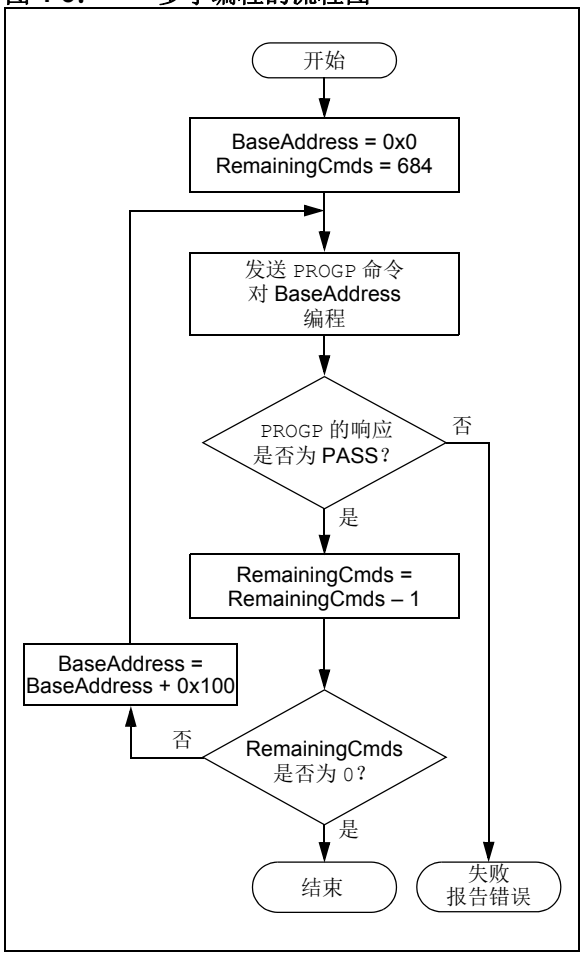


图 4-5： 多字编程的流程图

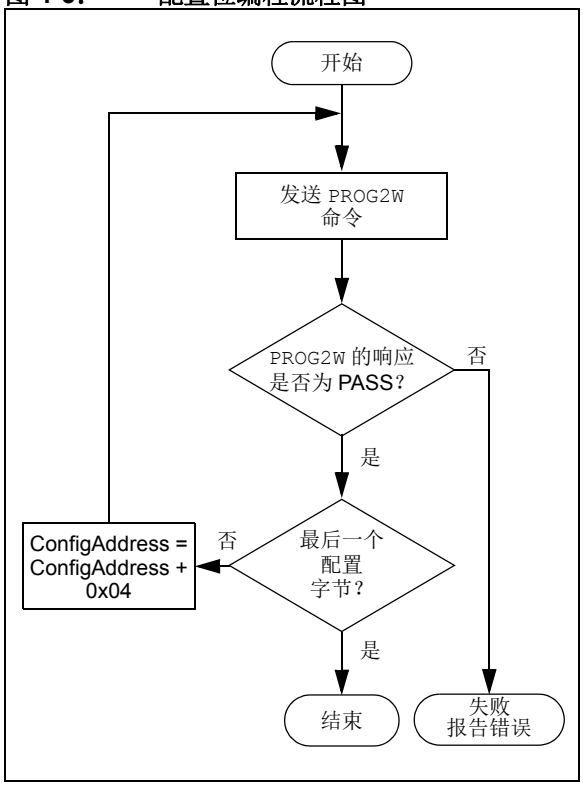


4.7 配置位编程

使用 PROG2W 命令一次可编程一个配置位。该命令指定配置数据和地址。编程配置位时，任何未实现位都必须编程为 1。

需要多条 PROG2W 命令才能对所有配置位进行编程。配置位编程的流程图如图 4-6 所示。

图 4-6： 配置位编程流程图



4.8 编程校验

一旦代码存储区编程完毕，就可以对存储区的内容进行校验以确保编程成功。校验需要对代码存储区执行读回操作，并将读回的数据与编程器缓冲区中存储的副本进行比较。

READP命令可对所有已编程的代码存储区和配置字执行读回操作。

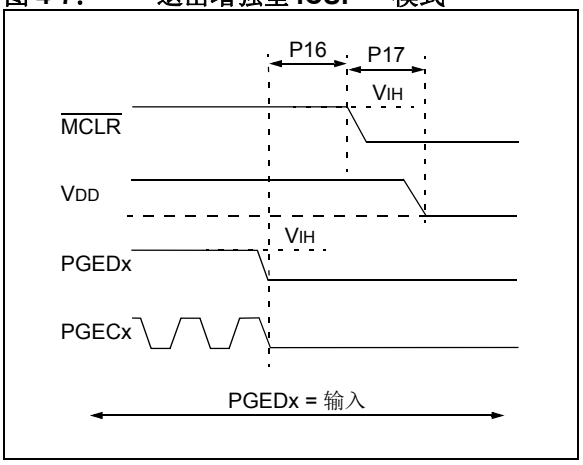
或者，也可以在对整个器件编程完毕后，让编程器通过计算校验和来执行校验。

关于计算校验和的更多信息，请参见**第 8.0 节 “校验和计算”**。

4.9 退出增强型 ICSP 模式

通过将 VIH 从 $\overline{\text{MCLR}}$ 引脚移除可退出编程 / 校验模式，如图 4-7 所示。退出操作的唯一要求是在 PGECx 和 PGEDx 引脚的最后一个时钟和编程信号之后的 P16 时间间隔后移除 VIH。

图 4-7： 退出增强型 ICSP™ 模式



5.0 将编程执行程序烧写到存储区

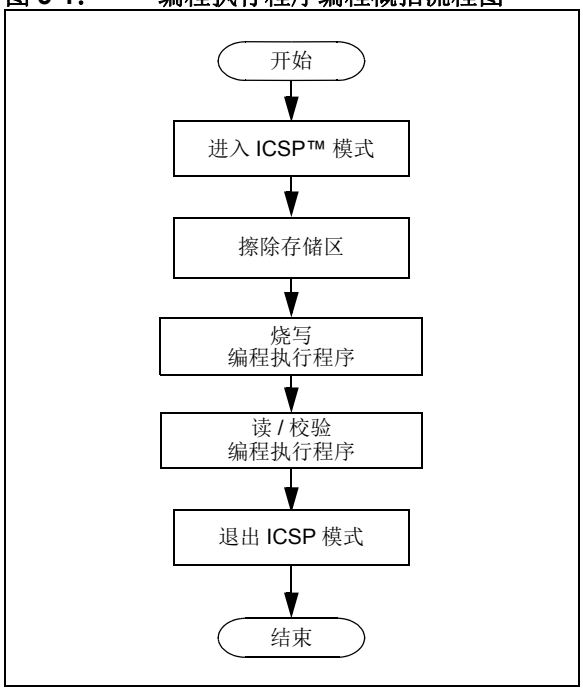
注： 编程执行程序（PE）可从 Microchip 网站上每款器件的页面获取：www.microchip.com。

5.1 概述

如果已确定 PE 不在执行程序存储区中（如**第 4.2 节“确认编程执行程序的存在”**中所述），则必须将 PE 烧写到执行程序存储区中。

图5-1 给出了将PE烧写到执行程序存储区的大概过程。首先，必须进入 ICSP 模式，并擦除执行程序存储区和用户存储区，然后才能烧写和校验PE。最后，退出ICSP模式。

