增量型模数转换器数据表 ADCINC V 1.1

Copyright © 2002-2011 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块				API 内存 （字节）			引脚 （根据外部 I/O）
	数字	模拟 CT	模拟 SC		闪存		RAM	
调制器 （一阶或二阶）		1st 和 2nd	1st	2nd	1st	2nd		
CY8C29xxx、CY8C24x94、CY8C23x33、CY7C64215、CY8CLED04/16、CY8CLED0xD、CY8CLED0xG、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C28x45、CY8C28x43、CY8C28x52、CY8CPLC20、 CY8CLED16P01								
	1	0	1	2	273	322	8	1
CY8C27/24/22xxx、CY8CLED08	1	0	1	2	226	275	8	1

请参阅 AN2239 模数转换器选择指南 以了解其他转换器。

有关一个或多个使用此用户模块的完全配置的功能性示例项目，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 6 到 14-bit 分辨率
- 可选同步 8-bit 脉冲宽度调制输出
- 可选差分输入
- 有符号或无符号数据格式
- 最高达 15.6 ksps （6-bit 分辨率）的采样率
- 内部和外部参考选项所定义的输入范围
- 内部或外部时钟

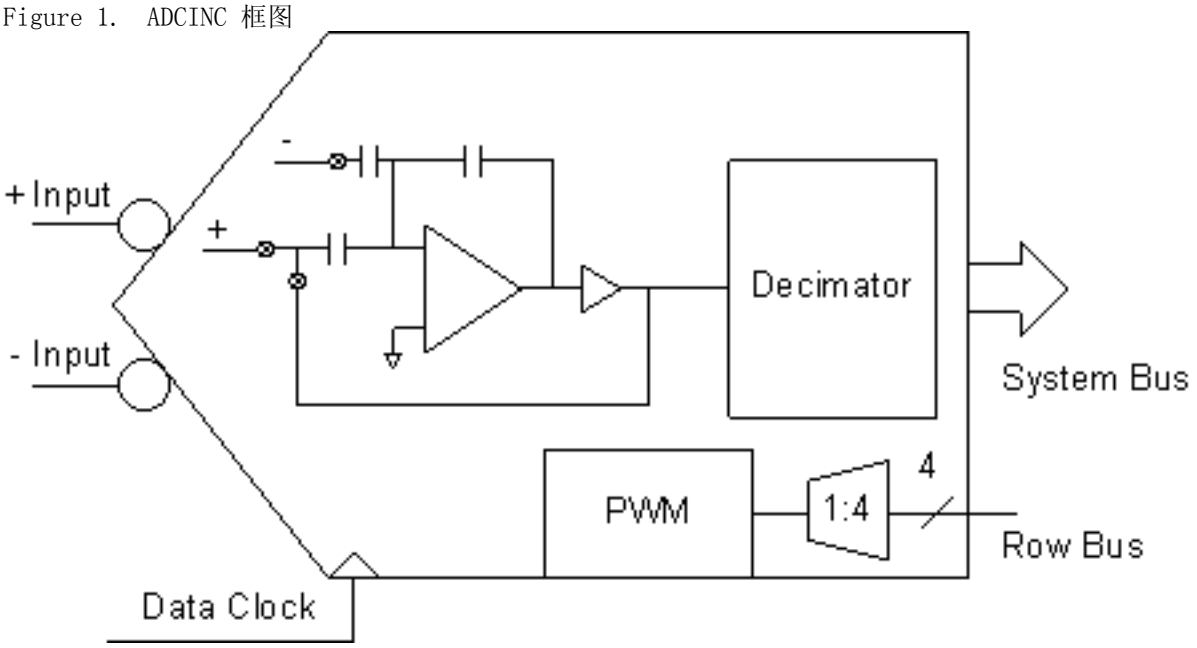
Note 如果此用户模块用于 29K 系列，它将额外消耗 6 毫安。 替代做法是使用 ADCINCVR 用户模块。

ADCINC 是返回 6 到 14 位结果的差分或单一输入模数转换器。 最高数据时钟频率为 8 MHz，但是 2 MHz 是推荐用于改善线性度的最高频率。 只能放置一次此模数转换器，这是因为其实现使用的是硬件抽取滤波器而不是数字模块。 这是最节约资源的模数转换器。 可以用附加开关电容模块实现二阶调制器，从而用 8 MHz 数据时钟获得更好的线性度。

时序是通过八位脉冲宽度调制实现的，该脉冲宽度调制为您提供了与输入采样同步的调制脉冲宽度。

ADCINC 需要 $2^n - 1$ 个积分周期才能生成分辨率为 n 位的输出。

Figure 1. ADCINC 框图



快速启动

可以从以下地址下载预先配置的示例项目： www.cypress.com/psocexampleprojects. 创建 ADCINC 项目：

1. 在 PSoC Designer 中，选择 **“ 新建项目 ”**。
2. 在 “ 新建项目 ” 对话框中，选择 **仅设计器项目**. 为项目选择名称和位置，然后单击 **“ 确定 ”**.
3. 在 **“ 选择新基准部件 ”** 对话框中，选择支持此用户模块的器件之一。 请参阅本数据表开头的 “ 资源 ” 表。
4. 在 **用户模块目录**，单击以展开 **模数转换器**的选择，然后展开 **ADCINC** 文件夹。 如果用户模块目录不可见，请选择 **“ 视图 ”>“ 用户模块目录 ”**.
5. ADCINC 可用于单段调制器或双段调制器。 如果不知道要使用哪个版本，可以通过下列方法查看数据表：右键单击其中一个版本，然后选择 **“ 数据表 ”** 以在决定版本之前查看数据表。
6. 右键单击用户模块，然后选择 **“ 放置 ”**.

