PIC32MM 系列闪存编程规范

1.0 器件概述

本文档定义了 PIC32MM 系列 32 位单片机的编程规范。本编程规范旨在为外部编程工具的开发人员提供指导。为 PIC32MM 器件开发应用的客户应该使用已支持器件编程的开发工具。

本文档包含以下器件的编程规范：

- PIC32MM0016GPL020
- PIC32MM0032GPL020
- PIC32MM0064GPL020
- PIC32MM0016GPL028
- PIC32MM0032GPL028
- PIC32MM0064GPL028
- PIC32MM0016GPL036
- PIC32MM0032GPL036
- PIC32MM0064GPL036
- PIC32MM0064GPM028
- PIC32MM0128GPM028
- PIC32MM0256GPM028
- PIC32MM0064GPM036
- PIC32MM0128GPM036
- PIC32MM0256GPM036
- PIC32MM0064GPM048
- PIC32MM0128GPM048
- PIC32MM0256GPM048
- PIC32MM0064GPM064
- PIC32MM0128GPM064
- PIC32MM0256GPM064

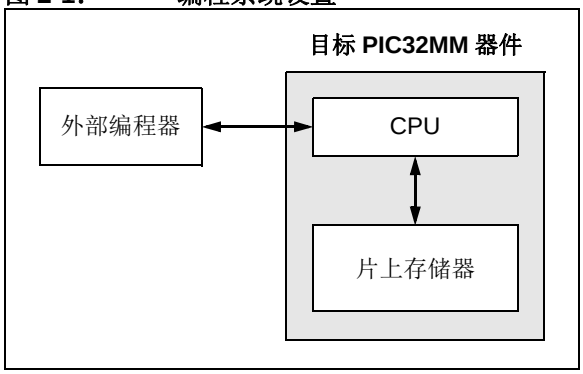
主要的讨论主题包括：

2.0 编程概述

在开发编程工具时，需要了解目标器件的内部闪存编程操作以及用于控制闪存编程的特殊功能寄存器（Special Function Register， SFR），因为这些操作和寄存器将由外部编程工具及其软件使用。特定器件数据手册的**“闪存程序存储器”**章节以及《PIC32 系列参考手册》的相关章节介绍了这些操作和控制寄存器。强烈建议将这些文档与本编程规范配合使用。

外部工具编程设置由外部编程器工具和目标 PIC32MM 器件组成。图 2-1 显示了典型的编程设置。编程器工具用于执行必要的编程步骤和完成编程操作。

图 2-1： 编程系统设置



2.1 编程接口

所有 PIC32MM 器件都能为外部编程器工具提供两个物理接口：

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

更多信息，请参见**第 4.0 节 “连接到器件”**。

这两种方法都可以使用可下载的编程执行程序（Programming Executive，PE）。PE 可从目标器件 RAM中执行并对编程器隐藏器件编程详细信息。它还可以免除与数据传输关联的开销并提高总数据吞吐量。Microchip 已经开发了可与任何外部编程器配合使用的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”说明了高级编程步骤，并简要介绍了每个步骤。有关详细介绍，可参见本文档的对应小节。

有关编程命令、EJTAG 和 DC 规范的更多信息，请参见以下小节：

- **第 18.0 节 “配置存储区器件 ID 和唯一器件标识符”**
- **第 19.0 节 “TAP 控制器”**
- **第 20.0 节 “交流 / 直流特性和时序要求”**

2.2 增强型 JTAG （EJTAG）

2 线 ICSP 和 4 线 JTAG 接口使用 EJTAG 协议来与编程器交换数据。本文档按需提供了此协议的工作原理，但建议高级用户访问 Imagination Technologies Limited 网站 （www.imgtec.com），以了解更多信息。

2.3 数据大小

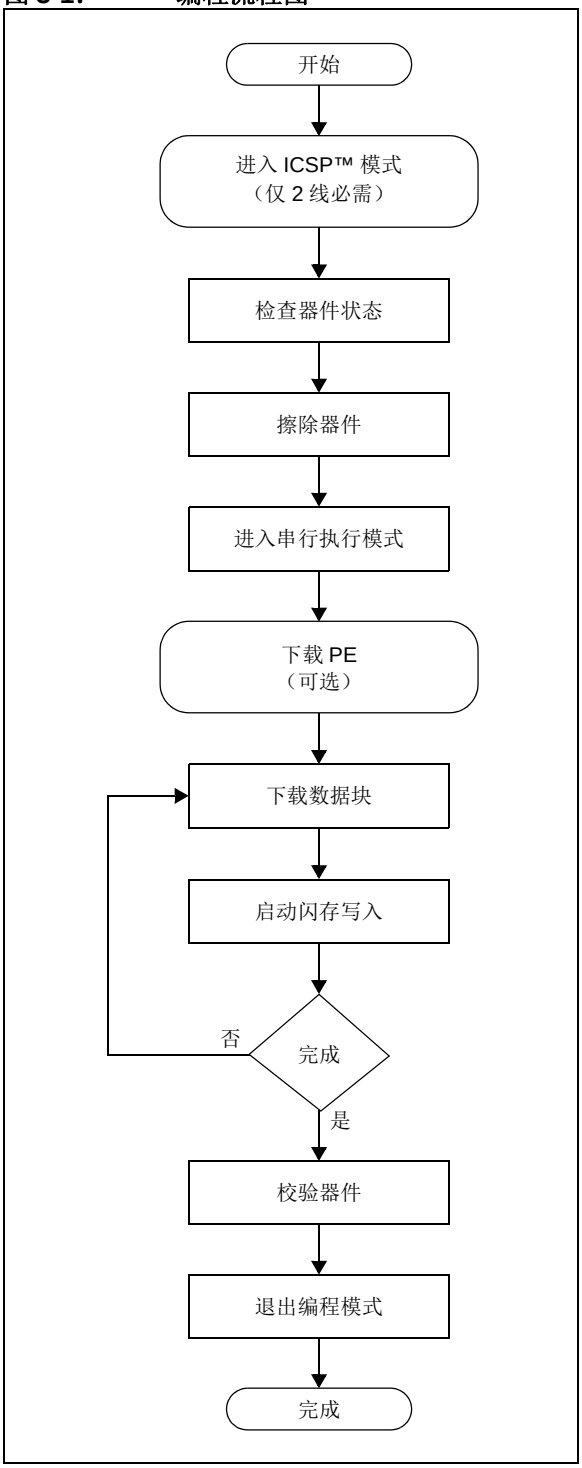
数据大小定义如下：

- 双字：64 位
- 单字：32 位
- 半字：16 位
- 四分之一字：8 位
- 一个字节：8 位

3.0 编程步骤

所有工具编程器都必须执行一系列常用步骤，与实际使用的方法无关。图 3-1 显示了用于 PIC32MM 器件编程的一系列步骤。

图 3-1： 编程流程图



下面按顺序列出了编程步骤，并简要介绍了每个步骤。有关各个步骤的更多详细信息，请参见以下小节。

1. 连接到目标器件。
   为确保成功编程，必须将所有必要的引脚都连接到对应的信号。更多信息，请参见本文档中的**第 4.0 节 “连接到器件”**。
2. 将目标器件置于编程模式。
   对于 2 线编程方法，必须在执行任何其他步骤之前将目标器件置于特殊的编程模式。

注： 对于 4 线编程方法，第 2 步并非必要步骤。

   更多信息，请参见**第 7.0 节 “进入 2 线 ICSP 模式”**。
3. 检查器件的状态。
   第 3 步用于检查器件状态，以确保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块为非空白，或器件处于代码保护状态，则必须在对任何新数据进行编程之前执行擦除步骤。
   更多信息，请参见**第 9.0 节 “擦除器件”**。
5. 进入编程模式。
   第 5 步用于校验器件是否未被代码保护，并引导 TAP 控制器，以开始将数据发送到 PIC32MM CPU 并从其接收数据。
   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 是可下载到目标器件 RAM 中的一小段可执行代码。它将接收实际数据并对其进行编程。

注： 如果使用的编程方法不需要 PE，则第 6 步并非必要步骤。

   更多信息，请参见**第 11.0 节 “下载编程执行程序 （PE）”**。
7. 下载要编程的数据块。
   不管是否需要 PE，所有方法都必须将所需编程数据下载到 RAM 中的存储块中。
   更多信息，请参见**第 12.0 节 “下载数据块”**。
8. 启动闪存写入。
   将每个数据块下载到 RAM 中后，必须启动编程序列，以将其编程到目标器件的闪存中。
   更多信息，请参见**第 13.0 节 “启动闪存行写入”**。
9. 重复第 7 步和第 8 步，直到所有数据块都已下载并编程。
10. 校验程序存储器。
    在将所有编程数据和配置位都编程后，应对目标器件存储器执行读回操作并校验其内容是否与编程数据匹配。
    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。
    只有在目标器件断开电源并向目标器件重新加电，或执行退出编程序列之后，新编程的数据才会有效。
    更多信息，请参见**第 15.0 节 “退出编程模式”**。

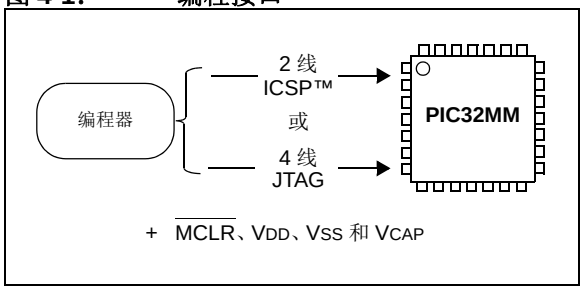
4.0 连接到器件

PIC32MM 系列为存储器内容的连接和编程提供了两个可用的物理接口（见图 4-1）。对于所有编程接口，都必须对目标器件供电并连接所有必需信号。此外，还必须使能接口，如 4 线 JTAG 接口通过其配置位使能， 2 线 ICSP 接口则通过特殊初始化序列使能。

在默认情况下，出厂的空白器件会使能 JTAG 接口。

有关如何使能 ICSP，请参见**第 7.0 节“进入 2 线 ICSP 模式”**。

图 4-1： 编程接口



4.1 4 线 JTAG 接口

其中一个可用的接口是 4 线 JTAG （IEEE 1149.1）端口。表 4-1 列出了必需的引脚连接。此接口使用以下四条通信线来将数据传输到要编程的 PIC32MM 器件以及从其接收数据：

- 测试时钟输入 （Test Clock Input， TCK）
- 测试模式选择输入 （Test Mode Select Input，TMS）
- 测试数据输入 （Test Data Input， TDI）
- 测试数据输出 （Test Data Output， TDO）

有关信号与器件引脚之间的连接，请参见特定器件的数据手册。

4.1.1 测试时钟输入 （TCK）

TCK 是通过指令寄存器（Instruction Register，IR）或选定数据寄存器控制 TAP 控制器更新和数据移位的时钟。 TCK 在频率和相位上都不依赖于处理器时钟。

4.1.2 测试模式选择输入 （TMS）

TMS 是 TAP 控制器的控制信号。将在 TCK 的上升沿对此信号进行采样。

4.1.3 测试数据输入 （TDI）

TDI 是指令寄存器或选定数据寄存器的测试数据输入。在某些 TAP 控制器状态下，将在 TCK 的上升沿对此信号进行采样。

4.1.4 测试数据输出 （TDO）

TDO 是来自指令寄存器或数据寄存器的测试数据输出。此信号在 TCK 的下降沿发生改变。TDO 只有在移出数据时才会被驱动，否则 TDO 处于三态。

表 4-1： 4 线接口引脚

器件引脚名称	引脚类型	引脚说明
MCLR	I	编程使能
VDD 和 AVDD(1)	P	电源
VSS 和 AVSS(1)	P	地
VCAP	P	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	测试模式选择

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）和地 （AVSS）。

4.2 2 线 ICSP 接口

另一个可用的接口是 2 线 ICSP 端口。表 4-2 列出了必需的引脚连接。此接口使用以下两条通信线将数据传输到要编程的 PIC32MM 器件以及从其接收数据：

- 串行编程时钟 （PGECx）
- 串行编程数据 （PGEDx）

以下两个小节介绍了这两个信号。有关信号与芯片引脚之间的连接，请参见特定器件的数据手册。

4.2.1 串行编程时钟 （PGECx）

PGECx 是通过指令寄存器或选定数据寄存器控制 TAP 控制器更新和数据移位的时钟。 PGECx 在频率和相位上都不依赖于处理器时钟。

4.2.2 串行编程数据 （PGEDx）

PGEDx 是指令寄存器或选定数据寄存器的数据输入 / 输出。它也是 TAP 控制器的控制信号。在某些 TAP 控制器状态下，将在 PGECx 的下降沿对此信号进行采样。

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
V_{DD} 和 AV_{DD}(1)	V_{DD}	P	电源
V_{SS} 和 AV_{SS}(1)	V_{SS}	P	地
V_{CAP}(2)	N/A	P	滤波电容连接
PGECx(2)	PGEC	I	编程引脚对：串行时钟
PGEDx(2)	PGED	I/O	编程引脚对：串行数据

图注： I = 输入 O = 输出 P = 电源

注 1： 所有电源和地引脚都必须连接，包括模拟电源 （AV_{DD}）和地 （AV_{SS}）。

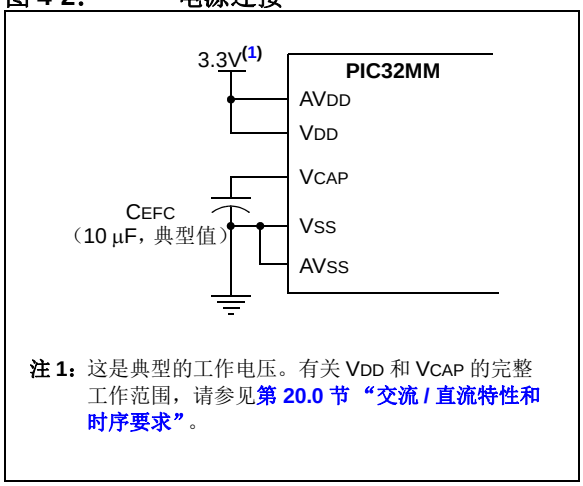
2： 可以使用特定器件上任何一对可用的编程引脚，但是引脚必须成对使用 （PGED1 必须与 PGEC1 一起使用，以此类推）。

4.3 电源要求

PIC32MM 系列中的器件为双电源设计。其中一个为内核供电，另一个为外设和 I/O 引脚供电。所有器件都包含一个用于低压内核电源的片上稳压器，因此无需附加外部稳压器。为确保器件正常运行，需要一个称为 VCAP 的外部电容。

有关器件的电源要求，请参见特定器件数据手册中的**第 20.0 节“交流 / 直流特性和时序要求”**和**“电气特性”**章节。

图 4-2： 电源连接



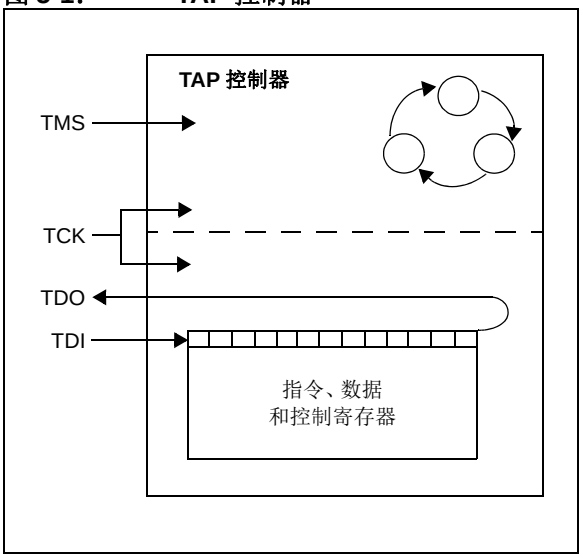
注 1：这是典型的工作电压。有关 VDD 和 VCAP 的完整工作范围，请参见**第 20.0 节“交流 / 直流特性和时序要求”**。

5.0 4 线 JTAG 与 ICSP

通过 CPU 内核中的 EJTAG 模块完成编程。 EJTAG 与一组完整的 JTAG 引脚连接，或者与精简的 ICSP 模式 2 线转 4 线 EJTAG 接口连接。在两种模式下，PIC32MM 闪存的编程都是通过 ETAP 控制器完成。TAP 控制器使用 TMS 引脚来确定是否应该在 TDI 和 TDO 之间的移位路径中访问指令寄存器或数据寄存器 （参见图 5-1）。

用于编程的基本EJTAG概念是使用一个称为DMSEG的特殊存储区 （0xFF200000 到 0xFF2FFFFF），该区域只有在处理器以调试模式运行时才可用。所有指令以串行方式移入内部缓冲区，然后装入指令寄存器中，并由 CPU 执行。指令通过 ETAP 状态机以 32 位字馈送。

