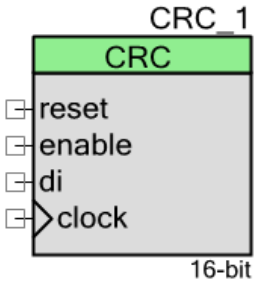
循环冗余校验 (Cyclic Redundancy Check, CRC)

2.40

特性



- 1 到 64 位
- 时分复用模式
- 需要时钟和数据以进行串行比特流输入
- 串行数据输入，并行结果
- 标准 [CRC-1（奇偶校验位）、CRC-4 (ITU-T G.704)、CRC-5-USB 等] 或自定义多项式
- 标准或自定义种子值
- 启用输入提供与其他组件的同步操作

概述

循环冗余校验 (CRC) 组件的默认用途是根据任意长度的串行比特流计算 CRC。在数据时钟的上升沿上对输入数据进行采样。在启动前，CRC 值复位为 0，或可用初始值作为种子值。完成比特流时，可读取计算出的 CRC 值。

何时使用 CRC

可使用默认 CRC 组件作为校验和，以在传输或存储过程中检测数据变化。CRC 很受欢迎，因为它们在二进制硬件中很容易实现，便于以数学方式进行分析，并特别适用于检测由传输通道中的噪声导致的常见错误。

输入/输出连接

本节介绍 CRC 的各种输入和输出连接。I/O 列表中的星号 (*) 表示，在 I/O 说明中列出的情况下，该 I/O 可能不可见。

时钟 — 输入

CRC 需要提供用于计算 CRC 的串行比特流的数据输入。同时需要数据时钟输入，以正确对串行数据输入进行采样。在数据时钟的上升沿上对输入数据进行采样。

复位 — 输入

复位输入用来定义要同步复位 CRC 的信号。

启用 — 输入

CRC 组件启动后，在使能输入保持高电平状态的情况下一直运行。此输入提供与其他组件的同步操作。

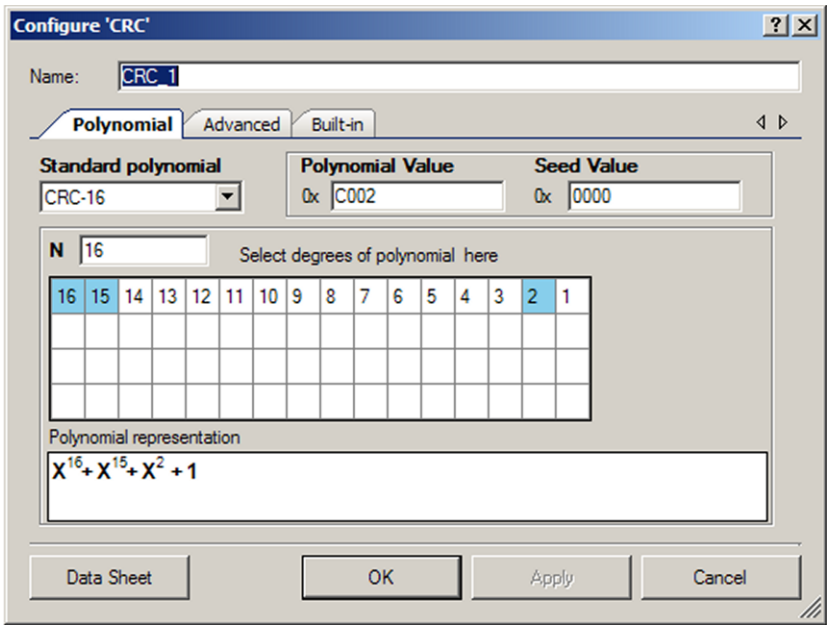
di — 输入

提供用于计算 CRC 的串行比特流的数据输入。

组件参数

将一个 CRC 组件拖放到您的设计上，并双击以打开 **Configure**（配置）对话框。该对话框有若干选项卡，可引导您完成 CRC 组件的设置过程。

“Polynomial”（多项式）选项卡



Standard Polynomial（标准多项式）

此参数允许您选择 **Standard polynomial**（标准多项式）组合框中的任意标准 CRC 多项式，或生成自定义多项式。在工具提示中给出有关各个标准多项式的更多信息。默认值为 **CRC-16**。

