三重输入 7- 至 13-Bit 增量型 ADC 数据表 TriADC V 1.2

Copyright © 2001-2010 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O 和 ADC 输入）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27xxx、CY8C28x43、CY8C28x52、CY8CLED08/16、CY8C28x45、CY8CPLC20、CY8CLED16P01	5	0	3	431	11	1
CY8C26/25xxx	5	0	3	628	11	1

请参见 器 AN2239，ADC 选择指南的信息。

如需获取一个或多个使用此用户模块的完整配置功能性示例项目，请转到 www.cypress.com/psocexampleprojects.

特性与概述

- 同时对三个输入进行采样
- 7- 至 13-bit 分辨率，2 的补码或无符号整数
- 从 4 到高于 10,000 sps 的采样率
- 最大输入范围 （Vss 到 Vdd）
- 积分转换器提供良好的正常模抑制
- 内部或外部时钟

TriADC 是一个三重输入积分模数转换器 (ADC)，拥有 7 到 13 位的可调分辨率。 可通过优化集成时间对其进行配置，以移除不需要的高频率。 输入电压范围 （包括轨至轨）可通过配置合适的参考电压和模拟接地进行测量。 基于集中在 AGND 处的输入电压 （ - Vref 和 +Vref 之间），输出可配置为 2 的补码或无符号整数。

采样率范围从 4 到 10,000 sps，具体取决于分辨率、数据时钟和 CalcTime 参数的选择。

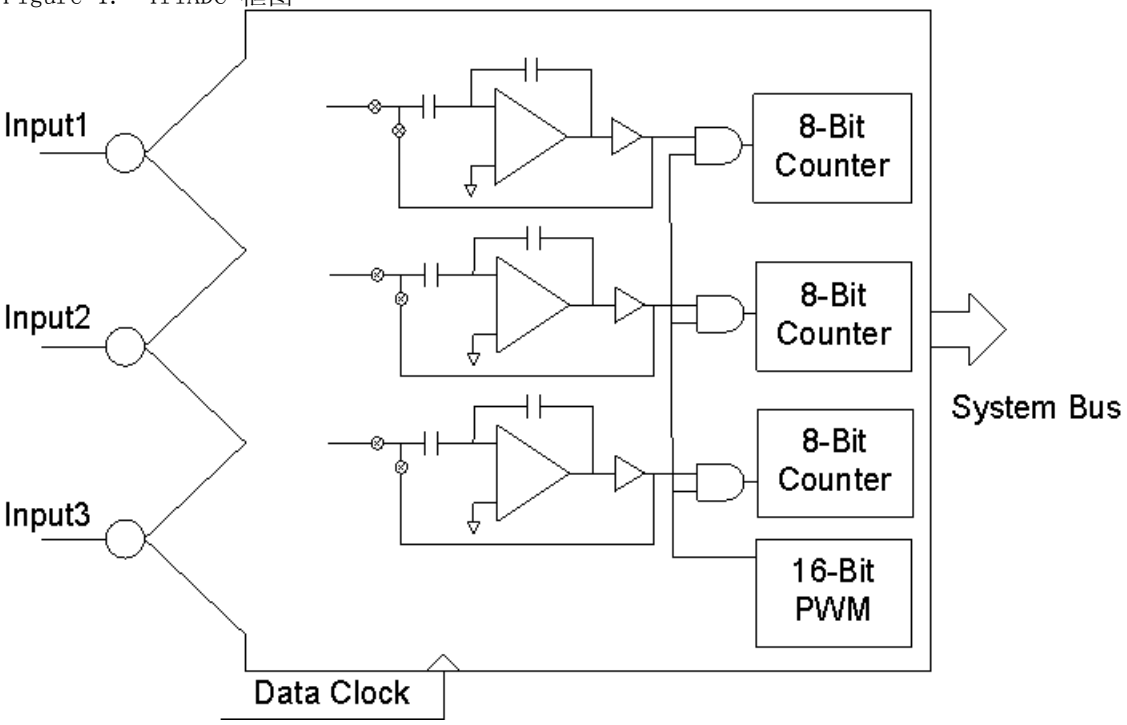
编程接口允许用户指定需要转换的顺序样本的数量或选择连续采样。CPU 负荷根据输入电平而变化。 例如，当 V_{in} = $+V_{ref}$，将会有 14,972 个 CPU 周期（最多 13 位）。 当 V_{in} = AGND，将会有 7,868 个 CPU 周期 （平均 13 位）。 当 V_{in} = $-V_{ref}$，将会有 764 个 CPU 周期 （最少 7-13 位）。

TriADC 除了比 ADCINCVR 多两个数字和模拟 SC 模块，其他操作完全相同。 由于三个输入通道由共同的信号控制，因此将同时进行采样，且采样持续时间也相同。 三个输入通道的电源设置、分辨率和速率均相同。

对于需要同步采样三个信号的应用程序，例如三相电压测量，TriADC 是理想的选择。 正如其他 PSoC ADC 一样，两个输入的信号可以复用。 请注意，在放置模块之前您需要查看 “ 参数 ” 部分。

Note 最初选择 TriADC 时，可能会出现警告，显示 “ 资源分配阻止放置 ”。 如果原始放置在同一列中有两个 ADC 模块，则将会显示此警告。 只需将各 ADC 模块移至其自身的列即可。

