PIC32 闪存编程规范

1.0 器件概述

本文档定义了 PIC32 系列 32 位单片机的编程规范。本编程规范旨在用于指导开发人员使用外部编程器工具。对于正在开发PIC32器件应用程序的客户应使用已提供器件编程支持的开发工具。

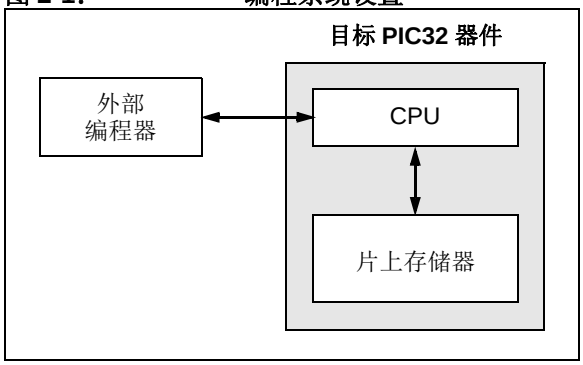
主要讨论的主题包括：

2.0 编程概述

当开发编程工具时，有必要了解目标器件的内部闪存的编程操作和用于控制闪存编程的特殊功能寄存器（Special Function Register， SFR），因为同样这些操作和寄存器会由外部编程工具及其软件使用。这些操作和控制寄存器在具体器件数据手册的 **“闪存程序存储器”** 章节和《PIC32 系列参考手册》的相关章节中进行了说明。强烈推荐将这些文档与本编程规范结合使用。

外部工具编程设置由外部编程器工具和目标PIC32器件组成。图 2-1 描绘了一个典型的编程设置。编程器工具负责执行所需的编程步骤并完成编程操作。

图 2-1： 编程系统设置



2.1 具有双闪存分区和双引导区域的器件

PIC32MK 和 PIC32MZ 系列器件集成了多个用于器件现场（自）编程的实用特性。这些特性包括具有双引导区域的双闪存分区、用于引导区域的别名机制 （允许在启动时自动选择引导代码）和用于程序闪存的分区交换特性。两个闪存分区及其相关的引导区域均可分别擦除和编程。关于这些特性的详细说明，请参见 《PIC32 系列参考手册》的**第 48 章 “存储器构成和权限”**（DS60001214）。

用于量产编程的开发工具将不用考虑大部分特性，但是以下内容除外：

- 确保 SWAP 位 （NVMCON<7>）设置正确。默认设置为 0，无分区交换。在生成用于编程工具的源文件时，开发工具应假设为默认设置。
- 正确处理校验和计算中的引导存储器别名。别名部分将与固定部分一致。关于别名区域校验和计算的更多信息，请参见**第 17.0 节 “校验和”**。

2.2 编程接口

所有PIC32器件都为外部编程器工具提供了两种物理接口：

- 2线在线串行编程（In-Circuit Serial Programming™，ICSP™）
- 4 线联合测试行动组织（Joint Test Action Group，JTAG）

更多信息，请参见**第 4.0 节 “连接至器件”**。

这两种方法都可以使用可下载的编程执行程序（Programming Executive，PE）。PE 可以从目标器件 RAM 中执行，并对于编程器隐藏器件编程细节。它还可以消除与数据传输相关的开销，提高整体数据吞吐量。Microchip 开发了可用于任意外部编程器的 PE（更多信息请参见**第 16.0 节 “编程执行程序”**）。

第 3.0 节 “编程步骤”介绍了一些高级编程步骤，并随后给出了每个步骤的简要说明。本文档的相应章节中则提供了详细的说明。

以下章节中提供了关于编程命令、EJTAG和直流规范的更多详细信息：

- **第 18.0 节 “配置存储器和器件 ID”**
- **第 19.0 节 “TAP 控制器”**
- **第 20.0 节 “交流 / 直流特性和时序要求”**

2.3 增强型 JTAG （EJTAG）

2线和4线接口都使用EJTAG协议来与编程器交换数据。虽然本文档根据需要提供了该协议的工作描述，还是建议高级用户访问 Imagination Technologies Limited 网站（www.imgtec.com）以获取更多信息。

2.4 数据大小

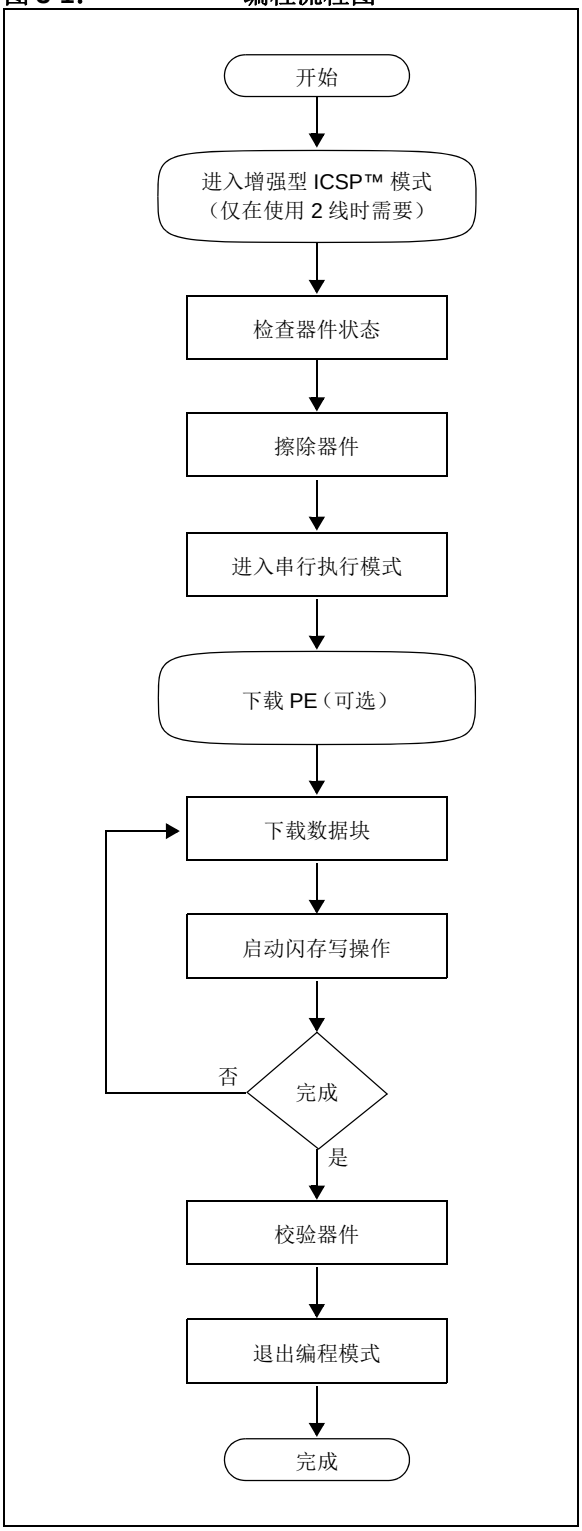
数据大小定义如下：

- 1 个字：32 位
- 1/2 个字：16 位
- 1/4 个字：8 位
- 1 个字节：2 位

3.0 编程步骤

无论所使用的实际方法如何，所有工具编程器都必须执行一组公共的步骤。图 3-1 显示了这组用于对 PIC32 器件进行编程的步骤。

图 3-1： 编程流程图



以下序列列出了这些步骤，并给出了每个步骤的简要说明。后续章节中提供了关于这些步骤更详细的信息。

1. 连接到目标器件。

   为了确保成功编程，所需引脚都必须与相应的信号连接。更多信息，请参见本文档中的**第 4.0 节“连接至器件”**。

2. 将目标器件置为编程模式。

   对于 2 线编程方法，必须在执行任何其他步骤之前，先将目标器件置为一种特殊的编程模式（增强型 ICSP™）。

注：	对于 4 线编程方法，不需要步骤 2。

   更多信息，请参见**第 7.0 节“进入 2 线增强型 ICSP 模式”**。

3. 检查器件的状态。

   步骤 3 用于检查器件的状态，确保它准备好接收来自编程器的信息。

   更多信息，请参见**第 8.0 节“检查器件状态”**。

4. 擦除目标器件。

   如果器件中的目标存储器块不为空，或者如果器件被代码保护，则在烧写任意新数据之前，必须先执行一个擦除步骤。

   更多信息，请参见**第 9.0 节“擦除器件”**。

5. 进入编程模式。

   步骤 5 用于验证器件未被代码保护，并引导 TAP 控制器，使之开始与 PIC32 CPU 进行数据发送和接收。

   更多信息，请参见**第 10.0 节“进入串行执行模式”**。

6. 下载编程执行程序 （PE）。

   PE 是下载到目标器件 RAM 中的一小块可执行代码。它将接收并烧写实际数据。

注：	如果所使用的编程方法不需要 PE，则不需要步骤 6。

   更多信息，请参见**第 11.0 节“下载编程执行程序 （PE）”**。

7. 下载要烧写的数据块。

   所有方法 （无论是否使用 PE）都必须将所需烧写的数据下载到 RAM 中的一个存储器块中。

   更多信息，请参见**第 12.0 节“下载数据块”**。

8. 启动闪存写操作。

   将每个数据块下载到 RAM 中之后，必须启动编程序列，将数据烧写到目标器件的闪存中。

   更多信息，请参见**第 13.0 节“初始化闪存行写操作”**。

9. 重复步骤 7 和 8，直到下载并烧写完所有数据块为止。

10. 校验程序存储器。

    烧写所有编程数据和配置位之后，应回读目标器件存储器，验证内容是否匹配。

    更多信息，请参见**第14.0节“校验器件存储器”**。

11. 退出编程模式。

    只有对目标器件断电并重新加电，或者执行退出编程序列之后，新烧写的数据才会生效。

    更多信息，请参见**第 15.0 节“退出编程模式”**。

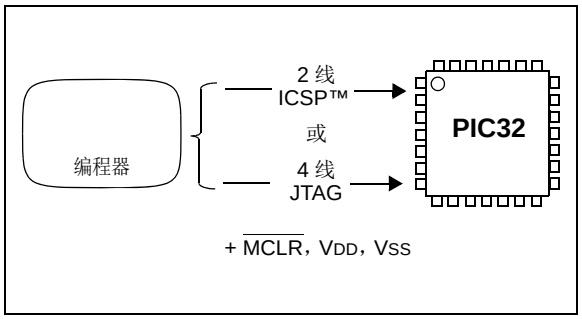
4.0 连接至器件

PIC32 系列提供了两种可供选择的物理接口，用于进行连接和存储器内容编程 （见图 4-1）。对于所有编程接口，都必须为目标器件供电，并且连接所有必需的信号。此外，接口必须使能，可以通过其配置位 （作为 JTAG 4 线接口），或通过专门的初始化序列（作为 2 线 ICSP 接口）实现。

JTAG 接口在空白器件出厂时默认为使能。

在**第 7.0 节 “进入 2 线增强型 ICSP 模式”**中对使能 ICSP 进行了说明。

图 4-1： 编程接口



4.1 4 线接口

一种可供选择的接口是 4 线 JTAG （IEEE 1149.1）端口。表 4-1 列出了所需的引脚连接。该接口使用 4 条通信线，用于与要编程的 PIC32 器件进行数据传输：

- 测试时钟输入 （TCK）
- 测试模式选择输入 （TMS）
- 测试数据输入 （TDI）
- 测试数据输出 （TDO）

关于信号与器件引脚的连接，请参见具体器件数据手册。

4.1.1 测试时钟输入 （TCK）

TCK 是时钟信号，它控制 TAP 控制器的更新和通过指令或选定数据寄存器的数据移位。TCK 的频率和相位均独立于处理器时钟。

4.1.2 测试模式选择输入 （TMS）

TMS 是 TAP 控制器的控制信号。该信号在 TCK 上升沿进行采样。

4.1.3 测试数据输入 （TDI）

TDI 是指令或选定数据寄存器的测试数据输入。该信号在某些 TAP 控制器状态的 TCK 上升沿进行采样。

4.1.4 测试数据输出 （TDO）

TDO 是指令或数据寄存器的测试数据输出。该信号在 TCK 下降沿发生变化。只有在移出数据时才会驱动 TDO，否则 TDO 为三态。

表 4-1： 4 线接口引脚

器件 引脚名称	引脚类型	引脚说明
$\overline{MCLR}$	I	编程使能
ENVREG(2)	I	片上稳压器使能
VDD 和 AVDD(1)	P	电源
VSS 和 AVSS(1)	P	地
VCAP(2)	P	CPU 逻辑滤波电容连接
TDI	I	测试数据输入
TDO	O	测试数据输出
TCK	I	测试时钟
TMS	I	测试模式状态

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，包括模拟电源 （AVDD）和模拟地 （AVSS）。

2： 不是所有器件都有该引脚。请参见具体器件数据手册中的 **“引脚图”**或 **“引脚表”**章节以确定可用性。

4.2 2 线接口

另一种可供选择的接口是 2 线 ICSP 端口。表 4-2 列出了所需的引脚连接。该接口使用 2 条通信线，用于与要编程的 PIC32 器件进行数据传输：

- 串行编程时钟 （PGECx）
- 串行编程数据 （PGEDx）

以下两节介绍了这些信号。关于信号与芯片引脚的连接，请参见具体器件数据手册。

4.2.1 串行编程时钟 （PGECx）

PGECx 是时钟信号，它控制 TAP 控制器的更新和通过指令或选定数据寄存器的数据移位。 PGECx 的频率和相位均独立于处理器时钟。

4.2.2 串行编程数据 （PGEDx）

PGEDx 是指令或选定数据寄存器的输入 / 输出，另外它也是 TAP 控制器的控制信号。该信号在某些 TAP 控制器状态的 PGECx 下降沿进行采样。

表 4-2： 2 线接口引脚

器件 引脚名称	编程器 引脚名称	引脚类型	引脚说明
$\overline{MCLR}$	$\overline{MCLR}$	P	编程使能
ENVREG(2)	N/A	I	片上稳压器使能
VDD 和 AVDD(1)	VDD	P	电源
VSS 和 AVSS(1)	VSS	P	地
VCAP(2)	N/A	P	CPU 逻辑滤波电容连接
PGEC1	PGEC	I	主编程引脚对：串行时钟
PGED1	PGED	I/O	主编程引脚对：串行数据
PGEC2	PGEC	I	辅助编程引脚对：串行时钟
PGED2	PGED	I/O	辅助编程引脚对：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，包括模拟电源 （AVDD）和模拟地 （AVSS）。

2： 不是所有器件都有该引脚。请参见具体器件数据手册中的 **“引脚图”**或 **“引脚表”**章节以确定可用性。

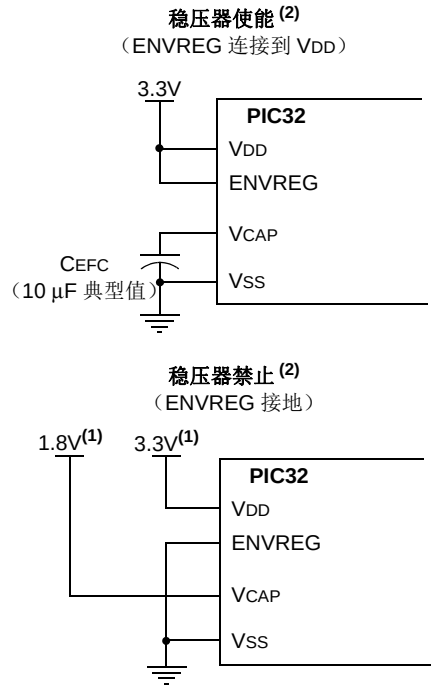
4.3 电源要求

PIC32 系列中的器件均采用双电源设计。一个电源为内核供电，而另一个电源为外设和 I/O 引脚供电。所有器件均包含一个用于较低电压内核电源的片上稳压器，从而无需外部稳压器。片上稳压器有三种实现方式：

- 第一个版本具有一个内部稳压器，可通过 ENVREG 引脚禁止。禁止时，必须使用外部电源为内核供电。如果使能，必须连接一个低 ESR 滤波电容到 VCAP 引脚，见图 4-2。
- 第二个版本具有一个不能被禁止的内部稳压器。必须始终连接一个低 ESR 滤波电容到 VCAP 引脚。
- 第三个版本具有一个不能被禁止的内部稳压器，且无需滤波电容。

请参见**第 20.0 节“交流 / 直流特性和时序要求”**和具体器件数据手册中的**“电气特性”**章节，以了解您器件的电源要求。

图 4-2：　　内部稳压器使能 / 禁止选项



注　1： 这些是典型的工作电压。请参见**第 20.0 节“交流 / 直流特性和时序要求”**了解 VDD 和 VCAP 的完整工作范围。

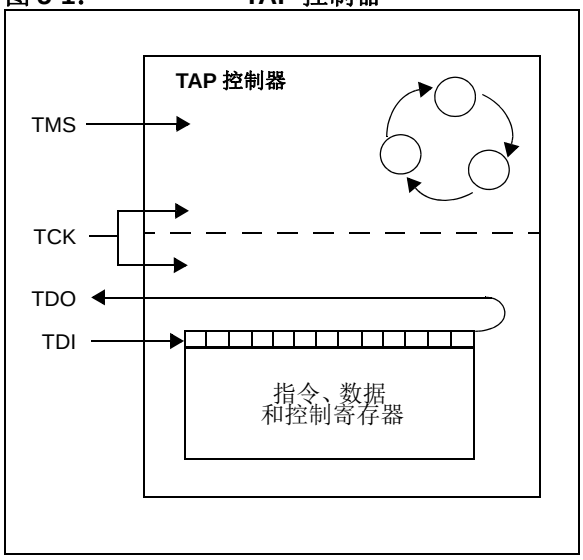
2： 不是所有器件都有稳压器使能和稳压器禁止模式。请参见具体器件的数据手册，以确定可用性。

5.0 EJTAG 与 ICSP

编程通过 CPU 内核中的 EJTAG 模块来完成。 EJTAG 与一组完整的 JTAG 引脚连接，或者与用于 ICSP 模式的精简的 2 线 /4 线转换 EJTAG 接口连接。在两种模式下，PIC32 闪存的编程都通过 ETAP 控制器来完成。TAP 控制器使用 TMS 引脚来确定是否应访问移位路径中 TDI 和 TDO 之间的指令或数据寄存器 （见图 5-1）。

EJTAG 用于编程的基本概念是使用一个称为 DMSEG（0xFF200000 至 0xFF2FFFFF）的特殊存储器区域，该区域仅在处理器以调试模式运行时可用。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并由 CPU 执行。指令按 32 位一组的形式送入 ETAP 状态机。

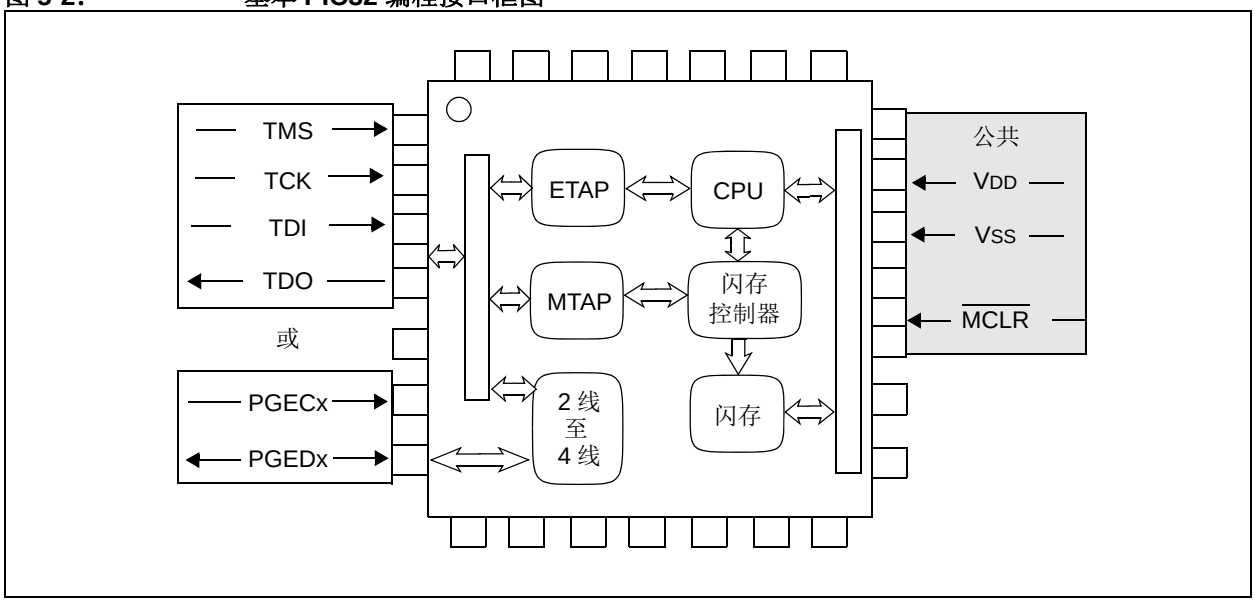
图 5-1： TAP 控制器



5.1 编程接口

图 5-2 显示了 PIC32 器件中的基本编程接口。以下子章节将对每个接口模块进行说明。

图 5-2： 基本 PIC32 编程接口框图



5.1.1 ETAP

该模块将指令和数据串行送入 CPU。

5.1.2 MTAP

除了 EJTAG TAP（ETAP）控制器之外，PIC32 器件还使用第二个专有的 TAP 控制器来进行附加操作。 Microchip TAP （MTAP）控制器支持两条与编程有关的指令：MTAP_COMMAND 和 TAP 切换指令。请参见表 19-1 以获取指令的完整列表。MTAP_COMMAND 指令为 JTAG 探针提供了通过其数据寄存器向器件发送命令的机制。

编程器发送命令的方式是：通过 SendCommand 伪操作移入 MTAP_COMMAND 指令，然后通过 XferData 伪操作发送 MTAP_COMMAND DR 命令。关于具体命令，请参见表 19-2。

探针不需要针对移入数据寄存器的每条命令都发出 MTAP_COMMAND 指令。

5.1.3 2 线 /4 线转换

该模块将 2 线 ICSP 接口转换为 4 线 JTAG 接口。

5.1.4 CPU

CPU 通过内部振荡器以 8 MHz 的速率执行指令。

5.1.5 闪存控制器

闪存控制器用于控制器件上闪存的擦除和编程。

5.1.6 闪存

PIC32 器件闪存分为两个逻辑闪存分区，包括引导闪存（Boot Flash Memory， BFM）和程序闪存 （Program Flash Memory， PFM）。BFM 从地址 0x1FC00000 开始，而 PFM 从地址 0x1D000000 开始。每个闪存分区细分为多个页，这是可擦除的最小存储器区块。根据不同的器件型号，页大小可以为 256 字 （1024 字节）、1024 字 （4096 字节）或 4096 字 （16,384 字节）。行大小指示行编程命令可以编程的字数。每页都为 8 行，因此页大小为 256、1024 和 4096 字的器件，其行大小分别为 32、128 和 512 字。表 5-1 显示了每款器件系列的 PFM、 BFM、行和页大小。

BFM 的存储单元保留用作器件配置寄存器，更多信息请参见**第 18.0 节 “配置存储器和器件 ID”**。

表 5-1：　　代码存储区容量

PIC32 器件	行大小（字）	页大小（字）	引导闪存地址 （字节）（见注 1）	编程执行程序（见注 2 和 3）
PIC32MX 110/120/130/150/170（仅 28/36/44 引脚器件）	32	256	0x1FC00000-0x1FC00BFF (3 KB)	RIPE_11_aabbcc.hex
PIC32MX 210/220/230/250/270（仅 28/36/44 引脚器件）				
PIC32MX 120/130/150/170/230/250/270/530/550/570（仅 64/100 引脚器件）				
PIC32MX 330/350/370/430/450/470	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	RIPE_06_aabbcc.hex
PIC32MX 320/340/360/420/440/460				
PIC32MX 534/564/664/764				
PIC32MX 575/675/695/795				
PIC32MM 0016/0032/0064	256	2048	0x1FC00000-0x1FC016FF (5.75K)	RIPE_20_aabbcc.hex
PIC32MK 0128/0256/0512/1024	512	4096	0x1FC00000-0x1FC04FFF (20 KB) 0x1FC20000-0x1FC24FFF (20 KB)	RIPE_15_aabbcc.hex
PIC32MZ 0256/0512/1024/2048	512	4096	0x1FC00000-0x1FC13FFF (80 KB) 0x1FC20000-0x1FC33FFF (80 KB)	RIPE_15_aabbcc.hex