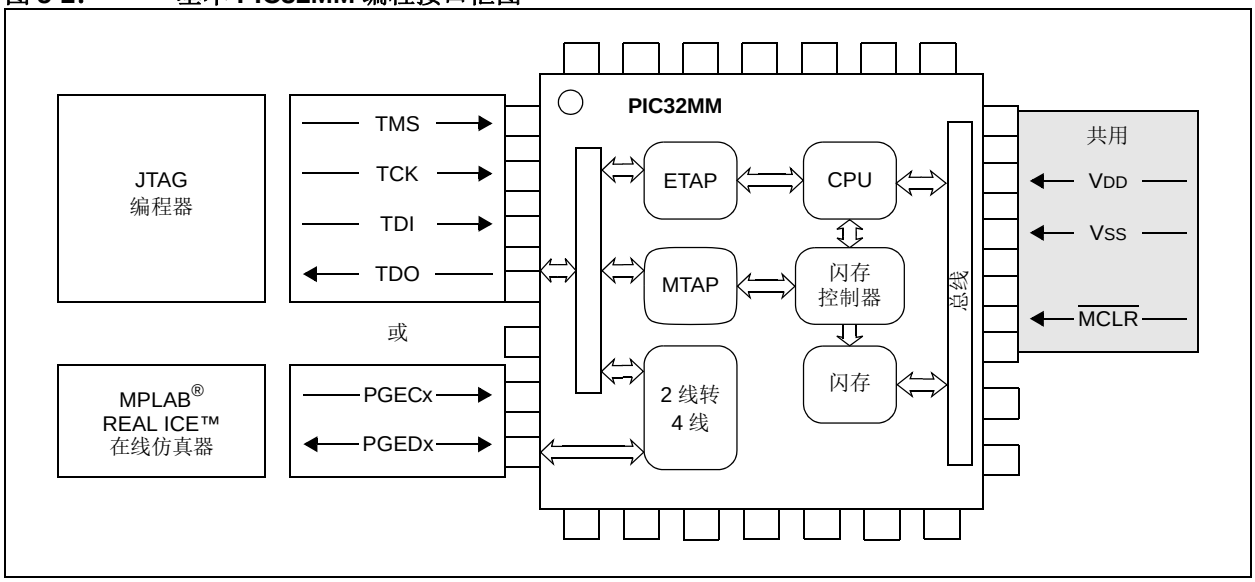
图 5-1： TAP 控制器



5.1 编程接口

图 5-2 显示了 PIC32MM 器件中的基本编程接口。后续小节中提供了有关每个接口模块的说明。

图 5-2： 基本 PIC32MM 编程接口框图



5.1.1 ETAP

此模块以串行方式将指令和数据馈送到 CPU 中。

5.1.2 MTAP

除了 EJTAG TAP（ETAP）控制器，PIC32MM 器件还使用另一个专有TAP控制器来执行其他操作。Microchip TAP（MTAP）控制器支持与编程相关的两条指令：MTAP_COMMAND 和 TAP 切换指令。有关完整的指令列表，请参见表 19-1。MTAP_COMMAND 指令为 JTAG 探针提供了通过其数据寄存器将命令发送到器件的机制。

编程器通过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（有关具体命令，请参见表 19-2）。

探针无需为移入数据寄存器的每条命令都发出MTAP_COMMAND 指令。

5.1.3 2 线转 4 线

此模块可将 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

PIC32MM 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，PFM 则从地址 0x1D000000 开始。每个闪存分区都可分成多页，页表示可以擦除的最小存储块。页大小为 256 字（1024 字节）、512 字（2048 字节）、1024 字（4096 字节）或 4096 字（16,384 字节），具体取决于器件。行大小表示使用行编程命令编程的字数。一页内通常有 8 行；因此，页大小为 256、512、1024 和 4096 字的器件的行大小分别为 32、64、128 和 512 字。表5-1显示了每个器件系列的PFM、BFM、行和页大小。

BFM 的某些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“配置存储区器件 ID 和唯一器件标识符”**。

表 5-1：　　代码存储区大小

PIC32MM 器件	行大小（字）	页大小（字）	引导闪存地址（字节）(1)	编程执行 (2,3)
PIC32MM0016/0032/0064/0128/0256GPL0XX	64	512	0x1FC00000-0x1FC016FF（5.75K）	RIPE_20_aabbcc.hex

注 1： 程序闪存地址的范围基于程序闪存大小，具体如下所示：

- 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）

并非每个系列都支持所有的程序闪存大小。

2： 编程执行程序可以从 Microchip 网站的相关产品页面获取，也可能位于以下 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 aabbcc 因文件版本而异。

5.2 4 线 JTAG 详细信息

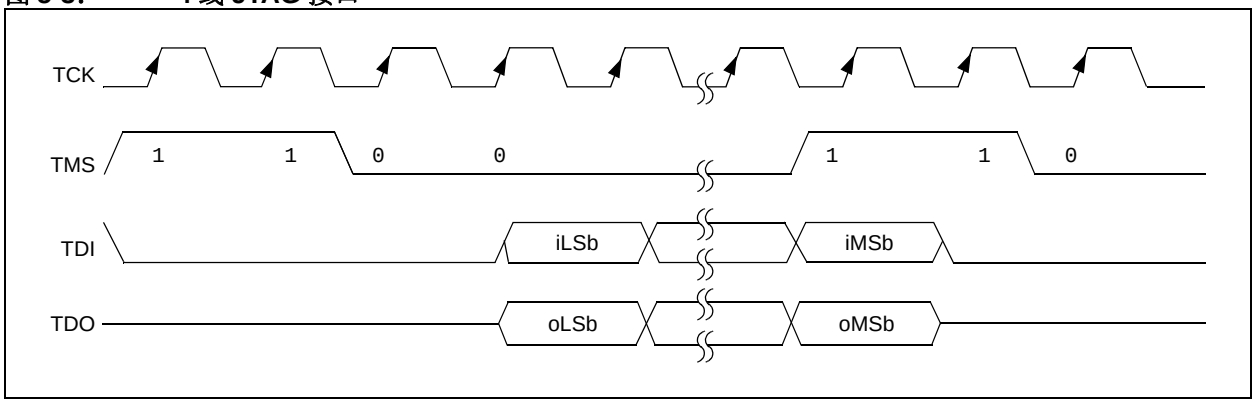
4 线接口使用标准的 JTAG （IEEE 1149.1-2001）接口信号。

- 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 —— 输入器件的数据
- TDO：测试数据输出 —— 输出器件的数据

由于只有一条数据线可用，因此必须采用串行传输协议（如 SPI）。时钟输入位于 TCK 引脚。通过 TMS 引脚逐位操控状态机来执行配置。在 TDI 和 TDO 引脚上每个 TCK 时钟脉冲输入和输出一位数据。可以装载不同的指令模式，以读取芯片 ID 或操控芯片功能。

传送到 TDI 的数据必须在 TCK 上升沿之前的芯片特定建立时间内以及 TCK 上升沿之后的保持时间内保持有效。TDO 数据在 TCK 下降沿之后的芯片特定时间内保持有效 （请参见图 5-3）。

图 5-3： 4 线 JTAG 接口



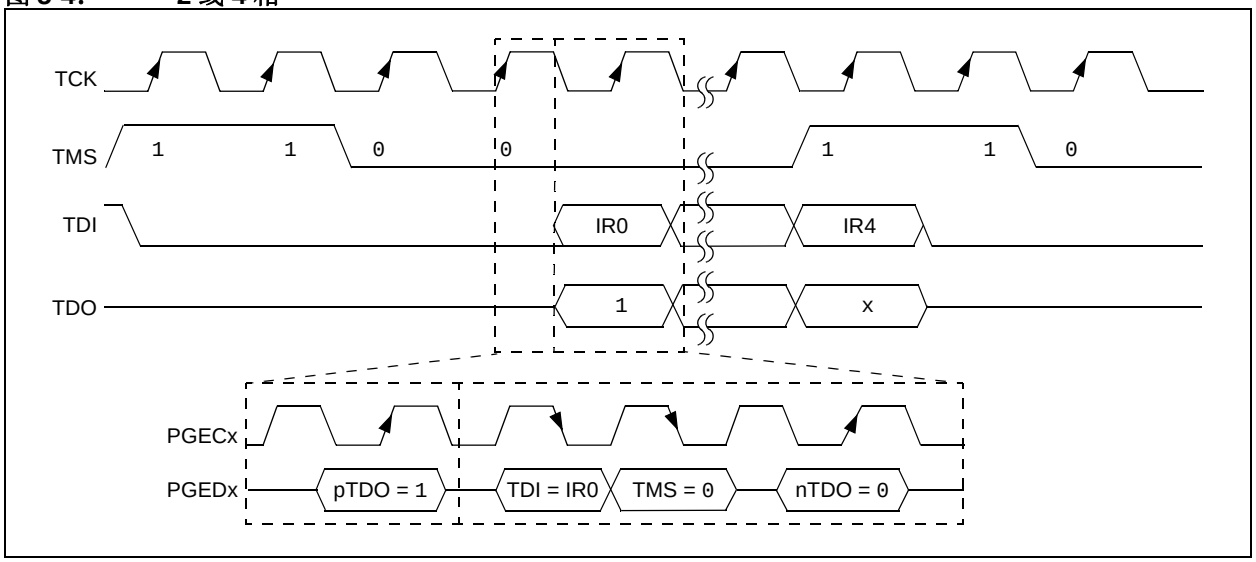
5.3 2 线 ICSP 详细信息

在 ICSP 模式下，2 线 ICSP 信号以时分复用的方式移入 2 线转 4 线模块。然后， 2 线转 4 线模块将转换信号，使其相当于 TAP 控制器的 4 线 JTAG 端口。

5.3.1 4 相 ICSP

在 4 相 ICSP 模式下，TDI、TDO 和 TMS 器件引脚在四个时钟内复用到 PGEDx（见图 5-4）。先移入最低有效位（Least Significant bit，LSb）。TDI 和 TMS 在 PGECx 的下降沿进行采样，而 TDO 则在 PGECx 的下降沿进行驱动。读取和写入数据传输都使用 4 相 ICSP 模式。

图 5-4： 2 线 4 相



6.0 伪操作

为简化编程详细信息的说明，所有操作都将使用伪操作进行说明。伪代码说明中使用了多个函数。这些函数可用于提高伪代码的可读性、抽象特定于实现的行为，或同时用于这两者。通过伪操作传递参数时，将使用以下语法：

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

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) *
  oData = **`ReadFromAddress`** (address)

6.1 `SetMode` 伪操作

格式：
`SetMode` (mode)

用途：
将 EJTAG 状态机设置为特定状态。

说明：
模式的值随时钟移入到器件的 TMS 信号引脚。TDI 设置为 `0`， TDO 会被忽略。

限制：
无。

示例：
`SetMode` (`6'b011111`)

图 6-1： `SetMode` 4 线

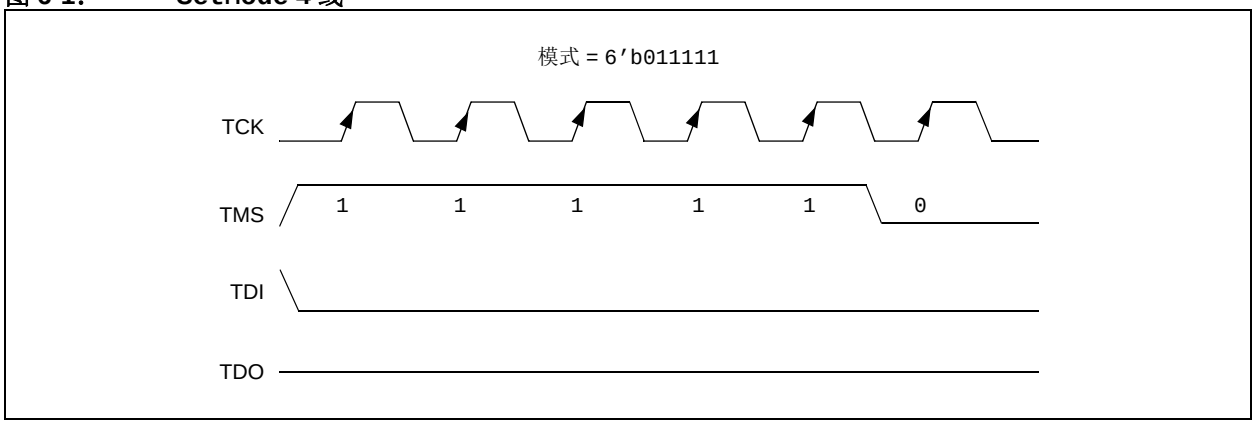
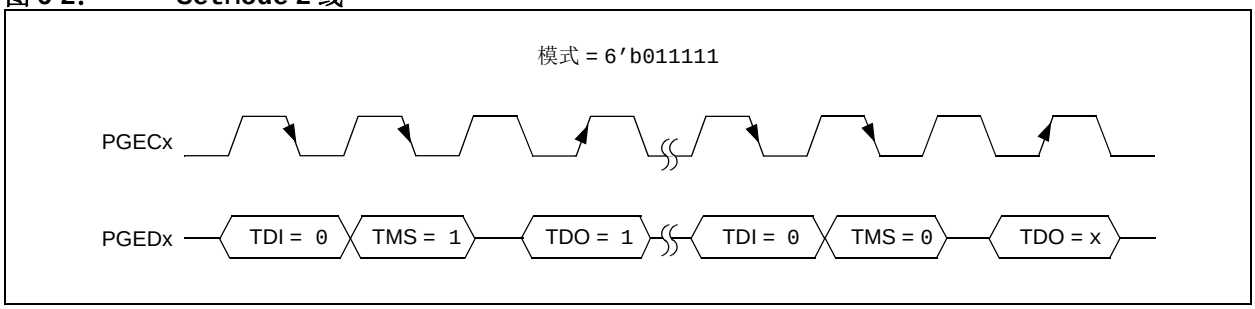


图 6-2： `SetMode` 2 线



6.2 SendCommand 伪操作

格式：
SendCommand (command)

用途：
发送命令以选择特定 TAP 寄存器。

说明 （按顺序）：

1. 将 TMS 头部移入器件，以选择移位 IR 状态。
2. 将 TMS 信号保持为低电平时，将命令移入器件的 TDI 引脚。
3. 在将 TMS 设置为高电平时，移入命令的最后一位即最高有效位 （Most Significant bit， MSb）。
4. 将 TMS 尾部移入器件的 TMS 引脚，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

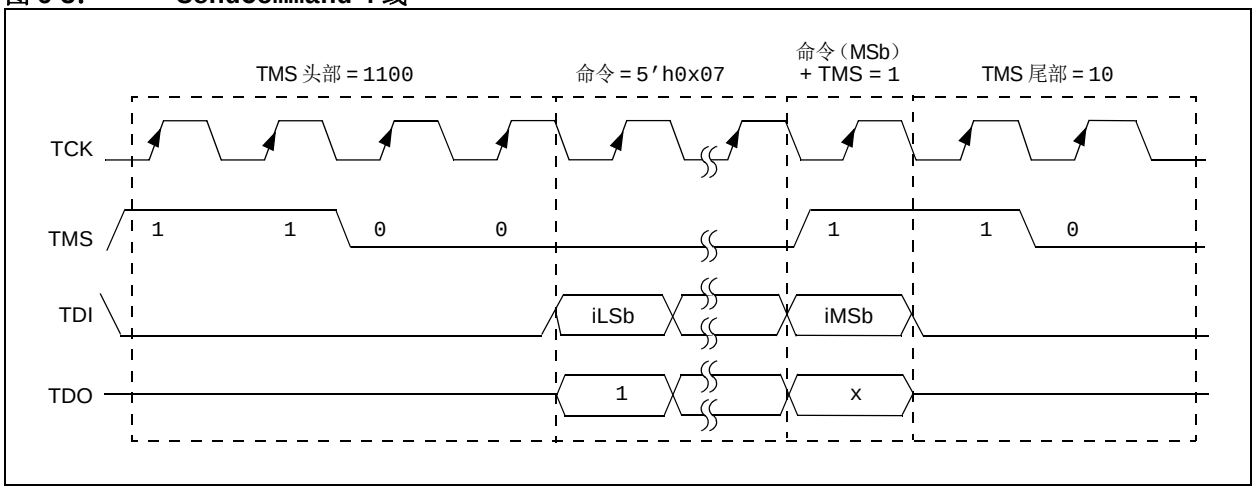
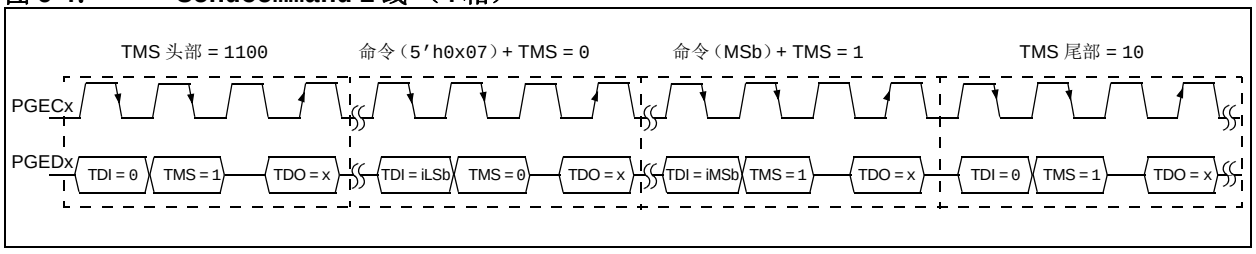