Figure 1. TriADC 框图

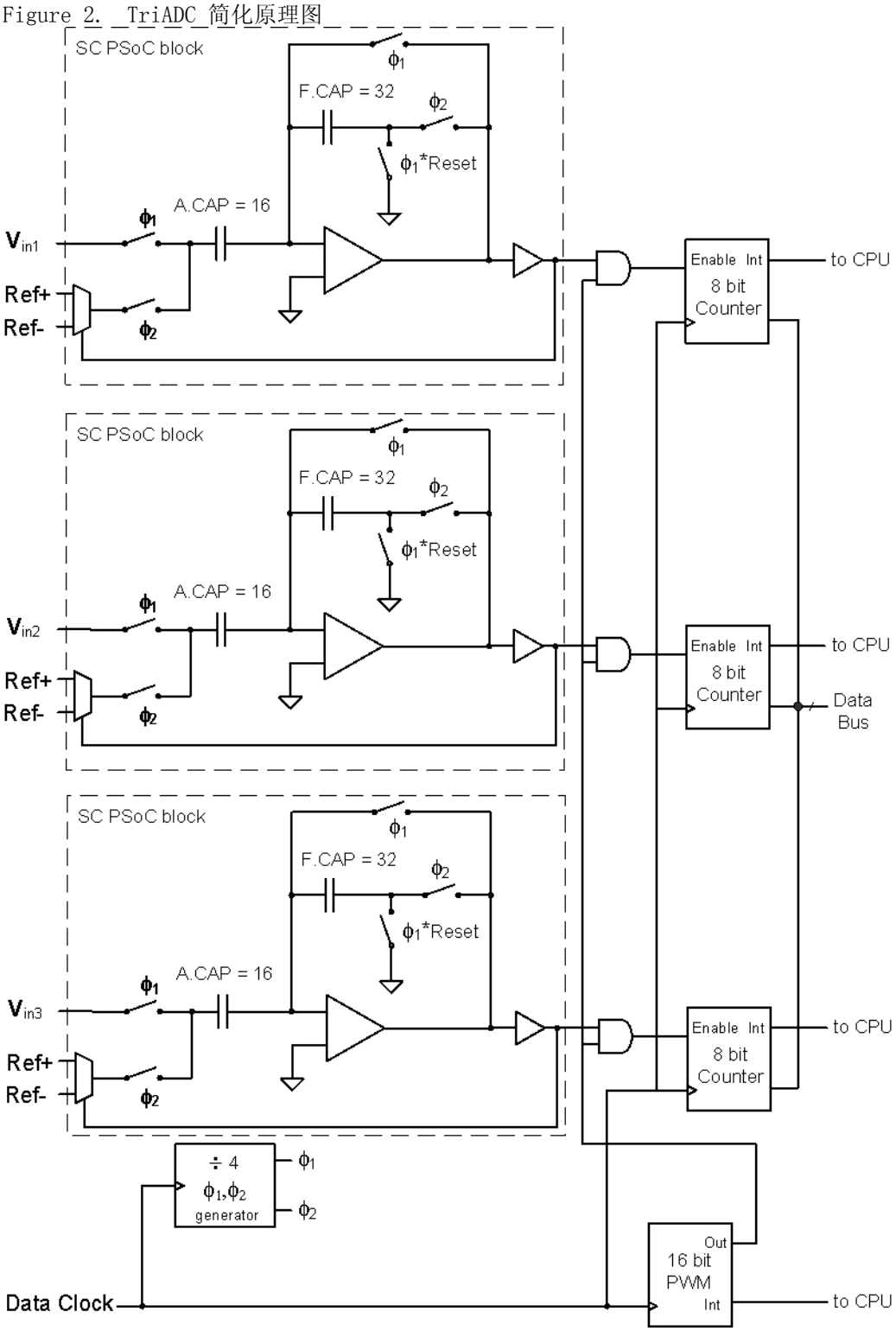


[+] Feedback

功能说明

在单个用户模块中，TriADC 是三个增量型 ADC。它们共享 16-bit 采样率定时器 (PWM)，以减少所需的数字模块。由于三个 ADC 使用同一个定时器，因此采样是完全同步的。如下图所示，总共需要五个数字 PSoC 模块和三个模拟开关电容 PSoC 模块。

Figure 2. TriADC 简化原理图



[+] Feedback

三个模拟模块均相同地配置成可复位积分器。 根据输出极性的不同来配置参考电压控制，以便在输入中增减参考电压，并置入积分器中。 此参考电压控制用于将积分器输出拉回至 AGND。 如果积分器操作 2^{Bits} 次，并且在这些操作中电压比较器输出为正的次数为 "n"，则输出的剩余电压 (V_{resid}) 为：

Equation 1

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

Equation 2

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

这个等式说明此 ADC 是 ± V_{ref}，分辨率 (LSB) 是 $V_{ref}/2^{Bits-1}$，计算后得到的输出电压就定义为剩余电压。 由于 V_{resid} 总是小于 V_{ref}，因此 $V_{resid}/2^{Bits}$ 小于 LSB 的一半且可以忽略不计。 得出的等式如下。

Equation 3

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

示例 1

对于 V_{ref} 为 1.3V 且分辨率为 8-bits 的情况，根据数据已就绪时从增量型 ADC 读取的值，我们可以很容易地计算出输入电压。 可以使用的等式如以下所示：

Equation 4

$$V_{in} = \frac{n - 128}{128} 1.3$$

计算结果相对于 AGND。 如果 ADC 数据值为 200，可计算出测量电压为 0.73V，计算方法如下：

Equation 5

$$V_{in} = \frac{200 - 128}{128} 1.3 = 0.73V$$

计算出的值是一个理想值，根据系统噪声和芯片偏移的不同，此值将很有可能不同。

如果已知特定输入电压，要确定期望的代码，我们可以重新排列该等式：

Equation 6

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

[+] Feedback

示例 2

对于 V_{ref} 为 1.3V 且分辨率为 8-bits 的情况，根据输入电压，我们可以很容易地计算出期望的 ADC 代码。 可以使用的等式如以下所示：

Equation 7

$$n = \frac{128 \cdot V_{in}}{1.3} + 128$$

对于 AGND 之下 -1V 输入电压，根据以下计算，可以预计 ADC 中的代码为 29.53：

Equation 8

$$n = \frac{128 \cdot (-1)}{1.3} + 128 = 29.53$$

计算出的值是一个理想值，根据系统噪声和芯片偏移的不同，此值将很有可能不同。

要使积分器作为增量型 ADC 使用，需利用以下数字资源：

- 8-bit 计数器，用于累计输出为正的周期数 （每个通道一个）。
- 16-bit PWM，用于计时集成时间并将时钟导入 8-bit 计数器 （三个通道间共享）。

将单个 DataClock 连接至 8-bit 计数器、16-bit PWM 以及连接至模拟 SC PSoC 模块的模拟列时钟。 模拟列时钟实际上是两个从 DataClock 中生成的时钟， Φ_1 和 Φ_2。 这两个附加时钟的频率刚好是 DataClock 频率的四分之一。 这意味着 PWM 和计数器的运行速度比所需速度快 4 倍，因此需要累计相当于 N+2 位的数据 （N 等于分辨率的位数）。

Note 放置此模块时，必须为所有三个模块配置同一个时钟。 否则，将导致运行错误。

计数器使用 8-bit 数字模块实现 LSB，而使用软件计数器实现 MSB。 每次硬件计数器溢出时将生成中断，且计数器的前端 MSB 将递增。 这就允许 TriADC 模块仅由五个而非八个数字模块实现。

Equation 9

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

采样率等于 DataClock 除以集成时间加上计算结果需要的时间 (CalcTime)。 集成时间是指 TriADC 对输入信号进行采样的周期。

计算结果需要的时间 CalcTime 与 CPU 时钟成反比变化。 CalcTime 的设定值必须大于计算结果所需要的时间。 最小 CalcTime 等于 371 个 CPU 周期，并且必须以 DataClock 来表示。 也可增加 CalcTime 至超过最小值，以优化采样率。

Note 2^{Bits+2} 加上 CalcTime 的总和不得超过 $2^{16}-1$ 或 65,535。

Equation 10

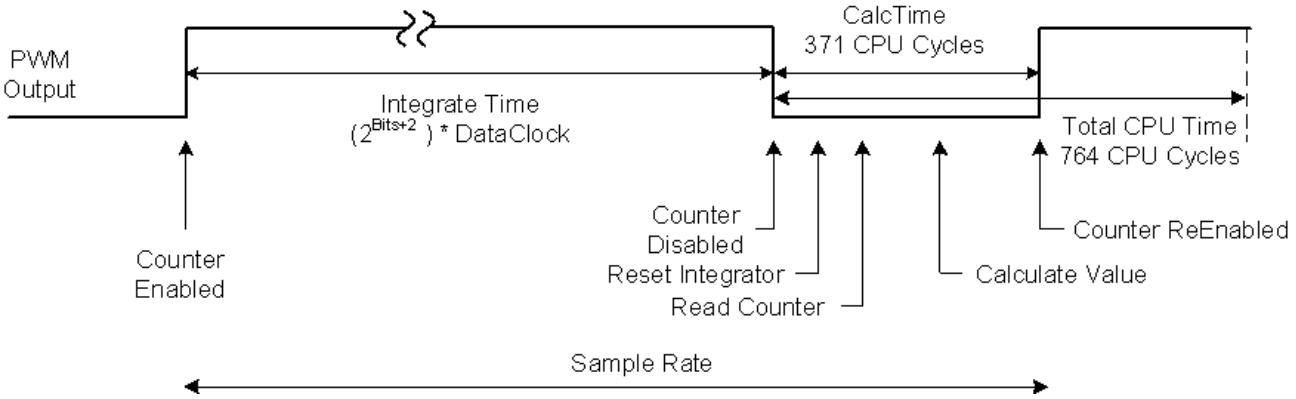
$$CalcTime \geq \frac{DataClock \cdot 371}{CPUClock}$$

16 位 PWM 已编程为输出 2^{Bits+2} 倍于 DataClock 的高信号。 例如，如果分辨率设置为 10 位，则 PWM 输出将保持 4096 (2^{10+2}) 个 DataClock 周期的高电平。 在进行最小结果计算和复位积分器时，PWM 输出将

