dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列编程规范

dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列

编程概述

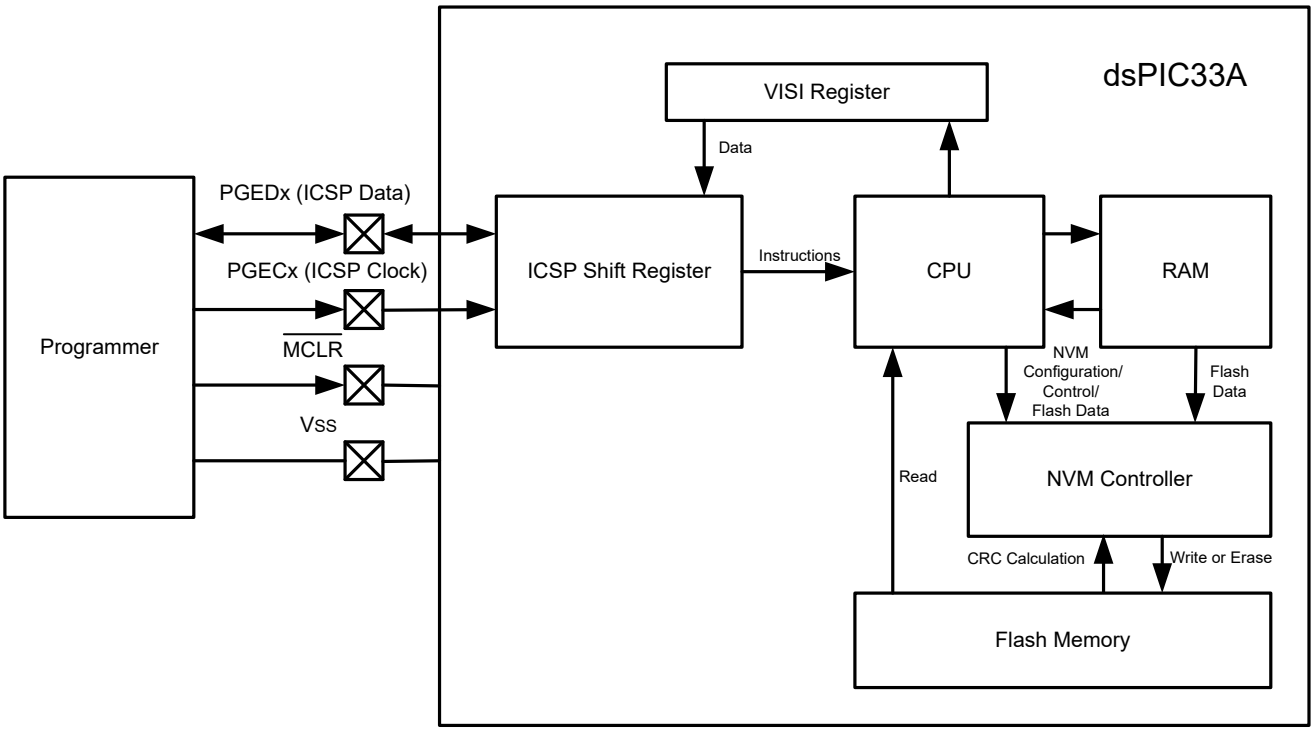
本文档定义了如下 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列数字信号控制器（Digital Signal Controller，DSC）器件的编程规范：

- dsPIC33AK256MC205
- dsPIC33AK256MC206
- dsPIC33AK256MC208
- dsPIC33AK256MC210
- dsPIC33AK256MC505
- dsPIC33AK256MC506
- dsPIC33AK256MC508
- dsPIC33AK256MC510
- dsPIC33AK512MC205
- dsPIC33AK512MC206
- dsPIC33AK512MC208
- dsPIC33AK512MC210
- dsPIC33AK512MC505
- dsPIC33AK512MC506
- dsPIC33AK512MC508
- dsPIC33AK512MC510
- dsPIC33AK256MPS205
- dsPIC33AK256MPS206
- dsPIC33AK256MPS208
- dsPIC33AK256MPS210
- dsPIC33AK256MPS212
- dsPIC33AK256MPS505
- dsPIC33AK256MPS506
- dsPIC33AK256MPS508
- dsPIC33AK256MPS510
- dsPIC33AK256MPS512
- dsPIC33AK512MPS205
- dsPIC33AK512MPS206
- dsPIC33AK512MPS208
- dsPIC33AK512MPS210
- dsPIC33AK512MPS212
- dsPIC33AK512MPS505
- dsPIC33AK512MPS506
- dsPIC33AK512MPS508
- dsPIC33AK512MPS510
- dsPIC33AK512MPS512

编程通过在线串行编程（In-Circuit Serial Programming™，ICSP™）接口来实现，具体包括时钟引脚和数据引脚（PGECx 和 PGEDx）。

dsPIC33AK512MC510 和 dsPIC33AK512MPS512 器件具有非易失性存储器（Nonvolatile Memory，NVM）控制器模块。该 NVM 模块用于编程器件闪存，其操作由 CPU 进行配置和管理。外部编程器工具使用串行编程接口（ICSP）来移入和执行 CPU 指令、将数据移入和移出器件，以及配置 NVM 控制器进行编程操作。编程器通过向 CPU 发送 MOV 指令，将要编程的闪存数据（准备好）存入器件 RAM 或 NVM 控制寄存器（具体取决于编程模式）。之后，编程器执行 CPU 指令以使用 NVM 控制器启动擦除操作或写操作。为了从器件读取闪存数据，实现了 VISI 寄存器。该寄存器的内容可移出至 ICSP 接口。为了快速验证闪存内容，NVM 控制器支持 CRC-32 校验和引擎。该引擎可减少应通过 ICSP 通信引脚读取的数据量。 图 1 说明了编程器与器件之间的交互。

图 1. 编程交互框图



目录

1. 闪存

dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件上的闪存分为四字（每个 16 字节）、写入行（每个 512 字节）和擦除页（每个 4096 字节）。四字（四个长度为 32 位的字）是可写入闪存的最小数据大小。每个四字的地址按 16 字节对齐。此外，还可按行写入闪存。每行由 32 个四字或 512 个字节组成。每行的地址按 512 字节边界对齐。

必须先擦除闪存才能启动写操作。可按页擦除闪存。每个擦除页由 8 行或 4096 个字节组成。每个擦除页的地址按 4096 字节边界对齐。

表 1-1 列出了 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件的闪存区域。

表 1-1. 闪存映射

存储器区域			地址	四字数	写入行数	擦除页数
用户 OTP			0x7F2C00-0x7F2FFC	64	2	不适用
用户配置 A1（User Configuration A1，UCA1）			0x7F3000-0x7F3FFC	256	8	1
用户配置 B（User Configuration B，UCB）			0x7F4000-0x7F4FFC	256	8	1
用户配置 A2（User Configuration A2，UCA2）			0x7FB000-0x7FBFFC	256	8	1
代码存储区	256 KB 闪存器件	单引导	800000-83FFFC	16,384	512	64
		双引导	800000-81FFFC	8,182	256	32
			C00000-C1FFFC	8,182	256	32
	512 KB 闪存器件	单引导	800000-87FFFC	32,768	1024	128
		双引导	800000-83FFFC	16,384	512	64
			C00000-C3FFFC	16,384	512	64

在编程过程中会使用 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件的一些寄存器。表 1-2 列出了这些寄存器的地址和说明。

表 1-2. 编程过程中使用的寄存器

寄存器名称	地址	说明
VISI	0x0007C0	该寄存器用于通过 ICSP™接口将数据移出器件。
NVMCON	0x003000	该寄存器用于选择类型以及启动闪存擦除操作或写操作。
NVMADR	0x003004	擦除操作或写操作的目标闪存地址。
NVMDATA0	0x003008	该寄存器包含使用四字写操作时要编程到闪存的数据。
NVMDATA1	0x00300C	该寄存器包含使用四字写操作时要编程到闪存的数据。
NVMDATA2	0x003010	该寄存器包含使用四字写操作时要编程到闪存的数据。
NVMDATA3	0x003014	该寄存器包含使用四字写操作时要编程到闪存的数据。
NVMSRCADR	0x003018	该寄存器应设置为装载要编程的闪存行数据的 RAM 缓冲区的地址。
NVMCRCCON	0x003048	该寄存器控制闪存区域的循环冗余校验（Cyclic Redundancy Check，CRC）计算。
NVMCRCST	0x00304C	该寄存器包含用于 CRC 计算的 4 KB 闪存块的起始地址。
NVMCRCEND	0x003050	该寄存器包含用于 CRC 计算的 4 KB 闪存块减去一个字节后的结束地址。
NVMCRCSEED	0x003054	该寄存器中应载入初始 CRC 值（种子）。如果对多个存储器块计算 CRC-32，则必须先将前一个存储器块的 CRC 结果直接写入 NVMCRCSEED，再对下一个存储器块进行 CRC 计算。
NVMCRCDATA	0x003058	该寄存器包含 CRC 计算结果。

1.1. 闪存控制器

该闪存（或非易失性存储器（NVM））控制器模块用于对器件闪存执行擦除操作或写操作。NVMCON 寄存器中提供编程配置和状态。NVMCON 寄存器中的 NVMOP[3:0]位用于选择操作类型。表 1-3 列出了可用操作的说明。

表 1-3. 编程操作

操作类型	NVMOP 位（NVMCON[3:0]）值	说明
全片擦除	1110	擦除整个代码存储区和配置位存储区。不会擦除 OTP 区域。
非活动分区擦除	0100	擦除非活动分区。
页擦除	0011	擦除代码存储区或配置存储区的一页。必须在 NVMADR 寄存器中指定闪存页的地址。对于该 NVMOP，忽略 NVMADR[11:0]（视为全 0），通过硬件强制按 4096 字节页对齐。
行写	0010	写入代码存储区或配置存储区的一行。必须在 NVMADR 寄存器中指定闪存行的地址。必须将行数据（准备好）存入 RAM。必须在 NVMSRCADR 寄存器中指定闪存数据 RAM 缓冲区的地址。必须先擦除闪存再执行写操作。通过硬件忽略 NVMADR[8:0]，强制按 512 字节目标行对齐。未实现 NVMSRCADR[1:0]，因此 RAM 中的源数据必须从按 32 位字对齐的边界开始。
四字写	0001	向代码存储区、OTP 存储区或配置存储区中写入四个长度为 32 位的字。必须在 NVMADR 寄存器中指定要写入的闪存单元的地址。必须将数据（准备好）存入 NVMDATA0、NVMDATA1、NVMDATA2 和 NVMDATA3 寄存器。必须先擦除闪存再执行写操作。未实现 NVMADR[3:0]，强制按 16 字节四字地址对齐。

擦除或写入闪存时应遵循以下步骤：

1. 在 NVMADR 寄存器中设置要擦除或写入的目标地址。
2. 将写操作的数据装入 NVMDATA0-NVMDATA3 寄存器或 NVMSRCADR 寄存器中的地址指定的 RAM 缓冲区。
3. 使用 NVMOPx 位（NVMCON[3:0]）选择编程操作。
4. 通过将 WREN 位（NVMCON[14]）置 1 使能 NVM 模块操作。
5. 通过将 WR 位（NVMCON[15]）置 1 启动所选的操作。
6. 读取/轮询 WR 位（NVMCON[15]）。如果仍在进行擦除操作或写操作，则 WR 位保持置 1 状态。当操作完成时，由硬件清零该位。当 WR 位清零时，NVM 中断标志（NVMCRCIF）置 1。