图 5-1： 编程执行程序编程概括流程图



5.2 擦除执行程序存储区

执行程序存储区可以通过一系列页擦除操作擦除，如图5-2所示。这些操作包括将NVMCON设置为0x4003，执行编程周期以及对执行程序存储区的其余页重复操作。

表 5-1 给出了批量擦除存储器的 ICSP 编程过程。

注： 必须始终先擦除 PE，再对其进行编程，如图 5-1 所述。

图 5-2： 批量擦除流程图

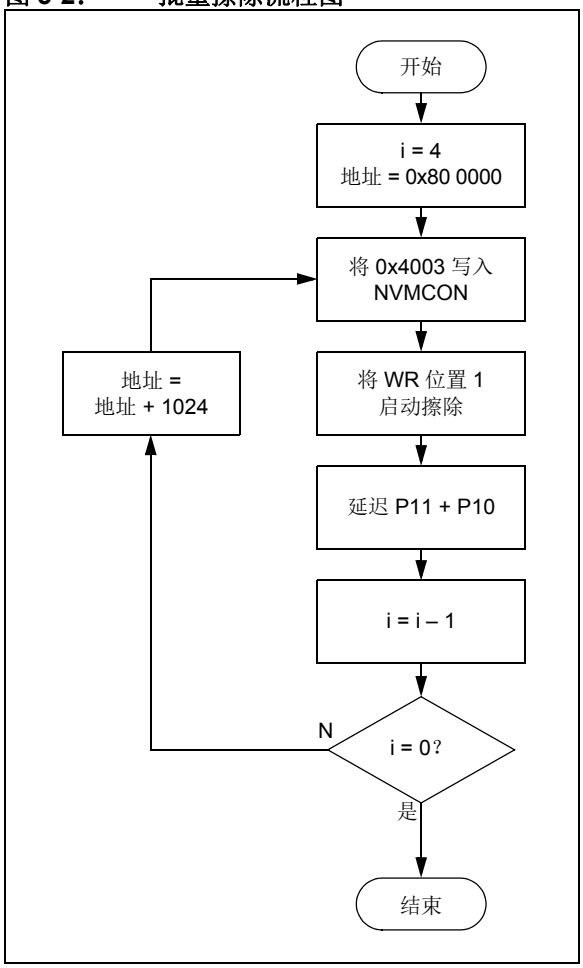


表 5-1： 擦除执行程序存储区的串行指令执行

命令 （二进制）	数据 （十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
0000	000000	NOP
第 2 步：设置 NVMCON 寄存器以擦除一页。		
0000	240030	MOV #0x4003, W0
0000	883B00	MOV W0, NVMCON
第 3 步：将待擦除页的地址装入 NVMADR 寄存器对。		
0000	200004	MOV #0000, W4
0000	883B14	MOV W4, NVMADR
0000	200800	MOV #0080, W0
0000	883B20	MOV W0, NVMADRU
第 4 步：将 WR 位置 1。		
0000	200550	MOV #0x55, W0
0000	883B30	MOV W0, NVMKEY
0000	200AA0	MOV #0xAA, W0
0000	883B30	MOV W0, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 5 步：重复该步骤以查询 WR 位，直到它被硬件清零为止。		
0000	040200	GOTO 0x200
0000	000000	NOP
0000	803B02	MOV NVMCON, W2
0000	883C22	MOV W2, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
第 6 步：W4 递增 1024 （0x400）。		
0000	204003	MOV #400, W3
0000	418204	ADD W3, W4, W4
0000	883B14	MOV W4, NVMADR
第 7 步：重复执行第 4 步至第 6 步，直到整个执行程序存储区被擦除为止。		
第 8 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

5.3 烧写编程执行程序

将 PE 存储到执行程序存储区的过程与对代码存储区的常规编程类似。首先必须擦除执行程序存储区，然后使用双字写操作（两个指令字）或行写操作（128 个指令字）进行编程。两种方法的控制流程与代码存储区编程的控制流程相同，如图 3-7 所示。

表 5-2 和表 5-3 给出了 PE 存储区的 ICSP 编程过程。为了使编程时间最短，将使用与 PE 相同的数据打包格式。关于数据打包格式的更多详细信息，请参见**第 6.2 节“编程执行程序命令”**。

表 5-2： 烧写编程执行程序 （双字锁存器写操作）

命令 （二进制）	数据 （十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：初始化 TBLPAG 寄存器以写入锁存器。		
0000	200FAC	MOV #0xFA, W12
0000	8802AC	MOV W12, TBLPAG
第 3 步：将接下来 2 个待编程的打包指令字装入 W0:W2。		
0000	2xxxx0	MOV #<LSW0>, W0
0000	2xxxx1	MOV #<MSB1:MSB0>, W1
0000	2xxxx2	MOV #<LSW1>, W2
第 4 步：设置读指针 （W6）和写指针 （W7）并装载写锁存器。		
0000	EB0300	CLR W6
0000	000000	NOP
0000	EB0380	CLR W7
0000	000000	NOP
0000	BB0BB6	TBLWTL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP
0000	BBDBB6	TBLWTH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BBEBB6	TBLWTH.B [W6++], [++W7]
0000	000000	NOP
0000	000000	NOP
0000	BB1BB6	TBLWTL.W [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
第 5 步：设置 NVMADRU/NVMADR 寄存器对以指向正确的行。		
0000	2xxxx3	MOV #DestinationAddress<15:0>, W3
0000	2xxxx4	MOV #DestinationAddress<23:16>, W4
0000	883B13	MOV W3, NVMADR
0000	883B24	MOV W4, NVMADRU
第 6 步：设置 NVMCON 寄存器来编程 2 个指令字。		
0000	24001A	MOV #0x4001, W10
0000	000000	NOP
0000	883B0A	MOV W10, NVMCON
0000	000000	NOP
0000	000000	NOP

表 5-2： 烧写编程执行程序 （双字锁存器写操作）（续）

命令 （二进制）	数据 （十六进制）	说明
第 7 步：启动写周期。		
0000	200551	MOV #0x55, W1
0000	883B31	MOV W1, NVMKEY
0000	200AA1	MOV #0xAA, W1
0000	883B31	MOV W1, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 8 步：等待编程操作完成并确认 WR 位清零。		
0000	000000	NOP
0000	803B00	MOV NVMCON, W0
0000	000000	NOP
0000	883C20	MOV W0, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
—	—	; Repeat until the WR bit is clear.
第 9 步：重复执行第 3 步至第 8 步，直到所有执行程序存储区都被编程为止。		
第 10 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

表 5-3： 烧写编程执行程序 （行写）

命令 （二进制）	数据 （十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：设置 NVMCON 寄存器以编程 128 个指令字。		
0000	240020	MOV #0X4002, W0
0000	883B00	MOV W0, NVMCON
第 3 步：初始化 TBLPAG 寄存器以写入锁存器。		
0000	200FAC	MOV #0xFA, W12
0000	8802AC	MOV W12, TBLPAG
第 4 步：将接下来 4 个待编程的指令字装入 W0:W5。		
0000	2xxxx0	MOV #<LSW0>, W0
0000	2xxxx1	MOV #<MSB1:MSB0>, W1
0000	2xxxx2	MOV #<LSW1>, W2
0000	2xxxx3	MOV #<LSW2>, W3
0000	2xxxx4	MOV #<MSB3:MSB2>, W4
0000	2xxxx5	MOV #<LSW3>, W5
第 5 步：设置读指针 （W6）并装载 （下一组）写锁存器。		
0000	EB0300	CLR W6
0000	000000	NOP
0000	EB0380	CLR W7
0000	000000	NOP
0000	BB0BB6	TBLWTL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP
0000	BBDBB6	TBLWTH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BBEBB6	TBLWTH.B [W6++], [++W7]
0000	000000	NOP
0000	000000	NOP
0000	BB1BB6	TBLWTL [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BB0BB6	TBLWTL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP
0000	BBDBB6	TBLWTH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BBEBB6	TBLWTH.B [W6++], [++W7]
0000	000000	NOP
0000	000000	NOP
0000	BB1BB6	TBLWTL [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
第 6 步：将第 4 步和第 5 步重复执行 32 次，将 128 个指令装入写锁存器。		

