AN4187
应用笔记
在STM32系列中使用CRC外设

引言

循环冗余校验（CRC）技术用于检测数字数据中的错误，但在检测到错误时不进行校正。该技术旨在检查数据传输或数据存储的完整性。在确保数据可靠性方面，CRC技术功能强大且易于实施。该技术的诊断覆盖范围可满足基本安全标准要求。这就是认证符合IEC 60335-1和 IEC 607030-1 标准（称为“B 级”要求）的ST固件时，在Flash内容完整性自检检查中使用CRC功能的原因。更多信息，请参阅应用笔记AN3307和适用于不同系列产品的相关固件包。当CRC校验和信息必须由链接器直接放入用户代码中时（主要以CRC描述符数据表的格式），建议在编译器手册中检查所有必要的CRC设置。

CRC以多项式算法为基础。计算GF（2）中一个多项式除以另一个多项式的余数。余数被称为校验和，而被除数是数据，除数是生成多项式。

注： *GF（2）（两元素伽罗瓦域）中的多项式是单变量x的多项式，其系数为0或1。*

本应用笔记描述了嵌入在所有STM32系列（F0、F1、F2、F3、F4、L1）中的循环冗余校验外设的功能，以及配置该外设所需的步骤。

本应用笔记的结构如下：

- *第 1 节*介绍了STM32 CRC实现算法及其硬件实现优势。
- *第 2 节*描述了DMA作为CRC数据传输控制器的用法。
- *第 3 节*描述了通过STM32器件移植CRC。

