# PIC24FJXXXGA2/GB2 系列闪存编程规范

## 1.0 器件概述

本文档定义了PIC24FJXXXGA2/GB2系列单片机器件的编程规范。本编程规范仅供为PIC24FJXXXGA2/GB2 器件开发编程支持的人员使用。仅使用这些器件中一款器件的客户应使用已为器件编程提供支持的开发工具。

本编程规范特定于以下器件：

- PIC24FJ128GA204
- PIC24FJ128GA202
- PIC24FJ128GB204
- PIC24FJ128GB202
- PIC24FJ64GA204
- PIC24FJ64GA202
- PIC24FJ64GB204
- PIC24FJ64GB202

## 2.0 PIC24FJXXXGA2/GB2 器件编程概述

在本编程规范中将讨论两种对 PIC24FJXXXGA2/GB2 器件编程的方法。它们是：

- 在线串行编程 （ICSP™）
- 增强型在线串行编程 （增强型 ICSP）

ICSP 编程方法是对器件编程最直接的方法，但也是两种方法中较慢的一种。它提供固有的低级编程功能来擦除、编程和校验芯片。

增强型在线串行编程 （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）。编程执行程序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 PIC24FJXXXGA2/GB2 MCU 的编程无需处理芯片的低级编程协议。

本规范划分为几个主要部分，独立地介绍编程方法。**第 3.0 节 “ 器件编程 ——ICSP”** 介绍了在线串行编程方法。**第 4.0 节 “ 器件编程 —— 增强型 ICSP”** 介绍了增强型在线串行编程方法。

**注：** 在本规范涵盖的器件中，特殊功能寄存器 TBLPAG 的地址已从 32h 移到 54h。

如果基于来自其他器件系列的旧式编程规范代码来实现PIC24FJXXXGA2/GB2器件的编程规范，则必须小心确保将所有现有代码中对 TBLPAG 和 NVMCON 的所有引用更新为助记符和操作数正确的操作码十六进制数据。

**PIC24FJXXXGA2/GB2 器件：**

| 命令（二进制） | 数据（十六进制） | 说明 |
|---|---|---|
| 0000 | 8802A0 | MOV W0, TBLPAG |

**先前的 PIC24F 系列：**

| 命令（二进制） | 数据（十六进制） | 说明 |
|---|---|---|
| 0000 | 880190 | MOV W0, TBLPAG |

**图 2-1：　　增强型ICSP™ 编程系统概览**

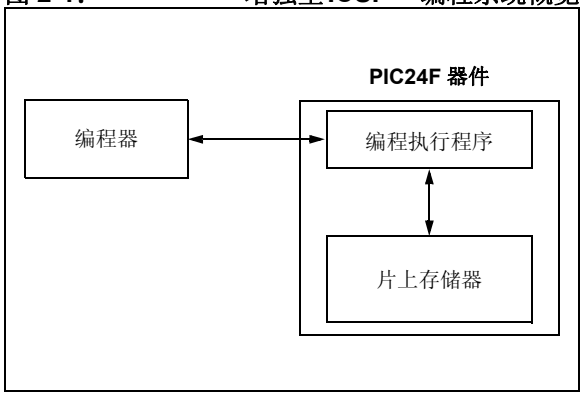
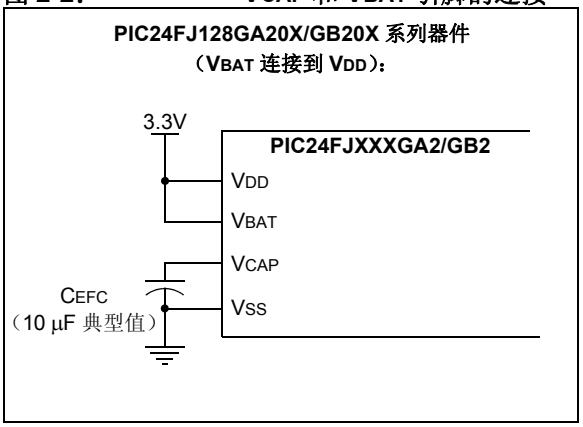



**图 2-2：　　VCAP 和 VBAT 引脚的连接**




## 2.1　　电源要求

所有 PIC24FJXXXGA2/GB2 器件使用标称值为 1.8V 的电压为其内核数字逻辑供电。为简化系统设计，PIC24FJXXXGA2/GB2 系列中的所有器件均包含一个片上稳压器，可使器件内核逻辑在 VDD 下工作。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 7.0 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。

PIC24FJ128GA204/GB204 系列中的器件还具有一个辅助电压源引脚（VBAT），使器件可以在移除 VDD 之后继续工作。在编程期间，建议总是通过 VDD 或备用电池源为 VBAT 引脚提供电压。

## 2.2　　程序存储器写 / 擦除要求

为了使 PIC24FJXXXGA2/GB2 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：在擦除将要写入数据的存储页之前，对该存储页中任何给定字的写操作次数不得超过两次。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。本规范中指定的编程方法就符合这一要求。

| **注：** | 不建议对同一个存储单元进行多次写操作而不进行擦除操作。 |
|---|---|

**表 2-1：　　引脚说明（编程期间）**

| 引脚名称 | 编程期间 | | |
|---|---|---|---|
| | 引脚名称 | 引脚类型 | 引脚说明 |
| MCLR | MCLR | P | 编程使能 |
| VDD，AVDD(1) | VDD | P | 电源 |
| VSS，AVSS(1) | VSS | P | 接地 |
| VBAT(2) | VBAT | P | VBAT 模式电源 |
| VCAP | VCAP | P | 送至内核的片上稳压器输出 |
| PGECx | PGECx | I | 编程引脚对 1、2 和 3：串行时钟 |
| PGEDx | PGEDx | I/O | 编程引脚对 1、2 和 3：串行数据 |
| VUSB3V3(3) | VUSB3V3 | P | USB 收发器电源输入电压（3.3V 标称值） |

**图注：**　I = 输入，O = 输出，P = 电源

**1：**　所有电源和地引脚都必须连接，包括模拟电源和地（实现了 AVDD/AVSS）。

**2：**　在编程期间，建议将 VBAT 引脚与电池或 VDD 连接。

**3：**　该引脚仅在 PIC24FJXXXGB2XX 系列器件中可用。


© 2014 Microchip Technology Inc.

## 2.3 引脚图

图 2-3 至图 2-6 给出了 PIC24FJXXXGA2/GB2 器件的引脚图。表 2-1 列出了进行编程所需的引脚，图中以粗体形式显示了这些引脚。关于完整的引脚说明，请参见具体器件的数据手册。

### 2.3.1 PGECx 和 PGEDx 引脚对

所有 PIC24FJXXXGA2/GB2 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 方法可以使用这些引脚对中的任意一个来进行器件编程。与电压源和地引脚不同，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。

**图 2-3： 引脚图 （28 引脚 PDIP/SSOP/SOIC）**

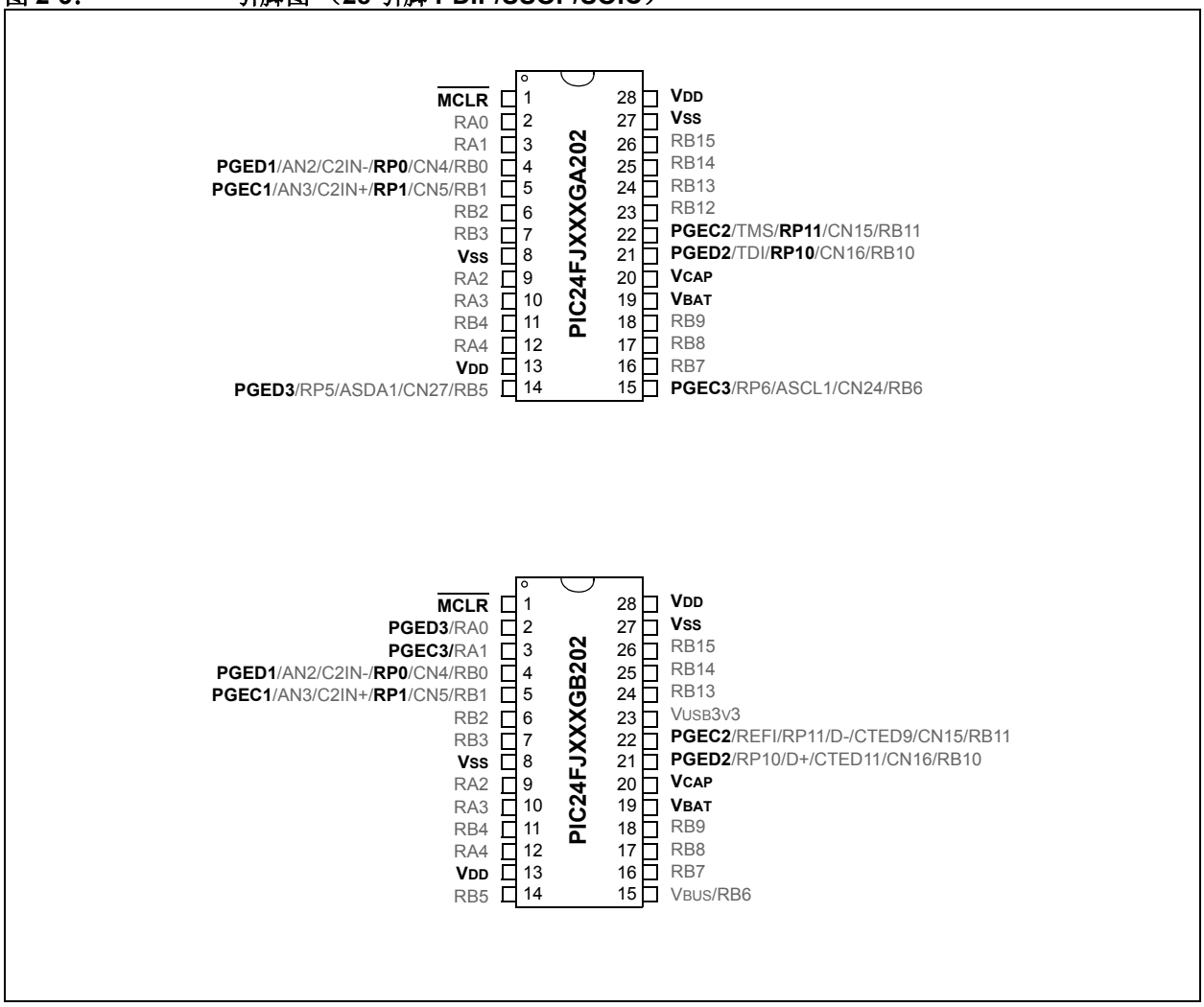
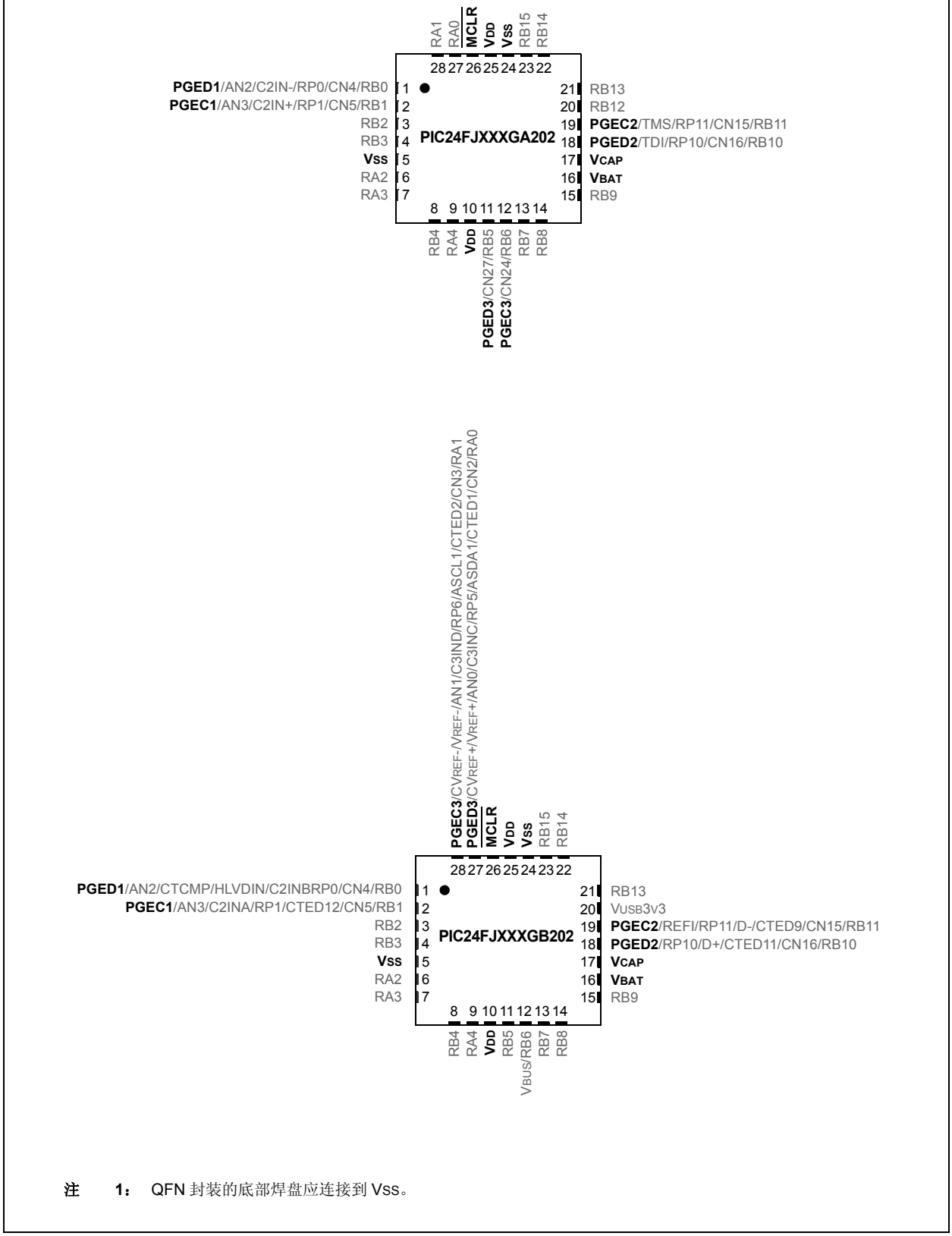