表 5-3： 烧写编程执行程序 （行写）（续）

命令 （二进制）	数据 （十六进制）	说明
第 7 步：设置 NVMADRU/NVMADR 寄存器对以指向正确的地址。		
0000	2xxxx3	MOV #DestinationAddress<15:0>, W3
0000	2xxxx4	MOV #DestinationAddress<23:16>, W4
0000	883B13	MOV W3, NVMADR
0000	883B24	MOV W4, NVMADRU
第 8 步：执行 WR 位解锁序列并启动写周期。		
0000	200550	MOV #0x55, W0
0000	883B30	MOV W0, NVMKEY
0000	200AA0	MOV #0xAA, W0
0000	883B30	MOV W0, NVMKEY
0000	A8E761	BSET NVMCON, #WR
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
0000	000000	NOP
第 9 步：重复该步骤以轮询 WR 位，直到它被硬件清零为止。		
0000	040200	GOTO 0x200
0000	000000	NOP
0000	803B00	MOV NVMCON, W2
0000	A8E761	MOV W2, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
第 10 步：复位器件的内部程序计数器。		
0000	040200	GOTO 0x200
0000	000000	NOP
第 11 步：重复执行第 3 步至第 9 步，直到所有执行程序存储区都被编程为止。		
第 12 步：清零 WREN 位。		
0000	200000	MOV #0000, W0
0000	883B00	MOV W0, NVMCON

5.4 读执行程序存储区

读执行程序存储区是通过执行一系列TBLRD指令和使用REGOUT 命令随时钟移出数据完成的。

表 5-4 给出了读执行程序存储区的 ICSP 编程的详细步骤。

为了使读时间最短，将使用与 PE 相同的数据打包格式。关于数据打包格式的更多详细信息，请参见**第 6.2 节“编程执行程序命令”**。

表 5-4： 读执行程序存储区的串行指令执行

命令 （二进制）	数据 （十六进制）	说明
第 1 步：退出复位向量。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 2 步：为执行 TBLRD 指令初始化 TBLPAG 寄存器和读指针 （W6）。		
0000	200xx0	MOV #<SourceAddress23:16>, W0
0000	8802A0	MOV W0, TBLPAG
0000	2xxxx6	MOV #<SourceAddress15:0>, W6
第 3 步：初始化写指针 （W7）并将执行程序存储区的接下来 4 个存储单元内容存储到 W0:W5。		
0000	EB0380	CLR W7
0000	000000	NOP
0000	BA1B96	TBLRDL [W6], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BADBB6	TBLRDH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BADBD6	TBLRDH.B [++W6], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BA1BB6	TBLRDL [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BA1B96	TBLRDL [W6], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BADBB6	TBLRDH.B [W6++], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BADBD6	TBLRDH.B [++W6], [W7++]
0000	000000	NOP
0000	000000	NOP
0000	BA0BB6	TBLRDL [W6++], [W7]
0000	000000	NOP
0000	000000	NOP

表 5-4： 读执行程序存储区的串行指令执行 （续）

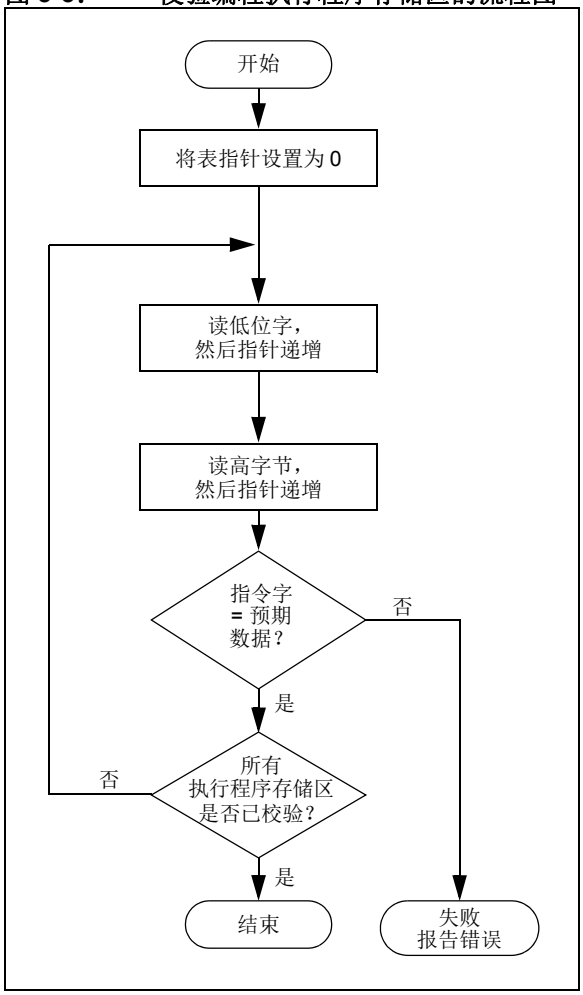
命令 （二进制）	数据 （十六进制）	说明
第 4 步：使用 VISI 寄存器和 REGOUT 命令输出 W0:W5 的内容。		
0000	883C20	MOV W0, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	883C21	MOV W1, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	883C22	MOV W2, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	883C23	MOV W3, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	883C24	MOV W4, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
0000	883C25	MOV W5, VISI
0000	000000	NOP
0001	<VISI>	; Clock out the contents of the VISI register.
0000	000000	NOP
第 5 步：复位器件的内部程序计数器。		
0000	000000	NOP
0000	040200	GOTO 0x200
0000	000000	NOP
第 6 步：重复第 3 步至第 5 步，直到读取完所需的所有执行程序存储区为止 （请注意，“复位器件的内部程序计数器”将作为第 5 步）。		

5.5 校验编程执行程序

校验的步骤涉及读回执行程序存储空间并将读到的值与存储在编程器缓冲区中的副本作比较。

校验过程如图 5-3 所述。读取指令的低位字，然后读取高位字的低字节，并将其与编程器缓冲区中存储的指令相比较。关于读取执行程序存储区的实现细节，请参见**第 5.4 节 “读执行程序存储区”**。

图 5-3： 校验编程执行程序存储区的流程图



6.0 编程执行程序

6.1 编程执行程序通信

编程器和PE存在主从关系，其中编程器是主编程设备，而 PE 是从设备。

所有通信都是由编程器以命令形式发起的。每次只能将一条命令发送给 PE。而 PE 在接收到命令并对其进行处理后，将只发送一个响应给编程器。**第 6.2 节 “编程执行程序命令”**中介绍了 PE 命令集。**第 6.3 节 “编程执行程序响应”**中介绍了响应集。

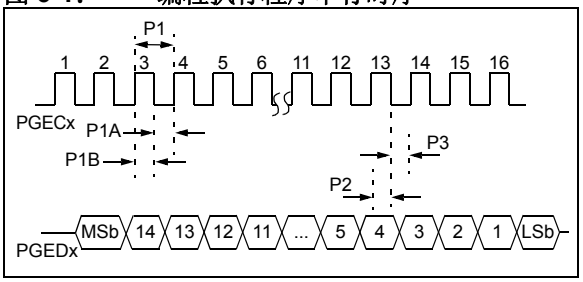
6.1.1 通信接口和协议

ICSP/ 增强型 ICSP 接口是使用 PGECx 和 PGEDx 引脚实现的双线 SPI 接口。PGECx 引脚用作时钟输入引脚，而时钟源必须由编程器提供。PGEDx 引脚用于向 PE 发送命令数据，以及从其接收响应数据。

注： 对于增强型ICSP，所有串行数据都在PGECx的下降沿发送，在 PGECx 的上升沿锁存。所有数据发送都使用 16 位模式，首先发送最高有效位 （见图 6-1）。

由于使用双线 SPI 接口，且数据发送是双向的，因而可使用一个简单的协议来控制 PGEDx 的方向。当编程器完成命令发送时，它会释放 PGEDx 线，并允许 PE 将此线驱动为高电平。PE 将 PGEDx 线保持为高电平以指示它正在处理命令。