注　1： 程序闪存地址范围基于程序闪存大小，如下所示：

- 0x1D000000-0x1D003FFF (16 KB)
- 0x1D000000-0x1D007FFF (32 KB)
- 0x1D000000-0x1D00FFFF (64 KB)
- 0x1D000000-0x1D01FFFF (128 KB)
- 0x1D000000-0x1D03FFFF (256 KB)
- 0x1D000000-0x1D07FFFF (512 KB)
- 0x1D000000-0x1D0FFFFF (1024 KB)
- 0x1D000000-0x1D1FFFFF (2048 KB)

不是每款系列均支持所有程序闪存大小。

2： 编程执行程序可从 Microchip 网站的相关产品页面上获取，或者位于下列 MPLAB® X IDE 安装文件夹中：
…\Microchip\MPLABX\mplab_ide\mplablibs\modules\ext\REALICE.jar
…\Microchip\MPLABX\mplab_ide\mplablibs\modules\ext\ICD3.jar
…\Microchip\MPLABX\mplab_ide\mplablibs\modules\ext\PICKIT3.jar

3： 文件名的最后几个字符 （即， aabbcc）因文件版本而异。

5.2 4 线 JTAG 细节

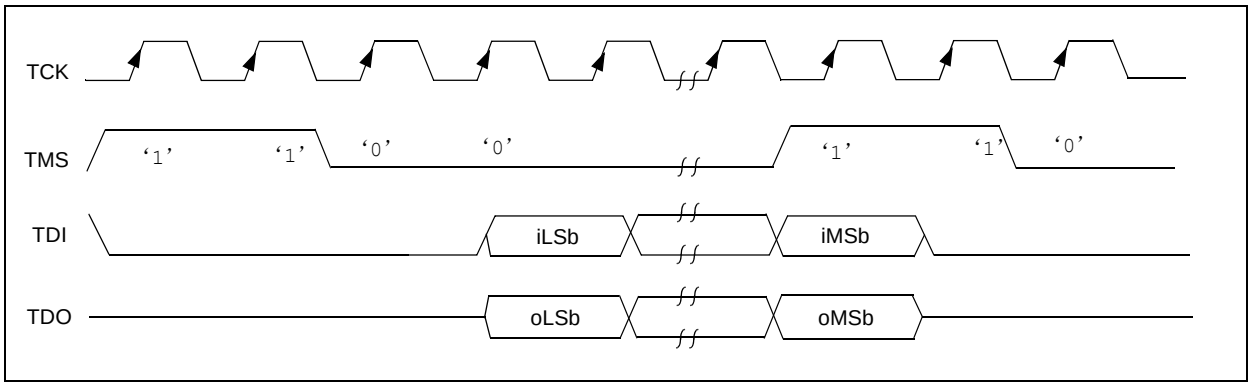
4 线接口使用标准的 JTAG （IEEE 1149.1-2001）接口信号。

- TCK：测试时钟 —— 驱动数据输入 / 输出
- TMS：测试模式选择 —— 选择工作模式
- TDI：测试数据输入 —— 数据移入器件
- TDO：测试数据输出 —— 数据移出器件

由于只有一条数据线可用，所以协议必须采用串行传输方式（类似于 SPI）。时钟输入位于 TCK 引脚上。执行配置的方法是通过 TMS 引脚来逐位操控状态机。在每个 TCK 时钟脉冲，TDI 和 TDO 引脚上会传入和传出一位数据。通过装入不同的指令模式，可以读取芯片 ID 或操控芯片功能。

送到 TDI 上的数据必须在 TCK 上升沿之前（建立时间）和之后（保持时间）保持一定的有效时间，具体时间取决于芯片。TDO 数据在 TCK 下降沿之后会保持一定的有效时间，具体时间取决于芯片，见图 5-3。

图 5-3： 4 线 JTAG 接口



5.3 2 线 ICSP 细节

在 ICSP 模式下，2 线 ICSP 信号通过时间复用的方式移入 2 线 /4 线转换模块。然后， 2 线 /4 线转换模块会对信号进行转换，使之对于 TAP 控制器相当于一个 4 线 JTAG 端口。下列为两种可能的工作模式：

- 4 相 ICSP
- 2 相 ICSP

5.3.1 4 相 ICSP

在 4 相 ICSP 模式下，TDI、TDO 和 TMS 器件引脚在 4 个时钟内在 PGEDx 上进行复用，请参见图 5-4。最低有效位 （Least Significant bit， LSb）先移； TDI 和 TMS在PGECx下降沿进行采样，而TDO同时在PGECx下降沿进行驱动。 4 相 ICSP 模式同时用于读写数据传输。

5.3.2 2 相 ICSP

在 2 相 ICSP 模式下，TMS 和 TDI 器件引脚在 2 个时钟内在 PGEDx 上进行复用，请参见图 5-5。 LSb 先移；TDI 和 TMS 在 PGECx 下降沿进行采样。该模式下不提供 TDO 输出。2 相 ICSP 模式旨在用于加快 2 线只写事务的速度。

注： 只有到下一个数据包的第一个时钟时，才会实际执行数据包。要进入 2 线、2 相 ICSP 模式， TDOEN 位 （DDPCON<0>）必须设置为 0。

图 5-4： 2 线、 4 相

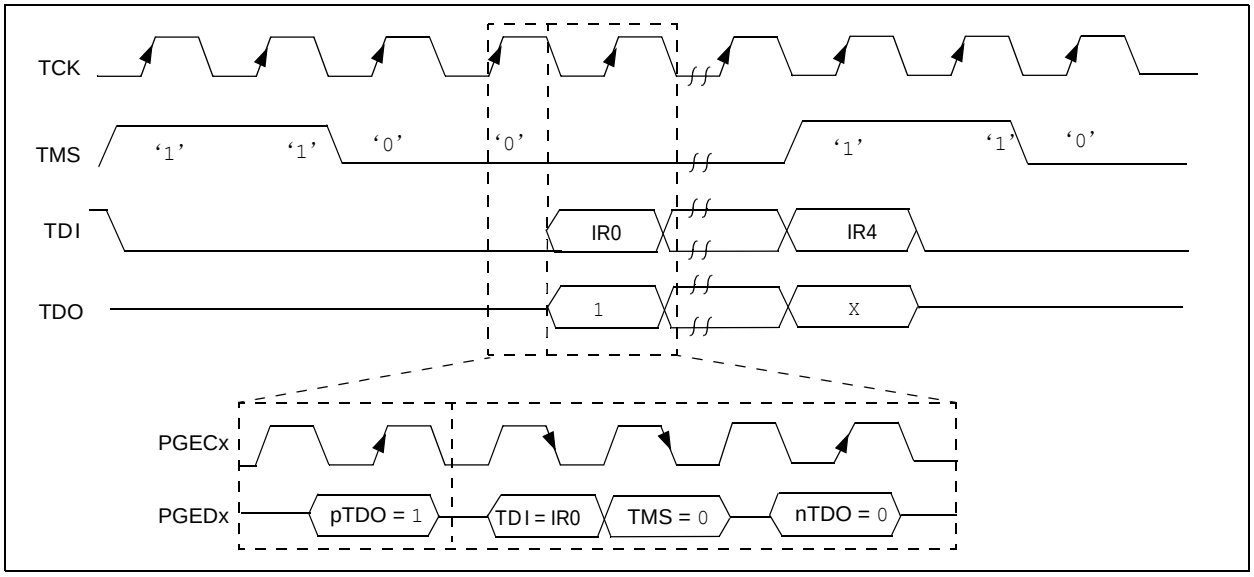
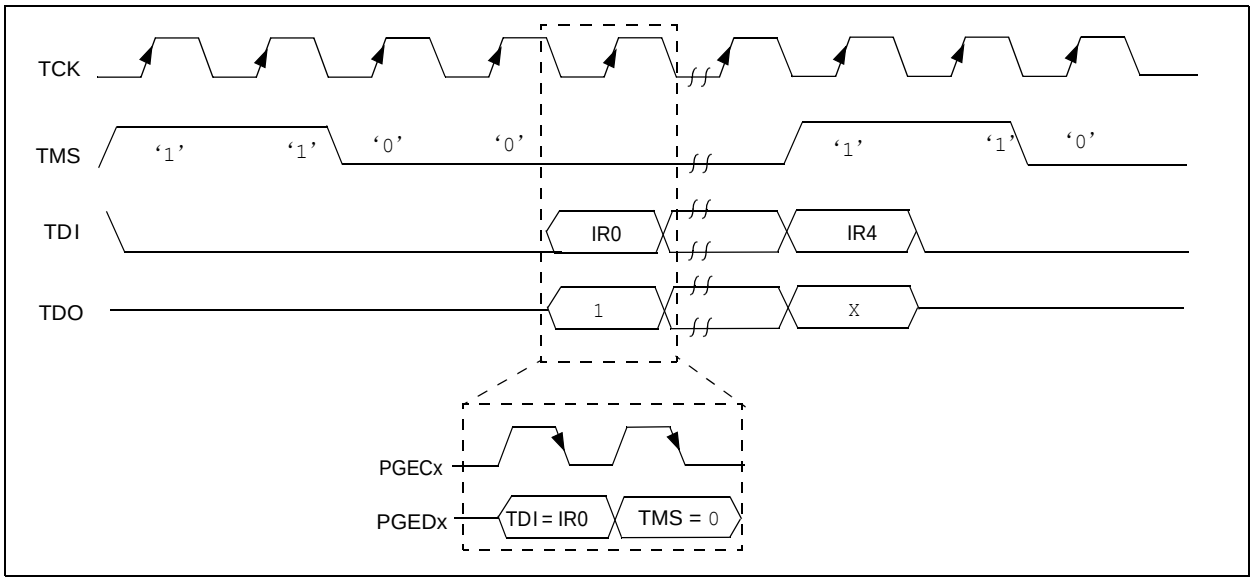


图 5-5： 2 线、 2 相



6.0 伪操作

为了简化编程细节的描述，所有操作都将使用伪操作进行描述。伪代码描述中使用了几个函数。使用这些函数是为了提高伪代码的可读性、抽象出特定于实现的行为，或者两者兼有。向伪操作传递参数时，将使用以下语法：

- 5'h0x03—— 发送 5 位十六进制值 3
- `6'b011111`—— 发送 6 位二进制值 31

本节定义了这些函数，包括以下操作：

- **SetMode** (mode)
- **SendCommand** (command)
- oData = **XferData** (iData)
- oData = **XferFastData** (iData)
- oData = **`XferInstruction`** (instruction)

6.1 `SetMode` 伪操作

格式：
SetMode (mode)

目的：
将 EJTAG 状态机设置为特定状态。

说明：
模式值在信号 TMS 上移入器件。TDI 设置为 0，TDO 会被忽略。

限制：
无。

示例：
SetMode (`6'b011111`)

图 6-1： SetMode 4 线

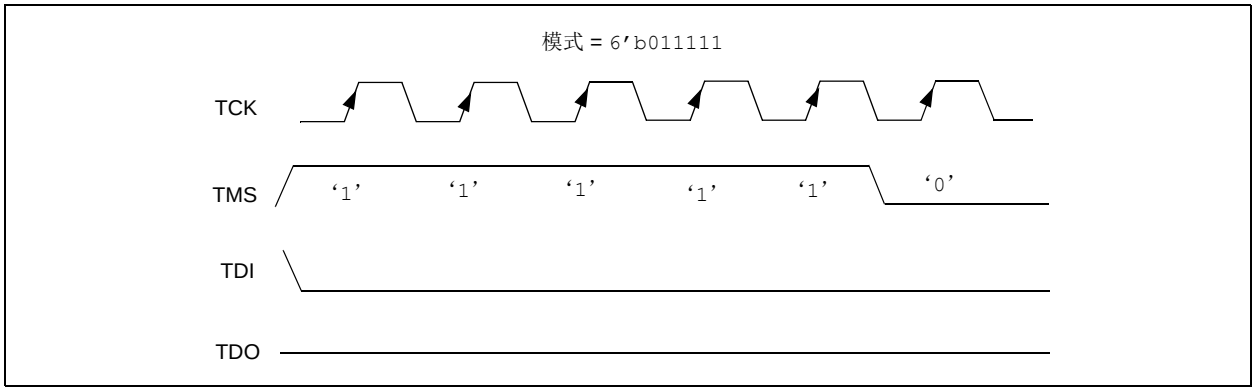
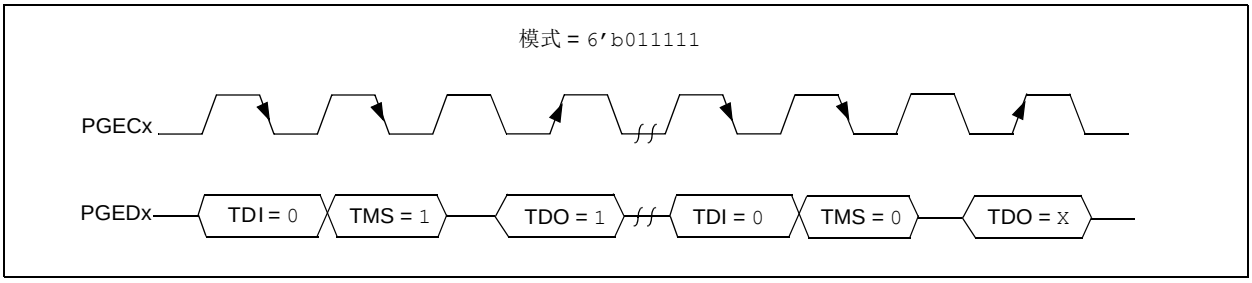


图 6-2： SetMode 2 线



6.2 SendCommand 伪操作

格式：

SendCommand (command)

目的：

发送一条命令来选择特定 TAP 寄存器。

说明 （按顺序）：

1. 将TMS头部移入器件，用于选择移位IR状态。
2. 在将信号 TMS 保持为低电平时，在 TDI 上将命令移入器件。
3. 在将TMS设置为高电平时，移入命令的最后一个最高有效位（Most Significant bit， MSb）。
4. 在 TMS 上移入 TMS 尾部，使 TAP 控制器恢复为运行 / 测试空闲状态。

限制：

无。

示例：

SendCommand (5'h0x07)

图 6-3： SendCommand 4 线

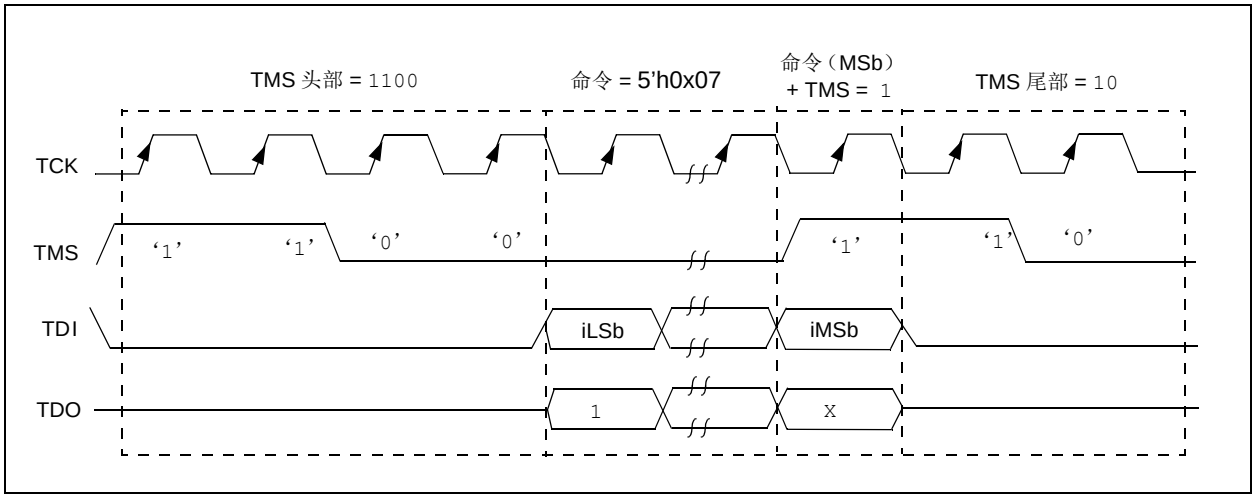
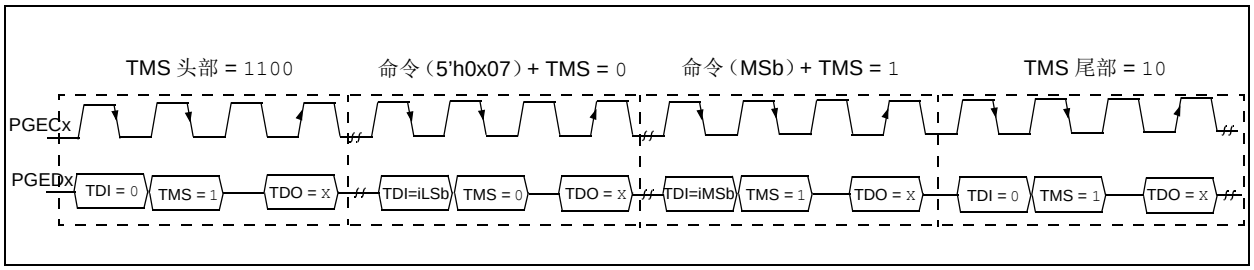


图 6-4： SendCommand 2 线 （4 相）



6.3 XferData 伪操作

格式：
oData = **XferData** (iData)

目的：
与通过命令选择的寄存器进行数据传输。

说明 （按顺序）：
1. 将TMS头部移入器件，用于选择移位DR状态。
2. 在将信号 TMS 保持为低电平时，在 TDI/TDO 上将数据移入 / 移出器件。
3. 在将 TMS 设置为高电平时，移入 / 移出数据的最后一个 MSb。
4. 在 TMS 上移入 TMS 尾部，使 TAP 控制器恢复为运行 / 测试空闲状态。

限制：
无。

示例：
oData = XferData (32'h0x12)

图 6-5： XferData 4 线

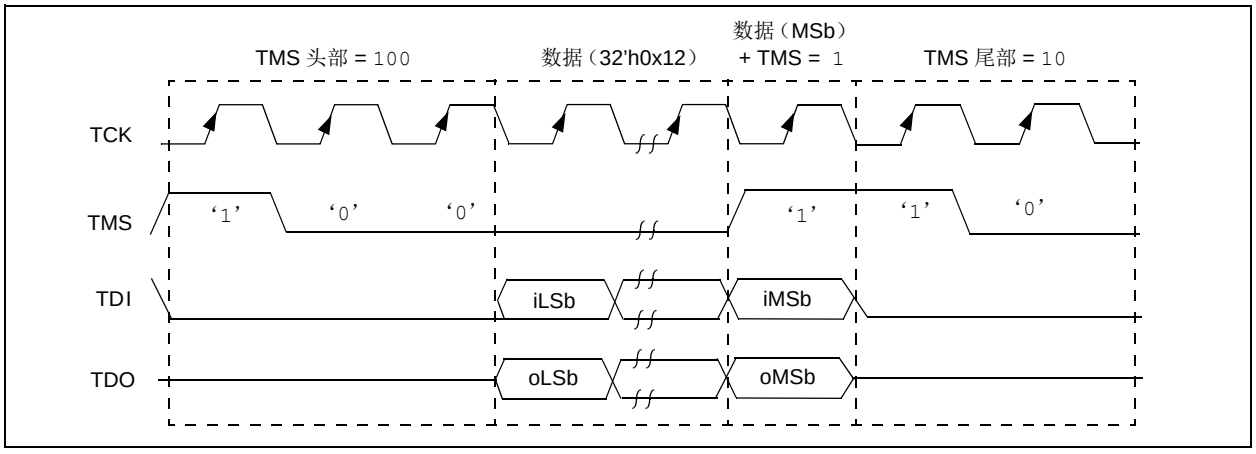
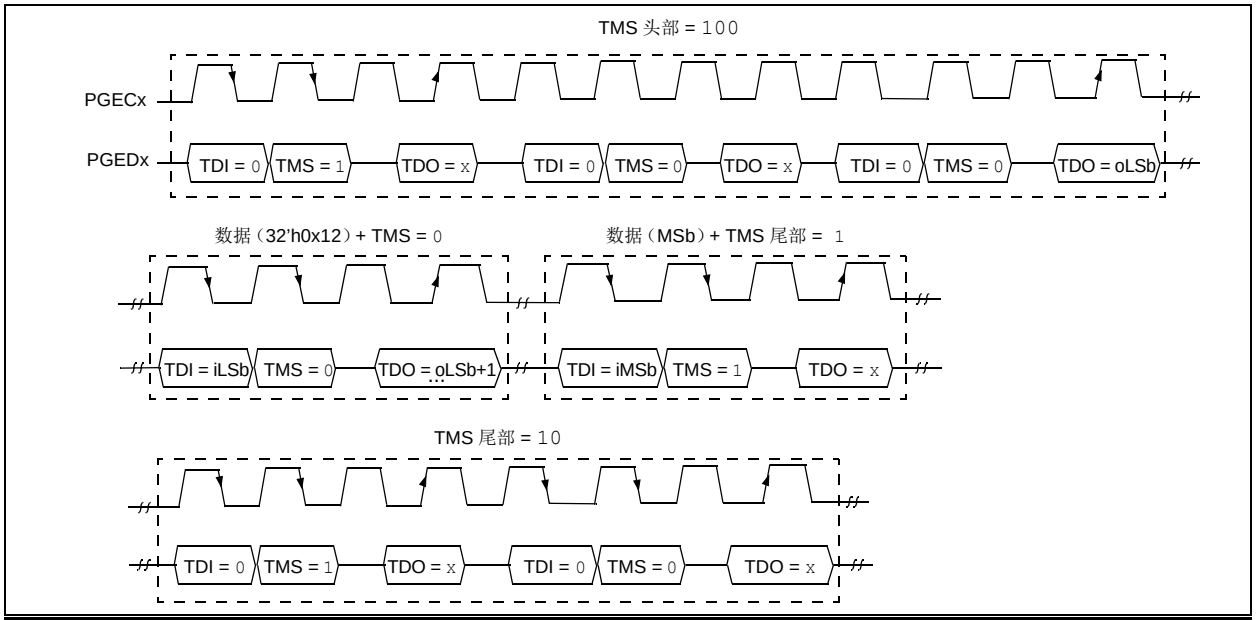


图 6-6： XferData 2 线 （4 相）



6.4 XferFastData 伪操作

格式：
oData = **XferFastData** (iData)

目的：
从器件快速移入 / 移出 32 位的数据。

说明 （按顺序）：
1. 将 TMS 头部移入器件，用于选择移位 DR 状态。

> **注：** 对于 2 线 （4 相） —— 在最后一个时钟，在移入TMS头部的同时，oPrAcc位在TDO上移出。如果 oPrAcc 的值不是 1，则必须重复整个操作。

2. 移入 PrAcc 位的输入值 （为 0）。

> **注：** 对于 2 线（4 相）—— 在该操作期间，TDO 将为输出数据的LSb。移入输入数据的其余 31 位，并移出输出数据的 31 位。对于输入数据的最后一位，将设置 TMS 尾部 = 1。

3. 移入 TMS 尾部 = 10，使 TAP 控制器恢复为运行 / 测试空闲状态。

限制：
必须先发送 SendCommand (ETAP_FASTDATA) 来选择 Fastdata 寄存器，如例 6-1 中所示。关于命令的详细说明，请参见表 19-4。

> **注：** 只有在与 PE 进行通信时，才使用 2 相 XferData。更多信息，请参见**第 16.0 节 “编程执行程序”**。

例 6-1： SendCommand

```
// Select the Fastdata Register
SendCommand(ETAP_FASTDATA)
// Send/Receive 32-bit Data
oData = XferFastData(32'h0x12)
```