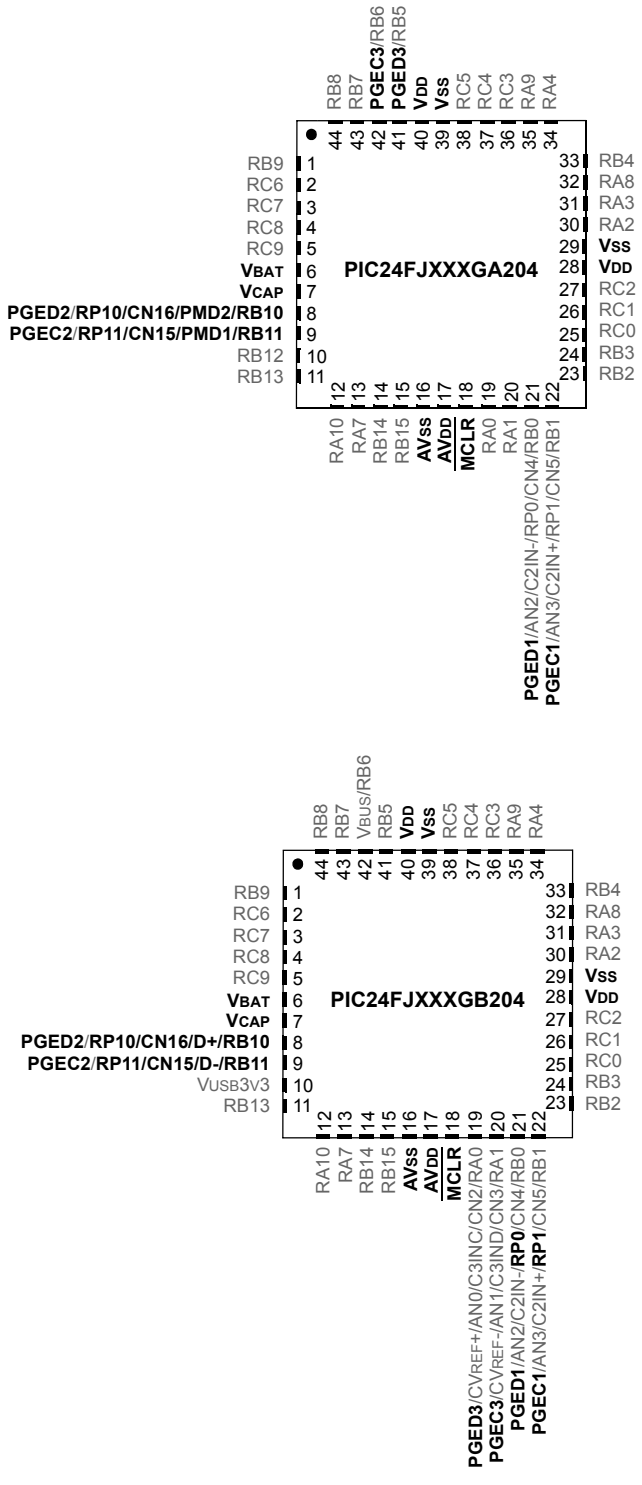
**图 2-4：　　引脚图 （28 引脚 QFN）(1)**




**注　　1：** QFN 封装的底部焊盘应连接到 VSS。

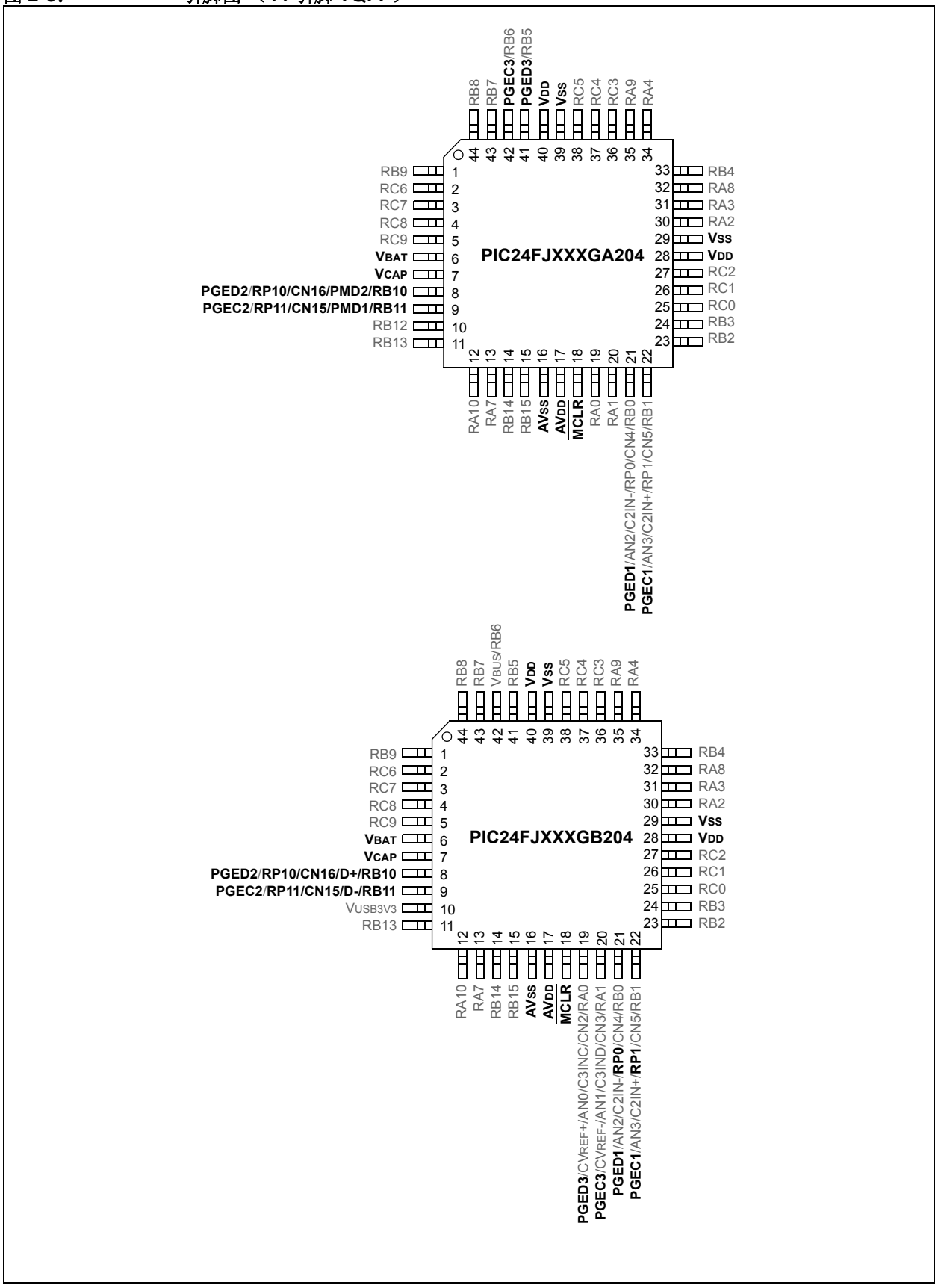

© 2014 Microchip Technology Inc.

**图 2-5：　　引脚图 （44 引脚 QFN）(1)**




**注　　1：** QFN 封装的底部焊盘应连接到 VSS。

**图 2-6：　　引脚图 （44 引脚 TQFP）**





© 2014 Microchip Technology Inc.

## 2.4 存储器映射

程序存储器映射从 000000h 延伸到 FFFFFEh。代码存储在存储器映射的基址部分，支持最多 87K 个指令字（约 256 KB）。表 2-2 给出了每款器件中程序存储器的容量以及擦除和编程块的数量。每个擦除块或页包含 512 个指令字，而每个编程块或行包含 64 个指令字。

地址从 800000h 至 8007FEh 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和调试执行程序。编程执行程序用于器件编程，而调试执行程序用于在线调试。存储器中的该区域不能用于存储用户代码。

所实现程序存储器的最后 4 个单元保留用作闪存配置字。表 2-2 列出了保留的地址。

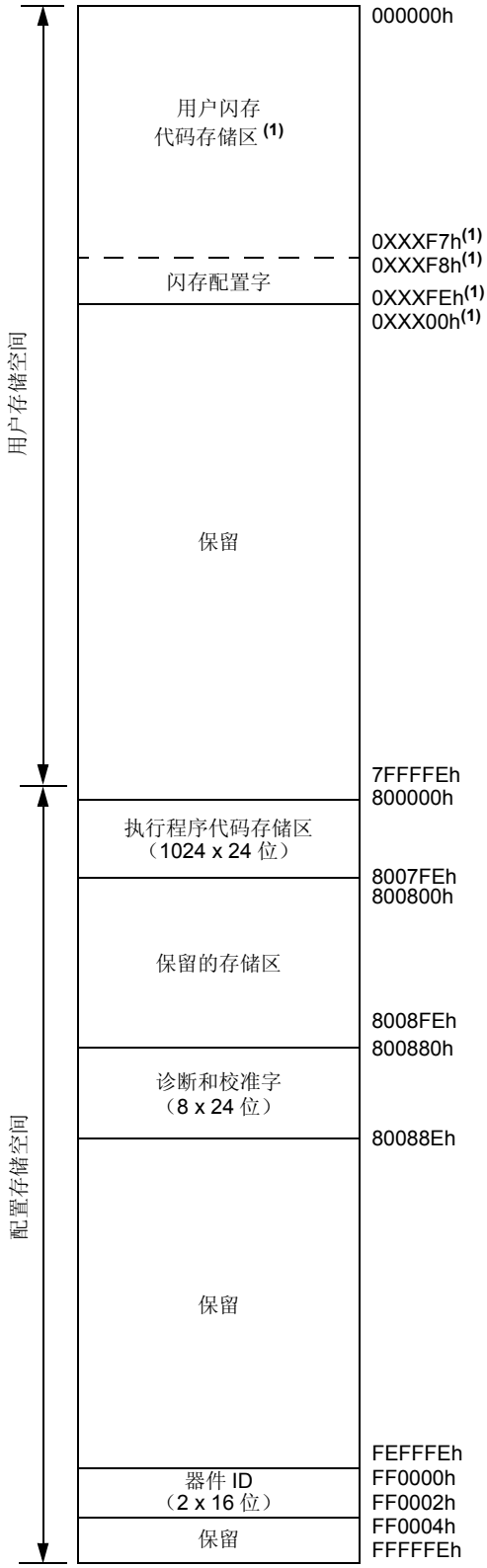
地址为 FF0000h 和 FF0002h 的两个存储单元保留用作器件 ID 寄存器。编程人员可使用这些位标识要编程的器件的类型。这将在**第 6.1 节 “器件 ID”** 中进行介绍。即使应用了代码保护，也可正常读出器件 ID 寄存器。

图 2-7 显示了 PIC24FJXXXGA2/GB2 器件的存储器映射。

**表 2-2： PIC24FJXXXGA2/GB2 器件的代码存储区容量和闪存配置字单元**

| 器件 | 用户存储器地址限制（指令字） | 写块 | 擦除块 | 配置字地址 | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 |
| PIC24FJ64GA2XX | 00ABFEh （22K） | 344 | 43 | 00ABFEh | 00ABFCh | 00ABFAh | 00ABF8h |
| PIC24FJ64GB2XX | | | | | | | |
| PIC24FJ128GA2XX | 0157FEh （44K） | 688 | 86 | 0157FEh | 0157FCh | 0157FAh | 0157F8h |
| PIC24FJ128GB2XX | | | | | | | |

**图 2-7：　　程序存储器映射**




**注　　1：** 用户闪存代码存储区的容量和地址边界随器件变化。详情请参见表 2-2。


© 2014 Microchip Technology Inc.

## 3.0 器件编程——ICSP

ICSP 模式是允许您对 PIC24FJXXXGA2/GB2 器件的存储器进行读写操作的一种特殊编程协议。 ICSP 模式是对器件编程采用的最直接的方法，但要注意增强型 ICSP 速度更快。ICSP 模式还具有读取执行程序存储区的内容以确定编程执行程序是否存在的功能。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。

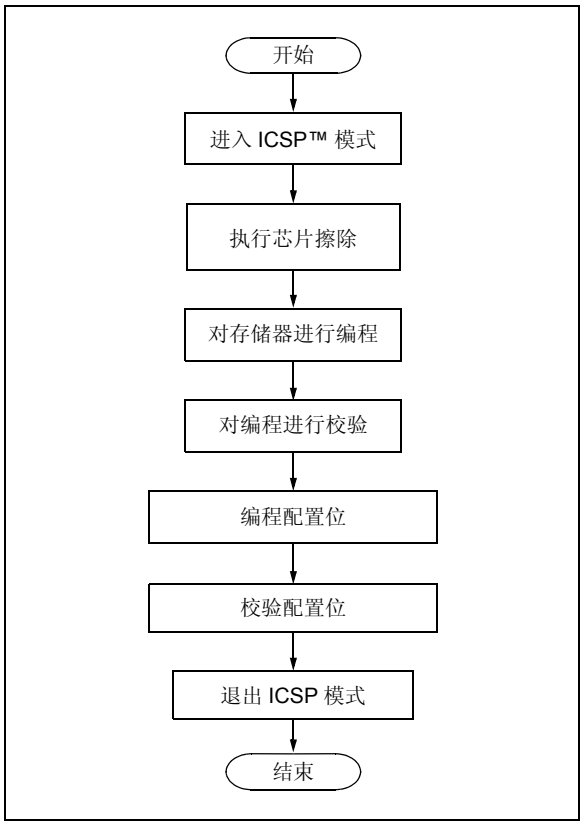
在 ICSP 模式下，系统时钟总是来自 PGECx 引脚，而无论器件的振荡器配置位的设置如何。首先，所有指令会串行移入内部缓冲区，然后装入指令寄存器（Instruction Register，IR）并执行。不会从内部存储器执行程序取操作。一次送入 24 位指令。PGEDx 用来移入数据，PGECx 既用作串行移位时钟又用作 CPU 执行时钟。

| **注：** | 在 ICSP 操作期间，PGECx 的工作频率不能超过 10 MHz。 |
|---|---|

## 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器件进行芯片擦除。然后，对代码存储区编程，并接着对器件配置寄存器编程。最后，对代码存储区 （包括配置寄存器）进行校验以确保编程成功。之后，如果需要，对代码保护配置位编程。

**图 3-1： ICSP™ 编程高阶流程图**




## 3.2 ICSP 工作原理

在进入 ICSP 模式时， CPU 为空闲。 CPU 执行由内部状态机控制。使用 PGECx 和 PGEDx 引脚在时钟控制下移入 4 位用来命令 CPU 的控制代码 （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 寄存器从器件中读取数据。

**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 （0784h）寄存器。 |
| 0010-1111 | N/A | 保留。 |

### 3.2.1 SIX 串行指令执行

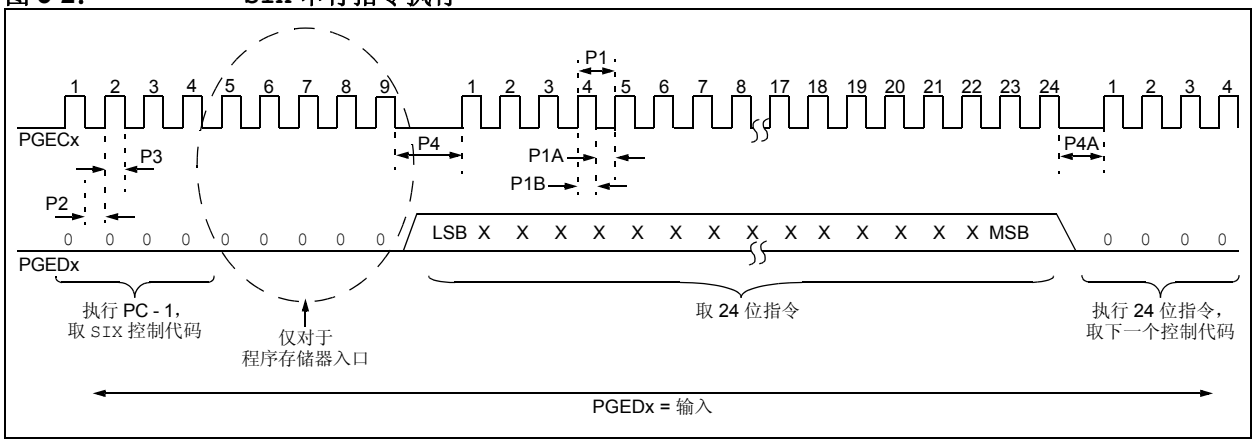
SIX 控制代码允许执行 PIC24F 的汇编指令。当接收到 SIX 代码时，CPU 暂停 24 个时钟周期，在这段时间指令被移入内部缓冲区。一旦移入指令，状态机就允许其在接下来的 4 个 PGECx 时钟周期内执行。当执行接收到的指令时，状态机会同时移入下一条 4 位命令（见图 3-2）。

退出复位后，第一个 4 位控制代码总是被强制为 SIX，并强制 CPU 执行一条 NOP 指令。起振时需要 5 个额外的 PGECx 时钟，从而会产生一个 9 位的 SIX 命令，而不是通常情况下的 4 位 SIX 命令。

在时钟控制下移入强制 SIX 后，ICSP 操作恢复正常。即，接下来的 24 个时钟周期会向 CPU 装入第一个指令字。

| | |
|---|---|
| **注：** | 考虑到这条强制执行的 NOP 指令，本规范中的所有示例代码都以 NOP 开始，以确保不会丢失数据。 |

**图 3-2： SIX 串行指令执行**




#### 3.2.1.1 执行 SIX 和正常指令之间的差异

正常执行指令和使用 SIX ICSP 命令之间存在一些差异。因此，本规范中的代码示例可能与用于在正常器件操作期间执行相同功能的代码不一致。

SIX ICSP 操作期间：

- 双字指令需要两个 SIX 操作来在时钟控制下移入所有必需数据。

  双字指令的示例为 GOTO 和 CALL。
- 双周期指令需要两个 SIX 操作。

  第一个 SIX 操作移入指令并开始执行它。第二个 SIX 操作（应移入 NOP 以避免丢失数据）提供完成执行指令所需的 CPU 时钟。

  双周期指令的示例为表读和表写指令。
- CPU 不会自动发生停顿来应对流水线变化。

  当某条指令修改了由随后的指令用于间接寻址的寄存器时，会发生 CPU 停顿。

正常器件操作期间：

- 在读取新数据时，CPU 会自动强制执行一条 NOP。在使用 ICSP 时，不会自动发生停顿，所以在对最近修改过的寄存器进行任何间接引用之前，都应先执行一条 NOP。

  例如，在指令 MOV #0x0,W0 和 MOV [W0],W1 之间，必须插入一条 NOP。

  如果双周期指令修改了被间接使用的寄存器，则需要两条 NOP：一条用于执行指令的后半部分，另一条用于停顿 CPU 来修正流水线。

  诸如 TBLWTL [W0++],[W1] 之类的指令应跟随两条 NOP。
- 在 ICSP 指令执行期间，即使并未使用闪存，器件程序计数器（Program Counter， PC）也会继续自动递增。

  因此， PC 可能会递增到指向无效的存储单元。无效的存储空间包括未实现的闪存地址和向量空间（单元：0x0 到 0x1FF）。

  如果 PC 指向这些单元，器件将会复位，可能会中断 ICSP 操作。为了防止这种情况，应通过定期执行指令来复位 PC，使之指向安全空间。实现该目的的最佳方法是执行 GOTO 0x200。


© 2014 Microchip Technology Inc.

