第 60 章 32 位可编程循环冗余校验 （CRC）

目录

本章包括下列主题：

注： 本系列参考手册章节旨在用作对器件数据手册的补充。本手册章节可能并不适用于所有 PIC32 器件，具体取决于器件型号。

请参见当前器件数据手册中 **“32 位可编程循环冗余校验 （CRC）发生器”** 章节开头部分的注，以检查本文档是否支持您所使用的器件。

器件数据手册和系列参考手册章节可从 Microchip 网站下载：
http://www.microchip.com。

60.1 简介

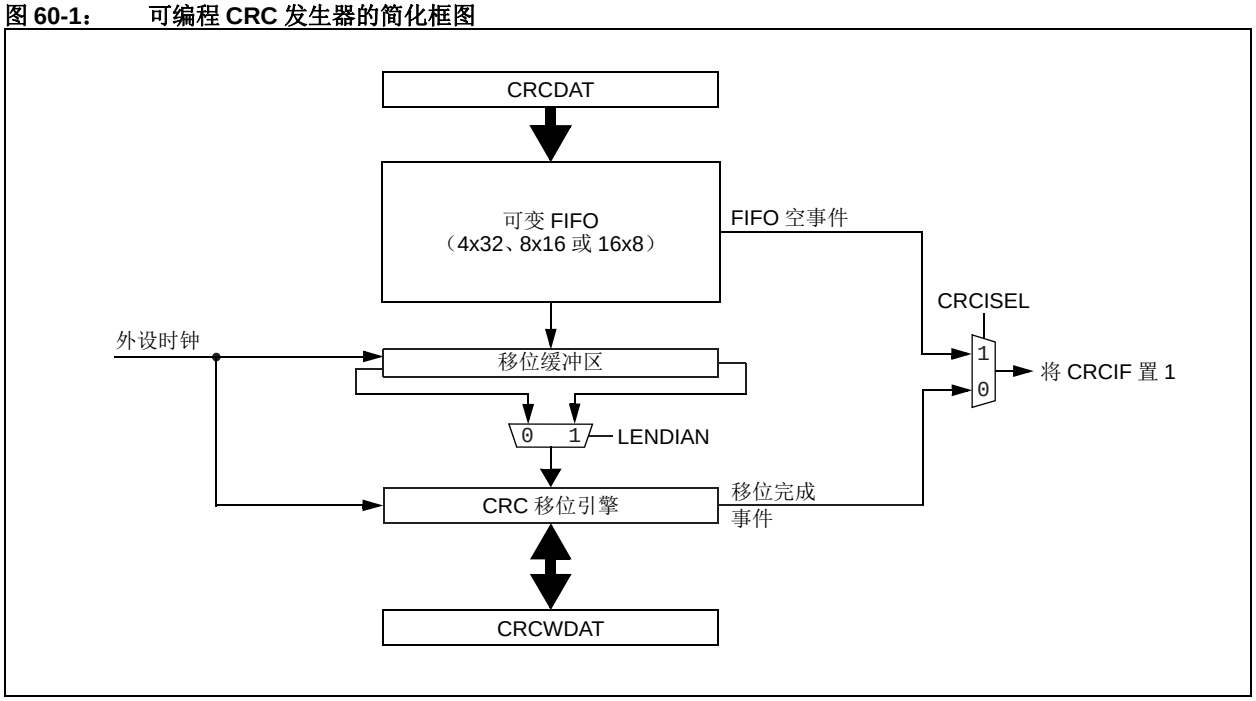
32 位可编程循环冗余校验（Cyclic Redundancy Check，CRC）模块是可用软件配置的 CRC 发生器。该模块提供了一种由硬件实现的方法，为各种通信和安防应用快速生成校验和。CRC 引擎可以在无需 CPU 干预的情况下计算 CRC 校验和；此外，它的速度远高于软件实现的速度。

可编程 CRC 发生器具有以下特性：

- 用户可编程 CRC 多项式方程，最多 32 位
- 可编程移位方向 （小尾数或大尾数）
- 独立的数据和多项式长度
- 可配置的中断输出
- 数据 FIFO

可编程 CRC 发生器模块可分为两个部分：控制逻辑和 CRC 引擎。控制逻辑中包含寄存器接口、数据 FIFO、中断发生器和 CRC 引擎接口。CRC 引擎中包含 CRC 计算器，该计算器使用带异或函数的串行移位器实现。图 60-1 给出了简化框图。

图 60-1： 可编程 CRC 发生器的简化框图



60.2 CRC 概述

校验和是与报文或包含若干字节的特定数据块关联的独特数字。无论是用于通信的数据包，还是存储在存储器中的数据块，诸如校验和这样的信息都可以在处理数据之前帮助验证数据。计算校验和最简单的方法是将报文中的所有数据字节相加。但是，当通过颠倒或交换字节组的顺序修改报文时，这种校验和计算方法会完全失效。此外，在报文中的任意位置添加空字节时，这种方法也会失效。

循环冗余校验和（Cyclic Redundancy Checksum， CRC）是一种更复杂但也更可靠的错误校验算法。CRC 算法中的主要原理是将报文视为二进制比特流，并将其除以固定的二进制数值。这种除法运算产生的余数将被视为校验和。与除法运算类似，CRC 计算也是一个迭代过程。唯一的区别在于这些运算是通过基于模 2 的模运算完成的。例如，除法运算被替换为异或运算（即，不带借位的减法）。CRC 算法使用多项式来执行所有计算。使用数字表示的除数、被除数和余数构造为带有二进制系数的多项式。例如，数字 25h （11001）表示为：

公式 60-1：

$$(1 * x^4) + (1 * x^3) + (0 * x^2) + (0 * x^1) + (1 * x^0) \text{ 或 } x^4 + x^3 + x^0$$

为了执行 CRC 计算，必须先选择适当的除数。该除数称为发生器多项式。由于使用 CRC 来检测错误，需要针对给定应用选择长度适合的发生器多项式，因为每一个多项式具有不同的错误检测能力。一些多项式广泛应用于许多应用，但任意特定多项式的错误检测能力超出了本文档的范围。

CRC 算法用软件实现比较简单。但是，它需要相当大的 CPU 带宽来实现基本的需求，例如移位、位测试和异或运算。此外，CRC 计算是一个迭代过程，数据传输指令所需的额外软件开销会对单片机的 MIPS 需求产生很大的负担。与之相对，可用软件配置的 CRC 硬件模块便于快速计算 CRC 校验和，并最大限度地降低了软件开销。

60.3 寄存器

CRC 模块使用以下特殊功能寄存器 （Special Function Register， SFR）：

- **CRCCON：CRC 控制寄存器**
  此寄存器控制所有模块功能，包括数据和多项式大小、累加器和尾数模式以及中断发生。
- **CRCXOR：CRC 异或寄存器**
  此寄存器配置 CRC 多项式。
- **CRCDAT：CRC FIFO 数据寄存器 （只写）**
  此寄存器装入计算的初始数据。
- **CRCWDAT：CRC 移位数据寄存器**
  此寄存器装入初始 CRC 值并在计算完成时保存最终结果。

表 60-1 和寄存器 60-1 至寄存器 60-4 包含 CRC 模块寄存器的说明。

表 60-1： CRC SFR 汇总 (1)

名称	位范围	Bit 31/15	Bit 30/14	Bit 29/13	Bit 28/12	Bit 27/11	Bit 26/10	Bit 25/9	Bit 24/8	Bit 23/7	Bit 22/6	Bit 21/5	Bit 20/4	Bit 19/3	Bit 18/2	Bit 17/1	Bit 16/0
CRCCON	31:16	—	—	—	DWIDTH<4:0>					—	—	—	PLEN<4:0>				
	15:0	ON	—	SIDL	VWORD<4:0>					CRCFUL	CRCMPT	CRCISEL	CRCGO	LENDIAN	MOD	—	—
CRCXOR	31:16	X<31:16>															
	15:0	X<15:0>															
CRCDAT	31:16	DATA<31:16>															
	15:0	DATA<15:0>															
CRCWDAT	31:16	SDATA<31:16>															
	15:0	SDATA<15:0>															

注 1： 所有寄存器均具有关联的清零、置 1 和取反寄存器，后缀为 -CLR、 -SET 和 -INV，地址偏移量分别为 0x4、 0x8 和 0xC 字节。这些相关寄存器用于修改其主寄存器。向相关寄存器中的位位置写入 1 时，会将主寄存器中的相应有效位清零、置 1 或取反。对这些寄存器的读操作将被忽略。

寄存器 60-1： CRCCON：CRC 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	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	DWIDTH<4:0>				
23:16	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	PLEN<4:0>				
15:8	R/W-0	U-0	R/W-0	R-0	R-0	R-0	R-0	R-0
	ON	—	SIDL	VWORD<4:0>				
7:0	R-0	R-1	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0
	CRCFUL	CRCMPT	CRCISEL	CRCGO	LENDIAN	MOD	—	—