此外还提供了两个示例：

- CRC_usage示例：如何将CPU用作数据传输控制器来配置CRC。
- CRC_DMA示例：如何将DMA用作CRC数据传输控制器。

两个例子均支持执行时间测量。

表1. 适用产品

类型	产品类别
微控制器	STM32

目录

表格索引

图片索引

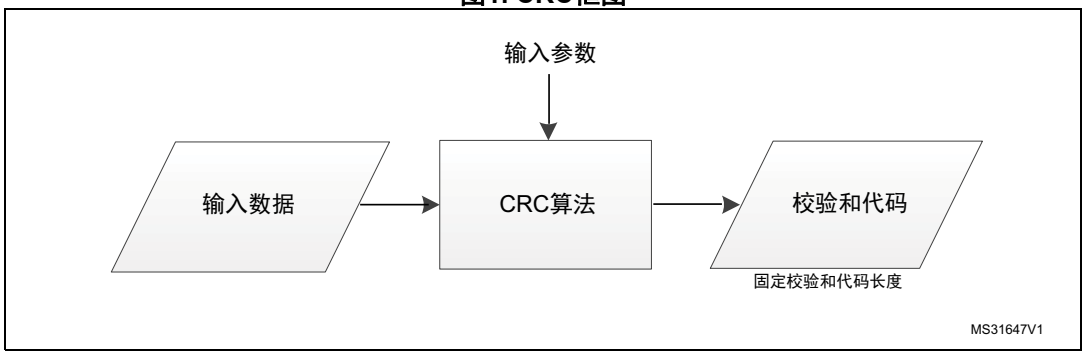
1 CRC外设概览

嵌入在所有STM32微控制器件中的CRC外设用于提供任意支持数据类型的CRC校验和代码。

1.1 CRC算法

如*图 1*中所示，CRC算法有一个数据输入，并根据输入参数生成固定的校验和代码长度。

图1. CRC框图



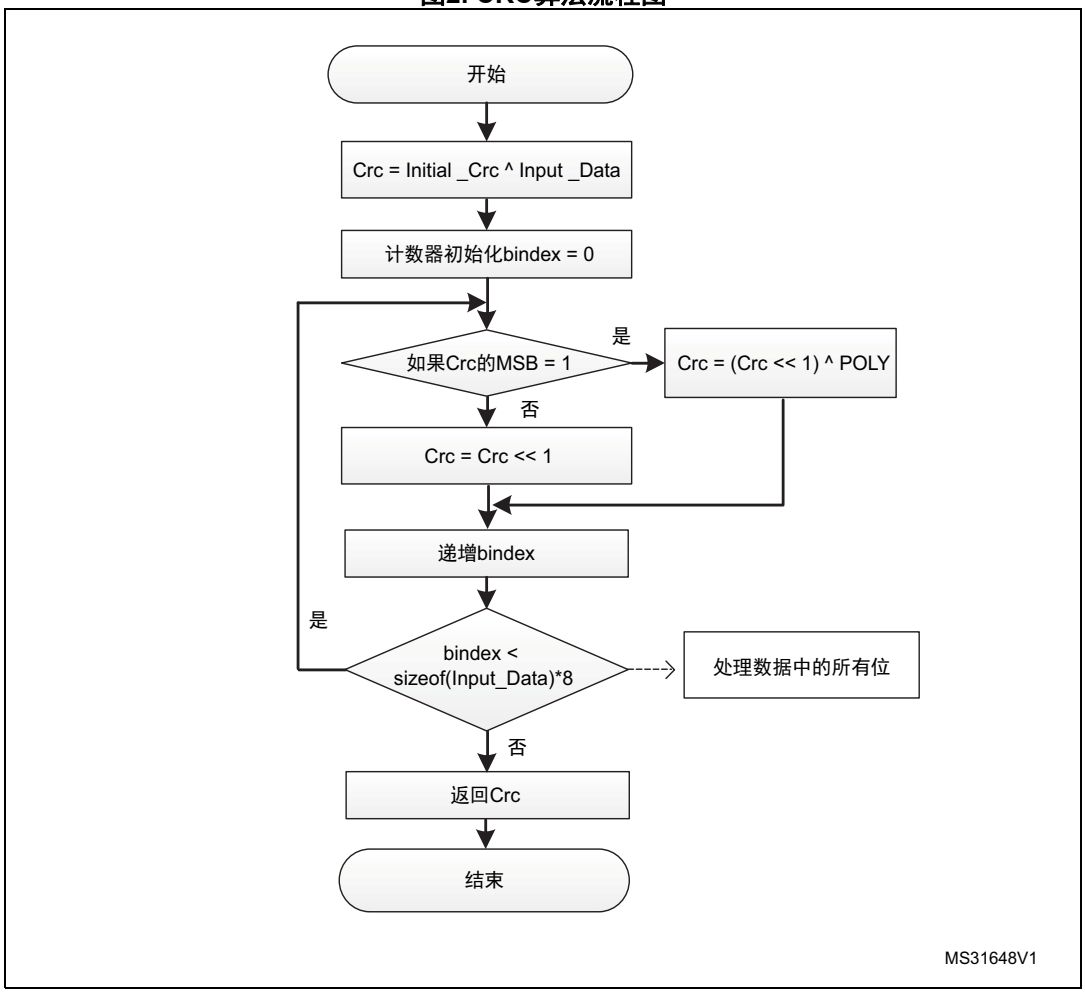
已知的CRC算法之一是使用按位消息XORing技术进行多项式除法。这是微控制器中使用的最适合硬件或底层实现的技术。

该算法的输入参数为：

- 被除数：也称为输入数据, 缩写为 *“Input_Data”*
- 除数：也称为生成多项式，缩写为“*POLY*”
- 初始CRC值：缩写为*Initial_Crc*”