[+] Feedback

功能说明

ADCINC 提供了由一个模拟开关电容 PSoC 模块、一个数字 PSoC 模块和抽取滤波器构成的一阶调制器，如下图所示：

Figure 2. 带一阶调制器的 ADCINC 的图示

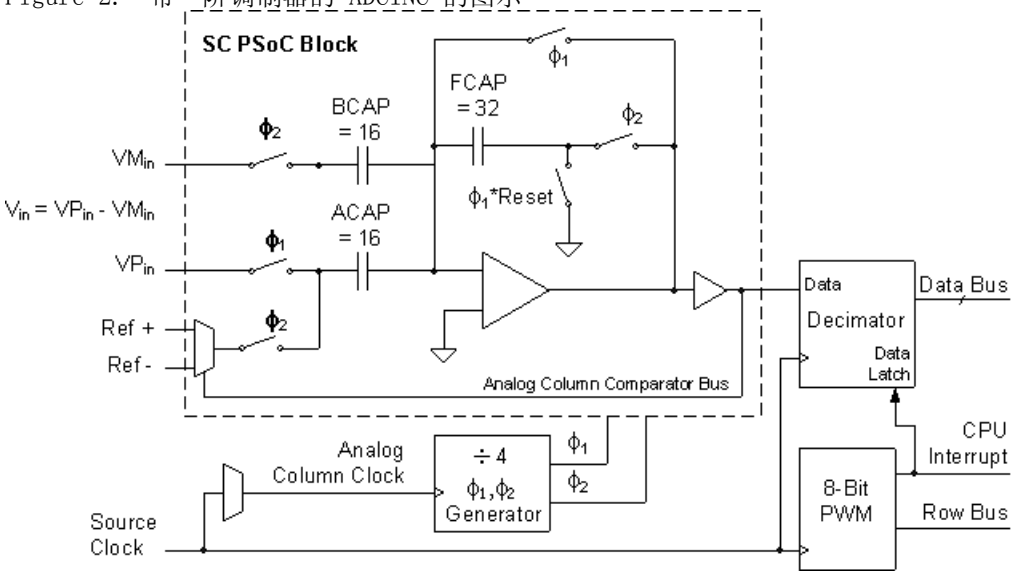
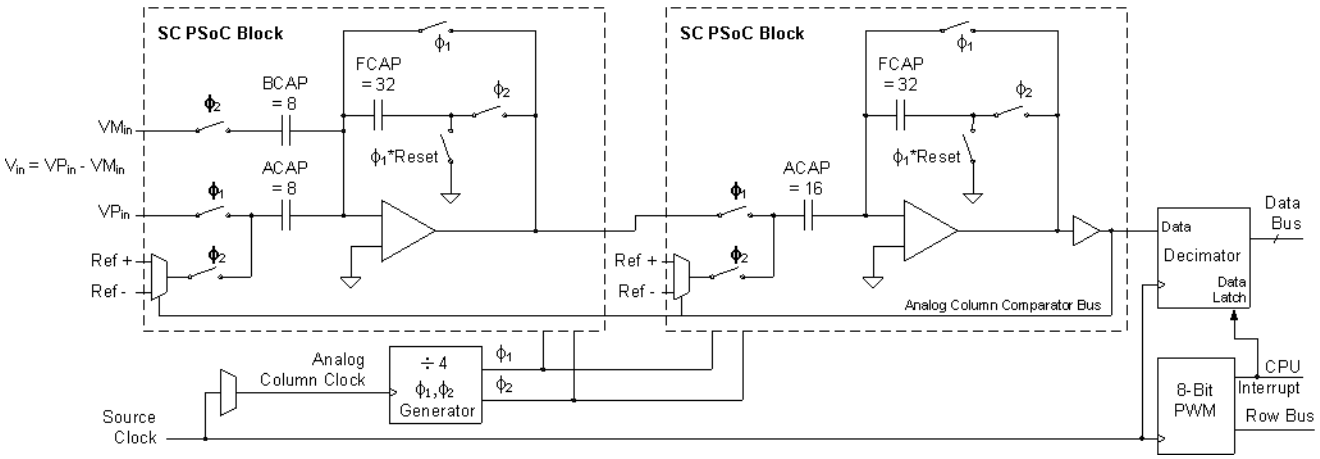


Figure 3. 带二阶调制器的 ADCINC 的图示



ADCINC 的范围设置为 $\pm V_{Ref}$。可以在 PSoC Designer 的“全局资源”窗口中设置 V_{Ref}。对于固定量程，V_{Ref} 设置为 $V_{Bandgap}$ 或 1.6 $V_{Bandgap}$。对于可调整量程，V_{Ref} 设置为端口 2[6]。对于供电比率计量程，V_{Ref} 设置为 $V_{DD}/2$。