有关 NVM 控制器的完整说明和编程详细信息，可参见器件数据手册。

此外，NVM 控制器还可以计算 32 位循环冗余校验（CRC）校验和。该校验和可用于验证闪存擦除或编程的结果。闪存区域的起始地址和结束地址按 4 KB（页）对齐。即使闪存区域受数据回读保护，也可以计算 CRC。

获取 CRC 校验和时应遵循以下步骤：

1. 将起始地址写入 NVMCRCST 寄存器。
2. 将结束地址写入 NVMCRCEND 寄存器。
3. 通过将 CRCEN 位（NVMCRCCON[15]）置 1 使能 CRC。
4. 通过将 START 位（NVMCRCCON[14]）置 1 开始计算。
5. 读取/轮询 START 位（NVMCRCCON[14]）。该位在 CRC 校验和就绪时清零。当 CRC 校验和就绪时，中断标志 NVMCRCIF 置 1。
6. 从 NVMCRCDATA 寄存器读取 CRC 结果。

尽管器件硬件实现了并行 CRC-32 计算算法，但在软件中通过使用 32 位移位寄存器的算法也可以获得相同的结果。例 1-1 中的伪代码说明了移位寄存器算法。

例 1-1. 移位寄存器 CRC 计算算法

```
// Initialize shift register
SeedValue(31:0) = 0
ShiftRegister(31:0) = INVERT SeedValue(31:0)
REPEAT FOR ALL DATA WORDS
    REPEAT 32 TIMES
        // Exclusive OR of data bit 31 and shift register bit 0
        CRCNext = DataWord(31) EXOR ShiftRegister(0)
        // Shift the register right
        ShiftRegister(30:0) = ShiftRegister(31:1)
        ShiftRegister(31) = 0
        // Shift data left to process the next bit of the word
        DataWord(31:1) = DataWord(30:0)
        IF CRCNext != 0 THEN
            // Exclusive OR with polynomial
            ShiftRegister(31:0) = ShiftRegister(31:0) EXOR 0xEDB88320
        ENDIF
    ENDREPEAT
ENDREPEAT
CRCResult = INVERT ShiftRegister(31:0)
```

其中：

- SeedValue[31:0]——初始 CRC 值（每个位的值均取反），写入 NVMCRCSEED 寄存器。对于标准 CRC-32 计算，NVMCRCIV 寄存器配置为 0xFFFF_FFFF。
- DataWord[31:0]——当前用于 CRC 计算的数据字
- CRCResult[31:0]——NVMCRCDATA 寄存器中的 CRC 校验和结果

1.2. 器件 ID（DEVID）

每个器件型号都有不同的存储器大小和封装选项，具体可通过 DEVID 寄存器中的器件 ID 来识别。器件版本存储在 REVID 寄存器中。表 1-4 列出了这两个寄存器的地址。

表 1-4. 器件 ID 寄存器和版本寄存器

寄存器名称	地址	说明
DEVID	0x7C2000	器件 ID（器件型号）
REVID	0x7C2004	器件版本

表 1-5 列出了每个器件型号（具有不同的存储器大小和引脚数）的器件 ID 值。

表 1-5. 器件 ID 值

器件	器件 ID 值
dsPIC33AK256MC205	0xA800
dsPIC33AK256MC206	0xA801
dsPIC33AK256MC208	0xA802
dsPIC33AK256MC210	0xA803
dsPIC33AK256MC505	0xA840
dsPIC33AK256MC506	0xA841
dsPIC33AK256MC508	0xA842
dsPIC33AK256MC510	0xA843
dsPIC33AK512MC205	0xA820
dsPIC33AK512MC206	0xA821
dsPIC33AK512MC208	0xA822
dsPIC33AK512MC210	0xA823
dsPIC33AK512MC505	0xA860
dsPIC33AK512MC506	0xA861
dsPIC33AK512MC508	0xA862
dsPIC33AK512MC510	0xA863
dsPIC33AK256MPS205	0xA818
dsPIC33AK256MPS206	0xA819

表 1-5. 器件 ID 值 （续）

器件	器件 ID 值
dsPIC33AK256MPS208	0xA81A
dsPIC33AK256MPS210	0xA81B
dsPIC33AK256MPS212	0xA81C
dsPIC33AK256MPS505	0xA858
dsPIC33AK256MPS506	0xA859
dsPIC33AK256MPS508	0xA85A
dsPIC33AK256MPS510	0xA85B
dsPIC33AK256MPS512	0xA85C
dsPIC33AK512MPS205	0xA838
dsPIC33AK512MPS206	0xA839
dsPIC33AK512MPS208	0xA83A
dsPIC33AK512MPS210	0xA83B
dsPIC33AK512MPS212	0xA83C
dsPIC33AK512MPS505	0xA878
dsPIC33AK512MPS506	0xA879
dsPIC33AK512MPS508	0xA87A
dsPIC33AK512MPS510	0xA87B
dsPIC33AK512MPS512	0xA87C

1.3. 器件惟一标识符（UDID）

所有 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件在最终制造期间都单独编码了器件惟一标识符（Unique Device Identifier，UDID）。全片擦除命令或其他任何用户可使用的方法都无法擦除 UDID。凭借该功能，可以在需要时跟踪应用中 Microchip 器件的制造信息。应用制造商也可将该功能用于任意个可能需要惟一标识的应用场合，例如：

- 跟踪器件
- 惟一序列号
- 惟一安全密钥

UDID 包含四个 32 位程序字。这些程序字组合在一起构成一个 128 位的惟一标识符。UDID 存储在器件配置空间中的四个只读单元中。 表 1-6 列出了 UDID 字。

表 1-6. UDID 字

寄存器名称	地址	说明
UDID1	0x7F2BE0	UDID bit 0 至 bit 31
UDID2	0x7F2BE4	UDID bit 32 至 bit 63
UDID3	0x7F2BE8	UDID bit 64 至 bit 95
UDID4	0x7F2BEC	UDID bit 96 至 bit 127

1.4. 配置位

配置位存储在用户配置 Ax 和用户配置 B 闪存区域中。通过编程（= 0）或擦除（= 1）这些位，可以选择不同的器件配置。有两种配置位类型：系统操作位和代码保护位。系统操作位确定系统级组件（如看门狗定时器）的上电设置。代码保护位阻止读写程序存储器。 表 1-7 列出了 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件的配置字。有关配置字寄存器的完整说明，请参见器件数据手册。

此外，两个区域（单引导）/三个区域（双引导）还具有备份副本。器件复位期间，配置数据会自动从闪存配置字装入到相应的配置影子寄存器中。表 1-7 列出了 CFGA1、CFGA2 和 CFGB 寄存器及其备份的位置。主寄存器与备份寄存器应同时编程，或者按照在双分区模式下编程中所述的顺序进行编程。

表 1-7. 配置字

配置字名称	地址	备份位置地址	配置区域	说明
FCP	0x7F3000	0x7F3800	UCA1(1)	该配置字用于使能闪存保护。
FICD	0x7F3010	0x7F3810		调试器配置寄存器。
FDEVOPT	0x7F3020	0x7F3820		该配置字用于配置一些外设的功能。
FWDT	0x7F3030	0x7F3830		该配置字用于控制看门狗定时器。

注：

1. 在双引导模式下工作时，双引导器件具有两个 UCA 页：UCA1 对应分区 1，UCA2 对应分区 2。

表 1-7. 配置字 （续）

配置字名称	地址	备份位置地址	配置区域	说明
FPRCTRL0	0x7F4000	0x7F4800	UCB	这些配置字可用于定义最多 8 个存储器区域，并对不同的闪存操作进行限制。FPRCTRLx 配置字用于使能区域并指定应限制哪类闪存访问（擦除、编程、读和 CRC 计算等）。FPRSTx 和 FPRENDx 应分别编程为区域中第一个 4 KB 页的地址和最后一个 4 KB 页的地址。
FPRST0	0x7F4004	0x7F4804		
FPREND0	0x7F4008	0x7F4808		
FPRCTRL1	0x7F4010	0x7F4810		
FPRST1	0x7F4014	0x7F4814		
FPREND1	0x7F4018	0x7F4818		
FPRCTRL2	0x7F4020	0x7F4820		
FPRST2	0x7F4024	0x7F4824		
FPREND2	0x7F4028	0x7F4828		
FPRCTRL3	0x7F4030	0x7F4830		
FPRST3	0x7F4034	0x7F4834		
FPREND3	0x7F4038	0x7F4838		
FPRCTRL4	0x7F4040	0x7F4840		
FPRST4	0x7F4044	0x7F4844		
FPREND4	0x7F4048	0x7F4848		
FPRCTRL5	0x7F4050	0x7F4850		
FPRST5	0x7F4054	0x7F4854		
FPREND5	0x7F4058	0x7F4858		
FPRCTRL6	0x7F4060	0x7F4860		
FPRST6	0x7F4064	0x7F4864		
FPREND6	0x7F4068	0x7F4868		
FPRCTRL7	0x7F4070	0x7F4870		
FPRST7	0x7F4074	0x7F4874		
FPREND7	0x7F4078	0x7F4878		
FIRT	0x7F4080	0x7F4880		该配置字用于使能不可变的可信根区域模式。
FSECDBG	0x7F4090	0x7F4890		该配置字用于使能安全调试模式。
FTPED	0x7F40A0	0x7F48A0		该配置字用于永久禁止全片擦除和外部编程。
FEPUCB	0x7F40B0	0x7F48B0		向该配置字中写入 0x84C1F396 将禁止对 UCB 区域进行擦除操作。
FWPUCB	0x7F40C0	0x7F48C0		向该配置字中写入 0x5B9B12E4 将禁止对 UCB 区域进行编程操作。
FBOOT	0x7F40D0	0x7F48D0		该配置字用于选择单/双引导模式。
FCP	0x7FB000	0x7FB800	UCA2(1)	该配置字用于使能闪存保护。
FICD	0x7FB010	0x7FB810		调试器配置寄存器。
FDEVOPT	0x7FB020	0x7FB820		该配置字用于配置一些外设的功能。
WDT	0x7FB030	0x7FB830		该配置字用于控制看门狗定时器。

注：

1. 在双引导模式下工作时，双引导器件具有两个 UCA 页：UCA1 对应分区 1，UCA2 对应分区 2。

1.5. 闪存保护

用户配置 Ax（UCAx）和用户配置 B（UCB）闪存区域中的配置字可用于限制对整个闪存或指定存储器区域的闪存操作。有关详细说明，请参见器件数据手册。

即使禁止对闪存区域进行读操作，NVM 控制器也可以计算 CRC-32 校验和，除非同时禁止 CRC 操作。

1.5.1. 受保护区域

在闪存中，最多可创建 8 个受保护区域。起始地址、结束地址和区域类型由 UCB 存储区中的配置字定义。通过 NVM 模块，可以限制对受保护区域的读操作、写操作、擦除操作（OTP）或 CRC 计算。

1.5.2. 用户配置 Ax 和用户配置 B 区域保护

UCAx 中的 FCP 配置字包含 WPUCA 位，可用于禁止对 UCAx 区域进行页擦除操作和页写操作。当进行全片擦除时，始终会擦除 UCAx 区域。

如果使用特殊密钥对 UCB 区域的两个配置字进行编程，则会禁止对 UCB 区域进行擦除操作或写操作。如果将 FEPUCB 编程为 0x84C1F396，则会永久禁止通过全片擦除和页擦除 NVM 命令擦除 UCB 页的所有功能。如果将 FWPUCB 编程为 0x5B9B12E4，则会永久禁止向 UCB 页写入更多数据的所有功能。FEPUCB 擦除保护和 FWPUCB 写保护适用于 ICSP 和运行时自编程代码执行。如果同时激活 UCB 编程和擦除保护，则会永久阻止 UCB 编程和擦除操作，并且无法停用保护。