### 3.2.2 REGOUT 串行指令执行

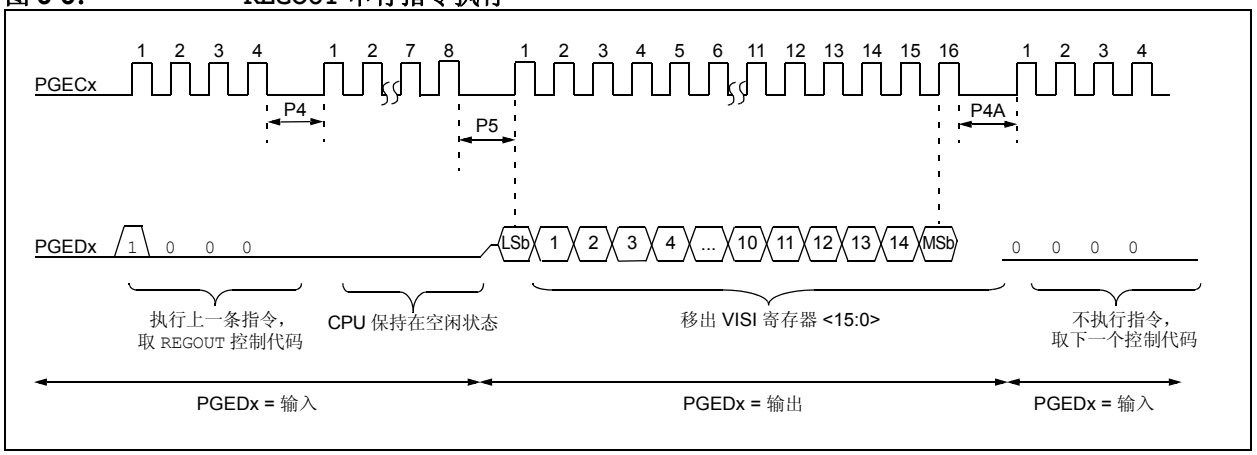
REGOUT 控制代码允许在 ICSP 模式下从器件中读取数据。它用于通过 PGEDx 引脚在时钟控制下移出器件中 VISI 寄存器的内容。接收到 REGOUT 控制代码后，CPU 在 8 个周期内保持空闲。经过这 8 个周期后，还需要额外的 16 个周期将数据移出 （见图 3-3）。

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 寄存器中的数据。

**注 1：** VISI 的内容被移出后，PIC24FJXXXGA2/GB2 器件会将 PGEDx 保持为输出，直到接收到下一个时钟的第一个上升沿为止。

**2：** 数据在 PGECx 下降沿发生更改，在上升沿发生锁存。对于所有的数据发送，都是先发送最低有效位（Least Significant bit，LSb）。

**图 3-3： REGOUT 串行指令执行**



## 3.3 进入 ICSP 模式

如图3-4所示，进入ICSP编程/校验模式需要三个步骤：

1. 将 MCLR 短暂地驱动为高电压，然后再驱动为低电压。
2. 将 32 位密钥序列在时钟控制下移入到 PGEDx 引脚。
3. 然后将 MCLR 驱动为高电压并保持一段指定的时间。

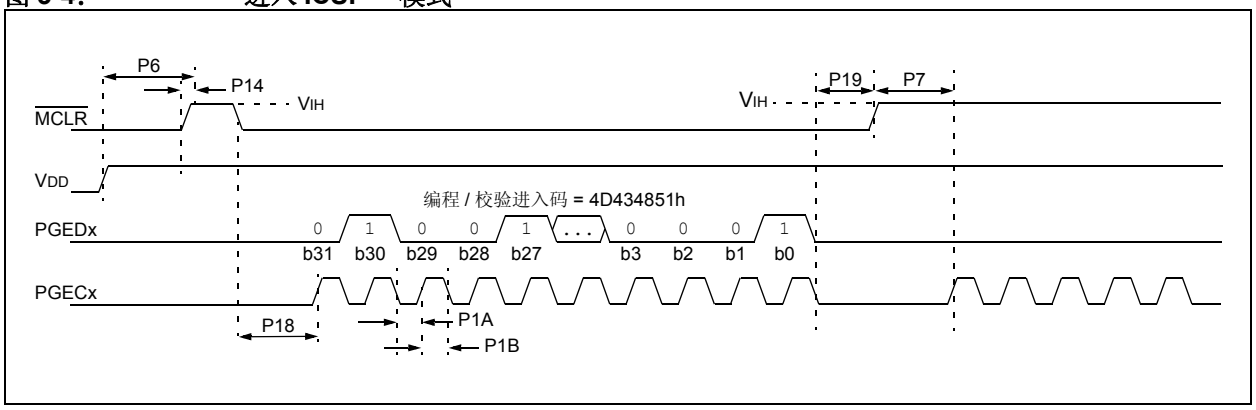
加到MCLR的编程电压为VIH，对于PIC24FJXXXGA2/GB2器件，该电压实际就是 VDD。对于 VIH 没有最短保持时间要求。在 VIH 移除后，必须经过至少 P18 时间间隔才能将密钥序列移入到 PGEDx 引脚。

密钥序列为一个特定的 32 位模式：“0100 1101 0100 0011 0100 1000 0101 0001”（更容易记忆的形式为十六进制的 4D434851h）。只有在密钥序列有效时，器件才能进入编程 / 校验模式。首先必须移入高 16 位的最高有效位 （Most Significant bit， MSb）。

一旦密钥序列完全移入，只要需要保持在编程 / 校验模式，就必须将 VIH 加到 MCLR 并保持该电压。必须经过至少 P19 和 P7 时间间隔才能将数据移入到 PGEDx 引脚。在 P7 之前出现在 PGECx 引脚上的信号被认定为无效。

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

**图 3-4： 进入 ICSP™ 模式**





© 2014 Microchip Technology Inc.

## 3.4 在 ICSP 模式下对闪存编程

### 3.4.1 编程操作

对闪存执行的写和擦除操作是通过 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（见表 3-2）或写操作（见表 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。在用户将 WR 控制位置 1 及其自动清零之间存在一段内部延时，延时结束后编程操作完成。关于与各种编程操作相关的延时信息，请参见**第 7.0 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 |
|---|---|
| 404Fh | 擦除所有代码存储区、执行程序存储区和配置寄存器（不擦除器件 ID 寄存器）。 |
| 4042h | 擦除代码存储区或执行程序存储区中的一页。 |

**表 3-3： NVMCON 写操作**

| NVMCON 值 | 写操作 |
|---|---|
| 4003h | 写入单个代码存储区字、配置字或执行程序存储区字。 |
| 4001h | 对代码存储区或执行程序存储区的 1 行（64 个指令字）进行编程。 |

### 3.4.2 启动和停止编程周期

WR 位（NVMCON<15>）用于启动擦除周期或写周期。将 WR 位置 1 启动编程周期。

所有擦除和写周期都是自定时的。应查询 WR 位以确定擦除或写周期是否已完成。启动编程周期的方法如下：

```
BSET    NVMCON, #WR
```

## 3.5 擦除程序存储器

擦除程序存储器（所有的代码存储区、数据存储区、执行程序存储区和代码保护位）的过程包括将 NVMCON 设置为 404Fh，并执行编程周期。

芯片擦除可以擦除所有用户存储区或用户和配置存储区。在执行芯片擦除之前，应先执行一条表写指令，用于选择要擦除哪些部分。

执行表写指令之后：

- 如果 TBLPAG 寄存器指向用户空间（小于 80h），则芯片擦除将仅擦除用户存储区和闪存配置字。
- 如果 TBLPAG 寄存器指向配置空间（大于等于 80h），则不允许芯片擦除。可以擦除配置空间，一次一页。

**注：** TBLPAG 指向配置空间时不允许芯片擦除，以避免诊断和校准字被擦除。

图 3-5 显示了执行擦除芯片的 ICSP 编程过程。该过程包括 ICSP 命令代码，对于每条指令必须首先使用 PGECx 和 PGEDx 引脚发送命令的 LSb（见图 3-2）。

**注：** 在向程序存储器写入任何数据之前必须先将其擦除。

**图 3-5： 芯片擦除流程**

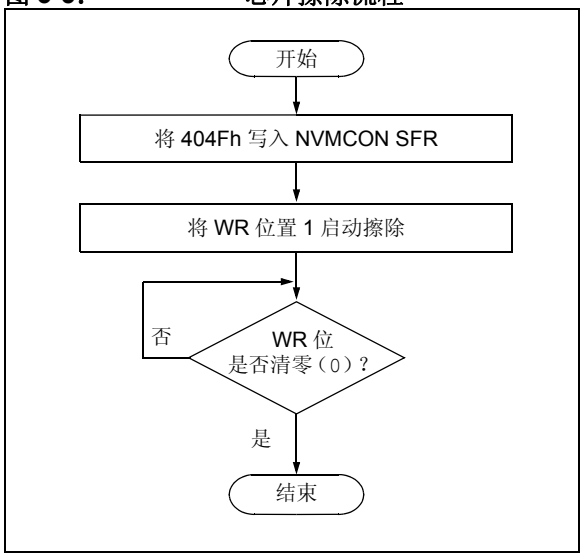


**表 3-4：　　芯片擦除的串行指令执行**

| 命令<br>（二进制） | 数据<br>（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 寄存器以擦除所有程序存储器。 | | |
| 0000 | 2404FA | MOV #0x404F, W10 |
| 0000 | 883B0A | MOV W10, NVMCON |
| **第 3 步：** 设置 TBLPAG 寄存器并执行假表写操作，选择要擦除存储器的哪些部分。 | | |
| 0000 | 2xxxx0 | MOV #<PAGEVAL>, W0 |
| 0000 | 8802A0 | MOV W0, TBLPAG |
| 0000 | 200000 | MOV #0x0000, W0 |
| 0000 | BB0800 | TBLWTL W0,[W0] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **第 4 步：** 启动擦除周期。 | | |
| 0000 | A8E761 | BSET NVMCON, #WR |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **第 5 步：** 重复该步骤来查询 WR 位 （NVMCON 的 bit 15），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 contents of the VISI register |
| 0000 | 000000 | NOP |


© 2014 Microchip Technology Inc.

## 3.6 写代码存储区

写代码存储区的过程与写配置寄存器的过程相同，只不过它一次编程 64 个指令字。为了便于操作，使用工作寄存器 W0:W5 用于存放待编程数据的临时保存寄存器。

表 3-5 给出了 ICSP 编程的详细步骤，包括 ICSP 命令代码的串行模式，必须使用 PGECx 和 PGEDx 引脚先发送命令代码的 LSb （见图 3-2）。

在第 1 步中，退出复位向量。在第 2 步中，为对代码存储区的一整行进行编程而初始化 NVMCON 寄存器。在第 3 步中，将要编程的存储区的 24 位起始目标地址装入 TBLPAG 寄存器和 W7 寄存器。（起始目标地址的高字节存储到 TBLPAG，而该目标地址的低 16 位存储到 W7。）

为了尽量减少编程时间，使用了打包指令格式 （见图 3-6）。

在第 4 步中，使用 MOV 指令将 4 个打包指令字存储在工作寄存器 W0:W5 中。初始化读指针 W6。保存打包指令字数据的 W0:W5 的内容如图 3-6 所示。

在第 5 步中，使用 8 条 TBLWT 指令将 W0:W5 中的数据复制到代码存储区的写锁存器。因为一次只能将 64 个指令字烧写到代码存储区，所以必须将第 4 步和第 5 步重复执行 16 次，以装载所有的写锁存器 （第 6 步）。

装载完写锁存器后，在第 7 步和第 8 步中写入 NVMCON 寄存器启动编程。在第 9 步中，将内部 PC 复位为 200h。当对大存储容量器件编程时，这是防止 PC 递增到未实现存储区的一种预防措施。最后在第 10 步中，重复第 3 步到第 9 步，直到所有代码存储区都被编程为止。

**图 3-6： W0:W5 中的打包指令字**

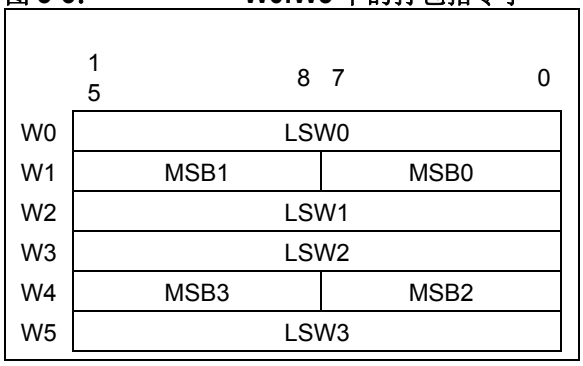



**表 3-5： 写代码存储区的串行指令执行**

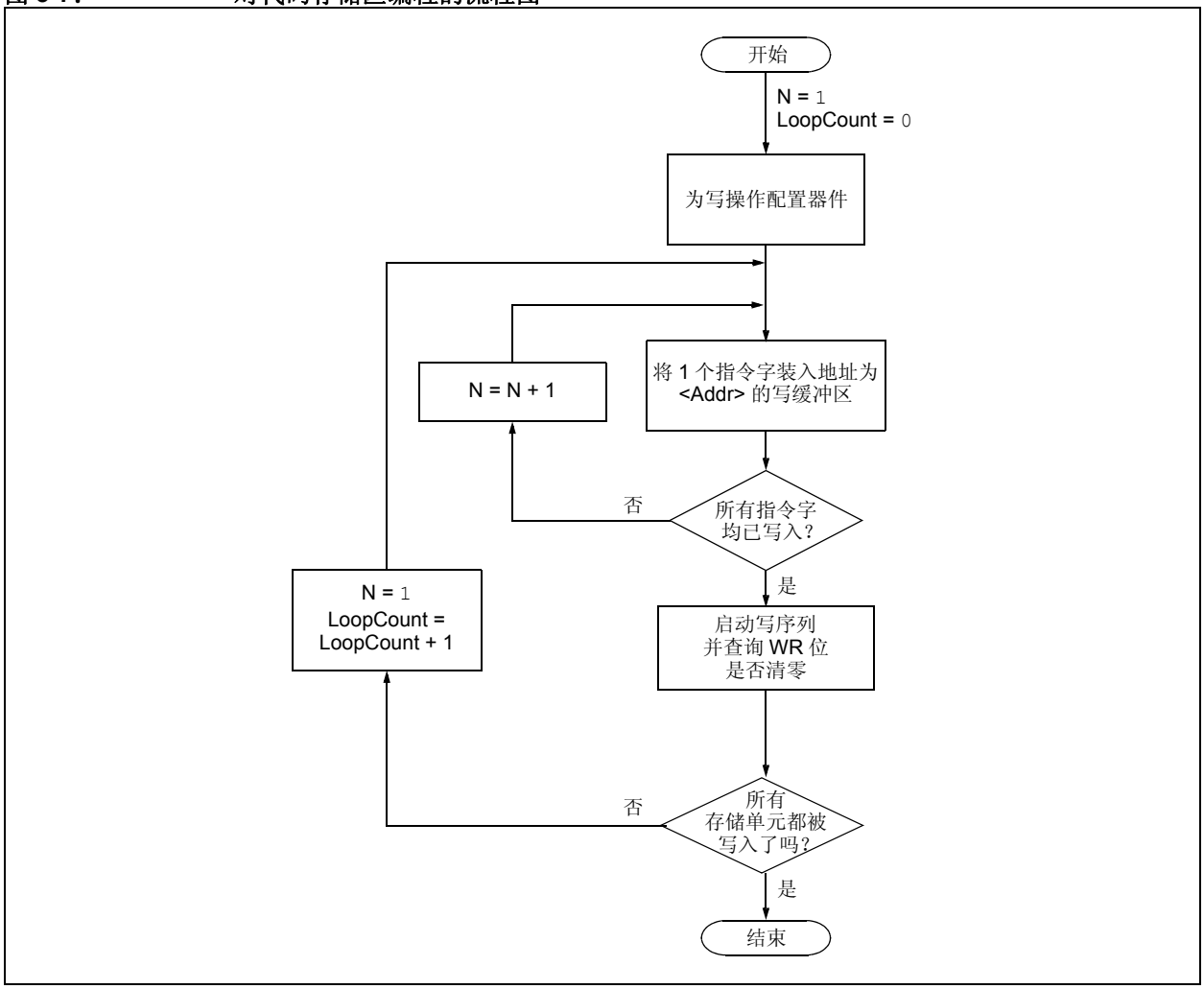
| 命令 （二进制） | 数据 （十六进制） | 说明 |
|---|---|---|
| **第 1 步：** 退出复位向量。 | | |
| 0000<br>0000<br>0000 | 000000<br>040200<br>000000 | NOP<br>GOTO 0x200<br>NOP |
| **第 2 步：** 设置 NVMCON 寄存器来烧写 64 个指令字。 | | |
| 0000<br>0000 | 24,001A<br>883B0A | MOV #0x4001, W10<br>MOV W10, NVMCON |
| **第 3 步：** 初始化 TBLWT 指令的写指针 （W7）。 | | |
| 0000<br>0000<br>0000 | 200xx0<br>8802A0<br>2xxxx7 | MOV #<DestinationAddress23:16>, W0<br>MOV W0, TBLPAG<br>MOV #<DestinationAddress15:0>, W7 |
| **第 4 步：** 向 W0:W5 装入要编程的下 4 个指令字。 | | |
| 0000<br>0000<br>0000<br>0000<br>0000<br>0000 | 2xxxx0<br>2xxxx1<br>2xxxx2<br>2xxxx3<br>2xxxx4<br>2xxxx5 | MOV #<LSW0>, W0<br>MOV #<MSB1:MSB0>, W1<br>MOV #<LSW1>, W2<br>MOV #<LSW2>, W3<br>MOV #<MSB3:MSB2>, W4<br>MOV #<LSW3>, W5 |

表 3-5： 写代码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 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 | 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 16 次，将 64 个指令字装入写锁存器。 | | |
| **第 7 步：** 启动写周期。 | | |
| 0000 | A8E761 | BSET NVMCON, #WR |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **第 8 步：** 重复该步骤来查询 WR 位 （NVMCON 的 bit 15），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 contents of the VISI register. |
| 0000 | 000000 | NOP |
| **第 9 步：** 复位器件内部 PC。 | | |
| 0000 | 040200 | GOTO 0x200 |
| 0000 | 000000 | NOP |
| **第 10 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。 | | |


© 2014 Microchip Technology Inc.