图 6-7： XferFastData 4 线

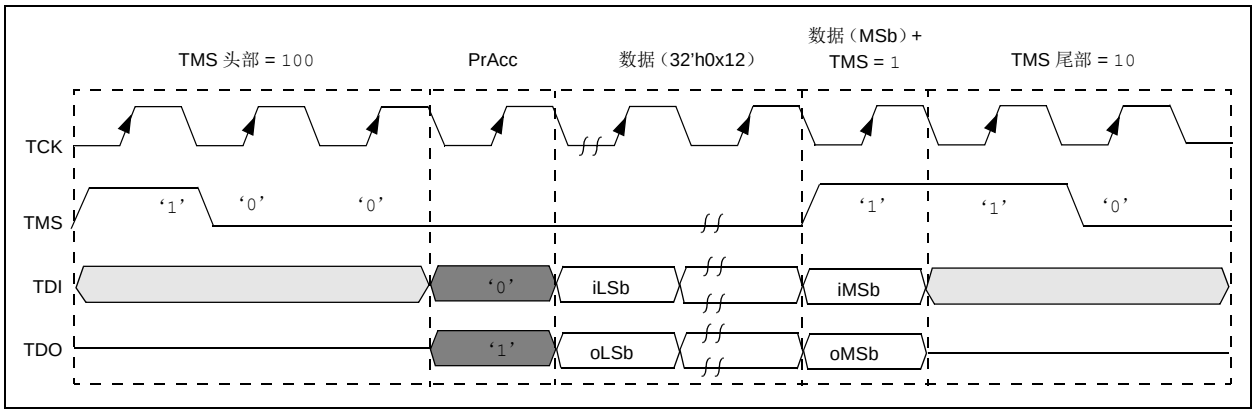


图 6-8： XferFastData 2 线 （2 相）

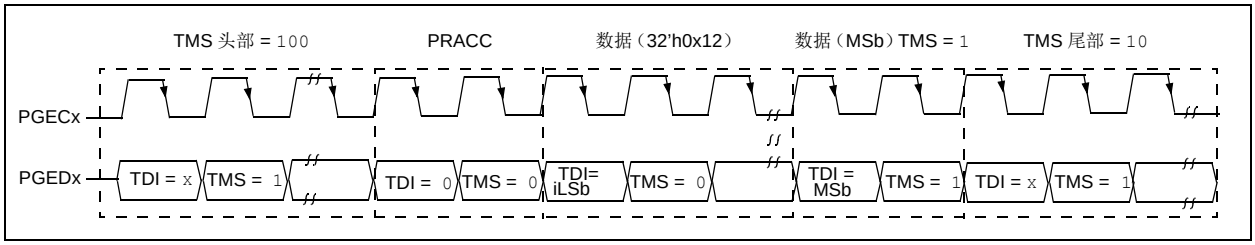
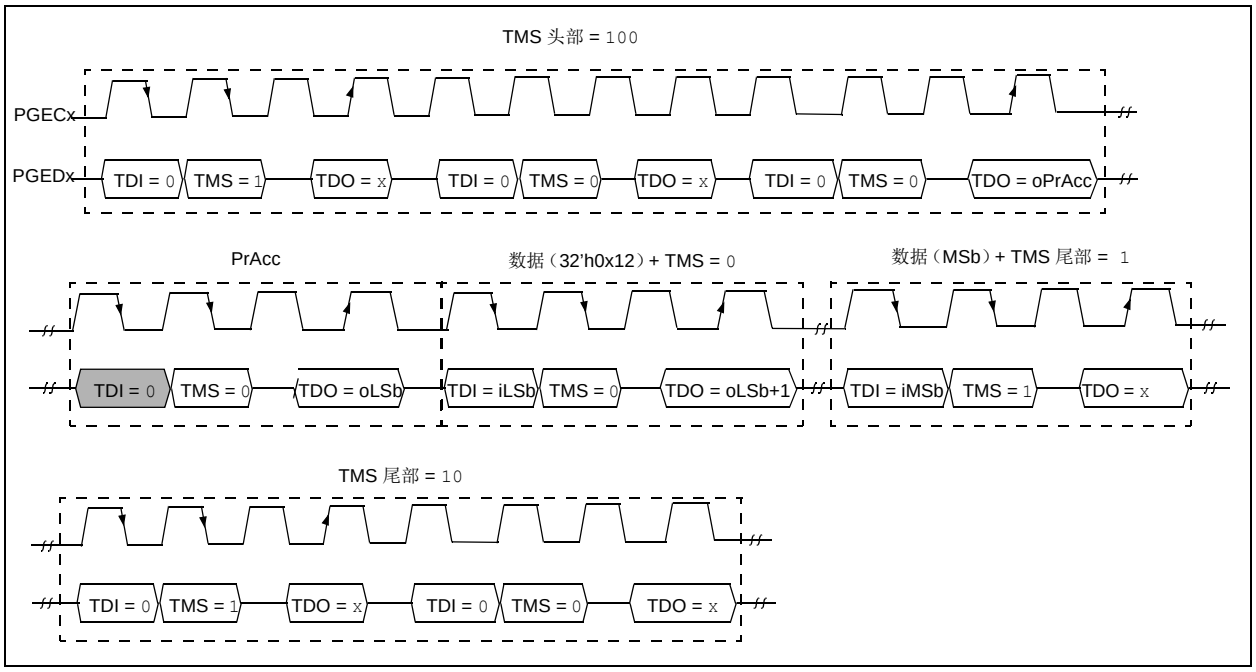


图 6-9： XferFastData 2 线 （4 相）



6.5 XferInstruction 伪操作

格式：

XferInstruction (instruction)

目的：

向器件发送要执行的 32 位数据。

说明：

先将指令移入器件，然后由 CPU 执行。

限制：

器件必须处于调试模式。

例 6-2： XferInstruction

```
XferInstruction (instruction)
{
    // Select Control Register
    SendCommand(ETAP_CONTROL);
    // Wait until CPU is ready
    // Check if Processor Access bit (bit 18) is set
    do {
        controlVal = XferData(32'h0x0004C000);
    } while( PrAcc(contorlVal<18>) is not  '1' );

    // Select Data Register
    SendCommand(ETAP_DATA);

    // Send the instruction
    XferData(instruction);

    // Tell CPU to execute instruction
    SendCommand(ETAP_CONTROL);
    XferData(32'h0x0000C000);
}
```

6.6 ReadFromAddress 伪操作

格式：

oData = ReadFromAddress (address)

目的：

发送 32 位数据到器件存储器。

说明：

从存储器中由“address”参数指定的地址处读取 32 位数据。

限制：

器件必须处于调试模式。

例 6-3： ReadFromAddress（针对 PIC32MX、 PIC32MZ 和 PIC32MK 器件）

```
ReadFromAddress (address)
{

// Load Fast Data register address to s3
instruction = 0x3c130000;
instruction |= (0xff200000>>16)&0x0000ffff;
XferInstruction(instruction); // lui s3, <FAST_DATA_REG_ADDRESS(31:16)> - set address of fast
data register

// Load memory address to be read into t0
instruction = 0x3c080000;
instruction |= (address>>16)&0x0000ffff;
XferInstruction(instruction); // lui t0, <DATA_ADDRESS(31:16)> - set address of data
instruction = 0x35080000;
instruction |= (address&0x0000ffff);
XferInstruction(instruction); // ori t0, <DATA_ADDRESS(15:0)> - set address of data

// Read data
XferInstruction(0x8d090000); // lw  t1, 0(t0)

// Store data into Fast Data register
XferInstruction(0xae690000); // sw  t1, 0(s3) - store data to fast data register
XferInstruction(0); // nop

// Shift out the data
SendCommand(ETAP_FASTDATA);
oData = XferFastData(32'h0x00000000);

return oData;

}
```

例 6-4：　　ReadFromAddress （针对 PIC32MM 器件）

```
ReadFromAddress (address)
{

// Load Fast Data register address to s3
instruction = 0x000041B3;
instruction |= (0xff200000&0xffff0000);
XferInstruction(instruction); // lui s3, <FAST_DATA_REG_ADDRESS(31:16)> - set address of fast
data register

// Load memory address to be read into t0
instruction = 0x000041A8;
instruction |= (address&0xffff0000);
XferInstruction(instruction); // lui t0, <DATA_ADDRESS(31:16)> - set address of data
instruction = 0x00005108;
instruction |= (address<<16)&0xffff0000;
XferInstruction(instruction); // ori t0, <DATA_ADDRESS(15:0)> - set address of data

// Read data
XferInstruction(0x0000FD28); // lw  t1, 0(t0)

// Store data into Fast Data register
XferInstruction(0x0000F933); // sw  t1, 0(s3) - store data to fast data register
XferInstruction(0x0C000C00); // 2 nops

// Shift out the data
SendCommand(ETAP_FASTDATA);
oData = XferFastData(32'h0x00000000);

return oData;

}
```

7.0 进入 2 线增强型 ICSP 模式

为了使用 2 线 PGEDx 和 PGECx 引脚进行编程，必须将其使能。请注意，可能会使用到特定器件上的可用编程引脚对，而且它们必须成对使用。例如 PGED1 必须与 PGEC1 一起使用，以此类推。

注：	如果使用 4 线 JTAG 接口，则以下步骤不是必需的。

进入 2 线增强型 ICSP 模式需要执行以下步骤：

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 位的密钥序列在时钟控制下移入到 PGEDx 引脚。
3. 然后将MCLR驱动为高电平并保持一段指定的时间。

时序要求请参见**第 20.0 节 “交流 / 直流特性和时序要求”**。

施加到 MCLR 引脚上的编程电压为 VIH，对于 PIC32 器件，该电压实际就是 VDD。对于 VIH 没有最短保持时间的要求。在 VIH 移除后，必须经过至少 P18 个时间间隔才能将密钥序列移入到 PGEDx 引脚。

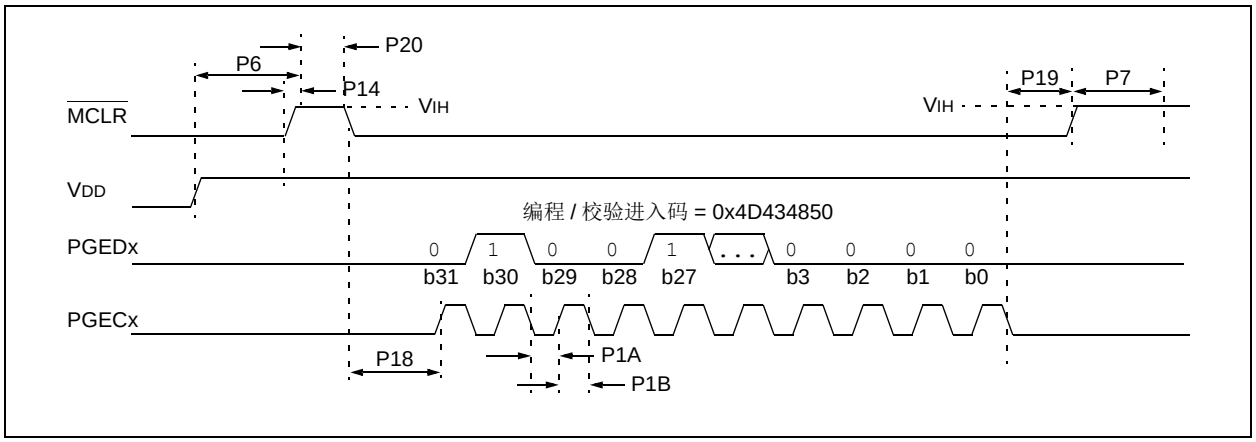
密钥序列为一个特定的32位模式：“0100 1101 0100 0011 0100 1000 0101 0000”（首字母缩写词“MCHP”，以 ASCII 表示）。只有在密钥序列有效时，器件才能进入编程 / 校验模式。首先必须移入高 4 位的 MSb。

一旦密钥序列完全移入，只要继续保持为 2 线增强型 ICSP 接口，就必须将 VIH 施加到 MCLR 上并保持该电平。必须经过至少 P19 和 P7 个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，就可以执行本文档后面各节中描述的编程操作了。在 2 线增强型 ICSP 模式下，所有未用的 I/O 引脚都被置于高阻态。

注：	进入 ICSP 模式会将器件复位，以防止指令执行。欲释放复位，必须执行 MCHP_DE_ASSERT_RST 命令。

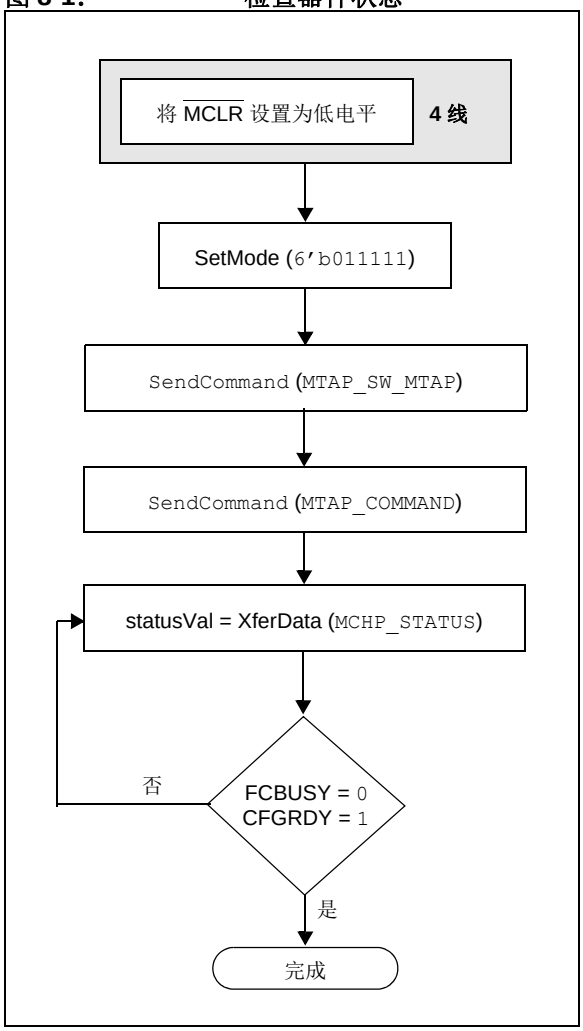
图 7-1： 进入增强型 ICSP™ 模式



8.0 检查器件状态

在对器件进行编程之前，编程器必须先检查器件的状态，确保它准备好接收信息。

图 8-1：　　检查器件状态



8.1 4 线接口

4 线 JTAG 编程是一种任务模式操作，因此用于开始编程的设置序列应当在 MCLR 置为有效时执行。将器件保持在复位状态可以防止处理器执行指令或驱动端口。

使用 4 线接口检查器件状态需要执行以下步骤：

1. 将 MCLR 引脚设置为低电平。
2. 通过 SetMode (6'b011111) 强制芯片 TAP 控制器进入运行测试 / 空闲状态。
3. SendCommand (MTAP_SW_MTAP)。
4. SendCommand (MTAP_COMMAND)。
5. statusVal = XferData (MCHP_STATUS)。
6. 如果CFGRDY（statusVal<3>）不为1且FCBUSY（statusVal<2>）不为 0，则转到步骤 5。

注： 如果使用 4 线接口，则由配置字选择的振荡器源必须存在以访问闪存。在未编程器件中，振荡器源为内部 FRC，允许进行闪存访问。如果配置字被重新设置为选择外部振荡器源，则其必须存在以进行闪存访问。关于使用配置字设定来选择振荡器的更多详情，请参见具体器件数据手册的**“特殊功能”**章节。

8.2 2 线接口

使用 2 线接口检查器件状态需要执行以下步骤：

1. 通过 SetMode (6'b011111) 强制芯片 TAP 控制器进入运行测试 / 空闲状态。
2. SendCommand (MTAP_SW_MTAP)。
3. SendCommand (MTAP_COMMAND)。
4. statusVal = XferData (MCHP_STATUS)。
5. 如果CFGRDY（statusVal<3>）不为1且FCBUSY（statusVal<2>）不为 0，则转到步骤 4。

注： 如果 CFGRDY 和 FCBUSY 未在 10 ms 内变为正确状态，则说明执行的序列可能不正确或器件已损坏。

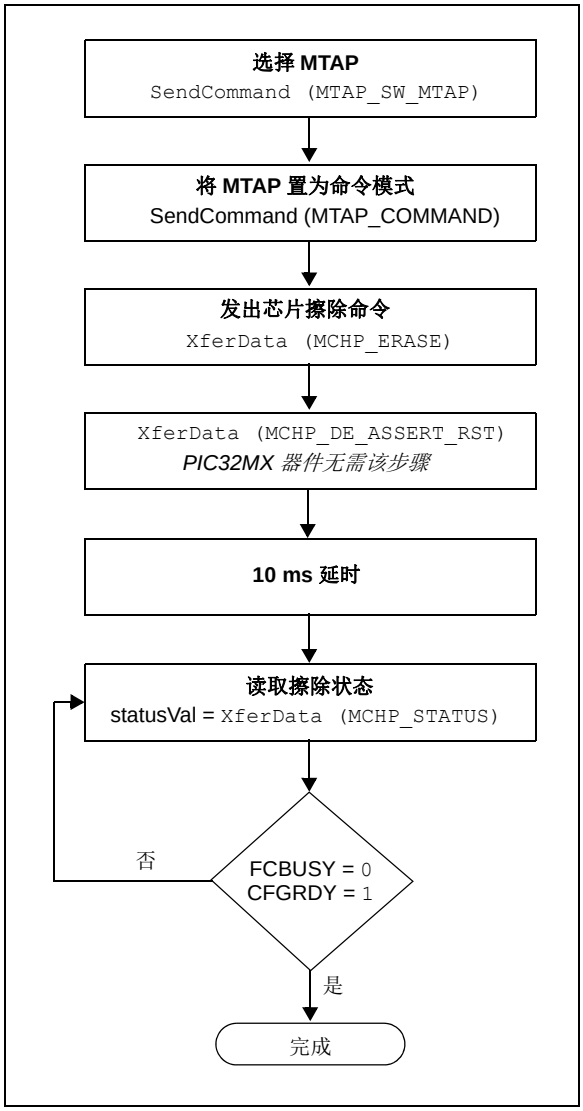
9.0 擦除器件

在对器件进行编程之前，必须先进行擦除。擦除操作会向闪存中写入全 1，并使之准备好烧写一组新数据。擦除器件之后，可以通过执行“空白检查”操作来进行校验。更多信息，请参见**第 9.1 节 “空白检查”**。

擦除程序存储器（程序、引导和配置存储区）的过程包括选择 MTAP 和发送 MCHP_ERASE 命令。然后，编程器必须通过读取并校验 MCHP_STATUS 值中的位，等待擦除操作完成。图9-1显示了执行芯片擦除操作的过程。

注：	器件 ID 存储单元是只读的，不能擦除。因此，芯片擦除对这些存储单元没有影响。

图 9-1： 擦除器件



擦除目标器件需要执行以下步骤：

1. SendCommand (MTAP_SW_MTAP)。
2. SendCommand (MTAP_COMMAND)。
3. XferData (MCHP_ERASE)。
4. XferData (MCHP_DE_ASSERT_RST)。
   PIC32MX 器件无需该步骤。
5. 延时 10 ms。
6. statusVal = XferData (MCHP_STATUS)。
7. 如果CFGRDY（statusVal<3>）不为1且FCBUSY（statusVal<2>）不为 0，则转到步骤 5。

注：	芯片擦除操作是自定时操作。如果FCBUSY 和 CFGRDY 位未在规定的芯片擦除时间内正确设置，则说明执行的序列可能不正确或器件已损坏。

9.1 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。

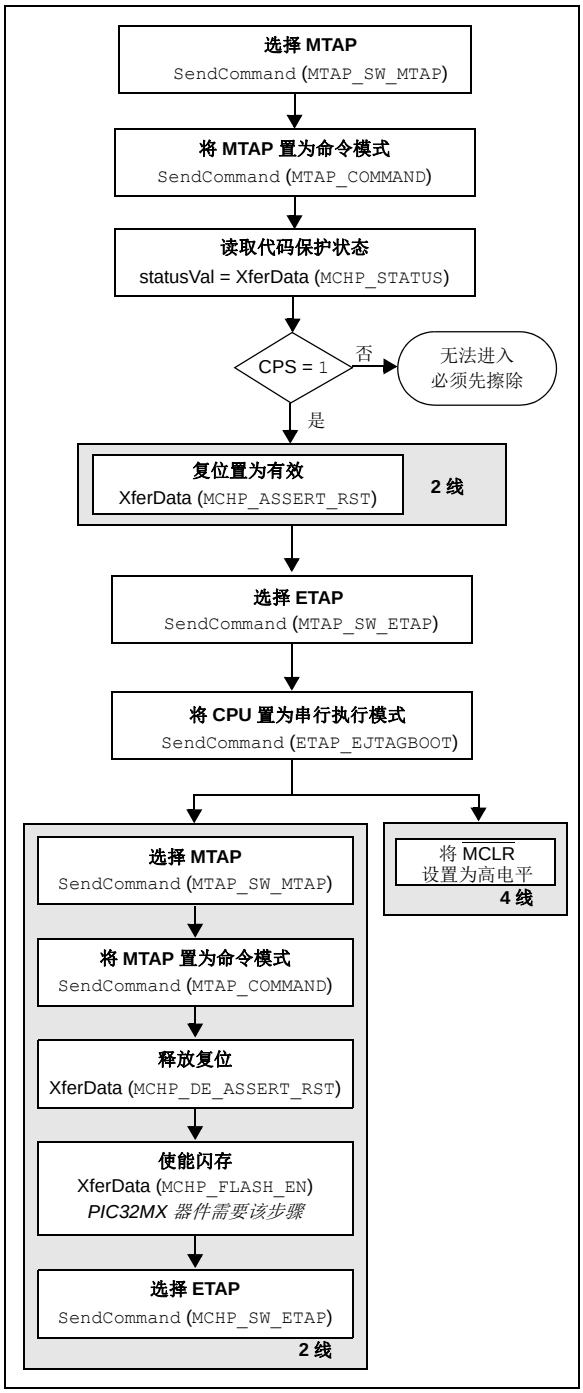
器件配置寄存器会被空白检查忽略。另外，所有未实现的存储空间都应该被空白检查忽略。

10.0 进入串行执行模式

在对器件进行编程之前，必须先将器件置为串行执行模式。

进入串行执行模式的过程包括验证器件未被写保护。如果器件被写保护，则必须执行芯片擦除操作。详情请参见**第 9.0 节 “擦除器件”**。

图 10-1： 进入串行执行模式



10.1 4 线接口

进入串行执行模式需要执行以下步骤：

注：	假设之前的检查器件状态步骤将 MCLR 驱动为低电平 （见图 8-1）。

1. SendCommand (MTAP_SW_MTAP)。
2. SendCommand (MTAP_COMMAND)。
3. statusVal = XferData (MCHP_STATUS)。
4. 如果 CPS （statusVal<7>）不为 1，则必须先擦除器件。
5. SendCommand (MTAP_SW_ETAP)。
6. SendCommand (ETAP_EJTAGBOOT)。
7. 将 MCLR 设置为高电平。

10.2 2 线接口

进入串行执行模式需要执行以下步骤：

1. SendCommand (MTAP_SW_MTAP)。
2. SendCommand (MTAP_COMMAND)。
3. statusVal = XferData (MCHP_STATUS)。
4. 如果 CPS （statusVal<7>）不为 1，则必须先擦除器件。
5. XferData (MCHP_ASSERT_RST)。
6. SendCommand (MTAP_SW_ETAP)。
7. SendCommand (ETAP_EJTAGBOOT)。
8. SendCommand (MTAP_SW_MTAP)。
9. SendCommand (MTAP_COMMAND)。
10. XferData (MCHP_DE_ASSERT_RST)。
11. XferData (MCHP_FLASH_ENABLE)。 *PIC32MX 系列器件需要该步骤。*
12. SendCommand (MTAP_SW_ETAP)。

11.0 下载编程执行程序 （PE）

PE 驻留在 RAM 存储器中， CPU 通过执行它来对器件进行编程。 PE 为编程器提供了一种使用简单命令集和通信协议来编程和校验 PIC32 器件的机制。 PE 可提供以下几项基本功能：

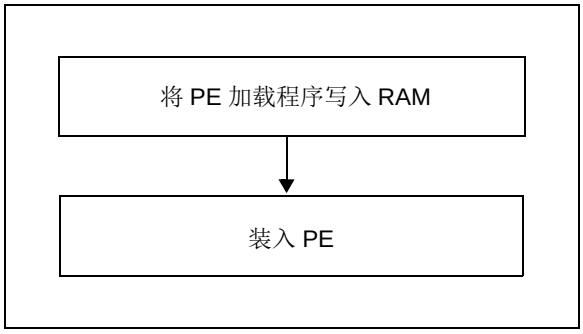
- 读存储器
- 擦除存储器
- 对存储器编程
- 空白检查
- 读执行程序固件版本
- 获取闪存单元的循环冗余校验（Cyclic Redundancy Check， CRC）

PE 执行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。关于每条命令的详细说明，请参见**第 16.2 节 “PE 命令集”**。

PE 使用器件的数据 RAM 来存放变量和编程执行程序。运行 PE 之后，数据 RAM 中的内容就无法确定了。

将 PE 装入数据 RAM 中之后，可以使用表 16-1 中列出的命令集对 PIC32 系列进行编程。

图 11-1： 下载 PE



将 PE 装入存储器是一个分为两步的过程：

1. 将 PE 加载程序装入数据 RAM。（PE 加载程序会将 PE 二进制文件装入数据 RAM 的适当位置，并在完成时跳转到编程执行程序处，并开始执行它。）
2. 将 PE 二进制数据送到 PE 加载程序。