下文的*图 2*显示了CRC算法流程图。

图2. CRC算法流程图



启动时，算法将CRC设置为带有*Input_Data*的*Initial_Crc* XOR。

一旦CRC MSB等于1，该算法将CRC向左移一位，并与POLY进行XOR。否则，只会将CRC向左移动一位。

*图 3*显示了以下条件的逐步算法执行：

- Input_Data = 0xC1
- POLY = 0xCB
- Initial_Crc = 0xFF

图3. 逐步CRC计算示例

bindex	执行步骤	二进制格式	十六进制
	Crc = *Initial_Crc* ^ *Input_Data*	1 1 1 1 1 1 1 1 (*Initial_Crc*) ^ 1 1 0 0 0 0 0 1 (*Input_Data*)	0xFF 0xC1
0	Crc << 1	[0] 0 1 1 1 1 1 0	0x3E
1	Crc << 1	[0] 1 1 1 1 1 0 0	0x7C
2	Crc << 1	[1] 1 1 1 1 0 0 0	0xF8
	Crc = Crc ^ POLY	[1] 1 1 1 0 0 0 0 ^ 1 1 0 0 1 0 1 1 (*POLY*)	0xF0 0xCB
3	Crc << 1	[0] 0 1 1 1 0 1 1	0x3B
4	Crc << 1	0 1 1 1 0 1 1 0	0x76
5	Crc << 1	[1] 1 1 0 1 1 0 0	0xEC
	Crc = Crc ^ POLY	[1] 1 0 1 1 0 0 0 ^ 1 1 0 0 1 0 1 1 (*POLY*)	0xD8 0xCB
6	Crc << 1	[0] 0 0 1 0 0 1 1	0x13
7	Crc << 1	[0] 0 1 0 0 1 1 0	0x26
	Crc（返回值）	0 1 0 0 1 1 0 0	0x4C

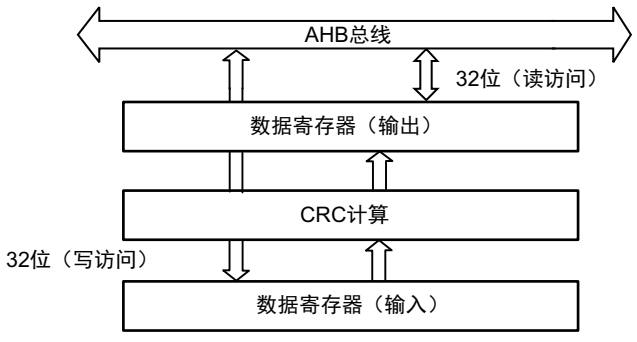
1. 返回的CRC值（0x4C）已由STM32F37x系列中的CRC外设，通过上述配置进行验证。
2. 该程序已在软件包中CRC_usage示例里的CrcSoftwareFunc函数下实现。

1.2 CRC外设配置

所有STM32器件均采用CRC外设，如*第 1.1 节*中所述。CRC计算单元具有单个32位读/写数据寄存器（CRC_DR)。它用于输入新数据（写访问）并保存以前的CRC计算结果（读访问）。

对数据寄存器的每个写操作都会对之前的CRC值（存储在CRC_DR中）和新值再做一次CRC计算。

图4. CRC计算单元框图



要计算任何支持数据的CRC，必须遵循以下步骤：

1. 通过RCC外设启用CRC外设时钟。
2. 通过配置初始CRC值寄存器（CRC_INIT），将CRC数据寄存器设置为初始CRC值 。(a)
3. 分别通过CRC控制寄存器（CRC_CR）中的REV_IN[1:0]和REV_OUT位设置I/O反转位顺序。(a)
4. 分别通过CRC控制寄存器（CRC_CR）和CRC多项式寄存器（CRC_POL）中的POLYSIZE[1:0]位设置多项式大小和系数。(b)
5. 通过CRC控制寄存器（CRC_CR）中的Reset位重置CRC外设。
6. 将数据设置到CRC数据寄存器。
7. 读取CRC数据寄存器的内容。
8. 禁用CRC外设时钟。

在固件包中，CRC_usage示例运行CRC校验和代码，计算256个支持数据类型的数组数据（*DataBuffer*）。有关完整说明，请参阅CRC_usage文件夹中的文件*Readme.txt* 。

a. 仅适用于STM32F0xx和STM32F3xx器件

b. 仅适用于STM32F3xx器件

1.3 CRC硬件实现优势

CRC_usage示例用于检查软件CRC算法实现与CRC外设之间的兼容性，并测量其执行时间。

该示例已在以下条件下执行：

- 硬件：STM32373C-EVAL板（STM32F37x器件）
- 系统时钟：HSE（8 MHz晶体振荡器）
- 工具链：Keil V4.60.0.0
- CRC配置：CRC寄存器的默认值
  CRC_CR：0x0000 0000；POLYSIZE为32，无REV_IN和REV_OUT
  CRC_INIT：0XFFFF FFFF
  CRC_POLY：0X04D11 CDB7
- 输入数据：256字