**图 3-7：　　对代码存储区编程的流程图**



## 3.7 写配置字

PIC24FJXXXGA2/GB2 器件的器件配置存储在用户空间程序存储器末尾的闪存配置字中，以及位于测试空间的多个配置字寄存器中。这些寄存器会反映在发生任何复位时从程序存储单元读取的值。表 3-6 列出了默认器件配置的配置字的值。

**表 3-6： 配置寄存器的默认值**

| 地址 | 名称 | 默认值 |
|---|---|---|
| 最后一个字 | CW1 | 7FFFh |
| 最后一个字 - 2 | CW2 | BFFFh |
| 最后一个字 - 4 | CW3 | FFFFh |
| 最后一个字 - 6 | CW4 | FFFFh |

更改这些值的唯一方式是对相应闪存配置字的内容进行编程并复位器件。复位会强制自动重新装入闪存存储的配置值，其方式是按顺序读取专用的闪存配置字，并将数据传送到配置寄存器中。

对于 PIC24FJXXXGA2/GB2 器件，一些保留的配置位具有必须总是保持的默认状态（不论其他配置位的设置如何），以保证器件功能。表 3-7 中列出了这些位及其值。

要在对闪存配置字编程之后更改闪存配置字的值，必须对器件进行芯片擦除（如**第 3.5 节 “擦除程序存储器”**所述）并重新编程为所需的值。将 0 编程为 1 是不可能的；可以从 1 编程为 0 来使能代码保护。

表 3-8 列出了对配置字单元进行编程的 ICSP 编程详细步骤。其中包括 ICSP 命令代码的串行模式，必须使用 PGECx 和 PGEDx 引脚先发送命令代码的 LSb（见图 3-2）。

在第 1 步中，退出复位向量。在第 2 步中，将源地址的低 16 位存储到 W7 中。在第 3 步中，为对代码存储区进行编程而初始化 NVMCON 寄存器。在第 4 步中，将要写入的存储区的 24 位起始源地址的高字节装入 TBLPAG 寄存器。

对于 64 KB 器件，TBLPAG 寄存器必须装入 00h，对于 128 KB 和 256 KB 器件，必须装入 01h。

要在按顺序执行写操作后通过读取配置字来校验数据，初始时应将代码保护位编程为 1，以确保可以正确执行校验。校验完毕后，可以通过对相应的配置字执行字写操作将代码保护位编程为 0。

**表 3-7： 保留的配置位单元**

| 系列 | 寄存器 / 位 | 值 | 备注 |
|---|---|---|---|
| 所有器件 | CW1<15> | 0 | 总是编程为 0；这是为了维持器件功能所必需的。 |
| | CW2<2> | 1 | 总是编程为 1。 |
| | CW2<14> | 0 | 总是编程为 0。 |
| | CW3<7> | 1 | 总是编程为 1。 |
| | CW4<9> | 1 | 总是编程为 1。 |
| PIC24FJXXGA2XX | CW2<11> | 1 | 总是编程为 1。 |


© 2014 Microchip Technology Inc.

**表 3-8：　　写配置寄存器的串行指令执行**

| 命令<br>（二进制） | 数据<br>（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 TBLWT 指令的写指针 （W7）。 | | |
| 0000 | 2xxxx7 | MOV #<CW1Address15:0>, W7 |
| **第 3 步：** 设置 NVMCON 寄存器来对 CW1 编程。 | | |
| 0000 | 24,003A | MOV #0x4003, W10 |
| 0000 | 883B0A | MOV W10, NVMCON |
| **第 4 步：** 初始化 TBLPAG 寄存器。 | | |
| 0000 | 200xx0 | MOV #<CW1Address23:16>, W0 |
| 0000 | 8802A0 | MOV W0, TBLPAG |
| **第 5 步：** 将配置寄存器数据装入 W6。 | | |
| 0000 | 2xxxx6 | MOV #<CW1_VALUE>, W6 |
| **第 6 步：** 将配置寄存器数据写入写锁存器并将写指针递减 1。 | | |
| 0000 | 200008 | MOV #0x0000, W8 |
| 0000 | 000000 | NOP |
| 0000 | BBCB88 | TBLWTH.B W8, [W7] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0000 | BB1386 | TBLWTL.W W6, [W7--] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **第 7 步：** 启动写周期。 | | |
| 0000 | A8E761 | BSET NVMCON, #WR |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **第 8 步：** 重复该步骤来查询 WR 位 （NVMCON 的 bit 15），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 contents of the VISI register. |
| 0000 | 000000 | NOP |
| **第 9 步：** 复位器件内部 PC。 | | |
| 0000 | 040200 | GOTO 0x200 |
| 0000 | 000000 | NOP |
| **第 10 步：** 重复第 5 步至第 9 步来写入配置字 2 至配置字 4。 | | |

## 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 命令在时钟控制下移出数据完成的。

表 3-9 给出了读代码存储区的 ICSP 编程的详细步骤。在第 1 步中，退出复位向量。在第 2 步中，初始化写指针 W7。在第 3 步中，将要读取的存储区的 24 位起始源地址装入 TBLPAG 寄存器和 W6 寄存器。起始源地址的高字节存储到 TBLPAG，而该源地址的低 16 位存储到 W6。

为了使读取时间最短，写操作时使用的指令字打包格式也同样用于读操作（见图 3-6）。在第 4 步中，从代码存储区中读取两个指令字，并使用 REGOUT 命令在时钟控制下通过 VISI 寄存器将其移出器件。重复执行第 4 步直到读完所需大小的代码存储区为止。

**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），使之指向 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下两个单元的内容。 | | |
| 0000 | BA0B96 | TBLRDL [W6], [W7] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out 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 contents of the VISI register. |
| 0000 | 000000 | NOP |
| 0000 | BA0BB6 | TBLRDL [W6++], [W7] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of the VISI register. |
| 0000 | 000000 | NOP |
| **第 5 步：** 复位器件内部 PC。 | | |
| 0000 | 040200 | GOTO 0x200 |
| 0000 | 000000 | NOP |
| **第 6 步：** 重复第 3 步到第 5 步，直到读取完所需的所有代码存储区为止（请注意，“复位器件内部 PC”将为第 5 步）。 | | |


© 2014 Microchip Technology Inc.

## 3.9 读配置字

读配置存储区的过程类似于读代码存储区的过程，只是读取的是 16 位数据字而不是 24 位字。读配置字时每次读取一个寄存器。

表 3-10 列出了读配置字的 ICSP 编程详细步骤。请注意，对于 64 KB 器件，TBLPAG 寄存器必须装入 00h，对于 128 KB 和 256 KB 器件，必须装入 01h。W6 初始化为配置字单元的低 16 位。

**表 3-10： 读所有配置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、读指针 （W6）和写指针 （W7）。 | | |
| 0000 | 200xx0 | MOV #<CW1Address23:16>, W0 |
| 0000 | 8802A0 | MOV W0, TBLPAG |
| 0000 | 2xxxx6 | MOV #<CW1Address15:0>, W6 |
| 0000 | 207847 | MOV #VISI, W7 |
| 0000 | 000000 | NOP |
| **第 3 步：** 读配置寄存器并将读到的值写入 VISI 寄存器（地址为 784h），并使用 REGOUT 命令在时钟控制下移出 VISI 寄存器的内容。 | | |
| 0000 | BA0BA6 | TBLRDL [W6--], [W7] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of the VISI register |
| 0000 | 000000 | NOP |
| **第 4 步：** 重复第 3 步来读取配置字 2 至配置字 4。 | | |
| **第 5 步：** 复位器件内部 PC。 | | |
| 0000 | 040200 | GOTO 0x200 |
| 0000 | 000000 | NOP |

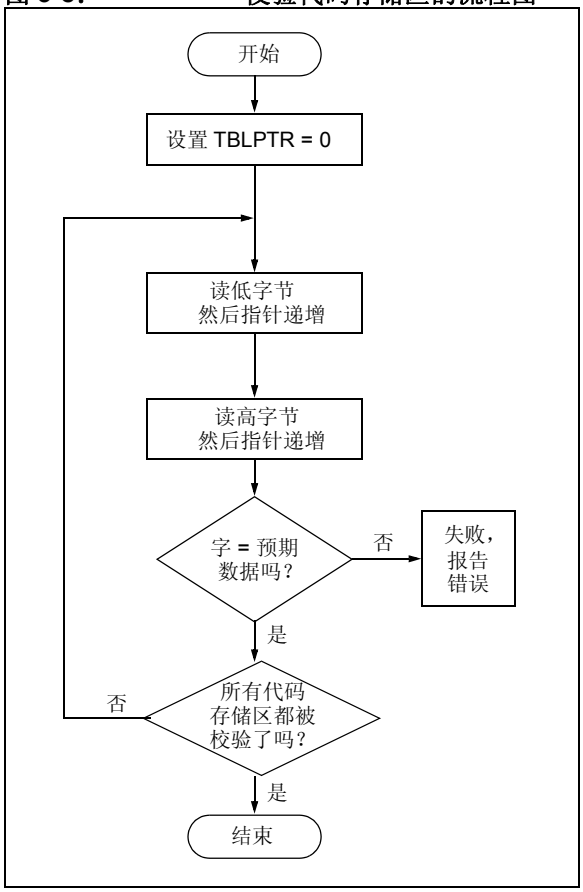
## 3.10 校验代码存储区和配置字

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寄存器。

图 3-8 给出了校验过程的流程图。对存储区执行读操作一次读取一个字节，所以必须读出两个字节以与编程器缓冲区中的字作比较。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：** | 由于配置寄存器包含器件代码保护位，应在写入代码存储区后立即对其进行校验（如果使能代码保护）。这是因为如果在 CW1 中的代码保护位清零后器件发生了复位，将无法读取或校验器件。 |

**图 3-8： 校验代码存储区的流程图**




## 3.11 读应用程序 ID 字

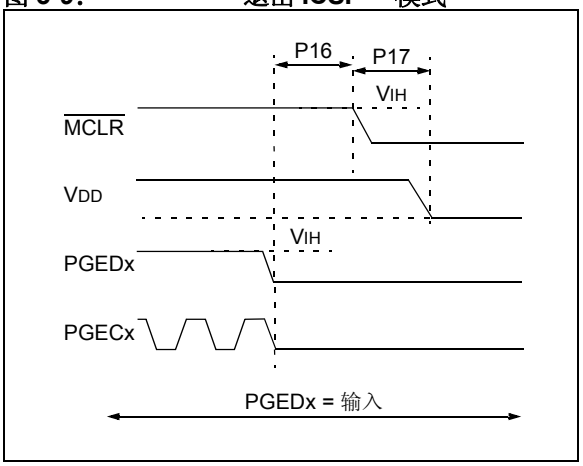
应用程序 ID 字存储在执行程序代码存储区内地址为 8007F0h 的存储单元中。要读取该存储单元，必须使用 SIX 控制代码将该程序存储单元中的内容移入 VISI 寄存器。然后，必须使用 REGOUT 控制代码在时钟控制下将 VISI 寄存器的内容移出器件。表 3-11 给出了必须串行发送给器件以执行该操作的相应控制和指令代码。

编程器在时钟控制下移出应用程序 ID 字后，就必须对其进行检查。如果应用程序 ID 的值为 CEh，表明编程执行程序位于执行程序存储区中，可以采用**第 4.0 节 “器件编程——增强型 ICSP”**中所述的机制对器件编程。但是，如果应用程序 ID 为其他值，则表明编程执行程序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 5.4 节 “将编程执行程序烧写到存储区中”**中介绍了将编程执行程序装入存储区的过程。

## 3.12 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 3-9 所示。退出操作的唯一要求是移除 VIH 要在 PGECx 和 PGEDx 引脚上最后一个时钟和编程信号后的 P16 时间间隔后进行。

**图 3-9： 退出 ICSP™ 模式**





© 2014 Microchip Technology Inc.

**表 3-11：　　读应用程序 ID 字的串行指令执行**

| 命令<br>（二进制） | 数据<br>（十六进制） | 说明 |
|---|---|---|
| **第 1 步：** 退出复位向量。 | | |
| 0000<br>0000<br>0000 | 000000<br>040200<br>000000 | NOP<br>GOTO 0x200<br>NOP |
| **第 2 步：** 为执行 TBLRD 指令初始化 TBLPAG 寄存器和读指针 （W0）。 | | |
| 0000<br>0000<br>0000<br>0000<br>0000<br>0000<br>0000<br>0000 | 200800<br>8802A0<br>207F00<br>207841<br>000000<br>BA0890<br>000000<br>000000 | MOV #0x80, W0<br>MOV W0, TBLPAG<br>MOV #0x07F0, W0<br>MOV #VISI, W1<br>NOP<br>TBLRDL [W0], [W1]<br>NOP<br>NOP |
| **第 3 步：** 使用 REGOUT 命令输出 VISI 寄存器的内容。 | | |
| 0001<br>0000 | <VISI><br>000000 | Clock out contents of the VISI register<br>NOP |

## 4.0 器件编程——增强型 ICSP

本章讨论通过增强型 ICSP 和编程执行程序对器件编程。编程执行程序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 编程模式时执行编程执行程序。编程执行程序为编程器（主机器件）提供了使用简单命令集和通信协议对 PIC24FJXXXGA2/GB2 器件进行编程和校验的机制。编程执行程序可提供以下几项基本功能：

- 读取存储器
- 擦除存储器
- 编程存储器
- 空白检查
- 读执行程序固件版本

编程执行程序执行擦除、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。表 4-1 汇总了这些命令。关于每条命令的详细说明，请参见**第 5.2 节 “编程执行程序命令”**。

**表 4-1： 命令集汇总**

| 命令 | 说明 |
|---|---|
| SCHECK | 工作状态检查 |
| READC | 读器件 ID 寄存器 |
| READP | 读代码存储区 |
| PROGP | 对代码存储区中的一行进行编程并校验 |
| PROGW | 对代码存储区中的一个字进行编程并校验 |
| QBLANK | 查询代码存储区是否为空白 |
| QVER | 查询软件版本 |

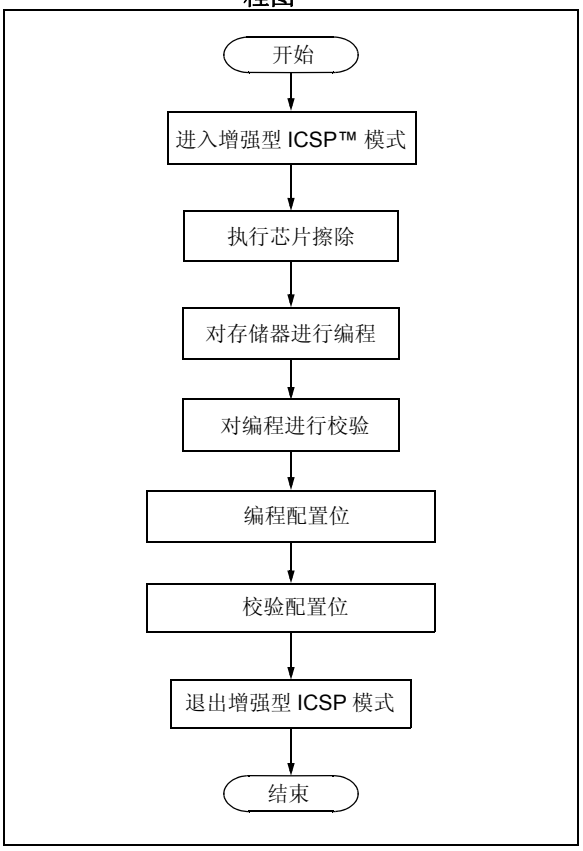
编程执行程序使用器件的数据 RAM 来存放变量和执行程序。运行了编程执行程序后，数据 RAM 中的内容就无法确定了。

## 4.1 编程过程概述

图 4-1 高度概括了编程过程。进入增强型 ICSP 模式后，对编程执行程序进行校验。接下来，擦除器件。然后，对代码存储区编程，并接着对配置单元编程。最后，对代码存储区（包括配置寄存器）进行校验以确保编程成功。

一旦验证了编程执行程序存在于存储区中（或者如果不存在也已装入），则可以使用表 4-1 中给出的命令集对 PIC24FJXXXGA2/GB2 器件编程。

**图 4-1： 高级增强型 ICSP™ 编程流程图**




## 4.2 确认编程执行程序的存在

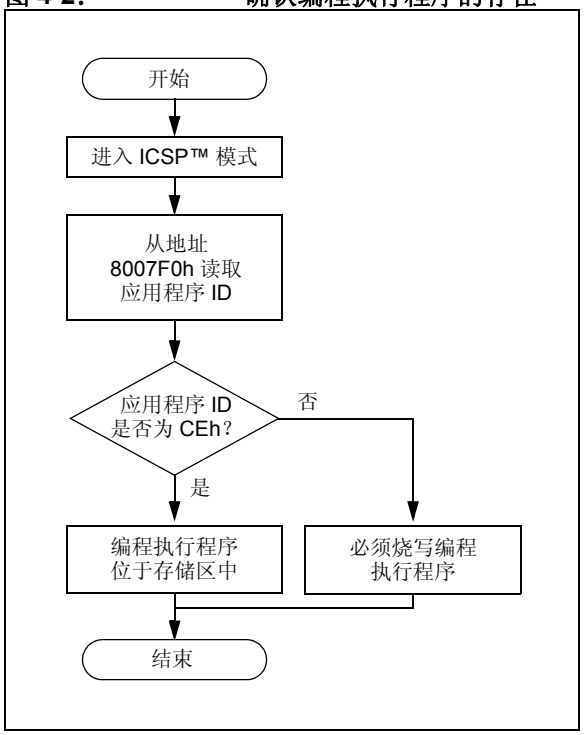
在开始编程之前，编程人员必须确认执行程序存储区中存放了编程执行程序。图 4-2 给出了该任务的流程。

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。如果编程执行程序已位于执行程序存储区中，则应用程序 ID 字为 CEh，这意味着可以进行常规编程。但是，如果应用程序 ID 字不为 CEh，则必须通过使用**第 5.4 节 “将编程执行程序烧写到存储区中”**中所述的方法将编程执行程序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 3.11 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。


© 2014 Microchip Technology Inc.