表 11-1 和表 11-3 列出了下载 PE 所需的步骤。

表 11-1： 下载 PE （仅针对 PIC32MX、 PIC32MZ 和 PIC32MK 器件）

操作	操作数

步骤 1： **仅 PIC32MX 器件：**将 BMXCON 初始化为 0x1F0040。由 PIC32 内核执行的指令序列如下：
`lui a0,0xbf88`
`ori a0,a0,0x2000 /* address of BMXCON */`
`lui a1,0x1f`
`ori a1,a1,0x40 /* $a1 has 0x1f0040 */`
`sw a1,0(a0) /* BMXCON initialized */`	

`XferInstruction`	`0x3c04bf88`
`XferInstruction`	`0x34842000`
`XferInstruction`	`0x3c05001f`
`XferInstruction`	`0x34a50040`
`XferInstruction`	`0xac850000`

步骤 2： **仅 PIC32MX 器件：**将 BMXDKPBA 初始化为 0x800。由 PIC32 内核执行的指令序列如下：
`li a1,0x800`
`sw a1,16(a0)`	

`XferInstruction`	`0x34050800`
`XferInstruction`	`0xac850010`

步骤 3： **仅 PIC32MX 器件：**将 BMXDUDBA 和 BMXDUPBA 初始化为 BMXDRMSZ 的值。由 PIC32 内核执行的指令序列如下：
`lw a1,64(a0) /* load BMXDMSZ */`
`sw a1,32(a0)`
`sw a1,48(a0)`	

`XferInstruction`	`0x8C850040`
`XferInstruction`	`0xac850020`
`XferInstruction`	`0xac850030`

步骤 4： 设置 PE 的 PIC32 RAM 地址。由 PIC32 内核执行的指令序列如下：
`lui a0,0xa000`
`ori a0,a0,0x800`	

`XferInstruction`	`0x3c04a000`
`XferInstruction`	`0x34840800`

表 11-1：　下载 PE（仅针对 PIC32MX、 PIC32MZ 和 PIC32MK 器件）（续）

操作	操作数

步骤 5： 装入 PE_Loader。重复该步骤（步骤 5），直到整个 PE_Loader 都装入 PIC32 存储器为止。在操作数字段中，“<PE_loader hi++>”代表表 11-2 中所列 PE 加载程序操作码的 MSb 31 至 16。类似地，“<PE_loader lo++>”代表表 11-2 中所列 PE 加载程序操作码的 LSb 15 至 0。“++”符号指示在连续执行这些操作时，需要从表 11-2 中所列的 PE 加载程序操作码列表中传输新的字。由 PIC32 内核执行的指令序列如下：
`lui a2, <PE_loader hi++>`
`ori a2,a2, <PE_loader lo++>`
`sw a2,0(a0)`
`addiu a0,a0,4`	

XferInstruction	(0x3c06 <PE_loader hi++>)
XferInstruction	(0x34c6 <PE_loader lo++>)
XferInstruction	0xac860000
XferInstruction	0x24840004

步骤 6： 跳转到 PE_Loader。由 PIC32 内核执行的指令序列如下：
`lui t9,0xa000`
`ori t9,t9,0x800`
`jr t9`
`nop`	

XferInstruction	0x3c19a000
XferInstruction	0x37390800
XferInstruction	0x03200008
XferInstruction	0x00000000
步骤 7： 使用 PE_Loader 装入 PE。重复该步骤（步骤 7）的最后一条指令，直到整个 PE 都装入 PIC32 存储器为止。在该步骤中，将会提供一个 Intel® Hex 格式的 PE 文件，您需要每次解析一定数量的 32 位字并将它们传输到 PIC32 存储器（见**附录 B：“Hex 文件格式”**）。表 11-2 中列出了由 PIC32 执行的指令序列。	
SendCommand	ETAP_FASTDATA
XferFastData	PE_ADDRESS (Address of PE program block from PE Hex file)
XferFastData	PE_SIZE (Number of 32-bit words of the program block from PE Hex file)
XferFastData	PE software op code from PE Hex file (PE Instructions)
步骤 8： 跳转到 PE。幻数（0xDEAD0000）指示 PE_Loader， PE 已完全装入存储器中。当 PE_Loader 发现幻数时，它会跳转到 PE。	
XferFastData	0x00000000
XferFastData	0xDEAD0000

表 11-2：　PE 加载程序操作码（仅针对 PIC32MX、 PIC32MZ 和 PIC32MK 器件）

操作码	指令
0x3c07dead	lui a3, 0xdead
0x3c06ff20	lui a2, 0xff20
0x3c05ff20	lui a1, 0xff20
	here1:
0x8cc40000	lw a0, 0 (a2)
0x8cc30000	lw v1, 0 (a2)
0x1067000b	beq v1, a3, <here3>
0x00000000	nop
0x1060fffb	beqz v1, <here1>
0x00000000	nop
	here2:
0x8ca20000	lw v0, 0 (a1)
0x2463ffff	addiu v1, v1, -1
0xac820000	sw v0, 0 (a0)
0x24840004	addiu a0, a0, 4
0x1460fffb	bnez v1, <here2>
0x00000000	nop
0x1000fff3	b <here1>
0x00000000	nop
	here3:
0x3c02a000	lui v0, 0xa000
0x34420900	ori v0, v0, 0x900
0x00400008	jr v0
0x00000000	nop

表 11-3：　下载 PE（仅针对 PIC32MM 器件）

操作	操作数

步骤 1： 将程序 RAM 地址初始化为 0x800。由 PIC32 内核执行的指令序列如下：
`lui a0, 0xbf80 /* upper 16 bit address of CFGCON */`
`lw a1, 0x03b0(a0) /* load CFGCON to a1*/`
`li a2, 0x0002 /* BMXDKBPA = 2, program RAM starts from 0x0800*/`
`ins a1, a2, 16, 8 /* modify BMXDKBPA field*/`
`sw a1, 0x03b0(a0) /* write CFGCON back*/`
`nop`	

XferInstruction	0x03B0FCA4
XferInstruction	0x00A6EF02
XferInstruction	0xF8A4BC0C
XferInstruction	0x0C0003B0

步骤 2： 设置 PE 的 PIC32MM RAM 地址。由 PIC32 内核执行的指令序列如下：
`lui a0,0xa000`
`ori a0,a0,0x800`	

XferInstruction	0xA00041A4
XferInstruction	0x8005084

表 11-3：　下载 PE（仅针对 PIC32MM 器件）（续）

操作	操作数

步骤 3： 装入 PE_Loader。重复该步骤（步骤 3），直到整个 PE_Loader 都装入 PIC32 存储器为止。在操作数字段中，“PE_loader hi++>”代表表 11-4 中所列 PE 加载程序操作码的 MSb 31 至 16。类似地，“<PE_loader lo++>”代表表 11-4 中所列 PE 加载程序操作码的 LSb 15 至 0。“++”符号指示在连续执行这些操作时，需要从表 11-4 中所列的 PE 加载程序操作码列表中传输新的字。由 PIC32 内核执行的指令序列如下：
`lui a2, <PE_loader hi++>`
`ori a2,a2, <PE_loader lo++>`
`sw a2,0(a0)`
`addiu a0,a0,4`	

XferInstruction	0x<PE_loader hi++>41A6
XferInstruction	0x<PE_loader lo++>50C6
XferInstruction	0x6E42EB40

步骤 4： 跳转到 PE_Loader。由 PIC32 内核执行的指令序列如下：
`lui t9,0xa000`
`ori t9,t9,0x800`
`jr t9`
`nop`	

XferInstruction	0xA00041B9
XferInstruction	0x08005339
XferInstruction	0x0C004599
步骤 5： 使用 PE_Loader 装入 PE。重复该步骤（步骤 5）的最后一条指令，直到整个 PE 都装入 PIC32 存储器为止。在该步骤中，将会提供一个 Intel® Hex 格式的 PE 文件，您需要每次解析一定数量的 32 位字并将它们传输到 PIC32 存储器（见**附录 B：“Hex 文件格式”**）。表 11-4 中列出了由 PIC32 执行的指令序列。	
SendCommand	ETAP_FASTDATA
XferFastData	PE_ADDRESS (Address of PE program block from PE Hex file)
XferFastData	PE_SIZE (Number of 32-bit words of the program block from PE Hex file)
XferFastData	PE software op code from PE Hex file (PE Instructions)
步骤 6： 跳转到 PE。幻数（0xDEAD0000）指示 PE_Loader， PE 已完全装入存储器中。当 PE_Loader 发现幻数时，它会跳转到 PE。	
XferFastData	0x00000000
XferFastData	0xDEAD0000

表 11-4：　PE 加载程序操作码（仅针对 PIC32MM 器件）

操作码	指令
0xDEAD41A7	lui a3, 0xdead
0xFF2041A6	lui a2, 0xff20
0xFF2041A5	lui a1, 0xff20
	here1:
0x6A60	lw a0, 0 (a2)
0x69E0	lw v1, 0 (a2)
0x94E3000B	beq v1, a3, here3
0x0C00	nop
0x8DFA	beqz v1, here1
0x0C00	nop
	here2:
0x6950	lw v0, 0 (a1)
0x6DBE	addiu v1, v1, -1
0xE940	sw v0, 0 (a0)
0x6E42	addiu a0, a0, 4
0xADFB	bnez v1, here2
0x0C00	nop
0xCFF2	b here1
0x0C00	nop
	here3:
0x41A2A000	lui v0, 0xa000
0x50420900	ori v0, v0, 0x900
0x4582	jr v0
0x0C00	nop

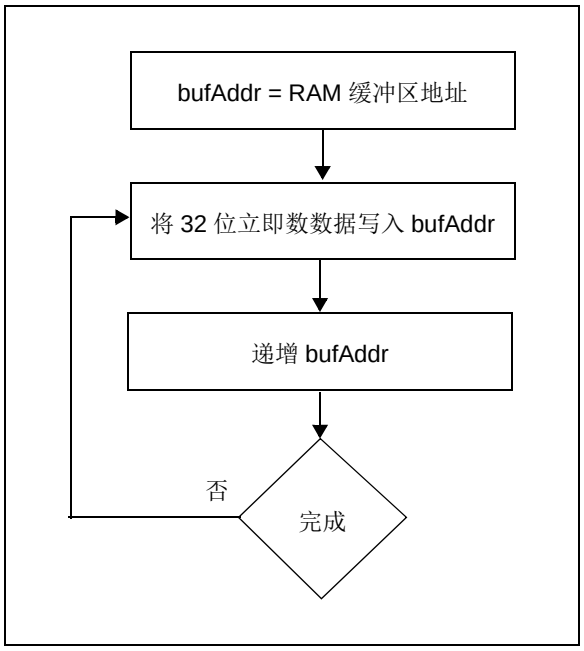
12.0 下载数据块

要将数据块烧写到 PIC32 器件中，必须将它装入 SRAM。

12.1 不使用 PE

要在不使用 PE 的情况下烧写存储器块，必须先将数据块写入 RAM。这种方法要求编程器传输带有嵌入（立即数）数据的实际机器指令，用于将数据块写入器件内部 RAM 存储器。

图 12-1： 不使用 PE 的情况下下载数据



下载数据块需要执行以下步骤：

1. `XferInstruction` (op code)。
2. 重复步骤1，直到最后一条指令传输到CPU为止。

表 12-1： 下载数据操作码（仅针对 PIC32MX、 PIC32MZ 和 PIC32MK 器件）

操作码	指令
步骤 1： 将 SRAM 基址初始化为 0xA0000000。	
3c10a000	lui $s0, 0xA000;
步骤 2： 将要编程的整行数据写入系统 SRAM。	

3c08<DATA>
3508<DATA>
ae08<OFFSET>
	lui $t0, <DATA(31:16)>;
	ori $t0, <DATA(15:0)>;
	sw $t0, <OFFSET>($s0);
	// OFFSET increments by 4

步骤 3： 重复步骤 2，直到装入一行数据为止。	

表 12-2： 下载数据操作码（仅针对 PIC32MM 器件）

操作码	指令
步骤 1： 将 SRAM 基址初始化为 0xA0000000。	
0xA00041A4	lui a0, 0xA000
步骤 2： 将要编程的整行数据烧写到系统 SRAM。	

0x<DATA(31:16)>41A5
0x<DATA(15:0)>50A5
0x<OFFSET>F8A4
 	
0x0C000C00
	lui a1, <DATA(31:16)>;
	ori a1, <DATA(15:0)>;
	sw a1, <OFFSET>(a0);
	//OFFSET increments by 4
	Two NOPs

步骤 3： 重复步骤 2，直到装入一行数据为止。	

12.2 使用 PE

当使用 PE 时，**第 12.0 节 “下载数据块”**和**第 13.0 节 “初始化闪存行写操作”**中的步骤由两条单命令处理：ROW_PROGRAM 和 PROGRAM。

ROW_PROGRAM 命令对一行闪存数据进行编程，而 PROGRAM命令可对多行闪存数据进行编程。这两条命令都位于**第 16.0 节 “编程执行程序”**中。

13.0 初始化闪存行写操作

注： 一些 PIC32 器件提供 ECC 存储器。当使用 ECC 特性时，闪存必须以 4 个 32 位字为一组进行编程，且使用 4 个 32 位字对齐。如果 ECC 为动态使用，则编程方法决定了何时使用该特性。在使用单字编程命令对字进行编程时，不使能 ECC。ECC 在以 4 字为一组进行编程或使用四字或行编程命令时使能。未遵循这些方法将在运行时导致 ECC DED 错误。关于使用和配置 ECC 的详细信息，请参见具体器件的数据手册。

将一行数据下载到器件 SRAM 中之后，必须启动编程序列来将数据块写入闪存中。

关于启动闪存行写操作的操作码和指令，请参见表 13-1。

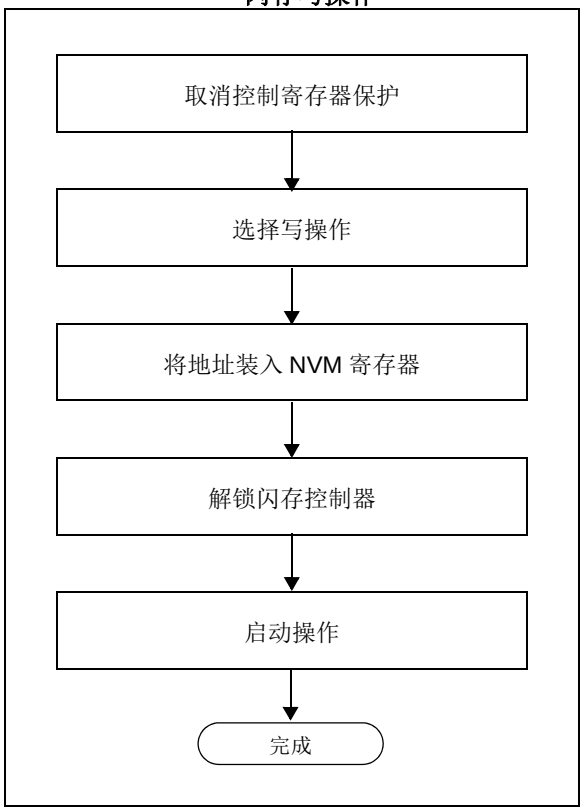
13.1 使用 PE

当使用 PE 时，数据会立即从 SRAM 写入闪存中。无需任何其他操作。

13.2 不使用 PE

闪存写操作通过 NVMCON 寄存器进行控制。编程的执行过程如下：设置 NVMCON 选择写操作的类型，并通过将 WR 控制位（NVMCON<15>）置 1 启动编程序列。

图 13-1： 不使用 PE 的情况下启动闪存写操作



在闪存写序列中（见表 13-1 和表 13-2），行编程作为最便利的方法用于编程闪存。根据所用器件，也可使用字和四字编程方法，并且根据您的应用程序可能会使用或需要这两种方法。更多信息，请参见具体器件数据手册中的 **“闪存程序存储器”** 章节和《PIC32 系列参考手册》中的相关章节。

启动闪存写操作需要执行以下步骤：

1. `XferInstruction` (op code)。
2. 重复步骤1，直到最后一条指令传输到CPU为止。

表 13-1： 启动闪存行写操作码（仅针对 PIC32MX、 PIC32MZ 和 PIC32MK 器件）

操作码	指令
步骤 1： **所有 PIC32 器件：**初始化常量。寄存器 a1、 a2 和 a3 分别设置为 WREN = 1 或 NVMOP<3:0> = 0011、WR = 1 和 WREN = 1。寄存器 s1 和 s2 设置为解锁数据值， s0 初始化为 0。	
34054003	ori a1,$0,0x4003
34068000	ori a2,$0,0x8000
34074000	ori a3,$0,0x4000
3c11aa99	lui s1,0xaa99
36316655	ori s1,s1,0x6655
3c125566	lui s2,0x5566
365299aa	ori s2,s2,0x99aa
3c100000	lui s0,0x0000
步骤 2： **仅 PIC32MX 器件：**设置寄存器 a0 为 NVM 寄存器的基地址（0xBF80_F400）。	
3c04bf80	lui a0,0xbf80
3484f400	ori a0,a0,0xf400
步骤 2： **仅 PIC32MK 和 PIC32MZ 系列器件：**设置寄存器 a0 为 NVM 寄存器的基地址（0xBF80_0600）。寄存器 s3 设置为用于禁止 NVMBPB 写保护的值。	
3c04bf80	lui a0,0xbf80
34840600	ori a0,a0,0x0600
34138080	ori s3,$0,0x8080
步骤 3： **仅 PIC32MK 和 PIC32MZ 系列器件：**解锁并禁止引导闪存写保护。	
ac910010	sw s1,16(a0)
ac920010	sw s2,16(a0)
ac930090	sw s3,144(a0)
00000000	nop
步骤 4： **所有 PIC32 器件：**使用要编程的闪存行地址设置 NVMADDR 寄存器。	
3c08<ADDR>	lui t0,<FLASH_ROW_ADDR(31:16)>
3508<ADDR>	ori t0,t0,
ac880020	<FLASH_ROW_ADDR(15:0)>
	sw t0,32(a0)

表 13-1： 启动闪存行写操作码（仅针对 PIC32MX、 PIC32MZ 和 PIC32MK 器件）（续）

操作码	指令
步骤 5： 仅 PIC32MX 器件：使用物理源 SRAM 地址设置 NVMSRCADDR 寄存器（偏移量为 64）。	
3610<ADDR> ac900040	ori s0,s0,<RAM_ADDR(15:0)> sw s0,64(a0)
步骤 5： 仅 PIC32MK 和 PIC32MZ 系列器件：使用物理源 SRAM 地址设置 NVMSRCADDR 寄存器（偏移量为 112）。	
3610<ADDR> ac900070	ori s0,s0,<RAM_ADDR(15:0)> sw s0,112(a0)
步骤 6： 所有 PIC32 器件：为进行写操作设置 NVMCON 寄存器。	
ac850000	sw a1,0(a0) delay (6 µs)
步骤 7： 仅 PIC32MX 器件：查询 LVDSTAT 寄存器。	
 8C880000 31080800 1500fffd 00000000	here1: lw t0,0(a0) andi t0,t0,0x0800 bne t0,$0,here1 nop
步骤 8： 所有 PIC32 器件：解锁 NVMCON 寄存器并启动写操作。	

ac910010
ac920010
ac860008
	sw s1,16(a0)
	sw s2,16(a0)
	sw a2,8(a0)

步骤 9： 所有 PIC32 器件：循环直到 WR 位（NVMCON<15>）清零。	
 8c880000 01064024 1500fffd 00000000	here2: lw t0,0(a0) and t0,t0,a2 bne t0,$0,here2 nop
步骤 10： 所有 PIC32 器件：在看到 WR 位（NVMCON<15>）为 0 后，在写入 NVM 寄存器之前，等待至少 500 ns。这需要在执行中插入 NOP。下列示例假设内核执行速度为 8 MHz，因此 4 条 NOP 指令等于 500 ns。	
00000000 00000000 00000000 00000000	nop nop nop nop
步骤 11： 所有 PIC32 器件：清零 WREN 位（NVMCONM<14>）。	
ac870004	sw a3,4(a0)
步骤 12： 所有 PIC32 器件：检查 WRERR 位（NVMCON<13>）以确定编程序列成功完成。如果发生任何错误，将跳转到错误处理程序。	
8c880000 30082000 1500<ERR_PROC> 00000000	lw t0,0(a0) andi t0,zero,0x2000 bne t0, $0, <err_proc_offset> nop

表 13-2： 启动闪存行写操作码（仅针对 PIC32MM 器件）

操作	操作数
步骤 1： 初始化常量。寄存器 a1 和 a2 分别设置为 WREN = 1 或 NVMOP<3:0> = 0011 和 WR = 1。寄存器 s1 和 s2 设置为解锁数据值， s0 初始化为 0。 ori a1,$0,0x4003 ori a2,$0,0x8000 lui s1,0xaa99 ori s1,s1,0x6655 lui s2,0x5566 ori s2,s2,0x99aa lui s0,0x0000	
XferInstruction	0x400350A0
XferInstruction	0x800050C0
XferInstruction	0xAA9941B1
XferInstruction	0x66555231
XferInstruction	0x556641B2
XferInstruction	0x99AA5252
XferInstruction	0x000041B0
步骤 2： 设置寄存器 a0 为 NVM 寄存器的基地址（0xBF80_2380）。寄存器 s3 设置为用于禁止 NVMBPB 写保护的值。 lui a0,0xbf80 ori a0,a0,0x2380 ori s3,$0,0x8000 /* BWPAUNLK bit mask */	
XferInstruction	0xBF8041A4
XferInstruction	0x23805084
XferInstruction	0x80005260

步骤 3： 解锁并禁止引导闪存写保护。
sw s1,16(a0)
/* NVMKEY = 0xaa996655 */
sw s2,16(a0)
/* NVMKEY = 0x556699aa */
sw s3,112(a0)
/*BWPAUNLK bit (NVMBPB register) = 1*/
nop	

XferInstruction	0xBF8041A4
XferInstruction	0x23805084
XferInstruction	0x80005260
步骤 4： 使用要编程的闪存行地址设置 NVMADDR 寄存器。 lui t0,<FLASH_ROW_ADDR(31:16)> ori t0,t0,<FLASH_ROW_ADDR(15:0)> sw t0,32(a0)	
XferInstruction	0x<FLASH_ROW_ADDR(31:16)>41A8
XferInstruction	0x<FLASH_ROW_ADDR(15:0)>5108
XferInstruction	0x0020F904
步骤 5： 使用物理源 SRAM 地址设置 NVMSRCADDR 寄存器。 ori s0,$0,<RAM_ADDR(15:0)> sw s0,80(a0)	
XferInstruction	0x<RAM_ADDR(15:0)>5200
XferInstruction	0x0050FA04

表 13-2： 启动闪存行写操作码 （仅针对 PIC32MM 器件）（续）

操作	操作数
步骤 6： 为进行写操作设置 NVMCON 寄存器。 sw a1,0(a0) /* NVMCON = 0x4003 */ delay (6 µs)	
XferInstruction	0x0C00EAC0

步骤 7： 解锁 NVMCON 寄存器并启动写操作 （WR 位 = 1）。
sw s1,16(a0)
/* NVMKEY = 0xaa996655 */
sw s2,16(a0)
/* NVMKEY = 0x556699aa */
sw a2,8(a0)
/* NVMCONSET = 0x8000 */	

XferInstruction	0xFA44E8C4
XferInstruction	0xEB420010
步骤 8： 读取 NVMCON 寄存器直到 WR 位 （NVMCON<15>）清零。	
ReadFromAddress	0xBF802380
步骤 9： 在看到 WR 位 （NVMCON<15>）为 0 后，在写入 NVM 寄存器之前，等待至少 500 ns。这需要在执行中插入 NOP。 nop nop nop nop	
XferInstruction	0x0C000C00
XferInstruction	0x0C000C00
步骤 10： 清零 WREN 位 （NVMCONM<14>）。 ori a3,$0,0x4000 sw a3,4(a0) nop	
XferInstruction	0x400050E0
XferInstruction	0x0C00EBC1

14.0 校验器件存储器

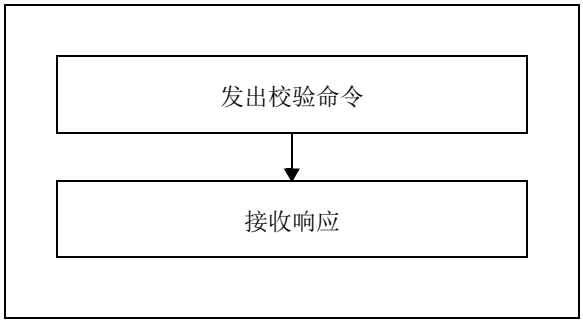
校验的步骤涉及回读代码存储空间并将读到的值与存储在编程器缓冲区中的副本作比较。使用其余代码校验配置寄存器。

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，如果在代码保护位清零之后发生器件复位，器件将不可读取和不可校验。