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 **未实现：**读为 0

bit 28-24 **DWIDTH<4:0>：**数据字宽度配置位
配置数据字的宽度 （数据字宽度 - 1）。

bit 23-21 **未实现：**读为 0

bit 20-16 **PLEN<4:0>：**多项式长度配置位
配置 CRC 多项式的长度 （多项式长度 - 1）。

bit 15 **ON：**CRC 使能位
1 = 使能模块
0 = 禁止模块

bit 14 **未实现：**读为 0

bit 13 **SIDL：**CRC 空闲模式停止位
1 = 当器件进入空闲模式时，停止模块工作
0 = 模块在空闲模式下继续工作

bit 12-8 **VWORD<4:0>：**计数器值位
指示 FIFO 缓冲区中的未处理字数；缓冲区深度由数据字长度决定。

bit 7 **CRCFUL：**CRC FIFO 满位
1 = FIFO 已满
0 = FIFO 未满

bit 6 **CRCMPT：**CRC FIFO 空位
1 = FIFO 为空
0 = FIFO 非空

bit 5 **CRCISEL：**CRC 中断选择位
1 = 在 FIFO 为空时产生中断 （数据最后一个字仍然在 CRC 中移位）
0 = 在移位完成时产生中断 （FIFO 为空且不从移位缓冲区中移动任何数据）

bit 4 **CRCGO：**启动 CRC 移位位
1 = 启动 CRC 串行移位器；清零该位将中止移位
0 = CRC 串行移位器关闭

bit 3 **LENDIAN：**数据字小尾数法配置位
1 = 数据字从 LSb 开始移入 CRC （小尾数法）；反映输入数据
0 = 数据字从 MSb 开始移入 CRC （大尾数法）；不反映输入数据

bit 2 **MOD：**CRC 工作模式选择位
1 = 备用模式：移位缓冲区数据在位 n 后使用 CRC 移位引擎进行异或运算
0 = 传统模式：移位缓冲区数据在位 0 前使用 CRC 移位引擎进行异或运算

bit 1-0 **未实现：**读为 0

寄存器 60-2： CRCXOR：CRC 异或寄存器

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/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	X<31:24>							
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	X<23:16>							
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	X<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	X<7:0>							

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 31-0 **X<31:0>：**多项式的项 x^n 的异或使能位
1 = 在 CRC 多项式中使用多项式项 x^n
0 = 不使用多项式项

寄存器 60-3： CRCDAT：CRC FIFO 数据寄存器 （只写）

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	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
	DATA<31:24>							
23:16	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
	DATA<23:16>							
15:8	W-0	R/W-0	W-0	W-0	W-0	W-0	W-0	W-0
	DATA<15:8>							
7:0	W-0	R/W-0	W-0	W-0	W-0	W-0	W-0	W-0
	DATA<7:0>							

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 31-0 **DATA<31:0>：**FIFO 数据位
写入此寄存器的数据将被写入 CRC FIFO 缓冲区中的下一个空位置。
读取此寄存器将返回值 0。

寄存器 60-4： CRCWDAT：CRC 移位数据寄存器

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/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SDATA<31:24>							
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SDATA<23:16>							
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SDATA<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SDATA<7:0>							

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 31-0 **SDATA<31:0>：**移位寄存器数据位
向此寄存器的写入将直接写入 CRC 引擎移位寄存器，覆盖任何当前值。
读取此寄存器将返回 CRC 引擎移位寄存器的当前值。

60.4 CRC 引擎

60.4.1 通用 CRC 引擎

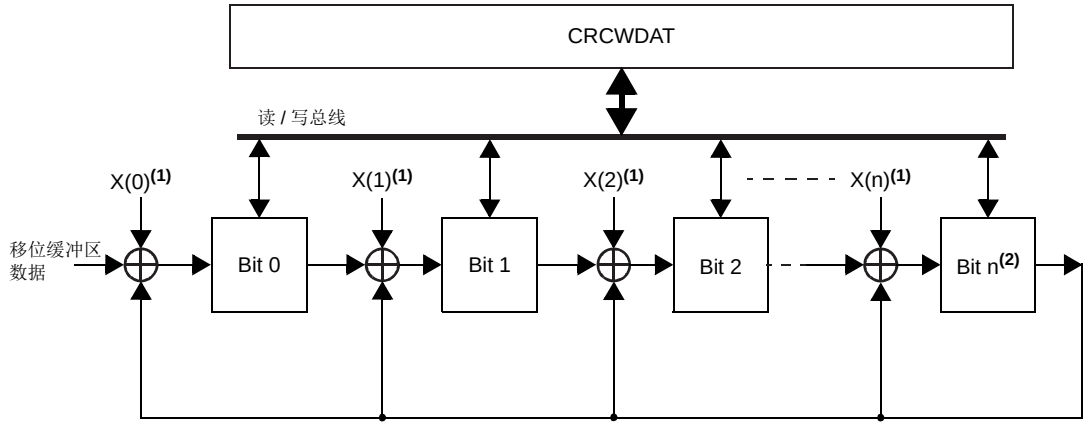
CRC 引擎是一个串行移位 CRC 计算器，可通过多路开关设置进行配置，还可以通过该引擎配置使用 MOD 位（CRCCON<2>）引入移位缓冲区数据的位置。图 60-2 给出了 CRC 移位引擎的简化框图。

CRC 算法使用简化的算术处理，即使用异或运算而不是二进制除法运算。发生器多项式的系数使用 CRCXOR<31:1> 位设定。向某个单元中写入 1 会使能对多项式中的对应元素的异或运算。多项式长度使用 CRCCON 寄存器（CRCCON<20:16>）中的 PLEN<4:0> 位设定。PLEN<4:0> 值用于控制多项式的长度，并切换多路开关来指示反馈来自哪个分接点。

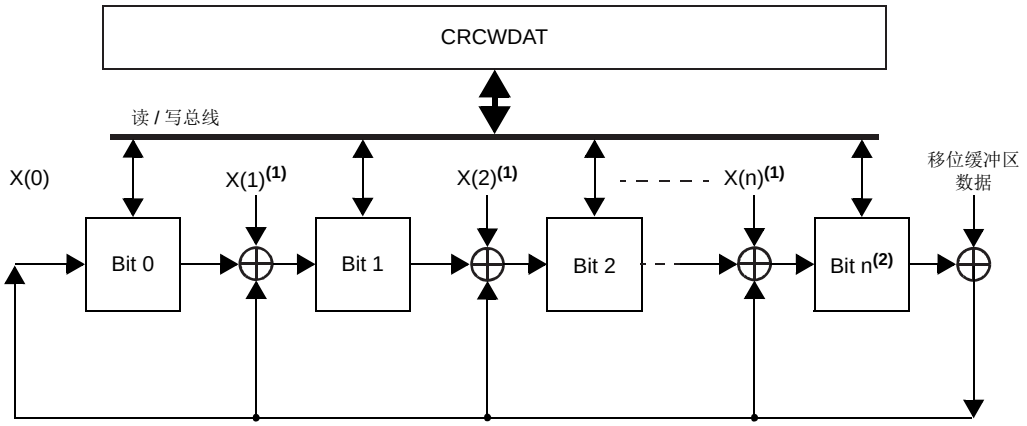
通过读取 CRCWDAT 寄存器，获得 CRC 计算的结果。

图 60-2： CRC 移位引擎详细信息

传统模式（MOD 位 = 0）



备用模式（MOD 位 = 1）



注 1： 移位引擎的每个逻辑异或（XOR）级都是可编程的。详情见正文。

2： 多项式长度 n 由（PLEN<4:0> + 1）决定。

60.5 控制逻辑

60.5.1 多项式接口

CRC 模块最多可使用 32 位，编程最高 32 阶的 CRC 多项式。多项式长度代表方程中的最高指数，它通过 PLEN<4:0> 位 （CRCCON<20:16>）进行选择。方程中包含 CRCXOR 寄存器控制的指数项。将特定位置 1 时，方程中将包含相应的指数项；从功能角度来说，将特定位置 1 时，在 CRC 引擎中将对相应位进行异或运算。清零该位将会禁止异或运算。

例如，假设有两个 CRC 多项式，一个是 16 位方程，另一个是 32 位方程 （公式 60-2）。要将这两个多项式编程到 CRC 发生器中，应按照表 60-2 中所示来设置寄存器位。

公式 60-2：

$$x^{16} + x^{12} + x^5 + 1$$

和

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

表 60-2： 16 位和 32 位多项式的 CRC 设置示例

CRC 控制位	位值	
	16 位多项式	32 位多项式
PLEN<4:0>	01111	11111
X<31:16>	0000 0000 0000 0000	0000 0100 1100 0001
X<15:1>	0001 0000 0010 0001	0001 1101 1011 0111

可以注意到，相应的一些位设置为 1，指示要在方程中使用它们 （例如， X26 和 X23）。多项式的最高有效位 （Most Significant bit， MSb）不会影响计算，并且可以设置为任意值。

60.5.2 数据移位方向

LENDIAN 位 （CRCCON<3>）用于控制移位方向。默认情况下， CRC 模块从 MSb 开始通过引擎进行数据移位 （LENDIAN = 0）。将 LENDIAN 置 1 时， CRC 模块会从 LSb 开始进行数据移位。通过该设置，可以更好地与各种通信方案进行集成，并且消除在软件中反转位顺序的开销。注意，该设置仅改变移入引擎的数据的方向。CRC 计算的结果仍然是正常的 CRC 结果，不是反转的 CRC 结果。

PIC32 器件为小尾数法。为大尾数法 （LENDIAN = 0）配置 CRC 模块时，输入数据字节和字必须在应用程序代码中进行交换，然后才能将其装入数据 FIFO （CRCDAT 寄存器）中。

60.5.3 数据 FIFO

模块具有可处理可变数据宽度的 FIFO。数据宽度由 DWIDTH<4:0> 位 （CRCCON<28:24>）定义。它可以配置为介于 1 到 32 位之间的任意值。与 FIFO 关联的逻辑包含一个 5 位计数器，即 VWORD<4:0> 位 （CRCCON<12:8>）。