多项式名称	多项式	使用说明
客户	用户定义的	一般
CRC-1	x + 1	奇偶校验
CRC-4-ITU	$x^4 + x + 1$	ITU G.704
CRC-5-ITU	$x^5 + x^4 + x^2 + 1$	ITU G.704
CRC-5-USB	$x^5 + x^2 + 1$	USB
CRC-6-ITU	$x^6 + x + 1$	ITU G.704
CRC-7	$x^7 + x^3 + 1$	电信系统，MMC
CRC-8-ATM	$x^8 + x^2 + x + 1$	ATM HEC
CRC-8-CCITT	$x^8 + x^7 + x^3 + x^2 + 1$	1- 单线总线

多项式名称	多项式	使用说明
CRC-8-Maxim	$x^8 + x^5 + x^4 + 1$	1- 单线总线
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$	一般
CRC-8-SAE	$x^8 + x^4 + x^3 + x^2 + 1$	SAE J1850
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$	一般
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$	电信系统
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$	CAN
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$	XMODEM,X.25、V.41、Bluetooth、PPP、IrDA、CRC-CCITT
CRC-16	$x^{16} + x^{15} + x^2 + 1$	USB
CRC-24-Radix64	$x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1$	一般
CRC-32-IEEE802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	以太网，MPEG2
CRC-32C	$x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{+x}9^{+x}8^{+x}6^{+1}$	一般
CRC-32K	$x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^7 + x^{+x}4^{+x}2^{+x+1}$	一般
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$	ISO 3309
CRC-64-ECMA	$x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{+x}35^{+x}33^{+x}32^{+x}31^{+x}29^{+x}27^{+x}24^{+x}23^{+x}22^{+x}21^{+x}19^{+x}17^{+x}13^{+x}12^{+x}10^{+x}9^{+x}7^{+x}4^{+x+1}$	ECMA-182

Polynomial Value（多项式值）

此参数使用十六进制格式。当选择了标准多项式中的其中一个时，将自动计算此参数。也可手动输入此参数（请参见 Custom Polynomials（自定义多项式））。

Seed Value（种子值）

此参数使用十六进制格式。最大可能值为 $2^N — 1$。

N

此参数定义多项式的次数。这些值可能为 1 - 64 位。带数字的表格指示包括的多项式次数。带选定数字的单元格显示为蓝色，其他为白色。活动单元格数等于 N。数字反向排列。可单击单元格以选择或取消选择数字。

Polynomial representation（多项式表示）

此参数以数学符号显示结果多项式。

Custom Polynomials（自定义多项式）

可通过三种不同的方法输入自定义多项式：

对标准多项式进行少量更改

- 选择标准多项式中的一个。
- 通过单击相应的单元格在表中选择必要的次数；**Standard polynomial**（标准多项式）中的文本更改为 **Custom**（自定义）。
- 根据显示的多项式自动重新计算多项式值。

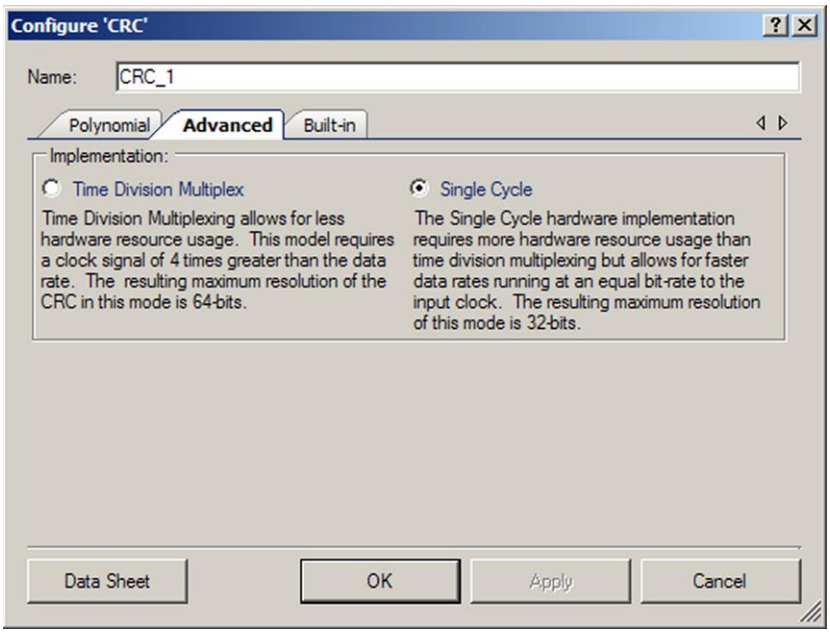
使用多项式次数

- 在 **N** 文本框中输入自定义多项式；**Standard polynomial**（标准多项式）中的文本更改为 **Custom**（自定义）。
- 通过单击相应的单元格在表中选择必要的次数。
- 在 **Polynomial representation**（多项式表示）中检查多项式视图。
- 根据显示的多项式自动重新计算多项式值。