14.1 使用 PE 的情况下校验存储器

存储器校验使用 GET_CRC 命令来执行。表 16-2 列出了操作码和指令。

图 14-1： 使用 PE 的情况下校验存储器



使用 PE 校验存储器需要执行以下步骤：

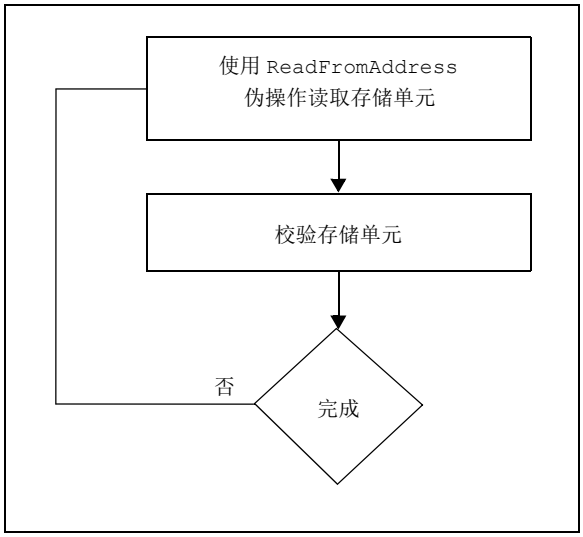
1. XferFastData (GET_CRC)。
2. XferFastData (start_Address)。
3. XferFastData (length)。
4. valCkSum = XferFastData (32'h0x00)。

验证 valCkSum 是否与编程器缓冲区中所保存副本的校验和匹配。

14.2 不使用 PE 的情况下校验存储器

读取闪存的方法是通过Fastdata寄存器执行一系列读访问。表 19-4 列出了 EJTAG 编程细节，包括用于执行处理器访问操作的地址和操作码数据。

图 14-2： 不使用PE的情况下校验存储器



校验存储器需要执行以下步骤：

1. XferInstruction (op code)。
2. 重复步骤1，直到最后一条指令传输到CPU为止。
3. 验证valRead是否与编程器缓冲区中保存的副本匹配。
4. 对于每个存储单元重复步骤 1-3。

表 14-1： 校验器件操作码

操作码	指令
步骤 1： 初始化一些常量。	
3c13ff20	lui $s3, 0xFF20
步骤 2： 读取存储单元。	
3c08<ADDR> 3508<ADDR>	lui $t0,<FLASH_WORD_ADDR(31:16)> ori $t0,<FLASH_WORD_ADDR(15:0)>
步骤 3： 写入 Fastdata 单元。	
8d090000 ae690000	lw $t1, 0($t0) sw $t1, 0($s3)
步骤 4： 从 Fastdata 寄存器 0xFF200000 读取数据。	
步骤 5： 重复步骤 2-4，直到读取所有配置单元为止。	

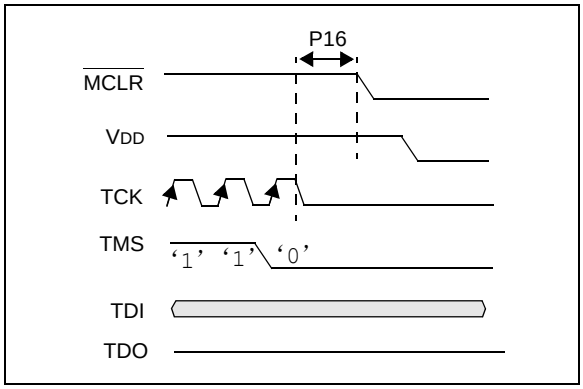
15.0 退出编程模式

对器件进行编程之后，必须使器件退出编程模式，以便开始正常执行新的程序存储器内容。

15.1 4 线接口

通过将 VIH 从 MCLR 引脚移除可退出编程模式，如图 15-1 所示。退出操作的惟一要求是移除 VIH 要在最后一个时钟和编程信号后的 P16 个时间间隔后进行。

图 15-1： 4 线退出编程模式



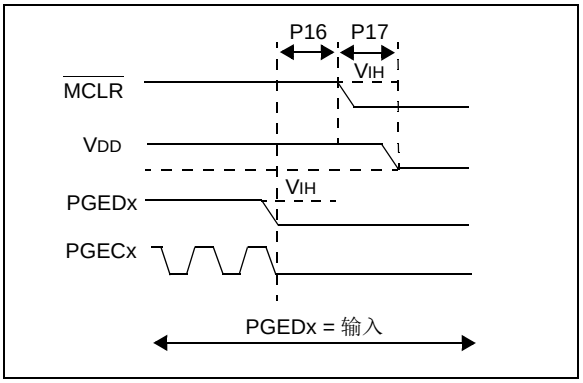
退出编程模式需要执行以下步骤：

1. SetMode (5’b11111)。
2. 将 MCLR 置为有效。
3. 断电 （如果器件已供电）。

15.2 2 线接口

通过将 VIH 从 MCLR 引脚移除可退出编程模式，如图 15-2 所示。退出操作的惟一要求是移除 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 15-2： 2 线退出编程模式



使用以下步骤来退出编程模式：

1. SetMode (5’b11111)。
2. 将 MCLR 置为有效。
3. 在 PGECx 上发出时钟脉冲。
4. 断电 （如果器件已供电）。

16.0 编程执行程序

注： 编程执行（PE）程序包含在 MPLAB® X IDE 的安装中。欲下载适合您器件的 PE 文件，请访问 Microchip 网站（www.microchip.com）的相关产品页面。

16.1 PE 通信

编程器和PE存在主从关系，其中编程器是主编程器件，而 PE 处于从动地位。

所有通信都是由编程器以命令形式发起的。 PE 每次只能接收一条命令。相应地，在接收并处理一条命令之后， PE 会向编程器发送单个响应。

16.1.1 2 线 ICSP EJTAG 速率

在增强型 ICSP 模式下， PIC32 系列器件使用内部快速 RC 振荡器作为时钟源工作，其标称频率为 8 MHz。

16.1.2 通信概述

编程器和 PE 使用 EJTAG 地址、数据和 Fastdata 寄存器进行通信。特别是，编程器通过使用 Fastdata 寄存器来向 PE 传输命令和数据。编程器使用地址和数据寄存器来接收来自 PE 的响应。下面的 `GetPEResponse` 伪操作中显示了接收一个响应的伪操作：

格式：

```
response = GetPEResponse()
```

目的：

使编程器从 PE 接收 32 位的响应值。

例 16-1： GetPEResponse 示例

```
WORD GetPEResponse()
{
        WORD response;

// Wait until CPU is ready
SendCommand(ETAP_CONTROL);
// Check if Processor Access bit (bit 18) is set
do {
      controlVal=XferData(32'h0x0004C000 );
} while( PrAcc(contorlVal<18>) is not  '1'
);

// Select Data Register
SendCommand(ETAP_DATA);
// Receive Response
response = XferData(0);
// Tell CPU to execute instruction
SendCommand(ETAP_CONTROL);
XferData(32'h0x0000C000);
// return 32-bit response
return response;
```

表 16-1 中列出了编程器和 PE 之间的典型通信序列。

序列在编程器向 PE 发送命令和可选的附加数据时开始，而 PE 会执行命令。

当 PE 完成命令执行时，它会向编程器回送响应。

响应中可能包含更多内容。例如，如果编程器发送了一条 `READ` 命令，则响应中将包含所读取的数据。

表 16-1： PE 的通信序列

操作	操作数
步骤 1： 从编程器向 PE 发送命令和可选数据。	
`XferFastData`	`(Command | data len)`
`XferFastData..`	`optional data..`
步骤 2： 编程器读取来自 PE 的响应。	
`GetPEResponse`	`response`
`GetPEResponse...`	`response...`

16.2 PE 命令集

表 16-2 给出了 PE 命令集的详细信息，如每条命令的操作码、助记符和简短说明。在**第 16.2.3 节“ROW_PROGRAM命令”**至**第16.2.14节“CHANGE_CFG命令”**中详细说明了每条命令的功能。

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，其中包含任何必需的响应数据或错误数据。

16.2.1 命令格式

所有 PE 命令都具有由 32 位头和该命令所需的所有数据组成的通用格式，请参见图 16-1。 32 位头由用于标识命令的 16 位操作码字段和 16 位命令操作数字段组成。所使用的操作数字段因命令而异。

注： 但是，一些命令没有操作数信息，而操作数字段是必须发送的，此时编程执行程序会忽略该数据。

图 16-1： 命令格式

操作码字段中的命令必须与表16-2中所列命令集中的命令匹配。如果接收到的命令与列表中的任何命令都不匹配，则会返回 NACK 响应，如表 16-3 中所示。

PE 使用命令操作数字段来确定要读取或写入的字节数。如果该字段的值不正确， PE 将无法正确地接收命令。

表 16-2： PE 命令集

操作码	助记符	说明
0x0	ROW_PROGRAM(1)	对闪存指定地址处的一行进行编程。
0x1	READ	从指定地址开始读取存储器的 N 个 32 位字。（N < 65,536）。
0x2	PROGRAM	从指定地址开始对闪存进行编程。
0x3	WORD_PROGRAM(3,5)	对闪存指定地址处的一个字进行编程。
0x4	CHIP_ERASE	对整个芯片进行芯片擦除。
0x5	PAGE_ERASE	从指定地址开始擦除代码存储区中的页。
0x6	BLANK_CHECK	空白检查代码。
0x7	EXEC_VERSION	读取 PE 软件版本。
0x8	GET_CRC	获取闪存的 CRC。
0x9	PROGRAM_CLUSTER	对指定地址编程指定的字节数。
0xA	GET_DEVICEID	返回器件的硬件 ID。
0xB	CHANGE_CFG(2)	由探针用来为 PE 设置不同的配置设定。
0xC	GET_CHECKSUM	获取闪存的校验和。
0xD	QUAD_WORD_PGRM(4)	在指定地址对闪存进行 4 字编程。
0xE	DOUBLE_WORD_PGRM(6)	在指定地址对闪存进行 2 字编程。

注 1： 关于每个器件的行大小，请参见表 5-1。

2： PIC32MX1XX/2XX 系列器件上不具备该命令。

3： PIC32MZ 系列器件上集成了 ECC，WORD_PROGRAM 命令将不会生成 ECC 奇偶校验位。在 ECC 使能时，读取由 WORD_PROGRAM 命令编程的存储单元将导致 DED 故障。

4: 该命令仅在 PIC32MK 和 PIC32MZ 系列器件上可用。

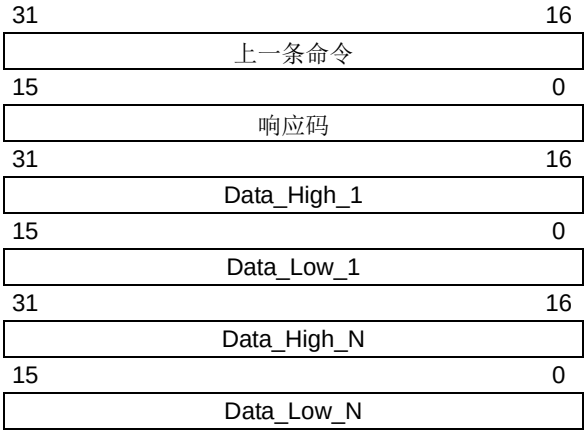
5: 该命令在 PIC32MM 系列器件上不可用。

6: 该命令仅在 PIC32MM 系列器件上可用。

16.2.2 响应格式

表 16-3 给出了 PE 响应集。所有 PE 响应都具有由 32 位头和该响应所需的所有数据组成的通用格式（见图16-2）。

图 16-2： 响应格式



16.2.2.1 Last_Cmd 字段

Last_Cmd 是响应的第一个字中的 16 位字段，该字段指示 PE 处理的命令。它可用于验证 PE 是否正确接收到编程器发送的命令。

16.2.2.2 响应码

响应码指示上一条命令是成功还是失败，还是命令为无法识别的值。表 16-3 中列出了响应码的值。

表 16-3： 响应值

操作码	助记符	说明
0x0	PASS	命令已成功处理
0x2	FAIL	命令未成功处理
0x3	NACK	命令未知

16.2.2.3 可选数据

对于某些命令（例如读操作），响应头后面可能会跟随可选数据。可选数据的 32 位字的数量会因上一个命令操作及其参数而不同。

16.2.3 ROW_PROGRAM 命令

ROW_PROGRAM 命令指示 PE 对位于指定地址的一行数据进行编程。

要编程到存储区中的数据位于命令字 Data_1 至 Data_N 中，且必须使用表 16-4 给出的指令字打包格式对它们进行编排 （该命令预期一整行数据）。

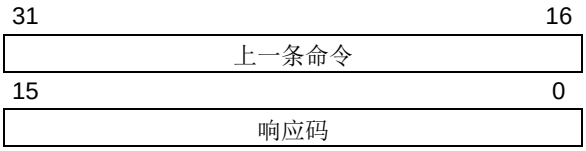
图 16-3： ROW_PROGRAM 命令

表 16-4： ROW_PROGRAM 格式

字段	说明
操作码	0x0
操作数	未使用
Addr_High	32 位目标地址的高 16 位
Addr_Low	32 位目标地址的低 16 位
Data_High_1	高 16 位数据字 1
Data_Low_1	低 16 位数据字 1
Data_High_N	高 16 位数据字 2 至 N
Data_Low_N	低 16 位数据字 2 至 N

预期的响应 （单字）：

图 16-4： ROW_PROGRAM 响应



16.2.4 READ 命令

READ 命令指示 PE 读取存储器中从 Addr_Low 和 Addr_High 字段指定的 32 位地址开始的多个 32 位字（字数由操作数字段指定）。该命令可用于读取闪存和配置字。作为对该命令的响应，返回的所有数据使用表 16-5 中所示的数据打包格式。

图 16-5： READ 命令

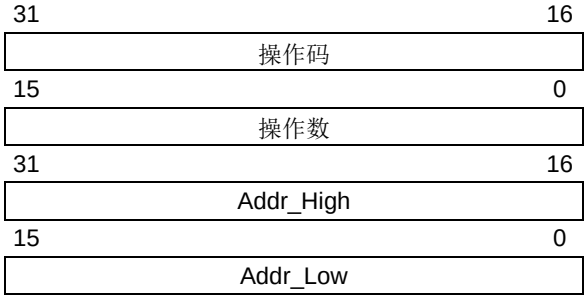


表 16-5： READ 格式

字段	说明
操作码	0x1
操作数	要读取的 32 位字的数量 N（最大为 65,535）
Addr_Low	32 位源地址的低 16 位
Addr_High	32 位源地址的高 16 位

预期的响应：

图 16-6： READ 响应

注： 读取未实现存储区将导致 PE 复位。请确保只访问在特定器件中存在的存储单元。

16.2.5 PROGRAM 命令

PROGRAM 命令指示 PE 对由 Addr_Low 和 Addr_High 字段指定的 32 位地址开始的程序闪存（包括配置字）进行编程。32 位的长度字段指定要编程的字节数。

地址必须与闪存行大小边界对齐，且长度必须为闪存行大小的倍数。闪存行大小为 32 字（128 字节）或 128 字（512 字节），请参见表 5-1。

图 16-7： PROGRAM 命令

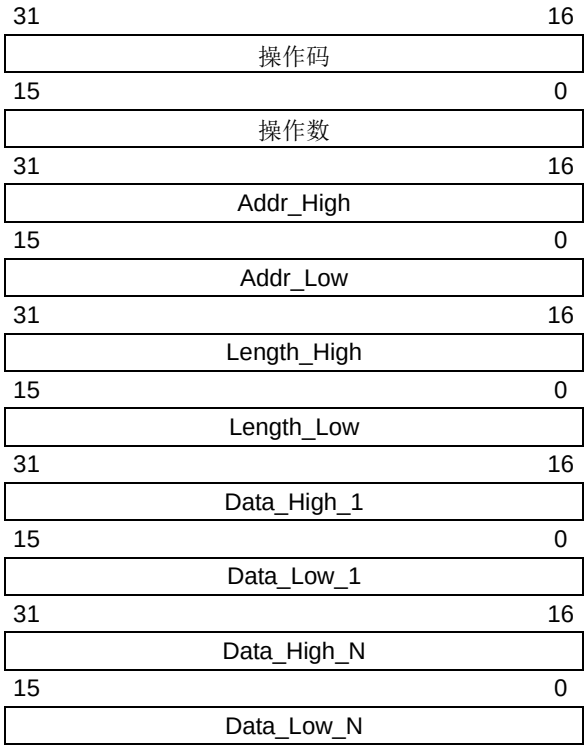


表 16-6： PROGRAM 格式

字段	说明
操作码	0x2
操作数	未使用
Addr_Low	32 位目标地址的低 16 位
Addr_High	32 位目标地址的高 16 位
Length_Low	长度的低 16 位
Length_High	长度的高 16 位
Data_Low_N	低 16 位数据字 2 至 N
Data_High_N	高 16 位数据字 2 至 N

存在以下三种编程情形：

- 编程数据的长度为闪存单行大小
- 编程数据的长度为闪存两行大小
- 编程数据的长度大于闪存两行大小

当数据长度等于512字节时，PE会接收来自探针的512字节数据块，并立即将该命令的响应回送给探针。

PE 将响应其接收到的每行数据。如果命令的数据长度等于单行，会产生单个 PE 响应。如果数据长度等于两行，则 PE 会等待两行数据都接收到后，为每个数据行发送背对背响应。如果数据长度大于两行， PE 将在接收到前两行数据后为第一行发送响应。后续响应将在接收到后续数据行数据包后发出。响应将使数据滞后一行。当接收到最后一行数据后， PE 将依次对倒数第二行数据和最后一行数据发出背对背响应。

如果 PE 在烧写任意数据块时遇到错误，它会向探针发送失败状态并中止 PROGRAM 命令。在接收到失败状态时，探针必须停止发送数据。 PE 将不会处理来自探针的、对应于该命令的任何其他数据。图 16-9 给出了该过程的图示。

注：	如果 PROGRAM 命令失败，编程器应使用 READ 命令从闪存中读取发生失败的行。接着，编程器应将从闪存中接收到的行与其本地副本逐字进行比较，以确定闪存编程发生失败的地址。

该命令的响应与其他命令的响应稍有不同。响应的16个MSb中将包含目标地址（最后一个数据块的编程位置）的16个LSb。这可以帮助探针和PE维持发送和接收数据及响应的正确同步。

预期的响应 （单字）：

图 16-8： PROGRAM 响应

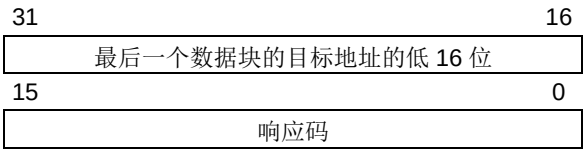
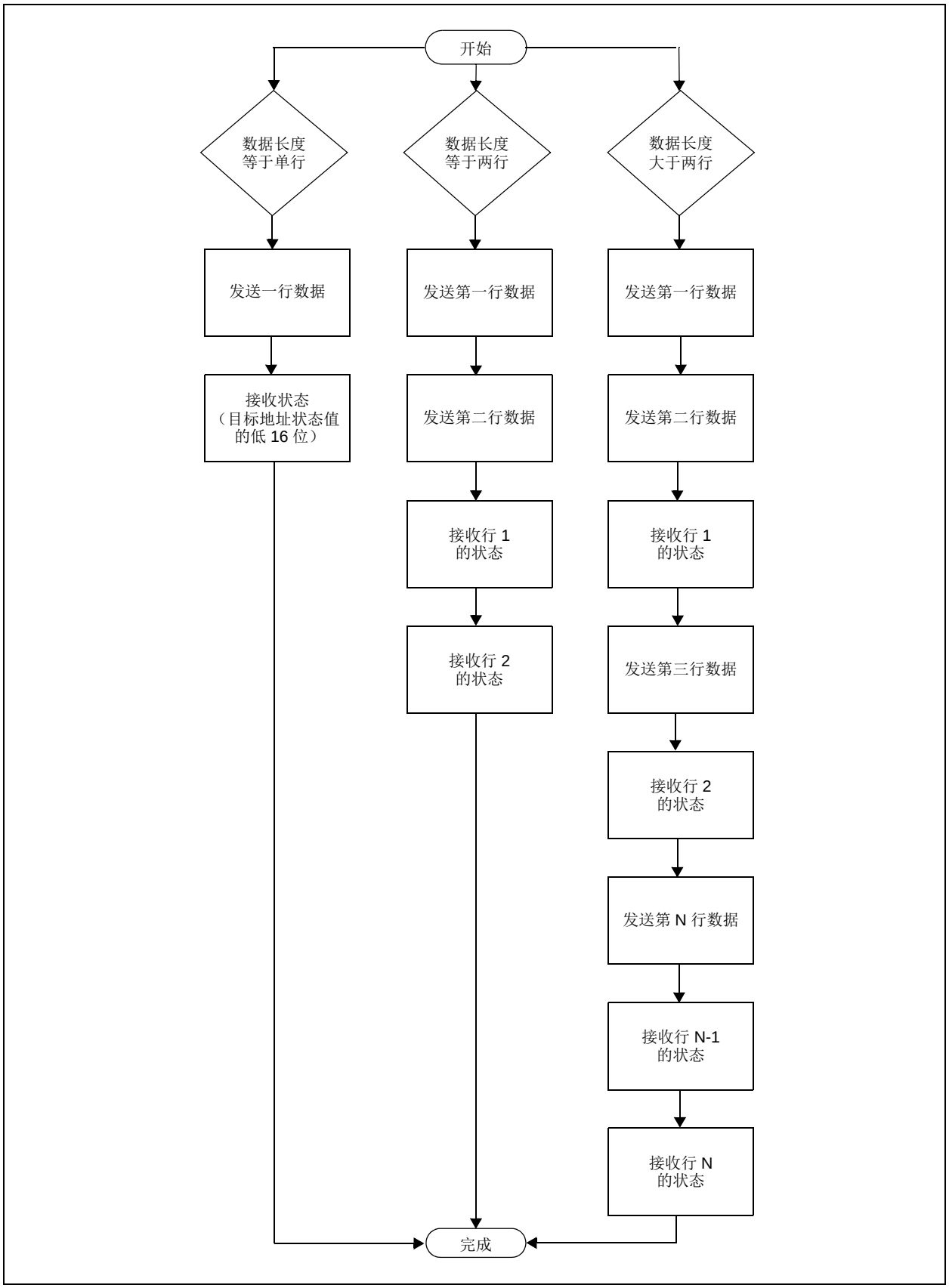


图 16-9：　　PROGRAM 命令算法



16.2.6 WORD_PROGRAM 命令

WORD_PROGRAM命令指示PE对位于指定地址的一个32位数据字进行编程。

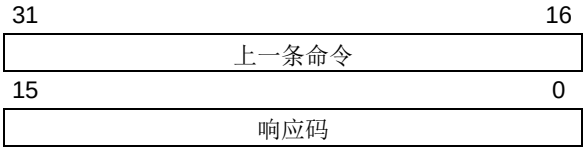
图 16-10： WORD_PROGRAM 命令

表 16-7： WORD_PROGRAM 格式

字段	说明
操作码	0x3
操作数	未使用
Addr_High	32 位目标地址的高 16 位
Addr_Low	32 位目标地址的低 16 位
Data_High	高 16 位数据字
Data_Low	低 16 位数据字

预期的响应 （单字）：

图 16-11： WORD_PROGRAM 响应



16.2.7 CHIP_ERASE 命令

CHIP_ERASE 命令用于擦除整个芯片，包括配置块。执行擦除操作之后，整个闪存将包含 0xFFFFFFFF。

图 16-12： CHIP_ERASE 命令

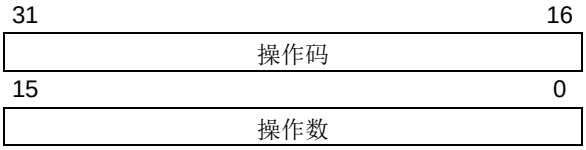
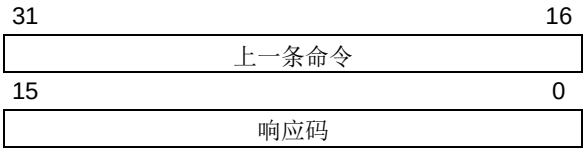