模拟模块配置为可复位的积分器。根据输出极性，对参考控制进行了配置，以便可以从输入中增减参考电压，并置于积分器中。此参考控制尝试将积分器输出拉回到 AGND。如果积分器运行 2^{Bits} 次，而输出电压比较器在这些次数中结果为正的次数为“n”，则输出残余电压（V_{resid}）为：

[+] Feedback

Equation 1

$$V_{resid} = 2^{Bits} \cdot V_{in} - n \cdot V_{ref} + (2^{Bits} - n) \cdot V_{ref}$$

Equation 2

$$V_{in} = \frac{n - 2^{Bits-1}}{2^{Bits-1}} V_{ref} + \frac{V_{resid}}{2^{Bits}}$$

此等式说明了此模数转换器的范围为 $\pm V_{Ref}$，分辨率 (LSB) 为 $V_{Ref}/2^{Bits-1}$，计算结束时输出电压定义为残余电压。 由于 V_{resid} 始终小于 V_{Ref}，因此 $V_{resid}/2^{Bits}$ 小于 LSB 的一半，可以忽略。

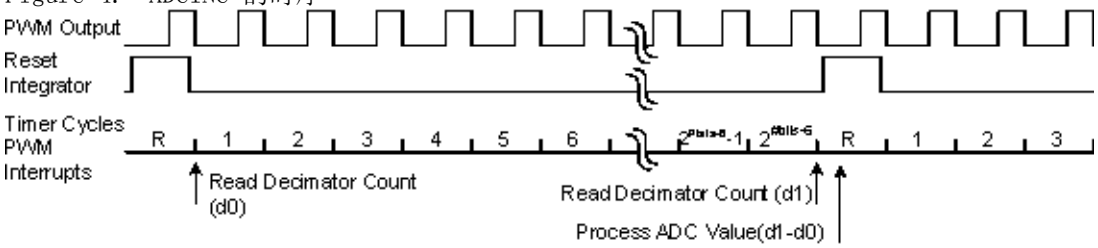
要将积分器函数作为增量型模数转换器，请使用下列数字资源：

- 一个脉冲宽度调制，用于对适当积分周期数进行计数。
- 一个在递增模式下配置的抽取滤波器，用于累计输出比较器为正的周期数。

Note **CAUTION:** 当放置此模块时，必须使用与模拟和数字模块相同的源时钟对其进行配置。 不这样做会导致其运行不正常。

脉冲宽度调制设置为每 256 个计数生成一个中断。 这会导致对输入采样 64 次，正好等于一个积分周期。 抽取滤波器计数器设置为累计这些积分周期中的 $2^{Bits}/64$ 个周期。 将在积分时间开始和结束时对累计值进行采样。 增加了一个周期，用于复位积分器和处理应答。

Figure 4. ADCINC 的时序



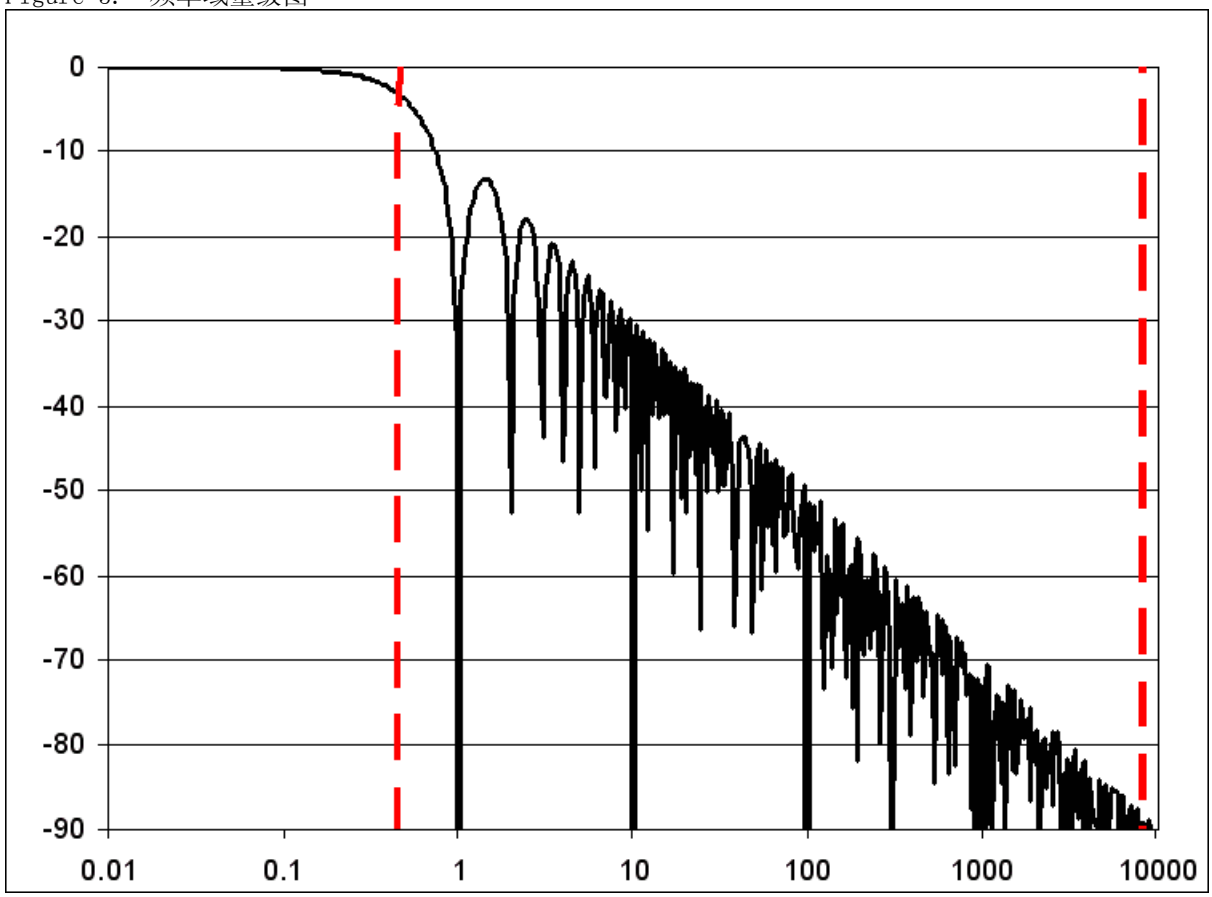
由于 ACDINC 控制基于中断，对于结果，采样时间相对较长，因此在处理采样时期望处理器等待，这是不合理的。 模数转换器子程式与主程序之间的主要通信是一个可轮询的数据可用标志。 API 可用于检查数据标志和检索数据。