[+] Feedback

为低电平。 由 CalcTime 参数控制此周期。 还可调整 CalcTime，以便与 DataClock 配合提供更精确的采样率。 PWM 的总周期是集成时间与 CalcTime 的总和。

Figure 3. 与 PWM 输出相关的 TriADC 时序



在进行首次读取时，将计算 PWM 配置，复位积分器，并将所有计数器复位至 FFh。 初始延迟将始终至少为计算时间的延迟。 仅在首次读取前对 PWM 进行初始化。 比较寄存器和周期寄存器一旦设置以后，就不需要对其进行重新初始化，除非分辨率或计算时间改变。 当 PWM 计数小于或等于集成值时，输出将变高，以使 8-bit 计数器递减。 PWM 将保持高输出直到计数器达到零。 此时，8-bit 计数器的时钟将禁用，并生成 PWM 中断。

8-bit 软件计数器的初始值设置为最大负值的 $2^{Bits}/64$ 倍。 每次 8-bit 计数器溢出时，将执行 8-bit 计数器中断，且软件计数器将递增 1。

当 ADC 的输入大于或等于最大正值时，8-bit 计数器将在 DataClock 每次发生正跃变时递增。 如果 ADC 的输入小于或等于最大负输入值，8-bit 计数器将不会递减，因此将不会生成中断。 在理想情况下，接近模拟接地的输入将使计数器总是递增。 很容易看出，取决于输入电压电平，8-bit 计数器中的中断数将在 0 到 $(2^{Bits+2})/256$ 个之间变化。 例如，如果分辨率设置为 10 位，则 PWM 比较值将设置为 2^{10+2} (4096)。 这意味着在集成周期内处理器最多可能被中断 4096/256 即 16 次。

基于 TriADC 控制中断以及为获取更高分辨率结果的长时间采样，在处理采样的过程中，期望处理器等待是不切实际的。 ADC 子程序与主程序之间的主要通信是一个可以轮询的标志。 当 TriADC_bfStatus 的最高有效位为非零值时，则 TriADC_iResultn (n=1,2,3) 中的新数据可用。 可使用 API 检查数据标志和检索数据。

此数据处理程序的设计原理为基于轮询。 如果需要基于中断的数据处理程序，用户可以将自己的数据处理程序代码插入到中断子程序 TriADC_CNTn_ISR 中，此子程序位于汇编文件 TriADC*INT.asm* 中。 最佳插入代码的位置已做明显标记。

通道之间的区别

使用 TriADC，测量同一个输入电压时，通道之间会有区别。 出现区别的原因是开关电容模块放大器和列 AGND 缓冲区中的输入偏移变化。 通过将同一个信号路由至各个 ADC 通道中，即可补偿通道间的偏移。 可将其中一个通道用作参考，随后的通道间的区别将在每次读数后消除。

[+] Feedback

CPU 利用率

TriADC 需要 CPU 时间来计算结果，并在每次硬件计数器溢出时递增软件计数器。 CPU 开销取决于三个变量： CPU 时钟、DataClock 和输入电压。 输入电压会影响 ADC 的 CPU 开销，起初看起来这可能有点奇怪。 接近或低于 -Vref 的输入电压需要很少的 CPU 开销。 接近或高于 +Vref 的输入电压要求更多的 CPU 开销。 下面的等式假设三个输入的输入信号都相同。 为给定的输入计算所需的 CPU 周期：

Equation 11

$$CPUcycles = PWM16_IRQ_CPUcycles + \left(\frac{2^{Bits}}{64} \cdot \left(\frac{Vref + Vin}{2 \cdot Vref}\right) \cdot (Counter_IRQ_CPUcycles \cdot 3)\right)$$

Equation 12

$$CPUcycles = 764 + \left(\frac{2^{Bits}}{64} \cdot \left(\frac{Vref + Vin}{2 \cdot Vref}\right) \cdot (37 \cdot 3)\right)$$

要计算 10-bits 分辨率时的最大 CPU 周期，则将 Vin 设置为 Vref。

Equation 13

$$'PUcycles = 764 + \left(\frac{2^{10}}{64} \cdot \left(\frac{Vref + Vref}{2 \cdot Vref}\right) \cdot 111\right) = 764 + (16 \cdot 1 \cdot 111) = 254$$

要计算 TriADC 的 CPU 利用率，可以使用以下等式。

Equation 14

$$Percent_CPU_Utilization = \frac{Sample_Rate \cdot CPUcycles}{CPU_frequency} \cdot 100$$

将分辨率设置为 10 位 （如上例），采样率设置为 1000 次采样 / 秒，CPU 时钟设置为 12 MHz，然后 （如以下等式）得出大约利用了 21% 的 CPU。

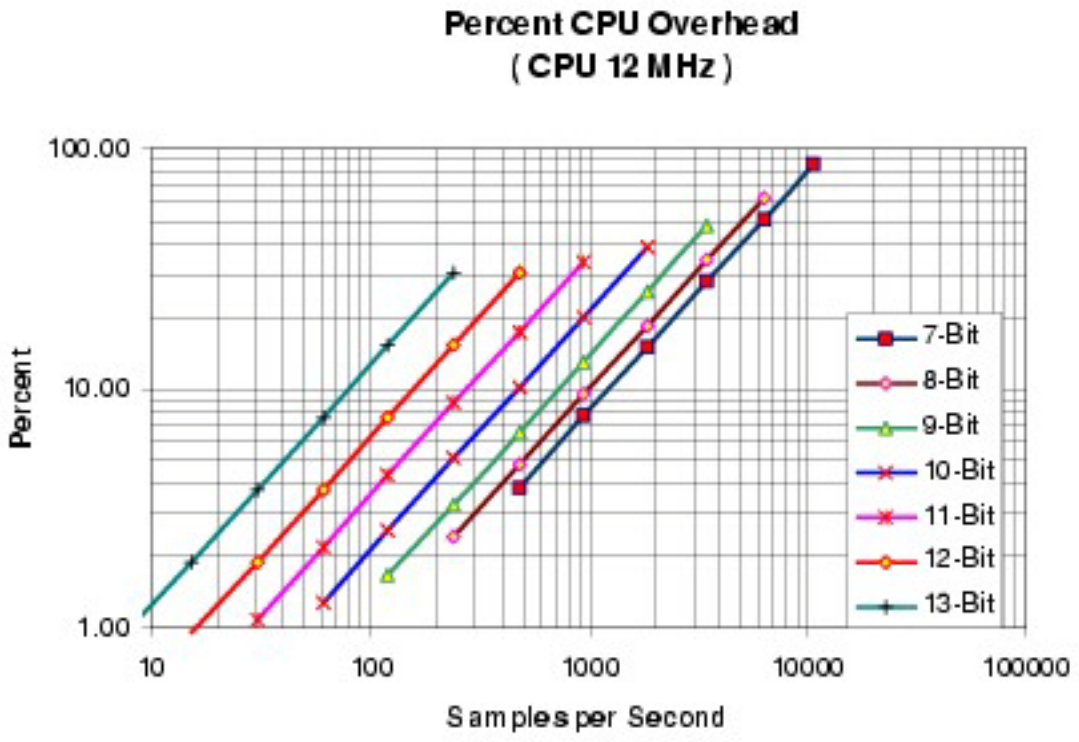
Equation 15

$$Percent_CPU_Utilization = \frac{1000 \cdot 2540}{12MHz} \cdot 100 = 21\%$$

[+] Feedback

下图展示了所支持的采样率和分辨率的 CPU 利用率。 CPU 速率默认设置为 12 MHz。

Figure 4. 所支持的采样率和分辨率的 CPU 利用率



频率抑制

通过选择合适的集成时间，可以抑制部分噪声源。 为抑制噪声源及其谐波，请选择等于噪音信号整数周期的集成时间。 如果要抑制多个信号，请选择等于两个信号的整数周期的集成时间。