*表 2*显示了CRC算法与STM32 CRC外设执行时间的比较结果。

表2. CRC算法与CRC外设执行时间的比较

优化级别	CRC算法（系统时钟周期）	CRC外设（系统时钟周期）
3级+优化时间	78094	1287

CRC的硬件实现速度大约是软件算法的60倍。CPU控制数据传输，在此阶段不允许任何指令处理。应用程序开发人员可以选择另一个外设作为控制器，以便将CPU用于其他任务。由于DMA能够管理数据传输，因此成为计算数据缓冲区CRC校验和代码的替代选择。

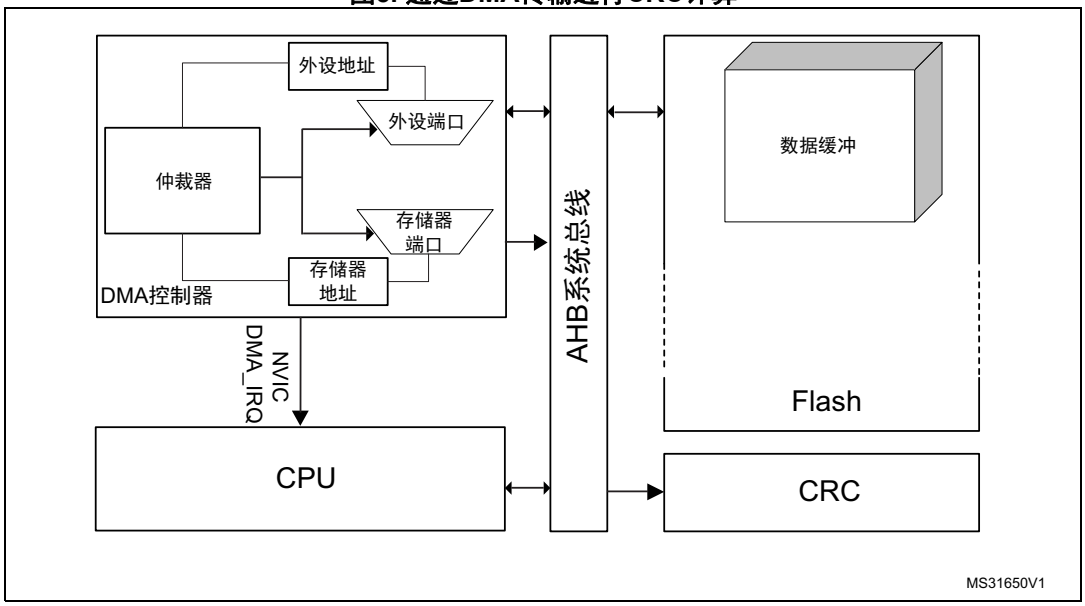
2 通过DMA使用CRC

STM32嵌入式DMA可用作数据传输处理器，以避免将CPU用作控制器。DMA配置取决于所选的体系结构。具体有两类：带通道的DMA和带数据流的DMA。两种DMA架构的配置几乎相同。

2.1 DMA背靠背传输模式

固件包中提供的CRC_DMA示例实现了*图 5*中所示的技术。这种技术是带DMA_IRQ处理程序回调的DMA背靠背数据传输模式。

图5. 通过DMA传输进行CRC计算



在这种情况下，DMA控制数据传输计数器，并等待传输完成标志执行NVIC DMA_IRQ处理程序。NVIC DMA_IRQ例程必须停止系统计时器（systick）计数器并返回CRC计算值。CPU的使用仅限于DMA中断请求程序的执行。

2.2 DMA配置

如上所述，STM32设备集成了多种DMA架构。以下配置步骤对于两种DMA架构都很常见：

1. 通过RCC外设启用CRC外设时钟。
2. 配置DMA通道/数据流（有关这两种配置，请参阅*表 3*）。
3. 按照*第 1.2 节*的前五个步骤配置CRC。
4. 启用DMA传输完成中断。
5. 配置DMA NVIC IRQ。
6. 启用DMA通道/数据流。
7. 等待DMA传输完成。
8. 禁用DMA通道。在DMA采用数据流构架的情况下，控制器于传输结束时自动禁用通道，而在另一种构架情况下，NVIC DMA_IRQ程序必须禁用通道以供进一步使用。
9. 清除DMA IT挂起位。