使用十六进制格式

- 在 **Polynomial Value**（多项式值）文本框中输入十六进制格式的多项式值。
- 按 **[Enter]** 或切换到另一个控件；**Standard polynomial**（标准多项式）中的文本更改为 **Custom**（自定义）。
- 基于输入的多项式值重新计算 N 值和多项式次数。

“高级”选项卡



实现

此参数定义 CRC 组件的实现：**Time Division Multiplex**（时分复用）或 **Single Cycle**（单周期）。默认值为 **Single Cycle**（单周期）。

本地参数（供 API 使用）

以下参数用于 API，不在 GUI 中使用：

- **PolyValueLower (uint32) —** 包含十六进制格式的多项式值的下半部分。默认值为 0xB8h (LFSR= [8,6,5,4])，因为默认分辨率为 8。
- **PolyValueUpper (uint32) —** 包含十六进制格式的多项式值的上半部分。默认值为 0x00h，因为默认分辨率为 8。
- **SeedValueLower (uint32) —** 包含十六进制格式的种子值的下半部分。默认值为 0xFFh，因为默认分辨率为 8。
- **SeedValueUpper (uint32) —** 包含十六进制格式的种子值的上半部分。默认值为 0，因为默认分辨率为 8。

时钟选择

此组件中没有内部时钟。您必须附加时钟源。

注意：如果针对 **Implementation**（实现）参数选择了 **Time Division Multiplex**（时分复用），当分辨率大于 8 时，要生成正确的 CRC 序列，时钟信号必须比数据速率大 4 倍。

应用程序编程接口

应用程序编程接口 (API) 子程序允许您使用软件配置组件。下表列出了每个函数的接口，并进行了说明。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 将实例名称“CRC_1”分配给指定设计中组件的第一个实例。您可以将其重命名为遵循标识符语法规则的任何唯一值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。出于可读性考虑，下表中使用的实例名称为“CRC”。

函数	说明
CRC_Start()	用初始值初始化种子和多项式寄存器。在输入时钟的上升沿上开始计算 CRC。
CRC_Stop()	停止 CRC 计算。
CRC_Wakeup()	恢复 CRC 配置，并在输入时钟的上升沿上启动 CRC 计算。
CRC_Sleep()	停止 CRC 计算，并保存 CRC 配置。
CRC_Init()	用初始值初始化种子和多项式寄存器。
CRC_Enable()	在输入时钟的上升沿上启动 CRC 计算。
CRC_SaveConfig()	保存种子和多项式寄存器。
CRC_RestoreConfig()	恢复种子和多项式寄存器。
CRC_WriteSeed()	写入种子值。
CRC_WriteSeedUpper()	写入种子值的上半部分。仅针对 33 到 64 位 CRC 生成。
CRC_WriteSeedLower()	写入种子值的下半部分。仅针对 33 到 64 位 CRC 生成。
CRC_ReadCRC()	读取 CRC 值。
CRC_ReadCRCUpper()	读取 CRC 值的上半部分。仅针对 33 到 64 位 CRC 生成。
CRC_ReadCRCLower()	读取 CRC 值的下半部分。仅针对 33 到 64 位 CRC 生成。
CRC_WritePolynomial()	写入 CRC 多项式值。
CRC_WritePolynomialUpper()	写入 CRC 多项式值的上半部分。仅针对 33 到 64 位 CRC 生成。
CRC_WritePolynomialLower()	写入 CRC 多项式值的下半部分。仅针对 33 到 64 位 CRC 生成。
CRC_ReadPolynomial()	读取 CRC 多项式值。

函数	说明
CRC_ReadPolynomialUpper()	读取 CRC 多项式值的上半部分。仅针对 33 到 64 位 CRC 生成。
CRC_ReadPolynomialLower()	读取 CRC 多项式值的下半部分。仅针对 33 到 64 位 CRC 生成。

全局变量

变量	说明
CRC_initVar	指示是否已初始化 CRC。变量将初始化为 0，并在第一次调用 CRC_Start() 时设置为 1。这样，第一次调用 ACRC_Start() 子程序后，组件不用重新初始化即可重启。 如需重新初始化组件，可在 CRC_Start() 或 CRC_Enable() 函数前调用 CRC_Init() 函数。

void CRC_Start(void)

说明： 用初始值初始化种子和多项式寄存器。在输入时钟的上升沿上开始计算 CRC。

参数： 无

返回值： 无

副作用： 无

void CRC_Stop(void)