1.5.3. 整个闪存保护

UCAx 区域中的 FCP 配置字可用于保护整个闪存，使其免受 ICSP 回读（CP 位）和 NVM 模块的 CRC 计算（CRC 位）的影响。

UCB 区域中的 FPED 配置字包含 PED 位，用于禁止对整个器件闪存进行全片擦除操作、页擦除操作、行写操作和四字写操作。使能该功能（PED 位 = 0）时，将限制在 ICSP 模式下对用户闪存、用户 OTP 和 UCA/UCB 配置页的所有修改。运行时自编程代码执行不受 PED 的影响。

当该编程或擦除保护功能（PED 位 = 0）与 UCB 擦除保护（FEPUCB 配置字为 0x84C1F396）一起使用时，一旦激活将永久阻止对整个闪存进行 ICSP 编程和擦除操作，并且无法停用保护。

1.6. 编程 FBOOT 配置寄存器

配置寄存器和用户 OTP 应在代码存储区之前进行编程。必须对 FBOOT 配置寄存器（位于地址 0x7F40D0 处）进行编程，以将器件配置为双分区闪存模式之一。BTMODE[1:0]位不能写为 00（保留）或 11（单分区闪存）。必须通过擦除 FBOOT 寄存器来设置单分区闪存模式。

1.7. 双分区闪存编程注意事项

dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件支持一个单分区闪存模式和两个双分区闪存模式。双分区闪存模式允许为器件烧写两个单独的应用程序，以方便自举；还允许在运行时烧写一个应用程序而不会导致 CPU 停顿。

器件的引导模式由 FBOOT 配置寄存器中的 BTMODE[1:0]位确定（见表 1-8）。器件将在复位时自动检查 FBOOT 并确定适当的引导模式。

表 1-8. 引导模式选择

FBOOT[1:0]	引导模式
11	单分区闪存模式（默认）
10	双分区闪存模式
01	受保护双分区闪存模式
00	保留

受保护双分区闪存模式可阻止对分区 1 执行运行时编程和擦除功能。ICSP 模式不受影响。

1.7.1. 双分区存储器构成

在双分区闪存模式下，器件的存储器被均分为两个物理部分，分别称作分区 1 和分区 2。其中的每个分区都包含自己的程序存储区和配置字。在程序执行期间，仅执行其中一个分区（即活动分区）中的代码。另一个分区（即非活动分区）不能用于执行，但可以编程。

活动分区始终映射到程序地址 0x800000，而非活动分区始终映射到程序地址 0xC00000。即使用户使代码在活动分区和非活动分区之间切换，活动分区的地址仍将为 0x800000，非活动分区的地址仍将为 0xC00000。

引导序列配置字（FBTSEQ）确定复位后是分区 1 还是分区 2 处于活动状态。如果器件在双分区模式下运行，则引导序列号较低的分区将作为活动分区（单分区模式下不使用 FBTSEQ）。可通过重新编程引导序列号使两个分区在活动分区和非活动分区之间交换，但活动分区在执行器件复位之前不会发生变化。如果两个分区的引导序列号相同，或者两个引导序列号均已损坏，则器件将使用分区 1 作为活动分区。如果只有一个引导序列号损坏，则器件将使用引导序列号未损坏的分区作为活动分区。

用户还可使用 BOOTSWP 指令在运行时更改哪个分区处于活动状态。BOOTSWP 指令在使用前必须先使能（位于 FICD 配置字中）。BOOTSWP 指令不会影响复位后哪个分区将是活动分区。 图 1-2 分别显示了如何在 FBTSEQ 重新编程和 BOOTSWP 执行期间使分区 1 和分区 2 在活动分区和非活动分区之间交换。

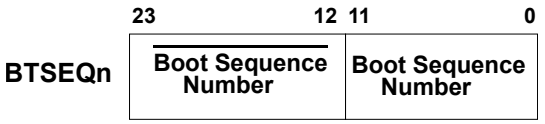
P2ACTIV（NVMCON[10]）位可用于确定哪个物理分区是活动分区。如果 P2ACTIV = 1，则分区 2 是活动分区；如果 P2ACTIV = 0，则分区 1 是活动分区。

1.7.2. 双引导序列号

序列号存储在用户闪存活动分区与非活动分区中最后一行的旧存储位置。

当配置为双引导模式时，复位后，NVM 控制器将读取 BTSEQn 字（即，每个用户程序分区的最后 128 位闪存字）中包含的引导序列号。该控制器将比较这些序列号，然后将序列号最小的分区映射到活动空间，并将其他分区映射到非活动空间。当前映射由 NVMCON.P2ACTIV 位指示。引导序列号为 12 位无符号值。BTSEQn 字格式由两个 12 位值组成，其中 BTSEQn[11:0]包含序列号真值，BTSEQn[23:12]包含其取反值，如图 1-1 所示。

图 1-1. 引导序列号字



BTSEQn[127:24]保留。如果出现以下任一情况，则将引导序列号字视为无效：

- BTSEQn[11:0] != ~BTSEQn[23:12]
  – 或 –
- 引导序列号字读操作导致纠错码（Error Correcting Code，ECC）双位错误检测（Double Error Detection，DED）错误。

如果 BTSEQn 字无效，则将为对应分区的引导序列号分配值 0xFFF。这是引导序列号的最大可能值。未编程的字将产生无效引导序列号。确定每个分区的引导序列号后，NVM 控制器会比较这些引导序列号，然后将引导序列号最小的分区映射到活动空间，并将其他分区映射到非活动空间。如果这些引导序列号相等，则将分区 1 映射到活动空间。在配置为双引导模式的器件中重新刷写或更新固件时，编程器或任务模式固件会将非活动分区引导序列号编程为小于活动分区引导序列号的值。后续复位后，器件将执行新固件。

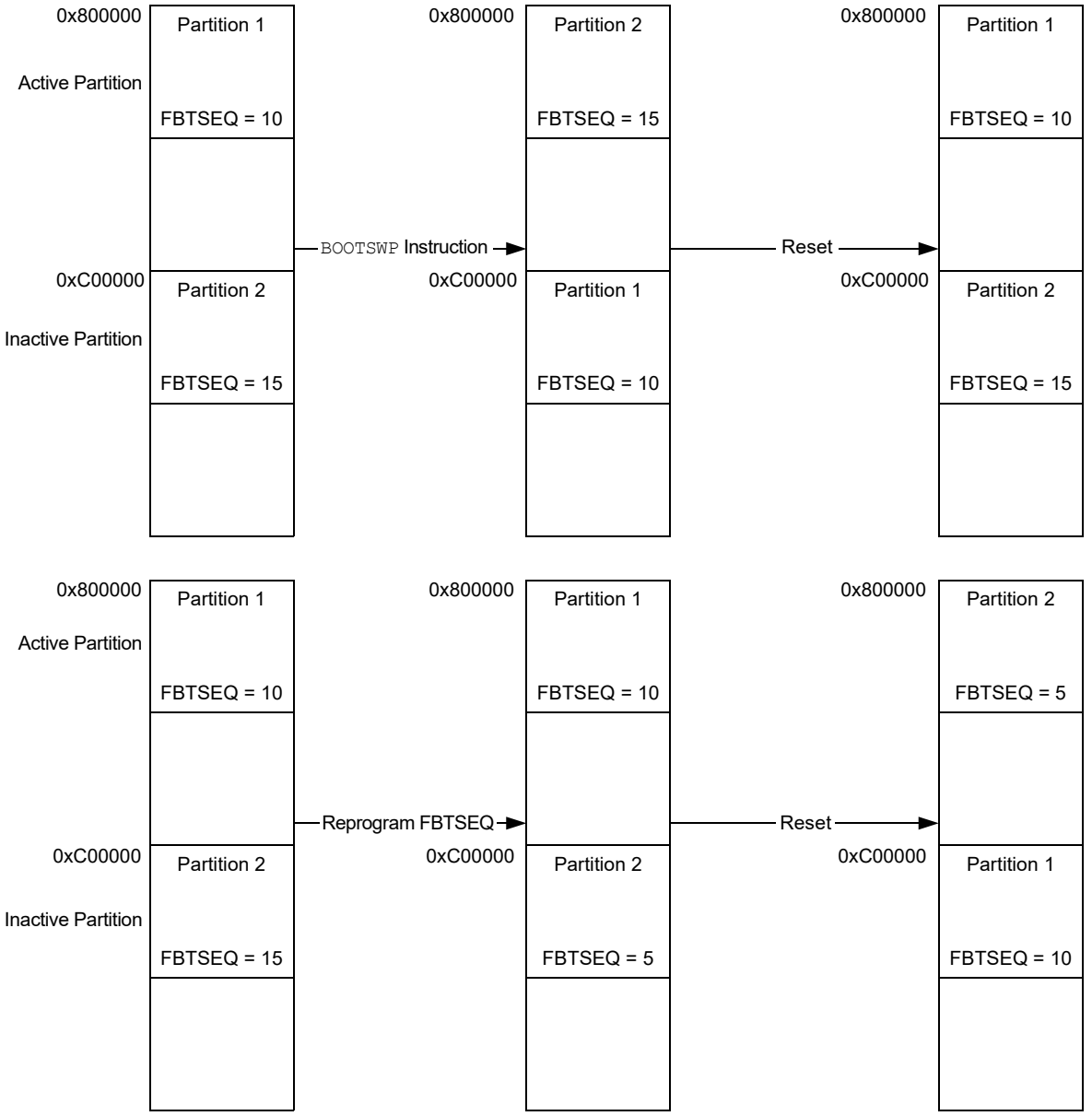
1.7.3. 双分区闪存的擦除操作

dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件支持以下三种擦除操作：全片擦除、非活动分区擦除和页擦除。

全片擦除操作将擦除所有用户存储区，包括闪存配置字和 FBOOT 配置寄存器。这会在复位后将器件的引导模式恢复为其默认的单分区模式。

可以在运行时从活动分区执行非活动分区擦除操作。这将擦除非活动分区中的所有用户存储区和闪存配置字。仅当器件处于双分区模式之一时，非活动分区擦除命令才起作用。

图 1-2. 分区 1/2 与活动/非活动分区之间的关系



1.7.4. 双分区配置字

在双分区模式下，每个分区都有自己的一组闪存配置字。活动分区中的整组配置寄存器用于确定器件的配置。在通过 FBTSEQ 将非活动分区编程为活动分区并发生复位之前，不使用非活动分区中的配置字。BOOTSWP 指令不会根据新活动分区的配置字来更改器件的有效配置。

1.7.5. 在双分区模式下编程

要对具有双分区闪存的 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 器件进行编程，需遵循以下步骤：

1. 对用户程序存储区执行全片擦除：
    1. 进入 ICSP 模式
    2. 全片擦除
    3. 退出 ICSP 模式
2. 进入 ICSP 模式，以便程序存储区地址映射以已知状态开始：
    a. 进入 ICSP 模式
3. 对 UCA1、UCA2 和 UCB 执行页擦除：
    a. 全片擦除
    b. 退出 ICSP 模式
4. 编程 FBOOT 备份寄存器位置，然后将 FBOOT 寄存器编程为双分区模式之一：
    a. 进入 ICSP 模式
    b. 编程 FBOOT_BACKUP
    c. 退出 ICSP 模式
    d. 进入 ICSP 模式
    e. 编程 FBOOT
    f. 退出 ICSP 模式
5. 发出器件复位。这会将存储区分为两个分区。
6. 进入 ICSP 模式：
    a. 将第一个应用程序及其引导序列号烧写到地址 0x800000 处的活动分区中
    b. 将第二个应用程序及其引导序列号烧写到地址 0xC00000 处的非活动分区中
    c. 写入用户配置 A1 备份存储区
    d. 写入用户配置 A2 备份存储区
    e. 写入用户配置 B 备份存储区
    f. 写入用户配置 A1 主存储区
    g. 写入用户配置 A2 主存储区
    h. 写入用户配置 B 主存储区
    i. 退出 ICSP 模式