例如，如果要抑制由 50 Hz 和 60 Hz 信号引起的噪声，则选择包含整数个 50 Hz 和 60 Hz 信号的周期。

Equation 16

$$IntegrateTime = 6 \cdot \frac{1}{60} = 5 \cdot \frac{1}{50} = 100mSec$$

IntegrateTime 为 100 ms 时，将同时抑制 50 Hz 和 60 Hz 的信号以及这些信号的任何谐波。 接下来，计算产生合适的 IntegrateTime 所需要的 DataClock。

Equation 17

$$DataClock = \frac{2^{Bits+2}}{IntegrateTime}$$

请注意，在此计算中未使用 CalcTime，尽管它对采样率有影响。 IntegrateTime 是指 TriADC 正在对输入电压进行实际采样的周期。 采样率基于 IntegrateTime 以及计算结果需要的时间。

示例

[+] Feedback

某给定的应用程序要求 IntegrateTime 为 100 ms 且 A/D 分辨率为 13 位。对于 100 ms 的 IntegrateTim，数据时钟必须为：

Equation 18

$$DataClock = \frac{2^{Bits+2}}{IntegrateTime} = \frac{2^{13+2}}{100ms} = 327.7kHz$$

与数据时钟相关的 CalcTime 必须根据 DataClock 和 CPU 时钟计算得出。如果 CPU 时钟为 12 MHz，则最短计算时间为：

Equation 19

$$CalcTime = \frac{DataClock \cdot 371}{CPUClock} = \frac{327.7kHz \cdot 371}{12000kHz} = 10_DataClocks$$

此 CalcTime 应四舍五入为最接近的整数（此例中为 10）。现在确定采样率：

Equation 20

$$SampleRate = \frac{DataClock}{2^{13+2} + CalcTime} = \frac{327.7kHz}{32768 + 10} = 9.99\ Samples/Second$$

如果需要更长的采样率，可以增大 CalcTime，最大为 CalcTime + 2^{13+2} 不超过 2^{16} -1 (65535)。

直流和交流电气特性

The following values are indicative of expected performance and based on initial characterization data. 如下表中无特别指明，均为 T_A = 25° C、Vdd = 5.0V、功耗高、运算放大器偏压低、P2[4] 上的输出相对于 2.5V 外部模拟接地，P2[6] 上为 1.25 外部 Vref，分辨率设为 13 位。

Table 1. 5.0V TriADC 的直流和交流电气特性，PSoC 器件的 CY8C27/24/22xxxFamily

参数	典型值	极限值	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		7 到 13	位	
采样率		4 到 10,000	sps	
信噪比	77	---	dB	
直流精度				
微分非线性误差	.25	---	LSB	列时钟 2 MHz
积分非线性误差	1.0	---	LSB	
偏移误差	9	---	mV	

[+] Feedback

参数	典型值	极限值	单位	条件和注释
增益误差				
包括参考增益误差	3.0	--	% FSR	
不含参考增益误差[2]	0.1	--	% FSR	
工作电流				
低功耗	500	---	µA	
中功耗	1600	---	µA	
高功耗	6000	---	µA	
数据时钟	---	0.125 到 8.0	MHz	输入到数字模块和模拟列时钟

The following values are indicative of expected performance and based on initial characterization data. 如下表中无特别指明，均为 T_A = 25° C、Vdd = 3.3V、功耗高、运算放大器偏压低、P2[4] 上的输出相对于 1.64V 外部模拟接地，P2[6] 上为 1.25 外部 Vref，分辨率设为 13 位。

Table 2. 3.3V TriADC 的直流和交流电气特性，PSoC 器件的 CY8C27/24/22xxxFamily

参数	典型值	极限值	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)	---		
分辨率		7 到 13	位	
采样率		4 到 10,000	sps	
信噪比	77	---	dB	
直流精度				
微分非线性误差	.25	---	LSB	列时钟 2 MHz
积分非线性误差	1.0	---	LSB	
偏移误差	4	---	mV	
增益误差				
包括参考增益误差	3.0	--	% FSR	
不含参考增益误差[2]	0.4	--	% FSR	
工作电流				

[+] Feedback

参数	典型值	极限值	单位	条件和注释
低功耗	440	---	µA	
中功耗	1500	---	µA	
高功耗	5700	---	µA	
数据时钟	---	0.125 到 8.0	MHz	输入到数字模块和模拟列时钟

电气特性注释

1. 包括 I/O 引脚。
2. 通过比较外部参考与路由经过测试复用器并回到引脚的 $V_{RefHigh}$ 和 V_{RefLow}，来测量参考增益误差。

如下表中无特别指明，所有限制均保证 T_A = -40° C 到 +85° C、Vdd = 5.0V ± 10%、功耗高、运算放大器偏压低、P2[4] 上的输出相对于 2.5V 外部模拟接地，P2[6] 上为 1.25 外部 Vref，分辨率设为 12 位。

Table 3. 5.0V TriADC 的直流和交流电气特性，PSoC 器件的 CY8C26/25xxxFamily

参数	典型值[1]	极限值	单位	条件和注释
输入				
输入电压范围[2]	---	Vss 到 Vdd		Ref Mux = Vdd/2 ± Vdd/2
输入电容[3]	0.8	---	pF	
输入阻抗[4,5]	1/(C*clk)	---		
分辨率	---	7 到 13	位	2 的补码
采样率	---	4 到 10,000[6]	sps	每秒采样数
信噪比[7]	68		dB	在 100 sps 时
直流精度				
积分非线性误差	0.5	1	LSB	
微分非线性误差	0.25	0.5	LSB	
偏移误差	12	49	mV	使用外部 AGND
增益误差	0.5	1.5	% FSR	相对于参考输入
工作电流				
低功耗	250	---	µA	
中功耗	600	---	µA	
高功耗	1920	---	µA	
数据时钟	---	0.125 到 8.0[6]	MHz	输入到数字模块和模拟列时钟

[+] Feedback

如下表中无特别指明，所有限制均保证 T_A = -40° C 到 +85° C、Vdd = 3.0 到 3.6V、功耗高、运算放大器偏压低、P2[4] 上的输出相对于 1.64V 外部模拟接地，P2[6] 上为 1.25 外部 Vref，分辨率设为 12 位。

Table 4. 3.3V TriADC 的直流和交流电气特性，PSoC 器件的 CY8C26/25xxxFamily

参数	典型值[1]	极限值	单位	条件和注释
输入				
输入电压范围[2]	---	Vss 到 Vdd		Ref Mux = Vdd/2 ± Vdd/2
输入电容[3]	0.8	---	pF	
输入阻抗[4,5]	1/(C*clk)	---		
分辨率	---	7 到 13	位	2 的补码
采样率	---	4 到 10,000[6]	sps	每秒采样数
信噪比[7]	65		dB	在 100 sps 时
直流精度				
积分非线性误差	0.5	1	LSB	
微分非线性误差	0.25	0.5	LSB	
偏移误差	12	49	mV	
增益误差	0.5	2.5	% FSR	相对于参考输入
工作电流				
低功耗	220	---	µA	sps = 10，13 位分辨率
中功耗	520	---	µA	sps = 100，13 位分辨率
高功耗	1680	---	µA	sps = 240，13 位分辨率
数据时钟	---	0.125 到 8[6]	MHz	输入到数字模块和模拟列时钟