图 6-1： 编程执行程序串行时序



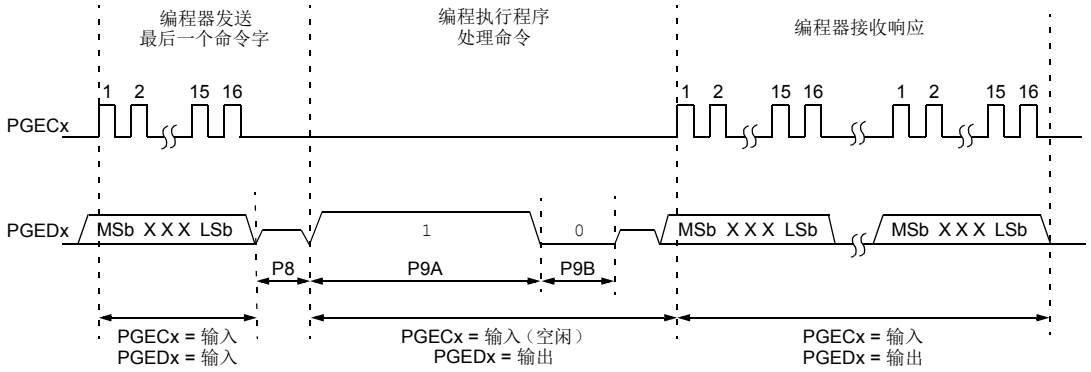
PE 处理完命令之后，会将 PGEDx 拉为低电平 （P9B）以通知编程器可以在时钟控制下接收响应。编程器将在最长等待时间 （P9B）后开始接收响应，并且它必须提供必需的时钟脉冲数以从 PE 接收整个响应。

一旦接收到整个响应，编程器应该终止 PGECx 上的时钟直到向 PE 发送另一条命令为止。该协议如图 6-2 所示。

6.1.2 SPI 速率

在增强型 ICSP 模式下， PIC24FJ256GA705 系列器件使用内部快速 RC （FRC）振荡器作为时钟源工作，其标称频率为 8 MHz。该振荡器频率产生 4 MHz 的有效系统时钟频率。为了确保编程器的时钟速度不至于太快，建议由编程器提供 2 MHz 的时钟。

图 6-2： 编程执行程序 —— 编程器通信协议



注 1： 关于时序信息，请参见表 9-1。

6.1.3 超时

在向编程器发送响应的过程中，PE 不使用看门狗定时器或超时。如果编程器不遵守**第 6.1.1 节 “通信接口和协议”**中所述的使用 PGECx 的流控制机制，在尝试向编程器发送响应时，PE 可能会表现异常。由于 PE 没有超时，所以编程器必须正确地遵守所述的通信协议。

作为安全措施，编程器应该使用表 6-1 中标识的命令超时。如果命令超时结束，编程器应该将 PE 复位并再次开始对器件编程。

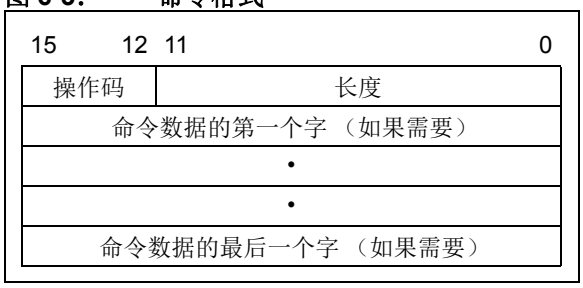
6.2 编程执行程序命令

表 6-1 列出了 PE 命令集。该表包含每条命令的操作码、助记符、长度、超时和说明。在命令说明（见**第 6.2.4 节 “命令说明”**）中详细说明了每条命令的功能。

6.2.1 命令格式

所有 PE 命令都具有由 16 位头和命令所需的所有数据组成的通用格式（见图 6-3）。16 位头由用于标识命令的 4 位操作码字段和随后的 12 位命令长度字段组成。

图 6-3： 命令格式



命令操作码必须与命令集中的一条命令匹配。接收到任何与表 6-1 中列出的命令不匹配的命令时将返回 “NACK” 响应 （见**第 6.3.1.1 节 “操作码字段”**）。

由于 SPI 在 16 位模式下工作，所以命令长度以 16 位字表示。 PE 使用命令长度字段来确定要从 SPI 端口读取的字数。如果该字段的值不正确， PE 将无法正确地接收命令。

表 6-1： 编程执行程序命令集

操作码	助记符	长度 （16 位字）	超时	说明
0x0	SCHECK	1	1 ms	工作状态检查。
0x1	READC	3	1 ms	从指定配置寄存器或器件 ID 寄存器中读取一个 8 位字。
0x2	READP	4	1 ms/ 行	读取主闪存的 N 个 24 位指令字，从指定地址开始。
0x3	PROG2W	6	5 ms	对位于代码存储区指定地址处的双指令字进行编程并校验。
0x4	保留	N/A	N/A	该命令是保留的；它将返回 NACK。
0x5	PROGP	99	5 ms	对起始于程序存储器指定地址处的 128 个字进行编程并校验。
0x6	保留	N/A	N/A	该命令是保留的；它将返回 NACK。
0x7	ERASEB	1	125 ms	整片擦除器件。
0x8	保留	N/A	N/A	该命令是保留的；它将返回 NACK。
0x9	ERASEP	3	25 ms	擦除页的命令。
0xA	保留	N/A	N/A	该命令是保留的；它将返回 NACK。
0xB	QVER	1	1 ms	查询 PE 软件版本。
0xC	CRCP	5	1s	对指定的存储区范围执行 CRC-16。
0xD	保留	N/A	N/A	该命令是保留的；它将返回 NACK。
0xE	QBLANK	5	700 ms	查询代码存储区是否为空白。

6.2.2 数据打包格式

当通过 16 位 SPI 接口传输 24 位指令字时，会使用图 6-4 中所示的格式将这些指令字打包以节省空间。该格式最大限度地降低了通过 SPI 的通信量，并为 PE 提供了正确对齐以便执行表写操作的数据。

图 6-4： 指令字的打包格式

LSWx：指令字的低 16 位
MSBx：指令字的最高字节

注： 当传输的指令字数为奇数时，MSB2 为零，且不能发送 LSW2。

6.2.3 编程执行程序错误处理

PE 将不应答（NACK）所有不支持的命令。另外，由于 PE 存储空间的限制，将不会对包含在编程器命令中的数据执行校验。由编程器负责向 PE 发送带有有效命令参数的命令，否则可能导致编程操作失败。关于错误处理的更多信息，请参见**第 6.3.1.3 节 “QE_Code 字段”**。

6.2.4 命令说明

第 6.2.4.1 节 “SCHECK 命令”至**第 6.2.4.10 节 “QBLANK 命令”**介绍了 PE 支持的所有命令。

6.2.4.1 SCHECK 命令

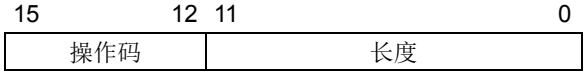


表 6-2 给出了 SCHECK 命令的说明。

表 6-2： 命令说明

字段	说明
操作码	0x0
长度	0x1

SCHECK 命令指示 PE 仅产生响应而不执行其他操作。该命令用作“工作状态检查”以验证 PE 是否正常工作。

预期的响应（双字）：

0x1000 0x0002

注： 该指令不是编程必需的，仅供开发使用。

6.2.4.2 READC 命令

15 12	11 8	7 0
操作码	长度	
N		Addr_MSB
Addr_LS		

表 6-3 给出了 READC 命令的说明。

表 6-3： 命令说明

字段	说明
操作码	0x1
长度	0x3
N	要读取的 8 位配置寄存器或器件 ID 寄存器的数量（最大为 256）
Addr_MSB	24 位源地址的 MSB
Addr_LS	24 位源地址的低 16 位

READC 命令指示 PE 从由 Addr_MSB 和 Addr_LS 指定的 24 位地址开始读取 N 个配置寄存器或器件 ID 寄存器。该命令只能用于读取 8 位或 16 位数据。

当使用该命令读取配置寄存器时，PE 返回的每个数据字的高字节为 0x00，低字节包含配置寄存器的值。

预期的响应（4 + 3 * (N - 1)/2 个字，其中 N 为奇数）：

0x1100

2 + N

配置寄存器或器件 ID 寄存器 1

...