**图 4-2：　　确认编程执行程序的存在**




## 4.3　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 $\overline{\text{MCLR}}$ 引脚短暂地驱动为高电压，然后再驱动为低电压。
2. 将 32 位密钥序列在时钟控制下移入到 PGEDx 引脚。
3. 然后将 $\overline{\text{MCLR}}$ 驱动为高电压并保持一段指定的时间。

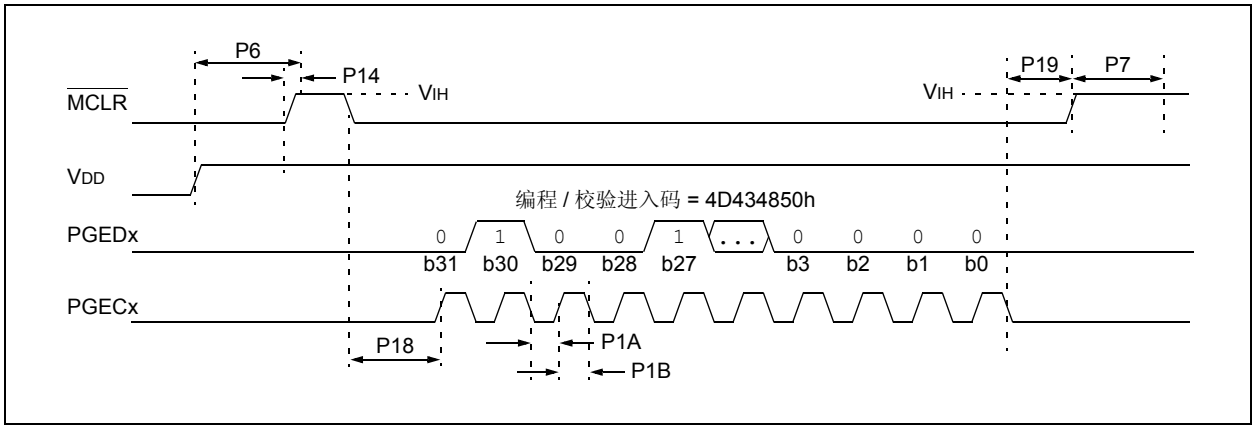
加到 $\overline{\text{MCLR}}$ 的编程电压为 VIH，对于 PIC24FJXXXGA2/GB2 器件，该电压实际就是 VDD。对于 VIH 没有最短保持时间要求。在 VIH 移除后，必须经过至少 P18 时间间隔才能将密钥序列移入到 PGEDx 引脚。

密钥序列为一个特定的 32 位模式：“0100 1101 0100 0011 0100 1000 0101 0000”（更容易记忆的形式为十六进制格式的 4D434850h）。只有在密钥序列有效时，器件才能进入编程/校验模式。首先必须移入高16位的 MSb。

一旦密钥序列完全移入，只要需要保持在编程 / 校验模式，就必须将 VIH 加到 MCLR 并保持该电压。必须经过至少 P19 和 P7 时间间隔才能将数据移入到 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，就能以串行的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。

**图 4-3：　　进入增强型 ICSP™ 模式**



## 4.4 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总是读为 1。

器件 ID 寄存器（FF0002h:FF0000h）将被空白检查忽略，因为此区域存储了不可擦除的器件信息。器件配置寄存器也会被空白检查忽略。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。该命令通过检测代码存储区，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）作为响应。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。

## 4.5 代码存储区编程

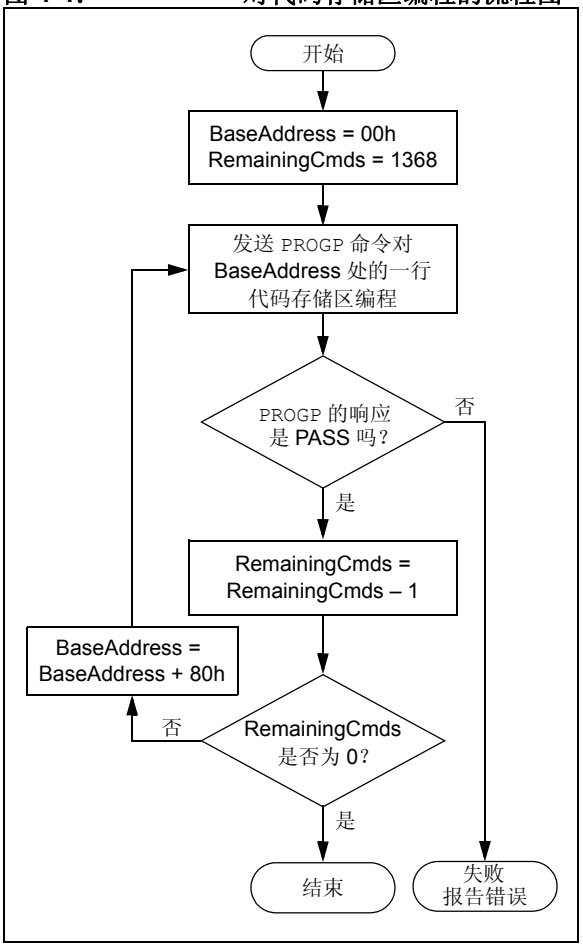
### 4.5.1 编程方法

代码存储区使用 PROGP 命令进行编程。 PROGP 命令对在命令中指定存储地址开始的代码存储区的一行进行编程。对器件编程所需的 PROGP 命令数取决于器件中必须被编程的写块数。

图 4-4 给出了对 PIC24FJXXXGA2/GB2 器件的代码存储区进行编程的流程图。在该示例中，对 256 KB 器件的所有 87K 指令字进行编程。首先，将要发送的命令数（在流程图中称为“RemainingCmds”）设置为 1368，将目标地址（称为“BaseAddress”）设置为 0。接着，使用 PROGP 命令对器件中的一个写块进行编程。每条 PROGP 命令包含将写入器件代码存储区中一行的数据。在成功处理第一条命令后，“RemainingCmds”将递减 1 并与 0 作比较。由于还有 PROGP 命令需要发送，“BaseAddress”将递增 80h 以指向存储区的下一行。

执行第二条 PROGP 命令时，将对第二行进行编程。将重复执行此过程直到整个器件都被编程为止。在越过分区边界时，不需要执行任何特殊处理。

**图 4-4： 对代码存储区编程的流程图**




### 4.5.2 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，并将读回的数据与保存在编程器的缓冲区中的副本作比较。

READP 命令可对整个已编程的代码存储区执行读回操作。

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执行校验。


© 2014 Microchip Technology Inc.

## 4.6 配置位编程

### 4.6.1 概述

PIC24FJXXXGA2/GB2 器件具有一些存储在所实现程序存储器最后 4 个单元的配置位（关于这些单元，请参见表 2-2）。可以将这些位置 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 4-2 列出了 PIC24FJXXXGA2/GB2 器件系列的配置位的说明。

**注：** 虽然一些配置位单元并未实现特定的功能，但它们具有必须总是保持的默认状态（不论其他配置位的设置如何），以保证器件功能。这些位单元及其默认状态的列表，请参见表 3-7。

**表 4-2： PIC24FJXXXGA2/GB2 配置位说明**

| 位域 | 寄存器 (1) | 说明 |
|---|---|---|
| ALTCMPI | CW2<12> | 备用比较器输入位<br>1 = C1INC 处于 RB13 上， C2INC 处于 RB9 上， C3INC 处于 RA0 上<br>0 = C1INC、 C2INC 和 C3INC 都处于 RB9 上 |
| ALTRB6(2,3) | CW2<11> | 备用 RB6 引脚功能单元使能位<br>1 = 将 RB6 的 RP6/ASCL1/PMD6 功能附加到 RA1 引脚功能上<br>0 = 将 RP6/ASCL1/PMD6 功能保留在 RB6 上 |
| BOREN | CW3<12> | 欠压复位使能位<br>1 = 用硬件使能 BOR （在深度休眠之外）<br>0 = 禁止 BOR |
| $\overline{\text{DEBUG}}$ | CW1<11> | 后台调试器使能位<br>1 = 器件复位到工作模式<br>0 = 器件复位到调试模式 |
| DSSWEN | CW4<8> | 深度休眠软件控制位<br>1 = 使能深度休眠操作，并由 DSEN 位控制<br>0 = 总是禁止深度休眠操作 |
| DSWDTEN | CW4<7> | 深度休眠看门狗定时器 （DSWDT）使能位<br>1 = 在深度休眠模式下使能 DSWDT<br>0 = 禁止 DSWDT |
| DSBOREN | CW4<6> | 深度休眠 BOR 使能位<br>1 = 在深度休眠模式下使能 BOR<br>0 = 在深度休眠模式下禁止 BOR （不影响休眠模式） |
| DSWDTOSC | CW4<5> | DSWDT 参考时钟选择位<br>1 = DSWDT 使用 LPRC 作为参考时钟<br>0 = DSWDT 使用 SOSC 作为参考时钟 |

**注 1：** bit <23:16> 的值应设定为 00h，以确保在意外地将任何配置字作为程序执行时，它们会被解释为 NOP 操作码。

**2：** 每当使用 VBUS 功能时，该位必须为 1。

**3：** 该位在 PIC24FJXXXGA20X 器件中是保留的；默认值为 1。

**4：** 只能使用在线串行编程 （ICSP™）修改 JTAGEN 位。

**5：** 无论 WPCFG 状态如何，如果 WPEND = 1 或 WPFP<6:0> 对应于配置字页，都会对配置字页进行保护。

**6：** 保留；对于 PIC24FJXXXGA20X 系列不要使用。

表 4-2： PIC24FJXXXGA2/GB2 配置位说明 （续）

| 位域 | 寄存器 (1) | 说明 |
|---|---|---|
| DSWDPS<4:0> | CW4<4:0> | 深度休眠看门狗定时器后分频比选择位<br>DSWDT 预分频比为 32；这产生一个约为 1 ms 的基本时间单位。<br>11111 = 1:68,719,476736 （25.7 天）<br>11110 = 1:34,359,738368 （12.8 天）<br>11101 = 1:17,179,869184 （6.4 天）<br>11100 = 1:8,589,934592 （77.0 小时）<br>11011 = 1:4,294,967296 （38.5 小时）<br>11010 = 1:2,147,483648 （19.2 小时）<br>11001 = 1:1,073,741824 （9.6 小时）<br>11000 = 1:536,870912 （4.8 小时）<br>10111 = 1:268,435456 （2.4 小时）<br>10110 = 1:134,217728 （72.2 分钟）<br>10101 = 1:67,108864 （36.1 分钟）<br>10100 = 1:33,554432 （18.0 分钟）<br>10011 = 1:16,777216 （9.0 分钟）<br>10010 = 1:8,388608 （4.5 分钟）<br>10001 = 1:4,194304 （135.3 s）<br>10000 = 1:2,097152 （67.7 s）<br>01111 = 1:1,048576 （33.825 s）<br>01110 = 1:524288 （16.912 s）<br>01101 = 1:262114 （8.456 s）<br>01100 = 1:131072 （4.228 s）<br>01011 = 1:65536 （2.114 s）<br>01010 = 1:32768 （1.057 s）<br>01001 = 1:16384 （528.5 ms）<br>01000 = 1:8192 （264.3 ms）<br>00111 = 1:4096 （132.1 ms）<br>00110 = 1:2048 （66.1 ms）<br>00101 = 1:1024 （33 ms）<br>00100 = 1:512 （16.5 ms）<br>00011 = 1:256 （8.3 ms）<br>00010 = 1:128 （4.1 ms）<br>00001 = 1:64 （2.1 ms）<br>00000 = 1:32 （1 ms） |
| FCKSM<1:0> | CW2<7:6> | 时钟切换和故障保护时钟监视器选择配置位<br>1x = 禁止时钟切换和故障保护时钟监视器<br>01 = 使能时钟切换，禁止故障保护时钟监视器<br>00 = 使能时钟切换，使能故障保护时钟监视器 |

注 **1：** bit <23:16> 的值应设定为 00h，以确保在意外地将任何配置字作为程序执行时，它们会被解释为 NOP 操作码。

**2：** 每当使用 VBUS 功能时，该位必须为 1。

**3：** 该位在 PIC24FJXXXGA20X 器件中是保留的；默认值为 1。

**4：** 只能使用在线串行编程 （ICSP™）修改 JTAGEN 位。

**5：** 无论 WPCFG 状态如何，如果 WPEND = 1 或 WPFP<6:0> 对应于配置字页，都会对配置字页进行保护。

**6：** 保留；对于 PIC24FJXXXGA20X 系列不要使用。


© 2014 Microchip Technology Inc.

**表 4-2： PIC24FJXXXGA2/GB2 配置位说明 （续）**

| 位域 | 寄存器 (1) | 说明 |
|---|---|---|
| FNOSC<2:0> | CW2<10:8> | 初始振荡器源选择位<br>111 = 带后分频器模块的快速 RC （FRCDIV）振荡器<br>110 = 保留<br>101 = 低功耗 RC （LPRC）振荡器<br>100 = 辅助 （SOSC）振荡器<br>011 = 带 PLL 模块的主振荡器 （XTPLL、 HSPLL 和 ECPLL）<br>010 = 主 （XT、 HS 或 EC）振荡器<br>001 = 带后分频器和 PLL 模块的快速 RC （FRCPLL）振荡器<br>000 = 快速 RC （FRC）振荡器 |
| FWDTEN<1:0> | CW1<7:6> | 看门狗定时器配置位<br>11 = 用硬件使能看门狗定时器<br>10 = 看门狗定时器由 SWDTEN 位设置控制<br>01 = 仅当器件工作时使能看门狗定时器，休眠模式下禁止看门狗定时器；禁止 SWDTEN 位<br>00 = 用硬件禁止看门狗定时器；禁止 SWDTEN 位 |
| FWPSA | CW1<4> | 看门狗定时器预分频比选择位<br>1 = 看门狗定时器预分频比为 1:128<br>0 = 看门狗定时器预分频比为 1:32 |
| GCP | CW1<13> | 通用段程序存储器代码保护位<br>1 = 禁止代码保护<br>0 = 对整个程序存储空间使能代码保护 |
| GWRP | CW1<12> | 通用段代码闪存写保护位<br>1 = 允许写程序存储器<br>0 = 禁止写程序存储器 |
| I2C1SEL | CW4<14> | 备用 I2C1 单元选择位<br>1 = I2C1 与 SDA1 和 SCL1 进行复用 （默认）<br>0 = I2C1 与 ASDA1 和 ASCL1 进行复用 |
| ICS<1:0> | CW1<9:8> | ICD 仿真器引脚位置选择位<br>11 = 仿真器功能与 PGEC1/PGED1 共用<br>10 = 仿真器功能与 PGEC2/PGED2 共用<br>01 = 仿真器功能与 PGEC3/PGED3 共用<br>00 = 保留；不要使用 |
| IESO | CW2<15> | 内 / 外部切换位<br>1 = 使能双速启动<br>0 = 禁止双速启动 |
| IOL1WAY | CW4<15> | IOLOCK 位一次置 1 使能位<br>1 = IOLOCK 位（OSCCON<6>）可以置 1 一次，前提是已经完成解锁序列。置 1 后，不能再次写入外设引脚选择 （PPS）寄存器。<br>0 = 根据需要清零 IOLOCK （前提是已执行解锁序列） |