电气特性注释

1. 典型值表示在 +25° C 时的参数标准。
2. 输入电压超出最高值将生成最大的正读数。 输入电压超出最低值将生成最大的负读数。
3. 仅指用户模块，不含 I/O 引脚。
4. 输入电容或阻抗仅适用于模拟模块的输入直接发送到引脚的情况。
5. C = 输入电容，clk = 数据时钟 （模拟列时钟）。
6. 除非另有注明，否则规范为 100 sps 采样率以及 8 MHz 数据时钟。 采样率取决于数据时钟和分辨率。
7. 信噪比 = 全量程单音功率与集成在 Fsample/2 中的总噪声之比。

[+] Feedback

放置

ADC （开关电容）模块可置于任意开关电容 PSoC 模块中。 每一个模块都必须能独立驱动所放置到的特定列的比较器总线。 也就是说，三个模块中的每一个都必须位于不同的列，且不可与其他连接到比较器总线的开关电容模块共享列。

计数器模块可置于任何可用的数字模块中，但 PWM16 只能置于特定位置。 在 CY8C27xxx 和 CY8CLED08 器件系列中，可能用于 PWB16 （最低有效位 / 最高有效位）的位置是 DBB00/DBB01、DBB01/DCB02、DBB10/DBB11 和 DBB11/DCB12。 在 CY8C29/24/22xxx 和 CY8CLED04/16 器件系列中，PWM16 能置于任意两个连续的数字模块中。 对于 CY8C26/25xxx PSoC 器件而言，PWB16 （最低有效位 / 最高有效位）的合法位置是 DBA01/DBA02 和 DCA05/DCA06。

三个计数器模块和脉冲宽度调制模块均含有中断服务子程序。 计数器模块应当比 PWM16 模块具有更高的中断优先级。 因此，建议将计数器模块置于比 PWM16 模块更低的数字模块位置。

Note 最初选择 TriADC 时，可能会出现警告，显示 “ 资源分配阻止放置 ”。 如果原始放置在同一列中有两个 ADC 模块，则将会显示此警告。 只需将各 ADC 模块移至其自身的列即可。

参数和资源

ADC Input1、ADC Input2、ADC Input3

在完成模拟 PSoC 模块放置后，再进行输入选择。 八个开关电容模块具有不同的输入选择。 每一个模块都可与大部分相邻模块连接，部分可直接与外部输入引脚连接。 放置模拟数字模块时，必须考虑如何使其获得输入信号。 部分放置允许输入从封装引脚直接路由到输入。 这种直接连接实现了对 40 mV 供电轨内的输入进行精确测量。 信号也可路由经过列复用器之一、CT 模块测试复用器之一并抵达模拟列，在此处 TriADC 也可对功率供电轨附近的信号进行测量。 三个模数转换器输入各自均含有一个选择。

ClockPhase1、ClockPhase2、ClockPhase3

时钟相位的选择用于将一个开关电容模拟 PSoC 模块的输出与另一个模块的输入同步。开关电容模拟 PSoC 模块使用两相位时钟 （Φ_1、Φ_2）来获取和传输信号。 通常，TriADC 的输入是在 Φ_1（常规设置）上采样的。 这便产生一个问题：许多用户模块在 Φ_1 期间将其输出自动归零，并且仅在 Φ_2 期间给出有效输出。 如果此类模块的输出馈送到 TriADC 的输入，则 TriADC 将获取已自动归零的输出而非有效的信号。 时钟相位选择允许交换相位，因此输入信号是在 Φ_2（交换设置）期间获取的。 三个开关电容模块各自均含有一个选择。

ADCResolution

此选择实现了在器件编辑器中设置 TriADC 分辨率。 虽然 API 子程序可以设置或更改分辨率，但是如果可以在器件编辑器中完成设置，则无需使用 API 子程序。 使用 API 调用也可随时更改分辨率，但是更改后 TriADC 将停止，必须进行重启。 有效分辨率设置为 7 到 13 （含 7 和 13）。

CalcTime

CalcTime 表示下一次集成循环开始前 CPU 计算中间集成结果所花费的时间。 计算结果所花费的时间 “CalcTime” 与 CPU 时钟成反比。此值必须以数据时钟为依据。CPU 最短计算时间是 371 个 CPU 时钟。 也可增加 CalcTime 以优化采样率。

Note 应注意要确保 CalcTime + 2^{Bits+2} 不超过 $2^{16}-1$ 或 65,535。

下列等式用于确定 CalcTime 应设置的大小：

Equation 21

$$CalcTime \geq \frac{DataClock \cdot 371}{CPUClock}$$

[+] Feedback

下表显示的是 CalcTime 参数的可选范围。 使用上面的等式设置对于给定应用程序此参数可用范围的下限。

Table 5. CalcTime 参数范围

分辨率	集成时间 （DataClock 计数）	CalcTime 范围 （DataClock 计数）
7	512	1 到 65,023
8	1,024	1 到 64,511
9	2,048	1 到 63,487
10	4,096	1 到 61,439
11	8,192	1 到 57,343
12	16,384	1 到 49,151
13	32,768	1 到 32,767

例如，如果 DataClock 设为 1.5 MHz，CPU 在 12 MHz 下运行时，CalcTime 应不小于 47。 如以下等式所示。

Equation 22

$$CalcTime \geq \frac{DataClock \cdot 371}{CPUClock} = \frac{1.50MHz \cdot 371}{12.0MHz} = 46.4_DataClocks$$

DataClock 和积分器列时钟

DataClock 用于确定采样率和信号采样窗口。 此时钟必须路由至计数器模块、16-bit 脉冲宽度调制模块以及包含积分器列的列时钟的时钟输入。

此参数设置仅设置计数器模块和脉冲宽度调制模块的时钟。

Note 必须手动将积分器开关电容模块的列时钟设置为同一时钟。 为全部八个模块设置同一时钟非常重要，否则会造成此用户模块无法正常工作。

此时钟可以是时钟频率在 125 kHz 到 8 MHz 之间的任意源。

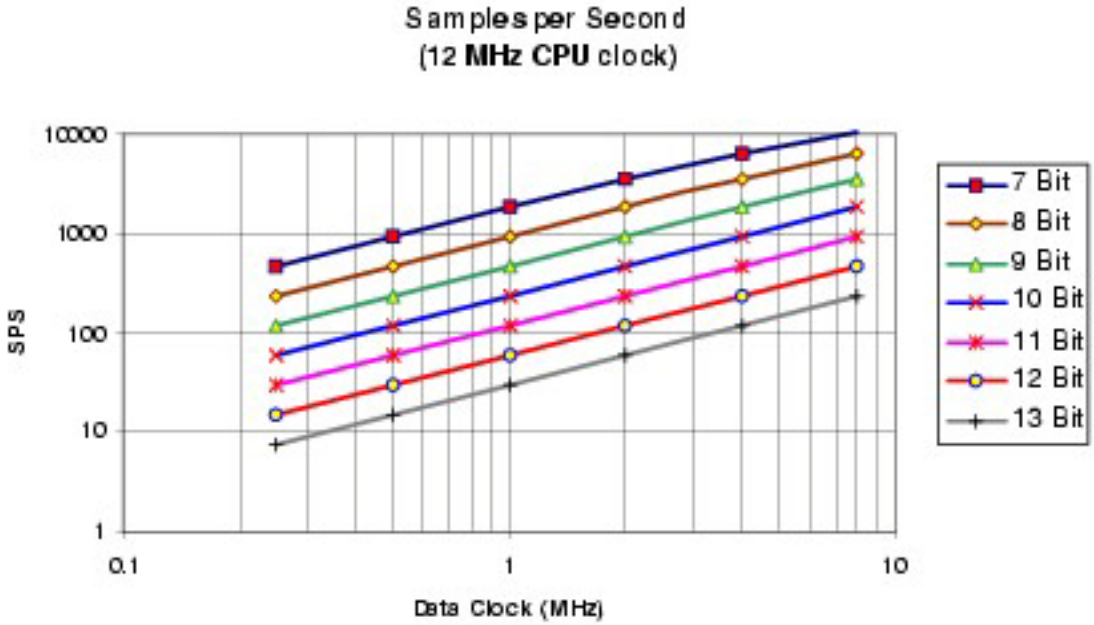
Equation 23

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

[+] Feedback

下图显示的是不同 TriADC 分辨率选项下的可能采样率。

Figure 5. TriADC 各个分辨率选项的采样率



参考复用器全局资源

器件编辑器 “ 全局资源 ” (Global Resource) 部分中的 “ 参考复用器 ” (Ref Mux) 的选择决定了可用输入电压范围。参考复用器的选择决定模拟接地以及有关模拟接地的输入电压可用范围。例如，如果选择 “Vdd/2 ± BandGap” 且 Vdd = 5 伏特，则可用输入范围为 2.5 +/- 1.3 伏特 (1.2 到 3.8 伏特)。下表显示的是 Vdd 为 5 伏特和 3.3 伏特时的有效范围。