数据处理程序设计成基于轮询。 如果需要基于中断的数据处理程序，可以将特定于应用程序的数据处理程序代码添加到程序集文件 ANDINCINT.asm 中的中断子程式 _ADCINC_ADConversion_ISR。 插入代码的位置已做清晰标记。

[+] Feedback

下面绘制的频率域量级图使频率标准化，因此 14-bit 采样率 F_{nom} = 1.0。 -3 dB 点位于 .443 ∞ F_{nom}，函数零点位于 F_S 的每个整数倍数的位置。 由于 ADCINCPWM 是针对 14 位分辨率设置的，实际采样率比额定输出速率快 16385 倍，奈奎斯特限制比 F_{nom} 高 8,192 （即 13 个八度），这极大降低了对抗锯齿滤波器的要求。 对于 13 位分辨率，奈奎斯特限制为 12 个八度；对于 12 位分辨率，该限制为 11 个八度，依此类推。

Figure 5. 频率域量级图



[+] Feedback

直流和交流电气特性

以下值均为基于初始特性数据的预期性能指标。除非另有指定，否则 T_A = 25° C，V_{dd} = 5.0V，功耗 = 高，运算放大器偏压 = 低，输出在 P2[6] 上有 1.25 外部 V_{ref} 的情况下参考 P2[4] 上的 2.5V 外部模拟接地。

Table 1. 5.0V 二阶调制器直流和交流电气特性

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 Vdd	V	Ref Mux = Vdd/2 ± Vdd/2
输入电容[1]	3	---	pF	
输入阻抗	1/(C*clk)	---	Ω	
分辨率	---	8	位	
采样率	---	.125 到 31.25	ksps	
信噪比	46	---	dB	
直流精度				
微分非线性误差	0.1	---	最低有效位	列时钟 2 MHz
积分非线性误差	0.5	---	最低有效位	
偏移误差	10	---	mV	
增益误差				
包括参考增益误差	3.0	--	% FSR	
不包括参考增益误差[2]	0.1	--	% FSR	
工作电流				
低功耗	180	---	uA	
中功耗	840	---	uA	
高功耗	3450	---	uA	
数据时钟	---	0.032 to 8.0	MHz	数字模块和模拟列时钟的输入

[+] Feedback

Table 2. 5.0V 一阶调制器直流和交流 5.0V 电气特性

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 Vdd		Ref Mux = Vdd/2 ± Vdd/2
输入电容[1]	3	---	pF	
输入阻抗	1/(C*clk)	---	Ω	
分辨率	---	8	位	
采样率	---	.125 到 31.25	ksps	
信噪比	44	---	dB	
直流精度				
微分非线性误差	0.6	---	最低有效位	列时钟 2 MHz
积分非线性误差	0.7	---	最低有效位	
偏移误差	5	---	mV	
增益误差				
包括参考增益误差	3.0	--	% FSR	
不包括参考增益误差[2]	0.1	--	% FSR	
工作电流				
低功耗	50	---	uA	
中功耗	500	---	uA	
高功耗	1900	---	uA	
数据时钟	---	0.032 to 8.0	MHz	数字模块和模拟列时钟的输入

[+] Feedback

以下值均为基于初始特性数据的预期性能指标。除非另有指定，否则 T_A = 25° C，V_{dd} = 3.3V，功耗 = 高，运算放大器偏压 = 低，输出在 P2[6] 上有 1.25 外部 V_{ref} 的情况下参考 P2[4] 上的 1.64V 外部模拟接地。