1.8. 闪存操作时序

表 1-9 列出了闪存操作的最长时间。

表 1-9. 闪存操作的最长时间

闪存操作	最长时间
全片擦除（未定义永久区域）	80 ms
全片擦除（已定义永久区域）	20 ms x (可擦除闪存页数) + 40 ms
页擦除	20 ms
四字编程	15 µs
行编程	500 µs

2. 在线串行编程（ICSP）

在在线串行编程（ICSP）模式下，可通过 PGECx/PGEDx 引脚移入和执行 CPU 指令以及读取 VISI 寄存器内容。PGECx 为串行时钟信号，PGEDx 为串行数据信号。

2.1. ICSP 所需的连接

对于 ICSP 操作，必须使用封装中提供的所有电源引脚和接地引脚为器件供电。器件内部电路由内核稳压器（降压转换器）供电。降压转换器需要两个外部元件：10 µH 电感和 4.7 µF 陶瓷电容，这两个元件应连接到 V_{DDCORE} 和 L_X 引脚，如图 2-1 所示。除了电源引脚之外，编程接口还包括 $\overline{MCLR}$（复位）引脚和一对 ICSP 编程引脚（PGEDx/PGECx）。所有 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 器件都具有 3 对独立的编程引脚，分别标记为 PGEC1/PGED1、PGEC2/PGED2 和 PGEC3/PGED3。

图 2-1. 降压转换器元件连接

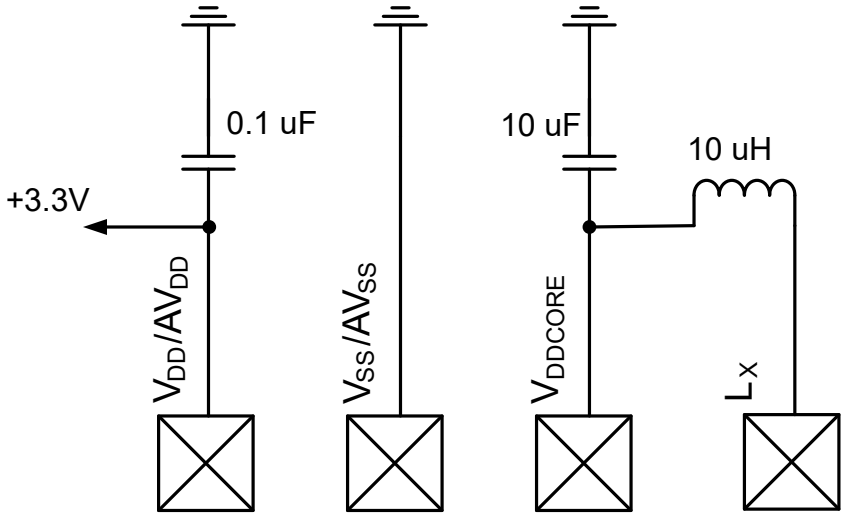


表 2-1 详细说明了所需的连接。

表 2-1. 用于编程的引脚

引脚名称	引脚类型	说明
V_{DD}/AV_{DD}	电源	电源。必须处于 3.0V-3.6V 范围内，并且必须连接所有电源引脚。这些引脚应连接 0.1 µF 的旁路电容。
V_{SS}/AV_{SS}	电源	地。必须连接所有接地引脚。
L_X	降压转换器	必须在 Lx 引脚与 V_{DDCORE} 引脚之间连接 10 µH 的电感。
V_{DDCORE}	降压转换器	应在 V_{DDCORE} 引脚与地之间连接 10 µF 的电容，并且必须在 L_X 引脚与 V_{DDCORE} 引脚之间连接 10 µH 的电感。
$\overline{MCLR}$	输入	目标器件复位/编程使能。
PGECx	输入	ICSP™编程时钟，其中 x 为编程引脚对编号。
PGEDx	输入/输出	ICSP 编程数据，其中 x 为编程引脚对编号。

表 2-2 中规定的 CMOS 逻辑阈值适用于 $\overline{MCLR}$ 以及 PGECx/PGEDx 信号。

表 2-2. 编程接口信号的电气规范

参数	最小值	最大值	说明
V_{IL}	V_{SS}	0.2 * V_{DD}	输入低电平电压

表 2-2. 编程接口信号的电气规范 （续）

参数	最小值	最大值	说明
V_{IH}	0.8 * V_{DD}	V_{DD}	输入高电平电压

对于将数据随时钟移入目标器件的所有 ICSP 操作，PGEDx 输出状态必须在 PGECx 时钟上升沿之前有效且稳定。所有 PGEDx 数据必须均为最低有效位（Least Significant bit，LSb）在前。

图 2-2、图 2-3 和表 2-3 规定了 ICSP 接口信号的时序。

图 2-2. ICSP™接口信号时序参数（编程器驱动 PGEDx 引脚）

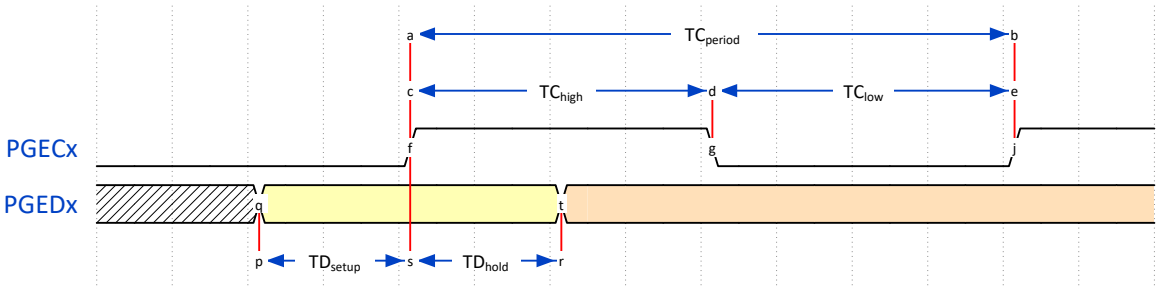


图 2-3. ICSP™接口信号时序参数（目标器件驱动 PGEDx 引脚）

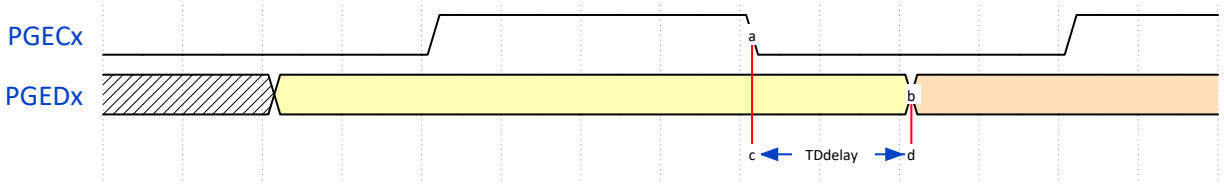


表 2-3. ICSP™接口信号时序参数

参数	最小值	最大值	说明
TD_{setup}	20 ns	—	PGECx 上升沿之前的最短 PGEDx 数据建立时间。
TD_{hold}	1 ns	—	PGECx 上升沿之后的最短 PGEDx 数据保持时间。
TC_{period}	60 ns	—	最小 PGECx 时钟周期。
TC_{low} 或 TC_{high}	20 ns	—	最小 PGECx 时钟低电平或高电平脉冲宽度。
TD_{delay}	—	20 ns	PGECx 下降沿之后的最大数据更新延时。

图 2-4 至图 2-9 提供了 dsPIC33AK512MC510 和 dsPIC33AK512MPS512 器件不同封装的引脚图。有关引脚的详细说明，可参见器件数据手册。