图 6-4： SendCommand 2 线 （4 相）



6.3 XferData 伪操作

格式：
oData = XferData (iData)

用途：
将数据移入和移出命令选择的寄存器。

说明 （按顺序）：

1. 将 TMS 头部移入器件，以选择移位 DR 状态。
2. 将 TMS 信号保持为低电平时，将数据移入 / 移出器件的 TDI/TDO 引脚。
3. 在将 TMS 设置为高电平时，移入 / 移出数据的最后一位即 MSb。
4. 将 TMS 尾部移入器件的 TMS 引脚，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

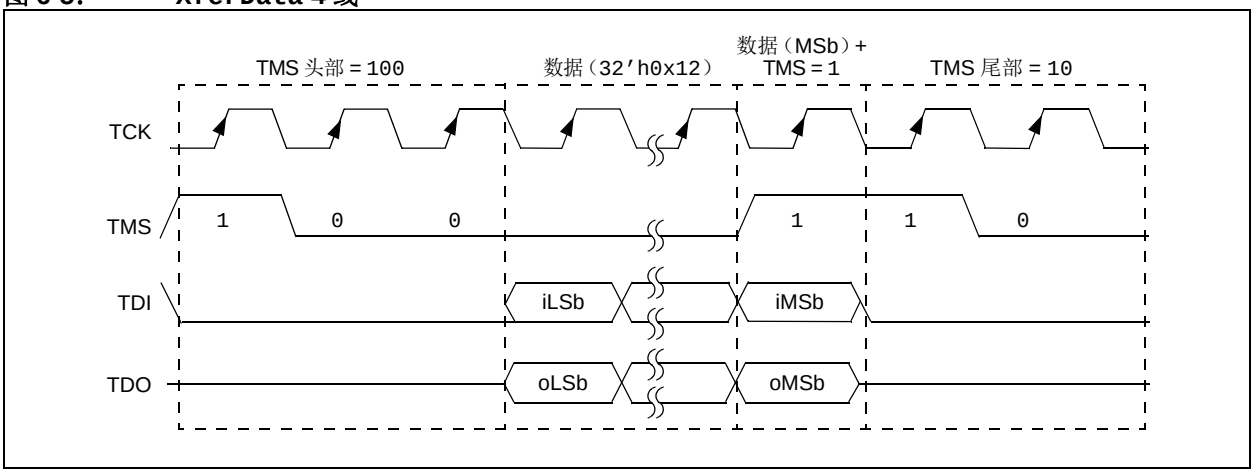
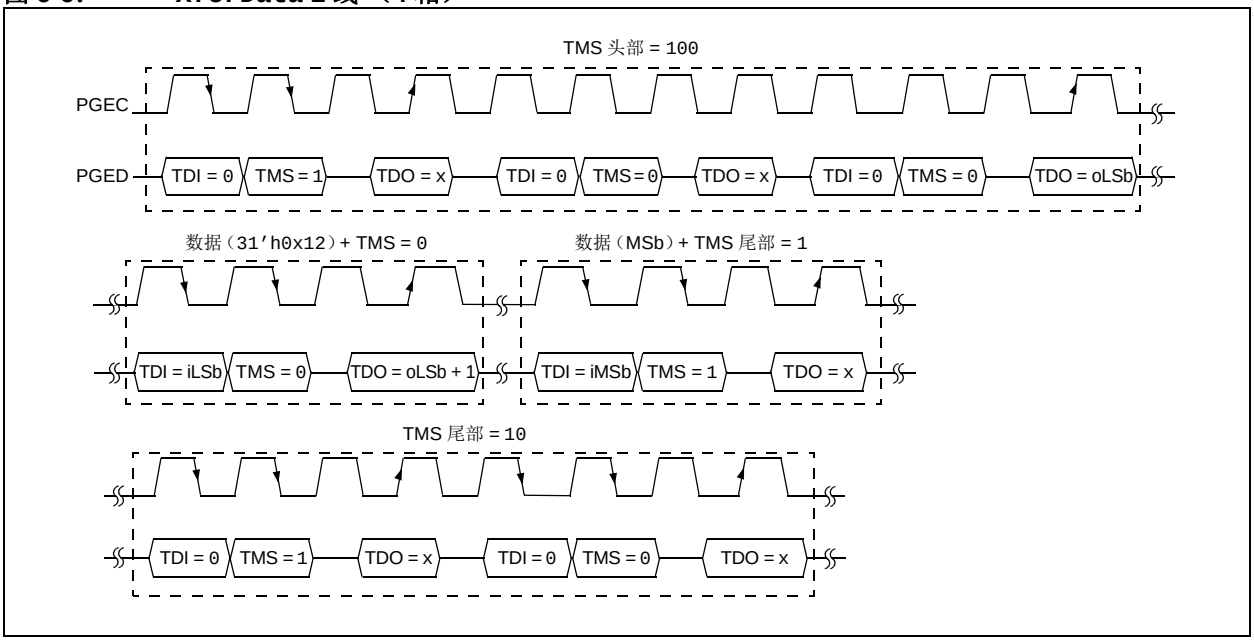


图 6-6： XferData 2 线 （4 相）



6.4 XferFastData 伪操作

格式：
oData = XferFastData (iData)

用途：
将 32 位数据快速输入 / 输出器件。

说明 （按顺序）：

1. 将 TMS 头部移入器件，以选择移位 DR 状态。

> **注：** 对于 2 线 （4 相） —— 在最后一个时钟，移入 TMS 头部的同时， oPrAcc 位从 TDO 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，即 0。

> **注：** 对于 2 线（4 相） —— 在此操作期间， TDO 将成为输出数据的LSb。移入输入数据剩余的 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)，以选择 Fastdata 寄存器，如例 6-1 中所示。有关命令的详细说明，请参见表 19-4。

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

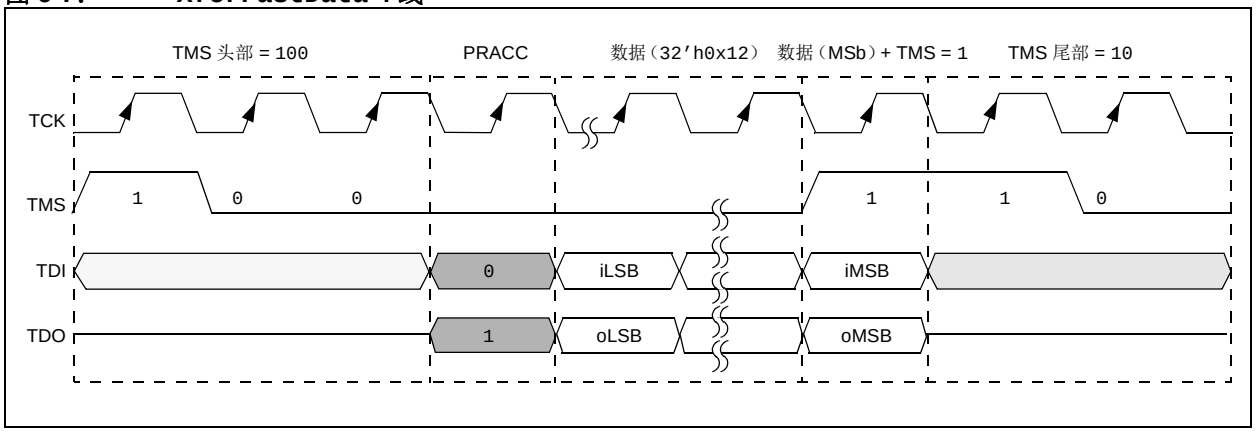
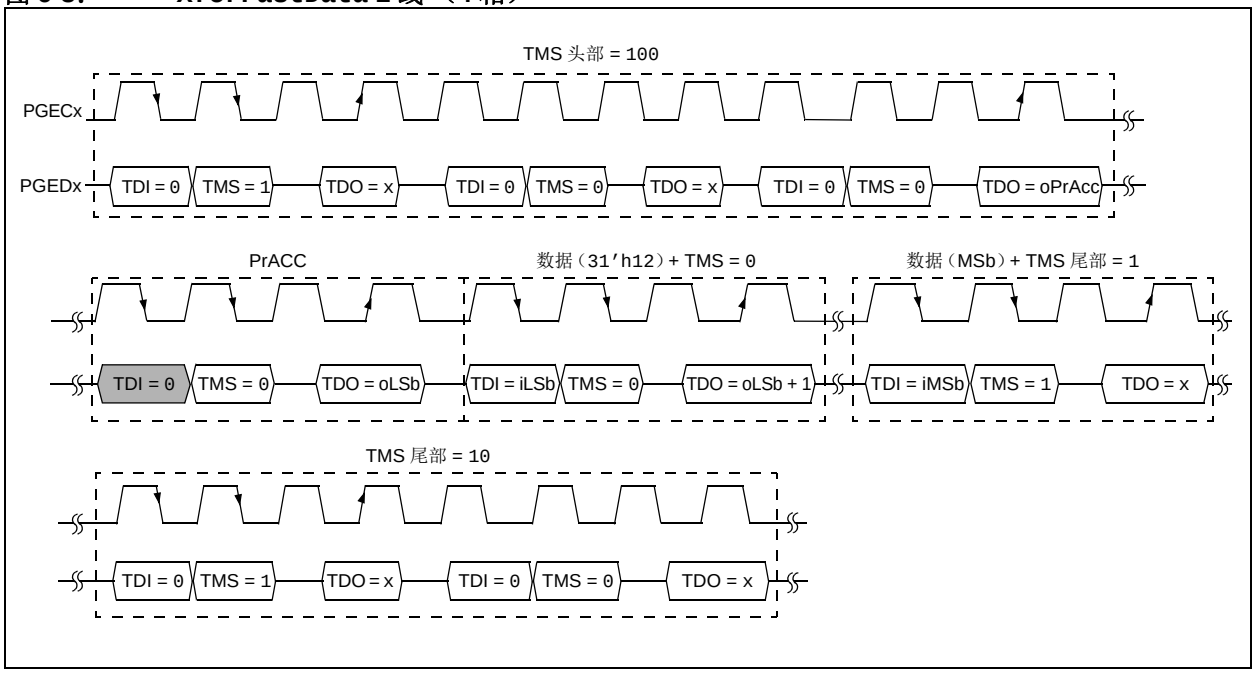


图 6-8： XferFastData 2 线 （4 相）



6.5 XferInstruction 伪操作

格式：
XferInstruction (instruction)

用途：
发送用于器件执行的 32 位数据。

说明：
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： PIC32MM 器件的 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)