**注 1：** bit <23:16> 的值应设定为 00h，以确保在意外地将任何配置字作为程序执行时，它们会被解释为 NOP 操作码。

**2：** 每当使用 VBUS 功能时，该位必须为 1。

**3：** 该位在 PIC24FJXXXGA20X 器件中是保留的；默认值为 1。

**4：** 只能使用在线串行编程 （ICSP™）修改 JTAGEN 位。

**5：** 无论 WPCFG 状态如何，如果 WPEND = 1 或 WPFP<6:0> 对应于配置字页，都会对配置字页进行保护。

**6：** 保留；对于 PIC24FJXXXGA20X 系列不要使用。

表 4-2： PIC24FJXXXGA2/GB2 配置位说明 （续）

| 位域 | 寄存器 (1) | 说明 |
|---|---|---|
| JTAGEN (4) | CW1<14> | JTAG 使能位<br>1 = 使能 JTAG 端口<br>0 = 禁止 JTAG 端口 |
| $\overline{\text{LPCFG}}$ | CW1<10> | 低功耗 （低电压）稳压器控制使能位<br>1 = 禁止低电压稳压器，不论 RETEN 位如何<br>0 = 低电压稳压器功能可用，在休眠期间由 RETEN 位控制 |
| OSCIOFCN | CW2<5> | OSC2 引脚功能位 （XT 和 HS 模式除外）<br>如果 POSCMD<1:0> = 11 或 00：<br>1 = OSCO/CLKO/RA3 用作 CLKO （FOSC/2）<br>0 = OSCO/CLKO/RA3 用作端口 I/O （RA3）<br>如果 POSCMD<1:0> = 10 或 01：<br>OSCIOFCN 对 OSCO/CLKO/RA3 没有影响。 |
| PLLDIV<3:0> | CW4<13:10> | PLL 输入预分频比选择位<br>1111 = 禁止 PLL<br>1110 = 选择 8 x PLL<br>1101 = 选择 6 x PLL<br>1100 = 选择 4 x PLL<br>1011 = }<br>. }<br>. } 保留；不要使用<br>. }<br>1001 = }<br>0111 = 振荡器被 12 分频 （48 MHz 输入） (6)<br>0110 = 振荡器被 8 分频 （32 MHz 输入） (6)<br>0101 = 振荡器被 6 分频 （24 MHz 输入） (6)<br>0100 = 振荡器被 5 分频 （20 MHz 输入） (6)<br>0011 = 振荡器被 4 分频 （16 MHz 输入） (6)<br>0010 = 振荡器被 3 分频 （12 MHz 输入） (6)<br>0001 = 振荡器被 2 分频 （8 MHz 输入） (6)<br>0000 = 直接使用振荡器 （4 MHz 输入） (6) |
| PLLSS | CW3<11> | PLL 辅助时钟源选择位<br>1 = 主振荡器<br>0 = FRC 振荡器 （8 MHz 时钟源） |
| POSCMD<1:0> | CW2<1:0> | 主振荡器模式选择位<br>11 = 禁止主振荡器模式<br>10 = 选择 HS 振荡器模式<br>01 = 选择 XT 振荡器模式<br>00 = 选择 EC 振荡器模式 |
| SOSCSEL | CW3<8> | SOSC 选择配置位<br>1 = 辅助晶体振荡器 （SOSC）模式<br>0 = 数字外部时钟 （SCLKI）模式 |

注 1： bit <23:16> 的值应设定为 00h，以确保在意外地将任何配置字作为程序执行时，它们会被解释为 NOP 操作码。

2： 每当使用 VBUS 功能时，该位必须为 1。

3： 该位在 PIC24FJXXXGA20X 器件中是保留的；默认值为 1。

4： 只能使用在线串行编程 （ICSP™）修改 JTAGEN 位。

5： 无论 WPCFG 状态如何，如果 WPEND = 1 或 WPFP<6:0> 对应于配置字页，都会对配置字页进行保护。

6： 保留；对于 PIC24FJXXXGA20X 系列不要使用。


© 2014 Microchip Technology Inc.

**表 4-2： PIC24FJXXXGA2/GB2 配置位说明 （续）**

| 位域 | 寄存器 (1) | 说明 |
|---|---|---|
| WDTCLK<1:0> | CW2<4:3> | WDT 时钟源选择位<br>当 WDTCMX = 1 时：<br>11 = WDT 使用 LPRC<br>10 = 在窗口 WDT 模式下工作时和不使用 LPRC 作为系统时钟时，WDT 使用来自 FRC 的 31 kHz 时钟源；在休眠模式期间和对于其他所有设置使用 LPRC<br>01 = WDT 使用 SOSC 输入<br>00 = 在工作时 WDT 使用系统时钟，在休眠模式下使用 LPRC<br>当 WDTCMX = 0 时：<br>WDTCLKx 位将被忽略， LPRC 为 WDT 时钟源。 |
| WDTCMX | CW2<13> | 看门狗定时器时钟多路开关选择位<br>1 = WDT 时钟源由 WDTCLKx 配置位决定<br>0 = WDT 总是使用 LPRC 作为其时钟源 |
| WDTPS<3:0> | CW1<3:0> | 看门狗定时器后分频比位<br>1111 = 1:32,768<br>1110 = 1:16,384<br>.<br>.<br>.<br>0001 = 1:2<br>0000 = 1:1 |
| WDTWIN<1:0> | CW3<10:9> | 看门狗定时器窗口宽度位<br>00 = 75%<br>01 = 50%<br>10 = 37.5%<br>11 = 25% |
| WINDIS | CW1<5> | 窗口 WDT 位<br>1 = 使能标准看门狗定时器<br>0 = 使能窗口看门狗定时器； FWDTEN<1:0> 不得为 00 |
| WPCFG | CW3<14> | 配置字代码页写保护选择位<br>1 = 最后一页 （位于程序存储器的顶部）和闪存配置字不进行写保护 (5)<br>0 = 最后一页和闪存配置字进行写保护，前提是 WPDIS = 0 |
| WPDIS | CW3<13> | 段写保护禁止位<br>1 = 禁止段代码保护<br>0 = 使能段代码保护；受保护的段由 WPEND、WPCFG 和 WPFPx 配置位定义 |

**注 1：** bit <23:16> 的值应设定为 00h，以确保在意外地将任何配置字作为程序执行时，它们会被解释为 NOP 操作码。

**2：** 每当使用 VBUS 功能时，该位必须为 1。

**3：** 该位在 PIC24FJXXXGA20X 器件中是保留的；默认值为 1。

**4：** 只能使用在线串行编程 （ICSP™）修改 JTAGEN 位。

**5：** 无论 WPCFG 状态如何，如果 WPEND = 1 或 WPFP<6:0> 对应于配置字页，都会对配置字页进行保护。

**6：** 保留；对于 PIC24FJXXXGA20X 系列不要使用。

**表 4-2： PIC24FJXXXGA2/GB2 配置位说明 （续）**

| 位域 | 寄存器 (1) | 说明 |
|---|---|---|
| WPEND | CW3<15> | 段写保护结束页选择位<br>1 = 受保护代码段的上边界位于程序存储器的最后一页；下边界是由 WPFP<6:0> 指定的代码页 (5)<br>0 = 受保护代码段的下边界位于程序存储器的底部（000000h）；上边界是由 WPFP<6:0> 指定的代码页 |
| WPFP<6:0> | CW3<6:0> | 写保护程序闪存页面位 （WPDIS = 0 时有效） (5)<br>当 WPEND = 0 时：<br>对闪存页面进行擦除 / 写保护，从页面 0 开始，在页面 WPFP<6:0> 结束。<br>当 WPEND = 1 时：<br>对闪存页面进行擦除 / 写保护，从页面 WPFP<6:0> 开始，在用户闪存中的最后一页结束。 |

**注 1：** bit <23:16> 的值应设定为 00h，以确保在意外地将任何配置字作为程序执行时，它们会被解释为 NOP 操作码。

**2：** 每当使用 VBUS 功能时，该位必须为 1。

**3：** 该位在 PIC24FJXXXGA20X 器件中是保留的；默认值为 1。

**4：** 只能使用在线串行编程 （ICSP™）修改 JTAGEN 位。

**5：** 无论 WPCFG 状态如何，如果 WPEND = 1 或 WPFP<6:0> 对应于配置字页，都会对配置字页进行保护。

**6：** 保留；对于 PIC24FJXXXGA20X 系列不要使用。


© 2014 Microchip Technology Inc.

### 4.6.2 编程方法

使用 PROGW 命令对配置位进行编程，一次可编程一个字。该命令指定配置数据和配置寄存器的地址。对配置位进行编程时，任何未实现位或保留位都必须编程为 1。

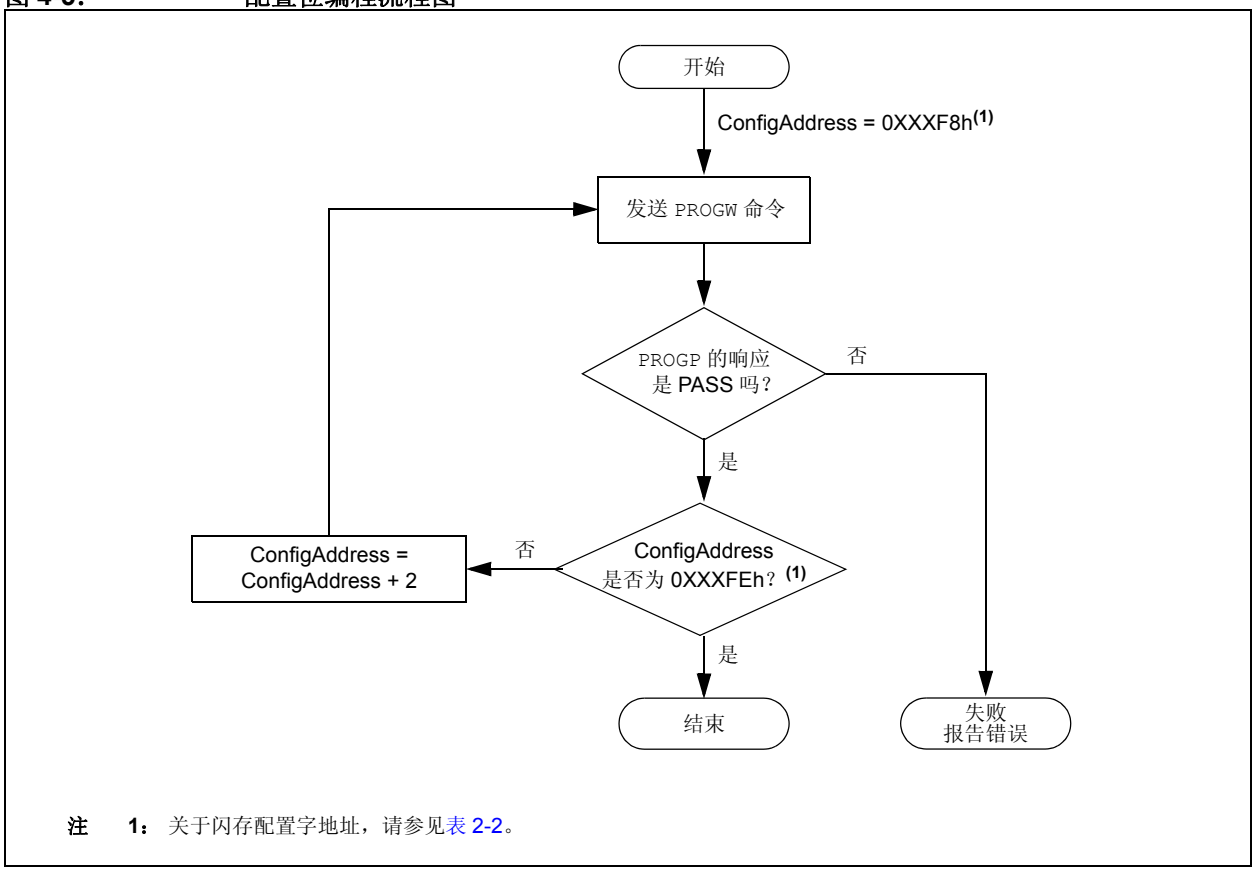
需要 4 条 PROGW 命令来对配置位进行编程。配置位编程的流程图如图 4-5 所示。

**注：** 如果通用段代码保护位（GCP）被编程为 0，则代码存储区被代码保护，不能被读取。在使能读保护之前，必须对代码存储区进行校验。关于代码保护配置位的更多信息，请参见**第 4.6.4 节“代码保护配置位”**。

### 4.6.3 编程校验

一旦配置位编程完毕，就应对存储区的内容进行校验以确保编程成功。校验需要对配置位执行读回操作，并将读回的数据与保存在编程器的缓冲区中的副本作比较。READP 命令读回编程的配置位以校验编程是否成功。

**图 4-5：** **配置位编程流程图**




**注 1：** 关于闪存配置字地址，请参见表 2-2。

### 4.6.4 代码保护配置位

PIC24FJXXXGA2/GB2 器件提供了两种补充方法来防止应用程序代码被改写和擦除。这两种方法还可以防止器件配置在运行过程中被意外更改。更多信息，请参见产品数据手册。

#### 4.6.4.1 通用段保护

对于 PIC24FJXXXGA2/GB2 器件，片上程序存储空间被当作一个单独的块，称为通用段（General Segment，GS）。配置位 GCP 控制该块的代码保护。该位禁止外部对程序存储空间的读和写操作；但对正常执行模式没有直接影响。

写保护由配置字中的 GWRP 位控制。当 GWRP 被编程为 0 时，禁止在内部写和擦除程序存储器。

#### 4.6.4.2 代码段保护

除了全局通用段保护外，可以单独保护程序存储空间的独立子区域以防止对其进行写和擦除操作。该区域在需要写和擦除受保护代码的单独存储块中有多种用途，例如自举应用。与公共引导块实现不同，用户可将 PIC24FJXXXGA2/GB2 器件中特别保护的段放在程序空间中的任何位置，并且可以在较宽范围内进行配置。

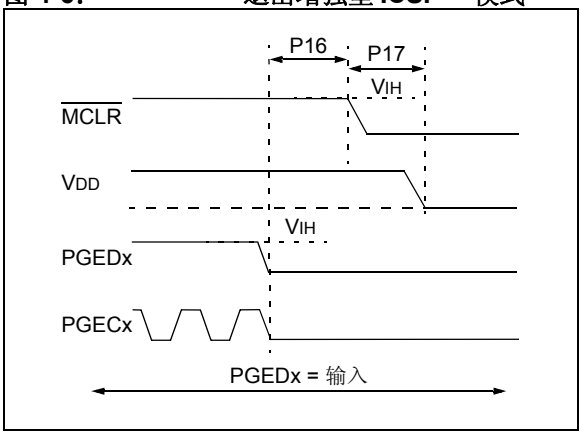
代码段保护还提供了对程序存储器的指定区域的加强级别的保护，这是通过在写或擦除地址发生在指定范围内时禁止 NVM 安全互锁实现的。它不会改写由 GCP 或 GWRP 位控制的通用段保护。例如，如果使能了 GCP 和 GWRP，那么使能程序存储器低半部分的段代码保护不会取消高半部分的通用段保护。

| | |
|---|---|
| **注：** | ICSP模式下的芯片擦除是将代码保护位从开启状态（0）重新编程为关闭状态（1）的唯一方式。 |

## 4.7 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 4-6 所示。退出操作的唯一要求是移除 VIH 要在 PGECx 和 PGEDx 引脚上最后一个时钟和编程信号后的 P16 时间间隔后进行。

**图 4-6： 退出增强型 ICSP™ 模式**





© 2014 Microchip Technology Inc.