注： *有关任何其他信息，请参阅CRC_DMA示例文件夹下的文件Readme.txt。*

表3. DMA配置总结

DMA配置	带通道的DMA	带数据流的DMA
传输方向	存储器到存储器	
外设地址	Flash基指针	
存储器地址	CRC数据寄存器	
外设地址递增	启用	
存储器地址递增	禁用	
缓存大小	数据缓存大小	
外设数据大小	支持的数据类型（字节、半字或字）	
存储器数据大小	支持的数据类型（字节、半字或字）	
传输模式	正常	
外设突发	NA	单
存储器突发	NA	单
FIFO模式(1)	NA	禁用
FIFO阈值(1)	NA	--

1. 启用FIFO模式不会影响执行时间。因此，FIFO模式和FIFO阈值配置无影响。

开发CRC_DMA示例旨在检查使用CPU和DMA作为传输处理程序之间的兼容性结果，并测量使用DMA作为传输处理程序期间的CPU负载。

2.3 DMA的使用优势

在DMA在背靠背数据传输模式期间，CPU仅在CRC和DMA配置期间以及DMA中断处理程序执行期间进行干预。

该示例的执行条件与*第 1.3 节*中列出的相同，只是输入数据大小变为支持的数据类型的8192。

表4. CPU与DMA执行时间使用情况的比较结果

传输处理器	外设配置(1)	CRC算法(1)	CPU负荷
CPU	66(2)	40962	100%
DMA	295(3)	40968	0.72%

1. systick timer tick是测量单位，systick时钟源是CPU时钟。
2. CRC配置时间
3. CRC配置、DMA配置和DMA IRQ处理程序执行时间

总体而言，需要注意的是，当CPU用作数据传输处理器时，CPU负载等于100%，而使用DMA时，CPU负载减少到0.72%。

3 通过STM32系列进行CRC移植

CRC外设的功能因STM32系列而异。*表 5*列出了CRC功能，并为每个STM32系列提供了软件兼容性分析。

表5. STM32 CRC外设功能

特征	F1系列	L1系列	F2 系列	F4 系列	F0 系列	F3 系列
单输入/输出 32 位数据寄存器	是					
通用 8位寄存器	是					
输入缓冲器可避免计算期间发生总线阻塞	否				是	
可逆性选项 关于I/O数据	否				是	
CRC初始值	固定为0XFFFFFFFF				32位可编程	8、16、32位可编程
处理的数据大小（位）	32					8, 16, 32
多项式大小（位）	32					7, 8, 16, 32
多项式系数	固定为0x4C11DB7					可编程

4 参考文档

- STM32F101xx、STM32F102xx、STM32F103xx、STM32F105xx和STM32F107xx基于ARM内核的32位高级MCU参考手册（RM0008）
- STM32F205xx、STM32F207xx、STM32F215xx和STM32F217xx基于ARM内核的32位高级MCU参考手册（RM0033）
- STM32L151xx、STM32L152xx和STM32L162xx基于ARM内核的32位高级MCU参考手册（RM0038）
- STM32F100xx基于ARM内核的32位高级MCU参考手册（RM0041）
- STM32F405xx、STM32F407xx、STM32F415xx和STM32F417xx基于ARM内核的32位高级MCU参考手册（RM0090）
- STM32F05xxx基于ARM内核的32位高级MCU参考手册（RM0091）
- STM32F37xx和STM32F38xx基于ARM内核的32位高级MCU参考手册（RM00313）
- STM32F302xx、STM32F303xx和STM32F313xx基于ARM内核的32位高级MCU参考手册（RM00316）
- STM32应用中获得IEC 60335 B级认证指南（AN3307）

注： *上述文件可通过以下URL获取：http://www.st.com/stm32*

5 版本历史

表6. 文档版本历史

日期	版本	变更
2013年6月6日	1	初始版本。

表7. 中文文档版本历史

日期	版本	变更
2022年4月10日	1	中文初始版本。