Table 3. 3.3V 一阶调制器直流和交流电气特性

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 Vdd	V	Ref Mux = Vdd/2 ± Vdd/2
输入电容[1]	3	---	pF	
输入阻抗	1/(C*clk)	---	Ω	
分辨率	---	8	位	
采样率	---	.125 到 31.25	ksps	
信噪比	46	---	dB	
直流精度				
微分非线性误差	0.1	---	最低有效位	列时钟 2 MHz
积分非线性误差	0.5	---	最低有效位	
偏移误差	10	---	mV	
增益误差				
包括参考增益误差	3.0	--	% FSR	
不包括参考增益误差[2]	0.3	--	% FSR	
工作电流				
低功耗	130	---	uA	
中功耗	840	---	uA	
高功耗	3370	---	uA	
数据时钟	---	0.032 to 8.0	MHz	数字模块和模拟列时钟的输入

[+] Feedback

Table 4. 3.3V 一阶调制器直流和交流电气特性

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 Vdd	V	Ref Mux = Vdd/2 ± Vdd/2
输入电容[1]	3	---	pF	
输入阻抗	1/(C*clk)	---	Ω	
分辨率	---	8	位	
采样率	---	.125 到 31.25	ksps	
信噪比	44	---	dB	
直流精度				
微分非线性误差	0.6	---	最低有效位	列时钟 2 MHz
积分非线性误差	0.8	---	最低有效位	
偏移误差	6	---	mV	
增益误差				
包括参考增益误差	3.0	--	% FSR	
不包括参考增益误差[2]	0.3	--	% FSR	
工作电流				
低功耗	50	---	uA	
中功耗	500	---	uA	
高功耗	1900	---	uA	
数据时钟	---	0.032 to 8.0	MHz	数字模块和模拟列时钟的输入

电气特性说明

1. 包括 I/O 引脚。
2. 参考增益误差测量方法是将外部参考与通过测试复用器进行路由并返回至引脚的 VRefHigh 和 VRefLow 进行比较。

[+] Feedback

放置

一阶调制器设计需要两个 PSoC 模块，一个为数字模块，另一个为模拟模块。 模拟模块可以放置在任何开关电容 PSoC 模块中。 唯一考虑因素是输入和时钟可用性。 不过，数字模块必须能够向硬件抽取滤波器提供信号。 在 CY8C27xxx 系列中，符合要求的数字模块为 DBB01、DBB02、DBB11 和 DCB12。 在其他器件系列中，可以使用任何数字模块。 必须为数字模块和模拟模块选择相同的时钟，否则此用户模块无法正常工作。

二阶调制器设计需要第二个开关电容 PSoC 模块。 两个模拟模块必须位于同一列中，以便它们可以共享列比较器总线。 在一阶和二阶调制器中，对数字模块的限制是相同的。

虽然有大量可用于模拟模块和数字模块的放置，但是 ADCINC 还使用 PSoC 器件的唯一硬件抽取过滤器。 对于给定配置，只能放置一个 ADCINC 实例。 对于动态重新配置，只要模块不重叠，就可以加载多个配置。 虽然两个实例似乎都可以工作，但是只有最近加载的实例的输出是正确的。

参数和资源

在放置 ADCINC 脉冲宽度调制实例后，必须配置下列参数才能正确操作：分辨率、数据格式、数据时钟、PosInput 信号复用器选择、NegInput 复用器选择、NegInput 增益、时钟相位、脉冲宽度和脉冲宽度调制输出。

数据格式

此选择确定返回结果的数据格式。 有符号结果是具有选定分辨率的 2 的补码值。

分辨率

此选择确定返回结果的数据格式。 有效分辨率选项为 6 到 14 位。

数据时钟

数据时钟决定采样率。 此时钟传递到一阶调制器设计的两个 PSoC 模块和二阶设计的所有三个 PSoC 模块。

Note **IMPORTANT:** 必须为数字模块和模拟列时钟选择相同的时钟，否则此用户模块无法正常工作。

当 CPU 以 24 MHz 运行时，数据时钟不能设置为小于 250 kHz。 否则，它可以设置为 125 kHz 那么低。 数据时钟不能超过 CPU 时钟，它必须始终等于或小于 CPU 时钟。 脉冲宽度调制设置为每 256 个数据时钟计数提供一个中断。计数器对这些周期中的 2^{Bits-6} 个周期的信号进行积分。 需要一个额外的周期来复位积分器和处理数据。 采样率按下面的等式 3 定义：

Equation 3

$$SampleRate = \frac{DataClock}{256 \cdot (2^{Bits-6} + 1)}$$

可使用的最大数据时钟为 8 MHz。 这是由开关电容模块的限制所致。 可以使用 8 MHz 时钟速率计算每个位速率的最大采样率。 下表中列出了采样率：

[+] Feedback

分辨率	最大采样率
6-bit	15.6 ksps
7-bit	10.4 ksps
8-bit	6.25 ksps
9-bit	3.4 ksps
10-bit	1.8 ksps
11-bit	976 sps
12-bit	480 sps
13-bit	242 sps
14-bit	121 sps