用途：

从器件存储器读取 32 位数据。

说明：

从存储器中 “address”参数指定的地址读取 32 位数据。

限制：

器件必须处于调试模式。

例 6-3： ReadFromAddress

```
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 ICSP 模式

可以使用任何一对 2 线 PGEDx 和 PGECx 引脚进行编程，但是，它们必须成对使用。PGED1 必须与 PGEC1 一起使用，以此类推。

注： 如果使用的是 4 线 JTAG 接口，则不必执行以下步骤。

进入 2 线 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 位密钥序列随时钟移入到 PGEDx 引脚。
3. 然后将 MCLR 驱动为高电平并保持一段指定的时间。

更多时序详细信息，请参见**第 20.0 节“交流 / 直流特性和时序要求”**。

加到 MCLR 的编程电压为 VIH，对于 PIC32MM 器件而言，该电压实际就是 VDD。对于 VIH 没有最短保持时间要求。在移除 VIH 后，必须经过至少 P18 时间间隔才能将密钥序列移入 PGEDx 引脚。

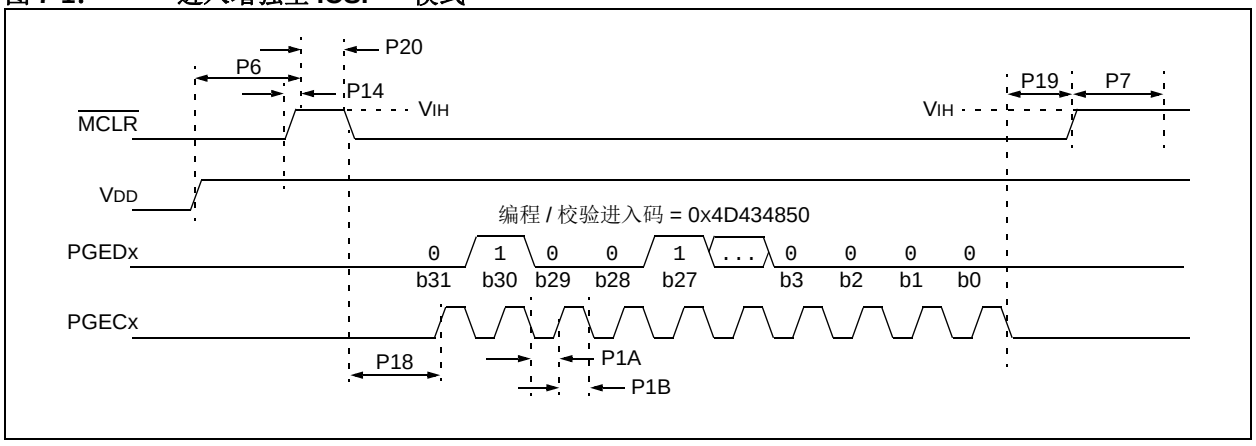
该密钥序列为一个特定的 32 位形式：0100 1101 0100 0011 0100 1000 0101 0000（首字母缩写词 “MCHP”，使用 ASCII 表示）。只有在密钥序列有效时，器件才能进入编程 / 校验模式。首先必须移入高 4 位的 MSb。

一旦密钥序列完全移入，就必须将 VIH 加到 MCLR 并在 2 线 ICSP 接口下始终保持该电压。必须经过至少 P19 和 P7 的时间间隔才能将数据移入 PGEDx 引脚。在 P7 之前出现在 PGEDx 引脚上的信号被解析为无效。

成功进入 2 线 ICSP 模式后，就可以执行后续小节中所述的编程操作。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 模式后将器件置于“复位”状态，以阻止指令执行。要解除“复位”状态，必须发出 MCHP_DE_ASSERT_RST 命令。

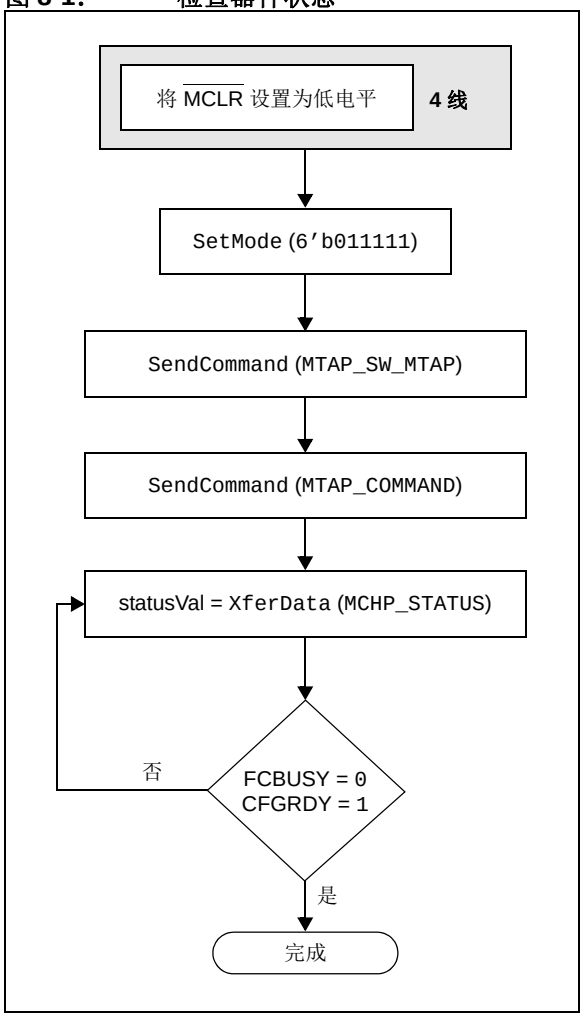
图 7-1： 进入增强型 ICSP™ 模式



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状态，以确保其准备好接收信息。

图 8-1： 检查器件状态



8.1 4 线接口

要使用 4 线 JTAG 接口检查器件状态，需要执行以下步骤。将 MCLR 置为有效保持器件在复位状态，以阻止代码执行。

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，则转到第 5 步。

注： 如果使用的是 4 线接口，则必须存在配置字所选择的振荡器源，才能访问闪存。在未编程器件中，振荡器源是内部 FRC，可通过它访问闪存。如果已将配置字重新编程为选择外部振荡器源，则必须存在该振荡器源，才能访问闪存。有关使用配置字设置选择振荡器的详细信息，请参见特定器件数据手册中的 **“特殊功能”** 章节。

8.2 2 线接口

要使用 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，则转到第 4 步。

注： 如果 CFGRDY 和 FCBUSY 未在 10 ms 内进入正确状态，则说明未正确执行序列或者器件已损坏。

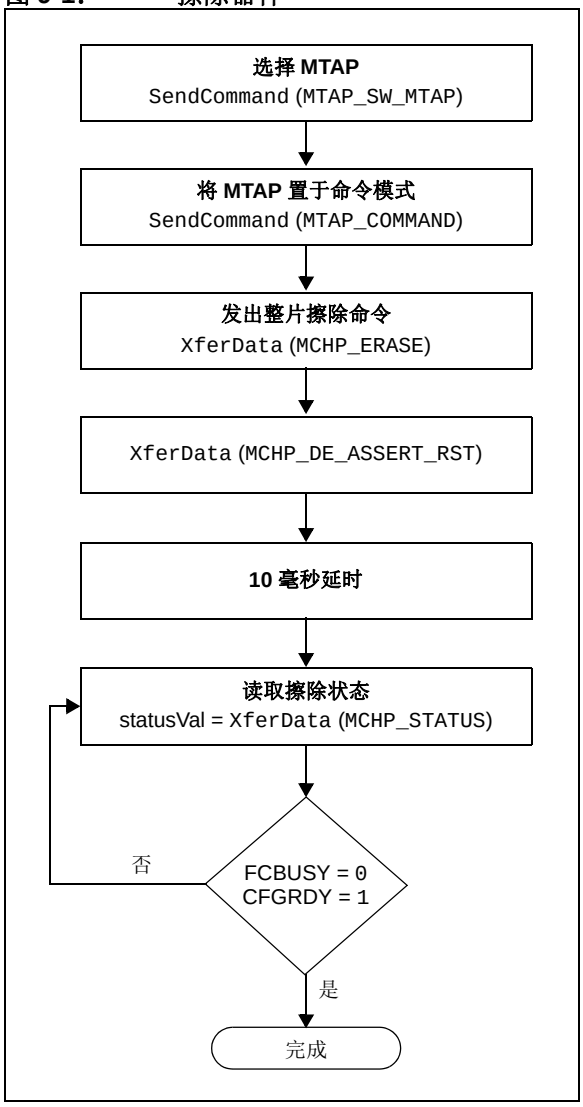
9.0 擦除器件

在对器件进行编程前，必须先将其擦除。擦除操作会向闪存中写入全 1，使其准备好编程新的一组数据。擦除器件后，可以通过执行“空白检查”操作进行校验。更多信息，请参见**第 9.1 节 “空白检查”**。

擦除闪存程序存储器（程序闪存、引导闪存和配置存储区）的步骤包括选择 MTAP 和发送 MCHP_ERASE 命令。然后，编程器必须通过读取和校验 MCHP_STATUS 值中的位，等待擦除操作完成。图 9-1 显示了执行整片擦除的过程。

注：	器件 ID 存储单元为只读，无法擦除。因此，整片擦除对这些存储单元没有影响。

图 9-1： 擦除器件



要擦除目标器件，需要完成以下步骤：

1. SendCommand (MTAP_SW_MTAP)。
2. SendCommand (MTAP_COMMAND)。
3. XferData (MCHP_ERASE)。
4. XferData (MCHP_DE_ASSERT_RST)。
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，则转到第 5 步。

注：	整片擦除操作是自定时操作。如果未在指定的整片擦除时间内正确设置 FCBUSY 和 CFGRDY 位，则说明未正确执行序列或者器件已损坏。

9.1 空白检查

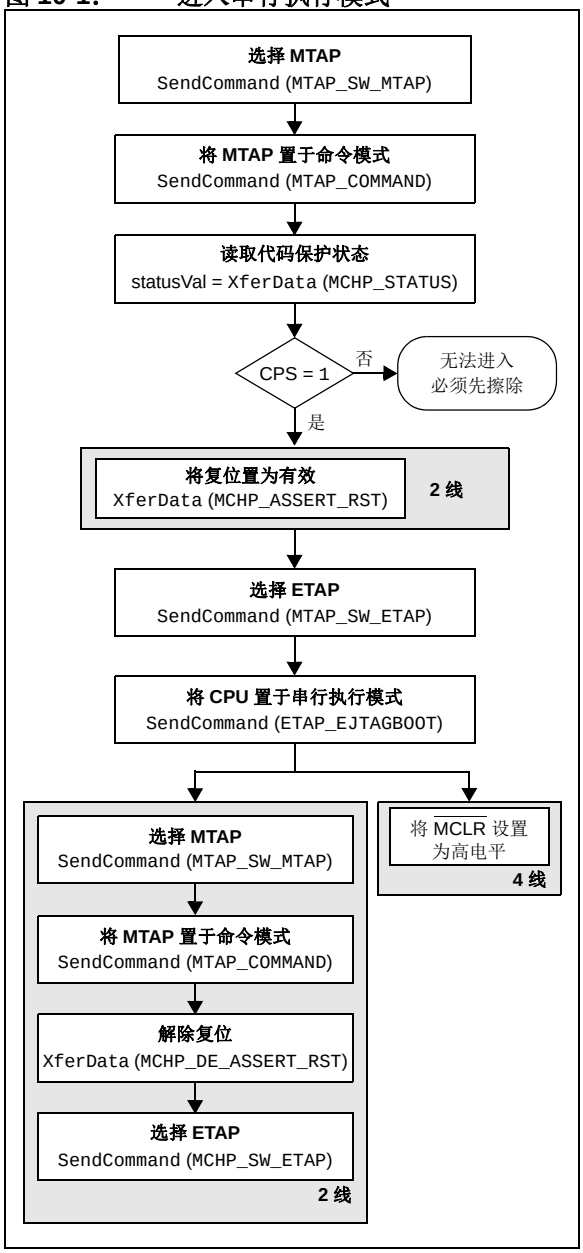
术语“空白检查”的含义是校验器件是否已被成功擦除且没有已编程的存储单元。空白或已擦除的存储单元始终读为 1。

器件配置寄存器将被空白检查忽略。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

串行执行模式可通过 JTAG 接口将指令馈送到 CPU。在对器件进行编程前，必须先将器件置于串行执行模式。如果器件处于代码保护状态，则必须在进入测试串行执行模式之前执行整片擦除。有关详细信息，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 4 线串行执行模式需要执行以下步骤：

注： 假定 $\overline{\text{MCLR}}$ 在前面的 “检查器件状态” 步骤（见图 8-1）中已驱动为低电平。

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 $\overline{\text{MCLR}}$ 设置为高电平。

10.2 2 线接口

进入 2 线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. SendCommand (MTAP_SW_ETAP)。

11.0 下载编程执行程序 （PE）

PE 使用一个简单的命令集和通信协议为编程器提供了对 PIC32MM 器件进行编程和校验的机制。 PE 驻留在 RAM 存储器中，由 CPU 执行以对器件编程。PE 可提供以下几项基本功能：

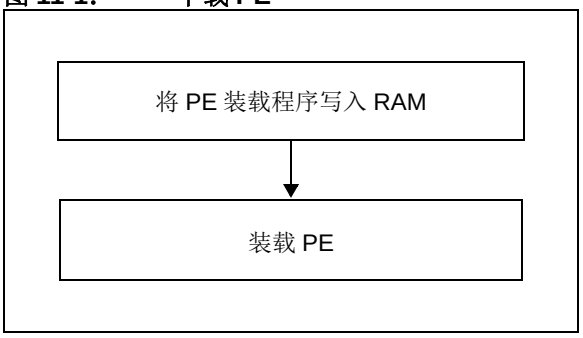
- 读存储器
- 擦除存储器
- 对存储器编程
- 空白检查
- 读取执行程序固件版本
- 获取闪存存储单元的循环冗余校验 （Cyclic Redundancy Check， CRC）

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。**第 16.2 节 “PE 命令集”** 提供了每条命令的详细说明。

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

将 PE 装入数据 RAM 中之后，可以使用表 16-2 中所示的命令集对 PIC32MM 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 装载程序装入数据 RAM。（PE 装载程序将 PE 二进制文件装入数据 RAM 的适当位置，在完成后跳转到编程执行程序，并开始执行它。）
2. 将 PE 二进制代码馈送到 PE 装载程序。

表 11-1 列出了下载 PE 所需的步骤。

表 11-1： 下载 PE

操作	操作数
第 1 步： 设置 PE 的 PIC32MMRAM 地址。 PIC32MM 内核执行的指令序列为： lui a0, 0xa000 ori a0, a0, 0x800	
XferInstruction	0xA00041A4
XferInstruction	0x08005084

第 2 步： 装载 PE_loader。重复此步骤 （第 2 步），直到整个 PE_loader 都装入 PIC32MM 存储器中。在操作数字段中，“<PE_loader hi++>”表示表 11-2 中所示的 PE_loader 操作码的 MSb 31 到 16。同样，“<PE_loader lo++>”表示表 11-2 中所示的 PE_loader 操作码的 LSb 15 到 0。“++”符号表示当连续执行这些操作时，将从表 11-2 中所示 PE_loader 的操作码列表中传输新字。 PIC32MM 内核执行的指令序列为：
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
第 3 步： 跳转到 PE_loader。 PIC32MM 内核执行的指令序列为： lui t9, 0xa000 ori t9, t9,0x800 jr t9 nop	
XferInstruction	0xA00041B9
XferInstruction	0x08005339
XferInstruction	0x0C004599
第 4 步： 使用 PE_loader 装载 PE。重复此步骤 （第 4 步）的最后一个指令，直到整个 PE 都装入 PIC32MM 存储器中。在此步骤中，将提供一个 Intel® Hex 格式的 PE 文件，您每次需要解析多个 32 位字并将其传输到 PIC32MM 存储器中 （参见**附录 B：“Hex 文件格式”**）。 PIC32MM 执行的指令序列如表 11-2 中所示。	
SendCommand	ETAP_FASTDATA
XferFastData	PE_ADDRESS （来自 PE Hex 文件的 PE 程序块的地址）
XferFastData	PE_SIZE （来自 PE Hex 文件的程序块的 32 位字数）
XferFastData	来自 PE Hex 文件的 PE 软件操作码 （PE 指令）
第 5 步： 跳转到 PE。幻数 （0xDEAD0000）指示 PE_loader PE 已完全装入存储器。当 PE_loader 识别幻数时，将跳转到 PE。	
XferFastData	0x00000000
XferFastData	0xDEAD0000

表 11-2： PE 装载程序操作码

操作码	指令