说明： 停止 CRC 计算。

参数： 无

返回值： 无

副作用： 无

void CRC_Sleep(void)

说明： 停止 CRC 计算，并保存 CRC 配置。

参数： 无

返回值： 无

副作用： 无

void CRC_Wakeup(void)

说明： 恢复 CRC 配置，并在输入时钟的上升沿上启动 CRC 计算。

参数： 无

返回值： 无

副作用： 无

void CRC_Init(void)

说明： 用初始值初始化种子和多项式寄存器。

参数： 无

返回值： 无

副作用： 无

void CRC_Enable(void)

说明： 在输入时钟的上升沿上启动 CRC 计算。

参数： 无

返回值： 无

副作用： 无

void CRC_SaveConfig(void)

说明： 保存初始种子和多项式寄存器。

参数： 无

返回值： 无

副作用： 无

void CRC_RestoreConfig(void)

说明： 恢复初始种子和多项式寄存器。

参数： 无

返回值： 无

副作用： 无

void CRC_WriteSeed(uint8/16/32 seed)

说明： 写入种子值。

参数： uint8/16/32 seed：种子值

返回值： 无

副作用： 根据掩码 = $2^{分辨率} — 1$ 剪切种子值。

例如，如果 CRC 分辨率为 14 位，掩码值为：
掩码 = $2^{14} — 1 = 0x3FFFu$。

种子值 = 0xFFFFu 被剪切：种子和掩码 = 0xFFFFu 和 0x3FFFu = 0x3FFFu。

void CRC_WriteSeedUpper(uint32 seed)

说明： 写入种子值的上半部分。仅针对 33 到 64 位 CRC 生成。

参数： uint32 seed：种子值的上半部分

返回值： 无

副作用： 根据掩码 = $2^{分辨率 - 32} — 1$ 剪切种子值的上半部分。

例如，如果 CRC 分辨率为 35 位，掩码值为：

$2^{(35 - 32)} — 1 = 2^3 — 1 = 0x0000\ 0007u$。

种子值的上半部分 = 0x0000 00FFu 被剪切。

种子和掩码的上半部分 = 0x0000 00FFu 和 0x0000 0007u = 0x0000 0007u。

void CRC_WriteSeedLower(uint32 seed)

说明： 写入种子值的下半部分。仅针对 33 到 64 位 CRC 生成。

参数： uint32 seed：种子值的下半部分

返回值： 无

副作用： 无

uint8/16/32 CRC_ReadCRC(void)

说明： 读取 CRC 值。

参数： 无

返回值： uint8/16/32：返回 CRC 值

副作用： 无

uint32 CRC_ReadCRCUpper(void)

说明： 读取 CRC 值的上半部分。仅针对 33 到 64 位 CRC 生成。

参数： 无

返回值： uint32：返回 CRC 值的上半部分

副作用： 无

uint32 CRC_ReadCRCLower(void)

说明： 读取 CRC 值的下半部分。仅针对 33 到 64 位 CRC 生成。

参数： 无

返回值： uint32：返回 CRC 值的下半部分

副作用： 无

void CRC_WritePolynomial(uint8/16/32 polynomial)

说明： 写入 CRC 多项式值。

参数： uint8/16/32 polynomial：CRC 多项式

返回值： 无

副作用： 根据掩码 = $2^{分辨率} — 1$ 剪切多项式值。例如，如果 CRC 分辨率为 14 位，掩码值为：掩码 = $2^{14} — 1$ = 0x3FFFu。

多项式值 = 0xFFFFu 被剪切：

多项式和掩码 = 0xFFFFu 和 0x3FFFu = 0x3FFFu。

void CRC_WritePolynomialUpper(uint32 polynomial)

说明： 写入 CRC 多项式值的上半部分。仅针对 33 到 64 位 CRC 生成。

参数： uint32 polynomial：CRC 多项式值的上半部分

返回值： 无

副作用： 根据掩码 = $2^{(分辨率 — 32)} — 1$ 剪切多项式值的上半部分。例如，如果 CRC 分辨率为 35 位，掩码值为：

$2^{(35 — 32)} — 1 = 2^3 — 1$ = 0x0000 0007u。

多项式值的上半部分 = 0x0000 00FFu 被剪切：

多项式和掩码的上半部分 = 0x0000 00FFu 和 0x0000 0007u = 0x0000 0007u。

void CRC_WritePolynomialLower(uint32 polynomial)

说明： 写入 CRC 多项式值的下半部分。仅针对 33 到 64 位 CRC 生成。

参数： uint32 polynomial：CRC 多项式值的下半部分

返回值： 无

副作用： 无

uint8/16/32 CRC_ReadPolynomial(void)