配置寄存器或器件 ID 寄存器 N

注： 读取未实现的存储区将导致 PE 复位。为了防止发生这种情况，请确保只访问在特定器件中存在的存储单元。

6.2.4.3 READP 命令

15 12	11 8	7 0
操作码	长度	
N		
保留		Addr_MSB
Addr_LS		

表 6-4 给出了 READP 命令的说明。

表 6-4： 命令说明

字段	说明
操作码	0x2
长度	0x4
N	要读取的 24 位指令数（最大为 32768）
保留	0x0
Addr_MSB	24 位源地址的 MSB
Addr_LS	24 位源地址的低 16 位

READP 命令指示 PE 从由 Addr_MSB 和 Addr_LS 指定的 24 位地址开始读取代码存储区的 N 个 24 位字。该命令只能用于读取 24 位数据。作为对该命令的响应返回的所有数据使用**第 6.2.2 节“数据打包格式”**中所述的数据打包格式。

预期的响应（2 + 3 * N/2 个字，其中 N 为偶数）：

0x1200

2 + 3 * N/2

最低有效程序存储字 1

...

最低有效程序存储字 N

预期的响应（4 + 3 * (N - 1)/2 个字，其中 N 为奇数）：

0x1200

4 + 3 * (N – 1)/2

最低有效程序存储字 1

...

程序存储字 N 的 MSB（补零）

注： 读取未实现的存储区将导致 PE 复位。为了防止发生这种情况，请确保只访问在特定器件中存在的存储单元。

6.2.4.4 PROG2W 命令

15 12	11 8	7 0
操作码	长度	
保留		Addr_MSB
Addr_LS		
DataL_LS		
DataH_MSB		DataL_MSB
DataH_LS		

表 6-5 给出了 PROG2W 命令的说明。

表 6-5： 命令说明

字段	说明
操作码	0x3
长度	0x6
DataL_MSB	低指令字 24 位数据的 MSB
DataH_MSB	高指令字 24 位数据的 MSB
Addr_MSB	24 位目标地址的 MSB
Addr_LS	24 位目标地址的低 16 位
DataL_LS	低指令字 24 位数据的低 16 位
DataH_LS	高指令字 24 位数据的低 16 位

PROG2W 命令指示 PE 对位于指定存储地址的代码存储区的两个指令字 （6 个字节）进行编程。

当指令字被编程到代码存储区后， PE 将对照命令中的数据对已编程的数据进行校验。

预期的响应 （双字）：

0x1300

0x0002

6.2.4.5 PROGP 命令

15 12	11 8	7 0
操作码	长度	
保留		Addr_MSB
Addr_LS		
D_1		
D_2		
...		
D_N		

表 6-6 给出了 PROGP 命令的说明。

表 6-6： 命令说明

字段	说明
操作码	0x5
长度	0x63
保留	0x0
Addr_MSB	24 位目标地址的 MSB
Addr_LS	24 位目标地址的低 16 位
D_1	16 位数据字 1
D_2	16 位数据字 2
...	16 位数据字 3 至 95
D_96	16 位数据字 96

PROGP 命令指示 PE 对位于指定存储地址的代码存储区的一行 （128 个指令字）进行编程。编程从命令中指定的行地址开始。目标地址应该是 0x100 的倍数。

要编程到存储区中的数据位于命令字 D_1 至 D_96 中，且必须使用图 6-4 所示的指令字打包格式对它们进行打包。

在所有数据被编程到代码存储区之后， PE 将对照命令中的数据对已编程的数据进行校验。

预期的响应 （双字）：

0x1500

0x0002

注： 关于代码存储区大小的信息，请参见表 2-2。

6.2.4.6 ERASEB 命令

15 12	11 8	7 0
操作码		长度

表 6-7 给出了 ERASEB 命令的说明。

表 6-7： 命令说明

字段	说明
操作码	0x7
长度	0x1

ERASEB 命令指示 PE 执行整片擦除 （即，擦除所有主闪存和代码保护位）。

预期的响应 （双字）：

0x1700

0x0002

6.2.4.7 ERASEP 命令

15 12	11 8	7 0
操作码	长度	
NUM_PAGES	Addr_MSB	
Addr_LS		

表 6-8 给出了 ERASEP 命令的说明。

表 6-8： 命令说明

字段	说明
操作码	0x9
长度	0x3
NUM_PAGES	最多 255 页
Addr_MSB	24 位地址的最高有效字节
Addr_LS	24 位地址的低 16 位

ERASEP 命令指示 PE 对代码存储区执行页擦除 （擦除 [NUM_PAGES] 页）。代码存储区必须按偶数个 1024 指令字的地址边界擦除。

预期的响应 （双字）：

0x1900

0x0002

6.2.4.8 QVER 命令

15 12	11 0
操作码	长度

表 6-9 给出了 QVER 命令的说明。

表 6-9： 命令说明

字段	说明
操作码	0xB
长度	0x1

QVER 命令查询存储在测试存储区中的 PE 软件的版本。在响应的 QE_Code 中返回 “版本 . 修订版” 信息，使用一个以下格式的字节：主版本由高 4 位表示，修订版本由低 4 位表示 （例如， 0x23 表示 PE 软件的版本为 2.3）。

预期的响应 （双字）：

0x1BMN （其中 “MN” 代表版本 M.N）

0x0002

6.2.4.9 CRCP 命令

15 12	11 8	7 0
操作码	长度	
保留		Addr_MSB
Addr_LSW		
保留		Size_MSB
Size_LSW		

表 6-10 给出了 CRCP 命令的说明。

表 6-10： 命令说明

字段	说明
操作码	0xC
长度	0x5
Addr_MSB	24 位地址的最高有效字节
Addr_LSW	24 位地址的低 16 位
大小	24 位存储单元的数量 （地址范围除以 2）

CRCP 命令对指定的存储区范围执行 CRC-16。该命令可以代替整片校验。数据以图 6-4 中所示的打包方法按字节进行移位，先移位最低有效字节（Least Significant Byte，LSB）。

示例：

测试数据“123456789”的CRC-CCITT-16将为0x29B1

预期的响应 （3 字）：

QE_Code：0x1C00

长度：0x0003

CRC 值：0xXXXX

6.2.4.10 QBLANK 命令

15 12	11 0
操作码	长度
保留	Size_MSB
Size_LSW	
保留	Addr_MSB
Addr_LSW	

表 6-11 给出了 QBLANK 命令的说明。

表 6-11： 命令说明

字段	说明
操作码	0xE
长度	0x5
大小	要检查的程序存储区的长度（用 24 位字数表示） + Addr_MS
Addr_MSB	24 位地址的最高有效字节
Addr_LSW	24 位地址的低 16 位

QBLANK 命令查询 PE 以确定代码存储区的内容是否为空白 （包含全 1）。必须在命令中指定要检查的代码存储区的大小。

代码存储区的空白检查始于 [Addr]，并朝着地址更大的方向检查指定数目的指令字。

如果指定的代码存储区为空白， QBLANK 将返回 QE_Code 0xF0 ； 否则， QBLANK 将返回 QE_Code 0x0F。

预期的响应 （空白器件的双字）：

0x1DF0

0x0002

预期的响应 （非空白器件的双字）：

0x1D0F

0x0002

注： QBLANK 命令不会对系统操作配置位进行检查，因为这些位在执行整片擦除时不会置为 1。

6.3 编程执行程序响应

每接收到一条命令，PE 就会向编程器发送一个响应。该响应指示命令是否已被正确处理，其中包含所有必需的响应数据或错误数据。

表 6-12 列出了 PE 响应集。该表包含每个响应的操作码、助记符和说明。**第 6.3.1 节 “响应格式”**中介绍了响应格式。

表 6-12： 编程执行程序响应集

操作码	助记符	说明
0x1	PASS	命令已成功处理。
0x2	FAIL	命令未成功处理。
0x3	NACK	命令未知。

6.3.1 响应格式

所有 PE 响应都具有由双字响应头和该命令所需的所有数据组成的通用格式。

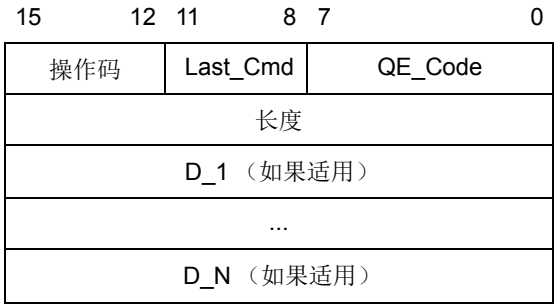


表 6-13 给出了响应格式的说明。

表 6-13： 响应格式说明

字段	说明
操作码	响应操作码。
Last_Cmd	产生该响应的编程器命令。
QE_Code	查询代码或错误代码。
长度	以 16 位字数表示的响应长度 （包含双字响应头）。
D_1	第一个 16 位数据字 （如果适用）。
D_N	最后一个 16 位数据字 （如果适用）。