采样窗口确定模数转换器抑制的正常模式频率。 它定义为：

Equation 4

$$SampleWindow = \frac{2^{Bits}}{DataClock}$$

要抑制较高的频率及其谐波，请选择采样窗口，以便它是抑制频率的偶数倍。

时钟相位

时钟相位的选择用于将一个模拟 PSoC 模块的输出与另一个模拟 PSoC 模块的输入同步。 开关电容模拟 PSoC 模块使用两相时钟（phi1、phi）获取和传输信号。 通常，ADCINC 的输入是在 phi1 上采样的。 这便产生一个问题：许多用户模块在 phi1 期间将其输出自动归零，仅在 phi2 期间给出有效输出。 如果将此类模块的输出提供给 ADCINC 的输入，则 ADCINC 会获取自动归零的输出而不是有效信号。 时钟相位选择允许交换相位，因此输入信号是在 phi2 期间获取的。

PosInput

模数转换器的主输入。 PSoC Designer 允许您选择任何合法输入。

NegInput

允许为模数转换器创建差分输入。 通过使用 NegInput 增益参数，可以加权此输入。 如果需要单一输入（而不是差分输入），则将 NegInput 增益参数值设置为 “ 断开连接 ”。 对于 NegInput 参数，PSoC Designer 允许用户选择任何合法输入。

NegInput 增益

选择负输入的增益。 如果需要单端输入，请将此值设置为 “ 断开连接 ”。

脉冲宽度

允许脉冲宽度调制脉冲宽度设置为从 1 到 255 的数值。 如果未设置任何值，则在调用 GetSamples 函数时，用户模块将脉冲宽度调制脉冲宽度自动设置为 1。

脉冲宽度调制输出

可以禁用该输出参数，或者将其路由到四个全局输出信号之一。

[+] Feedback

中断生成控制

只有当在 PSoC Designer 中选中 “ 启用中断生成控制 ” 复选框的情况下，才能访问以下参数。 可以在以下菜单下找到此复选框： **项目 > 设置 ... > 器件编辑器。**

IntDispatchMode

IntDispatchMode 参数用于指定中断请求的处理方式，这些中断由同一模块不同外覆层中的多个用户模块共享。 选择 “ActiveStatus” 会导致固件在为共享的中断请求提供服务之前测试哪一个外覆层正处于活动状态。 每次请求共享中断时，都会进行此测试。 这会增加延迟，还会产生为共享中断请求提供服务的不确定过程，但是不需要任何 RAM。 选择 “OffsetPreCalc” 参数会导致固件仅在最初加载重叠层时计算共享中断请求的来源。 这种计算可减少中断延迟，并产生为共享中断请求提供服务的确定过程，但会占用一个字节的 RAM 空间。

应用程序编程接口

应用程序编程接口 (API) 子程序作为用户模块的一部分提供，从而使设计人员能够采用更高级的方式处理模块。 本节指定每个函数的接口，以及 “ 引用 ” 文件所提供的相关常量。

Note 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能通过调用 API 函数发生更改。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 选择此 “ 寄存器易失 ” 策略是为了提高效率，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

每次放置用户模块时，都会为其分配一个实例名称。 默认情况下，PSoC Designer 会为指定项目中此用户模块的第一个实例分配 ADCINC_1。 可将该值更改为符合标识符语法规则的任意唯一值。 分配的实例名称成为每个全局函数名称、变量和常量符号的前缀。 为简便起见，在以下说明中将实例名称缩写为 ADCINC。

ADCINC_Start

说明：

对此用户模块执行所有必需的初始化，并设置开关电容 PSoC 模块的功耗水平。 启动了 PWM。

C 原型：

```
void  ADCINC_Start (BYTE bPowerSetting)
```

汇编：

```
mov   A, bPowerSetting
lcall  ADCINC_Start
```

参数：

bPowerSetting： 1 个用于指定功耗水平的字节。 在复位和配置之后，分配给 ADCINC 的模拟 PSoC 模块关闭电源。 下表列出了以 C 和 汇编语言提供的符号名称及其相关值：

[+] Feedback

符号名称	值
ADCINC_OFF	0
ADCINC_LOWPOWER	1
ADCINC_MEDPOWER	2
ADCINC_HIGHPOWER	3

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

ADCINC_SetPower

说明：

设置开关电容 PSoC 模块的功率水平。

C 原型：

```
void  ADCINC_SetPower (BYTE bPowerSetting)
```

汇编：

```
mov   A, bPowerSetting
lcall  ADCINC_SetPower
```

参数：

bPowerSetting： 与用于起始进入点的 bPowerSetting 参数相同。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

ADCINC_Stop

说明：

将开关电容 PSoC 模块的功耗水平设置为 “ 关闭 ”。

C 原型：

```
void  ADCINC_Stop (void)
```

汇编：

```
lcall  ADCINC_Stop
```


[+] Feedback

参数：

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

ADCINC_GetSamples

说明：

针对指定数量的采样，运行 ADC。

C 原型：

```
void  ADCINC_GetSamples (BYTE bNumSamples)
```

汇编：

```
mov   A, bNumSamples
lcall  ADCINC_GetSamples
```

参数：

bNumSamples：用于设置要转换的采样数的 8-bit 值。 值 “0” 会导致 ADC 连续运行。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