## 5.0 编程执行程序

**注：** 编程执行程序（PE）可从 Microchip 网站 www.microchip.com 上的每款器件页面获取。

## 5.1 编程器和编程执行程序之间的通信

编程器和编程执行程序存在主从关系，其中编程器是主编程设备，而编程执行程序处于从动地位。

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编程执行程序。而在编程执行程序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 5.2 节 “编程执行程序命令”**中介绍了编程执行程序命令集。**第 5.3 节 “编程执行程序响应”**中介绍了响应集。

### 5.1.1 通信接口和协议

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 引脚用于向编程执行程序发送命令数据，以及从其接收响应数据。

发送给器件的数据必须在上升沿发生更改，在下降沿保持。从器件接收的数据必须在下降沿发生更改，在上升沿保持。

所有数据传输均先发送MSb，使用16位模式（见图5-1）。

**图 5-1： 从器件接收的数据的编程执行程序串行时序**

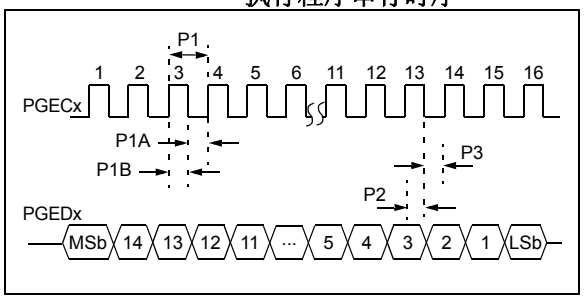
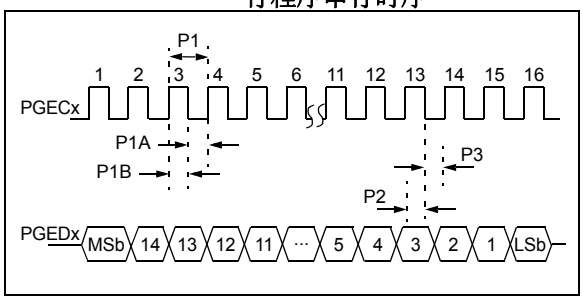



**图 5-2： 发送给器件的数据的编程执行程序串行时序**




由于使用双线 SPI 接口，且数据发送是半双工的，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，并允许编程执行程序将此线驱动为高电平。编程执行程序将 PGEDx 线保持为高电平以指示它正在处理命令。

编程执行程序处理完命令之后，会将 PGEDx 拉为低电平（持续 15 ms），以通知编程器可以在时钟控制下接收响应。编程器将在 PGEDx 变为低电平的 23 ms 后开始接收响应，并且必须提供必需的时钟脉冲数以从编程执行程序接收整个响应。

一旦接收到整个响应，编程器应该终止 PGECx 上的时钟直到向编程执行程序发送另一条命令为止。该协议如图 5-3 所示。

### 5.1.2 SPI 速率

在增强型 ICSP 模式下，PIC24FJXXXGA2/GB2 器件使用内部快速 RC 振荡器（FRCDIV）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 4 MHz 的时钟。

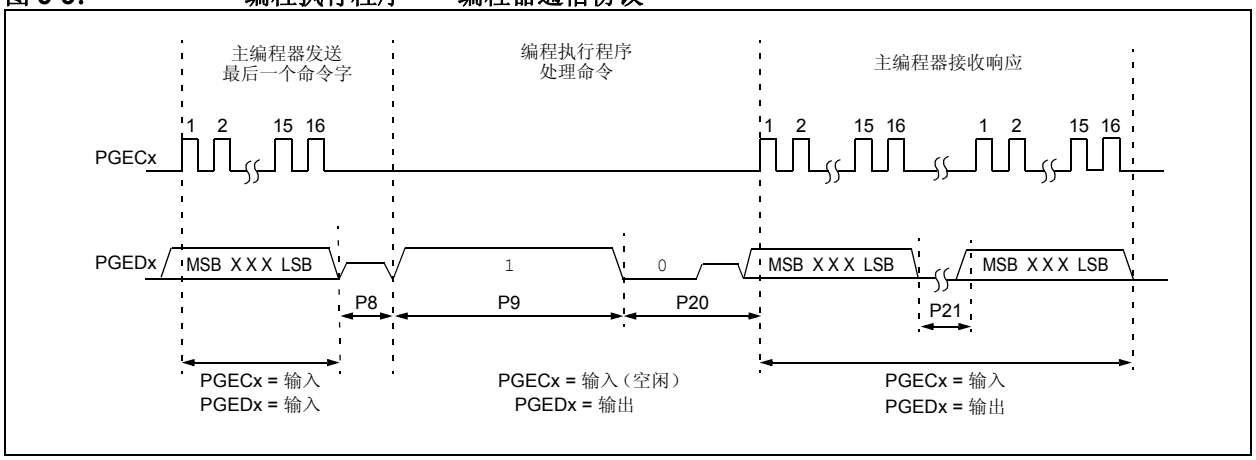
### 5.1.3 超时

在向编程器发送响应的过程中，编程执行程序不使用看门狗定时器或超时。如果编程器不遵守**第 5.1.1 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，编程执行程序可能会表现异常。

由于编程执行程序没有超时，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 5-1 中标识和列出的命令超时。如果命令超时结束，编程器应该将编程执行程序复位并再次开始对器件编程。

**图 5-3：　　编程执行程序——编程器通信协议**




## 5.2 编程执行程序命令

表 5-1 给出了编程执行程序命令集。该表包含每条命令的操作码、助记符、长度、超时和说明。**第 5.2.4 节“命令说明”**中详细说明了每条命令的功能。

### 5.2.1 命令格式

所有编程执行程序命令都具有由 16 位头和该命令所需的所有数据组成的通用格式（见图 5-4）。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 5-4：　　命令格式**

| 15 12 | 11 0 |
|---|---|
| 操作码 | 长度 |
| 命令数据的第一个字 （如果需要） | |
| • | |
| • | |
| 命令数据的最后一个字 （如果需要） | |

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 5-1 中列出的命令不匹配的命令将返回“NACK”响应 （见**第 5.3.1.1 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。编程执行程序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，编程执行程序将无法正确地接收命令。


© 2014 Microchip Technology Inc.

### 5.2.2 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 5-5 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，并为编程执行程序提供了正确对齐以供执行表写操作的数据。

**图 5-5：　　指令字的打包格式**

**注：** 当传输的指令字数为奇数时，MSB2 为零，且 LSW2 不能发送。

### 5.2.3 编程执行程序错误处理

编程执行程序将不应答 （NACK）所有不支持的命令。另外，编程执行程序由于存储空间的限制，所以将不对包含在编程器命令中的数据执行校验。由编程器负责向编程执行程序发送带有有效命令参数的命令，如果命令参数无效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 5.3.1.3 节 “QE_Code 字段”**。

**表 5-1：　　编程执行程序命令集**

| 操作码 | 助记符 | 长度（16 位字） | 超时 | 说明 |
|---|---|---|---|---|
| 0h | SCHECK | 1 | 1 ms | 工作状态检查。 |
| 1h | READC | 3 | 1 ms | 从指定器件 ID 寄存器中读取一个 8 位字。 |
| 2h | READP | 4 | 1 ms/ 行 | 从指定地址开始，读取代码存储区的 N 个 24 位指令字。 |
| 3h | 保留 | N/A | N/A | 该命令是保留的；它将返回 NACK。 |
| 4h | PROGC | 4 | 5 ms | 将一个 8 位字写入指定的器件 ID 寄存器。 |
| 5h | PROGP | 99 | 5 ms | 在指定地址处对代码存储区的一行进行编程，然后进行校验。(1) |
| 6h | 保留 | 5 | 5 ms | 该命令是保留的；它将返回 NACK。 |
| 7h | 保留 | N/A | N/A | 该命令是保留的；它将返回 NACK。 |
| 8h | 保留 | N/A | N/A | 该命令是保留的；它将返回 NACK。 |
| 9h | 保留 | N/A | N/A | 该命令是保留的；它将返回 NACK。 |
| Ah | 保留 | N/A | N/A | 该命令是保留的。 |
| Bh | QVER | 1 | 1 ms | 查询编程执行程序的软件版本。 |
| Ch | 保留 | N/A | N/A | 该命令是保留的；它将返回 NACK。 |
| Dh | PROGW | 4 | 5 ms | 在指定地址处对代码存储区的一个指令字进行编程，然后进行校验。 |
| Eh | QBLANK | 3 | 30 ms/KB | 查询代码存储区是否为空白。 |

**注　1：** 代码存储区的一行由 （64 个） 24 位字组成。请参见表 2-2 以获取特定于器件的信息。

### 5.2.4 命令说明

**第 5.2.5 节 “SCHECK 命令”**至**第 5.2.12 节 “QVER 命令”**介绍了编程执行程序支持的所有命令。

### 5.2.5 SCHECK 命令

| 15 12 | 11 0 |
|---|---|
| 操作码 | 长度 |

| 字段 | 说明 |
|---|---|
| 操作码 | 0h |
| 长度 | 1h |

SCHECK 命令指示编程执行程序仅产生响应而不执行其他操作。该命令用作“工作状态检查”以校验编程执行程序是否正常工作。

**预期的响应（双字）：**

1000h

0002h

**注：** 该指令不是编程必需的，仅供开发使用。

### 5.2.6 READC 命令

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

| 字段 | 说明 |
|---|---|
| 操作码 | 1h |
| 长度 | 3h |
| N | 要读取的 8 位器件 ID 寄存器数量（最大值为 256） |
| Addr_MSB | 24 位源地址的 MSB |
| Addr_LS | 24 位源地址的低 16 位 |

READC 命令指示编程执行程序从由 Addr_MSB 和 Addr_LS 指定的 24 位地址开始读取 N 个器件 ID 寄存器。该命令只能用于读取 8 位或 16 位数据。

当使用该命令读取器件 ID 寄存器时，编程执行程序返回的每个数据字的高字节为 00h，低字节包含器件 ID 寄存器的值。

**预期的响应（4 + 3 * (N - 1)/2 个字，其中 N 为奇数）：**

1100h

2 + N

器件 ID 寄存器 1

...

器件 ID 寄存器 N

**注：** 读取未实现存储区将导致编程执行程序复位。请确保只访问在特定器件中存在的存储单元。


© 2014 Microchip Technology Inc.

### 5.2.7 READP 命令

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

| 字段 | 说明 |
|---|---|
| 操作码 | 2h |
| 长度 | 4h |
| N | 要读取的 24 位指令字数（最大值为 32768） |
| 保留 | 0h |
| Addr_MSB | 24 位源地址的 MSB |
| Addr_LS | 24 位源地址的低 16 位 |

READP 命令指示编程执行程序从由 Addr_MSB 和 Addr_LS 指定的 24 位地址开始读取代码存储区（包括配置字）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 5.2.2 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）：**

1200h

2 + 3 * N/2

最低有效程序存储字 1

...

最低有效数据字 N

**预期的响应（4 + 3 * (N - 1)/2 个字，其中 N 为奇数）：**

1200h

4 + 3 * (N – 1)/2

最低有效程序存储字 1

...

程序存储字 N 的 MSB（进行零填充）

| | |
|---|---|
| **注：** | 读取未实现存储区将导致编程执行程序复位。请确保只访问在特定器件中存在的存储单元。 |

### 5.2.8 PROGC 命令

| 15 12 | 11 8 | 7 0 |
|---|---|---|
| 操作码 | 长度 | |
| 保留 | Addr_MSB | |
| Addr_LS | | |
| 数据 | | |

| 字段 | 说明 |
|---|---|
| 操作码 | 4h |
| 长度 | 4h |
| 保留 | 0h |
| Addr_MSB | 24 位目标地址的 MSB |
| Addr_LS | 24 位目标地址的低 16 位 |
| 数据 | 8 位数据字 |

PROGC命令指示编程执行程序对位于指定存储地址的一个器件 ID 寄存器进行编程。

在指定数据字被编程到代码存储区之后，编程执行程序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）：**

1400h

0002h

### 5.2.9 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_96 | | |

| 字段 | 说明 |
|---|---|
| 操作码 | 5h |
| 长度 | 63h |
| 保留 | 0h |
| 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命令指示编程执行程序对位于指定存储地址的代码存储区（包括配置字）的一行（64 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 80h 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 5-5 给出的指令字打包格式对它们进行编排。

在所有数据被编程到代码存储区之后，编程执行程序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）：**

1500h

0002h

**注：** 关于代码存储区容量的信息，请参见表2-2。

### 5.2.10 PROGW 命令

| 15 12 | 11 8 | 7 0 |
|---|---|---|
| 操作码 | 长度 | |
| Data_MSB | | Addr_MSB |
| Addr_LS | | |
| Data_LS | | |

| 字段 | 说明 |
|---|---|
| 操作码 | Dh |
| 长度 | 4h |
| 保留 | 0h |
| Addr_MSB | 24 位目标地址的 MSB |
| Addr_LS | 24 位目标地址的低 16 位 |
| Data_MSB | 24 位数据的 MSB |
| Data_LS | 24 位数据的低 16 位 |

PROGW 命令指示编程执行程序将代码存储区的一个字（3 字节）编程到特定的存储地址。

在该字被编程到代码存储区之后，编程执行程序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）：**

1600h

0002h


© 2014 Microchip Technology Inc.

### 5.2.11 QBLANK 命令

| 15 12 | 11 0 |
|---|---|
| 操作码 | 长度 |
| PSize_MSW | |
| PSize_LSW | |

| 字段 | 说明 |
|---|---|
| 操作码 | Ah |
| 长度 | 3h |
| PSize | 要以 24 位字为单位检查的程序存储器的长度加 1（最大值为 49152） |

QBLANK 命令查询编程执行程序以确定代码存储区和代码保护配置位（GCP 和 GWRP）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 0h，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和代码保护位为空白，QBLANK 返回的 QE_Code 为 F0h；否则，QBLANK 返回的 QE_Code 为 0Fh。

**预期的响应（空白器件的双字）：**

1AF0h

0002h

**预期的响应（非空白器件的双字）：**

1A0Fh

0002h

**注：** QBLANK不会对系统操作配置位进行检查，因为这些位在执行芯片擦除时不会被设置为 1。

### 5.2.12 QVER 命令

| 15 12 | 11 0 |
|---|---|
| 操作码 | 长度 |

| 字段 | 说明 |
|---|---|
| 操作码 | Bh |
| 长度 | 1h |

QVER 命令查询存储在测试存储空间中的编程执行程序软件的版本。在响应的 QE_Code 中返回“版本 . 修订版”信息，QE_Code 使用一个以下格式的字节：主版本由高半字节表示，修订版本由低半字节表示（例如，23h 表示编程执行程序软件的版本为 2.3）。

**预期的响应（双字）：**

1BMNh（其中“MN”代表版本 M.N）

0002h

## 5.3 编程执行程序响应

每接收到一条命令，编程执行程序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，其中包含任何必需的响应数据或错误数据。

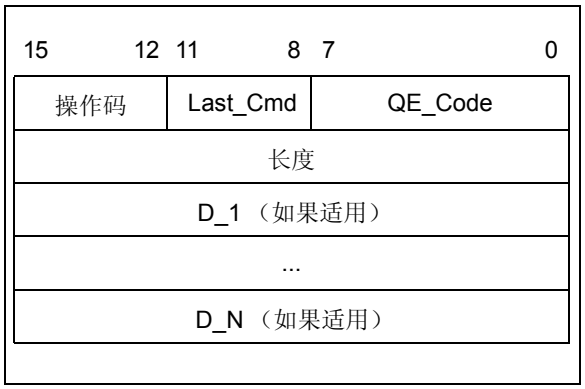
表 5-2 给出了编程执行程序响应集。该表包含每个响应的操作码、助记符和说明。**第 5.3.1 节 “响应格式”**中介绍了响应格式。

**表 5-2： 编程执行程序响应操作码集**

| 操作码 | 助记符 | 说明 |
|---|---|---|
| 1h | PASS | 命令已成功处理 |
| 2h | FAIL | 命令未成功处理 |
| 3h | NACK | 命令未知 |

### 5.3.1 响应格式

所有编程执行程序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。




| 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 （包含双字的响应头） |
| 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） |

#### 5.3.1.1 操作码字段

操作码是响应的第一个字中的 4 位字段。操作码指示命令的处理情况 （见表 5-2）。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。如果发送给编程执行程序的命令未被识别，则编程执行程序将返回 NACK 响应。

#### 5.3.1.2 Last_Cmd 字段

Last_Cmd 是响应的第一个字中的 4 位字段，该字段指示编程执行程序处理的命令。由于编程执行程序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，它可以用来校验编程执行程序是否已正确地接收了编程器发送的命令。


© 2014 Microchip Technology Inc.

#### 5.3.1.3 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。

当编程执行程序处理两种查询命令（QBLANK 或 QVER）之一时，返回的操作码总是 PASS，并且 QE_Code 将保留查询响应数据。表 5-3 给出了两种查询的 QE_Code 的格式。

**表 5-3： 针对查询命令的 QE_Code**

| 查询 | QE_Code |
|---|---|
| QBLANK | 0Fh = 代码存储区非空白<br>F0h = 代码存储区空白 |
| QVER | 0xMN，其中编程执行程序软件版本 = M.N（即， 32h 表示软件版本 3.2） |

当编程执行程序处理除查询外的任何命令时， QE_Code 代表一个错误代码。表 5-4 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 0h，指示在处理命令过程中没有发生错误。如果对 PROGP 或 PROGC 命令编程进行校验时失败，则 QE_Code 将被设置为 1h。对于所有其他编程执行程序错误， QE_Code 为 2h。

**表 5-4： 针对非查询命令的 QE_Code**

| QE_Code | 说明 |
|---|---|
| 0h | 无错误 |
| 1h | 校验失败 |
| 2h | 其他错误 |

#### 5.3.1.4 响应长度

响应长度指示编程执行程序的响应的长度，以 16 位字数表示。该字段包含双字响应头。

除了对 READP 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 5.2.2 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。

## 5.4 将编程执行程序烧写到存储区中

### 5.4.1 概述

如果已确定编程执行程序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），必须要通过**第 3.0 节“器件编程——ICSP”**中所述的 ICSP 方法将其烧写到执行程序存储区中。

将编程执行程序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。即，必须擦除执行程序存储区，然后才能烧写编程执行程序，每次烧写 64 个指令字。表 5-5 给出了该控制流程。

**注：** 编程执行程序必须总是先进行擦除，然后再进行编程，如表 5-5 所述。

**表 5-5： 对执行程序编程**

| 命令（二进制） | 数据（十六进制） | 说明 |
|---|---|---|
| **第 1 步：** 退出复位向量，擦除执行程序存储区。 | | |
| 0000 | 000000 | NOP |
| 0000 | 040200 | GOTO 0x200 |
| 0000 | 000000 | NOP |
| **第 2 步：** 初始化 NVMCON 寄存器以擦除执行程序存储区。 | | |
| 0000 | 240420 | MOV #0x4042, W0 |
| 0000 | 883B00 | MOV W0, NVMCON |
| **第 3 步：** 将擦除指针初始化为执行程序的第一页，然后启动擦除周期。 | | |
| 0000 | 200800 | MOV #0x80, W0 |
| 0000 | 8802A0 | MOV W0, TBLPAG |
| 0000 | 200001 | MOV #0x0, W1 |
| 0000 | 000000 | NOP |
| 0000 | BB0881 | TBLWTL W1, [W1] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0000 | A8E761 | BSET NVMCON, #15 |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **第 4 步：** 重复该步骤来查询 WR 位（NVMCON 的 bit 15），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 |
| 0001 | 000000 | NOP |
| | <VISI> | Clock out contents of the VISI register. |
| 0000 | 000000 | NOP |
| **第 5 步：** 重复第 3 步和第 4 步，擦除执行程序存储区的第二页。 W1 指针应递增 400h，以指向第二页。 | | |
| **第 6 步：** 初始化 NVMCON 寄存器来烧写 64 个指令字。 | | |
| 0000 | 240010 | MOV #0x4001, W0 |
| 0000 | 883B00 | MOV W0, NVMCON |
| **第 7 步：** 初始化 TBLPAG 寄存器和写指针（W7）。 | | |
| 0000 | 200800 | MOV #0x80, W0 |
| 0000 | 8802A0 | MOV W0, TBLPAG |
| 0000 | EB0380 | CLR W7 |
| 0000 | 000000 | NOP |


© 2014 Microchip Technology Inc.