Table 6. CY8C26/25xxx 参照 RefMux 设置的输入电压范围

RefMux 设置	Vdd = 5 伏特	Vdd = 3.3 伏特
(Vdd/2) ± BandGap	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
(Vdd/2) ± (Vdd/2)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
(2*BandGap) ± BandGap	$1.3 < V_{in} < 3.9$	不适用
(2*BandGap) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	不适用
P2[4] ± BandGap	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

[+] Feedback

Table 7. CY8C27/24/22xxx 参照各个参考复用器设置的输入电压范围

RefMux 设置	Vdd = 5 伏特	Vdd = 3.3 伏特
(Vdd/2) ± BandGap	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
(Vdd/2) ± (Vdd/2)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
BandGap ± BandGap	$0 < V_{in} < 2.6$	$0 < V_{in} < 2.6$
(1.6*BandGap) ± (1.6*BandGap)	$0 < V_{in} < 4.16$	不适用
(2*BandGap) ± BandGap	$1.3 < V_{in} < 3.9$	不适用
(2*BandGap) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	不适用
P2[4] ± BandGap	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

DataFormat

此选择决定返回结果所用的格式。 如果选择 “ 含符号 ” (Signed)，且分辨率选择 “N”，则结果范围在 2^{N-1} 到 $2^{N-1}-1$ 之间。 如果选择 “ 无符号 ”(Unsigned)，则结果在 0 到 $2^{N}-1$ 之间。 请参见下表中不同数据格式和分辨率的结果范围。

Table 8. 数据格式结果范围

分辨率设置	含符号的数据格式	无符号的数据格式
7	-64 到 63	0 到 127
8	-128 到 127	0 到 255
9	-256 到 255	0 到 511
10	-512 到 511	0 到 1023
11	-1024 到 1023	0 到 2047
12	-2048 到 2047	0 到 4095
13	-4096 到 4095	0 到 8191

中断生成控制

下列参数仅在选中了 PSoC Designer 中 “ 启用中断生成控制 ”(Enable interrupt generation control) 复选框时可用。该复选框位于 “ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。 **“ 启用中断生成控制 ”** 复选框时，有两个附加参数将变为可用。 此复选框位于 **“ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。**. 当拥有多个程序层而且多个用户模块在不同的程序层共享中断时，中断生成控制是非常重要的。

IntDispatchMode

IntDispatchMode 参数用于指定如何处理中断请求（当处于同一模块的多个用户模块在不同的程序层共享该中断时） 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程序层正处于活动状态。 这项测试发生在每次请求共享中断的时候。 这样会导致延迟增加并且会产生

[+] Feedback

一个服务于共享中断请求的非确定性的程序，但并不要求任何 RAM 资源。 选择 “OffsetPreCalc” 参数会导致固件只在最初已经有一个程序层运行时计算共享中断请求的来源。 这项计算减少了中断延迟会产生一个服务于共享中断请求的确定性程序，但其代价是 1 个字节的 RAM 资源。

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，使设计人员能够在较高的层级处理模块。本部分具体指明了每个函数的接口以及由 “ 包含 ” 文件所提供的常数。

Note 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能因调用 API 函数而更改。 发起调用 API 的函数有责任在调用之前保留 A 和 X 的值 （如果调用后需要再次用到它们）。 选择这种 “ 寄存器易变 ” 策略是为了提高效率，并且自从 PsoC Designer 的 1.0 版本起使用。 C 语言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。 虽然有些用户模块的 API 函数可能保留 A 和 X 不变，但并不保证在未来也将如此。

API 子程序能够初始化、配置、启动采样、停止以及读取来自模数转换器的结果数据。 在所有情况下，模块的 “ 实例名称 ” 都将取代下列输入点中显示的 “TriADC” 前缀。

TriADC_Start

说明：

执行此用户模块所需的所有初始化，并设置开关电容 PSoC 模块的功耗水平。

C 语言原型：

```
void  TriADC_Start (BYTE bPowerSetting)
```

汇编语言：

```
mov   A, TriADC_HIGHPOWER
lcall  TriADC_Start
```

参数：

PowerSetting: 用于指定功耗水平的 1 个字节。 在复位和配置之后，关闭分配给 TriADC 的模拟 PSoC 模块的电源。 下表给出了 C 语言和汇编语言中提供的符号名称及其相关数值。

符号名称	值
TriADC_OFF	0
TriADC_LOWPOWER	1
TriADC_MEDPOWER	2
TriADC_HIGHPOWER	3

功耗水平会影响模拟性能。 正确的功耗设置与数据时钟的采样率相关，必须为每个应用程序确定正确的功耗设置。 建议在开始开发时选择满功率。 稍后再进行测试，以确定功耗设置的最低值。

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 （CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

[+] Feedback

TriADC_SetPower

说明：

设置开关电容 PSoC 模块的功耗水平。

C 语言原型：

```
void  TriADC_SetPower (BYTE bPowerSetting)
```

汇编语言：

```
mov   A, [bPowerSetting]
lcall  TriADC_SetPower
```

参数：

PowerSetting: 与之前所述用于 “ 启动 ”(Start) API 子程序的 PowerSetting 参数相同。 用户在运行模数转换器时，也可更改功耗水平。

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_SetResolution

说明：

设置模数转换器的分辨率。

C 语言原型：

```
void  TriADC_SetResolution (BYTE bResolution)
```

汇编语言：

```
mov   A, [bResolution]
lcall  TriADC_SetResolution
```

参数：

分辨率： 可在器件编辑器或用户固件中设置模数转换器的分辨率。 若未在固件中设置，ADC 将使用器件编辑器中设置的默认分辨率。 分辨率的设置值范围为 7 到 13 位。

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

[+] Feedback

TriADC_Stop

说明：

将开关电容积分器模块的功耗水平设置为关闭。 在未使用 TriADC 的情况下用户想要降低功耗，则可使用此方法。 此子程序可降低模拟开关电容器模块的功耗并禁用数字模块。 为达到最低功耗水平，还应移去数字模块的时钟。

C 语言原型：

```
void  TriADC_Stop()
```

汇编语言：

```
lcall  TriADC_Stop
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TriADC_GetSamples

说明：

初始化并启动 ADC 算法以收集指定采样数。 请牢记要通过调用 *M8C.inc* 或 *M8C.h* 中的 M8C_EnableGInt 宏调用来启用全局中断。

C 语言原型：

```
void  TriADC_GetSamples (BYTE bNumSamples)
```

汇编语言：

```
mov   A, [bNumSamples]
lcall  TriADC_GetSamples
```

参数：

NumSamples：8-bit 值，用于设置要检索的采样数。 值为 “0” 时，将使 ADC 连续运行。

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_StopAD

说明：

立即暂停 ADC。

[+] Feedback

C 语言原型：

```
void  TriADC_StopAD()
```

汇编语言：

```
lcall  TriADC_StopAD
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TriADC_fIsDataAvailable, fIsData