VWORD<4:0> 位中的值指示 FIFO 中的未处理数据元素的数量。 FIFO 为：

- DWIDTH<4:0> ≤ 7 时为 16 字深 （数据字， 8 位宽或更小）
- DWIDTH<4:0> ≤ 15 时为 8 字深 （数据字， 9 位至 16 位宽）
- DWIDTH<4:0> ≤ 31 时为 4 字深 （数据字， 17 位至 32 位宽）

必须先使用 CRCDAT 寄存器将 CRC 计算需要的数据写入 FIFO。读取 CRCDAT 将始终返回 0。为适应从 MSb 开始的移位方法（LENDIAN = 0），必须在填充 FIFO 时在软件中完成字节和字的交换。

注： 当 CRCFUL 位置 1 时，请确保不要将新数据写入 CRCDAT 寄存器；如果写入新数据，它会被忽略。

完成所有移位（即 FIFO 为空且 CRC 移位引擎为空闲）时，可以更改 FIFO 宽度（DWIDTH<4:0> 位），而不会有任何信息丢失或 CRC 结果损坏。

由于数据宽度为 8 位或更小，FIFO 将在每次写 CRCDAT 寄存器的低字节时递增 1（必须使用对 CRCDAT 寄存器的字节访问）。可写入 FIFO 的最小数据元素是 1 个字节。

例如，如果 DWIDTH<4:0> 值为 5，那么数据宽度为 DWIDTH<4:0> + 1 （即 6）。数据以整个字节的形式写入；将忽略两个未用的最高位。数据字节写入 CRCDAT 寄存器后，VWORD<4:0> 位（CRCCON<12:8>）的值将递增 1。

由于数据宽度超过 8 位，且小于或等于 16 位，FIFO 将在每次写 CRCDAT 寄存器的低 16 位字时递增 1（必须使用对 CRCDAT 寄存器的 16 位字访问）。将忽略未使用的高数据位。VWORD<4:0> 位的值在每次写 CRCDAT 寄存器时递增 1。

在数据宽度大于 16 位时，对 CRCDAT 寄存器的任意写操作都会使 VWORD<4:0> 位递增 1。将忽略未使用的高数据位。

60.5.4 CRC 引擎接口

60.5.4.1 FIFO 到 CRC 移位引擎

要启动从 FIFO 到 CRC 移位缓冲区的数据移动，必须将 CRCGO 位（CRCCON<4>）置 1。CRCGO = 1 且 VWORD<4:0> 的值大于 0 时，串行移位器从移位缓冲区开始将数据移入 CRC 移位引擎，对于 LENDIAN = 0，先从 MSb 开始移位，对于 LENDIAN = 1，先从 LSb 开始移位。如果 CRCFUL 位先前置 1，则在 VWORDx 位递减 1 时，它会被清零。当 FIFO 位置移到移位缓冲区时，VWORD<4:0> 位会递减 1。串行移位器会继续进行移位，直到 VWORD<4:0> 位变为零；此时，CRCMPT 位会置 1，指示 FIFO 为空。如果在 CRC 计算期间将 CRCGO 位清除，则 CRC 移位引擎将停止计算，直到 CRCGO 位置 1。

应用程序可以在移位操作正在进行时向 FIFO 写入数据。应该对 CRCFUL 位进行监视。如果 CRCFUL 位未置 1，则可以向 FIFO 中写入另一个字。写入 CRCDAT 寄存器后必须经过至少一个指令周期才能读取 VWORD<4:0> 位的有效值。

当 VWORD<4:0> 位达到对应于 DWIDTH<4:0> 位配置的最大值时， CRCFUL 位会置 1。当 VWORD<4:0> 位变为零时，CRCMPT 位会置 1。每当 ON 为 0 时，FIFO 会清空，且 VWORD<4:0> 位会设置为 00000。

60.5.4.2 数据移位的时钟周期数

FIFO 中的数据将传送到移位缓冲区。要开始将数据字从 FIFO 移入移位缓冲区，需要 2 个外设时钟周期。然后会将移位缓冲区中的数据移入 CRC 移位引擎。要将数据从移位缓冲区完全移入 CRC 移位引擎，需要（DWIDTH<4:0> + 1）个时钟周期。例如，如果 DWIDTH<4:0> = 5，则数据长度为 6 位（DWIDTH<4:0> + 1），需要 6 个周期来进行数据移位。在这种情况下，仅会移出一个字节的 6 位。每个字节的高 2 位是 “无关”位。同样，对于 12 位多项式选择，将忽略每个字的高 4 位。

60.5.4.3 CRC 初始值

通过 CRCWDAT 寄存器访问 CRC 移位引擎。在开始计算之前，可以向此寄存器装入所需的 CRC 初始值。此初始值的形式取决于 MOD 位（CRCCON<2>）所选择的操作模式。

在备用模式（MOD = 1）中， CRC 初始值必须使用直接形式。

在传统模式（MOD = 0）中，CRC 初始值必须使用非直接形式。非直接初始值是 CRC 计算赋予所需直接 CRC 初始值的值。例如，如果应用程序使用 CRC-32 多项式 0x04C11DB7，且必须从 CRC 直接初始值 0xFFFFFFFF 开始计算，那么必须在 CRCWDAT 寄存器中装入非直接值 0x46AF6449（此非直接值 0x46AF6449 的 CRC 为 0xFFFFFFFF）。使用 CRCWDAT 寄存器将非直接初始值写入移位引擎时， CRC 模块会在 （PLEN<4:0> + 1）个外设时钟周期之后将其转换为直接初始值。

注： 向 CRCWDAT 寄存器写入数据将清除 / 复位移位缓冲区。

通常，CRC 计算每次都从相同的初始值开始。在这种情况下，非直接初始值只能找到一次，然后可以在应用程序代码中将其定义为常量。

注： 0 的 CRC 非直接初始值为 0。

例 60-1 给出了一个可行的软件程序，该程序可从直接初始值获得非直接初始值。

例 60-1： 计算非直接初始值的软件程序

```
unsigned   long   CalculateNonDirectSeed(
unsigned   long   seed,                     // direct CRC initial value
unsigned   long   polynomial,               // polynomial
unsigned   char   polynomialOrder)          // polynomial order
{
unsigned   char   lsb;
unsigned   char   i;
unsigned   long   msbmask;

    msbmask = ((unsigned long)1)<<(polynomialOrder-1);
    for (i=0; i<polynomialOrder; i++) {
        lsb = seed & 1;
        if (lsb) seed ^= polynomial;
        seed >>= 1;
        if (lsb) seed |= msbmask;
    }
return     seed;                            // return the non-direct CRC initial value
}
```

CRC 模块可用于获得非直接初始值。为此：

1. 使能 CRC 模块 （ON = 1）和移位 （CRCGO = 1）。
2. 将多项式值向右移动 1 位。
3. 反转移位的多项式值的位顺序。
4. 在 CRCXOR 寄存器中写入此结果。
5. 将数据宽度和多项式长度（DWIDTH<4:0> 和 PLEN<4:0> 位）设置为多项式阶（长度）。
6. 反转所需直接初始值的位顺序。
7. 在 CRCWDAT 寄存器中写入反向初始值。
8. 将无效数据写入 CRCDAT 寄存器并等待 2 个外设时钟周期，以将数据从 FIFO 移入移位缓冲区，并等待 （PLEN<4:0> + 1）个外设时钟周期，以将结果移出。

   或者，也可以清除 CRC 中断选择位 （CRCISEL = 0），以在完成从移位缓冲区的移位时获得中断，清除 CRC 中断标志，在 CRCDAT 寄存器中写入无效数据，并等待将 CRC 中断标志置 1。
9. 从 CRCWDAT 寄存器读取值。
10. 反转读取结果的位顺序；这会提供最终的非直接初始值。

例 60-2 给出了一种实现此过程的方法。

为继续计算整个数据报文，在计算中途必须读取中间 CRC 和的应用程序中，必须再次计算非直接值并将该值设置到 CRCWDAT 寄存器中。在这种情况下，CRC 直接初始值将是一个中间 CRC 结果读取值。

例 60-2： 计算非直接初始值 （MOD 位 = 0）

```
unsigned    long    CalculateNonDirectSeed(unsigned long seed,  // direct CRC initial value
                    unsigned long polynomial,                   // polynomial
unsigned    char    polynomialOrder)                            // polynomial order (valid values
                                                                // are 8, 16, 32 bits)
{
    CRCCON = 0;
    CRCCONbits.ON = 1;                                          // enable CRC
    CRCCONbits.CRCISEL = 0;                                     // interrupt when all shifts are done
    CRCCONbits.DWIDTH = polynomialOrder-1;                      // data width
    CRCCONbits.PLEN = polynomialOrder-1;                        // polynomial length
    CRCCONbits.CRCGO = 1;                                       // start CRC calculation

    polynomial >>= 1;                                           // shift the polynomial right
    polynomial = ReverseBitOrder(polynomial, polynomialOrder);  // reverse bits order of the
                                                                // polynomial
    CRCXOR = polynomial;                                        // set the reversed polynomial
    seed = ReverseBitOrder(seed, polynomialOrder);              // reverse bits order of seed value
    CRCWDAT = seed;                                             // set seed value
    IFS0CLR = _IFS0_CRCIF_MASK;                                 // clear interrupt flag
    switch(polynomialOrder)                                     // load dummy data to shift out
                                                                // seed result
    {
        case 8:
             *((unsigned char*)&CRCDAT) = 0;                    // load byte
             while(!IFS0bits.CRCIF);                            // wait until shifts are done
             seed = CRCWDAT&0x0FFFF;                            // read reversed seed
        case 16:
             *((unsigned short*)&CRCDAT) = 0;                   // load short
             while(!IFS0bits.CRCIF);                            // wait until shifts are done
             seed = CRCWDAT&0x00FFFF;                           // read reversed seed
             break;
        case 32:
             // load long
             CRCDAT = 0;
             while(!IFS0bits.CRCIF);                            // wait for shifts are done
             seed = CRCWDAT;                                    // read reversed seed
             break;
        default:
             ;
    }
    seed = ReverseBitOrder(seed, polynomialOrder);              // reverse the bit order to get
                                                                // the non-direct seed
    return seed;                                                // return non-direct CRC initial value
}
```