说明： 读取 CRC 多项式值。

参数： 无

返回值： uint8/16/32：返回 CRC 多项式值

副作用： 无

uint32 CRC_ReadPolynomialUpper(void)

说明： 读取 CRC 多项式值的上半部分。仅针对 33 到 64 位 CRC 生成。

参数： 无

返回值： uint32：返回 CRC 多项式值的上半部分

副作用： 无

uint32 CRC_ReadPolynomialLower(void)

说明： 读取 CRC 多项式值的下半部分。仅针对 33 到 64 位 CRC 生成。

参数： 无

返回值： uint32：返回 CRC 多项式值的下半部分。

副作用： 无

MISRA 合规性

本节介绍了本组件与 MISRA-C:2004 的合规和偏差情况。定义了两种类型的偏差：

- 项目偏差 - 适用于所有 PSoC Creator 组件的偏差
- 特定偏差 - 仅适用于此组件的偏差

本节提供了有关组件特定偏差的信息。*系统参考指南*的“MISRA 合规性”章节中介绍项目偏差以及有关 MISRA 合规性验证环境的信息。

此 CRC 组件具有如下特定偏差：

MISRA-C:2004 规则	规则类别（必须/建议）	规则说明	偏差说明
14.3	R	预处理前，会自动显示一行空语句；如果该空语句后的第一个字符是空白字符，则其后可能会附随一个注释。	如果使用的宏具有一组用花括号包含的语句，将会导致偏差。宏为 CRC_EXECUTE_DFF_RESET。实现 MISRA 合规性需要使用该宏，并在其后加分号。这样做会导致直接使用该宏的设计失败。
19.4	R	C 语言的宏仅限扩展至使用花括号的初始化表达式、常量、使用圆括号的表达式、类型限定符、存储类的限定符或“do-while-zero”结构。	如果使用的宏具有一组用花括号包含的语句，将会导致偏差。宏为 CRC_EXECUTE_DFF_RESET。实现 MISRA 标准需要使用该宏，并在其后加分号。这样做会导致直接使用该宏的设计失败。
19.7	A	函数应该优先于类似于函数的宏。	宏 CRC_IS_CRC_ENABLE 导致偏差。如果删除该宏，会导致直接使用该宏的设计失败。

固件源代码示例

PSoC Creator 在“查找示例项目”对话框中提供了很多包括原理图和代码示例的示例项目。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开 Start Page（开始页）或 **File**（文件）菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（筛选选项）可缩小可选项目的列表。

有关更多信息，请参见 PSoC Creator 帮助中的“Find Example Project（查找示例项目）”主题。

功能描述

将 CRC 作为线性反馈移位寄存器 (LFSR) 来实现。移位寄存器计算 LFSR 函数，多项式寄存器保留定义 LFSR 多项式的多项式，种子寄存器启用启动数据的初始化。

启动组件之前，必须初始化种子和多项式寄存器。

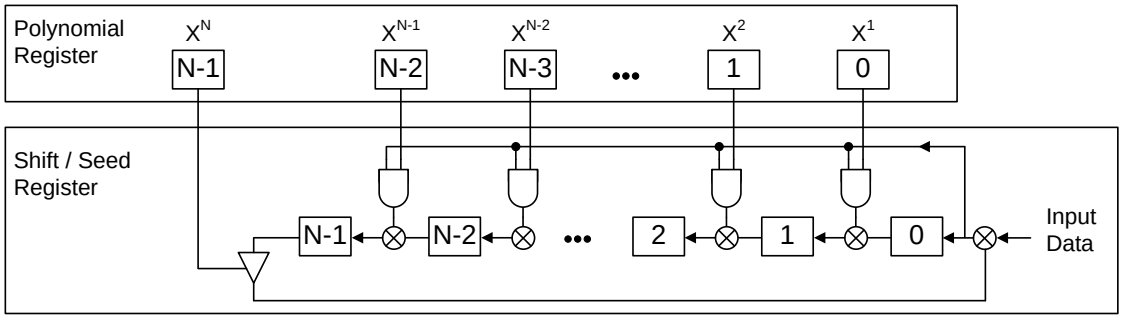
N 位 LFSR 结果的计算由带 N + 1 个乘积项的多项式指定，最后一个乘积项为 X^0 乘积项，其中 $X^0 = 1$。例如，广泛使用的 CRC-CCITT 16 位多项式为 $X^{16} + X^{12} + X^5 + 1$。CRC 算法假设存在 X^0 乘积项，这样，对于 N 位结果，可用 N 位而非 (N + 1) 位规范来表达多项式。