图 2-4. 48 引脚 VQFN/TQFP

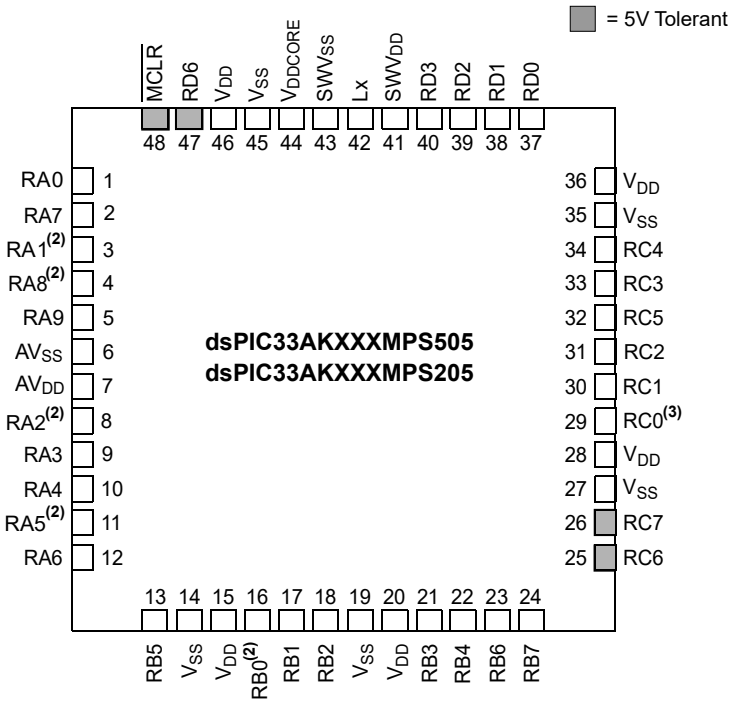


图 2-5. 64 引脚 VQFN/TQFP

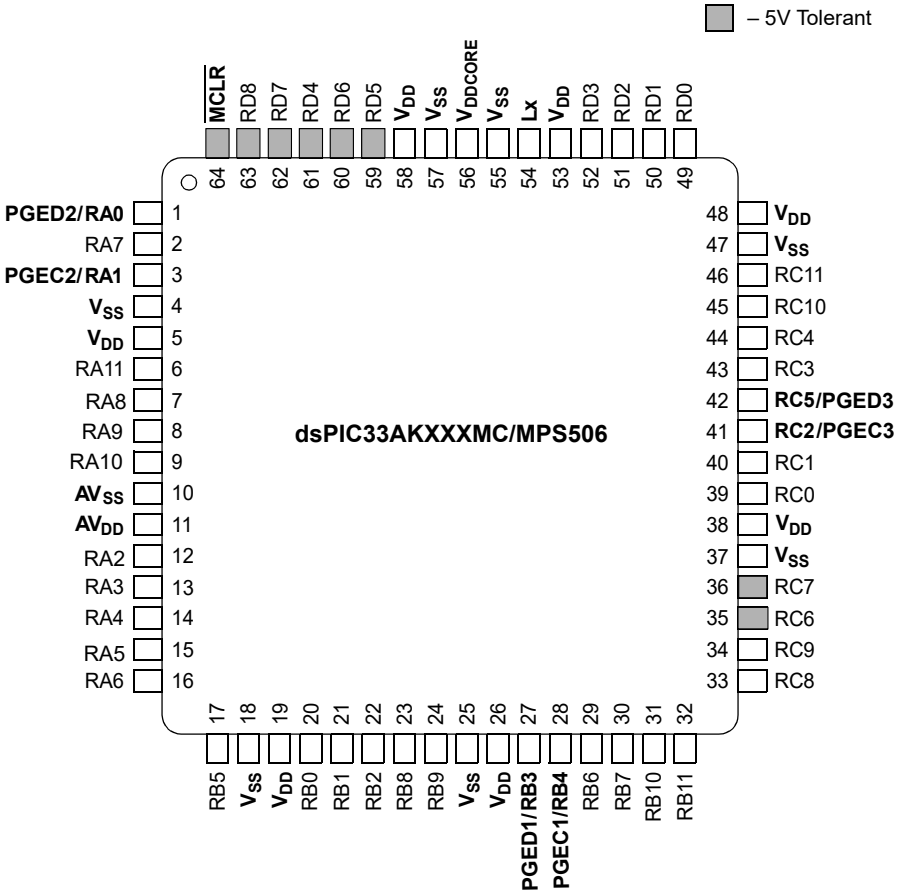


图 2-6. 80 引脚 TQFP

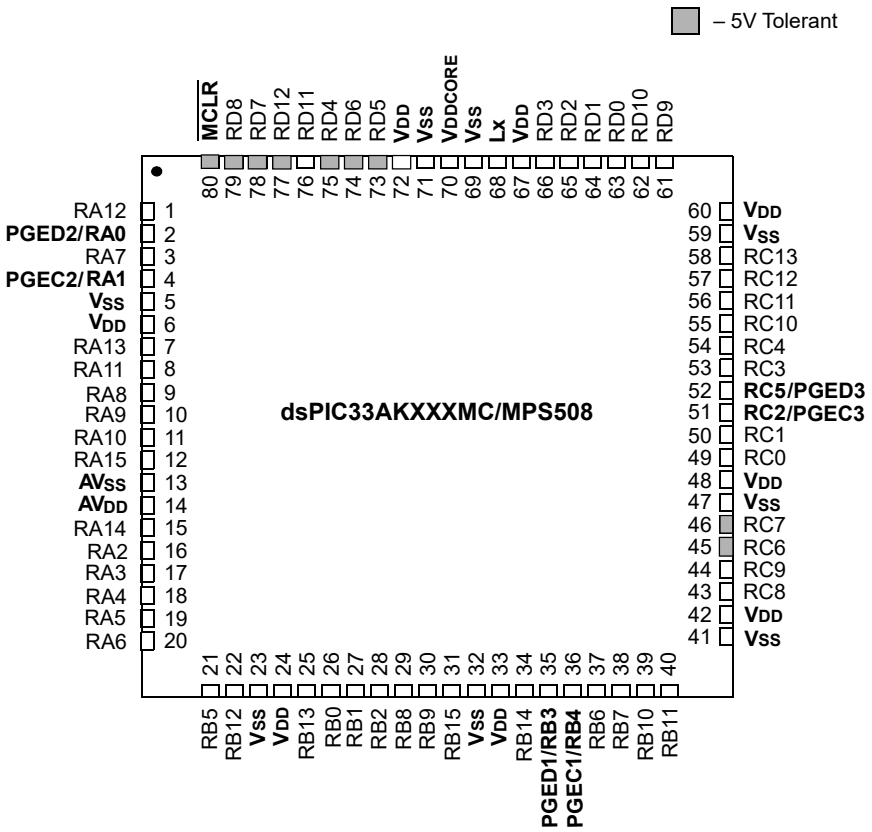


图 2-7. 100 引脚 TQFP

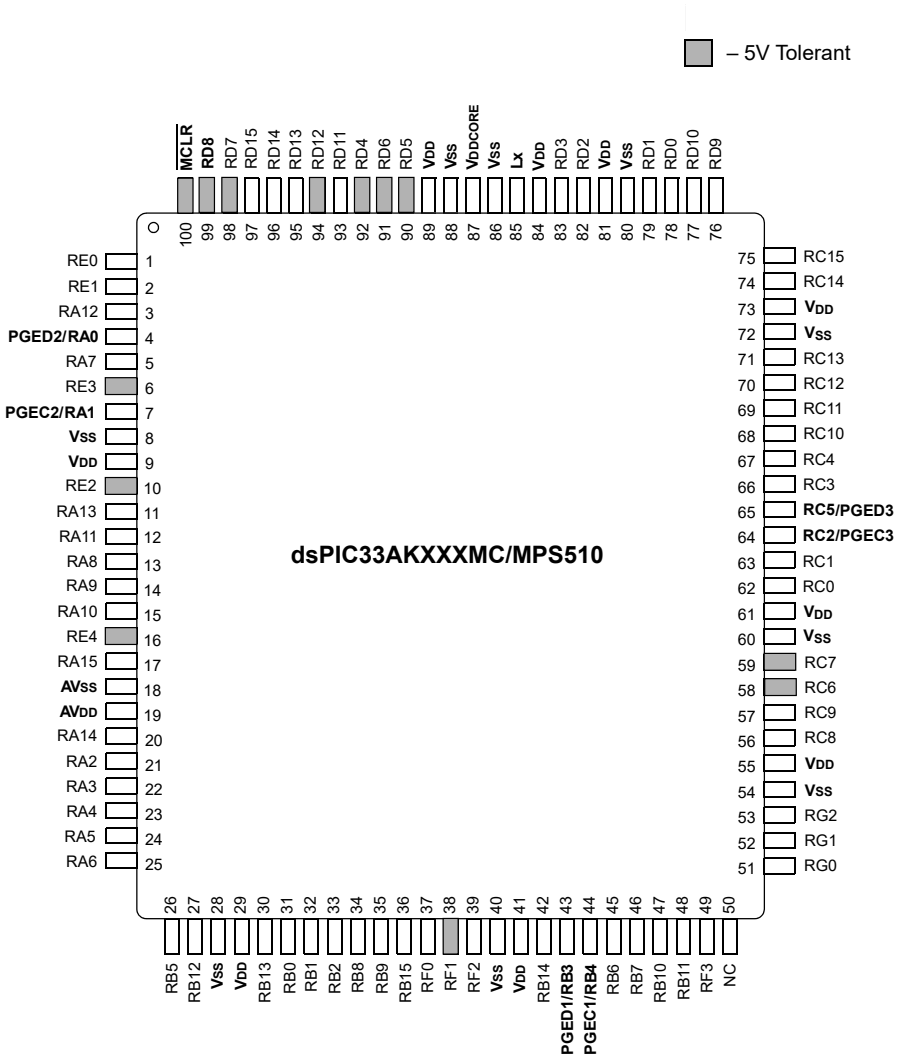


图 2-8. 128 引脚 TQFP

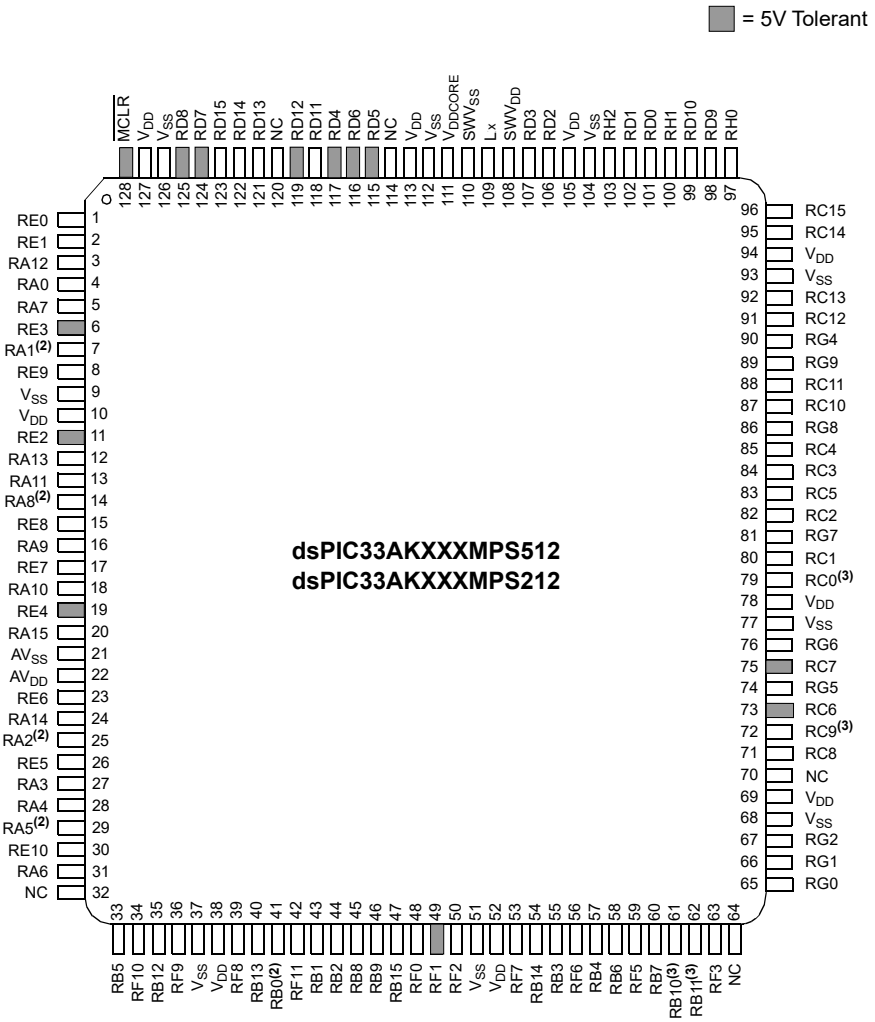
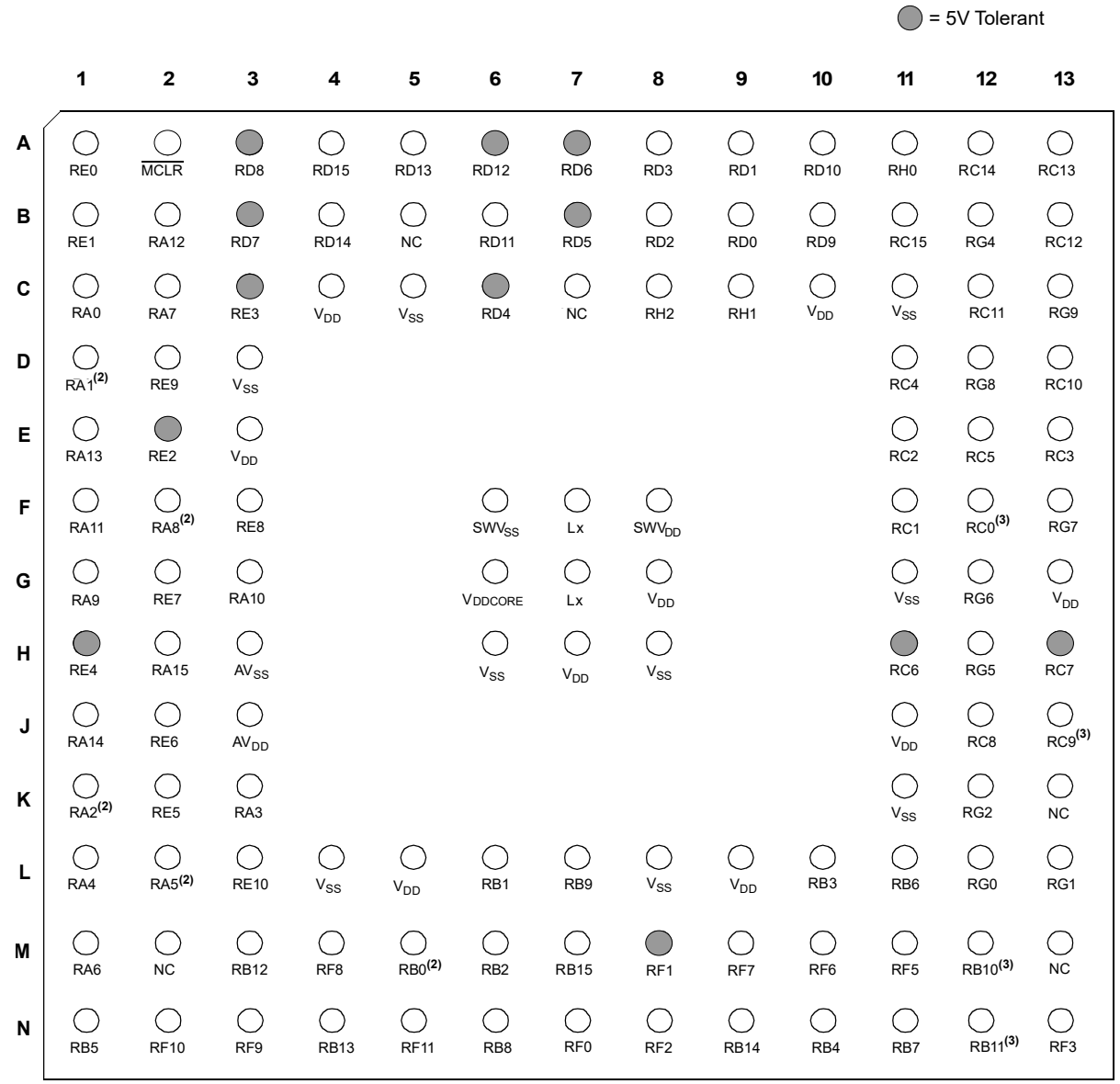