6.3.1.1 操作码字段

操作码是响应的第一个字中的一个 4 位字段。操作码指示命令的处理情况（见表 6-12）。如果命令已成功处理，响应操作码是 PASS。如果在处理命令过程中发生错误，则响应操作码是 FAIL，而且 QE_Code 将指示失败的原因。如果发送给 PE 的命令未被识别，则 PE 将返回 NACK 响应。

6.3.1.2 Last_Cmd 字段

Last_Cmd 是响应的第一个字中的一个 4 位字段，该字段指示 PE 处理的命令。由于 PE 每次只能处理一条命令，所以从技术角度而言不需要此字段。但是，它可以用来验证 PE 是否正确地接收了编程器发送的命令。

6.3.1.3 QE_Code 字段

QE_Code 是响应的第一个字中的一个字节。该字节用于为查询命令返回数据，为所有其他命令返回错误代码。

当 PE 处理两种查询命令 （QBLANK 或 QVER）之一时，返回的操作码总是 PASS，并且 QE_Code 将保存查询响应数据。表 6-14 给出了这两种查询的 QE_Code 的格式。

表 6-14： 针对查询命令的 QE_Code

查询	QE_Code
QBLANK	0x0F = 代码存储区非空白 0xF0 = 代码存储区空白
QVER	0xMN，其中 PE 软件版本 = M.N （例如， 0x32 表示软件版本为 3.2）

当 PE 处理除查询外的任何命令时，QE_Code 代表一个错误代码。表 6-15 给出了支持的错误代码。如果命令已成功处理，返回的 QE_Code 将设置为 0x0，指示在处理命令过程中没有发生错误。如果对 PROGW 命令的编程进行校验时失败，则 QE_Code 将设置为 0x1。对于所有其他 PE 错误， QE_Code 为 0x2。

表 6-15： 针对非查询命令的 QE_Code

QE_Code	说明
0x0	无错误
0x1	校验失败
0x2	其他错误

6.3.1.4 响应长度

响应长度指示 PE 的响应的长度，以 16 位字数表示。该字段包含双字响应头。

除了对读命令的响应外，对其他命令的响应的长度都只有两个字。

对 READP 命令的响应使用**第 6.2.2 节“数据打包格式”**中所述的指令字打包格式。当读取奇数个程序存储字（N 为奇数）时，对 READP 命令的响应是（3 * (N + 1)/2 + 2）个字。当读取偶数个程序存储字 （N 为偶数）时，对 READP 命令的响应是 （3 * N/2 + 2）个字。

7.0 器件 ID

器件 ID 存储区可用于确定芯片的型号和生产信息。该存储区只读，并且能在使能代码保护时读取。DEVID 寄存器（0xFF 0000）标识器件的特定部件编号，而 DEVREV（0xFF 0002）则给出硅片版本。

表 7-1 列出了每个器件的标识信息。表 7-2 给出了器件 ID 寄存器，表 7-3 介绍了每个寄存器的位域。

表 7-1： 器件 ID

器件	DEVID
PIC24FJ64GA702	0x7506
PIC24FJ128GA702	0x750A
PIC24FJ256GA702	0x750E
PIC24FJ64GA704	0x7505
PIC24FJ128GA704	0x7509
PIC24FJ256GA704	0x750D
PIC24FJ64GA705	0x7507
PIC24FJ128GA705	0x750B
PIC24FJ256GA705	0x750F

表 7-2： PIC24FJ256GA705 系列器件 ID 寄存器

地址	名称	Bit															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xFF 0000	DEVID	FAMID<7:0>								DEV<7:0>							
0xFF 0002	DEVREV	—												REV<3:0>			

表 7-3： 器件 ID 位域说明

位域	寄存器	说明
FAMID<7:0>	DEVID	对器件的系列 ID 进行编码。
DEV<7:0>	DEVID	对每款器件的 ID 进行编码。
REV<3:0>	DEVREV	对器件按顺序排列的 （数字）版本标识符进行编码。

7.1 器件唯一标识符 （UDID）

在最终制造过程中，会为所有 PIC24FJ256GA705 系列器件单独地编码一个器件唯一标识符 （Unique Device Identifier，UDID）。批量擦除命令或其他任何用户可使用的方法都无法擦除 UDID。在需要对 Microchip Technology 器件进行制造追踪的应用中，可以利用该功能来实现该目的。应用制造商也可以将它用于可能需要唯一标识的任意应用场合，例如：

- 追踪器件
- 唯一序列号
- 唯一安全密钥

UDID 包含 5 个 24 位程序字。这些字段组合在一起构成一个唯一的 120 位标识符。

UDID 存储在器件配置空间的 0x80 1600 和 0x80 1608 之间的 5 个只读单元中。表 7-4 列出了标识符字的地址，并给出了其内容。

表 7-4： UDID 地址

UDID	地址	说明
UDID1	0x80 1600	UDID 字 1
UDID2	0x80 1602	UDID 字 2
UDID3	0x80 1604	UDID 字 3
UDID4	0x80 1606	UDID 字 4
UDID5	0x80 1608	UDID 字 5

8.0 校验和计算

器件校验和大小为 16 位。通过对以下内容求和可以计算校验和：

- 代码存储单元的内容
- 配置字的内容

通过将每个存储器地址的所有三个字节相加来对所有存储单元（包括配置字）求和。校验和“按字节”计算，最终结果截短为 16 位。在 dsPIC33 架构中，每个闪存地址包含两个字节（如果为偶地址）或一个字节（如果为奇地址；因为最高字节已实现且始终为 0x00）。计算校验和时，给定地址的字的高字节和低字节都应作为单独字节（而不是一个 16 位字）加到正在运行的求和中。

例如，在包含两个字和如下内容的程序中：

地址 0x0000 处的内容 = 0xABCD

地址 0x0001 处的内容 = 0x00EF

0x0000:0x0001 区域的校验和为：0xCD + 0xAB + 0xEF + 0x00 = 0x0267。

同样，CFGB 块校验和是配置块地址区域的所有内容的“按字节”总和，其计算方式与 PROG 区域大致相同，区别在于前者的校验和计算过程中，某些配置寄存器可能需要将某些位通过逻辑与运算进行掩码和排除。

表 8-1： 配置位掩码

器件	配置位掩码	
	FSIGN	**FICD**
PIC24FJ256GA705	0xFF 7FFF	0xFF FFDF

表 8-2： 校验和计算

器件	读代码保护	校验和计算	擦除后的值	**0x0 和最后一个代码地址处为0xAAAAAA 时的值**
PIC24FJ256GA705	禁止	PROG[0x00:0x02 AEFF] + CFGB[0x02 AF00:0x02 AFFF]	0xF760(1)	0xF562(1)
	使能	0	0x0000	0x0000
PIC24FJ128GA705	禁止	PROG[0x00:0x01 5EFF] + CFGB[0x01 5F00:0x01 5FFF]	0xEF60(1)	0xED62(1)
	使能	0	0x0000	0x0000
PIC24FJ64GA705	禁止	PROG[0x00:0xAEFF] + CFGB[0xAF00:0xAFFF]	0xF760(1)	0xF562(1)
	使能	0	0x0000	0x0000