要指定多项式规范，写入对应完全多项式的 (N + 1) 位二进制数，各个乘积项带 1。CRC-CCITT 多项式为 10001000000100001b。然后，放弃最右边的位（X^0 乘积项）以获取 CRC 多项式值。要实现 CRC-CCITT 示例，加载值为 8810h 的多项式寄存器。

输入时钟的上升沿将输入数据流的每一位（先 MSB）通过移位寄存器移入，计算指定 CRC 算法。需要八个时钟以计算输入数据的各个字节的 CRC。

注意，初始种子值丢失。这无关紧要，因为种子值仅用于针对各个数据集初始化移位寄存器一次。

框图和配置



时序图

图 1. 时分复用实现模式

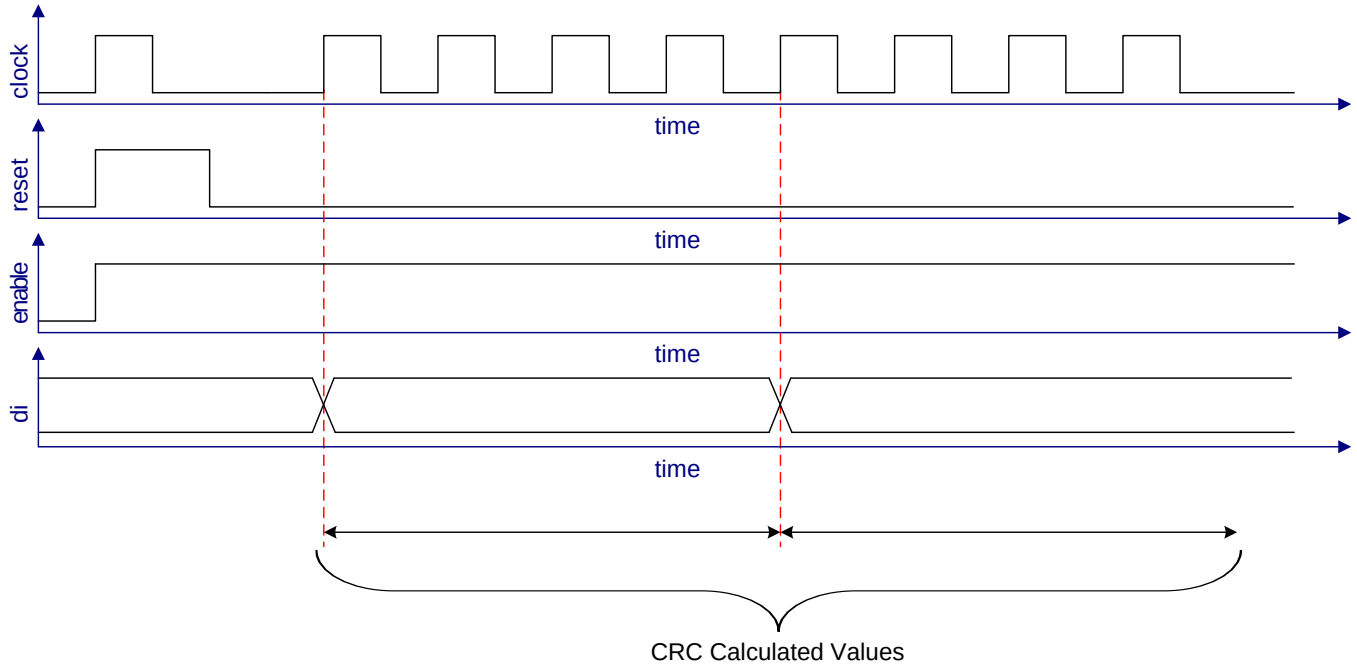
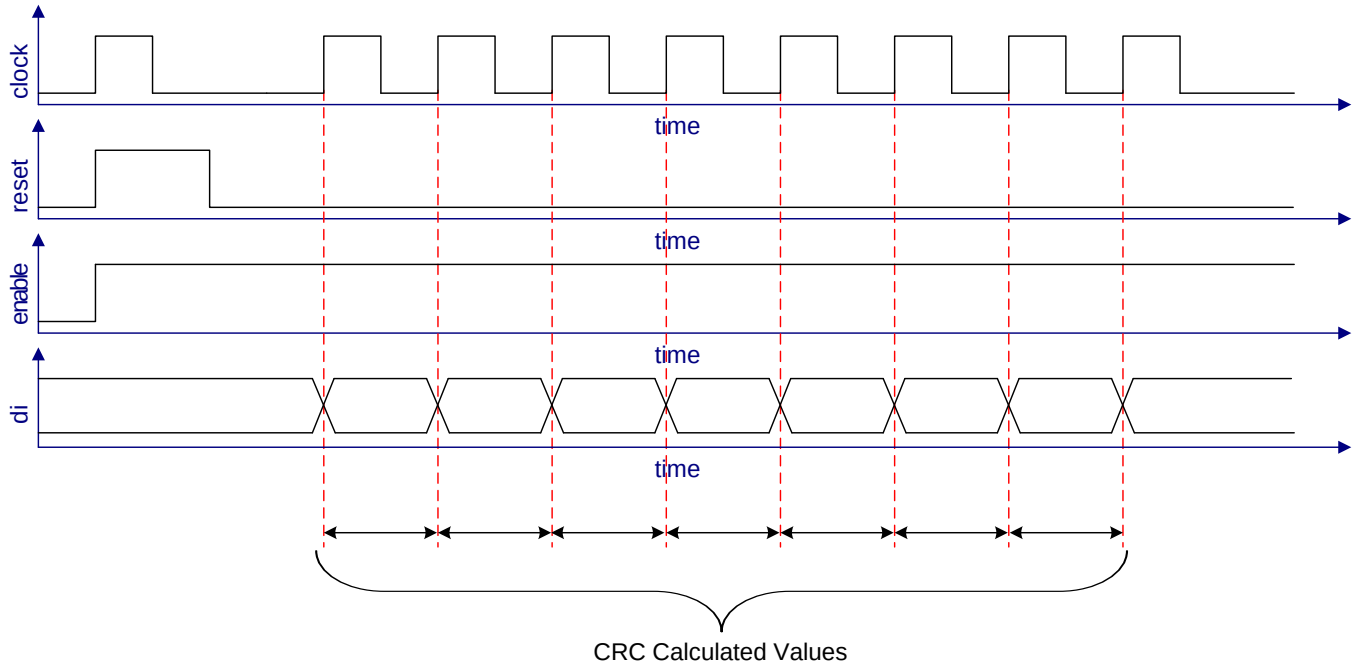