例 60-2： 计算非直接初始值的程序 （MOD 位 = 0）（续）

```
 // WHERE THE FUNCTION TO REVERSE THE BIT ORDER CAN BE

unsigned long  ReverseBitOrder(unsigned long data,       // input data
unsigned char numberOfBits)                              // width of the input data,
                                                         // valid values are 8,16,32 bits
{
unsigned long  maskin  = 0;
unsigned long  maskout = 0;
unsigned long  result = 0;
unsigned char  i;

    switch(numberOfBits)
    {
        case 8:
           maskin = 0x80;
           maskout = 0x01;
           break;
        case 16:
           maskin = 0x8000;
           maskout = 0x0001;
           break;
        case 32:
           maskin = 0x80000000;
           maskout = 0x00000001;
           break;
        default:
           ;
    }
    for(i=0; i<numberOfBits; i++)
    {
        if(data&maskin){
           result |= maskout;
        }
           maskin  >>= 1;
           maskout <<= 1;
    }
    return result;
}
```

60.5.4.4 CRC 结果

CRC 计算结果的读取取决于所选的操作模式。

在备选模式（MOD = 1）中，如果 CRC FIFO 缓冲区中的所有数据都已处理，可在 CRCWDAT 寄存器中获得结果。需要提交无效数据才能生成额外周期。

在传统模式（MOD = 0）中， CRC 模块需要（PLEN<4:0> + 1）个额外的外设时钟周期来完成计算。为生成这些额外周期，必须将宽度等于多项式阶（长度）的无效数据装入 CRCDAT 寄存器中。完成移位后，可以从 CRCWDAT 寄存器读取最终的 CRC 结果（CRCWDAT 寄存器提供对 CRC 移位寄存器的直接访问）。

获得 CRC 结果后，应该将 CRC 非直接初始值重新写入 CRCWDAT 寄存器，以清除 / 复位之前装入的无效数据的移位缓冲区，以便启动新的计算。

60.5.5 中断操作

模块可以在两种条件下产生中断，可由用户配置。如果 CRCISEL 为 1，在 VWORD<4:0> 位的值从 1 变为 0 时将产生中断。如果 CRCISEL 为 0，在 FIFO 为空且完成从移位缓冲区的移位时将产生中断。有些 CRC 模块可以使用持久中断（也称为电平中断）。对于这些模块，如果数据 FIFO 为空，则无法清除中断标志。在将某些数据写入 FIFO 后，应该清除中断标志。

关于中断和中断优先级设置的更多详细信息，请参见《PIC32 系列参考手册》中的**第 8 章 “中断”**（DS60001108）。

60.6 CRC 模块的应用

在数字通信中，对于包含若干个字节或字的报文， CRC 是一种可靠的错误校验算法。在计算之后，校验和会被附加在报文末尾，并传送给接收站。接收器会使用接收到的报文计算校验和，以验证数据完整性。

60.6.1 变化

32 位可编程 CRC 模块可以设定为先移出 MSb 或先移出 LSb。先移出 MSb 是一种流行的实现方式，XMODEM 协议就采用了这种方式。CRC 计算还有另外一种形式（CCITT 协议），采用这种方式时，LSb 先移出。讨论所有形式超出了本文档的内容范围，但可以使用此模块实现多种 CRC 形式。

多项式长度和多项式自身的选择取决于应用。在各种标准实现中，常用的多项式长度为 5、7、8、10、 12、 16 和 32。以下章节说明了 CRC 计算的建议分步过程。用户可以决定是否需要在报文流末尾附加零或任意其他值。根据应用，用户可以决定究竟是否要附加任何值。

60.6.2 典型操作

使用模块进行典型的 CRC 计算：

1. 将 ON 位置 1 以使能模块。
2. 将模块配置为所需的操作：
   a) 使用 CRCXOR 寄存器，以及 PLEN<4:0> 位编程所需的多项式。
   b) 使用 DWIDTH<4:0> 和 LENDIAN 位配置数据宽度和移位方向。
3. 将 CRCGO 位置 1 以启动计算。
4. 在 CRCWDAT 寄存器中将所需 CRC 初始值置 1，如**第 60.5.4.3 节“CRC 初始值”**中所述。
5. 在具有可用空间时通过写入 CRCDAT 寄存器将所有数据装入 FIFO （下一次数据装入前 CRCFUL 位必须为 0）。
6. 等待直到数据 FIFO 为空 （CRCMPT 位被置 1）。
7. 读取 CRC 结果，如**第 60.5.4.4 节 “CRC 结果”**中所述。

60.6.3 具有边沿中断的 CRC 模块应用示例

对于具有边沿中断的模块，例 60-3 至例 60-12 给出了各种不同组合的多项式长度、数据宽度、移位方向和 CRC 引擎模式的典型代码。

例 60-3： CRC-SMBus（具有 32 位数据、大尾数且 MOD 位 = 1 的 8 位多项式）

```
// This macro is used to swap bytes for big endian
#define   Swap(x) __extension__({ \
unsigned  long __x = (x), __v; \
__asm__   ("wsbh %0,%1;\n\t" \
"rotr %0,16" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})

// ASCII bytes "12345678"
volatile  unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
volatile  unsigned char crcResultCRCSMBUS = 0;
int       main (void)
{
unsigned  long*  pointer;
unsigned  short length;
unsigned  long  data;

// standard CRC-SMBUS

#define   CRCSMBUS_POLYNOMIAL ((unsigned long)0x00000007)
#define   CRCSMBUS_SEED_VALUE ((unsigned long)0x00000000)   // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                     // alternate mode
    CRCCONbits.ON = 1;                                      // enable CRC
    CRCCONbits.LENDIAN = 0;                                 // big endian
    CRCCONbits.CRCISEL = 0;                                 // interrupt when all shifts are done
    CRCCONbits.DWIDTH = 32-1;                               // 32-bit data width
    CRCCONbits.PLEN = 8-1;                                  // 8-bit polynomial order
    CRCXOR = CRCSMBUS_POLYNOMIAL;                           // set polynomial
    CRCWDAT = CRCSMBUS_SEED_VALUE;                          // set initial value
    CRCCONbits.CRCGO = 1;                                   // start CRC calculation

    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                           // wait if FIFO is full
        data = *pointer++;                                  // load from little endian
        data = Swap(data);                                  // swap bytes for big endian
        length--;
        if(length == 0)
        {
          break;
        }
        CRCDAT = data;                                      // 32-bit word access to FIFO
    }
    CRCCONbits.CRCGO = 0;                                   // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                             // clear the interrupt flag
    CRCDAT = data;                                          // write last data into FIFO
    CRCCONbits.CRCGO = 1;                                   // resume CRC calculation

    while(!IFS0bits.CRCIF);                                 // wait until shifts are done
    crcResultCRCSMBUS = (unsigned char)CRCWDAT&0x00ff;      // get CRC result (must be 0xC7)

    while(1);
    return 1;
}
```

例 60-4： CRC-SMBus （具有 32 位数据、小尾数且 MOD 位 = 0 的 8 位多项式）

```
// This macro is used to swap bytes for big endian
#define   Swap(x) __extension__({ \
unsigned  long __x = (x), __v; \
__asm__   ("wsbh %0,%1;\n\t" \
"rotr %0,16" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})

// ASCII bytes "12345678"
volatile  unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
volatile  unsigned char crcResultCRCSMBUS = 0;
int       main (void)
{
unsigned  long* pointer;
unsigned  short length;
unsigned  long  data;

// standard CRC-SMBUS

#define   CRCSMBUS_POLYNOMIAL ((unsigned long)0x00000007)
#define   CRCSMBUS_SEED_VALUE ((unsigned long)0x00000000) // non-direct initial value of zero

    CRCCON = 0;
    CRCCONbits.MOD = 0;                                   // legacy mode
    CRCCONbits.ON = 1;                                    // enable CRC
    CRCCONbits.LENDIAN = 0;                               // big endian
    CRCCONbits.CRCISEL = 0;                               // interrupt when all shifts are done
    CRCCONbits.DWIDTH = 32-1;                             // 32-bit data width
    CRCCONbits.PLEN = 8-1;                                // 8-bit polynomial order
    CRCXOR = CRCSMBUS_POLYNOMIAL;                         // set polynomial
    CRCWDAT = CRCSMBUS_SEED_VALUE;                        // set initial value
    CRCCONbits.CRCGO = 1;                                 // start CRC calculation
    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                         // wait if FIFO is full
        data = *pointer++;                                // load from little endian
        data = Swap(data);                                // swap bytes for big endian
        length--;
        if(length == 0)
        {
          break;
        }
        CRCDAT = data;                                    // 32-bit word access to FIFO
    }

    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear the interrupt flag
    CRCDAT = data;                                        // write last data into FIFO
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done
    CRCCONbits.DWIDTH = 8-1;                              // switch data width to polynomial length 8-bit
    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear the interrupt flag
    *((unsigned char*)&CRCDAT) = 0;                       // 8-bit dummy data to shift out the CRC result
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done
    crcResultCRCSMBUS = (unsigned char)CRCWDAT&0x00ff;    // get CRC result (must be 0xC7)

    while(1);
    return 1;
}
```