**表 5-5：　　对执行程序编程 （续）**

| 命令 （二进制） | 数据 （十六进制） | 说明 |
|---|---|---|
| **第 8 步：** 向 W0:W5 中装入打包编程执行程序代码的下 4 个字并初始化 W6 来进行编程。编程从执行程序存储区的基址 （800000h）开始，使用 W6 作为读指针， W7 作为写指针。 | | |
| 0000 | 2<LSW0>0 | MOV #<LSW0>, W0 |
| 0000 | 2<MSB1:MSB0>1 | MOV #<MSB1:MSB0>, W1 |
| 0000 | 2<LSW1>2 | MOV #<LSW1>, W2 |
| 0000 | 2<LSW2>3 | MOV #<LSW2>, W3 |
| 0000 | 2<MSB3:MSB2>4 | MOV #<MSB3:MSB2>, W4 |
| 0000 | 2<LSW3>5 | MOV #<LSW3>, W5 |
| **第 9 步：** 设置读指针 （W6）并装载 （下 4 个写）锁存器。 | | |
| 0000 | EB0300 | CLR W6 |
| 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 10 步：**将第 8 步和第 9 步重复执行 16 次，将 64 个指令字装入写锁存器。 | | |
| **第 11 步：**启动编程周期。 | | |
| 0000 | A8E761 | BSET NVMCON, #15 |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **第 12 步：**重复该步骤来查询 WR 位 （NVMCON 的 bit 15），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 contents of the VISI register. |
| 0000 | 000000 | NOP |
| **第 13 步：**复位器件内部 PC。 | | |
| 0000 | 040200 | GOTO 0x200 |
| 0000 | 000000 | NOP |
| **第 14 步：**重复执行第 8 步至第 13 步，直到执行程序存储区的全部 16 行都被编程为止。 | | |

### 5.4.2 编程校验

使用 ICSP 将编程执行程序烧写到执行程序存储区后，必须进行校验。通过读出执行程序存储区的内容并将其与存储在编程器中的编程执行程序的镜像比较来进行校验。

可以使用**第 3.8 节“读代码存储区”**中所述的相同技术来读取执行程序存储区的内容。表 5-6 给出了读取执行程序存储区的过程。请注意，在第 2 步中，将 TBLPAG 寄存器的值设置为 80h，从而可读取执行程序存储区。应使用诊断和校准字的存储值来校验执行程序存储区的最后 8 个字，以确保准确性。

**表 5-6： 读执行程序存储区**

| 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 | 200800 | MOV #0x80, W0 |
| 0000 | 8802A0 | MOV W0, TBLPAG |
| 0000 | EB0300 | CLR W6 |
| **第 3 步：** 初始化写指针 （W7），使之指向 VISI 寄存器。 | | |
| 0000 | 207847 | MOV #VISI, W7 |
| 0000 | 000000 | NOP |
| **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执行程序存储区下两个单元的内容。 | | |
| 0000 | BA0B96 | TBLRDL [W6], [W7] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out 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 contents of the VISI register |
| 0000 | 000000 | NOP |
| 0000 | BA0BB6 | TBLRDL [W6++], [W7] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of the VISI register |
| 0000 | 000000 | NOP |
| **第 5 步：** 复位器件内部 PC。 | | |
| 0000 | 040200 | GOTO 0x200 |
| 0000 | 000000 | NOP |
| **第 6 步：** 重复执行第 4 步和第 5 步，直到读完执行程序存储区的全部 1024 个指令字为止。 | | |


© 2014 Microchip Technology Inc.

# 6.0 器件详细信息

## 6.1 器件 ID

器件 ID 存储区可用于确定芯片的掩膜、型号和生产信息。器件 ID 区为 2 x 16 位，可以使用 READC 命令读取。该存储区只读，并且能在使能代码保护时读取。

表 6-1 列出了每款器件的器件 ID，表 6-2 列出了器件 ID 寄存器，表 6-3 介绍了每个寄存器的位域。

**表 6-1： 器件 ID**

| 器件 | DEVID |
|---|---|
| PIC24FJ128GB204 | 4C5B |
| PIC24FJ128GB202 | 4C5A |
| PIC24FJ64GB204 | 4C59 |
| PIC24FJ64GB202 | 4C58 |
| PIC24FJ128GA204 | 4C53 |
| PIC24FJ128GA202 | 4C52 |
| PIC24FJ64GA204 | 4C51 |
| PIC24FJ64GA202 | 4C50 |

**表 6-2： PIC24FJXXXGA2/GB2 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 |
| FF0000h | DEVID | FAMID<7:0> | | | | | | | | DEV<7:0> | | | | | | | |
| FF0002h | DEVREV | — | | | | | | | | | | | | REV<3:0> | | | |

**表 6-3： 器件 ID 位说明**

| 位域 | 寄存器 | 说明 |
|---|---|---|
| FAMID<7:0> | DEVID | 对器件的 ID 进行编码 |
| DEV<7:0> | DEVID | 对器件的各个 ID 进行编码 |
| REV<3:0> | DEVREV | 对器件的序列 （数字）修订标识符进行编码 |

## 6.2 校验和计算

PIC24FJXXXGA2/GB2 器件的校验和为 16 位大小。校验和通过将以下内容相加来进行计算：

- 代码存储单元的内容
- 配置寄存器的内容

表 6-4 介绍了如何计算每种器件的校验和。将所有存储单元的内容相加，一次一个字节，仅使用它们的本机数据大小。更具体地说，对配置寄存器进行相加时，将加上这些单元的低两个字节（忽略高字节），而对代码存储区进行相加时，将加上代码存储区的全部 3 个字节。

**表 6-4： 校验和计算**

| 器件 | 读代码保护 | 校验和计算 | 擦除后的校验和值 | 0x0 和最后一个代码地址处为 0xAAAAAA 时的校验和 |
|---|---|---|---|---|
| PIC24FJ128GA204 | 禁止 | CFGB + SUM(0:157F7) | F744h | F546h |
| | 使能 | 0 | 0000h | 0000h |
| PIC24FJ128GA202 | 禁止 | CFGB + SUM(0:157F7) | F744h | F546h |
| | 使能 | 0 | 0000h | 0000h |
| PIC24FJ128GB204 | 禁止 | CFGB + SUM(0:157F7) | F744h | F546h |
| | 使能 | 0 | 0000h | 0000h |
| PIC24FJ128GB202 | 禁止 | CFGB + SUM(0:157F7) | F744h | F546h |
| | 使能 | 0 | 0000h | 0000h |
| PIC24FJ64GA204 | 禁止 | CFGB + SUM(0:0ABF7) | F944h | F746h |
| | 使能 | 0 | 0000h | 0000h |
| PIC24FJ64GA202 | 禁止 | CFGB + SUM(0:0ABF7) | F944h | F746h |
| | 使能 | 043 | 0000h | 0000h |
| PIC24FJ64GB204 | 禁止 | CFGB + SUM(0:0ABF7) | F944h | F746h |
| | 使能 | 0 | 0000h | 0000h |
| PIC24FJ64GB202 | 禁止 | CFGB + SUM(0:0ABF7) | F944h | F746h |
| | 使能 | 0 | 0000h | 0000h |

**图注：** 项 说明

SUM[a:b] = a 至 b 的所有存储单元的字节和 （代码存储区的全部 3 字节）

CFGB = 配置块 （掩码）字节和 (CW1 & 0x7FFF) + (CW2 & 0xFFFF) + (CW3 & 0xFFFF) + (CW4 & 0xFFFF)

**图注：** CW1 地址为所实现程序存储器的最后一个单元； CW2 为 （最后一个单元 - 2）； CW3 为 （最后一个单元 - 4）； CW4 为 （最后一个单元 - 6）。


© 2014 Microchip Technology Inc.

## 7.0 交流 / 直流特性和时序要求

**标准工作条件**
工作温度：0°C 至 +70°C。建议在 +25°C 下编程。

| 参数编号 | 符号 | 特性 | 最小值 | 最大值 | 单位 | 条件 |
|---|---|---|---|---|---|---|
| D111 | VDD | 编程时的供电电压 | 2.20 | 3.60 | V | 常规编程 (1) |
| D112 | IPP | $\overline{MCLR}$ 引脚上的编程电流 | — | 5 | μA | |
| D113 | IDDP | 编程时的供电电流 | — | 16 | mA | |
| D031 | VIL | 输入低电压 | VSS | 0.2 VDD | V | |
| D041 | VIH | 输入高电压 | 0.8 VDD | VDD | V | |
| D080 | VOL | 输出低电压 | — | 0.4 | V | IOL = 8.5 mA （3.6V 时） |
| D090 | VOH | 输出高电压 | 3.0 | — | V | IOH = -3.0 mA （3.6V 时） |
| D012 | CIO | I/O 引脚 （PGEDx）上的容性负载 | — | 50 | pF | 符合交流规范 |
| D013 | CF | VCAP 上的滤波电容值 | 4.7 | 10 | μF | 控制器内核需要 |
| P1 | TPGEC | 串行时钟 （PGECx）周期 | 100 | — | ns | ICSP™ 模式 |
| | | | 250 | — | ns | 增强型 ICSP 模式 |
| P1A | TPGECL | 串行时钟 （PGECx）的低电平时间 | 40 | — | ns | ICSP 模式 |
| | | | 100 | — | ns | 增强型 ICSP 模式 |
| P1B | TPGECH | 串行时钟 （PGECx）的高电平时间 | 40 | — | ns | ICSP 模式 |
| | | | 100 | — | ns | 增强型 ICSP 模式 |
| 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 | 4 位命令操作数和下一条 4 位命令之间的延时 | 40 | — | ns | |
| P5 | TDLY2 | 命令字节的最后一个 PGECx↓到读取数据字时出现的第一个 PGECx↑的延时 | 20 | — | ns | |
| P6 | TSET2 | VDD↑到 $\overline{MCLR}$↑的建立时间 | 100 | — | ns | |
| P7 | THLD2 | 在 $\overline{MCLR}$↑后输入数据的保持时间 | 25 | — | ms | |
| P8 | TDLY3 | 命令字节的最后一个 PGECx↓到编程执行程序的 PGEDx↑之间的延时 | 12 | — | μs | |
| P9 | TDLY4 | 编程执行程序命令处理时间 | 40 | — | μs | |
| P10 | TDLY6 | 编程后 PGECx 处于低电平的时间 | 400 | — | ns | |
| P11 | TDLY7 | 芯片擦除时间 | 20 | 40 | ms | |
| P12 | TDLY8 | 页擦除时间 | 20 | 40 | ms | |
| P13 | TDLY9 | 行编程时间 | 1.5 | — | ms | |
| P14 | TR | 进入 ICSP 模式的 $\overline{MCLR}$ 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 $\overline{MCLR}$↓之间的延时 | 0 | — | s | |
| P17 | THLD3 | $\overline{MCLR}$↓到 VDD↓的时间 | 100 | — | ns | |
| P18 | TKEY1 | 从第一个 $\overline{MCLR}$↓到向 PGEDx 输入密钥序列的第一个 PGECx↑之间的延时 | 10 | — | ms | |

**注 1：** 在编程期间还必须将 VDD 加到 AVDD 引脚。 AVDD 和 AVSS 应总是分别处于 VDD 和 VSS 的 ±0.3V 范围内。使能内部稳压器 （即， ENVREG = VDD）时，标称 VCAP 为 1.8V。

## 7.0 交流 / 直流特性和时序要求 （续）

| 标准工作条件<br>工作温度：0°C 至 +70°C。建议在 +25°C 下编程。 | | | | | | |
|---|---|---|---|---|---|---|
| **参数编号** | **符号** | **特性** | **最小值** | **最大值** | **单位** | **条件** |
| P19 | TKEY2 | 从向 PGEDx 输入密钥序列的上一个 PGECx ↓ 到第二个 MCLR ↑ 之间的延时 | 1 | — | ms | |
| P20 | TDLY11 | 编程执行程序的 PGEDx ↓ 到主机驱动 PGEDx 之间的延时 | 23 | — | µs | |
| P21 | TDLY12 | 编程执行程序命令响应字之间的延时 | 8 | — | ns | |

**注 1：** 在编程期间还必须将 VDD 加到 AVDD 引脚。 AVDD 和 AVSS 应总是分别处于 VDD 和 VSS 的 ±0.3V 范围内。使能内部稳压器 （即， ENVREG = VDD）时，标称 VCAP 为 1.8V。


© 2014 Microchip Technology Inc.

## 附录 A： 版本历史

版本 A （2012 年 3 月）

本文档的初始版本，针对 PIC24FJ128GB204 系列器件发布。

版本 B （2012 年 8 月）

在规范中增加了 PIC24FJ128GA204 器件系列：

更改了表 3-4、表 3-5、表 3-8、表 5-5 和**第 2.0 节“PIC24FJXXXGA2/GB2 器件编程概述”**中的操作码。

版本 C （2013 年 1 月）

为清晰起见，重新排布了引脚图，并增加了图 2-6。

更新了表 2-2，包含了新的器件。

更新了表 6-1，更新了所有器件 ID。

更正了关于规范值 P18 （TKEY1）的注释，并删除了备用值； 10 ms 的关键规范值保持不变。

更新了版本 B 的版本历史说明 （先前未提到增加了 PIC24FJ128GA204 器件系列）。

版本 D （2014 年 2 月）

该版本包括以下更新：

- 图片：
  - 更新了图 2-3、图 2-4、图 2-5 和图 2-6。
- 表格：
  - 更新了表 3-7、表 4-2 和表 5-5 。

版本 E （2014 年 5 月）

该版本包括以下更新：

- 表格：
- 更新了表 3-6、表 3-7、表 4-2 和表 6-4。

注：


© 2014 Microchip Technology Inc.

**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。

**提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc. 及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc. 的英文原版文档。**

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。**Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。** Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。如果将 Microchip 器件用于生命维持和 / 或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切伤害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任，并加以赔偿。在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

**商标**

Microchip 的名称和徽标组合、 Microchip 徽标、 dsPIC、 FlashFlex、 flexPWR、 JukeBlox、 KEELOQ、 KEELOQ 徽标、 Kleer、 LANCheck、 MediaLB、 MOST、 MOST 徽标、 MPLAB、 OptoLyzer、 PIC、 PICSTART、 PIC32 徽标、 RightTouch、 SpyNIC、 SST、 SST 徽标、 SuperFlash 及 UNI/O 均为 Microchip Technology Inc. 在美国和其他国家或地区的注册商标。

The Embedded Control Solutions Company 和 mTouch 为 Microchip Technology Inc. 在美国的注册商标。

Analog-for-the-Digital Age、BodyCom、chipKIT、chipKIT 徽标、 CodeGuard、 dsPICDEM、 dsPICDEM.net、 ECAN、 In-Circuit Serial Programming、 ICSP、 Inter-Chip Connectivity、 KleerNet、 KleerNet 徽标、 MiWi、 MPASM、 MPF、 MPLAB Certified 徽标、 MPLIB、 MPLINK、 MultiTRAK、 NetDetach、 Omniscient Code Generation、 PICDEM、 PICDEM.net、 PICkit、 PICtail、 RightTouch 徽标、 REAL ICE、 SQI、 Serial Quad I/O、 Total Endurance、 TSHARC、 USBCheck、 VariSense、 ViewSpan、 WiperLock、 Wireless DNA 和 ZENA 均为 Microchip Technology Inc. 在美国和其他国家或地区的商标。

SQTP 为 Microchip Technology Inc. 在美国的服务标记。

Silicon Storage Technology 为 Microchip Technology Inc. 在除美国外的国家或地区的注册商标。

GestIC 为 Microchip Technology Inc. 的子公司 Microchip Technology Germany II GmbH & Co. & KG 在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

© 2014, Microchip Technology Inc. 版权所有。

ISBN：978-1-63276-617-5

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

*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。*

# 全球销售及服务网点

## 美洲

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

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

**奥斯汀 Austin, TX**
Tel: 1-512-257-3370

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

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

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

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

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

**休斯敦 Houston, TX**
Tel: 1-281-894-5983

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

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

**纽约 New York, NY**
Tel: 1-631-435-6000

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

**加拿大多伦多 Toronto**
Tel: 1-905-673-0699
Fax: 1-905-673-6509

## 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 - 杭州**
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

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

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

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

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

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

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

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

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

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

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

## 亚太地区

**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

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

**德国 Germany - Dusseldorf**
Tel: 49-2129-3766400

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

**德国 Germany - Pforzheim**
Tel: 49-7231-424750

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

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

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

**波兰 Poland - Warsaw**
Tel: 48-22-3325737

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

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

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

03/25/14


© 2014 Microchip Technology Inc.