表 16-8： CHIP_ERASE 格式

字段	说明
操作码	0x4
操作数	未使用
Addr_Low	32 位目标地址的低 16 位
Addr_High	32 位目标地址的高 16 位

预期的响应 （单字）：

图 16-13： CHIP_ERASE 响应



16.2.8 PAGE_ERASE 命令

PAGE_ERASE 命令用于从指定基址开始，擦除代码存储区的指定页数。根据不同的器件，所指定的基址必须为 0x400 或 0x100 的倍数。

执行擦除操作之后，代码存储区的所有目标字将包含 0xFFFFFFFF。

图 16-14： PAGE_ERASE 命令

31	16
操作码	
15	0
操作数	
31	16
Addr_High	
15	0
Addr_Low	

表 16-9： PAGE_ERASE 格式

字段	说明
操作码	0x5
操作数	要擦除的页数
Addr_Low	32 位目标地址的低 16 位
Addr_High	32 位目标地址的高 16 位

预期的响应 （单字）：

图 16-15： PAGE_ERASE 响应

31	16
上一条命令	
15	0
响应码	

16.2.9 BLANK_CHECK 命令

BLANK_CHECK 命令查询 PE 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）。

图 16-16： BLANK_CHECK 命令

31	16
操作码	
15	0
操作数	
31	16
Addr_High	
15	0
Addr_Low	
31	16
Length_High	
15	0
Length_Low	

表 16-10： BLANK_CHECK 格式

字段	说明
操作码	0x6
操作数	未使用
地址	空白检查的起始地址
长度	要检查的程序存储单元数量（以字节为单位）

预期的响应 （空白器件的单字）：

图 16-17： BLANK_CHECK 响应

31	16
上一条命令	
15	0
响应码	

16.2.10 EXEC_VERSION 命令

EXEC_VERSION查询存储在RAM中的PE软件的版本。

图 16-18： EXEC_VERSION 命令

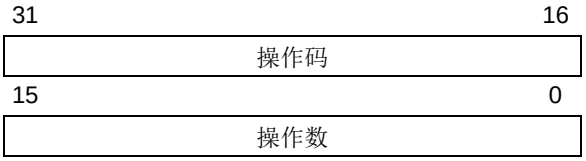
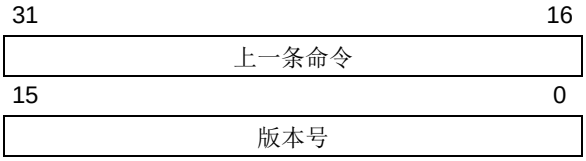


表 16-11： EXEC_VERSION 格式

字段	说明
操作码	0x7
操作数	未使用

预期的响应 （单字）：

图 16-19： EXEC_VERSION 响应



16.2.11 GET_CRC 命令

GET_CRC 使用表查询方法，计算从指定地址开始、大小等于指定长度的缓冲区的 CRC。

CRC 细节如下：

- CRC-CCITT， 16 位
- 多项式：X^16+X^12+X^5+1，十六进制 0x00011021
- 种子：0xFFFF
- 最高有效字节（Most Significant Byte，MSB）先移入

注 1： 在响应中，只有CRC的低16位是有效的。

2： PE 将自动确定硬件 CRC 是否可用并默认使用它。 PIC32MX1XX/2XX 系列器件不使用硬件 CRC。

图 16-20： GET_CRC 命令

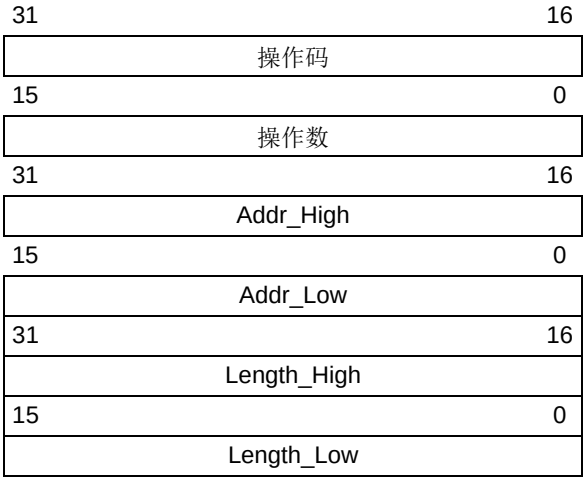
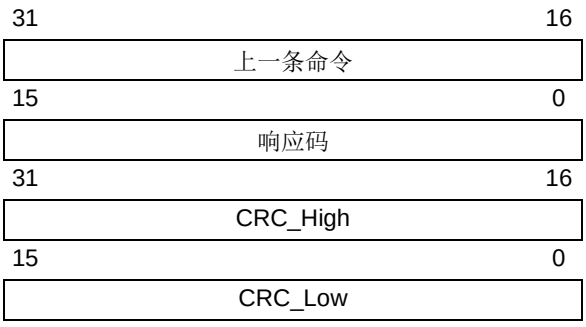


表 16-12： GET_CRC 格式

字段	说明
操作码	0x8
操作数	未使用
地址	计算 CRC 的起始地址
长度	计算 CRC 的缓冲区的长度 （以字节为单位）

预期的响应 （双字）：

图 16-21： GET_CRC 响应



16.2.12 PROGRAM_CLUSTER 命令

PROGRAM_CLUSTER 用于向指定地址烧写指定的字节数。地址必须 32 位对齐，并且字节数必须为 32 位字的倍数。

图 16-22： PROGRAM_CLUSTER 命令

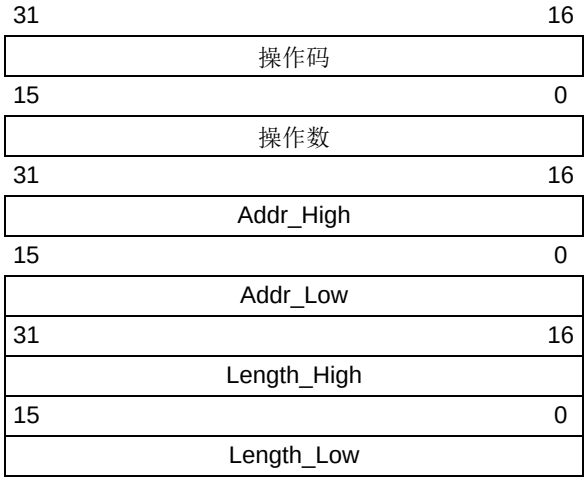


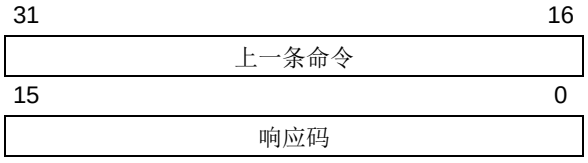
表 16-13： PROGRAM_CLUSTER 格式

字段	说明
操作码	0x9
操作数	未使用
地址	编程的起始地址
长度	编程的区域长度 （以字节为单位）

注： 如果 PROGRAM_CLUSTER 命令失败，编程器应使用 READ 命令从闪存中读取发生失败的行。接着，编程器应将从闪存中接收到的行与其本地副本逐字进行比较，以确定闪存编程发生失败的地址。

预期的响应 （单字）：

图 16-23： PROGRAM_CLUSTER 响应



16.2.13 GET_DEVICEID 命令

GET_DEVICEID 命令返回器件的硬件 ID。

图 16-24： GET_DEVICEID 命令

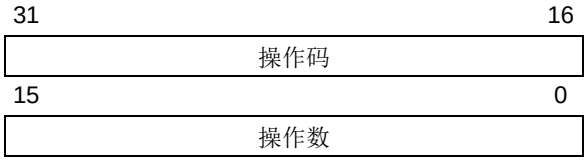


表 16-14： GET_DEVICEID 格式

字段	说明
操作码	0xA
操作数	未使用

预期的响应 （单字）：

图 16-25： GET_DEVICEID 响应

16.2.14 CHANGE_CFG 命令

CHANGE_CFG 供探针用于设置 PE 的各种配置设置。当前，单个配置设置决定 PE 应使用以下何种计算方法：

- 软件 CRC 计算方法
- 硬件计算方法

图 16-26： CHANGE_CFG 命令

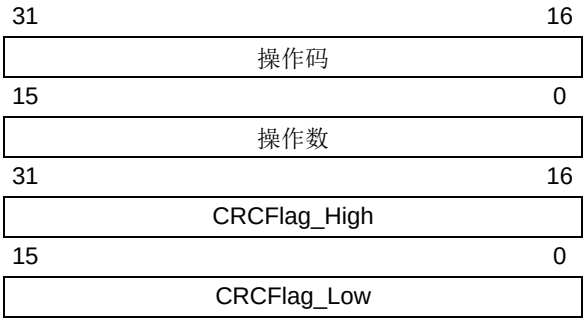
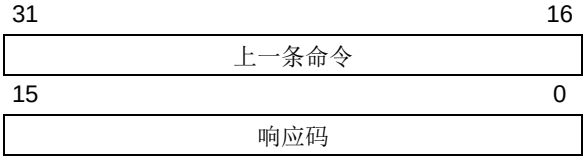


表 16-15： CHANGE_CFG 格式

字段	说明
操作码	0xB
操作数	未使用
CRCFlag	如果值为 0，则 PE 使用软件 CRC 计算方法。 如果值为 1，则 PE 使用硬件 CRC 单元来计算 CRC。

预期的响应 （单字）：

图 16-27： CHANGE_CFG 响应



注：	PIC32MX1XX/2XX 器件不具有 CHANGE_CFG 命令。

16.2.15 GET_CHECKSUM 命令

GET_CHECKSUM 返回所有从地址参数开始到长度参数结束的所有字节之和。结果为一个 32 位字。

图 16-28： CHANGE_CFG 命令

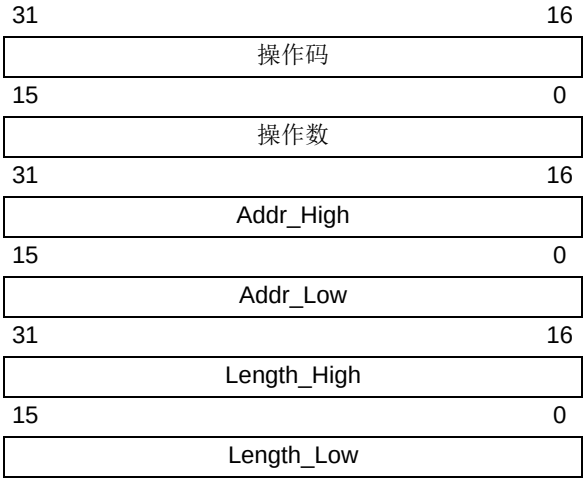


表 16-16： GET_CHECKSUM 格式

字段	说明
操作码	0x0C
操作数	未使用
Addr_High	用于计算校验和的 32 位数据起始地址的高 16 位。
Addr_Low	用于计算校验和的 32 位数据起始地址的低 16 位。
Length_High	用于计算校验和的 32 位数据长度的高 16 位 （以字节为单位）。
Length_Low	用于计算校验和的 32 位数据长度的低 16 位 （以字节为单位）。

预期的响应 （单字）：

图 16-29： GET_CHECKSUM 响应

16.2.16 QUAD_WORD_PROGRAM 命令

QUAD_WORD_PROGRAM指示PE在指定地址编程4个32位字。地址必须在四字边界对齐 （bit 0-1 必须为 0）。否则，命令将返回一个失败（FAIL）响应值且不会编程数据。

图 16-30： QUAD_WORD_PROGRAM 命令

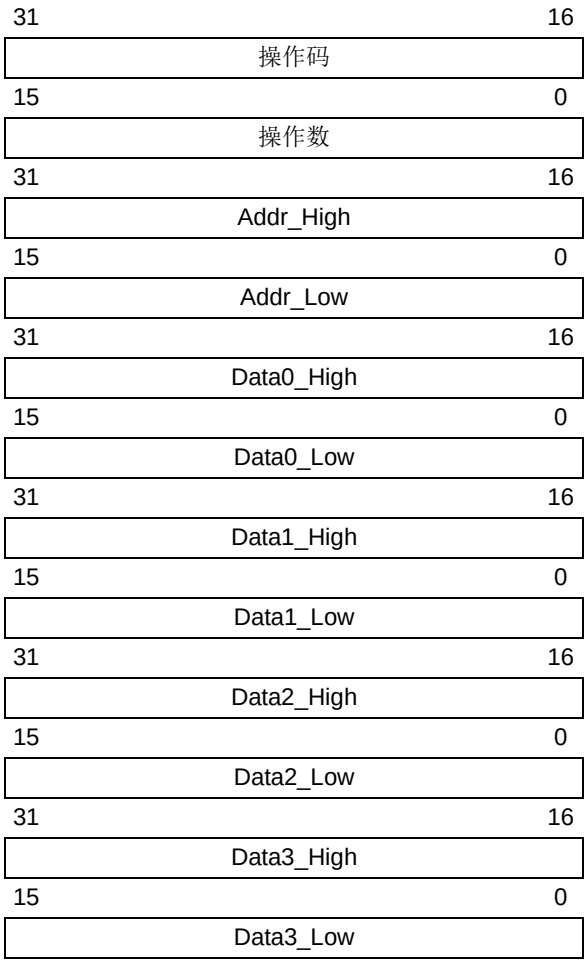
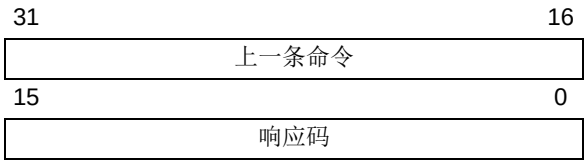


表 16-17： QUAD_WORD_PROGRAM 格式

字段	说明
操作码	0x0D
操作数	未使用
Addr_High	32 位起始地址的高 16 位。
Addr_Low	32 位起始地址的低 16 位。
Data0_High	数据字 0 的高 16 位。
Data0_Low	数据字 0 的低 16 位。
Data1_High	数据字 1 的高 16 位。
Data1_Low	数据字 1 的低 16 位。
Data2_High	数据字 2 的高 16 位。
Data2_Low	数据字 2 的低 16 位。
Data3_High	数据字 3 的高 16 位。
Data3_Low	数据字 3 的低 16 位。

预期的响应 （单字）：

图 16-31： QUAD_WORD_PROGRAM 响应



17.0 校验和

17.1 原理

校验和的计算方式为：将程序闪存、引导闪存（器件配置字除外）、带适用掩码的器件 ID 寄存器，以及带适用掩码的器件配置字中的所有字节（8 位量）相加，得到一个 32 位的相加和。然后，计算该相加和的二进制补码。这个最终的 32 位数字就是校验和。

17.2 掩码值

器件配置的掩码值通过将所有未实现位设置为 0，所有已实现位设置为 1 而计算得到。

例如，寄存器 17-1 显示了 PIC32MX360F512L 器件的 DEVCFG0 寄存器。该寄存器的掩码值为：

mask_value_devcfg0 = 0x110FF00B

寄存器 17-1： PIC32MX360F512L 的 DEVCFG0 寄存器

位范围	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	r-0	r-1	r-1	R/P-1	r-1	r-1	r-1	R/P-1
	—	—	—	CP	—	—	—	BWP
23:16	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1
	—	—	—	—	PWP19	PWP18	PWP17	PWP16
15:8	R/P-1	R/P-1	R/P-1	R/P-1	r-1	r-1	r-1	r-1
	PWP15	PWP14	PWP13	PWP12	—	—	—	—
7:0	r-1	r-1	r-1	r-1	R/P-1	r-1	R/P-1	R/P-1
	—	—	—	—	ICESEL	—	DEBUG<1:0>	

图注：			
R = 可读位	P = 可编程位	r = 保留位	
-n =POR 时的值	W = 可写位	U = 未实现位，读为 0	
	1 = 置 1	0 = 清零	x = 未知

表 17-1 列出了要在校验和计算中使用的 4 个器件配置寄存器和器件 ID 寄存器的掩码值 （针对 PIC32MX、PIC32MZ 和 PIC32MK 器件）。

表 17-1： 器件配置寄存器的掩码值 （当前支持 PIC32MX、 PIC32MZ 和 PIC32MK 器件）

器件系列	闪存大小 （KB）	DEVCFG0	DEVCFG1	DEVCFG2	DEVCFG3	DEVID
PIC32MX 110/120/130/150 （仅 28/36/44 引脚器件）	16, 32, 64, 128	0x1100FC1F	0x03DFF7A7	0x00070077	0xF0000000	0x0FFFFFFF
PIC32MX 210/220/230/250 （仅 28/36/44 引脚器件）	16, 32, 64, 128	0x1100FC1F	0x03DFF7A7	0x00078777	0xF0000000	0x0FFFFFFF
PIC32MX 320/340/360	32, 64, 128, 256, 512	0x110FF00B	0x009FF7A7	0x00070077	0x00000000	0x000FF000
PIC32MX 420/440/460	32, 64, 128, 256, 512	0x110FF00B	0x009FF7A7	0x00078777	0x00000000	0x000FF000
PIC32MX 120/130/150/170/230/250/270/530/550/570 （仅 64/100 引脚器件）	64, 128, 256, 512	0x110FFC1F	0x03DFF7A7	0x00078777	0xF0070000	0x0FFFFFFF
PIC32MX 330/350/370	64, 128, 256, 512	0x110FF01F	0x03DFF7A7	0x00070077	0x30070000	0x0FFFFFFF
PIC32MX 430/450/470	64, 128, 256, 512	0x110FF01F	0x03DFF7A7	0x00078777	0xF0070000	0x0FFFFFFF
PIC32MX 534/564	64, 128	0x110FF00F	0x009FF7A7	0x00078777	0xC4070000	0x0FFFF000
PIC32MX 664	64, 128	0x110FF00F	0x009FF7A7	0x00078777	0xC3070000	0x0FFFF000
PIC32MK 0128/0256/0512/1024	128, 256, 512, 1024	0x7FFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFF0000	0x0FFFFFFF
PIC32MX 764	128	0x110FF00F	0x009FF7A7	0x00078777	0xC7070000	0x0FFFF000
PIC32MX 170 （仅 28/36/44 引脚器件）	256	0x1107FC1F	0x03DFF7A7	0x00070077	0xF0000000	0x0FFFFFFF
PIC32MX 270 （仅 28/36/44 引脚器件）	256	0x1107FC1F	0x03DFF7A7	0x00078777	0xF0000000	0x0FFFFFFF
PIC32MZ 0256/0512/1024/2048	256, 512, 1024, 2048	0x7FFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFF0000	0x0FFFFFFF
PIC32MX 575	256, 512	0x110FF00F	0x009FF7A7	0x00078777	0xC4070000	0x000FF000
PIC32MX 675/695	256, 512	0x110FF00F	0x009FF7A7	0x00078777	0xC3070000	0x000FF000
PIC32MX 775/795	256, 512	0x110FF00F	0x009FF7A7	0x00078777	0xC7070000	0x000FF000

表 17-2 列出了要在校验和计算中使用的器件配置寄存器和器件 ID 寄存器的掩码值（针对 PIC32MM 器件）。

表 17-2： PIC32MM 器件的器件配置寄存器的掩码值

	配置字						DEVID
	FDEVOPT	FICD	FPOR	FWDT	FOSCEL	FSEC	
寄存器掩码	0xFFFFFFFA	0xFFFFFFF4	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0x3FFFFFFF	0xFFFFFFFF

17.3 算法

图 17-1 给出了计算 PIC32 器件校验和的高级算法的示例，以说明一种用于得到校验和的方法。这只是一个说明可以如何实现实际计算的示例，最终使用的方法由软件开发人员自行决定。

如前面所述，PIC32 校验和的计算方式为：将程序闪存、引导闪存（器件配置字除外）、带适用掩码的器件 ID 寄存器，以及带适用掩码的器件配置字中的所有字节 （8 位量）相加，得到一个 32 位的相加和。

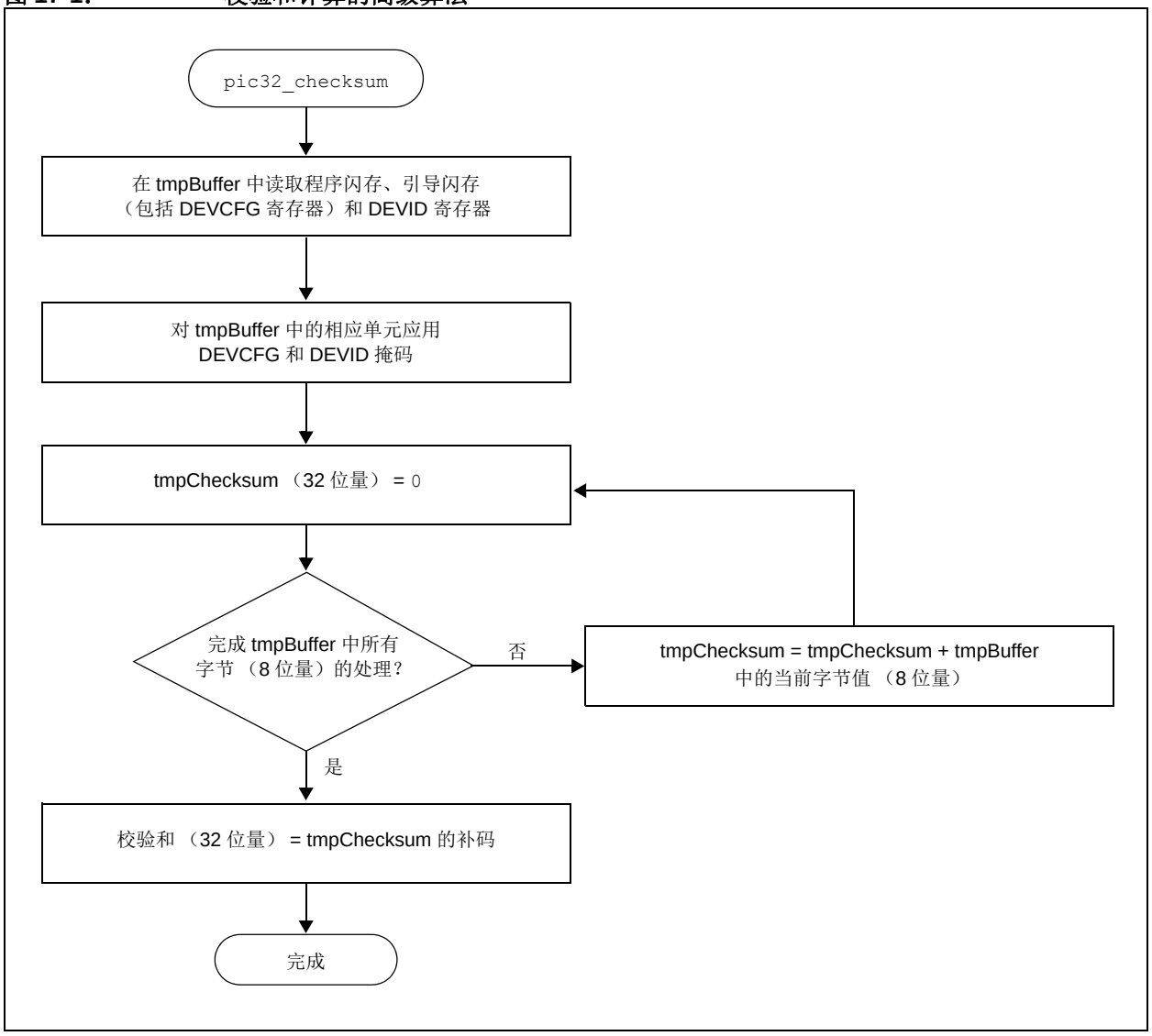
然后，计算该相加和的补码。这个最终的 32 位数字就是校验和。

器件配置和器件ID寄存器的掩码值根据前一节**第 17.2 节 “掩码值”**中介绍的方式得到。

在将这些寄存器的字节加到校验和之前，需要使用相应的掩码值对这些器件配置寄存器值执行算术与运算。

类似地，在将器件 ID 寄存器的字节加到校验和之前，需要使用相应的掩码值对器件ID寄存器值执行算术与运算，更多信息请参见**第 18.0 节 “配置存储器和器件 ID”**。

图 17-1： 校验和计算的高级算法



公式 17-1 给出了用于计算 PIC32 器件校验和的公式。

公式 17-1：　　校验和公式

$$校验和 = (PF + BF + DCR + DIR) \text{ 的二进制补码}$$

其中，

PF = 程序闪存中所有字节的 32 位相加和

BF = 引导闪存 （器件配置寄存器除外）中所有字节的 32 位相加和

$$DCR = \sum_{X=0}^{3} \text{字节的 32 位相加和 } (MASK_{DEVCFGX} \,\&\, DEVCFGx)$$

DIR = 字节的 32 位相加和 $(MASK_{DEVID} \,\&\, DEVID)$

$MASK_{DEVCFGX}$ = 来自表 17-1 的掩码值

$MASK_{DEVID}$ = 来自表 17-1 的掩码值

17.4　校验和计算示例

以下 5 节说明了如何使用公式 17-1 来计算 PIC32MX360F512L 器件的校验和。

对于该校验和计算示例，进行了以下假设：

- 程序闪存和引导闪存处于已擦除状态 （所有字节均为 0xFF）
- 器件配置处于器件的默认状态 （未进行任何配置更改）

需要单独计算公式右侧的每一项 （PF、 BF、 DCR 和 DIR）。得到这些值之后，可以计算校验和的最终值。

17.4.1　计算校验和公式中的 “PF”