图 2-9. 129 引脚 TQFP



2.2. 进入 ICSP 模式

进入 ICSP 模式需通过 $\overline{\text{MCLR}}$、PGECx 和 PGEDx 引脚上的特定序列来实现。共有三对 PGECx/PGEDx 引脚可供选择：PGEC1/PGED1、PGEC2/PGED2 或 PGEC3/PGED3。

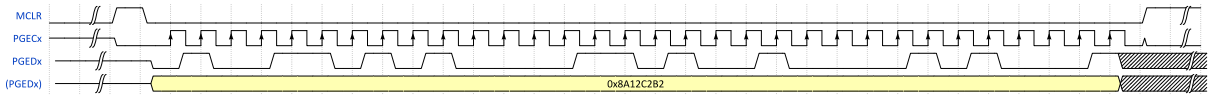
要进入 ICSP 模式，必须遵循以下步骤：

1. 为器件供电。V_{DD} 应在所有 ICSP 操作过程中保持稳定。由于无需掉电再上电，因此可为外部编程工具与目标芯片或电路单独供电，但是需要共用参考地 V_{SS}。
2. 将 $\overline{\text{MCLR}}$ 驱动为低电平。
3. 开始将 PGECx 和 PGEDx 驱动为低电平。PGEDx 在该阶段状态“无关”，但为了简化后续操作，应将其设为输出并驱动为低电平。
4. 至少等待 1 ms（最长持续时间不限）。
5. 采用脉冲方式将 $\overline{\text{MCLR}}$ 先驱动为高电平，再驱动为低电平。脉冲持续时间必须至少为 20 ns，但应尽可能短。如果脉冲持续时间较长，目标器件闪存中的现有代码会开始执行。如果该用户代码中包含 RTSP

（运行时自编程）程序，则可能会修改闪存内容。在这种情况下，某些编程步骤（如验证已编程的闪存数据）可能会失败。因此，$\overline{\text{MCLR}}$ 高电平持续时间不得超过 2 µs，即便实在无法避免，也要尽量缩短 $\overline{\text{MCLR}}$ 在高电平状态的停留时间。

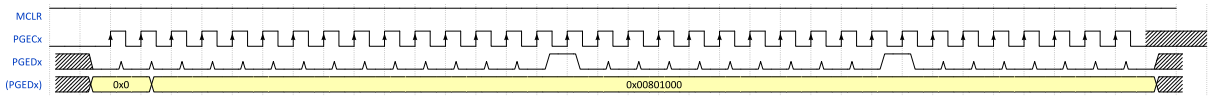
6. 产生 32 个 PGECx 时钟脉冲，同时将值 0x8A12C2B2 移至 PGEDx 引脚上（最低有效位在前）。如果在线路上解码为大尾数格式的八位字节，则将生成以下字节序列：0x4D、0x43、0x48 和 0x51。
7. 在 PGECx 时钟的第 32 个下降沿之后，将 $\overline{\text{MCLR}}$ 驱动为高电平。ICSP 信号波形如图 2-10 所示。

图 2-10. ICSP™进入序列波形（PGECx 脉冲 1 至 32）



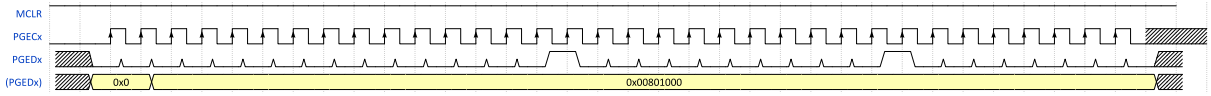
8. 至少等待 500 µs。在此期间，PGECx 必须保持低电平，但 PGEDx 状态无关。
9. 产生 34 个 PGECx 时钟脉冲以初始化内部时钟选择逻辑。这 34 个时钟的 PGEDx 应为位模式 00 后跟 32 位常量 0x00801000。该步骤的 ICSP 信号波形如图 2-11 所示。

图 2-11. ICSP™进入序列波形（PGECx 脉冲 33 至 66）



10. 产生 34 个 PGECx 时钟脉冲以设置（无意义的）CPU 复位向量目标。这 34 个时钟的 PGEDx 应为位模式 00 后跟 32 位常量 0x00801000（与步骤 9 中相同）。该步骤的 ICSP 信号波形如图 2-12 所示。

图 2-12. ICSP™进入序列波形（PGECx 脉冲 67 至 100）



编程器应使 $\overline{\text{MCLR}}$ 继续保持高电平，以便进行所有后续的 ICSP 操作。当 $\overline{\text{MCLR}}$ 拉为低电平 1 ms 或更长时间时，或者移除 V_{DD} 时，ICSP 模式将终止。

2.3. 退出 ICSP 模式

要退出 ICSP 模式并开始执行内部代码，应将 $\overline{\text{MCLR}}$ 驱动为低电平，将 PGECx 和 PGEDx 置于三态，将 $\overline{\text{MCLR}}$ 保持低电平 1 ms，然后将 $\overline{\text{MCLR}}$ 驱动为高电平（最好改为将 $\overline{\text{MCLR}}$ 置于三态，然后通过一个连接至 V_{DD} 的板上上拉电阻来决定此时是否需要执行用户代码）。

退出 ICSP 模式会在内部引发伪 POR 事件。因此，在 ICSP 会话期间写入的所有 SFR 都将被清除回其复位默认状态，通常包括明确记录为仅在 POR 或 BOR 类型复位时复位的 SFR。ICSP 会话期间产生的 RAM 状态将保持，直到被 RAM BIST 硬件改写、被应用程序代码执行改写或目标器件进行掉电再上电为止。

2.4. ICSP 命令

ICSP 命令代码用于通知目标器件下一项操作是什么。代码后跟一连串时钟，用于将数据移入/移出目标器件。在发出任何命令之前，必须先将器件置于 ICSP 模式（见进入 ICSP 模式一节）。

命令代码的长度固定为 2 位，需要两个 PGECx 时钟进行传输，具体根据下面的表 2-4 来实现。

表 2-4. ICSP™命令代码

二进制命令	助记符	操作
00	CMDEXEC	将一条 32 位指令、两条 16 位指令或半条 64 位指令移入 CPU。目标器件随后执行指令，同时接收下一个命令及其数据。

表 2-4. ICSP™命令代码 （续）

二进制命令	助记符	操作
01	CMDRD	将 PGEDx 引脚设置为输出，将 32 位数据从 VISI 寄存器移至该引脚上，然后将 PGEDx 恢复为输入以接收下一个命令。
10	CMDSEQWR	依次将 32 位数据装入目标器件上 W0 指向的存储器，然后将 W0 递增 4。该命令在内部将 32 位 PGEDx 移位数据分解为由目标处理器执行的 64 位 MOV.L 指令： MOV.L #PGEDx<31:0>, [W0++]
11	CMDSEQRD	返回 VISI 寄存器内容并在内部执行以下指令： MOV.L [W0++], [W8] 当 W0 初始化为源存储器地址且 W8 包含 VISI 寄存器的地址时，该命令允许以最小的开销从目标器件读取连续的存储器地址。

传入或传出目标器件的 2 位命令代码和 32 位数据始终以最低有效位在前的方式在线路上进行传输。

所有命令代码和数据位（由编程器驱动到 PGEDx 引脚）均由目标器件在 PGECx 时钟的上升沿进行锁存。因此，PGEDx 状态必须在 PGECx 上升沿的至少一个建立时间（TD_{setup}）之前有效且稳定，并在 PGECx 上升后至少一个保持时间（TD_{hold}）内保持不变。

从目标器件读取数据时，目标器件在 PGECx 时钟（仍由编程器产生）的下降沿在 PGEDx 上输出/更改每个数据位（共 32 个）。由于数据在下一个下降沿之前保持不变，因此编程器应在时钟上升沿或之后，但在产生另一个 PGECx 下降沿之前对数据进行采样。

2.4.1. ICSP 指令执行（CMDEXEC）

指令执行序列（CMDEXEC）用于将一条 32 位指令、两条 16 位指令或半条 64 位指令移入目标 CPU。

指令执行序列如下：

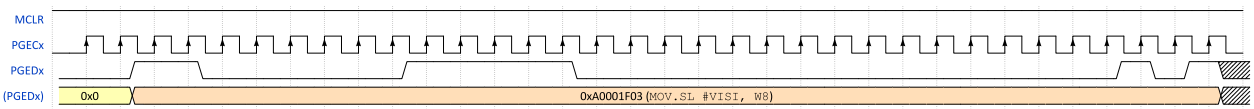
1. 编程器将 00 位移入器件。
2. 之后，编程器将 32 位指令操作码数据发送到器件。每个 PGEDx 位均由器件在 PGECx 脉冲的上升沿进行锁存。CPU 在接下来的 5-10 个 PGECx 时钟内执行指令，同时发送下一个 ICSP 命令。

提供 16 位指令时，要执行的第一条 16 位指令必须与低 16 位对齐，并且后跟 32 位数据传输高半部分的另一条 16 位指令。如果只提供一条 16 位指令，则必须与低 16 位对齐并经过零扩展至 32 位（高 16 位编码为 NEOP）。

执行 64 位指令时，第一个 CMDEXEC 序列将装载第 1 个指令字（操作码的 bit 0-31）。必须发出第二个 CMDEXEC 序列以装载第 2 个指令字（操作码的 bit 32-63），从而允许 CPU 开始执行该指令。对于 64 位指令，在发出第一个 CMDEXEC 序列之后必须立即发出第二个 CMDEXEC 序列，中间不能有任何其他 ICSP 命令；否则，CPU 的指令流将被破坏，并且可能产生非法操作码复位。

图 2-13 给出了将操作码为 0xA0001F03 的“MOV.SL #VISI, W8”指令（将 VISI 寄存器地址装入 W8 寄存器）移入 CPU 的示例。

图 2-13. 将操作码为 0xA0001F03 的 MOV.SL #VISI, W8 移入 CPU 的 CMDEXEC 序列



2.4.2. VISI 寄存器的 ICSP 读（CMDRD）

该命令用于将 VISI 寄存器的内容移至 PGEDx 引脚上，以供编程器捕捉。与写入 VISI 的指令执行序列（CMDEXEC）结合使用时，可用于从目标器件读取状态寄存器和存储器的内容。