0xDEAD41A7	lui a3, 0xdead
0xFF2041A6	lui a2, 0xff20
0xFF2041A5	lui a1, 0xff20
	here1
0x69E06A60	lw a0, 0(a2)
	lw v1, 0(a2)
0x000C94E3	beq v1, a3, <here3>
0x8DFA0C00	nop
	beqz v1, <here1>
	here2
	lw v0, 0(a1)
0xE9406DBE	addiu v1, v1,-1
	sw v0, 0(a0)
0xADFB6E42	addiu a0, a0, 4
	bnez v1, <here2>
0xCFF20C00	nop
	b <here1>
0x0C000C00	nop; nop
	here3
0xA00041A2	lui v0, 0xa000
0x09015042	ori v0, v0, 0x901
0x0C004582	jr v0
	nop

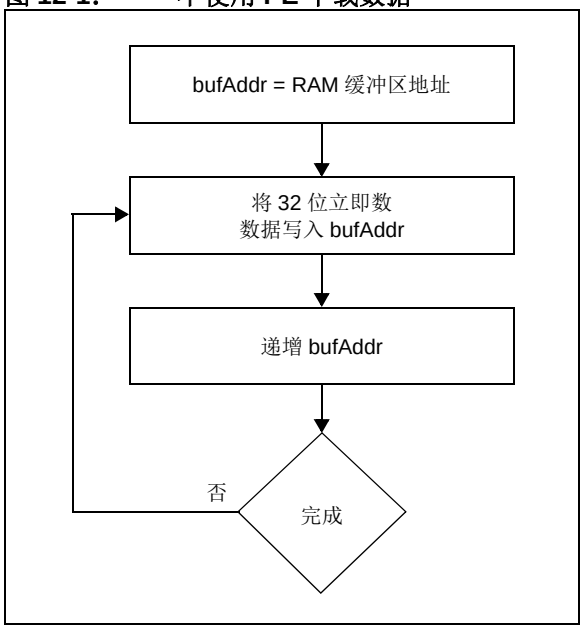
12.0 下载数据块

要将数据块编程到 PIC32MM 器件中，必须将其装入 SRAM 中。

12.1 不使用 PE

要在不使用 PE 的情况下对存储块进行编程，必须先将数据块写入 RAM。此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，以将数据块写入器件的内部 RAM 存储器。

图 12-1： 不使用 PE 下载数据



下载数据块需要执行以下步骤：

1. `XferInstruction` (opcode)。
2. 重复第 1 步，直到最后一条指令传输到 CPU 中。

表 12-1： 下载数据操作码

操作码	指令
第 1 步：将 SRAM 基地址初始化为 0xA0000000。	
0xA00041A4	lui a0, 0xA000
第 2 步：将要编程的整行数据写入系统 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

第 3 步：重复第 2 步，直到装入一行数据。	

12.2 使用 PE

使用 PE 时，将通过两条命令处理**第 12.0 节 “下载数据块”**和**第 13.0 节 “启动闪存行写入”**中的步骤：ROW_PROGRAM 和 PROGRAM。

ROW_PROGRAM 命令可对单行闪存数据进行编程，PROGRAM 命令则可以对多行闪存数据进行编程。这两条命令在**第 16.0 节 “编程执行程序”**中有详细说明。

13.0 启动闪存行写入

一旦将一行数据下载到器件的 SRAM 中，就必须启动编程序列来将数据块写入闪存中。

有关启动闪存行写入的操作码和指令，请参见表 13-1。

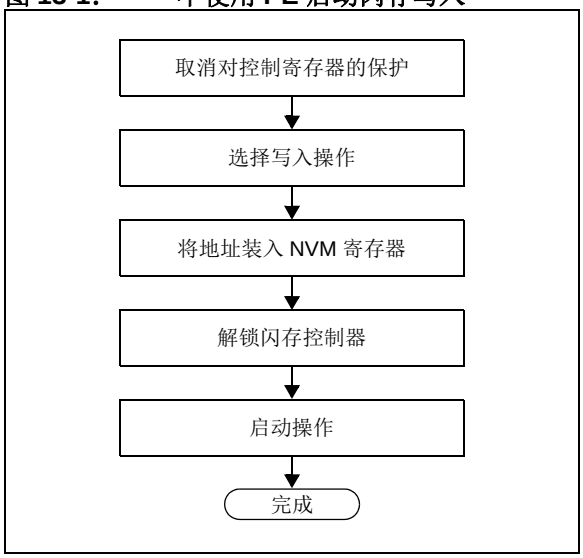
13.1 使用 PE

使用 PE 时，数据会立即从 SRAM 写入闪存中。不需要进一步的操作。

13.2 不使用 PE

对闪存执行的写入操作是通过 NVMCON 寄存器控制的。编程的执行过程如下：设置 NVMCON 选择写入操作的类型，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 启动闪存写入



在闪存写入步骤（见表 13-1）中，使用了行编程方法来对闪存进行编程，因为这通常是最便捷的方法。此外，还提供了双字编程方法，根据您的应用，可能需要使用此方法。有关更多信息，请参见特定器件数据手册中的**“闪存程序存储器”**章节和《PIC32 系列参考手册》的相关章节。

启动闪存写入需要执行以下步骤：

1. `XferInstruction` (opcode)。
2. 重复第 1 步，直到最后一条指令传输到 CPU 中。

表 13-1：　　启动闪存行写入操作码

操作	操作数

第 1 步：初始化常量。寄存器 a1 和 a2 设置为 WREN = 1 或 NVMOP<3:0> = 0011，且 WR = 1。寄存器 s1 和 s2 设置为解锁数据值，且 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

第 2 步：将寄存器 a0 设置为 NVM 控制器的基地址（0xBF80_2380）。寄存器 s3 设置为用于禁止 NVMBPB 中写保护的值。
`lui a0, 0xbf80`
`ori a0, a0, 0x2380`
`ori s3, $0, 0x8000`
`/* BWPAUNLK bit mask */`	

XferInstruction	0xBF8041A4
XferInstruction	0x23805084
XferInstruction	0x80005260

第 3 步：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

表 13-1：　　启动闪存行写入操作码 （续）

操作	操作数

第 4 步：使用要编程的闪存行的地址设置 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

第 5 步：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

第 6 步：设置 NVMCON 寄存器进行写保护。
`sw a1, 0(a0)`
`/* NVMCON = 0x4003 */`
`delay (6 µs)`	

XferInstruction	0x0C00EAC0

第 7 步：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
第 8 步：读取 NVMCON 寄存器，直到 WR 位（NVMCON<15>）清零。	
ReadFromAddress	0xBF802380
第 9 步：在 WR 位 （NVMCON<15>）中发现 0 后至少等待 500 ns，然后再写入任意 NVM 寄存器。这需要在执行过程中插入 NOP 指令。 `nop` `nop` `nop` `nop`	
XferInstruction	0x0C000C00
XferInstruction	0x0C000C00

第 10 步：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

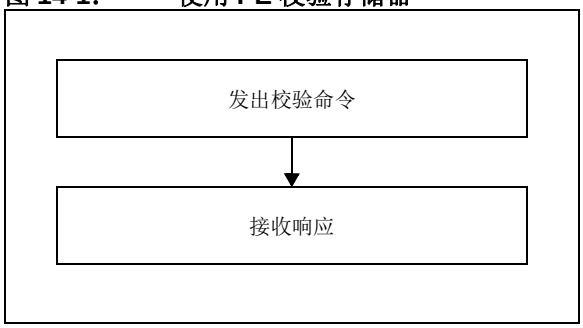
校验的步骤涉及读回代码存储空间并将读到的值与存储在编程器缓冲区中的副本作比较。使用其余代码校验配置寄存器。

注：	由于配置寄存器包含器件代码保护位，因此应在写入后立即对代码存储区进行校验（如果使能代码保护）。这是因为如果代码保护位清零后发生器件复位，将无法读取或校验器件。

14.1 使用 PE 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 校验存储器



使用 PE 校验存储器需要执行以下步骤：

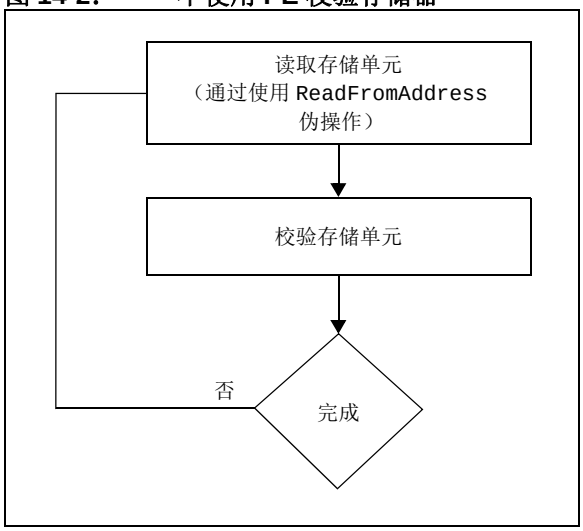
1. XferFastData (GET_CRC)。
2. XferFastData (start_Address)。
3. XferFastData (length)。
4. valCkSum = XferFastData (32'h0x00)。

校验 valCkSum 是否与编程器缓冲区中保存的副本的校验和相匹配。

14.2 不使用 PE 校验存储器

读取闪存是通过从Fastdata寄存器执行一系列读访问来执行的。表 19-4 显示了 EJTAG 编程详细信息，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 校验存储器



校验存储器需要执行以下步骤：

1. XferInstruction (opcode)。
2. 重复第 1 步，直到最后一条指令传输到 CPU 中。
3. 校验 valRead 是否与编程器缓冲区中保存的副本相匹配。
4. 对每个存储单元重复第 1-3 步。

表 14-1： 校验器件操作码

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

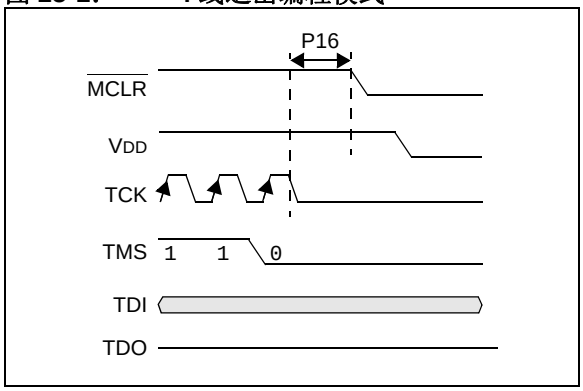
15.0 退出编程模式

对器件进行编程后，必须使器件退出编程模式，以开始正确执行其新的程序存储器内容。

15.1 4 线接口

通过将 VIH 从 MCLR 移除，可退出编程模式，如图 15-1 所示。退出操作的唯一要求是在最后一个时钟和编程信号之后的 P16 时间间隔后移除 VIH。

图 15-1： 4 线退出编程模式



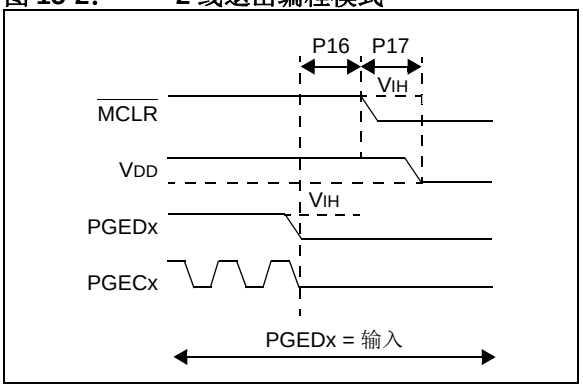
退出编程模式需要执行以下步骤：

1. SetMode (5'b11111)。
2. 将 MCLR 置为有效。
3. 断开电源 （如果器件已通电）。

15.2 2 线接口

通过将 VIH 从 MCLR 移除，可退出编程模式，如图 15-2 所示。退出操作的唯一要求是在 PGECx 和 PGEDx 引脚上最后一个时钟和编程信号之后的 P16 时间间隔后移除 VIH。

图 15-2： 2 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. 在 PGECx 上发出时钟脉冲。
4. 断开电源 （如果器件已通电）。

16.0 编程执行程序

注： 编程执行程序（PE）随附于您安装的 MPLAB® X IDE 中。要下载器件对应的 PE 文件，请访问 Microchip 网站（www.microchip.com）上的相关产品页面。

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设备，而 PE 是从设备。

所有通信都是由编程器以命令形式发起的。 PE 一次只能接收一个命令。相应地，在接收并处理命令后，PE 将向编程器发送一个响应。

16.1.1 2 线 ICSP EJTAG 速率

在 ICSP 模式下，PIC32MM 系列器件使用内部快速 RC（FRC）振荡器工作，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。尤其是，编程器使用 Fastdata 寄存器将命令和数据传输到 PE。编程器使用地址和数据寄存器接收来自 PE 的响应。接收响应的伪操作如下面的 `GetPEResponse` 伪操作所示：

格式：