程序闪存的大小为 512 KB，等于 524288 字节。由于假设程序闪存处于已擦除状态，所以 “PF” 的值可以通过以下算式求解：

PF = 0xFF + 0xFF + … 524288 次

PF = 0x7F80000 （32 位数字）

17.4.2　计算校验和公式中的 “BF”

引导闪存的大小为 12 KB，等于 12288 字节。但是，最后 16 字节是器件配置寄存器，需要对它们单独进行处理。因此，在此步骤中考虑的引导闪存字节数为 12272。由于假设引导闪存处于已擦除状态，所以 “BF” 的值可以通过以下算式求解：

BF = 0xFF + 0xFF + … 12272 次

BF = 0x002FC010 （32 位数字）

17.4.3　计算校验和公式中的 “DCR”

由于器件配置寄存器处于它们的默认状态，所以基于相应的 DEVCFG 寄存器的值 （由 PIC32 内核读取）、它相应的掩码值、应用掩码之后得到的值，以及字节的 32 位相加和（表 17-3 中列出了全部这些值），可以得到字节的 32 位相加和的总和。

根据表 17-3，“DCR” 的值为：

DCR = 0x000003D6 （32 位数字）

表 17-3：　　DCR 计算示例

寄存器	POR 默认值	掩码	POR 默认值和掩码	字节的 32 位相加和
DEVCFG0	0x7FFFFFFF	0x110FF00B	0x110FF00B	0x0000011B
DEVCFG1	0xFFFFFFFF	0x009FF7A7	0x009FF7A7	0x0000023D
DEVCFG2	0xFFFFFFFF	0x00070077	0x00070077	0x0000007E
DEVCFG3	0xFFFFFFFF	0x00000000	0x00000000	0x00000000
			字节 32 位相加和的总和 =	**0x000003D6**

17.4.4 计算校验和公式中的 “DIR”

表 17-4 中列出了器件 ID 寄存器的值、其掩码值、应用掩码之后得到的值，以及字节的 32 位相加和。

根据表 17-4，“DIR”的值为：

DIR = 0x00000083 （32 位数字）

表 17-4： DIR 计算示例

寄存器	POR 默认值	掩码	POR 默认值和掩码	字节的 32 位相加和
DEVID	0x00938053	0x000FF000	0x00038000	0x00000083

17.4.5 完成 PIC32 校验和计算

在前面几节中得到的值（PF、BF、DCR 和 DIR）用于计算校验和值。对前面几节中得到的 PF、BF、DCR 和 DIR 执行 32 位加法，并将结果存储到名为 *temp* 的变量中，如例 17-1 中所示。

例 17-1： 校验和计算过程

1. 首先， *temp* = PF + BF + DCR + DIR，它转换为：
   temp = 0x7F80000 + 0x002FC010 + 0x000003D6 + 0x00000083
2. 全部 4 个值相加得到 *temp* 等于 0x0827C406
3. 接着，计算 *temp* 的反码，命名为 *temp1*：
   temp1 = （*temp*）的反码，现在它等于 0xF7D83B96
4. 最后， *temp* 的补码为校验和：
   校验和 = （*temp*）的补码；即，校验和 = *temp1* + 1，得到 0xF7D83B97

17.4.6 器件被代码保护时的校验和值

由于在 PIC32 器件处于代码保护状态时，器件配置字不可读，所以校验和对于所有器件均为 0。

18.0 配置存储器和器件 ID

PIC32 器件具有几项特殊的功能旨在最大限度地提高应用的灵活性和可靠性，并通过减少外部元件将成本降至最低。这些功能可通过特定的配置位对每个器件进行配置。

关于功能可用性、配置位和器件 ID 寄存器的完整列表，请参见具体器件数据手册中的 **“特殊功能”**章节。

关于具体器件的当前芯片版本和版本 ID，请参见相关的系列芯片勘误表和数据手册说明。这些文档可通过访问 Microchip 网站：http://www.microchip.com/PIC32 或导航：*文档* > *勘误表*获得。

18.1 器件配置

在 PIC32 器件中，配置字可用于选择各种器件配置，这些配置字在器件复位时在执行任何代码之前设置。这些值位于引导闪存（BFM）的最高单元，由于它们是程序存储器的一部分，因而他们与可执行代码和程序常量一起包含在编程文件中。这些配置字的名称和单元列于表 18-1 至表 18-5 中。

此外，表 18-3 和表 18-4 包含了分别用于具有双引导和双分区闪存的 PIC32MZ 和 PIC32MK 系列器件的配置字。关于双引导区域的详细说明，请参见《PIC32 系列参考手册》中的**第 48 章“存储器构成和权限”**（DS60001214）。

表 18-5 列出了用于 PIC32MM 系列器件的配置字单元。

表 18-1： DEVCFG 单元（仅针对 PIC32MX3XX/4XX/5XX/6XX/7XX和64/100引脚PIC32MX1XX/2XX/5XX 器件）

配置字	物理地址
DEVCFG0	0x1FC02FFC
DEVCFG1	0x1FC02FF8
DEVCFG2	0x1FC02FF4
DEVCFG3	0x1FC02FF0

表 18-2： DEVCFG 单元（仅针对 28/36/44 引脚 PIC32MX1XX/2XX 器件）

配置字	物理地址
DEVCFG0	0x1FC00BFC
DEVCFG1	0x1FC00BF8
DEVCFG2	0x1FC00BF4
DEVCFG3	0x1FC00BF0

在发生上电复位（Power-on Reset， POR）或任意复位时，这些配置字将从引导闪存复制到相应的配置寄存器中。配置位只能编程为 = 0（未编程状态 = 1）。

在编程期间，配置字最多可以编程两次（针对 PIC32MX 器件）和一次（针对 PIC32MZ、PIC32MK 和 PIC32MM 器件），之后就必须执行页擦除操作。

编程配置字之后，器件复位将使新值加载到配置寄存器中。因此，编程器应在校验器件之前编程配置字。最后一步是编程代码保护配置字。

这些配置字确定振荡器源。如果使用 2 线增强型 ICSP 模式，将忽略配置字且器件将始终使用 FRC；然而 4 线模式不同。如果复位后配置字选择的振荡器源在器件上不存在，则编程器在复位后将不能够在器件上执行闪存操作。关于使用配置字选择振荡器的详细信息，请参见具体器件数据手册的 **“特殊功能”**章节。

表 18-3： 配置字单元 （针对 PIC32MZ 系列器件）

配置字（见注 1）	寄存器物理地址			
	固定引导区域 1	固定引导区域 2	活动引导别名区域（见注 2）	非活动引导别名区域（见注 2）
引导序号	0x1FC4FFF0	0x1FC6FFF0	0x1FC0FFF0	0x1FC2FFF0
代码保护	0x1FC4FFD0	0x1FC6FFD0	0x1FC0FFD0	0x1FC2FFD0
DEVCFG0	0x1FC4FFCC	0x1FC6FFCC	0x1FC0FFCC	0x1FC2FFCC
DEVCFG1	0x1FC4FFC8	0x1FC6FFC8	0x1FC0FFC8	0x1FC2FFC8
DEVCFG2	0x1FC4FFC4	0x1FC6FFC4	0x1FC0FFC4	0x1FC2FFC4
DEVCFG3	0x1FC4FFC0	0x1FC6FFC0	0x1FC0FFC0	0x1FC2FFC0
备用引导序号	0x1FC4FF70	0x1FC6FF70	0x1FC0FF70	0x1FC2FF70
备用代码保护	0x1FC4FF50	0x1FC6FF50	0x1FC0FF50	0x1FC2FF50
备用 DEVCFG0	0x1FC4FF4C	0x1FC6FF4C	0x1FC0FF4C	0x1FC2FF4C
备用 DEVCFG1	0x1FC4FF48	0x1FC6FF48	0x1FC0FF48	0x1FC2FF48
备用 DEVCFG2	0x1FC4FF44	0x1FC6FF44	0x1FC0FF44	0x1FC2FF44
备用 DEVCFG3	0x1FC4FF40	0x1FC6FF40	0x1FC0FF40	0x1FC2FF40

注 1： 下列配置字组均须采用 QUAD_WORD_PROGRAM 命令进行编程，以确保正确的 ECC 配置：
- 引导序号 （单四字编程操作）
- 代码保护 （单四字编程操作）
- DEVCFG3、 DEVCFG2、 DEVCFG1 和 DEVCFG0 （单四字编程操作）
- 备用引导序号 （单四字编程操作）
- 备用代码保护 （单四字编程操作）
- 备用 DEVCFG3、备用 DEVCFG2、备用 DEVCFG1 和备用 DEVCFG0 （单四字编程操作）

2： 活动 / 非活动引导别名选择假设用于未编程器件，其中固定区域 1 为活动，固定区域 2 为非活动。关于别名引导区域的详细说明，请参见 《PIC32 系列参考手册》中的**第 48 章 “存储器构成和权限”**（DS60001214）。

表 18-4： 配置字单元 （针对 PIC32MK 系列器件）

配置字（见注 1）	寄存器物理地址			
	固定引导区域 1	固定引导区域 2	活动引导别名区域（见注 2）	非活动引导别名区域 （见注 2）
引导序号	0x1FC43FF0	0x1FC63FF0	0x1FC03FF0	0x1FC23FF0
代码保护	0x1FC43FD0	0x1FC63FD0	0x1FC03FD0	0x1FC23FD0
DEVCFG0	0x1FC43FAC	0x1FC63FAC	0x1FC03FAC	0x1FC23FAC
DEVCFG1	0x1FC43FA8	0x1FC63FA8	0x1FC03FA8	0x1FC23FA8
DEVCFG2	0x1FC43FA4	0x1FC63FA4	0x1FC03FA4	0x1FC23FA4
DEVCFG3	0x1FC43FA0	0x1FC63FA0	0x1FC03FA0	0x1FC23FA0

注 1： 下列配置字组均须采用 QUAD_WORD_PROGRAM 命令进行编程：
- 引导序号 （单四字编程操作）
- 代码保护 （单四字编程操作）
- DEVCFG3、 DEVCFG2、 DEVCFG1 和 DEVCFG0 （单四字编程操作）

2： 活动 / 非活动引导别名选择假设用于未编程器件，其中固定区域 1 为活动，固定区域 2 为非活动。关于别名引导区域的详细说明，请参见 《PIC32 系列参考手册》中的**第 48 章 “存储器构成和权限”**（DS60001214）。

表 18-5： DEVCFG 单元（仅针对 PIC32MM 器件）

配置字	物理地址
DEVID	0x1F803B20
FDEVOPT	0x1F803B50
FICD	0x1F803B60
FPOR	0x1F803B70
FWDT	0x1F803B80
FOSCEL	0x1F803B90
FSEC	0x1F803BA0

18.1.1 配置寄存器保护

为了防止在代码执行期间配置位被意外更改，所有的可编程配置位只可被写入一次。在上电周期内对一个位进行初始化编程之后就不能再次写该位了。更改器件配置时，需要先更改引导闪存中的配置数据，然后关闭再打开器件电源。

为了确保 128 位数据的完整性，器件会不断在每个配置位和其存储补码之间进行比较。如果检测到不匹配，将会产生配置不匹配复位，导致器件复位。

18.2 器件代码保护位（CP）

PIC32 系列器件具有代码保护功能，使能时可防止外部编程设备读取闪存。一旦使能代码保护，只能通过芯片擦除命令（MCHP_ERASE）擦除器件来禁止。

当编程器件选择使用代码保护时，编程器件必须在使能代码保护之前执行校验。使能代码保护应该是编程流程的最后一个步骤。代码保护使能位位置因器件而异。详情请参见具体器件数据手册的**“特殊功能”**章节。

注： 一旦使能代码保护，就不能再读取闪存，并且只能通过芯片擦除命令（MCHP_ERASE）由外部编程器禁止。

18.3 程序写保护位（PWP）

PIC32 系列器件包含写保护功能，可以在程序执行期间防止擦除或写入指定的引导和程序闪存区域。

在 PIC32MX 器件中，写保护通过器件配置字在配置存储器中实现，而在 PIC32MZ、PIC32MK 和 PIC32MM 器件系列中，该功能通过特殊功能寄存器（Special Function Register， SFR）在闪存控制器中实现。

当通过器件配置字实现写保护时，写保护寄存器只有在所有引导和编程闪存编程后才能写入。详情请参见具体器件数据手册的**“特殊功能”**章节。

如果通过 SFR 实现写保护，则在编程闪存区域前，可能需要通过外部编程器在器件初始化期间执行一些步骤。详情请参见具体器件数据手册的**“闪存程序存储器”**章节。

19.0 TAP 控制器

表 19-1： MCHP TAP 指令

命令	值	说明
MTAP_COMMAND	5'h0x07	TDI 和 TDO 与 MCHP 命令移位寄存器连接 （见表 19-2）
MTAP_SW_MTAP	5'h0x04	将 TAP 控制器切换为 MCHP TAP 控制器
MTAP_SW_ETAP	5'h0x05	将 TAP 控制器切换为 EJTAG TAP 控制器
MTAP_IDCODE	5'h0x01	选择芯片标识数据寄存器

19.1 Microchip TAP 控制器 （MTAP）

19.1.1 MTAP_COMMAND 指令

MTAP_COMMAND 用于选择 MCHP 命令移位寄存器。关于可用的命令，请参见表 19-2。

19.1.1.1 MCHP_STATUS 指令

MCHP_STATUS 用于返回 Microchip TAP 控制器的 8 位状态值。表 19-3 列出了所返回状态值的格式。

19.1.1.2 MCHP_ASSERT_RST 指令

MCHP_ASSERT_RST 用于执行持久性器件复位。它类似于将 MCLR 置为有效并保持有效。它关联的状态位为 DEVRST。

19.1.1.3 MCHP_DE_ASSERT_RST 指令

MCHP_DE_ASSERT_RST 用于去除持久性器件复位。它类似于将 MCLR 置为无效。它关联的状态位为 DEVRST。

19.1.1.4 MCHP_ERASE 指令

MCHP_ERASE 用于执行芯片擦除。CHIP_ ERASE 命令会将内部某个用于请求闪存控制器执行擦除操作的位置 1。在控制器正忙时 （由 FCBUSY 状态位指示），该内部位会清零。

19.1.1.5 MCHP_FLASH_ENABLE 指令

MCHP_FLASH_ENABLE 用于将 FAEN 位置 1，该位控制对闪存的处理器访问。 FAEN 位的状态在同名字段中返回。该命令在 CPS = 0 时没有任何作用。该命令需要一条 NOP 指令来完成。

注： PIC32MZ、 PIC32MK 和 PIC32MM 系列器件无需该命令。

19.1.1.6 MCHP_FLASH_DISABLE 指令

MCHP_FLASH_DISABLE 用于将 FAEN 位清零，该位控制对闪存的处理器访问。 FAEN 位的状态在同名字段中返回。该命令在 CPS = 0 时没有任何作用。该命令需要一条 NOP 指令来完成。

注： PIC32MZ、 PIC32MK 和 PIC32MM 系列器件无需该命令。

19.1.2 MTAP_SW_MTAP 指令

MTAP_SW_MTAP 用于将 TAP 指令集切换为 MCHP TAP 指令集。

19.1.3 MTAP_SW_ETAP 指令

MTAP_SW_ETAP 用于将 TAP 指令集切换为 EJTAG TAP 指令集。它的实现方式是，将 EJTAG TAP 控制器保持在运行测试 / 空闲状态，直到 MCHP TAP 控制器解码 MTAP_SW_ETAP 指令为止。

19.1.4 MTAP_IDCODE 指令

MTAP_IDCODE 的返回值存放于 DEVID 寄存器中。

表 19-2： MTAP_COMMAND DR 命令

命令	值	说明
MCHP_STATUS	8'h0x00	执行 NOP 指令并返回状态。
MCHP_ASSERT_RST	8'h0xD1	请求复位控制器将器件复位置为有效。
MCHP_DE_ASSERT_RST	8'h0xD0	去除器件复位请求，如果没有其他请求复位的源 （即， $\overline{\text{MCLR}}$），这会导致复位控制器将器件复位置为无效。
MCHP_ERASE	8'h0xFC	导致闪存控制器执行芯片擦除。
MCHP_FLASH_ENABLE(1)	8'h0xFE	使能取数据并装入闪存 （从处理器中）。
MCHP_FLASH_DISABLE(1)	8'h0xFD	禁止取数据并装入闪存 （从处理器中）。

注 1： PIC32MK 和 PIC32MZ 系列器件无需该命令。

表 19-3： MCHP 状态值

位范围	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
7:0	CPS	0	NVMERR(1)	0	CFGRDY	FCBUSY	FAEN(2)	DEVRST

bit 7 **CPS：**代码保护状态位
1 = 器件不被代码保护
0 = 器件被代码保护

bit 6 **未实现：**读为 0

bit 5 **NVMERR：**NVMCON 状态位 (1)
1 = 在 NVM 工作期间发生错误
0 = 在 NVM 工作期间未发生错误

bit 4 **未实现：**读为 0

bit 3 **CFGRDY：**代码保护状态位
1 = 已读取配置，并且 CP 有效
0 = 未读取配置

bit 2 **FCBUSY：**闪存控制器忙状态位
1 = 闪存控制器正忙 （正在进行擦除操作）
0 = 闪存控制器不忙 （擦除尚未开始或已完成）

bit 1 **FAEN：**闪存访问使能位 (2)
该位反映 CFGCON.FAEN 的状态。
1 = 使能闪存访问
0 = 禁止闪存访问 （即，阻止处理器访问）

bit 0 **DEVRST：**器件复位状态位
1 = 器件复位有效
0 = 器件复位无效

注 1： 该位在 PIC32MX320/340/360/420/440/460 器件上未实现。
2： 该位在 PIC32MK 和 PIC32MZ 系列器件上未实现。

表 19-4： EJTAG TAP 指令

命令	值	说明
ETAP_ADDRESS	5'h0x08	选择地址寄存器。
ETAP_DATA	5'h0x09	选择数据寄存器。
ETAP_CONTROL	5'h0x0A	选择 EJTAG 控制寄存器。
ETAP_EJTAGBOOT	5'h0x0C	将 EjtagBrk、 ProbEn 和 ProbTrap 设置为 1，作为复位值。
ETAP_FASTDATA	5'h0x0E	选择数据寄存器和 Fastdata 寄存器。

19.2 EJTAG TAP 控制器

19.2.1 ETAP_ADDRESS 命令

ETAP_ADDRESS用于选择地址寄存器。只读的地址寄存器提供用于处理器访问的地址。如果存在待完成的处理器访问，则在该寄存器中读取的值有效，否则该值处于未定义状态。

寄存器的2或3个最低有效字节（Least Significant Byte，LSB）与 EJTAG 控制器寄存器的 Psz 字段一起用于指示待完成的处理器访问传输的大小和数据位置。这些位不是直接取自装入 / 存储操作引用的地址。

19.2.2 ETAP_DATA 命令

ETAP_DATA用于选择数据寄存器。在处理器访问期间，读 / 写数据寄存器用于操作码和数据传输。只有存在待完成的处理器访问，并且它用于写操作时，在数据寄存器中读取的值才有效；这种情况下，数据寄存器保存存储值。对于向数据寄存器中写入的值，只有之后完成待完成读操作的处理器访问时，才会使用该值；这种情况下，写入的数据值是用于取数据或装入操作的值。这种行为意味着数据寄存器不是一个可以在写入之后读取先前写入值的存储单元。

19.2.3 ETAP_CONTROL 命令

ETAP_CONTROL 用于选择控制寄存器。 EJTAG 控制器寄存器 （ECR）可以处理：处理器复位和软复位指示、调试模式指示、访问开始、完成与大小，以及读 / 写指示。 ECR 还提供了以下功能：

- 控制所处理的处理器访问的调试向量位置和指示
- 允许调试中断请求
- 指示处理器低功耗模式
- 支持取决于实现的处理器和外设复位

19.2.3.1 EJTAG 控制寄存器 （ECR）

在更新 DR 状态下，除非发生复位，否则 EJTAG 控制器（见寄存器 19-1）寄存器不会被更新 / 写入；即，除非此时 ROCC （bit 31）已经为 0 或被写入 0。这种条件可以确保在复位之后正确地进行处理器访问。

为了在处理器时钟域和TCK时钟域之间正确进行同步，在处理器时钟域中处理器复位后的一定TCK周期之后，可通过 ROCC 位在 TCK 时钟域中指示处理器复位。

除非定义了其他行为，否则在随后发生读操作时，寄存器中的读 / 写 （R/W）位会返回它们的写入值。

内部同步可以确保，即使在 TAP 控制器采用从更新 DR 到捕捉 DR 状态的最短路径时，也可以更新写入值，供之后立即读取。

寄存器 19-1：　　ECR：EJTAG 控制寄存器

位范围	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R-0	R-0	U-0	U-0	U-0	U-0	U-0
	Rocc	Psz<1:0>		—	—	—	—	—
23:16	R-0	R-0	R-0	R/W-0	R-0	R/W-0	U-0	R/W-0
	VPED	Doze	Halt	PerRst	PrnW	PrACC	—	PrRst
15:8	R/W-0	R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0
	ProbEn	ProbTrap	—	EjtagBrk	—	—	—	—
7:0	U-0	U-0	U-0	U-0	R-0	U-0	U-0	U-0
	—	—	—	—	DM	—	—	—

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 31-29　见**注 1**

bit 28-24　**未实现：**读为 0

bit 23-19　见**注 1**

bit 18　**PrACC：**等待处理器访问和控制位

该位指示等待处理器访问并控制等待处理器访问的完成。写入 0 会完成正在等待的处理器访问。写入 1 会被忽略。成功的 FASTDATA 访问将清零该位。

1 = 等待处理器访问
0 = 没有等待的处理器访问

bit 17　**未实现：**读为 0

bit 16　见**注 1**

bit 15　**ProbEn：**处理器访问服务控制位

该位控制探针通过处理器访问服务处理对 DMSEG 段的访问。

1 = 探针服务处理器访问
0 = 探针不服务处理器访问

bit 14　**ProbTrap：**调试异常向量控制单元位

该位控制调试异常向量的位置。

1 = 0xFF200200
0 = 0xBFC00480

bit 13　**未实现：**读为 0

bit 12　**EjtagBrk：**调试中断异常请求位

当该位写为 1 时，该位向处理器请求调试中断异常。写为 0 则忽略。
1 = 调试中断异常请求正在等待
0 = 没有等待的调试中断异常请求

bit 11-4　**未实现：**读为 0

bit 3　见**注 1**

bit 2-0　**未实现：**读为 0

注　1：　关于这些位的说明，请参见 Imagination Technologies Limited 网站（www.imgtec.com）。

19.2.4 ETAP_EJTAGBOOT 命令

ETAP_EJTAGBOOT 命令会使处理器在复位后从调试异常向量处取代码。这允许编程器发送指令到处理器执行，而无需从常规复位向量处取指令。EjtagBrk、ProbTrap和ProbEn位的复位值遵从内部EJTAGBOOT指示的设置。

如果提供了EJTAGBOOT指令且内部EJTAGBOOT有效，则这三位的复位值会置1（1），否则复位值会清零（0）。

这些位置 1 的结果是：

- EjtagBrk 置 1 会导致在 EJTAGBOOT 指令复位处理器之后立刻请求调试中断异常。
- 由于 ProbTrap 置 1，指示调试向量位于 EJTAG 存储器 0xFF200200 处，所以调试处理程序将从 EJTAG 存储器中执行。
- 由于ProbEn置1，所以会指示对处理器访问的处理。

配置后，将发生中断异常且处理器将从 DMSEG 0xFF200200处取处理程序。由于ProbEn置1，处理器将等待探针提供的指令。

19.2.5 ETAP_FASTDATA 命令

ETAP_FASTDATA 命令提供在处理器和探针之间快速传输数据的机制。 Fastdata 寄存器的宽度为 1 位。在快速数据访问期间，将会读写 Fastdata 寄存器（即，移入一位和移出一位）。在快速数据访问期间，移入的 Fastdata 寄存器值指定是否应完成快速数据访问。移出的值是一个标志，指示快速数据访问是否成功（如果已请求要完成访问）。 FASTDATA 访问用于在 DMSEG 段（在探针上）和目标存储器（在处理器上）之间进行高效的块传输。“上传”定义为处理器从目标存储器装入数据并存储到 DMSEG 段中的序列。“下载”是处理器从 DMSEG 段装入数据并存储到目标存储器中的序列。“Fastdata 区域”指定可用于上传和下载的 DMSEG 段地址（0xFF200000 至 0xFF20000F）的合法范围。通过数据和 Fastdata 寄存器（使用 FASTDATA 指令选择）可高效完成待完成的 Fastdata 区域访问。