例 60-5： CRC-16（具有 32 位多项式、小尾数且 MOD 位 = 1 的 16 位数据）

```
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRC16 = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-16

#define   CRC16_POLYNOMIAL ((unsigned long)0x00008005)
#define   CRC16_SEED_VALUE ((unsigned long)0x00000000)    // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                   // alternate mode
    CRCCONbits.ON = 1;                                    // enable CRC
    CRCCONbits.CRCISEL = 0;                               // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                               // little endian
    CRCCONbits.DWIDTH = 16-1;                             // 16-bit data width
    CRCCONbits.PLEN = 16-1;                               // 16-bit polynomial order
    CRCXOR = CRC16_POLYNOMIAL;                            // set polynomial
    CRCWDAT = CRC16_SEED_VALUE;                           // set initial value
    CRCCONbits.CRCGO = 1;                                 // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                         // wait if FIFO is full
        data = *pointer++;                                // load data
        length--;
        if(length == 0)
        {
          break;
        }
        *((unsigned short*)&CRCDAT) = data;               // 16-bit word access to FIFO
    }

    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear the interrupt flag
    *((unsigned short*)&CRCDAT) = data;                   // write last data into FIFO
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done
    crcResultCRC16 = (unsigned short)CRCWDAT;             // get CRC result (must be 0xE716)

    while(1);
    return 1;
}
```

例 60-6： CRC-16（具有 16 位多项式、小尾数且 MOD 位 = 0 的 16 位数据）

```
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRC16 = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-16

#define  CRC16_POLYNOMIAL ((unsigned long)0x00008005)
#define  CRC16_SEED_VALUE ((unsigned long)0x00000000)    // non-direct initial value of zero

    CRCCON = 0;
    CRCCONbits.MOD = 0;                                  // legacy mode
    CRCCONbits.ON = 1;                                   // enable CRC
    CRCCONbits.CRCISEL = 0;                              // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                              // little endian
    CRCCONbits.DWIDTH = 16-1;                            // 16-bit data width
    CRCCONbits.PLEN = 16-1;                              // 16-bit polynomial order
    CRCXOR = CRC16_POLYNOMIAL;                           // set polynomial
    CRCWDAT = CRC16_SEED_VALUE;                          // set initial value
    CRCCONbits.CRCGO = 1;                                // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                        // wait if FIFO is full
        data = *pointer++;                               // load data
        *((unsigned short*)&CRCDAT) = data;              // 16-bit word access to FIFO
    }

    while(CRCCONbits.CRCFUL);                            // wait if FIFO is full
    CRCCONbits.CRCGO = 0;                                // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                          // clear the interrupt flag
    *((unsigned short*)&CRCDAT) = 0;                     // 16-bit dummy data to shift out CRC result
    CRCCONbits.CRCGO = 1;                                // resume CRC calculation
    while(!IFS0bits.CRCIF);                              // wait until shifts are done
    crcResultCRC16 = (unsigned short)CRCWDAT;            // get CRC result (must be 0xE716)

    while(1);
    return 1;
}
```

例 60-7： CRC-CCITT（具有 16 位数据、大尾数且 MOD 位 = 1 的 16 位多项式）

```
// This macro is used to swap bytes for big endian
#define  Swap(x) __extension__({ \
unsigned long __x = (x), __v; \
__asm__  ("wsbh %0,%1;\n\t" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRCCCITT = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-CCITT

#define   CRCCCITT_POLYNOMIAL ((unsigned long)0x00001021)
#define   CRCCCITT_SEED_VALUE ((unsigned long)0x0000FFFF)     // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                       // alternate mode
    CRCCONbits.ON = 1;                                        // enable CRC
    CRCCONbits.CRCISEL = 0;                                   // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 0;                                   // big endian
    CRCCONbits.DWIDTH = 16-1;                                 // 16-bit data width
    CRCCONbits.PLEN = 16-1;                                   // 16-bit polynomial order
    CRCXOR = CRCCCITT_POLYNOMIAL;                             // set polynomial
    CRCWDAT = CRCCCITT_SEED_VALUE;                            // set initial value
    CRCCONbits.CRCGO = 1;                                     // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                             // wait if FIFO is full
        data = *pointer++;                                    // load data
        data = Swap(data);                                    // swap bytes for big endian
        length--;
        if(length == 0)
        {
          break;
        }
        *((unsigned short*)&CRCDAT) = data;                   // 16-bit word access to FIFO
    }

    CRCCONbits.CRCGO = 0;                                     // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                               // clear the interrupt flag
    *((unsigned short*)&CRCDAT) = data;                       // write last data into FIFO
    CRCCONbits.CRCGO = 1;                                     // resume CRC calculation
    while(!IFS0bits.CRCIF);                                   // wait until shifts are done
    crcResultCRCCCITT = (unsigned short)CRCWDAT;              // get CRC result (must be 0x9B4D)

    while(1);
    return 1;
}
```

例 60-8： CRC-CCITT（具有 16 位数据、大尾数且 MOD 位 = 0 的 16 位多项式）

```
// This macro is used to swap bytes for big endian
#define  Swap(x) __extension__({ \
unsigned long __x = (x), __v; \
__asm__  ("wsbh %0,%1;\n\t" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRCCCITT = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-CCITT

#define  CRCCCITT_POLYNOMIAL ((unsigned long)0x00001021)
#define  CRCCCITT_SEED_VALUE ((unsigned long)0x000084CF)    // non-direct initial value of 0xFFFF

    CRCCON = 0;
    CRCCONbits.MOD = 0;                                     // legacy mode
    CRCCONbits.ON = 1;                                      // enable CRC
    CRCCONbits.CRCISEL = 0;                                 // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 0;                                 // big endian
    CRCCONbits.DWIDTH = 16-1;                               // 16-bit data width
    CRCCONbits.PLEN = 16-1;                                 // 16-bit polynomial order
    CRCXOR = CRCCCITT_POLYNOMIAL;                           // set polynomial
    CRCWDAT = CRCCCITT_SEED_VALUE;                          // set initial value
    CRCCONbits.CRCGO = 1;                                   // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                           // wait if FIFO is full
        data = *pointer++;                                  // load data
        data = Swap(data);                                  // swap bytes for big endian
        *((unsigned short*)&CRCDAT) = data;                 // 16-bit word access to FIFO
    }

    while(CRCCONbits.CRCFUL);                               // wait if FIFO is full
    CRCCONbits.CRCGO = 0;                                   // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                             // clear the interrupt flag
    *((unsigned short*)&CRCDAT) = 0;                        // 16-bit dummy data to shift out CRC result
    CRCCONbits.CRCGO = 0;                                   // resume CRC calculation
    while(!IFS0bits.CRCIF);                                 // wait until shifts are done
    crcResultCRCCCITT = (unsigned short)CRCWDAT;            // get CRC result (must be 0x9B4D)

    while(1);
    return 1;
}
```

例 60-9： CRC-32（具有 32 位数据、小尾数且 MOD 位 = 1 的 32 位多项式）

```
// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
// function to reverse the bit order (OPTIONAL)
unsigned long ReverseBitOrder(unsigned long data);
volatile unsigned int crcResultCRC32 = 0;
int      main(void)
{
unsigned long*  pointer;
unsigned short  length;

// standard CRC-32

#define  CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define  CRC32_SEED_VALUE ((unsigned long)0xFFFFFFFF)   // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                 // alternate mode
    CRCCONbits.ON = 1;                                  // enable CRC
    CRCCONbits.CRCISEL = 0;                             // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                             // little endian
    CRCCONbits.DWIDTH = 32-1;                           // 32-bit data width
    CRCCONbits.PLEN = 32-1;                             // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                          // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                         // set initial value
    CRCCONbits.CRCGO = 1;                               // start CRC calculation
    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                       // wait if FIFO is full
        length--;
        if(length == 0)
        {
          break;
        }
        CRCDAT = *pointer++;                            // 32-bit word access to FIFO
    }
    CRCCONbits.CRCGO = 0;                               // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                         // clear the interrupt flag
    CRCDAT = *pointer;                                  // write last data into FIFO
    CRCCONbits.CRCGO = 1;                               // resume CRC calculation
    while(!IFS0bits.CRCIF);                             // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                           // get the final CRC result
    // OPTIONAL reverse CRC value bit order and invert (must be 0x9AE0DAAF)
    crcResultCRC32 = ~ReverseBitOrder(crcResultCRC32);
    while(1);
    return 1;
}
unsigned long ReverseBitOrder(unsigned long data)
{
unsigned long maskin;
unsigned long maskout;
unsigned long result = 0;
unsigned char i;
maskin = 0x80000000;
maskout = 0x00000001;
    for(i=0; i<32; i++)
    {
        if(data&maskin){
          result |= maskout;
        }
        maskin >>= 1;
        maskout <<= 1;
    }
return   result;
}
```