说明：

若已完成数据转换，而且数据可读，则返回非零值。

C 语言原型：

```
CHAR  TriADC_fIsDataAvailable()
CHAR  TriADC_fIsData()
```

汇编语言：

```
lcall  TriADC_fIsDataAvailable
```

参数：

无

返回值：

若数据可用，则返回非零值。

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_iGetData1

说明：

为 ADC Input1 返回上次转换的数据。在获取数据之前应首先调用 fIsDataAvailable()，以确保数据有效。 必须在下一个转换周期结束之前检索数据，否则数据将会被覆盖。 如果恰好在集成周期结束时调用此函数，则返回数据可能被破坏。 因此强烈建议以高于采样率的频率检索数据。如果无法保证，请在调用此函数之前关闭中断。

C 语言原型：

```
INT   TriADC_iGetData1()
```

汇编语言：

```
lcall  TriADC_iGetData1
```


[+] Feedback

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 在 X 寄存器中返回，LSB 在累加器中返回。

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_iGetData2

说明：

为 ADC Input2 返回上次转换的数据。在获取数据之前应首先调用 fIsDataAvailable()，以确保数据有效。 必须在下一个转换周期结束之前检索数据，否则数据将会被覆盖。 如果恰好在集成周期结束时调用此函数，则返回数据可能被破坏。 因此强烈建议以高于采样率的频率检索数据。如果无法保证，请在调用此函数之前关闭中断。

C 语言原型：

```
INT   TriADC_iGetData2()
```

汇编语言：

```
lcall  TriADC_iGetData2
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 在 X 寄存器中返回，LSB 在累加器中返回。

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_iGetData3

说明：

为 ADC Input3 返回上次转换的数据。在获取数据之前应首先调用 fIsDataAvailable()，以确保数据有效。 必须在下一个转换周期结束之前检索数据，否则数据将会被覆盖。 如果恰好在集成周期结束时调用此函数，则返回数据可能被破坏。 因此强烈建议以高于采样率的频率检索数据。如果无法保证，请在调用此函数之前关闭中断。

C 语言原型：

```
INT   TriADC_iGetData3()
```

汇编语言：

```
lcall  TriADC_iGetData3
```

参数：

无

[+] Feedback

返回值：

返回转换的整数值。 在汇编程序中，MSB 在 X 寄存器中返回，LSB 在累加器中返回。

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_ClearFlag

说明：

清除 “ 数据可用 ”(Data Available) 标志。

C 语言原型：

```
void  TriADC_ClearFlag()
```

汇编语言：

```
lcall  TriADC_ClearFlag
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_iGetData1ClearFlag

说明：

为 ADC Input1 返回上次转换的数据并清除 “ 数据可用 ”(Data Available) 标志。在获取数据之前应首先调用 fIsDataAvailable()，以确保数据有效。 必须在下一个转换周期结束之前检索数据，否则数据将会被覆盖。 如果恰好在集成周期结束时调用此函数，则返回数据可能被破坏。 因此强烈建议以高于采样率的频率检索数据。如果无法保证，请在调用此函数之前关闭中断。

C 语言原型：

```
INT   TriADC_iGetData1ClearFlag()
```

汇编语言：

```
lcall  TriADC_iGetData1ClearFlag
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 在 X 寄存器中返回，LSB 在累加器中返回。

[+] Feedback

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_iGetData2ClearFlag

说明：

为 ADC Input2 返回上次转换的数据并清除 “ 数据可用 ”(Data Available) 标志。在获取数据之前应首先调用 fIsDataAvailable()，以确保数据有效。 必须在下一个转换周期结束之前检索数据，否则数据将会被覆盖。 如果恰好在集成周期结束时调用此函数，则返回数据可能被破坏。 因此强烈建议以高于采样率的频率检索数据。如果无法保证，请在调用此函数之前关闭中断。

C 语言原型：

```
INT   TriADC_iGetData2ClearFlag()
```

汇编语言：

```
lcall  TriADC_iGetData2ClearFlag
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 在 X 寄存器中返回，LSB 在累加器中返回。

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

TriADC_iGetData3ClearFlag

说明：

为 ADC Input3 返回上次转换的数据并清除 “ 数据可用 ”(Data Available) 标志。在获取数据之前应首先调用 fIsDataAvailable()，以确保数据有效。 必须在下一个转换周期结束之前检索数据，否则数据将会被覆盖。 如果恰好在集成周期结束时调用此函数，则返回数据可能被破坏。 因此强烈建议以高于采样率的频率检索数据。如果无法保证，请在调用此函数之前关闭中断。

C 语言原型：

```
INT   TriADC_iGetData3ClearFlag()
```

汇编语言：

```
lcall  TriADC_iGetData3ClearFlag
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 在 X 寄存器中返回，LSB 在累加器中返回。

[+] Feedback

Note 函数 ClearFlag、iGetData1ClearFlag、iGetData2ClearFlag 和 iGetData3ClearFlag 均清除相同的标志。 当清除转换完成标志时，它们可发挥最大限度的灵活性。 当 A/D 转换完成时，用户可选择忽略一个或两个通道的结果，还可选择在不检索数据的情况下清除标志。

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型（CY8C29xxx 和 CY8CLED16）中，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

固件源代码示例

此示例代码将启动连续转换、轮询 “ 数据可用 ”(Data Available) 标志以及将转换的字节发送至用户函数。

```
;;;  Sample Code for the TRIADC
;;;  Continuously Sample and call a user routine with the converted
;;;  data sample.
;;;
;;;  NOTE: The User Routine must complete operation within one
;;;  conversion cycle in order to retrieve the next converted sample
;;;  data.
;;;
include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main
_main:
    M8C_EnableGInt            ;Enable interrupts
    mov   a, 10               ;Set resolution to 10 Bits
    call  TRIADC_SetResolution
    mov   a, TRIADC_HIGHPOWER ;Set Power and Enable A/D
    call  TRIADC_Start
    mov   a, 00h              ;Start A/D in continuous sampling mode
    call  TRIADC_GetSamples
;A/D conversion loop
loop1:
wait:                         ;Poll until data is complete
    call  TRIADC_fIsDataAvailable
    jz    wait
    call  TRIADC_ClearFlag    ;Reset flag
    call  TRIADC_iGetData1    ;Get Data - X=MSB A=LSB
    call  User_Function       ;Call user routine to use data from
                              ;ADC Input1
    call  TRIADC_iGetData2    ;Get Data - X=MSB A=LSB
    call  User_Function       ;Call user routine to use data
                              ;ADC Input2
    call  TRIADC_iGetData3    ;Get Data - X=MSB A=LSB
    call  User_Function       ;Call user routine to use data
                              ;ADC Input3
    jmp   loop1
```


[+] Feedback

以 C 语言编写的示例项目。

```
//-------------------------------------------------------------------------
//  Sample C Code for the TriADC
//  Continuously Sample and call a user function with the data.
//  This example differs from the ASM example, in that the DataAvailable
//  flag is automatically cleared when the third value is read instead
//  of clearing the flag prior to reading the data.
//
//-------------------------------------------------------------------------