图 2. 单周期实现模式



资源

CRC 组件放置在整个 UDB 阵列中。该组件利用以下资源。

配置	资源类型					
	数据路径单元	宏单元	状态单元	控制单元	DMA 通道	中断
8 位单周期	1	3	–	1	–	–
16 位单周期	2	3	–	1	–	–
24 位单周期	3	3	–	1	–	–
32 位单周期	4	3	–	1	–	–
16 位时分	1	9	–	1	–	–
24 位时分	2	10	–	1	–	–
32 位时分	2	9	–	1	–	–
40 位时分	3	10	–	1	–	–
48 位时分	3	9	–	1	–	–
56 位时分	4	10	–	1	–	–
64 位时分	4	9	–	1	–	–

API 存储器使用

根据编译器、组件、所用 API 数量和组件配置的不同，组件内存使用会出现较大变化。下表提供指定组件配置中可用的 API 的存储器使用。

已利用释放模式中配置的相关编译器进行了测量，大小采用了优化设定。对于特定设计，可以分析编译器生成的映射文件以确定存储器使用。

配置	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	闪存字节	SRAM 字节	闪存字节	SRAM 字节	闪存字节	SRAM 字节
8 位单周期	155	2	242	8	246	8
16 位单周期	202	2	260	8	260	8
24 位单周期	294	2	320	8	294	8
32 位单周期	212	2	280	8	282	8

配置	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节
16 位时分	234	2	320	8	320	8
24 位时分	548	2	404	8	398	8
32 位时分	567	2	432	8	434	8
40 位时分	687	2	568	8	580	8
48 位时分	839	2	610	8	638	8
56 位时分	938	2	662	8	702	8
64 位时分	1045	2	710	8	718	8

直流和交流电气特性

除非另有说明，否则这些规范的适用条件是 -40 °C ≤ T_A ≤ 85 °C 且 T_J ≤ 100 °C。除非另有说明，否则这些规范的适用范围为 1.71 V 到 5.5 V。

直流特性

参数	说明	最小值	典型值[1]	最大值	单位
I_{DD}	组件电流消耗				
	8 位单周期	–	10	–	µA/MHz
	16 位单周期	–	16	–	µA/MHz
	24 位单周期	–	26	–	µA/MHz
	32 位单周期	–	33	–	µA/MHz
	16 位时分	–	17	–	µA/MHz
	24 位时分	–	29	–	µA/MHz
	32 位时分	–	29	–	µA/MHz
	40 位时分	–	35	–	µA/MHz
	48 位时分	–	35	–	µA/MHz
	56 位时分	–	43	–	µA/MHz