```
response = GetPEResponse()
```

用途：

使编程器从 PE 接收 32 位响应值。

例 16-1： GetPEResponse 示例

```
WORD    GetPEResponse()
{
        WORD response;

// Wait until CPU is ready
SendCommand(ETAP_CONTROL);

// Check if Proc.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;
}
```

编程器和 PE 之间的典型通信序列如表 16-1 中所示。

序列在编程器将命令和可选附加数据发送到 PE 时开始，然后 PE 会执行命令。

PE 完成命令执行时，会将响应发送回编程器。

响应可以有多个。例如，如果编程器发送 READ 命令，响应将包含读取的数据。

表 16-1： PE 的通信序列

操作	操作数
第 1 步：将命令和可选数据从编程器发送到 PE。	
XferFastData	(Command | data len)
XferFastData..	optional data..
第 2 步：编程器读取来自 PE 的响应。	
GetPEResponse	response
GetPEResponse...	response...

16.2 PE 命令集

表 16-2 提供了 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”**到**第 16.2.14 节 “DOUBLE_WORD_PROGRAM 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，其中包含所有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 位头和命令所需的所有数据组成的通用格式（见图 16-1）。32 位头由用于标识命令的 16 位操作码字段以及 16 位命令操作数字段组成。操作数字段的使用因命令而异。

注： 有些命令没有操作数信息；但是，必须发送操作数字段，且编程执行程序将忽略该数据。

图 16-1： 命令格式

操作码字段中的命令必须与表16-2中所列命令集中的一个命令相匹配。如果接收到的命令与列表中的任何命令都不匹配，将返回 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

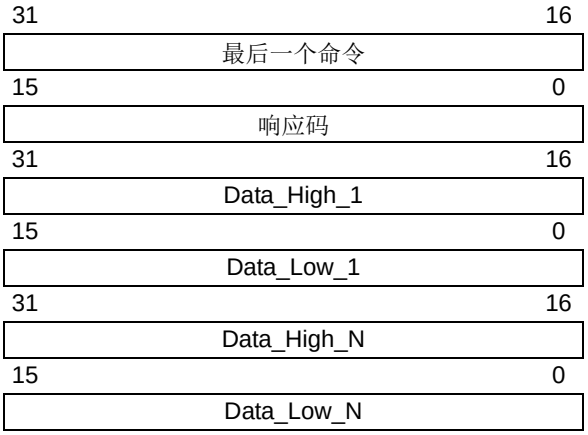
操作码	助记符	说明
0x0	ROW_PROGRAM(1)	对指定地址处的一行闪存进行编程。
0x1	READ	从指定地址开始读取存储器的 N 个 32 位字 （N < 65,536）。
0x2	PROGRAM	从指定地址开始对闪存进行编程。
0x4	CHIP_ERASE	对整个芯片进行芯片擦除。
0x5	PAGE_ERASE	从指定地址擦除代码存储区页。
0x6	BLANK_CHECK	代码空白检查。
0x7	EXEC_VERSION	读取 PE 软件版本。
0x8	GET_CRC	获取闪存的 CRC。
0x9	PROGRAM_CLUSTER	将指定的字节数编程到指定地址。
0xA	GET_DEVICEID	返回器件的硬件 ID。
0xC	GET_CHECKSUM	获取闪存的校验和。
0xE	DOUBLE_WORD_PGRM	对指定地址处的闪存的两个字进行编程。

注 1： 关于每个器件的行大小，请参见表 5-1。

16.2.2 响应格式

PE 响应集如表 16-3 中所示。所有 PE 响应都具有由 32 位头和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一个 16 位字段，该字段指示 PE 处理的命令。它可以用来校验 PE 是否正确地接收了编程器发送的命令。

16.2.2.2 响应码

响应码指示最后一个命令是成功还是失败，或者命令是否为无法识别的值。响应码的值如表 16-3 中所示。

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	命令处理失败
0x3	NACK	命令未知

16.2.2.3 可选数据

对于某些命令（如读取），响应头后面可能会跟随可选数据。可选数据的 32 位字数量取决于最后一个命令操作及其参数。

16.2.3 ROW_PROGRAM 命令

ROW_PROGRAM命令可指示PE对指定地址处的一行数据进行编程。

要编程到存储器中的数据位于命令字 Data_1 至 Data_N 中，必须使用表 16-4 给出的指令字打包格式对它们进行打包（此命令需要整行数据）。

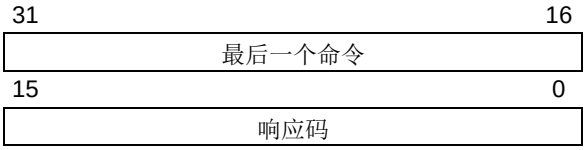
图 16-3： ROW_PROGRAM 命令

表 16-4： ROW_PROGRAM 格式

字段	说明
操作码	0x0
操作数	不使用
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	数据字 1 的高 16 位
Data_Low_1	数据字 1 的低 16 位
Data_High_N	数据字 2 至 N 的高 16 位
Data_Low_N	数据字 2 至 N 的低 16 位

预期的响应（1 字）：

图 16-4： ROW_PROGRAM 响应



16.2.4 READ 命令

READ命令指示PE从Addr_Low和Addr_High字段指定的32 位地址开始，从存储器读取在操作数字段中指定的 N 个 32 位字。此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都使用表16-5中所述的数据打包格式。

图 16-5： READ 命令

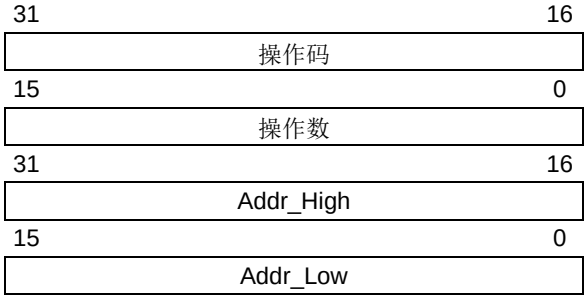


表 16-5： READ 格式

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： 读取未实现的存储区将导致 PE 复位。确保只访问特定器件上存在的存储单元。

16.2.5 PROGRAM 命令

PROGRAM命令指示PE从Addr_Low和Addr_High字段指定的 32 位地址开始对闪存（包括配置字）进行编程。一个 32 位长度字段指定要编程的字节数。

该地址必须与闪存行大小边界对齐，且长度必须是闪存行大小的倍数。有关器件行大小，请参见表 5-1。

图 16-7： PROGRAM 命令

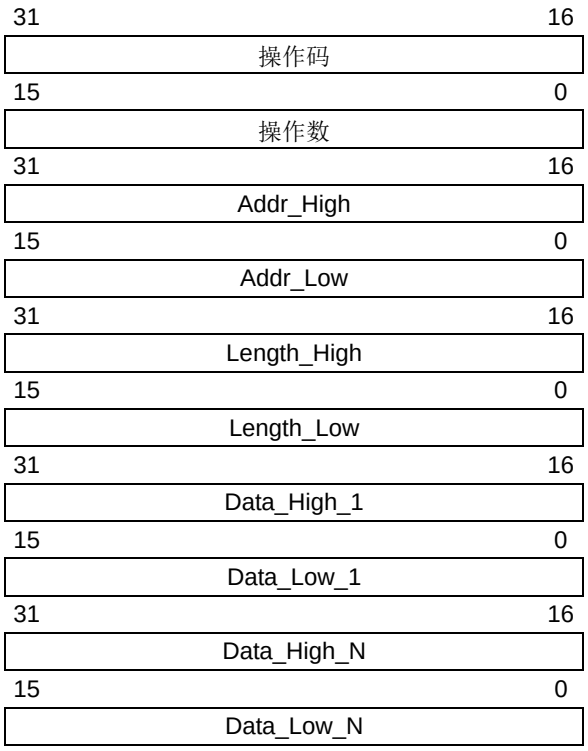


表 16-6： PROGRAM 格式

字段	说明
操作码	0x2
操作数	不使用
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	数据字 2 至 N 的低 16 位
Data_High_N	数据字 2 至 N 的高 16 位

以下是三个编程情形：

- 要编程的数据长度等于单个闪存行的大小
- 要编程的数据长度等于两个闪存行的大小
- 要编程的数据长度大于两个闪存行的大小

当数据长度等于以字节为单位的行大小时， PE 将从探针接收数据块，并立即将此命令的响应发送回探针。

PE 将对其接收的每行数据做出响应。如果命令的数据长度等于单行，将生成单个 PE 响应。如果数据长度等于两行， PE 将等待，以接收这两行数据，然后为每个数据行发送连续响应。如果数据长度大于两行， PE 将在接收前两行数据后发送第一行的响应。接收后续数据行包后将发送后续响应。响应将比数据滞后一行。接收最后一行数据时， PE 将发出倒数第二个数据行的连续响应，后面跟随最后一行的响应。

如果 PE 在任何块的编程中遇到错误，它将向探针发送故障状态，并中止 PROGRAM 命令。接收到故障状态时，探针必须停止发送数据。 PE 将不会处理此命令来自探针的任何其他数据。该过程如图 16-9 中所示。

注：	如果 PROGRAM 命令失败，则编程器应使用 READ 命令从闪存中读取失败的行。然后，编程器应将从闪存接收的行与其本地副本逐字比较，以确定闪存编程失败的地址。

此命令的响应与其他命令的响应稍有不同。响应的16个 MSb 包含目标地址的 16 个 LSb，其中最后一个块已编程。这有助于探针和 PE 在发送和接收数据及响应时保持正确同步。

预期的响应 （1 字）：

图 16-8： PROGRAM 响应

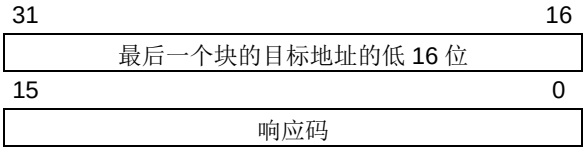
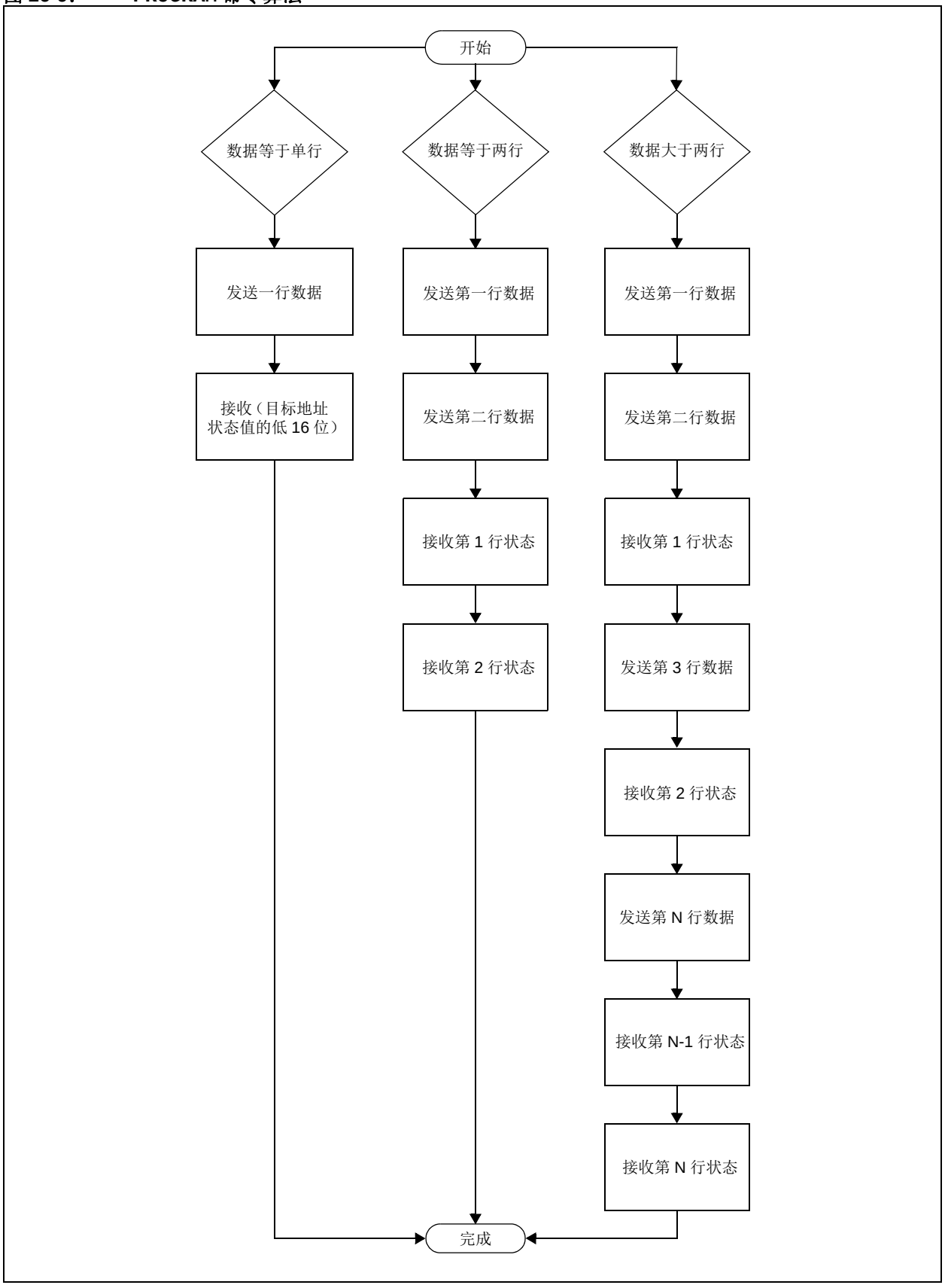


图 16-9： PROGRAM 命令算法



16.2.6 CHIP_ERASE 命令

CHIP_ERASE 命令擦除整个芯片，包括配置块。

在执行擦除后，整个闪存将包含 0xFFFFFFFF。

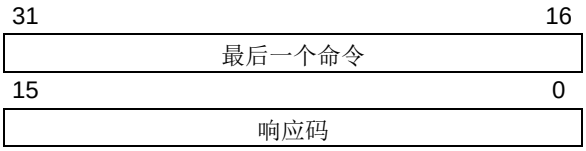
图 16-10： CHIP_ERASE 命令

表 16-7： CHIP_ERASE 格式

字段	说明
操作码	0x4
操作数	不使用
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 （1 字）：

图 16-11： CHIP_ERASE 响应



16.2.7 PAGE_ERASE 命令

PAGE_ERASE命令从指定基地址擦除代码存储区的指定页数。指定的基地址必须为 0x400 或 0x100 的倍数，具体取决于器件。

在执行擦除后，代码存储区的所有目标字都将包含 0xFFFFFFFF。

图 16-12： PAGE_ERASE 命令

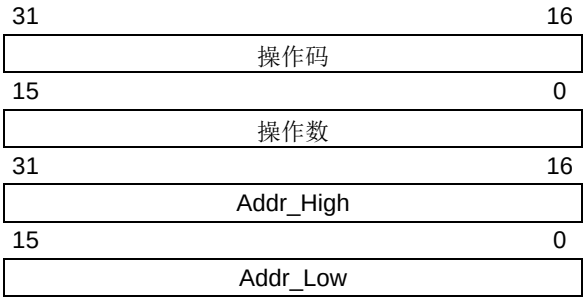
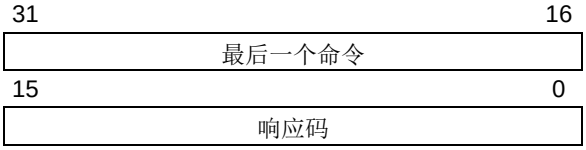


表 16-8： PAGE_ERASE 格式

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 （1 字）：

图 16-13： PAGE_ERASE 响应



16.2.8 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 16-14： BLANK_CHECK 命令

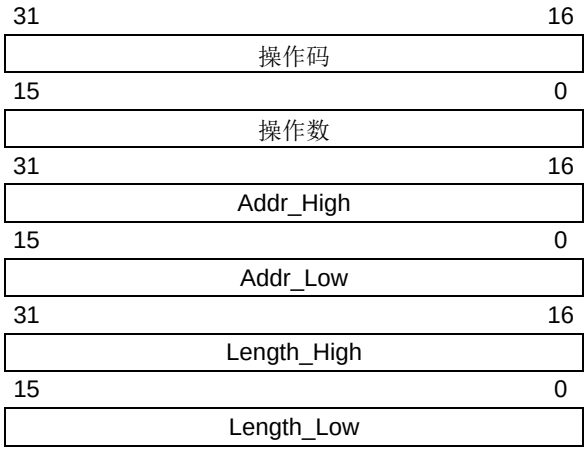
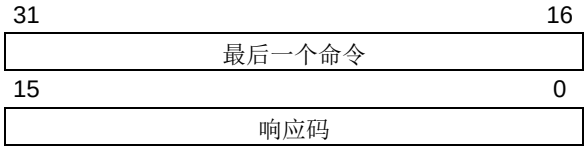


表 16-9： BLANK_CHECK 格式

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

预期的响应 （空白器件为 1 字）：

图 16-15： BLANK_CHECK 响应



16.2.9 EXEC_VERSION 命令

EXEC_VERSION 查询 RAM 中存储的 PE 软件的版本。

图 16-16： EXEC_VERSION 命令

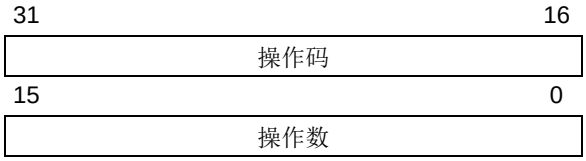
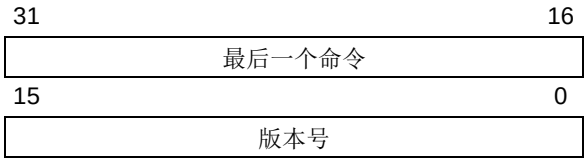


表 16-10： EXEC_VERSION 格式

字段	说明
操作码	0x7
操作数	不使用

预期的响应 （1 字）：

图 16-17： EXEC_VERSION 响应



16.2.10 GET_CRC 命令

GET_CRC 使用查表方法计算从指定地址开始指定长度的缓冲区的 CRC。 CRC 详细信息如下：

- CRC-CCITT， 16 位
- 多项式：X^16+X^12+X^5+1， Hex 0x00011021
- 种子：0xFFFF
- 高字节 （Most Significant Byte， MSB）首先移入

注： 在响应中，只有 CRC 的最低 16 位是有效的。

图 16-18： GET_CRC 命令

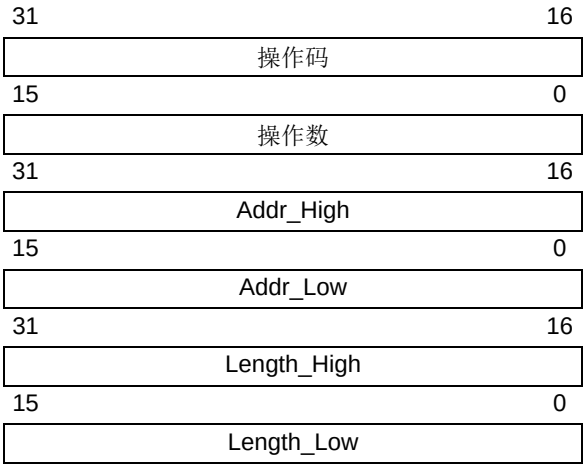


表 16-11： GET_CRC 格式

字段	说明
操作码	0x8
操作数	不使用
地址	计算 CRC 的起始地址
长度	进行 CRC 计算的缓冲区的长度（以字节数为单位）

预期的响应 （2 字）：

图 16-19： GET_CRC 响应

16.2.11 PROGRAM_CLUSTER 命令

PROGRAM_CLUSTER 将指定的字节数编程到指定地址。该地址必须按 64 位对齐，且字节数必须是 64 位字的倍数。

图 16-20： PROGRAM_CLUSTER 命令

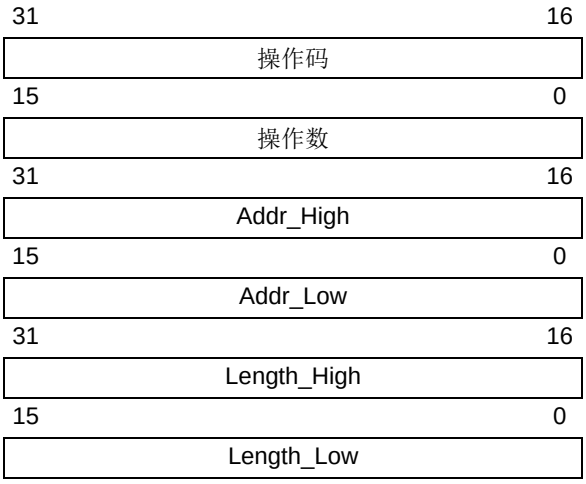


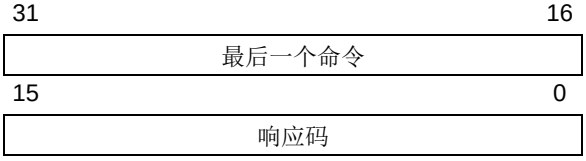
表 16-12： PROGRAM_CLUSTER 格式

字段	说明
操作码	0x9
操作数	不使用
地址	编程的起始地址
长度	要编程的存储区长度 （以字节数为单位）

注： 如果 PROGRAM_CLUSTER 命令失败，则编程器应使用 READ 命令从闪存中读取失败的行。然后，编程器应将从闪存接收的行与其本地副本逐字比较，以确定闪存编程失败的地址。

预期的响应 （1 字）：

图 16-21： PROGRAM_CLUSTER 响应



16.2.12 GET_DEVICEID 命令

GET_DEVICEID 命令返回器件的硬件 ID。

图 16-22： GET_DEVICEID 命令

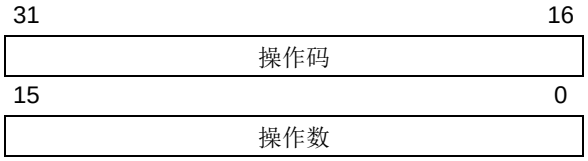


表 16-13： GET_DEVICEID 格式

字段	说明
操作码	0xA
操作数	不使用

预期的响应 （1 字）：

图 16-23： GET_DEVICEID 响应

16.2.13 GET_CHECKSUM 命令

GET_CHECKSUM 返回所有字节的总和，从地址参数开始一直到长度参数。结果为一个 32 位字。

图 16-24： GET_CHECKSUM 命令

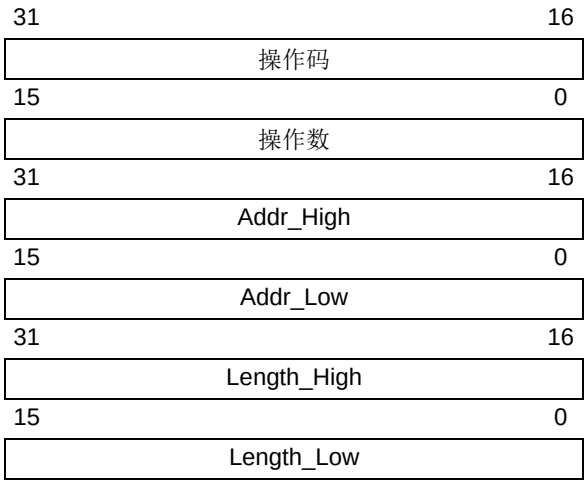


表 16-14： GET_CHECKSUM 格式

字段	说明
操作码	0x0C
操作数	不使用
Addr_High	用于计算校验和的数据的 32 位起始地址的高 16 位
Addr_Low	用于计算校验和的数据的 32 位起始地址的低 16 位
Length_High	用于计算校验和的数据的 32 位长度的高 16 位 （以字节为单位）
Length_Low	用于计算校验和的数据的 32 位长度的低 16 位 （以字节为单位）

预期的响应 （1 字）：

图 16-25： GET_CHECKSUM 响应

16.2.14 DOUBLE_WORD_PROGRAM 命令

DOUBLE_WORD_PROGRAM指示PE对指定地址处的两个32 位字进行编程。该地址必须是一个对齐的两字边界（bit 0 必须为 0）。如果不是，该命令将返回 FAIL 响应值，且不对任何数据进行编程。

图 16-26： DOUBLE_WORD_PROGRAM 命令

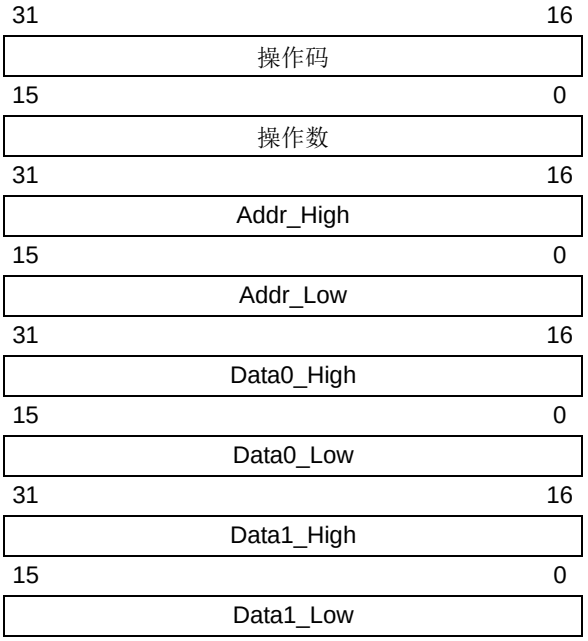
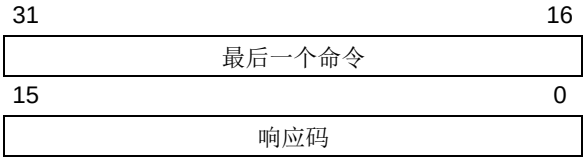


表 16-15： DOUBLE_WORD_PROGRAM 格式

字段	说明
操作码	0x0E
操作数	不使用
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

预期的响应 （1 字）：

图 16-27： DOUBLE_WORD_PROGRAM 响应



16.3 JTAGEN 配置位编程

对于 PIC32MM00XXGPL0XX 系列器件，如果 FICD 或 AFICD 配置字中的 JTAGEN 配置位已编程，则必须紧接着任何 PROGRAM 命令（如 DOUBLE_WORD_PROGRAM 或 ROW_PROGRAM）选择 MCHP TAP 控制器。这可以通过 SendCommand(MTAP_SW_MTAP)伪操作完成。在400 µs 后，可以通过调用 SendCommand(MTAP_SW_ETAP) 伪操作再次选择 EJTAG TAP 控制器。

17.0 校验和

17.1 原理

校验和的计算方式为：对程序闪存、引导闪存（除器件配置字外）、带适用掩码的器件 ID 寄存器和带适用掩码的器件配置字中的所有字节（8 位数量）进行 32 位求和。然后，将计算该和的二进制补码。这个最终的 32 位数就是校验和。

17.2 掩码值

器件配置的掩码值是通过将所有未实现位设置为 0，并将所有已实现位设置为 1 计算得出。

例如，寄存器 17-1 显示了 PIC32MM0064GPL036 器件的 FICD 寄存器。此寄存器的掩码值为：

mask_value_FCID = 0xFFFFFFF4

寄存器 17-1： PIC32MM0064GPL036 的 FICD 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-1	r-1	r-1	r-1	R/P-1
	—	—	—	—	—	—	—	BWP
23:16	r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
	—	—	—	—	—	—	—	—
15:8	r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
	—	—	—	—	—	—	—	—
7:0	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
	—	—	—	ICESEL<1:0>		FJTAGEN	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 PIC32MM 器件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。

表 17-1： PIC32MM 器件的器件配置寄存器掩码值

	配置字						DEVID
	FDEVOPT	FICD	FPOR	FWDT	FOSCEL	FSEC	