ADCINC_StopADC

说明：

立即停止 ADC。 PWM 继续运行。

C 原型：

```
void  ADCINC_StopADC (void)
```

汇编：

```
lcall  ADCINC_StopADC
```

参数：

无

返回值：

无

[+] Feedback

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

ADCINC_fIsDataAvailable

说明：

检查采样数据的可用性。

C 原型：

```
BYTE  ADCINC_fIsDataAvailable(void)
```

汇编：

```
lcall  ADCINC_fIsDataAvailable
```

参数：

无

返回值：

如果数据已转换且可以读取，则返回非零值。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

ADCINC_iGetData

说明：

以有符号整数的形式返回转换的数据。必须调用 ADCINC_bfIsDataAvailable() 才能验证数据采样是否已就绪。

C 原型：

```
INT   ADCINC_iGetData(void)
```

汇编：

```
lcall  ADCINC_iGetData           ; Data will be in A and X upon return
```

参数：

无

返回值：

以 16-bit 2 的补码格式返回转换的数据采样。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

[+] Feedback

ADCINC_wGetData

说明：

以无符号整数的形式返回转换的数据。必须调用 ADCINC_bfIsDataAvailable() 才能验证数据采样是否已就绪。

C 原型：

```
WORD  ADCINC_wGetData(void)
```

汇编：

```
lcall  ADCINC_wGetData           ; Data will be in A and X upon return
```

参数：

无

返回值：

返回转换的 16-bit 无符号数据采样。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

ADCINC_cGetData

说明：

以有符号字符的形式返回转换的数据。必须调用 ADCINC_bfIsDataAvailable() 才能验证数据采样是否已就绪。

C 原型：

```
CHAR  ADCINC_cGetData(void)
```

汇编：

```
lcall  ADCINC_cGetData           ; Data will be in A upon return
```

参数：

无

返回值：

以 8-bit 2 的补码格式返回转换的数据采样。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

[+] Feedback

ADCINC_bGetData

说明：

以无符号字符的形式返回转换的数据。必须调用 ADCINC_bfIsDataAvailable() 才能验证数据采样是否已就绪。

C 原型：

```
BYTE  ADCINC_bGetData(void)
```

汇编：

```
lcall  ADCINC_bGetData           ; Data will be in A upon return
```

参数：

无

返回值：

返回转换的 8-bit 无符号数据采样。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

ADCINC_iClearFlagGetData

说明：

清除数据就绪标志，以有符号整数的形式获取转换的数据。 查看数据标志是否仍处于复位状态。 如果不是，则再次检索数据。 这可以确保 ADC 中断子程序在收集应答时不会更新应答。

C 原型：

```
INT   ADCINC_iClearFlagGetData(void)
```

汇编：

```
lcall  ADCINC_iClearFlagGetData   ; Data will be in A and X upon return
```

参数：

无

返回值：

以 16-bit 2 的补码格式返回转换的数据采样。

副作用：

全局变量 ADCINC_bfStatus 设置为零。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

[+] Feedback

ADCINC_wClearFlagGetData

说明：

清除数据就绪标志，以无符号整数的形式获取转换的数据。 查看数据标志是否仍处于复位状态。 如果数据标志不是处于复位状态，则再次检索数据。 这可以确保 ADC 中断子程序在收集应答时不会更新应答。

C 原型：

```
WORD  ADCINC_wClearFlagGetData(void)
```

汇编：

```
lcall  ADCINC_wClearFlagGetData   ; Data will be in A and X upon return
```

参数：

无

返回值：

返回转换的 16-bit 无符号数据采样。

副作用：

全局变量 ADCINC_bfStatus 设置为零。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

ADCINC_cClearFlagGetData

说明：

清除数据就绪标志，以有符号字符的形式获取转换的数据。 查看数据标志是否仍处于复位状态。 如果不是，则再次检索数据。 这可以确保 ADC 中断子程序在收集应答时不会更新应答。

C 原型：

```
CHAR  ADCINC_cClearFlagGetData(void)
```

汇编：

```
lcall  ADCINC_cClearFlagGetData   ; Data will be in A upon return
```

参数：

无

返回值：

以 8-bit 2 的补码格式返回转换的数据采样。

副作用：

全局变量 ADCINC_bfStatus 设置为零。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

[+] Feedback

ADCINC_bClearFlagGetData

说明：

清除数据就绪标志，以无符号字符的形式获取转换的数据。 查看数据标志是否仍处于复位状态。 如果不是，则再次检索数据。 这可以确保 ADC 中断子程序在收集应答时不会更新应答。

C 原型：

```
BYTE  ADCINC_bClearFlagGetData(void)
```

汇编：

```
lcall  ADCINC_bClearFlagGetData   ; Data will be in A upon return
```

参数：

无

返回值：

返回转换的 8-bit 无符号数据采样。

副作用：

全局变量 ADCINC_bfStatus 设置为零。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

ADCINC_fClearFlag

说明：

返回数据可用变量的内容，并复位标志。

C 原型：

```
BYTE  ADCINC_fClearFlag(void)
```

汇编：

```
lcall  ADCINC_fClearFlag
```

参数：

无

返回值：

这将返回状态寄存器的值。

副作用：

全局变量 ADCINC_bfStatus 设置为零。可以通过此函数的此实现或将来实现修改 A 和 X 寄存器。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