图注： PROG[a:b] = a 至 b （含 a 和 b）的所有存储单元的程序存储器字节和 （代码存储区的全部 3 个字节）
CFGB[c:d] = c 至 d （含 c 和 d）的所有存储单元的配置存储器字节和 （代码存储区的全部 3 个字节）

注 1： 对于校验和计算示例，配置位在擦除器件之后设置为默认配置值。

9.0 交流 / 直流特性和时序要求

表 9-1 列出了交流 / 直流特性和时序要求。

表 9-1： 交流 / 直流特性和时序要求

标准工作条件 工作温度：-40ºC 至 +85ºC。建议在 +25ºC 下编程。						
参数编号	**符号**	**特性**	**最小值**	**最大值**	**单位**	**条件**
D111	VDD	编程时的电源电压	2.0	3.6	V	请参见**注 1** 和 **2**
D113	IDDP	编程时的电源电流	—	8	mA	请参见**注 2**
D114	IPEAK	启动期间的瞬时峰值电流	—	—	mA	请参见**注 2**
D031	VIL	输入低电压	—	0.2 VDD	V	请参见**注 2**
D041	VIH	输入高电压	0.8 VDD	VDD	V	请参见**注 2**
D080	VOL	输出低电压	—	0.8	V	请参见**注 2**
D090	VOH	输出高电压	0.8 VDD	VDD	V	请参见**注 2**
D012	CIO	I/O 引脚 （PGEDx）上的容性负载	—	50	pF	请参见**注 2**
P1	TPGC	串行时钟 （PGECx）周期 （ICSP™）	200	—	ns	
P1	TPGC	串行时钟 （PGECx）周期 （增强型 ICSP）	500	—	ns	
P1A	TPGCL	串行时钟 （PGECx）的低电平时间 （ICSP）	80	—	ns	
P1A	TPGCL	串行时钟 （PGECx）的低电平时间 （增强型 ICSP）	200	—	ns	
P1B	TPGCH	串行时钟 （PGECx）的高电平时间 （ICSP）	80	—	ns	
P1B	TPGCH	串行时钟 （PGECx）的高电平时间 （增强型 ICSP）	200	—	ns	
P2	TSET1	输入数据到串行时钟 ↓ 的建立时间	15	—	ns	
P3	THLD1	在 PGECx ↓ 后输入数据的保持时间	15	—	ns	
P4	TDLY1	4 位命令和命令操作数之间的延时	40	—	ns	
P4A	TDLY1A	命令操作数和下一条 4 位命令之间的延时	40	—	ns	
P5	TDLY2	命令的最后一个 PGECx ↓ 到读取数据字时出现的第一个 PGECx ↑ 之间的延时	20	—	ns	
P6	TSET2	VDD ↑ 到 $\overline{\text{MCLR}}$ ↑ 的建立时间	100	—	ns	
P7	THLD2	在 $\overline{\text{MCLR}}$ ↑ 后输入数据的保持时间	50	—	ms	
P8	TDLY3	命令字节的最后一个 PGECx ↓ 到 PE 将 PGEDx 驱动为 ↑ 之间的延时	12	—	µs	

注 1： 在编程期间还必须将 VDD 加到 AVDD 引脚。 AVDD 和 AVSS 应该总是分别在 VDD±0.3V 和 VSS±0.3V 以内。

2： 时间取决于 FRC 精度和 FRC 振荡器调节寄存器的值。请参见具体器件数据手册的 **“电气特性”** 章节。

3： 该时间适用于程序存储字、配置字和用户 ID 字。

表 9-1： 交流 / 直流特性和时序要求 （续）

标准工作条件 工作温度：-40ºC 至 +85ºC。建议在 +25ºC 下编程。						
参数编号	**符号**	**特性**	**最小值**	**最大值**	**单位**	**条件**
P9A	TDLY4	PE 命令处理时间	10	—	µs	
P9B	TDLY5	PE 将 PGEDx 驱动为↓到 PE 释放 PGEDx 之间的延时	15	23	µs	
P10	TDLY6	编程后 PGECx 处于低电平的时间	400	—	ns	
P11	TDLY7	整片擦除时间	16	20	ms	
P12	TDLY8	页擦除时间	16	20	ms	请参见**注 2**
P13	TDLY9	双字编程时间	16	20	µs	请参见**注 2** 和 **3**
P14	TR	进入 ICSP 模式的 $\overline{\text{MCLR}}$ 上升时间	—	1.0	µs	
P15	TVALID	PGECx ↑后的数据输出有效时间	10	—	ns	
P16	TDLY10	最后一个 PGECx ↓和 $\overline{\text{MCLR}}$ ↓之间的延时	0	—	s	
P17	THLD3	$\overline{\text{MCLR}}$ ↓到 VDD ↓的时间	100	—	ns	
P18	TKEY1	从第一个 $\overline{\text{MCLR}}$ ↓到向 PGEDx 输入密钥序列的第一个 PGECx ↑之间的延时	1	—	ms	
P19	TKEY2	从向 PGEDx 输入密钥序列的最后一个 PGECx ↓到第二个 $\overline{\text{MCLR}}$ ↑之间的延时	25	—	ns	
P21	TMCLRH	$\overline{\text{MCLR}}$ 高电平时间	—	500	µs	

注 1： 在编程期间还必须将 VDD 加到 AVDD 引脚。 AVDD 和 AVSS 应该总是分别在 VDD±0.3V 和 VSS±0.3V 以内。

2： 时间取决于 FRC 精度和 FRC 振荡器调节寄存器的值。请参见具体器件数据手册的 **“电气特性”**章节。

3： 该时间适用于程序存储字、配置字和用户 ID 字。

附录 A： SPI 端口扫描软件程序

例 9-1： SPI 端口扫描软件代码示例

```
/*******************************************************************************
 * © 2016 Microchip Technology Inc.
 *
 * FileName: spi_search.s
 * Dependencies: none
 * Processor:PIC24
 * Compiler:XC16
 * IDE:MPLAB® X
 * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Description:This file performs a search of the correct
 * PGEC/PGED port and connect SPI2 slave to it.
 *
 *******************************************************************************/
/*******************************************************************************
 * MICROCHIP SOFTWARE NOTICE AND DISCLAIMER:You may use this software, and
 * any derivatives created by any person or entity by or on your behalf,
 * exclusively with Microchip's products in accordance with applicable
 * software license terms and conditions, a copy of which is provided for
 * your reference in accompanying documentation.Microchip and its licensors
 * retain all ownership and intellectual property rights in the
 * accompanying software and in all derivatives hereto.
 *
 * This software and any accompanying information is for suggestion only.
 * It does not modify Microchip's standard warranty for its products.You
 * agree that you are solely responsible for testing the software and
 * determining its suitability.Microchip has no obligation to modify,
 * test, certify, or support the software.
 *
 * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS".NO WARRANTIES, WHETHER
 * EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED
 * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE, ITS INTERACTION WITH
 * MICROCHIP'S PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY
 * APPLICATION.
 *
 * IN NO EVENT, WILL MICROCHIP BE LIABLE, WHETHER IN CONTRACT, WARRANTY,
 * TORT (INCLUDING NEGLIGENCE OR BREACH OF STATUTORY DUTY), STRICT
 * LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY INDIRECT,
 * SPECIAL, PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE,
 * FOR COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE,
 * HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY
 * OR THE DAMAGES ARE FORESEEABLE.TO THE FULLEST EXTENT ALLOWABLE BY LAW,
 * MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS
 * SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID
 * DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
 *
 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF
 * THESE TERMS.
 *******************************************************************************/
```