寄存器掩码	0xFFFFFFFF	0xFFFFFFFC	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0x3FFFFFFF	0x0FFFF000

17.3 算法

图 17-1 给出了计算 PIC32MM 器件校验和的高级算法的示例，该示例说明了得出校验和的方法。这仅仅是一个说明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。

如前所述， PIC32MM 校验和的计算方式为：对程序闪存、引导闪存 （除器件配置字外）、带适用掩码的器件 ID 寄存器和带适用掩码的器件配置字中的所有字节 （8 位量）进行 32 位求和。

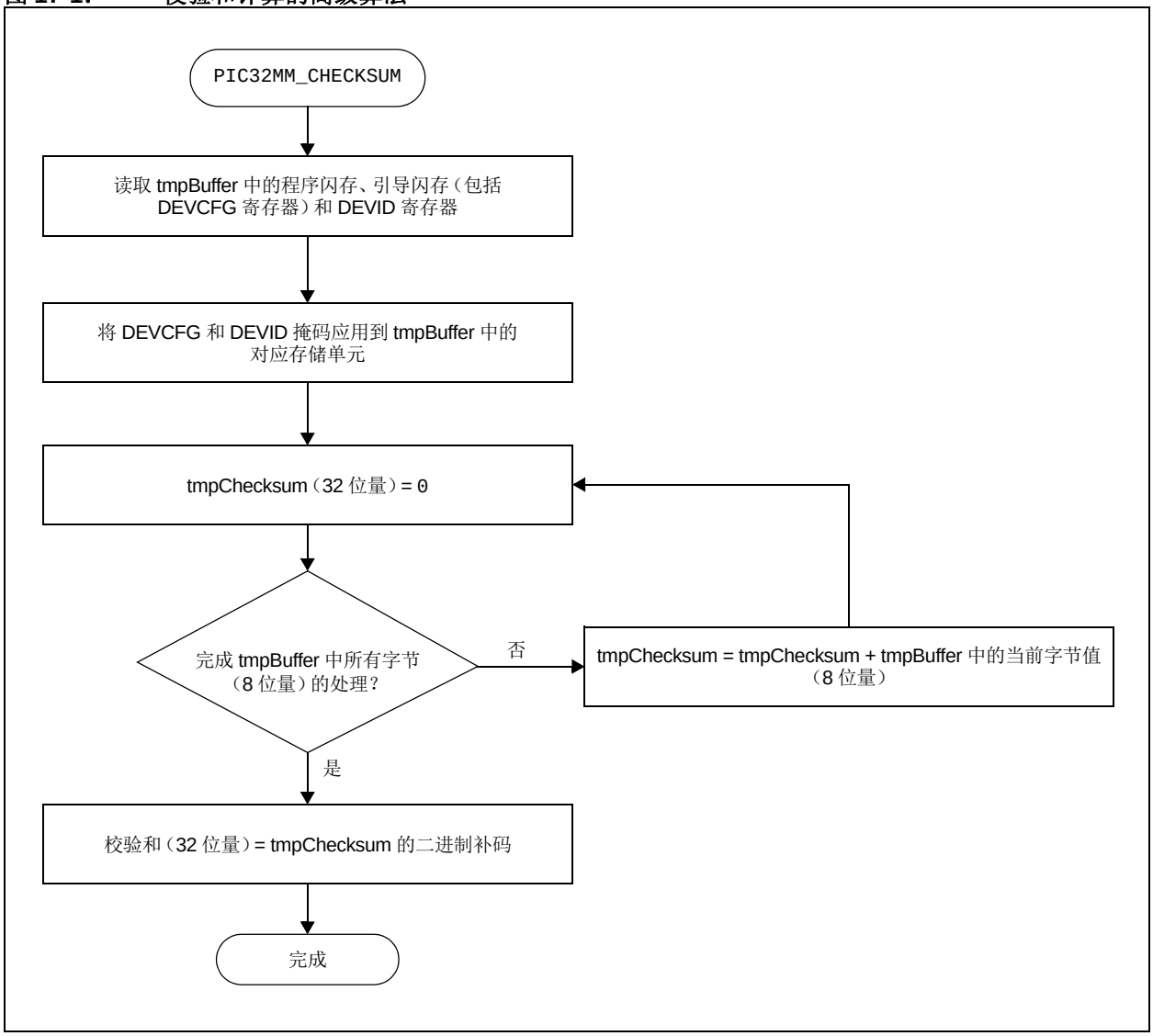
然后，将计算该和的二进制补码。这个最终的 32 位数就是校验和。

器件配置和器件ID寄存器的掩码值的计算方法如前面的**第 17.2 节 “掩码值”**中所述。

在将器件配置寄存器值的字节加到校验和之前，需要使用适当掩码值来执行这些寄存器值的算术与运算。

同样，在将器件 ID 寄存器的字节加到校验和之前，需要使用适当掩码值来执行该寄存器的算术与运算（更多信息，请参见**第 18.0 节 “配置存储区器件 ID 和唯一器件标识符”**）。

图 17-1： 校验和计算的高级算法



用于计算 PIC32MM 器件校验和的公式如公式 17-1 中所示。

公式 17-1： 校验和公式

校验和 = 二进制补码（PF + BF + DCR + DIR）

其中：

PF = 程序闪存中所有字节的 32 位求和

BF = 引导闪存 （器件配置寄存器除外）中所有字节的 32 位求和

DCR = 各字节的 32 位求和 （FDEVOPT + FICD + FPOR + FWDT + FOSCSEL + FSEC）

DIR = 字节的 32 位求和

MASKID = 来自表 17-1 的掩码值

17.4 校验和计算的示例

以下五节说明了如何使用公式 17-1 计算 PIC32MM0064GPL036 器件的校验和。

对此校验和计算示例进行了以下假定：

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

公式右侧的每一项（PF、BF、DCR 和 DIR）都需要单独计算。得出值后，就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 64 KB，相当于 65536 字节。由于假定程序闪存处于已擦除状态，因此 “PF” 的值可通过以下计算求得：

PF = 0xFF + 0xFF + … 65536 次

PF = 0x00FF0000 （32 位数）

17.4.2 计算校验和公式中的 “BF”

引导闪存的大小为 6 KB，相当于 6144 字节。但是，最后 256 字节是器件配置寄存器和保留的存储单元，需单独处理。因此，我们在此步骤中考虑的引导闪存中的字节数为 5888。由于假定引导闪存处于已擦除状态，因此 “BF” 的值可通过以下计算求得：

BF = 0xFF + 0xFF + … 5888 次

BF = 0x0016E900 （32 位数）

17.4.3 计算校验和公式中的 “DCR”

由于器件配置寄存器处于其默认状态，因此通过相应配置寄存器的值 （如 PIC32MM 内核所读取）、其各自的掩码值、应用掩码得出的值和字节的 32 位求和（全部如表 17-2 中所示）可以得到字节的 32 位求和的总和。

根据表 17-2，“DCR” 的值为：

DCR = 0x000003D6 （32 位数）

表 17-2： 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 位求和
FDEVOPT	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FICD	FFFFFFFF	FFFFFFFC	FFFFFFFC	0x000003F9
FPOR	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FWDT	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FOSCEL	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FSEC	FFFFFFFF	3FFFFFFF	3FFFFFFF	0x0000033C
			字节的 32 位求和的总和 =	**0x00001725**

17.4.4 计算校验和公式中的 “DIR”

器件 ID 寄存器的值、其掩码值、应用掩码得出的值和字节的 32 位求和如表 17-3 中所示。

根据表 17-3，“DIR”的值为：

DIR = 0x00000083 （32 位数）

表 17-3： 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 位求和
DEVID	0x07708053	0x0FFFF000	0x07708000	0x000000F7

17.4.5 完成校验和计算

在前面几节中得出的值（PF、BF、DCR 和 DIR）用于计算校验和值。执行前面几节中得出的 PF、BF、DCR 和 DIR 的 32 位求和，然后将其存储在一个称为 *temp* 的变量中，如例 17-1 中所示。

17.4.6 器件处于代码保护状态时的校验和值

由于器件配置字在 PIC32MM 器件处于代码保护状态时不可读取，因此对于所有器件而言校验和值均为零。

例 17-1： 校验和计算过程

1. 首先， *temp = PF + BF + DCR + DIR*，即：
   temp = 0x00FF0000 + 0x0016E900 + 0x00001725 + 0x000000F7
2. 将全部四个值相加可得到 *temp* 等于 0x0116011C。
3. 然后，计算 *temp* 的二进制反码，称为 *temp1*：
   temp1 = 二进制补码 （temp），现在等于 0xFEE9FEE3
   校验和 = 二进制补码 （temp），即 *校验和 = temp1 + 1*，得到 0xFEE9FEE4

18.0 配置存储区器件 ID 和唯一器件标识符

PIC32MM 器件具有多个功能，旨在最大程度提高应用灵活性和可靠性，并通过消除外部元件最大限度降低成本。这些功能可通过每个器件的特定配置位进行配置。

有关可用功能、配置位和器件 ID 寄存器的完整列表，请参见特定器件数据手册中的 **“特殊功能”** 章节。

有关特定器件的当前硅片版本和版本 ID，请参见相关系列硅片勘误表和数据手册澄清。这些文档可通过访问 Microchip 网站 http://www.microchip.com/PIC32 并导航到以下位置获取：*文档 > 勘误表*。

18.1 器件 ID

器件 ID 存储区可用于确定芯片的型号和生产信息。该存储区只读，并且能在使能代码保护时读取。 DEVID 寄存器可识别器件的具体部件编号、器件芯片版本和制造 ID。

表 18-1 列出了每个器件的标识信息。表 18-2 给出了器件 ID 寄存器，表 18-3 介绍了每个寄存器的位域。

表 18-1： 器件 ID

器件	DEVID
PIC32MM0016GPL020	0x6B04
PIC32MM0032GPL020	0x6B0C
PIC32MM0064GPL020	0x6B14
PIC32MM0016GPL028	0x6B02
PIC32MM0032GPL028	0x6B0A
PIC32MM0064GPL028	0x6B12
PIC32MM0016GPL036	0x6B06
PIC32MM0032GPL036	0x6B0B
PIC32MM0064GPL036	0x6B16
PIC32MM0064GPM028	0x7708
PIC32MM0128GPM028	0x7710
PIC32MM0256GPM028	0x7718
PIC32MM0064GPM036	0x770A
PIC32MM0128GPM036	0x7712
PIC32MM0256GPM036	0x771A
PIC32MM0064GPM048	0x772C
PIC32MM0128GPM048	0x7734
PIC32MM0256GPM048	0x773C
PIC32MM0064GPM064	0x770E
PIC32MM0128GPM064	0x7716
PIC32MM0256GPM064	0x771E

表 18-2： PIC32MM 器件 ID 寄存器

地址	名称	Bit															
		31/15	30/14	29/13	28/12	27/11	26/10	25/9	24/8	23/7	22/6	21/5	20/4	19/3	18/2	17/1	16/0
0xBF803B20	ID	VER<3:0>				DEVID<15:4>											
		DEVID<3:0>				制造 ID<11:0>											

表 18-3： 器件 ID 位域说明

位域	说明
VER<3:0>	器件硅片版本
DEVID<15:0>	对每个器件 ID 进行编码
制造 ID<11:0>	0x053

18.2 器件唯一标识符 （UDID）

在最终制造过程中，会为所有 PIC32MM 器件单独编码器件唯一标识符 （Unique Device Identifier， UDID）。整片擦除命令或其他任何用户可使用的方法都无法擦除 UDID。在需要对 Microchip 器件进行制造追踪的应用中，可以利用该功能来实现该目的。应用制造商也可以将它用于可能需要唯一标识的任意应用场合，例如：

- 追踪器件
- 唯一序列号
- 唯一安全密钥

UDID 包含 5 个连续的 32 位只读存储单元，它们位于器件配置空间的地址 0x1FC41840 至 0x1FC41850。这些数据组合在一起，构成一个 160 位标识符。表 18-4 列出了标识符字的地址。

表 18-4： UDID 地址

UDID	地址	说明
UDID1	0x1FC41840	UDID 字 1
UDID2	0x1FC41844	UDID 字 2
UDID3	0x1FC41848	UDID 字 3
UDID4	0x1FC4184C	UDID 字 4
UDID5	0x1FC41850	UDID 字 5

注： 给定部件的UDID值仅在其器件系列中保持唯一。有关系列中的器件列表，请参见器件数据手册中的系列器件表 **（表 1）**。

18.3 器件配置

在 PIC32MM 器件中，配置字可在执行任何代码之前选择在器件复位时设置的各种器件配置。这些值位于引导闪存（BFM）的高地址存储单元，且由于它们是程序存储器的一部分，因此将与可执行代码和程序常量一起包含在编程文件中。表 18-1 至表 18-5 中列出了这些配置字的名称和存储单元。表 18-5 列出了 PIC32MM 系列器件的配置字存储单元。

发生上电复位 （Power-on Reset， POR）或任何复位时，会将配置字从引导闪存复制到其对应的配置寄存器。配置位只能编程为 0 （未编程状态 = 1）。

注： 在不进行擦除的情况下多次写入闪存单元将违反闪存规范，并且可能缩短闪存分区的使用寿命。由于 ECC 实现，用不同的数据重写闪存单元将导致在读取该存储单元时发生 ECC 错误。

在对配置字进行编程后，器件复位将导致向配置寄存器中装载新值。因此，编程器应在器件校验之前对配置字进行编程。最后一个步骤是对代码保护配置字进行编程。

这些配置字可确定振荡器源。如果使用 2 线增强型 ICSP 模式，将忽略配置字，且器件将总是使用 FRC；但是，在 4 线模式中，情况并非如此。如果在复位后器件中并不存在的配置字选择了振荡器源，则编程器将无法在器件复位后对器件执行闪存操作。有关使用配置字选择振荡器的详细信息，请参见特定器件数据手册中的 **“特殊功能”** 章节。

表 18-5： 主要配置字

配置字	物理地址	
—(1)	0x1FC017C0	BCFG4(2)
FDEVOPT	0x1FC017C4	
FICD	0x1FC017C8	BCFG3(2)
FPOR	0x1FC017CC	
FWDT	0x1FC017D0	BCFG2(2)
FOSCEL	0x1FC017D4	
FSEC	0x1FC017D8	BCFG1(2)
—(1)	0x1FC017DC	
FSIGN	0x1FC017E0	BCFG0(2)
—(1)	0x1FC017E4	

注 1： 这些存储单元不包含器件配置信息，但必须进行编程，因为它们是通过配置信息共享的双字的一部分。建议使用值 0xFFFFFFFF 对它们进行编程。

2： 每个 BCFGx 包含组成一个双字的 2 个闪存单元。双字中的两个存储单元都必须通过一个双字或一个行编程操作一起进行编程。

表 18-6：　　备用配置字

配置字	物理地址	
__(1)	0x1FC01740	ABCFG4(2)
AFDEVOPT	0x1FC01744	
AFICD	0x1FC01748	ABCFG3(2)
AFPOR	0x1FC0174C	
AFWDT	0x1FC01750	ABCFG2(2)
AFOSCEL	0x1FC01754	
AFSEC	0x1FC01758	ABCFG1(2)
__(1)	0x1FC0175C	
AFSIGN	0x1FC01760	ABCFG0(2)
__(1)	0x1FC01764	

注 1： 这些存储单元不包含器件配置信息，但必须进行编程，因为它们是通过配置信息共享的双字的一部分。建议使用值 0xFFFFFFFF 对它们进行编程。

2： 每个 BCFGx 包含组成一个双字的 2 个闪存单元。双字中的两个存储单元都必须通过一个双字或一个行编程操作一起进行编程。

18.3.1　　配置寄存器编程

配置寄存器必须先擦除才能重新写入。未执行擦除操作就重新编程将违反闪存规范，损害闪存耐用性，并产生 ECC 错误。

18.3.2　　备用配置字

PIC32MM 器件包含两组配置字：主配置字和备用配置字。主配置字在器件复位后从闪存读取。如果在读取过程中发生不可纠正的 ECC 错误，将读取备用配置字。在典型的应用中，主和备用配置字应编程为相同的值。

为了计算校验和，将仅使用主配置字。

18.4　器件代码保护位 （CP）

PIC32MM 系列器件具有代码保护功能，可防止外部编程器件读取闪存。使能代码保护之后，只能使用整片擦除命令 （MCHP_ERASE）擦除器件来禁止此功能。

对选择使用代码保护的器件进行编程时，必须在使能代码保护之前对编程器件执行校验。使能代码保护应该是编程过程的最后一步。代码保护使能位的位置因器件而异。有关详细信息，请参见特定器件数据手册中的**“特殊功能”**章节。

注： 使能代码保护之后，无法再读取闪存，而只能使用整片擦除命令 （MCHP_ERASE）通过外部编程器禁止闪存。

18.5　程序写保护位 （PWP）

PIC32MM 系列器件具有写保护功能，可防止在程序执行期间擦除或写入指定的引导和程序闪存。写保护通过闪存控制器中的 SFR 实现。

外部编程器对器件进行初始化期间需要执行某些步骤，然后才能对闪存进行编程。有关详细信息，请参见特定器件数据手册中的 **“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 连接到 MCHP 命令移位寄存器 （见表 19-2）
MTAP_SW_MTAP	5'h0x04	将 TAP 控制器切换为 Microchip 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选择MCHP命令移位寄存器。有关可用命令，请参见表 19-2。

19.1.1.1 MCHP_STATUS 指令

MCHP_STATUS 返回 Microchip TAP 控制器的 8 位状态值。表 19-3 提供返回的状态值的格式。

19.1.1.2 MCHP_ASSERT_RST 指令

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

19.1.1.3 MCHP_DE_ASSERT_RST 指令

MCHP_DE_ASSERT_RST 取消持久性器件复位。它类似于将 MCLR 置为无效。其关联的状态位为 DEVRST。

19.1.1.4 MCHP_ERASE 指令

MCHP_ERASE 执行整片擦除。CHIP_ERASE 命令将一个请求闪存控制器执行擦除的内部位置 1。一旦控制器变为繁忙状态 （通过 FCBUSY （状态位）指示），内部位将清零。

19.1.2 MTAP_SW_MTAP 指令

MTAP_SW_MTAP 将 TAP 指令集切换为 MCHP TAP 指令集。

19.1.3 MTAP_SW_ETAP 指令

MTAP_SW_ETAP 将 TAP 指令集有效切换为 EJTAG TAP 指令集。为此，它会将 EJTAG TAP 控制器保持在运行测试 / 空闲状态，直到 MCHP TAP 控制器对 MTAP_SW_ETAP 指令进行解码。

19.1.4 MTAP_IDCODE 指令

MTAP_IDCODE 返回存储在 DEVID 寄存器中的值。

表 19-2： MTAP_COMMAND DR 命令

命令	值	说明
MCHP_STATUS	8'h0x00	NOP 和返回状态。
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	取消器件复位请求，如果没有其他请求复位的源 （如 $\overline{\text{MCLR}}$），这会使复位控制器将器件复位置为无效。
MCHP_ERASE	8'h0xFC	使闪存控制器执行整片擦除。

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	0	CFGRDY	FCBUSY	0	DEVRST

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

bit 6 **未实现**：读为 0

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

bit 4 **未实现**：读为 0

bit 3 **CFGRDY**：代码保护状态位
1 = 配置已读取且 CP 为有效
0 = 配置未读取

bit 2 **FCBUSY**：闪存控制器忙状态位
1 = 闪存控制器正忙 （正在擦除）