[+] Feedback

ADCINC_WritePulseWidth

说明：

更改 PWM 的脉冲宽度。

C 原型：

```
void  ADCINC_WritePulseWidth(BYTE bPulseWidth)
```

汇编：

```
mov   A, bPulseWidth
lcall  ADCINC_WritePulseWidth
```

参数：

bPulseWidth: 此值设置 PWM 的宽度。 此值不能为零，否则 ADC 停止工作。

返回值：

无。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

固件源代码示例

下面的代码示例轮询 Flag 寄存器，并将数据发送到用于将数据移出 I/O 引脚之一的子程序。

```
;;; Sample Code for the ADCINC
;;; Continuously Sample and Output Data to a pin.
;;;
;;; The user must provide the function to shift the data out.
;;;
include "m8c.inc"                ; part specific constants and macros
include "PSoCAPI.inc"            ; PSoC API definitions for all User Modules
export _main
_main:
   M8C_EnableGInt                  ; enable global interrupts
   mov  a,ADCINC_HIGHPOWER         ; set Power
   call ADCINC_Start
   mov  a,00h                      ; set ADC to continuous sampling
   call ADCINC_GetSamples
loop1:
wait:
   call ADCINC_fIsDataAvailable    ; poll flag
   jz wait
   call ADCINC_iClearFlagGetData   ; reset flag and retrieve data
   ;; call shift_it_out            ; (user provided) send data to output pin
   jmp loop1
```


[+] Feedback

下面是以 C 语言编写的相同项目：

```
//-------------------------------------------------------------------------
//  Sample C Code for the ADCINC
//  Continuously Sample input voltage
//
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
INT  iData;
void main(void)
{
   M8C_EnableGInt;                        // Enable Global Interrupts
   ADCINC_Start(ADCINC_HIGHPOWER);        // Apply power to the SC Block
   ADCINC_GetSamples(0);                  // Have ADC run continuously
   for(;;){
      while(ADCINC_fIsDataAvailable() == 0);    // Loop until value ready
      ADCINC_iClearFlagGetData();               // Clear ADC flag and get data
      // Add user code here to use or display result
   }
 }
```

配置寄存器

Table 5. 由 “ADC” 模拟开关电容 PSoC 模块使用的寄存器

寄存器	7	6	5	4	3	2	1	0
CR0	1	ClockPhase	0	ACap				
CR1	PosInputSource			NegInputGain				
CR2	0	1	AZ	0	0	0	0	0
CR3	1	1	1	FSW0	NegInputSource		0	0

ADC 使用一个或两个开关电容 PSoC 模块。 这些模块配置为生成模拟调制器。 要生成调制器，需要将这些模块配置为带有参考反馈的积分器，该积分器将输入值转换为数字脉冲流。 输入复用器确定将对哪些信号数字化。

ClockPhase 控制开关电容模块中的比较器时钟相位，以及开关的时钟相位。

ACap 包含与电容 ACap 的 32 种可能电容大小相对应的二进制编码。

InputSource 字段选择由转换器数字化的输入信号。 此参数在器件编辑器中设定。

PWM 中断处理器和各种 API 使用 AZ 和 FSW0 来复位积分器。

Table 6. PWM 数字 PSoC 模块使用的寄存器

寄存器	7	6	5	4	3	2	1	0
功能	0	0	1	1	0	0	0	1
输入	0	0	0	1	时钟			
输出	0	0	0	0	0	0	0	0

[+] Feedback

寄存器	7	6	5	4	3	2	1	0
DR0	定时器倒计数值 （API 从不访问）							
DR1	1	1	1	1	1	1	1	1
DR2	脉冲宽度							
CR0	0	0	1	0	0	0	0	启用

PWM 是配置为具有 256 个计数周期的定时器的数字 PSoC 模块。 在中断时，读取抽取滤波器并计算 ADC 值。

时钟从 16 个源之一选择输入时钟。 此参数在器件编辑器中设定。

Note 还必须使用选择的源来控制 ADC 模块驻留的列的模拟时钟。

设置为 “ 启用 ” 后，PWM 便可以工作了。 它由 ADCINC API 修改和控制。

Table 7. 抽取控制寄存器

位	7	6	5	4	3	2	1	0
DEC_CR0	0	0	0	0	ICLKS0	0	DCol	DCLKS0
DEC_CR1	1	1	0	0	ICLKS1			DCLKS1
DEC_DH	抽取滤波器的高字节输出							
DEC_DL	抽取滤波器的低字节输出							

[+] Feedback

版本历史记录

版本	创作者	说明
1.1	DHA	增加了 DRC 以检查是否出现下列情况： 1. 数字和模拟资源的源时钟不同。 2. ADC 时钟大于 CPU 时钟。

Note PSoC Designer 5.1 在所有的用户模块数据表中提供版本历史记录。 本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2002-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.