例 60-10：CRC-32（具有 32 位数据、小尾数且 MOD 位 = 0 的 32 位多项式）

```
// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
// function to reverse the bit order (OPTIONAL)
unsigned long ReverseBitOrder(unsigned long data);
volatile unsigned int crcResultCRC32 = 0;
int      main(void)
{
unsigned long*  pointer;
unsigned short  length;

// standard CRC-32

#define   CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define   CRC32_SEED_VALUE ((unsigned long)0x46AF6449)   // non-direct initial value of 0xFFFFFFFF

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                  // alternate mode
    CRCCONbits.ON = 1;                                   // enable CRC
    CRCCONbits.CRCISEL = 0;                              // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                              // little endian
    CRCCONbits.DWIDTH = 32-1;                            // 32-bit data width
    CRCCONbits.PLEN = 32-1;                              // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                           // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                          // set initial value
    CRCCONbits.CRCGO = 1;                                // start CRC calculation

    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                        // wait if FIFO is full
        CRCDAT = *pointer++;                             // 32-bit word access to FIFO
    }
    while(CRCCONbits.CRCFUL);                            // wait if FIFO is full
    CRCCONbits.CRCGO = 0;                                // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                          // clear the interrupt flag
    CRCDAT = 0;                                          // 32-bit dummy data to shift out the CRC result
    CRCCONbits.CRCGO = 1;                                // resume CRC calculation
    while(!IFS0bits.CRCIF);                              // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                            // get the final CRC result
    // OPTIONAL reverse CRC value bit order and invert (must be 0x9AE0DAAF)
    crcResultCRC32 = ~ReverseBitOrder(crcResultCRC32);
    while(1);
    return 1;
}

unsigned long ReverseBitOrder(unsigned long data)
{
unsigned long maskin;
unsigned long maskout;
unsigned long result = 0;
unsigned char i;
maskin = 0x80000000;
maskout = 0x00000001;
    for(i=0; i<32; i++)
    {
        if(data&maskin){
          result |= maskout;
        }
        maskin >>= 1;
        maskout <<= 1;
    }
return   result;
}
```

例 60-11：数据宽度切换（32 位多项式、小尾数且 MOD 位 = 1）

```
// ASCII bytes "12345678"
volatile unsigned long message1[] = {0x34333231,0x38373635};
// ASCII bytes "123"
volatile unsigned char message2[] = {'1','2','3'};
volatile unsigned long crcResultCRC32 = 0;
int      main(void)
{
unsigned char*  pointer8;
unsigned long* pointer32;
unsigned short  length;
#define   CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define   CRC32_SEED_VALUE ((unsigned long)0xFFFFFFFF)    // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                   // alternate mode
    CRCCONbits.ON = 1;                                    // enable CRC
    CRCCONbits.CRCISEL = 0;                               // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                               // little endian
    CRCCONbits.DWIDTH = 32-1;                             // 32-bit data width
    CRCCONbits.PLEN = 32-1;                               // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                            // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                           // set initial value
    CRCCONbits.CRCGO = 1;                                 // start CRC calculation

    pointer32 = (unsigned long*)message1;
    length = sizeof(message1)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                         // wait if FIFO is full
        length--;
        if(length == 0)
        {
         break;
        }
        CRCDAT = *pointer32++;                            // 32-bit word access to FIFO
    }
    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear the interrupt flag
    CRCDAT = *pointer32;                                  // write last 32-bit data into FIFO
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done
    CRCCONbits.DWIDTH = 8-1;                              // switch the data width to 8-bit

    pointer8 = (unsigned char*)message2;                  // calculate CRC
    length = sizeof(message2)/sizeof(unsigned char);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                         // wait if FIFO is full
        length--;
        if(length == 0)
        {
         break;
        }
        *((unsigned char*)&CRCDAT) = *pointer8++;         // byte access to FIFO
    }
    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear the interrupt flag
    *((unsigned char*)&CRCDAT) = *pointer8;               // write last 8-bit data into FIFO
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                             // get the final CRC result (must be 0xE092727E)

    while(1);
    return 1;
}
```

例 60-12：数据宽度切换（32 位多项式、小尾数且 MOD 位 = 0）

```
// ASCII bytes "12345678"
volatile  unsigned long message1[] = {0x34333231,0x38373635};
// ASCII bytes "123"
volatile  unsigned char message2[] = {'1','2','3'};
volatile  unsigned long crcResultCRC32 = 0;
int       main(void)
{
unsigned  char*  pointer8;
unsigned  long* pointer32;
unsigned  short  length;
#define   CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define   CRC32_SEED_VALUE ((unsigned long)0x46AF6449)    // non-direct initial value of 0xFFFFFFFF

    CRCCON = 0;
    CRCCONbits.MOD = 0;                                   // alternate mode
    CRCCONbits.ON = 1;                                    // enable CRC
    CRCCONbits.CRCISEL = 0;                               // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                               // little endian
    CRCCONbits.DWIDTH = 32-1;                             // 32-bit data width
    CRCCONbits.PLEN = 32-1;                               // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                            // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                           // set initial value
    CRCCONbits.CRCGO = 1;                                 // start CRC calculation

    pointer32 = (unsigned long*)message1;
    length = sizeof(message1)/sizeof(unsigned long)-1;
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                         // wait if FIFO is full
        CRCDAT = *pointer32++;                            // 32-bit word access to FIFO
    }
    while(CRCCONbits.CRCFUL);                             // wait if FIFO is full
    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear interrupt flag
    CRCDAT = *pointer32;                                  // write last 32-bit data into FIFO
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done

    CRCCONbits.DWIDTH = 8-1;                              // switch the data width to 8-bit
    pointer8 = (unsigned char*)message2;                  // calculate CRC
    length = sizeof(message2)/sizeof(unsigned char)-1;
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                         // wait if FIFO is full
        *((unsigned char*)&CRCDAT) = *pointer8++;         // byte access to FIFO
    }
    while(CRCCONbits.CRCFUL);                             // wait if FIFO is full
    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear interrupt flag
    *((unsigned char*)&CRCDAT) = *pointer8;               // write last 8-bit data into FIFO
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done
    CRCCONbits.DWIDTH = 32-1;                             // switch data width to polynomial length 32-bit

    CRCCONbits.CRCGO = 0;                                 // suspend CRC calculation
    IFS0CLR = _IFS0_CRCIF_MASK;                           // clear interrupt flag
    CRCDAT = 0;                                           // write dummy data to shift out the CRC result
    CRCCONbits.CRCGO = 1;                                 // resume CRC calculation
    while(!IFS0bits.CRCIF);                               // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                             // get the final CRC result (must be 0xE092727E)

    while(1);
    return 1;
}
```

60.6.4 带有持久中断的 CRC 模块应用示例

对于使用持久中断的模块，例 60-13 至例 60-22 给出了各种不同组合的多项式长度、数据宽度、移位方向和 CRC 引擎模式的典型代码。

例 60-13：CRC-SMBus（具有 32 位数据、大尾数且 MOD 位 = 1 的 8 位多项式）

```
// This macro is used to swap bytes for big endian
#define  Swap(x) __extension__({ \
unsigned long __x = (x), __v; \
__asm__  ("wsbh %0,%1;\n\t" \
"rotr %0,16" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})

// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
volatile unsigned char crcResultCRCSMBUS = 0;
int      main (void)
{
unsigned long*  pointer;
unsigned short length;
unsigned long  data;

// standard CRC-SMBUS

#define  CRCSMBUS_POLYNOMIAL ((unsigned long)0x00000007)
#define  CRCSMBUS_SEED_VALUE ((unsigned long)0x00000000) // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                  // alternate mode
    CRCCONbits.ON = 1;                                   // enable CRC
    CRCCONbits.LENDIAN = 0;                              // big endian
    CRCCONbits.CRCISEL = 0;                              // interrupt when all shifts are done
    CRCCONbits.DWIDTH = 32-1;                            // 32-bit data width
    CRCCONbits.PLEN = 8-1;                               // 8-bit polynomial order
    CRCXOR = CRCSMBUS_POLYNOMIAL;                        // set polynomial
    CRCWDAT = CRCSMBUS_SEED_VALUE;                       // set initial value
    CRCCONbits.CRCGO = 1;                                // start CRC calculation

    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                        // wait if FIFO is full
        data = *pointer++;                               // load from little endian
        data = Swap(data);                               // swap bytes for big endian
        length--;
        if(length == 0)
        {
          break;
        }
        CRCDAT = data;                                   // 32-bit word access to FIFO
    }
    CRCDAT = data;                                       // write last data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                          // clear the interrupt flag

    while(!IFS0bits.CRCIF);                              // wait until shifts are done
    crcResultCRCSMBUS = (unsigned char)CRCWDAT&0x00ff;   // get CRC result (must be 0xC7)

    while(1);
    return 1;
}
```

例 60-14：CRC-SMBus（具有 32 位数据、小尾数且 MOD 位 = 0 的 8 位多项式）

```
// This macro is used to swap bytes for big endian
#define  Swap(x) __extension__({ \
unsigned long __x = (x), __v; \
__asm__  ("wsbh %0,%1;\n\t" \
"rotr %0,16" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})

// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
volatile unsigned char crcResultCRCSMBUS = 0;
int      main (void)
{
unsigned long* pointer;
unsigned short length;
unsigned long  data;

// standard CRC-SMBUS

#define  CRCSMBUS_POLYNOMIAL ((unsigned long)0x00000007)
#define  CRCSMBUS_SEED_VALUE ((unsigned long)0x00000000) // non-direct initial value of zero

    CRCCON = 0;
    CRCCONbits.MOD = 0;                                 // legacy mode
    CRCCONbits.ON = 1;                                  // enable CRC
    CRCCONbits.LENDIAN = 0;                             // big endian
    CRCCONbits.CRCISEL = 0;                             // interrupt when all shifts are done
    CRCCONbits.DWIDTH = 32-1;                           // 32-bit data width
    CRCCONbits.PLEN = 8-1;                              // 8-bit polynomial order
    CRCXOR = CRCSMBUS_POLYNOMIAL;                       // set polynomial
    CRCWDAT = CRCSMBUS_SEED_VALUE;                      // set initial value
    CRCCONbits.CRCGO = 1;                               // start CRC calculation
    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                       // wait if FIFO is full
        data = *pointer++;                              // load from little endian
        data = Swap(data);                              // swap bytes for big endian
        length--;
        if(length == 0)
        {
         break;
        }
        CRCDAT = data;                                  // 32-bit word access to FIFO
    }

    CRCDAT = data;                                      // write last data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                         // clear the interrupt flag
    while(!IFS0bits.CRCIF);                             // wait until shifts are done
    CRCCONbits.DWIDTH = 8-1;                            // switch data width to polynomial length 8-bit
    *((unsigned char*)&CRCDAT) = 0;                     // 8-bit dummy data to shift out the CRC result
    IFS0CLR = _IFS0_CRCIF_MASK;                         // clear the interrupt flag
    while(!IFS0bits.CRCIF);                             // wait until shifts are done
    crcResultCRCSMBUS = (unsigned char)CRCWDAT&0x00ff;  // get CRC result (must be 0xC7)

    while(1);
    return 1;
}
```