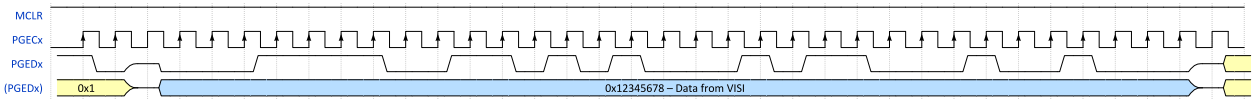
VISI 寄存器读序列如下：

1. 编程器将 01 位移入器件。由于所有数据都遵循最低有效位在前的顺序，因此 1 位必须在 0 位之前出现在 PGEDx 线路上。
2. 编程器生成一个空闲 PGECx 时钟以适应 PGEDx 引脚方向的切换；编程器应在该时钟的下降沿之前将 PGEDx 置于三态，因为目标器件将从此时开始驱动 VISI 的 bit 0。
3. 编程器移入器件中的 32 位 VISI 寄存器内容。VISI 寄存器内容的最低有效位在前。器件在 PGECx 脉冲的下降沿更新每个数据位。为了适应时钟进入器件的传播延时和返回下一位的输出延时，编程器应在 PGECx 上升沿或之后从器件采样数据。
4. 编程器生成一个空闲 PGECx 时钟以适应 PGEDx 引脚方向切换回输入；器件在完成 VISI bit 31 传输的下降沿将 PGEDx 置于三态，因此编程器可在该时钟期间的任何时候或结束时开始驱动 PGEDx。

所有写入 VISI 寄存器的指令执行序列（CMDEXEC）都要等到属于下一个命令的几个时钟出现之后才能完成执行。因此，如果后面紧跟 VISI 寄存器读序列（CMDRD），则任何写入 VISI 的操作都不会生效。如果在 VISI 寄存器读序列（CMDRD）之前放置一个立即写入 VISI 的操作，仍会正常执行，只是会延迟返回数据（将包含旧 VISI 数据）。若要接收通过前一 CMDEXEC 序列移动的最新 VISI 内容，应在 CMDRD 序列之前执行第二个 CMDEXEC（任何操作码）。

图 2-14 给出了将 0x12345678 数据从 VISI 寄存器移出至编程器的 CMDRD 序列示例。

图 2-14. 将数据从 VISI 寄存器移出至编程器的 CMDRD 数据序列



2.4.3. ICSP 连续写（CMDSEQWR）

ICSP 连续写序列（CMDSEQWR）允许通过间接指针访问 RAM 或器件寄存器来快速写入 32 位数据。当该序列完成时，将在内部生成一条 64 位 MOV.L 指令，对全部 32 位移位数据进行编码并传递给 CPU 执行。结果相当于需要发出两个 CMDEXEC 序列以执行如下 64 位指令：

```
MOV.L #PGEDx<31:0>, [W0++]
```

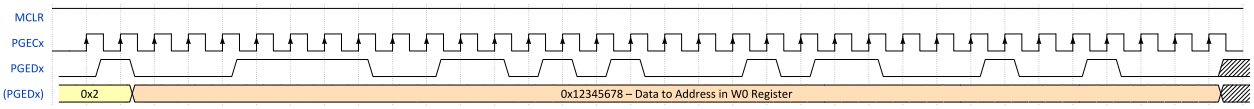
该指令将提供的 32 位数据写入 W0 寄存器中按 4 字节对齐的地址指向的存储单元，然后将 W0 指针递增 4。通过将 W0 预先初始化为 RAM 或寄存器地址，只需执行几个连续的 CMDSEQWR 序列即可写入递增单元，这种方法的开销最小。一般来说，编程器对按 4 字节对齐的连续地址进行的所有四字节或更多字节的数据写操作都应该使用 CMDSEQWR。

连续写序列如下：

1. 编程器将 10 位移入器件。由于所有数据都遵循最低有效位在前的顺序，因此 0 位必须在 1 位之前出现在 PGEDx 线路上。
2. 之后，编程器将 32 位 RAM 或 SFR 数据发送到器件。每个 PGEDx 位均由器件在 PGECx 脉冲的上升沿进行锁存。CPU 在接下来的 5-10 个 PGECx 时钟内执行生成的 MOV.L 指令，同时发送下一个 ICSP 命令。

图 2-15 给出了编程器将 0x12345678 写入 W0 寄存器中的地址的 CMDSEQWR 序列示例。

图 2-15. 编程器将数据写入 W0 寄存器中的地址的 CMDSEQWR 序列



2.4.4. ICSP 连续读（CMDSEQRD）

该连续读序列用于将 32 位 VISI 寄存器内容移出至 PGEDx 引脚上（与 CMDRD 一样），但还会在内部执行以下指令：

```
MOV.L [W0++], [W8]
```

执行该 MOV.L 指令之前会对现有的 VISI 内容进行缓冲，因此 VISI 的所有 32 位内容都会原封不动地出现在 PGEDx 上，与执行该指令之前一样。但是，该指令通常用于从 SFR、RAM 或程序空间向 VISI 重载更多数据，并且在所有位都移出时完成执行。因此，CMDSEQRD 针对在循环中使用进行了优化，以便将多个连续的存储器内容从目标器件传输到编程器中。

由于生成的 MOV.L 指令引用 W0 和 W8，因此该命令前面必须有 CMDEXEC 命令，以便将 VISI 寄存器的地址装入 W8 并将目标器件上的读地址装入 W0：

```
MOV.SL #VISI, W8

MOV.SL #ReadAddress<23:0>, W0
```

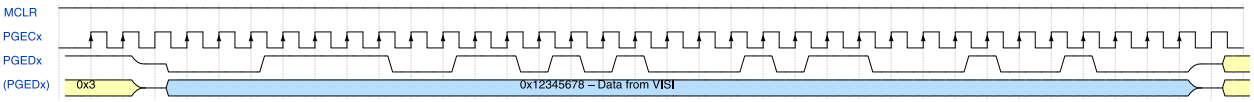
连续读序列如下：

1. 编程器将 11 位移入器件。
2. 编程器生成一个空闲 PGECx 时钟以适应 PGEDx 引脚方向的切换；编程器应在该时钟的下降沿之前将 PGEDx 置于三态，因为目标器件将从此时开始驱动 VISI 的 bit 0。
3. 编程器移入器件中的 32 位 VISI 寄存器内容。VISI 寄存器内容的最低有效位在前。器件在 PGECx 脉冲的下降沿更新每个数据位。为了适应时钟进入器件的传播延时和返回下一位的输出延时，编程器应在 PGECx 上升沿或之后从器件采样数据。
4. 编程器生成一个空闲 PGECx 时钟以适应 PGEDx 引脚方向切换回输入；器件在完成 VISI bit 31 传输的下降沿将 PGEDx 置于三态，因此编程器可在该时钟期间的任何时候或结束时开始驱动 PGEDx。

与 CMDEXEC 和 CMDSEQWR 不同的是，CPU 会先将内部生成的 MOV.L [W0++], [W8]指令执行完毕再移出 VISI 最终（经缓冲）的 bit 31。这样可以连续重发 CMDSEQRD 以返回存储器中的下一个 32 位内容，即在写入 VISI 之后无需提供额外的时钟。

图 2-16 给出了编程器从 VISI 接收 0x12345678 数据的 CMDSEQRD 示例。

图 2-16. 编程器从 VISI 接收数据的 CMDSEQRD 示例



3. 编程流程

dsPIC33AK512MC510 和 dsPIC33AK512MPS512 系列器件的编程流程包含以下步骤：

1. 进入 ICSP 模式
2. 全片擦除整个闪存：
   a. 进入 ICSP 模式
   b. 全片擦除
   c. 退出 ICSP 模式
3. 编程 FBOOT_BACKUP 和 FBOOT：
   a. 进入 ICSP 模式
   b. 编程 FBOOT_BACKUP
   c. 退出 ICSP 模式
   d. 进入 ICSP 模式
   e. 编程 FBOOT
   f. 退出 ICSP 模式
4. 写入代码存储区
5. 回读 CRC 以验证代码存储区
6. 写入用户配置 A1 备份存储区
   a. 回读并验证
   b. 写入用户配置 B 备份存储区
   c. 回读并验证
7. 写入用户配置 A1 主存储区：
   a. 回读并验证
   b. 写入用户配置 B 主存储区
   c. 回读并验证
8. 退出 ICSP 模式

如果在 UCAx 和 UCB 区域的配置字中禁止编程或擦除操作，则闪存擦除或写入过程将失败，即对受保护区域不起作用。要擦除所有闪存，必须通过擦除 UCB 并退出/重新进入 ICSP 模式来禁止 UCB 区域中定义的受保护区域。

按照以下步骤解除代码保护：

1. 进入 ICSP 模式
2. 对 UCAx 存储区（包括 UCB 存储区的代码保护位）进行全片擦除
3. 退出 ICSP 模式以重载配置字
4. 进入 ICSP 模式
5. 对 UCB 存储区（包括代码保护位）进行全片擦除。此外，该过程还将擦除整个代码闪存
6. 退出 ICSP 模式以重载配置字

当代码闪存未受到编程或擦除保护时，全片擦除将执行并行擦除操作并快速完成（有关全片擦除时间，请参见数据手册）。相反，当在代码闪存中定义了任何具有擦除、写入或 OTP 保护的区域时，仍将接受全片

擦除命令。但是，NVM 控制器会在内部将全片擦除作为自迭代连续页擦除来执行，这会导致全片擦除时间明显延长。

3.1. 全片擦除

该算法用于擦除用户闪存以及 UCAx 和 UCB 用户配置页。用户 OTP 会被保持，并且永远不会被擦除。如果在 UCAx 和 UCB 页内容中使能了硬件写保护功能，则 UCAx 和 UCB 页以及 UCB 中的安全区域描述符定义的任意闪存地址范围也可能会被永久保持，而不受全片擦除的影响。

表 3-1. 全片擦除算法

ICSP™序列	ICSP 命令代码	数据/操作码	执行的指令
步骤 1：初始化指针。将 VISI 地址传送至 W8，将 NVMCON 地址传送至 W9。			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
步骤 2：将 0x400E 写入 NVMCON 以设置为进行全片擦除 NVM 操作。			
CMDEXEC	00	0x8A9004E1	MOVS.W #0x400E, [W9]
步骤 3：将 WR 位（NVMCON[15]）置 1 以启动 NVM 操作并将 NVMCON 的内容传送至 VISI。			
CMDEXEC	00	0x8E9004E1	MOVS.W #0xC00E, [W9]
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
步骤 4：提供时钟以再次将 NVMCON 的内容传送至 VISI，以此确保上一步骤中将 NVMCON 的内容传送至 VISI 的操作成功执行，然后将 VISI 的内容移至 PGEDx 上。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
步骤 5：重复步骤 4 以轮询 WR 位（NVMCON[15]），直到其清零（表示成功执行）。			

3.2. 页擦除

该算法（表 3-2）用于擦除代码闪存的一页。擦除地址按页大小对齐，其低位部分为“无关”位。页擦除过程对用户 OTP 存储区不起作用。如果已使能 FPED 或代码保护，则页擦除将对用户配置 A 和 B 区域不起作用。与全片擦除类似，当以 UCA 页、UCB 页或通过安全描述符将页定义为硬件写保护的地址范围为目标时，页擦除命令可能不起作用。