0 = 闪存控制器不忙 （擦除未开始或已完成）

bit 1 **未实现**：读为 0

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

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可选择地址寄存器。只读地址寄存器提供处理器访问的地址。如果处理器访问处于待处理状态，则在该寄存器中读取的值为有效，否则该值处于未定义状态。

该寄存器的两个或三个最低有效字节（Least Significant Byte，LSB）与 EJTAG 控制寄存器的 Psz<1:0> 位域一起用于指示待处理的处理器访问传输的大小和数据位置。这些位并非直接取自装载 / 存储所引用的地址。

19.2.2 ETAP_DATA 命令

ETAP_DATA 选择数据寄存器。读 / 写数据寄存器用于处理器访问期间的操作码和数据传输。只有当处理器写访问处于待处理状态，在该寄存器中读取的值才为有效，在这种情况下，数据寄存器将保留存储的值。只有处理器将在之后完成待处理读访问时，才会使用写入到数据寄存器中的值，在这种情况下，写入的数据值为取出或装载的值。此行为暗示数据寄存器并非之前写入的值可以在之后读取的存储单元。

19.2.3 ETAP_CONTROL 命令

ETAP_CONTROL 选择 EJTAG 控制寄存器。EJTAG 控制寄存器（EJTAG Control Register，ECR）处理以下各项操作：处理器复位、软复位指示、调试模式指示、访问开始、完成和大小以及读 / 写指示。ECR 还提供以下功能：

- 控制调试向量地址和处理的处理器器访问的指示
- 可发出调试中断请求
- 指示处理器的低功耗模式
- 支持依赖于实现的处理器和外设复位

19.2.3.1 EJTAG 控制寄存器 （ECR）

除非发生了复位，否则将不会在“更新 -DR”状态中更新 / 写入 EJTAG 控制寄存器 （见寄存器 19-1），也就是说，Rocc（bit 31）已经为 0，或将被同时写入为 0。此条件可确保在复位后正确处理处理器访问。

在处理器被从处理器时钟域中移除若干个TCK周期后，处理器的复位可通过 TCK 域中的 Rocc 位进行指示，以便能够在两个时钟域之间实现正确同步。

除非定义了其他行为，否则在寄存器中可读 / 写（Read/Write，R/W）的位将在后续读取时返回其写入的值。

内部同步可确保写入的值为紧接之后的读取进行更新，即使 TAP 控制器采取从“更新 -DR”到“捕捉 -DR”状态的最短路径时也是如此。

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　请参见**注 1**

bit 28-24　**未实现：**读为 0

bit 23-19　请参见**注 1**

bit 18　**PrACC：**待处理的处理器访问和控制位
此位指示待处理的处理器访问，并控制待处理的处理器访问的完成。写入 0 可完成待处理的处理器访问。写入 1 则将忽略。成功的 Fastdata 访问会将此位清零。
1 = 待处理的处理器访问
0 = 无待处理的预处理器访问

bit 17　**未实现：**读为 0

bit 16　请参见**注 1**

bit 15　**ProbEn：**处理器访问处理控制位
此位控制探针通过处理处理器访问而处理 DMSEG 段访问的位置。
1 = 探针处理处理器访问
0 = 探针不处理处理器访问

bit 14　**ProbTrap：**调试异常向量地址控制位
此位控制调试异常向量的地址。
1 = 0xFF200200
0 = 0xBFC00480

bit 13　**未实现：**读为 0

bit 12　**EjtagBrk：**调试中断异常请求位
此位可在写为 1 时向处理器请求调试中断异常。写入 0 则将忽略。
1 = 调试中断异常请求处于待处理状态
0 = 调试中断异常请求未处于待处理状态

bit 11-4　**未实现：**读为 0

bit 3　请参见**注 1**

bit 2-0　**未实现：**读为 0

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

19.2.4 ETAP_EJTAGBOOT 命令

ETAP_EJTAGBOOT 命令使处理器在复位后从调试异常向量取代码。这使编程器可以将指令发送到处理器执行，而不是使处理器从正常的复位向量取指令。在设置内部 EJTAGBOOT 指示之后是 EjtagBrk、 ProbTrap 和 ProbEn 的复位值。

如果 EJTAGBOOT 指令已给出，且内部 EJTAGBOOT 指示处于有效状态，则这三个位的复位值将被置 1（1）；否则复位值将被清零 （0）。

这些位置 1 的结果为：

- 如将 EjtagBrk 置 1，则将在紧接处理器从 EJTAGBOOT 指令复位后请求调试中断异常
- 将从 EJTAG 存储区执行调试处理程序，因为 ProbTrap 被置 1，指示 EJTAG 存储区中的调试向量位于 0xFF200200
- ProbEn 被置 1 将指示处理器访问的处理

完成此配置后，将发生中断异常，且处理器将在 0xFF200200 从 DMSEG 取处理程序。在 ProbEn 置 1 后，处理器将等待探针探针提供指令。

19.2.5 ETAP_FASTDATA 命令

ETAP_FASTDATA 命令提供了在处理器和探针之间快速传输数据的机制。Fastdata 寄存器的宽度为一位。在快速数据访问期间，将写入和读取 Fastdata 寄存器（即移入一位，并移出一位）。在快速数据访问期间，移入的 Fastdata 寄存器值可指定是否应完成快速数据访问。移出的值是一个标志，用于指示快速数据访问是否成功（如果请求完成）。快速数据访问用于 DMSEG 段 （探针上）和目标存储器 （处理器上）之间的高效块传输。“上传”定义为一个处理器从目标存储器装载并存储到 DMSEG 段的序列。“下载”是指一个处理器从 DMSEG 段装载并存储到目标存储器的序列。“快速数据区”指定可用于上传和下载的 DMSEG 段地址的合法范围（0xFF200000 到 0xFF20000F）。数据和 Fastdata 寄存器 （使用 FASTDATA 指令选择）可高效地完成待处理快速数据区的访问。

在快速数据上传和下载期间，处理器将在访问快速数据区时暂停。 PrACC （处理器访问待处理）位将变为 1，这表示需要使用探针才能完成访问。上传和下载访问的尝试方式都是，移入一个零 SPrAcc 值 （以请求完成访问）并移出 SPrAcc，以查看尝试是否成功 （即存在一个待处理访问并且使用了合法的快速数据区地址）。

下载也将移入用于满足从 DMSEG 段快速数据区的装载的数据，同时上传将移出存储到 DMSEG 段快速数据区的数据。

如上文所述，以下两个条件必须为真，快速数据访问才能成功：

- PrACC 必须为 1 （即必须存在待处理的处理器访问）
- 快速数据操作必须使用 DMSEG 段中的有效快速数据区地址 （0xFF200000 到 0xFF20000F）

20.0 交流 / 直流特性和时序要求

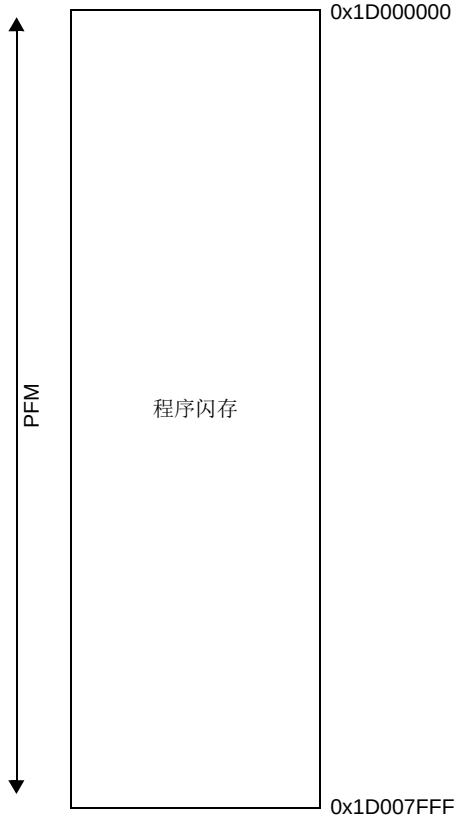
表 20-1： 交流 / 直流特性和时序要求

标准工作条件 工作温度：0ºC 到 +70ºC。建议在 +25ºC 下编程。						
参数编号	**符号**	**特性**	**最小值**	**最大值**	**单位**	**条件**
D111	VDD	编程时的电源电压	—	—	V	见**注 1**
D113	IDDP	编程时的电源电流	—	—	mA	见**注 1**
D114	IPEAK	启动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	整片擦除时间	—	—	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 输入密钥序列的最后一个 PGECx ↓到第二个 $\overline{\text{MCLR}}$ ↑之间的延时	40	—	ns	
P20	TMCLRH	$\overline{\text{MCLR}}$ 高电平时间	—	500	µs	

注 1：有关此参数的最小值和最大值，请参见特定器件数据手册中的 **“电气特性”**章节。

附录 A： PIC32MM 闪存映射

图 A-1： 闪存映射



注： 所示的存储器映射仅供参考。有关器件的存储器映射，请参见特定器件数据手册中的**“存储器构成”**章节。

附录 B： HEX 文件格式

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

Hex 文件的基本格式为：

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

每条数据记录以 9 个字符的前缀开始，并总是以 2 个字符的校验和结尾。不管什么格式，所有记录均以 : 开始。各个元素如下所述。

- BB—— 是一个两位的十六进制字节计数，表示出现在行中的数据字节数。将此数值除以二就可以得到每行的字数。
- AAAA—— 是一个四位的十六进制地址，表示数据记录的开始地址。格式为高字节在前，低字节在后。地址值翻倍，因为此格式仅支持 8 位。将此值除以二就可以得到实际的器件地址。
- TT—— 是一个两位记录类型，对于数据记录为 00，对于文件结束 （End-of-File， EOF）记录为 01，对于延长的地址记录为 04。
- HHHH—— 是一个四位的十六进制数据字。格式为低字节在前，高字节在后。在 TT 后将有 BB/2 个数据字。
- CC—— 是一个两位的十六进制校验和，是行记录中所有之前字节的总和的二进制补码。

附录 C： 版本历史

版本 A （2015 年 9 月）

这是支持 PIC32MMXXXXGPLXXX 器件的文档的初始版本。

版本 B （2015 年 10 月）

添加了 PIC32MMXXXXGPMXXX 器件。

版本 C （2016 年 5 月）

添加**第 16.3 节 “JTAGEN 配置位编程”**并更新了表 18.1。

从文档中删除了 PIC32MM0XXXGPM0XX 器件。

版本 D （2017 年 3 月）

在**第 1.0 节 “器件概述”**列表中新增了器件。

在表 18-1 中新增了器件。

更新了**第 5.3 节 “2 线 ICSP 详细信息”**。

删除了**第 5.3.2 小节 “2 相 ICSP”**。

删除了图 5-5 和图 6-8。

请注意以下有关 Microchip 器件代码保护功能的要点：

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

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

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

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

ISBN：978-1-5224-2818-3

全球销售及服务网点

美洲

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

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

奥斯汀 Austin, TX
Tel: 1-512-257-3370

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

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

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

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

休斯敦 Houston, TX
Tel: 1-281-894-5983

印第安纳波利斯 Indianapolis
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453
Tel: 1-317-536-2380

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

罗利 Raleigh, NC
Tel: 1-919-844-7510

纽约 New York, NY
Tel: 1-631-435-6000

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

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

亚太地区

中国 - 北京
Tel: 86-10-8569-7000

中国 - 成都
Tel: 86-28-8665-5511

中国 - 重庆
Tel: 86-23-8980-9588

中国 - 东莞
Tel: 86-769-8702-9880

中国 - 广州
Tel: 86-20-8755-8029

中国 - 杭州
Tel: 86-571-8792-8115

中国 - 南京
Tel: 86-25-8473-2460

中国 - 青岛
Tel: 86-532-8502-7355

中国 - 上海
Tel: 86-21-3326-8000

中国 - 沈阳
Tel: 86-24-2334-2829

中国 - 深圳
Tel: 86-755-8864-2200

中国 - 苏州
Tel: 86-186-6233-1526

中国 - 武汉
Tel: 86-27-5980-5300

中国 - 西安
Tel: 86-29-8833-7252

中国 - 厦门
Tel: 86-592-238-8138

中国 - 香港特别行政区
Tel: 852-2943-5100

中国 - 珠海
Tel: 86-756-321-0040

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600

台湾地区 - 新竹
Tel: 886-3-577-8366

亚太地区

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

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

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

印度 India - Pune
Tel: 91-20-4121-0141

日本 Japan - Osaka
Tel: 81-6-6152-7160

日本 Japan - Tokyo
Tel: 81-3-6880-3770

韩国 Korea - Daegu
Tel: 82-53-744-4301

韩国 Korea - Seoul
Tel: 82-2-554-7200

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

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

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

新加坡 Singapore
Tel: 65-6334-8870

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

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

欧洲

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

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

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

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

德国 Germany - Garching
Tel: 49-8931-9700

德国 Germany - Haan
Tel: 49-2129-3766400

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

德国 Germany - Karlsruhe
Tel: 49-721-625370

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

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

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

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

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

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

挪威 Norway - Trondheim
Tel: 47-7289-7561

波兰 Poland - Warsaw
Tel: 48-22-3325737

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

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

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

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

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