在Fastdata上传和下载期间，处理器会阻塞对于Fastdata区域的访问。 PrAcc（处理器访问待完成位）将为 1，指示探针需要完成访问。上传和下载访问都通过移入值为 0 的 SPrAcc （请求完成访问）来进行尝试，并通过移出 SPrAcc 来确定尝试是否成功（即，是否存在待完成的访问，并且是否使用了合法的 Fastdata 区域地址）。

下载还会移入数据，用于满足从 DMSEG 段 Fastdata 区域装入数据的操作，而上传则会移出要存储到 DMSEG 段 Fastdata 区域的数据。

如上面所述，要使 Fastdata 访问成功，下列两个条件必须为真：

- PrAcc 必须为 1 （即，必须存在待完成的处理器访问）
- Fastdata 操作必须使用 DMSEG 段（0xFF200000 至 0xFF20000F）中的有效 Fastdata 区域地址

20.0 交流 / 直流特性和时序要求

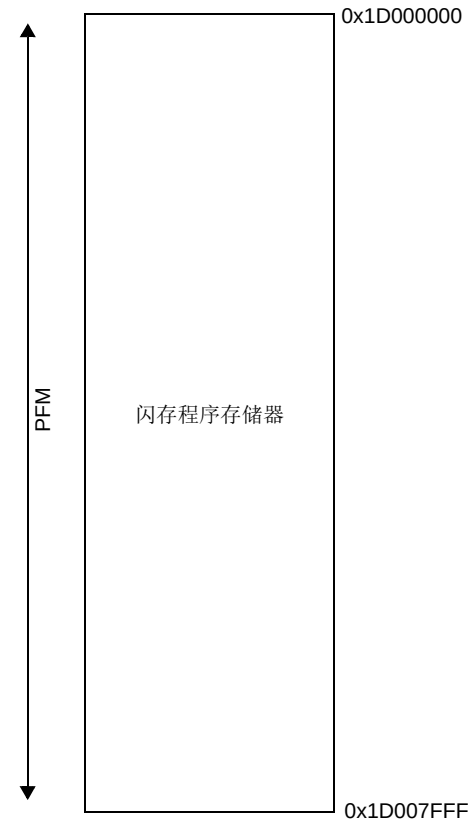
表 20-1： 交流 / 直流特性和时序要求

标准工作条件 工作温度：0ºC 至 +70ºC。建议在 +25ºC 下进行编程。						
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	—	—	V	见**注 1**
D113	IDDP	编程时的供电电流	—	—	mA	见**注 1**
D114	IPEAK	上电期间的瞬时峰值电流	—	—	mA	见**注 1**
D031	VIL	输入低电压	—	—	V	见**注 1**
D041	VIH	输入高电压	—	—	V	见**注 1**
D080	VOL	输出低电压	—	—	V	见**注 1**
D090	VOH	输出高电压	—	—	V	见**注 1**
D012	CIO	I/O 引脚 （PGEDx）上的容性负载	—	—	pF	见**注 1**
D013	CF	VCAP 上的滤波电容值	—	—	µF	见**注 1**
P1	TPGC	串行时钟 （PGECx）周期	100	—	ns	—
P1A	TPGCL	串行时钟 （PGECx）的低电平时间	40	—	ns	—
P1B	TPGCH	串行时钟 （PGECx）的高电平时间	40	—	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	500	—	ns	—
P9A	TDLY4	PE 命令处理时间	40	—	µs	—
P9B	TDLY5	PE 将 PGEDx 驱动为↓到 PE 释放 PGEDx 之间的延时	15	—	µs	—
P11	TDLY7	芯片擦除时间	—	—	ms	见**注 1**
P12	TDLY8	页擦除时间	—	—	ms	见**注 1**
P13	TDLY9	行编程时间	—	—	ms	见**注 1**
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	TDLY8	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	40	—	ns	—
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	40	—	ns	—
P20	TMCLRH	$\overline{\text{MCLR}}$ 高电平时间	—	500	µs	—

注 1： 关于该参数的最小值和最大值，请参见具体器件数据手册中的 **“电气特性”**章节。

附录 A： PIC32 闪存映射

图 A-1： 闪存映射



注： 此处显示的存储器映射仅用作参考。关于具体器件的存储器映射，请参见具体器件数据手册中的 **“存储器构成”** 章节。

附录 B： HEX 文件格式

闪存编程器可以处理Microchip开发工具使用的标准十六进制（hex）格式。支持的格式为 Intel® HEX32 格式（INHX32）。关于 hex 文件格式的更多信息，请参见《MPASM™汇编器/MPLINK™目标链接器/MPLIB™目标库管理器用户指南》（DS33014L_CN）中的**第 1.7.5 节 “Hex 文件格式”**。

hex 文件的基本格式为：

```
:BBAAAATTHHHH...HHHHCC
```

每个数据记录都以 9 字符前缀开始，并总是以 2 字符校验和结束。无论格式如何，所有记录都以 “:” 开始。以下介绍了其中的各个元素。

- BB——2 位数字的十六进制字节计数，代表行中出现的数据字节数。将该数值除以 2 可以得到每行的字数。
- AAAA——4 位数字的十六进制地址，代表数据记录的起始地址。格式为先高字节，后低字节。由于该格式仅支持 8 位，所以地址会增倍。将值除以 2 可以得到真正的器件地址。
- TT——2 位数字的记录类型，对于数据记录为“00”，对于文件结束记录为 “01”，对于扩展地址记录为“04”。
- HHHH——4 位数字的十六进制数据字。格式为先低字节，后高字节。在 TT 之后，将有 BB/2 个数据字。
- CC——2 位数字的十六进制校验和，它是行记录中所有先前字节的和的二进制补码。

由于 Intel hex 文件格式是针对字节的，而 16 位程序计数器不是，所以程序存储器段需要特殊的处理。每个 24 位程序字都通过插入所谓的 “虚拟字节”而扩展为 32 位。每个程序存储器地址都乘以 2 来获得字节地址。

例如，在程序存储器中位于 0x100 的段在 hex 文件中将表示为 0x200。

产生的 hex 文件将具有以下内容：

```
:020000040000fa

:04020000003322110096

:00000001FF
```

数据记录 （第 2 行）的装入地址为 0200，而源代码指定的地址为 0x100。数据是以 “小尾数法”格式表示的，表示最低有效字节先出现，虚拟字节最后出现，位于校验和之前。

附录 C： 版本历史

版本 A （2007 年 8 月）

这是本文档的初始版本。

版本 B （2008 年 2 月）

无版本更新记录。

版本 C （2008 年 4 月）

无版本更新记录。

版本 D （2008 年 5 月）

无版本更新记录。

版本 E （2009 年 7 月）

文档的该版本包含了以下新增内容和更新：

- 对整篇文档的样式和格式进行了少量更改
- 增加了以下器件：
  - PIC32MX565F256H
  - PIC32MX575F512H
  - PIC32MX675F512H
  - PIC32MX795F512H
  - PIC32MX575F512L
  - PIC32MX675F512L
  - PIC32MX795F512L
- 在图 7-1 中，将 MCLR 脉冲线更新为指示高电平有效 （P20）
- 更新了表 11-1 的步骤 7，以说明在步骤中需要重复*最后一条*指令
- 更新了表 13-1 中的以下指令：
  - 步骤 1 中的第 7 条、第 9 条和第 11 条指令
  - 步骤 2 中的所有指令
  - 步骤 3 中的第 1 条指令
  - 步骤 4 中的第 3 条指令
- 在表 17-1 中增加了以下器件：
  - PIC32MX565F256H
  - PIC32MX575F512H
  - PIC32MX575F512L
  - PIC32MX675F512H
  - PIC32MX675F512L
  - PIC32MX795F512H
  - PIC32MX795F512L
- 更新了表 17-2 中的地址值

版本 E （2009 年 7 月）（续）

- 在表 17-5 中增加了以下器件：
  - PIC32MX565F256H
  - PIC32MX575F512H
  - PIC32MX675F512H
  - PIC32MX795F512H
  - PIC32MX575F512L
  - PIC32MX675F512L
  - PIC32MX795F512L
- 在“DEVCFG 器件配置字汇总”和“DEVCFG3：器件配置字 3”中增加了注 1-3 和以下位 （见表 18-1 和寄存器 18-4）：
  - FVBUSIO
  - FUSBIDIO
  - FCANIO
  - FETHIO
  - FMIIEN
  - FPBDIV<1:0>
  - FJTAGEN
- 更新了 DEVID 汇总 （见表 18-1）
- 在“DEVCFG0：器件配置字 0”中更新了 ICESEL 位说明并增加了 FJTAGEN 位 （见寄存器 16-1）
- 更新了 “DEVID：器件和版本 ID 寄存器”
- 增加了 “器件 ID 和版本”表 （表 18-4）
- 在表 20-1 中增加了 MCLR 高电平时间（参数 P20）
- 增加了**附录 B：“Hex 文件格式”**和**附录 C：“版本历史”**

版本 F （2010 年 4 月）

文档的该版本包含了以下新增内容和更新：

- 进行了以下全局位名称更改：
  - NVMWR 重命名为 WR
  - NVMWREN 重命名为 WREN
  - NVMERR 重命名为 WRERR
  - FVBUSIO 重命名为 FVBUSONIO
  - FUPLLEN 重命名为 UPLLEN
  - FUPLLIDIV 重命名为 UPLLIDIV
  - POSCMD 重命名为 POSCMOD
- 更新了**第 2.0 节“编程概述”**第 4 段中的 PIC32 系列数据手册引用
- 更新了**第 5.2.2 节 “2 相 ICSP”**中的注释
- 更新了启动闪存行写操作码和指令 （见表 13-1 中的步骤 4、步骤 5 和步骤 6）

版本 F （2010 年 4 月）（续）

- 增加了以下器件：
  - PIC32MX534F064H
  - PIC32MX534F064L
  - PIC32MX564F064H
  - PIC32MX564F064L
  - PIC32MX564F128H
  - PIC32MX564F128L
  - PIC32MX575F256L
  - PIC32MX664F064H
  - PIC32MX664F064L
  - PIC32MX664F128H
  - PIC32MX664F128L
  - PIC32MX675F256H
  - PIC32MX675F256L
  - PIC32MX695F512H
  - PIC32MX605F512L
  - PIC32MX764F128H
  - PIC32MX764F128L
  - PIC32MX775F256H
  - PIC32MX775F256L
  - PIC32MX775F512H
  - PIC32MX775F512L

版本 G （2010 年 8 月）

文档的该版本包含了以下更新：

- 更新了 “表 11-1：下载 PE”中的步骤 3
- 对整篇文档的格式和文字进行了少量修正

版本 H （2011 年 4 月）

文档的该版本包含了以下更新：

- 对整篇文档的格式和文字进行了少量修正
- 增加了以下器件：
  - PIC32MX110F016B
  - PIC32MX110F016C
  - PIC32MX110F016D
  - PIC32MX120F032B
  - PIC32MX120F032C
  - PIC32MX120F032D
  - PIC32MX210F016B
  - PIC32MX210F016C
  - PIC32MX210F016D
  - PIC32MX220F032B
  - PIC32MX220F032C
  - PIC32MX220F032D
- 在表 17-1 中增加了以下两行：
  - PIC32MX1X0
  - PIC32MX2X0
- 在**第 17.4.6 节 “器件被代码保护时的校验和值”**中新增子章节
- 移除寄存器 18-1 至寄存器 18-5
- 移除表 17-2
- 移除第 17.5 节“PIC32 器件的校验和”及其子章节
- 移除 “闪存程序存储器写保护范围”表（之前的表 18-4）
- 增加了“仅针对 PIC32MX1X0 和 PIC32MX20X 器件的 DEVCFG 单元”（见表 18-3）
- 在**第 18.0 节 “配置存储器和器件 ID”**中，移除表 18-1，并更新表 18-2：DEVID 汇总作为表 18-1
- 在 MCHP 状态值表中增加 NVMERR 位（见表 19-3）
- 在表 18-4 中增加了以下硅片版本和版本 ID：
  - 0x5 - B6 版本
  - 0x1 - A1 版本
- 在闪存映射中增加了一个注释 （见图 A-1）
- 增加了**附录 C：“闪存程序存储器数据手册声明”**

版本 J （2011 年 8 月）

注：	本文档的版本历史从版本 H 直接跳到版本 J，这是为了避免混淆大写字母“I”（EY）与小写字母 “l”（EL）。

文档的该版本包含了以下更新：

- 将所有 VCORE/VCAP 修改为 VCAP
- 更新了**第 2.0 节 “编程概述”**的第四段
- 移除了 “2 线接口引脚”表中的 “编程器引脚名称”栏位，并更新了 MCLR 的引脚类型 （见表 4-2）
- 向 “代码存储区容量”表 （见表 5-1）和 “器件 ID 和版本”表 （见表 18-4）添加了以下新器件：
  - PIC32MX130F064B
  - PIC32MX130F064C
  - PIC32MX130F064D
  - PIC32MX150F128B
  - PIC32MX150F128C
  - PIC32MX150F128D
  - PIC32MX230F064B
  - PIC32MX230F064C
  - PIC32MX230F064D
  - PIC32MX250F128B
  - PIC32MX250F128C
  - PIC32MX250F128D
- 向 “代码存储区容量”表添加了 “行大小”和 “页大小”栏位 （见表 5-1）

版本 J（2011 年 8 月）（续）

- 更新了“进入增强型 ICSP 模式”图中的 PGCx 信号（见图 7-1）
- 更新了“擦除器件框图”（见图 9-1）
- 在**第 9.0 节“擦除器件”**中向擦除目标器件流程中添加了新的步骤 4
- 更新了“2 线退出测试模式”图中的 $\overline{\text{MCLR}}$ 信号（见图 15-2）
- 在“PE 命令集”表中更新了以下命令并修改了注 2（见表 16-2）：
  - PROGRAM_CLUSTER
  - GET_DEVICEID
  - CHANGE_CFG
- 向**第 16.2.11 节“GET_CRC 命令”**中添加了注 2
- 在“PROGRAM_CLUSTER 格式”表中更新了地址和长度说明（见表 16-13）
- 在“CHANGE_CFG 响应”图后添加注释（见图 16-27）
- 在表 17-1 中更新了所有 PIC32MX1XX 和 PIC32MX2XX 器件中 DEVCFG0 和 DEVCFG1 的值
- 对“交流 / 直流特性和时序要求”表进行了如下修改（见表 20-1）：
  - 更新了参数 D111（V_{DD}）中的最小值
  - 添加参数 D114（I_{PEAK}）
  - 移除参数 P2、P3、P4、P4A、P5、P8 和 P10
- 移除附录 C：“闪存程序存储器数据手册声明”
- 对整个文档的内容和格式进行了少量更新

版本 K（2012 年 7 月）

文档的该版本包含了以下更新：

- 将所有 PGC 和 PGD 分别更改为：PGEC 和 PGED
- 更新**第 1.0 节“器件概述”**，添加了该文档中的主要章节列表
- 添加了**第 2.3 节“数据大小”**
- 更新了**第 4.0 节“连接到器件”**
- 向“片上稳压器连接图”添加注 2（见图 4-2）
- 向 4 线和 2 线接口引脚表添加注 2（见表 4-1 和表 4-2）
- 更新了**第 7.0 节“进入 2 线增强型 ICSP 模式”**
- 更新了“进入串行执行模式”图（见图 10-1）
- 更新了**第 10.2 节“2 线接口”**中的步骤 11
- 更新了**第 12.2 节“使用 PE”**
- 更新了“启动闪存行写操作码”表中的步骤 3（见表 13-1）
- 更新了“校验器件操作码”表中的步骤 1（见表 14-1）
- 更新了**第 15.1 节“4 线接口”**和**第 15.2 节“2 线接口”**中的时间间隔
- 在**第 16.0 节“编程执行程序”**中添加了 PE 位置相关的注释
- 在整个**第 16.2 节“PE 命令集”**中，添加了操作数字段
- 更新了 PROGRAM 命令算法（见图 16-9）
- 对所有 PIC32MX1XX 和 PIC32MX2XX 器件更新了掩码值，对所有器件更新了 DEVCFG3（见表 17-1）
- 更新了 DCR 值（见**第 17.4.3 节“计算校验和公式中的“DCR””**和表 17-2）
- 更新了校验和计算过程（见例 17-1）
- 向“代码存储区容量”表（见表 5-1）和“器件 ID 和版本”表（见表 18-4）添加了以下新器件：
  - PIC32MX420F032H
  - PIC32MX330F064H
  - PIC32MX330F064L
  - PIC32MX430F064H
  - PIC32MX430F064L
  - PIC32MX340F128H
  - PIC32MX340F128L
  - PIC32MX350F128H
  - PIC32MX350F128L
  - PIC32MX350F256H
  - PIC32MX350F256L
  - PIC32MX440F128H
  - PIC32MX440F128L
  - PIC32MX450F128H
  - PIC32MX450F128L
  - PIC32MX440F256H
  - PIC32MX450F256H
  - PIC32MX450F256L
  - PIC32MX460F256L
  - PIC32MX340F512H
  - PIC32MX360F512H
  - PIC32MX370F512H
  - PIC32MX370F512L
  - PIC32MX440F512H
  - PIC32MX460F512L
  - PIC32MX470F512H
  - PIC32MX470F512L
- 向**第 18.2 节“器件代码保护位（CP）”**添加了一个注释
- 添加了 EJTAG 控制寄存器（见寄存器 19-1）
- 更新了**第 19.2.4 节“ETAP_EJTAGBOOT 命令”**
- 更新了“交流 / 直流特性和时序要求”表（见表 20-1）：
  - 移除了参数 D112
  - 将注 1 和 2 替换为新注 1
  - 更新了参数 D111、D113、D114、D031、D041、D080、D090、D012、D013、P11、P12 和 P13
- 对整个文档的内容和格式进行了少量更新

版本 L （2013 年 1 月）

文档的该版本包含了以下更新：

- 新增或更新了以下章节：
  - **第 2.1 节 “具有双闪存分区和双引导区域的器件”**（新）
  - **第 4.3 节 “电源要求”**
  - **第 13.0 节 “启动闪存行写操作”**
  - **第 16.1.1 节 “2 线 ICSP EJTAG 速率”**
- 更新了器件配置寄存器掩码值 （见表 17-1）
- 向 “代码存储区容量”表 （见表 5-1）和 “器件 ID 和版本”表 （见表 18-4）添加了以下新器件：
  - PIC32MZ0256ECE064
  - PIC32MZ0256ECE100
  - PIC32MZ0256ECE124
  - PIC32MZ0256ECE144
  - PIC32MZ0256ECF064
  - PIC32MZ0256ECF100
  - PIC32MZ0256ECF124
  - PIC32MZ0256ECF144
  - PIC32MZ0512ECE064
  - PIC32MZ0512ECE100
  - PIC32MZ0512ECE124
  - PIC32MZ0512ECE144
  - PIC32MZ0512ECF064
  - PIC32MZ0512ECF100
  - PIC32MZ0512ECF124
  - PIC32MZ0512ECF144
  - PIC32MZ1024ECE064
  - PIC32MZ1024ECE100
  - PIC32MZ1024ECE124
  - PIC32MZ1024ECE144
  - PIC32MZ1024ECF064
  - PIC32MZ1024ECF100
  - PIC32MZ1024ECF124
  - PIC32MZ1024ECF144
  - PIC32MZ1024ECG064
  - PIC32MZ1024ECG100
  - PIC32MZ1024ECG124
  - PIC32MZ1024ECG144
  - PIC32MZ1024ECH064
  - PIC32MZ1024ECH100
  - PIC32MZ1024ECH124
  - PIC32MZ1024ECH144
  - PIC32MZ2048ECG064
  - PIC32MZ2048ECG100
  - PIC32MZ2048ECG124
  - PIC32MZ2048ECG144
  - PIC32MZ2048ECH064
  - PIC32MZ2048ECH100
  - PIC32MZ2048ECH124
  - PIC32MZ2048ECH144
- 向 “PE 命令集”表中添加了注 3、注 4、`GET_CHECKSUM` 和 `QUAD_WORD_PRGM` 命令（见表 16-2）
- 添加**第 16.2.15 节 “GET_CHECKSUM 命令”**
- 添加**第 16.2.16 节 “QUAD_WORD_PROGRAM 命令”**
- 更新了 DEVCFG 单元中的全部地址（见表 18-1 和表 18-2）
- 向 PIC32MZ EC 系列器件添加配置字单元（见表 18-3）
- 更新了**第 18.2 节 “器件代码保护位 （CP）”**
- 更新了**第 18.3 节 “程序写保护位 （PWP）”**
- 在整个文档中，将所有 “测试模式”更新为 “编程模式”
- 对整个文档的内容和格式进行了少量更新

版本 M （2013 年 9 月）

文档的该版本包含了以下更新：

- 将所有 MIPS Technologies Inc. 和 www.mips.com 分别更新为 Imagination Technologies Limited 和 www.imgtec.com
- 更新了**第 2.0 节 “编程概述”**
- 更新了**第 5.1.6 节 “闪存”**中的最后一段
- 更新了 “代码存储区容量”表并添加了注 3 （见表 5-1）
- 更新了 “擦除器件”流程图 （见图 9-1）
- 更新表 11-1 中的步骤 1、 2、 3 和 5
- 在**第 13.2 节 “不使用 PE”**中添加了一个段落
- 更新了表 13-1 中的步骤 2、 3 和 5
- 更新了表 16-17 中的操作码说明
- 更新了 “器件配置掩码值”表 （见表 17-1）
- 移除了**第 17.3 节 “算法”**中第四段的第一句话
- 更新了 “器件 ID 和版本”表 （见表 18-4）

版本 N （2014 年 4 月）

文档的该版本包含了以下更新：

- 更新了**表 4-1：“4 线接口引脚”**中的注 2
- 更新了**表 4-2：“2 线接口引脚”**中的注 2
- 更新了**第 9.0 节 “擦除器件”**的步骤 5 中的延时值
- 在 “器件 ID 和版本”表 （见表 18-4）中，更新了 “版本 ID”和 “芯片版本”列表，并添加了以下新器件：3
  - PIC32MX170F256B
  - PIC32MX170F256D
  - PIC32MX270F256B
  - PIC32MX270F256D
  - PIC32MX330F064H
  - PIC32MX330F064L
  - PIC32MX350F128H
  - PIC32MX350F128L
  - PIC32MX350F256H
  - PIC32MX350F256L
  - PIC32MX430F064H
  - PIC32MX430F064L
  - PIC32MX450F128H
  - PIC32MX450F128L
  - PIC32MX450F256H
  - PIC32MX450F256L

版本 P （2014 年 10 月）

注： 本文档的版本历史从版本 N 直接跳到版本 P，这是为了避免混淆大写字母“O”（OH）与数字零“0”。

本编程规范更新了如下内容，以包含 PIC32MM 器件系列：

- 更新了**表 5-1：“代码存储区容量”**
- 新增了**第 6.6 节 “ReadFromAddress 伪操作”**
- 新增了**表 11-3：“下载 PE （仅针对 PIC32MM 器件）”**
- 新增了**表 11-4：“PE 加载程序操作码 （仅针对 PIC32MM 器件）”**
- 新增了**表 12-2：“下载数据操作码 （仅针对 PIC32MM 器件）”**
- 新增了**表 13-2：“启动闪存行写操作码 （仅针对 PIC32MM 器件）”**
- 更新了**表 16-2：“PE 命令集”**
- 新增了**表 17-2：“PIC32MM 器件的器件配置寄存器的掩码值”**
- 新增了**表 18-5：“DEVCFG 单元 （仅针对 PIC32MM 器件）”**

还进行了如下更新：

- 更新了**表 5-1：“代码存储区容量”**以包含 PIC32MK 器件信息
- 更新了**表 17-1：“器件配置寄存器的掩码值（当前支持 PIC32MX、 PIC32MZ 和 PIC32MK 器件）”**以包含 PIC32MK 器件信息
- 删除了原有的表 18-4 器件 ID 和版本，该信息现在可从当前的系列芯片勘误表中获得
- 新增了**表 18-4：“配置字单元 （针对 PIC32MK 系列器件）”**

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。

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。除非另外声明，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。

ISBN：978-1-63277-623-5

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 - 东莞
Tel: 86-769-8702-9880

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: 886-7-213-7828

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

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

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

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

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

印度 India - Pune
Tel: 91-20-3019-1500

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

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

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

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

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

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

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

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

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

欧洲

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

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

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

德国 Germany - Dusseldorf
Tel: 49-2129-3766400

德国 Germany - Karlsruhe
Tel: 49-721-625370

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

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

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

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

波兰 Poland - Warsaw
Tel: 48-22-3325737

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

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

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

07/14/15