例 60-15：CRC-16（具有 32 位多项式、小尾数且 MOD 位 = 1 的 16 位数据）

```
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRC16 = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-16

#define  CRC16_POLYNOMIAL ((unsigned long)0x00008005)
#define  CRC16_SEED_VALUE ((unsigned long)0x00000000) // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                 // alternate mode
    CRCCONbits.ON = 1;                                  // enable CRC
    CRCCONbits.CRCISEL = 0;                             // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                             // little endian
    CRCCONbits.DWIDTH = 16-1;                           // 16-bit data width
    CRCCONbits.PLEN = 16-1;                             // 16-bit polynomial order
    CRCXOR = CRC16_POLYNOMIAL;                          // set polynomial
    CRCWDAT = CRC16_SEED_VALUE;                         // set initial value
    CRCCONbits.CRCGO = 1;                               // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                       // wait if FIFO is full
        data = *pointer++;                              // load data
        length--;
        if(length == 0)
        {
          break;
        }
        *((unsigned short*)&CRCDAT) = data;             // 16-bit word access to FIFO
    }

    *((unsigned short*)&CRCDAT) = data;                 // write last data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                         // clear the interrupt flag
    while(!IFS0bits.CRCIF);                             // wait until shifts are done
    crcResultCRC16 = (unsigned short)CRCWDAT;           // get CRC result (must be 0xE716)

    while(1);
    return 1;
}
```

例 60-16：CRC-16（具有 16 位多项式、小尾数且 MOD 位 = 0 的 16 位数据）

```
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRC16 = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-16

#define  CRC16_POLYNOMIAL ((unsigned long)0x00008005)
#define  CRC16_SEED_VALUE ((unsigned long)0x00000000) // non-direct initial value of zero

    CRCCON = 0;
    CRCCONbits.MOD = 0;                          // legacy mode
    CRCCONbits.ON = 1;                           // enable CRC
    CRCCONbits.CRCISEL = 0;                      // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                      // little endian
    CRCCONbits.DWIDTH = 16-1;                    // 16-bit data width
    CRCCONbits.PLEN = 16-1;                      // 16-bit polynomial order
    CRCXOR = CRC16_POLYNOMIAL;                   // set polynomial
    CRCWDAT = CRC16_SEED_VALUE;                  // set initial value
    CRCCONbits.CRCGO = 1;                        // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                // wait if FIFO is full
        data = *pointer++;                       // load data
        *((unsigned short*)&CRCDAT) = data;      // 16-bit word access to FIFO
    }

    while(CRCCONbits.CRCFUL);                    // wait if FIFO is full
    *((unsigned short*)&CRCDAT) = 0;             // 16-bit dummy data to shift out CRC result
    IFS0CLR = _IFS0_CRCIF_MASK;                  // clear the interrupt flag
    while(!IFS0bits.CRCIF);                      // wait until shifts are done
    crcResultCRC16 = (unsigned short)CRCWDAT;    // get CRC result (must be 0xE716)

    while(1);
    return 1;
}
```

例 60-17：CRC-CCITT（具有 16 位数据、大尾数且 MOD 位 = 1 的 16 位多项式）

```
// This macro is used to swap bytes for big endian
#define  Swap(x) __extension__({ \
unsigned long __x = (x), __v; \
__asm__  ("wsbh %0,%1;\n\t" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRCCCITT = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-CCITT

#define  CRCCCITT_POLYNOMIAL ((unsigned long)0x00001021)
#define  CRCCCITT_SEED_VALUE ((unsigned long)0x0000FFFF) // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                 // alternate mode
    CRCCONbits.ON = 1;                                  // enable CRC
    CRCCONbits.CRCISEL = 0;                             // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 0;                             // big endian
    CRCCONbits.DWIDTH = 16-1;                           // 16-bit data width
    CRCCONbits.PLEN = 16-1;                             // 16-bit polynomial order
    CRCXOR = CRCCCITT_POLYNOMIAL;                       // set polynomial
    CRCWDAT = CRCCCITT_SEED_VALUE;                      // set initial value
    CRCCONbits.CRCGO = 1;                               // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                       // wait if FIFO is full
        data = *pointer++;                              // load data
        data = Swap(data);                              // swap bytes for big endian
        length--;
        if(length == 0)
        {
         break;
        }
        *((unsigned short*)&CRCDAT) = data;             // 16-bit word access to FIFO
    }

    *((unsigned short*)&CRCDAT) = data;                 // write last data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                         // clear the interrupt flag
    while(!IFS0bits.CRCIF);                             // wait until shifts are done
    crcResultCRCCCITT = (unsigned short)CRCWDAT;        // get CRC result (must be 0x9B4D)

    while(1);
    return 1;
}
```

例 60-18：CRC-CCITT（具有 16 位数据、大尾数且 MOD 位 = 0 的 16 位多项式）

```
// This macro is used to swap bytes for big endian
#define  Swap(x) __extension__({ \
unsigned long __x = (x), __v; \
__asm__  ("wsbh %0,%1;\n\t" \
: "=d" (__v) \
: "d" (__x)); \
__v; \
})
// ASCII bytes "87654321"
volatile unsigned short message[] = {0x3738,0x3536,0x3334,0x3132};
volatile unsigned short crcResultCRCCCITT = 0;
int      main (void)
{
unsigned short* pointer;
unsigned short  length;
unsigned short  data;

// standard CRC-CCITT

#define  CRCCCITT_POLYNOMIAL ((unsigned long)0x00001021)
#define  CRCCCITT_SEED_VALUE ((unsigned long)0x000084CF) // non-direct initial value of 0xFFFF

    CRCCON = 0;
    CRCCONbits.MOD = 0;                                 // legacy mode
    CRCCONbits.ON = 1;                                  // enable CRC
    CRCCONbits.CRCISEL = 0;                             // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 0;                             // big endian
    CRCCONbits.DWIDTH = 16-1;                           // 16-bit data width
    CRCCONbits.PLEN = 16-1;                             // 16-bit polynomial order
    CRCXOR = CRCCCITT_POLYNOMIAL;                       // set polynomial
    CRCWDAT = CRCCCITT_SEED_VALUE;                      // set initial value
    CRCCONbits.CRCGO = 1;                               // start CRC calculation

    pointer = (unsigned short*)message;
    length = sizeof(message)/sizeof(unsigned short);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                       // wait if FIFO is full
        data = *pointer++;                              // load data
        data = Swap(data);                              // swap bytes for big endian
        *((unsigned short*)&CRCDAT) = data;             // 16-bit word access to FIFO
    }

    while(CRCCONbits.CRCFUL);                           // wait if FIFO is full
    *((unsigned short*)&CRCDAT) = 0;                    // 16-bit dummy data to shift out CRC result
    IFS0CLR = _IFS0_CRCIF_MASK;                         // clear the interrupt flag
    while(!IFS0bits.CRCIF);                             // wait until shifts are done
    crcResultCRCCCITT = (unsigned short)CRCWDAT;        // get CRC result (must be 0x9B4D)

    while(1);
    return 1;
}
```

例 60-19：CRC-32（具有 32 位数据、小尾数且 MOD 位 = 1 的 32 位多项式）

```
// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
// function to reverse the bit order (OPTIONAL)
unsigned long ReverseBitOrder(unsigned long data);
volatile unsigned int crcResultCRC32 = 0;
int      main(void)
{
unsigned long*  pointer;
unsigned short  length;

// standard CRC-32

#define  CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define  CRC32_SEED_VALUE ((unsigned long)0xFFFFFFFF)    // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                                  // alternate mode
    CRCCONbits.ON = 1;                                   // enable CRC
    CRCCONbits.CRCISEL = 0;                              // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                              // little endian
    CRCCONbits.DWIDTH = 32-1;                            // 32-bit data width
    CRCCONbits.PLEN = 32-1;                              // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                           // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                          // set initial value
    CRCCONbits.CRCGO = 1;                                // start CRC calculation
    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                        // wait if FIFO is full
        length--;
        if(length == 0)
        {
         break;
        }
        CRCDAT = *pointer++;                             // 32-bit word access to FIFO
    }
    CRCDAT = *pointer;                                   // write last data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                          // clear the interrupt flag
    while(!IFS0bits.CRCIF);                              // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                            // get the final CRC result
    // OPTIONAL reverse CRC value bit order and invert (must be 0x9AE0DAAF)
    crcResultCRC32 = ~ReverseBitOrder(crcResultCRC32);
    while(1);
    return 1;
}
unsigned long ReverseBitOrder(unsigned long data)
{
unsigned long maskin;
unsigned long maskout;
unsigned long result = 0;
unsigned char i;
maskin = 0x80000000;
maskout = 0x00000001;
    for(i=0; i<32; i++)
    {
        if(data&maskin){
            result |= maskout;
        }
        maskin >>= 1;
        maskout <<= 1;
    }
return   result;
}
```