例 9-1： SPI 端口扫描软件代码示例 （续）

```
.include "xc.inc"

.global  _sd_Port_Search

.pushsection .text, code

;--------------------------------------------------------------------------------
; Constants
;--------------------------------------------------------------------------------

.equ  PPS_SPI2_DATA_SCK_SDI_INPUT, #RPINR22  ; PPS register for data and sck input

.equ  PPS_PGEC1_PGED1_INPUT, #0x0100
.equ  PPS_PGEC2_PGED2_INPUT, #0x0B0A
.equ  PPS_PGEC3_PGED3_INPUT, #0x0605

.equ  ANS_PGEC1_PGED1_REG, ANSB
.equ  ANS_PGEC2_PGED2_REG, ANSB
.equ  ANS_PGEC3_PGED3_REG, ANSB

.equ  ANS_PGEC1_PGED1_MASK, #0xFFFC
.equ  ANS_PGEC2_PGED2_MASK, #0xFFFF
.equ  ANS_PGEC3_PGED3_MASK, #0xFFFF

.equ  PORT_FOUND, #1
.equ  PORT_NOT_FOUND, #0

;--------------------------------------------------------------------------------
; Code
;--------------------------------------------------------------------------------

;--------------------------------------------------------------------------------
; Returns port number found on w0, 0 if no port found
; It will also leave SPI2 connected to the identified port
;--------------------------------------------------------------------------------
_sd_Port_Search:
    rcall PPS_Unlock                    ; Must unlock PPS to scan PGED/PGEC ports
    rcall SPI_Init                      ; SPI1 as master and SPI2 as slave for loopback test

    rcall ICSP_Scan1
    cp    w0, #PORT_FOUND
    bra   z, port_search_found

    rcall ICSP_Scan2
    cp    w0, #PORT_FOUND
    bra   z, port_search_found

    rcall ICSP_Scan3
    cp    w0, #PORT_FOUND
    bra   z, port_search_found

port_search_not_found:
    bclr  SPI1CON1L, #15                ; disable SPI1
    clr   w0                            ; return 0
    return

port_search_found:
    bclr  SPI1CON1L, #15                ; disable SPI1
    mov   w1, w0                        ; return port number
    return
```

例 9-1： SPI 端口扫描软件代码示例 （续）

```
;--------------------------------------------------------------------------------
PPS_Unlock:
    mov   #0, w0                        ; place zero in w0 to unlock PPS
    mov   #OSCCONL, w1                  ; OSCCONL (low byte) unlock sequence
    mov   #0x46, w2                     ; Preparing unlock sequence
    mov   #0x57, w3                     ; Preparing unlock sequence
    mov.b w2, [w1]                      ; Write 0x46
    mov.b w3, [w1]                      ; Write 0x9A
    mov.b w0, [w1]                      ; unlock PPS (IOLOCK bit = 0)
    return

;--------------------------------------------------------------------------------
; Initialize SPI1 as master and SPI2 as slave
; for loopback test on each programming port
;--------------------------------------------------------------------------------
SPI_Init:
    clr   SPI1BRGL                      ; clear SPI1BRGL
    clr   SPI1CON1H                     ; clear SPI1CON1H
    clr   SPI1CON1L                     ; clear SPI1CON1L
    mov   #0x8120, w0                   ; set SPIEN, CKE and MSTEN (master)
    mov   w0, SPI1CON1L

    clr   SPI2CON1H                     ; clear SPI2CON1H
    clr   SPI2CON1L                     ; clear SPI2CON1L
    mov   #0x0100, w0                   ; set CKE (slave)
    mov   w0, SPI2CON1L
    return

;--------------------------------------------------------------------------------
; Returns PORT_FOUND, or PORT_NOT_FOUND on w0
; Returns port number on w1
;--------------------------------------------------------------------------------
ICSP_Scan1:
    mov   #ANS_PGEC1_PGED1_MASK, w0     ; RB0 and RB1 are digital inputs for SCK and SDI
    mov   w0, ANS_PGEC1_PGED1_REG
    mov   #PPS_PGEC1_PGED1_INPUT, w0
    mov   w0, PPS_SPI2_DATA_SCK_SDI_INPUT

    rcall _sd_Loopback_Test

    mov   #1, w1
    return

;--------------------------------------------------------------------------------
ICSP_Scan2:
    mov   #ANS_PGEC2_PGED2_MASK, w0     ; RB0 and RB1 are digital inputs for SCK and SDI
    mov   w0, ANS_PGEC2_PGED2_REG
    mov   #PPS_PGEC2_PGED2_INPUT, w0
    mov   w0, PPS_SPI2_DATA_SCK_SDI_INPUT

    rcall _sd_Loopback_Test

    mov   #2, w1
    return
```

例 9-1： SPI 端口扫描软件代码示例 （续）

```
;--------------------------------------------------------------------------------
ICSP_Scan3:
    mov   #ANS_PGEC3_PGED3_MASK, w0      ; RB0 and RB1 are digital inputs for SCK and SDI
    mov   w0, ANS_PGEC3_PGED3_REG
    mov   #PPS_PGEC3_PGED3_INPUT, w0
    mov   w0, PPS_SPI2_DATA_SCK_SDI_INPUT

    rcall _sd_Loopback_Test

    mov   #3, w1
    return

;--------------------------------------------------------------------------------
; Returns PORT_FOUND, or PORT_NOT_FOUND on w0
;--------------------------------------------------------------------------------
_sd_Loopback_Test:

    bset  SPI2CON1L, #15                 ; enable SPI2

    mov   #0x00a5, w0                    ; 0xa5 pattern
    mov   w0, SPI1BUFL                   ; send

loopback_wait_data:
    btss  SPI1STATL, #0                  ; wait for the data
    bra   loopback_wait_data

    mov   SPI1BUFL, w1
    mov   SPI2BUFL, w1                   ; read data
    bclr  SPI2CON1L, #15                 ; disable SPI2

    cp    w0, w1
    bra   z, loopback_found

    mov   #PORT_NOT_FOUND, w0
    return

loopback_found:
    mov   #PORT_FOUND, w0
    return

.popsection
```

附录 B： 版本历史

版本 A（2015 年 7 月）

本编程规范的初始版本。

版本 B（2017 年 2 月）

替换了 **“引脚图（28 引脚 QFN）”**、**“引脚图（28 引脚 PDIP）”**、**“引脚图（44 引脚 TQFP/QFN）”** 和 **“引脚图（48 引脚 TQFP/QFN）”** 中的所有图。

对表 2-2 和表 2-3 进行了编辑。用新表替换了表 2-4。

对图 2-3 进行了编辑。

用新文本替换了**第 8.0 节 “校验和计算”**。

增加了**附录 A：“SPI 端口扫描软件程序”**。

更改了整篇文档的十六进制数据格式。

注：

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：978-1-5224-2678-3

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

中国 - 北京
Tel: 86-10-8569-7000

中国 - 成都
Tel: 86-28-8665-5511

中国 - 重庆
Tel: 86-23-8980-9588

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

中国 - 南京
Tel: 86-25-8473-2460

中国 - 青岛
Tel: 86-532-8502-7355

中国 - 上海
Tel: 86-21-3326-8000

中国 - 沈阳
Tel: 86-24-2334-2829

中国 - 深圳
Tel: 86-755-8864-2200

中国 - 苏州
Tel: 86-186-6233-1526

中国 - 武汉
Tel: 86-27-5980-5300

中国 - 西安
Tel: 86-29-8833-7252

中国 - 厦门
Tel: 86-592-238-8138

中国 - 香港特别行政区
Tel: 852-2943-5100

中国 - 珠海
Tel: 86-756-321-0040

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600

台湾地区 - 新竹
Tel: 886-3-577-8366

亚太地区

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

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

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

印度 India - Pune
Tel: 91-20-4121-0141

日本 Japan - Osaka
Tel: 81-6-6152-7160

日本 Japan - Tokyo
Tel: 81-3-6880-3770

韩国 Korea - Daegu
Tel: 82-53-744-4301

韩国 Korea - Seoul
Tel: 82-2-554-7200

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

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

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

新加坡 Singapore
Tel: 65-6334-8870

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

越南 Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

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

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