1. 未包括设备 IO 和时钟分配的电流。这些值是在 25 °C 时的值。

参数	说明	最小值	典型值[1]	最大值	单位
	64 位时分	–	44	–	µA/MHz

交流特性

参数	说明	最小值	典型值	最大值[2]	单位
f_{CLOCK}	组件时钟频率				
	8 位单周期	–	–	41	MHz
	16 位单周期	–	–	32	MHz
	24 位单周期	–	–	30	MHz
	32 位单周期	–	–	28	MHz
	16 位时分	–	–	34	MHz
	24 位时分	–	–	24	MHz
	32 位时分	–	–	29	MHz
	40 位时分	–	–	24	MHz
	48 位时分	–	–	27	MHz
	56 位时分	–	–	23	MHz
	64 位时分	–	–	28	MHz

组件更改

本节介绍组件与以前版本相比的主要更改。

版本	更改说明	更改/影响原因
2.40	更新了数据表，添加了 PSoC 4 的内存使用情况。 更新了对 PSoC 4 的 [25 – 32] 位单周期实现模式的定义。	
2.30	已添加 MISRA 合规性章节。	此组件具有所描述的特定偏差。
	添加了 PSoC 4A 支持	

2. 这些值提供了此组件的最大安全工作频率。可以在更高的时钟频率运行组件，在该频率将需要使用 STA 结果验证时序要求。

版本	更改说明	更改/影响原因
	更改了时序图	
2.20	添加了 PSoC 5LP 支持	
2.10	为实现参数更改了错误消息及其外观。	
	修正了多项式次数“N”设置，设为 64 位分辨率。	
	修正了多项式值验证。	
2.0.b	对数据表进行了少量编辑和更新	
2.0.a	向数据表中添加了特性数据	
	对数据表进行了少量编辑和更新	
2.0	添加了对 PSoC 3 ES3 芯片的支持。更改包括： ■ 添加了时分复用实现的 4x 时钟 ■ 1x 时钟上现可用 1 到 32 位的单周期实现。 ■ 4x 时钟上现可用 9 到 64 位的时分复用实现。 ■ 添加了异步输入信号复位。 ■ 添加了同步输入信号使能。 ■ 针对“Implementation”（实现）（Time Division Multiplex（时分复用实现）、Single Cycle（单周期实现））参数将新的“Advanced”（高级）页面添加到“Configure”（配置）对话框。	新增了支持 PSoC 3 ES3 组件的要求，因此创建了新的 2.0 版 CRC 组件。
	添加了 CRC_Sleep()/CRC_Wakeup() 和 CRC_Init()/CRC_Enable() API。	为支持低功耗模式并提供常用接口，以单独控制大多数组件的初始化和启用。
	更新了函数 CRC_WriteSeed() 和 CRC_WriteSeedUpper()。	掩码参数用于剪切种子值，以在写入时定义 CRC 分辨率。
	已将验证器添加到“Resolution”（分辨率）参数。	CRC 分辨率为 1 到 64 位。添加了验证器以限制输入值。
	将复位 DFF 触发器添加到多项式写入函数：CRC_WritePolynomial()、CRC_WritePolynomialUpper() 和 CRC_WritePolynomialLower()。	计算 CRC 之前，需要将 DFF 触发器设为正确的状态（多项式的最高有效位始终为 1）。要符合此条件，任何对种子或多项式寄存器的写入都将复位 DFF 触发器。
	已更新“Configure”（配置）对话框，以使用以下参数的“Expression View”（表达式视图）：“PolyValueLower”、“PolyValueUpper”、“SeedValueLower”、“SeedValueLower”	“Expression View”（表达式视图）用于直接访问符号参数。此视图允许您用外部参数连接组件参数（如果需要）。

版本	更改说明	更改/影响原因
	已更新“Configure”（配置）对话框，以为各种参数添加错误图标。	如果在文本框中输入了错误的值，将显示错误图标以及问题说明的工具提示。这种方法比单独提供错误消息更方便。
1.20	已更改 API 生成方式。在 1.10 版本中是根据定制器的设置来生成 API 的。在 1.20 版本中，像大部分其他组件那样，API 是由 .c 和 .h 文件提供的。	此更改使用户可查看和更改生成的 API 文件，并且在后续构建上不会被覆盖。
	“Seed”（种子）和“Polynomial”（多项式）参数已更改为以十六进制格式显示。	已进行更改，以符合公司标准。