例 60-20：CRC-32（具有 32 位数据、小尾数且 MOD 位 = 0 的 32 位多项式）

```
// ASCII bytes "12345678"
volatile unsigned char __attribute__((aligned(4))) message[] = {'1','2','3','4','5','6','7','8'};
// function to reverse the bit order (OPTIONAL)
unsigned long ReverseBitOrder(unsigned long data);
volatile unsigned int crcResultCRC32 = 0;
int      main(void)
{
unsigned long*  pointer;
unsigned short  length;

// standard CRC-32

#define  CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define  CRC32_SEED_VALUE ((unsigned long)0x46AF6449) // non-direct initial value of 0xFFFFFFFF

    CRCCON = 0;
    CRCCONbits.MOD = 1;                          // alternate mode
    CRCCONbits.ON = 1;                           // enable CRC
    CRCCONbits.CRCISEL = 0;                      // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                      // little endian
    CRCCONbits.DWIDTH = 32-1;                    // 32-bit data width
    CRCCONbits.PLEN = 32-1;                      // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                   // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                  // set initial value
    CRCCONbits.CRCGO = 1;                        // start CRC calculation

    pointer = (unsigned long*)message;
    length = sizeof(message)/sizeof(unsigned long);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                // wait if FIFO is full
        CRCDAT = *pointer++;                     // 32-bit word access to FIFO
    }
    while(CRCCONbits.CRCFUL);                    // wait if FIFO is full
    CRCDAT = 0;                                  // 32-bit dummy data to shift out the CRC result
    IFS0CLR = _IFS0_CRCIF_MASK;                  // clear the interrupt flag
    while(!IFS0bits.CRCIF);                      // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                    // get the final CRC result
    // OPTIONAL reverse CRC value bit order and invert (must be 0x9AE0DAAF)
    crcResultCRC32 = ~ReverseBitOrder(crcResultCRC32);
    while(1);
    return 1;
}

unsigned long ReverseBitOrder(unsigned long data)
{
unsigned long maskin;
unsigned long maskout;
unsigned long result = 0;
unsigned char i;
maskin = 0x80000000;
maskout = 0x00000001;
    for(i=0; i<32; i++)
    {
        if(data&maskin){
               result |= maskout;
        }
        maskin >>= 1;
        maskout <<= 1;
    }
return   result;
}
```

例 60-21：数据宽度切换 （32 位多项式、小尾数且 MOD 位 = 1）

```
// ASCII bytes "12345678"
volatile unsigned long message1[] = {0x34333231,0x38373635};
// ASCII bytes "123"
volatile unsigned char message2[] = {'1','2','3'};
volatile unsigned long crcResultCRC32 = 0;
int      main(void)
{
unsigned char*  pointer8;
unsigned long* pointer32;
unsigned short  length;
#define   CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define   CRC32_SEED_VALUE ((unsigned long)0xFFFFFFFF) // direct initial value

    CRCCON = 0;
    CRCCONbits.MOD = 1;                             // alternate mode
    CRCCONbits.ON = 1;                              // enable CRC
    CRCCONbits.CRCISEL = 0;                         // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                         // little endian
    CRCCONbits.DWIDTH = 32-1;                       // 32-bit data width
    CRCCONbits.PLEN = 32-1;                         // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                      // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                     // set initial value
    CRCCONbits.CRCGO = 1;                           // start CRC calculation

    pointer32 = (unsigned long*)message1;
    length = sizeof(message1)/sizeof(unsigned long);
    while(1)
    {
        while(CRCCONbits.CRCFUL);                   // wait if FIFO is full
        length--;
        if(length == 0)
        {
          break;
        }
        CRCDAT = *pointer32++;                      // 32-bit word access to FIFO
    }
    CRCDAT = *pointer32;                            // write last 32-bit data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                     // clear the interrupt flag
    while(!IFS0bits.CRCIF);                         // wait until shifts are done
    CRCCONbits.DWIDTH = 8-1;                        // switch the data width to 8-bit

    pointer8 = (unsigned char*)message2;            // calculate CRC
    length = sizeof(message2)/sizeof(unsigned char);
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                   // wait if FIFO is full
        length--;
        if(length == 0)
        {
          break;
        }
        *((unsigned char*)&CRCDAT) = *pointer8++;   // byte access to FIFO
    }
    *((unsigned char*)&CRCDAT) = *pointer8;         // write last 8-bit data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                     // clear the interrupt flag
    while(!IFS0bits.CRCIF);                         // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                       // get the final CRC result (must be 0xE092727E)

    while(1);
    return 1;
}
```

例 60-22：数据宽度切换（32 位多项式、小尾数且 MOD 位 = 0）

```
// ASCII bytes "12345678"
volatile unsigned long message1[] = {0x34333231,0x38373635};
// ASCII bytes "123"
volatile unsigned char message2[] = {'1','2','3'};
volatile unsigned long crcResultCRC32 = 0;
int      main(void)
{
unsigned char*  pointer8;
unsigned long* pointer32;
unsigned short  length;
#define  CRC32_POLYNOMIAL ((unsigned long)0x04C11DB7)
#define  CRC32_SEED_VALUE ((unsigned long)0x46AF6449)  // non-direct initial value of 0xFFFFFFFF

    CRCCON = 0;
    CRCCONbits.MOD = 0;                             // alternate mode
    CRCCONbits.ON = 1;                              // enable CRC
    CRCCONbits.CRCISEL = 0;                         // interrupt when all shifts are done
    CRCCONbits.LENDIAN = 1;                         // little endian
    CRCCONbits.DWIDTH = 32-1;                       // 32-bit data width
    CRCCONbits.PLEN = 32-1;                         // 32-bit polynomial order
    CRCXOR = CRC32_POLYNOMIAL;                      // set polynomial
    CRCWDAT = CRC32_SEED_VALUE;                     // set initial value
    CRCCONbits.CRCGO = 1;                           // start CRC calculation

    pointer32 = (unsigned long*)message1;
    length = sizeof(message1)/sizeof(unsigned long)-1;
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                   // wait if FIFO is full
        CRCDAT = *pointer32++;                      // 32-bit word access to FIFO
    }
    while(CRCCONbits.CRCFUL);                       // wait if FIFO is full
    CRCDAT = *pointer32;                            // write last 32-bit data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                     // clear interrupt flag
    while(!IFS0bits.CRCIF);                         // wait until shifts are done

    CRCCONbits.DWIDTH = 8-1;                        // switch the data width to 8-bit
    pointer8 = (unsigned char*)message2;            // calculate CRC
    length = sizeof(message2)/sizeof(unsigned char)-1;
    while(length--)
    {
        while(CRCCONbits.CRCFUL);                   // wait if FIFO is full
        *((unsigned char*)&CRCDAT) = *pointer8++;   // byte access to FIFO
    }
    while(CRCCONbits.CRCFUL);                       // wait if FIFO is full
    *((unsigned char*)&CRCDAT) = *pointer8;         // write last 8-bit data into FIFO
    IFS0CLR = _IFS0_CRCIF_MASK;                     // clear interrupt flag
    while(!IFS0bits.CRCIF);                         // wait until shifts are done
    CRCCONbits.DWIDTH = 32-1;                       // switch data width to polynomial length 32-bit

    CRCDAT = 0;                                     // write dummy data to shift out the CRC result
    IFS0CLR = _IFS0_CRCIF_MASK;                     // clear interrupt flag
    while(!IFS0bits.CRCIF);                         // wait until shifts are done
    crcResultCRC32 = CRCWDAT;                       // get the final CRC result (must be 0xE092727E)

    while(1);
    return 1;
}
```

60.7 节能模式下的操作

60.7.1 休眠模式

如果模块工作时进入休眠模式，模块将暂停于当前状态，直到时钟恢复运行。

60.7.2 空闲模式

要在空闲模式下继续使模块全功能工作， SIDL 位必须在进入该模式前清零。

如果 SIDL = 1，该模块的行为与在休眠模式下相同；将传递待处理的中断事件，即使模块时钟不可用。

60.8 复位的影响

在器件复位后，会将所有 CRC 控制寄存器 （CRCCON、 CRCXOR 和 CRCWDAT）都复位为 0x00000000。这会禁止 CRC 模块。正在进行中的所有计算都将终止。

60.9 相关应用笔记

本节列出了与手册本章内容相关的应用笔记。这些应用笔记可能并不是专为 PIC32 器件系列而编写的，但其概念是相近的，通过适当修改并受到一定限制即可使用。当前与 32 位可编程循环冗余校验（CRC）相关的应用笔记有：

标题	应用笔记编号

目前没有相关的应用笔记。

注： 如需获取更多 PIC32 系列器件的应用笔记和代码示例，请访问 Microchip 网站（www.microchip.com）。

60.10 版本历史

版本 A （2015 年 5 月）

这是本文档的初始版本。

版本 B （2016 年 3 月）

- 图：
  - 更新了图 60-1。
- 示例：
  - 更新了例 60-2 和例 60-3 至例 60-12。

版本 C （2016 年 6 月）

更新了**第 60.5.4.4 节 “CRC 结果”**和**第 60.5.5 节 “中断操作”**。

增加了**第 60.6.4 节 “带有持久中断的 CRC 模块应用示例”**。

注：

请注意以下有关 Microchip 器件代码保护功能的要点：

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

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

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

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

ISBN：978-1-5224-2375-1

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