表 3-2. 页擦除算法

ICSP™序列	ICSP 命令代码	数据/操作码	执行的指令
步骤 1：初始化指针。将 VISI 地址传送至 W8，将 NVMCON 地址传送至 W9。			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
步骤 2：将 NVMCON 地址传送至 W0。			
CMDEXEC	00	0x00000309	MOV.L W9, W0
步骤 3：将 0x4003 写入 NVMCON 以设置为进行页擦除 NVM 操作。			
CMDSEQWR	10	0x00004003	MOV.L #PGEDx<31:0>, [W0++]
步骤 4：使用目标页上的任意地址初始化 NVMADR。			
CMDSEQWR	10	擦除地址[31:0]	MOV.L #PGEDx<31:0>, [W0++]
步骤 5：将 WR 位（NVMCON[15]）置 1 以启动 NVM 操作并将 NVMCON 的内容传送至 VISI。			
CMDEXEC	00	0x8E900431	MOVS.W #0xC003, [W9]
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
步骤 6：提供时钟以再次将 NVMCON 的内容传送至 VISI，以此确保上一步骤中将 NVMCON 的内容传送至 VISI 的操作成功执行，然后将 VISI 的内容移至 PGEDx 上。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
步骤 7：重复步骤 6 以轮询 WR 位（NVMCON[15]），直到其清零（表示成功执行）为止。			

3.3. 四字编程

该算法（表 3-3）用于编程一个按 16 字节对齐的四字。目标地址的低四位在进行写操作时为“无关”位，在进行编程操作时会被强制设为零。这些器件上的闪存内置纠错码（ECC）机制。该机制会在写操作期间计算 ECC 校验和，并将计算结果与每个四字一起存储（未映射）。如果多次编程同一个闪存四字，其间没有设置擦除周期，则该 ECC 校验和会损坏。随后读取损坏的校验和所在的存储单元时，将产生 ECC 错误中断或陷阱。在擦除闪存之后，编程器最多只能对任何闪存四字存储单元（包括 OTP 区域）进行一次写操作。

表 3-3. 四字编程算法

ICSP™序列	ICSP 命令代码	数据/操作码	执行的指令
步骤 1：将 VISI 地址传送至 W8，将 NVMCON 地址传送至 W9。			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
步骤 2：使用 NVMCON 地址初始化 W0 以进行连续装载。将 WR 位置 1 的 NVMCON 值装入 W10。将 0x4001 写入 NVMCON 以设置为进行四字编程。			
CMDEXEC	00	0x00000309	MOV.L W9, W0
CMDEXEC	00	0xA8030007	MOV.SL #0xC001, W10
CMDSEQWR	10	0x00004001	MOV.L #PGEDx<31:0>, [W0++]
步骤 3：将目标地址装入 NVMADR。			
CMDSEQWR	10	目标地址[31:0]	MOV.L #PGEDx<31:0>, [W0++]
步骤 4：将数据装入 NVM 控制器数据寄存器（NVMDATA0、NVMDATA1、NVMDATA2 和 NVMDATA3）。			
CMDSEQWR	10	数据[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	数据[63:32]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	数据[95:64]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	数据[127:96]	MOV.L #PGEDx<31:0>, [W0++]
步骤 5：将 W0 恢复为 NVMCON 的地址。将 WR 位（NVMCON[15]）置 1 以启动 NVM 操作。将 NVMCON 的内容传送至 VISI。			
CMDEXEC	00	0x1F0A0309	MOV.L W9, W0
			MOV.L W10, [W0++]
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
步骤 6：将 NVMCON 的内容传送至 VISI。将 VISI 的内容移至 PGEDx 上。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
步骤 7：重复步骤 6 以轮询 WR 位（NVMCON[15]），直到其清零（表示成功执行）为止。			
步骤 8：重复步骤 3 至 7，直到所有连续的四字均已完成编程为止。			

3.4. 行编程

该算法（表 3-4）用于对一个或多个闪存行进行编程。行数据将预装入器件的 RAM。当 NVM 控制器将数据从 RAM 编程/传送至闪存时，可以将另一行数据传送至器件 RAM 中，以便进行后续的行编程操作。通过在将数据从编程器传送至器件 RAM 的同时使用 NVM 控制器将数据从器件 RAM 传送至闪存，该编程算法实现了接近 PGECx/PGEDx 引脚的原始时钟速率的最大编程吞吐量。当源数据适当对齐并且足够大以定义至少一行时，应始终使用该编程算法。

这些器件上的闪存内置纠错码（ECC）机制。该机制会在写操作期间计算 ECC 校验和，并将计算结果与数据一起存入闪存。如果多次编程闪存，则该 ECC 校验和会损坏。下次访问损坏的校验和所在的存储单元

时，可能会产生单个位 ECC 错误中断或双位 ECC 错误陷阱。在擦除闪存之后，编程器最多只能对任何闪存存储单元（包括 OTP 区域）进行一次写操作。

用户配置 Ax 和 B 区域禁止行编程，即行编程对这两个区域不起作用。

表 3-4. 行编程算法

ICSP™序列	ICSP 命令代码	数据/操作码	执行的指令
步骤 1：初始化指针。将 VISI 地址传送至 W8，将 NVMCON 地址传送至 W9。			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
步骤 2：将 W1 设置为 RAM 中的起始地址（0x4000），然后将该指针复制到 W0 中以供 CMDSEQWR 使用。使用 0x4002 初始化 NVMCON 以设置为进行行编程。			
CMDEXEC	00	0x84010003	MOV.SL #0x4000, W1
CMDEXEC	00	0x00000301	MOV.L W1, W0
CMDEXEC	00	0x8A900421	MOVS.W #0x4002, [W9]
步骤 3：将数据装入 RAM 缓冲区。			
CMDSEQWR	10	数据[31:0]	MOV.L #PGEDx<31:0>, [W0++]
步骤 4：重复步骤 3，直到所有行数据均传送至 RAM 缓冲区为止。必须总共发出 128 个 CMDSEQWR 命令，每个命令都会为递增的 RAM 存储单元生成 32 位数据。			
步骤 5：为了准备好轮询 WR 位，将 NVMCON 的内容传送至 VISI。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
步骤 6：将 NVMCON 的内容传送至 VISI。将 VISI 的内容移至 PGEDx 上。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
步骤 7：重复步骤 6 以轮询 WR 位（NVMCON[15]），直到其清零（表示成功执行上一次行编程操作）为止。			
步骤 8：将目标闪存行地址写入 NVMADR，将 RAM 源地址写入 NVMSRCADR；然后将 WR 位（NVMCON[15]）置 1 以开始硬件行编程。			
CMDEXEC	00	0x94030195	MOV.L W1, NVMSRCADR
CMDEXEC	00	0x8000C013	MOV.SL #NVMADR, W0
CMDSEQWR	10	目标地址[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDEXEC	00	0x8E900421	MOVS.W #0xC002, [W9]
步骤 9：当 NVM 控制器对前一个缓冲区进行编程时，更改 RAM 缓冲区地址以继续装载数据。地址将在 0x4000 与 0x4200 之间切换。将 RAM 缓冲区的新起始地址装入 W0。			
CMDEXEC	00	0x03014491	BTG.L W1, #9
			MOV.L W1, W0
步骤 10：重复步骤 3 至 9 以对后续的更多数据进行行编程。			
步骤 11：为了准备好针对最后一次行编程进行 WR 位轮询，将 NVMCON 传送至 VISI。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
步骤 12：将 NVMCON 的内容传送至 VISI。将 VISI 的内容移至 PGEDx 上。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
步骤 13：重复步骤 12 以轮询 WR 位（NVMCON[15]），直到其清零（表示成功执行最后一次行编程）为止。			

3.5. 读存储器

该算法（表 3-5）可用于读取任何已实现的存储空间，其中包括 DEVID、DEVREV、器件寄存器、RAM、用户闪存、用户 OTP、UCA 和 UCB 配置字以及 SFR。读地址需按 32 位对齐。

尝试读取未实现的存储器、保留地址、受代码保护的区域或受 UCB 安全区域描述符定义的持久安全限制保护的区域将返回零。

表 3-5. 读存储器算法

ICSP™序列	ICSP 命令代码	数据/操作码	执行的指令
步骤 1：将传输目标指针（W8）初始化为 VISI 寄存器的内容。			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
步骤 2：初始化指向读地址的指针。			
CMDEXEC	00	0x80000003 | （读地址[23:0] << 2）	MOV.SL #ReadAddress<23:0>, W0
步骤 3：发出空 CMDSEQRD 以将第一个读数据复制到 VISI 中。丢弃该空操作返回的数据，因为该数据是 VISI 的现有内容，而不是从读地址[23:0]读取的数据。			
CMDSEQRD	11	VISI	MOV.L [W0++], [W8]
步骤 4：使用 CMDSEQRD 读取 VISI 寄存器。			
CMDSEQRD	11	VISI	MOV.L [W0++], [W8]
步骤 5：重复步骤 4，直到从目标器件读取到所有连续数据为止。若要更新随机访问的地址，请返回步骤 2。			

3.6. CRC 校验和计算

该算法（表 3-6）用于计算闪存区域的 32 位循环冗余校验（CRC）校验和。该校验和可用于验证上述编程步骤中写入的数据，或者验证现有的闪存内容是否与已知模式匹配，该已知模式由于代码保护或 UCA 或 UCB 中定义的安全限制而无法直接读取。有关 CRC 计算算法的详细信息，请参见闪存控制器。

可通过用户配置 Ax 和用户配置 B 区域中的配置字禁止 NVM 控制器的 CRC 计算。

表 3-6. CRC 校验和计算算法

ICSP™序列	ICSP 命令代码	数据/操作码	执行的指令
步骤 1：将 VISI 地址传送至 W8，将 NVMCRCCON 地址传送至 W9。将 CRCEN 位（NVMCRCCON[15]）置 1 以使能 CRC 硬件。			
CMDEXEC	00	0x9C00C163	MOV.SL #NVMCRCDATA, W7
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C123	MOV.SL #NVMCRCCON, W9
CMDEXEC	00	0xC2F92008	BSET.L [W9], #15
步骤 2：将起始地址装入 NVMCRCST 寄存器，将结束地址装入 NVMCRCEND 寄存器，并将初始值装入 NVMCRCSEED 寄存器。			
CMDEXEC	00	0x8000C133	MOV.SL #NVMCRCST, W0
CMDSEQWR	10	起始地址[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	结束地址[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	初始值[31:0]	MOV.L #PGEDx<31:0>, [W0++]
步骤 3：将 START 位（NVMCRCCON[14]）置 1 以开始 CRC 计算。通过将 NVMCRCCON 的内容传送至 VISI 以准备好进行轮询。			
CMDEXEC	00	0xC2E92008	BSET.L [W9], #14
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
步骤 4：将 NVMCRCCON 的内容传送至 VISI。将 VISI 的内容移至 PGEDx 上。			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
步骤 5：重复步骤 4 以轮询 START 位（NVMCRCCON[14]），直到其清零（表示成功执行）。			
步骤 6：将 NVMCRCOUT 的内容传送至 VISI。将 VISI 的内容移至 PGEDx 上。			
CMDEXEC	00	0x83872400	MOV.L [W7], [W8]
CMDEXEC	00	0x00000000	NOP
CMDRD	01	VISI	

Microchip 信息

商标

“Microchip”的名称和徽标组合、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家或地区的注册商标或商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN： 979-8-3371-1485-9

法律声明

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。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。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，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或附带的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。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损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。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。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品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规范。
- Microchip 确信：在正常使用且符合工作规范的情况下，Microchip 系列产品非常安全。
- Microchip 注重并积极保护其知识产权。严禁任何试图破坏 Microchip 产品代码保护功能的行为，这种行为可能会违反《数字千年版权法案》（Digital Millennium Copyright Act）。
- 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。