#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

extern void User_Function(int iResult1, int iResult2, int iResult3);
void main(void)
{
  int iResult1, iResult2, iResult3;
  M8C_EnableGInt;                  // Enable global interrupts
  TRIADC_Start(TRIADC_HIGHPOWER); // Turn on Analog section
  TRIADC_SetResolution(10);       // Set resolution to 10 Bits
  TRIADC_GetSamples(0);           // Start ADC to read continuously
  for(;;)
  {
    while(TRIADC_fIsDataAvailable() == 0);  // Wait for data to be ready
    iResult1 = TRIADC_iGetData1();          // Get Data from ADC Input1
    iResult2 = TRIADC_iGetData2();          // Get Data from ADC Input2
    iResult3 = TRIADC_iGetData3ClearFlag(); // Get Data from ADC Input3
                                            // and clear data ready flag
    User_Function(iResult1,iResult2,iResult3); // User function to use
                                               // data
  }
}
```

配置寄存器

这些寄存器由初始化和 API 库配置。 用户不必直接更改或读取这些寄存器。 此部分为参考内容。

ADC 是开关电容 PSoC 模块。 它已配置用于创建模拟调制器。 为构建调制器，此模块已配置为包含基准反馈的积分器，该积分器可将输入值转换为数字脉冲流。 输入复用器确定哪个信号是数字化的。

Table 9. 模块 ADC1: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 10. 模块 ADC1: 寄存器 CR1

位	7	6	5	4	3	2	1	0
值	ACMux, AMux			0	0	0	0	0

若模块置于 “A” 型模块中，则使用 ACMux。 字段值取决于用户如何连接输入。 若模块置于 “B” 型模块中，则使用 AMux。 字段值取决于用户如何连接输入。

[+] Feedback

Table 11. 模块 ADC1: 寄存器 CR2

位	7	6	5	4	3	2	1	0
值	0	1	1	0	0	0	0	0

Table 12. 模块 ADC1: 寄存器 CR3

位	7	6	5	4	3	2	1	0
值	1	1	1	FSW0	0	0	0	0

FSW0 用于 PWM 中断处理程序和各种 API。 值为 “0” 时，将使 ADC 成为禁用积分器。 值为 “1” 时，将使 ADC 成为使能积分器。

Table 13. 模块 ADC2: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 14. 模块 ADC2: 寄存器 CR1

位	7	6	5	4	3	2	1	0
值	ACMux, AMux			0	0	0	0	0

若模块置于 “A” 型模块中，则使用 ACMux。 字段值取决于用户如何连接输入。 若模块置于 “B” 型模块中，则使用 AMux。 字段值取决于用户如何连接输入。

Table 15. 模块 ADC2: 寄存器 CR2

位	7	6	5	4	3	2	1	0
值	0	1	1	0	0	0	0	0

Table 16. 模块 ADC2: 寄存器 CR3

位	7	6	5	4	3	2	1	0
值	1	1	1	FSW0	0	0	0	0

FSW0 用于 PWM 中断处理程序和各种 API。 值为 “0” 时，将使 ADC 成为禁用积分器。 值为 “1” 时，将使 ADC 成为使能积分器。

Table 17. 模块 ADC3: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 18. 模块 ADC3: 寄存器 CR1

位	7	6	5	4	3	2	1	0
值	ACMux, AMux			0	0	0	0	0

若模块置于 “A” 型模块中，则使用 ACMux。 字段值取决于用户如何连接输入。 若模块置于 “B” 型模块中，则使用 AMux。 字段值取决于用户如何连接输入。

[+] Feedback

Table 19. 模块 ADC3: 寄存器 CR2

位	7	6	5	4	3	2	1	0
值	0	1	1	0	0	0	0	0

Table 20. 模块 ADC3: 寄存器 CR3

位	7	6	5	4	3	2	1	0
值	1	1	1	FSW0	0	0	0	0

FSW0 用于 PWM 中断处理程序和各种 API。 值为 “0” 时，将使 ADC 成为禁用积分器。 值为 “1” 时，将使 ADC 成为使能积分器。

PWM16 是用于控制 ADC 集成时间的数字 PsoC 模块。 比较值设置为 2^{Bits+2}，周期设置为 CalcTime 加比较值。

Table 21. 模块 PWM16_MSB: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	比较类型	中断类型	0	0	1

“ 比较类型 ”(Compare Type) 标志指示捕获比较是 “ 等于或小于 ”(equal to or less than) 还是 “ 小于 ”(less than)。 “ 中断类型 ”(Interrupt Type) 标志指示基于捕获事件还是终端条件触发中断。 这两个参数都在器件编辑器中设置。

Table 22. 模块 PWM16_LSB: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	0	比较类型	0	0	0	1

“ 比较类型 ”(Compare Type) 标志指示比较函数是 “ 等于或小于 ”(equal to or less than) 还是 “ 小于 ”(less than)。 此参数在器件编辑器中设置。

Table 23. 模块 PWM16_MSB: 寄存器输入

位	7	6	5	4	3	2	1	0
值	0	0	1	1	时钟			

“ 时钟 ”(Clock) 从 16 个源中选择一个时钟输入。 此参数在器件编辑器中设置。

Table 24. 模块 PWM16_LSB: 寄存器输入

位	7	6	5	4	3	2	1	0
值	使能				时钟			

“ 使能 ”(Enable) 从 16 个源中选择一个数据输入，“ 时钟 ”(Clock) 从 16 个源中选择一个时钟输入。 这两个参数都在器件编辑器中设置。

[+] Feedback

Table 25. 模块 PWM16_MSB: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	输出使能	输出 Sel	

“输出使能”(Output Enable) 标志指示输出处于使能状态。“输出 Sel”(Output Sel) 标志指示 PWM16 输出将路由到的位置。这两个参数都在器件编辑器中设置。

Table 26. 模块 PWM16_LSB: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 27. 模块 PWM16_MSB: 计数寄存器 DR0

位	7	6	5	4	3	2	1	0
值	计数 (MSB)							

计数: PWM16 停止 PWM 的 MSB (最高有效位)。可使用 PWM16 API 读取此参数。

Table 28. 模块 PWM16_LSB: 计数寄存器 DR0

位	7	6	5	4	3	2	1	0
值	计数 (LSB)							

计数: PWM16 停止 PWM 的 LSB (最低有效位)。可使用 PWM16 API 读取此参数。

Table 29. 模块 PWM16_MSB: 周期寄存器 DR1

位	7	6	5	4	3	2	1	0
值	周期 (MSB)							

“周期”(Period) 保留周期值的 MSB，周期值根据使能或终端计数条件加载到计数器寄存器。可通过器件编辑器和 PWM16 API 对其进行设置。

Table 30. 模块 PWM16_LSB: 周期寄存器 DR1

位	7	6	5	4	3	2	1	0
值	周期 (LSB)							

“周期”(Period) 保留周期值的 LSB，周期值根据使能或终端计数条件加载到计数器寄存器。可通过器件编辑器和 PWM16 API 对其进行设置。

Table 31. 模块 PWM16_MSB: 脉冲宽度寄存器 DR2

位	7	6	5	4	3	2	1	0
值	脉冲宽度 (MSB)							

PulseWidth 保留用于生成比较事件的脉冲宽度值的 MSB。可通过器件编辑器和 PWM16 API 对其进行设置。

[+] Feedback

Table 32. 模块 PWM16_LSB: 脉冲宽度寄存器 DR2

位	7	6	5	4	3	2	1	0
值	脉冲宽度 (LSB)							

PulseWidth 保留用于生成比较事件的脉冲宽度值的 LSB。 可通过器件编辑器和 PWM16 API 对其进行设置。

Table 33. 模块 PWM16_MSB: 控制寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启动/停止 (0)

“ 启动 / 停止 ”(Start/Stop) 由 LSB 控制寄存器值控制，设置为 0。

Table 34. 模块 PWM16_LSB: 控制寄存器 CR0

位	7	6	5	4	3	2	1	0
值								启动 / 停止

若设置了 “ 启动 / 停止 ”(Start/Stop)，则表示 PWM16 处于使能状态。 可使用 PWM16 API 对其进行修改。

CNT 是配置为计数器的数字 PSoC 模块。 当 DR0 中的值递减至终端计数时，将调用中断以递减较高值软件计数器，且 CNT 从 DR1 重新加载。 数据通过 DR2 输出。

Table 35. 模块 CNT1: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 36. 模块 CNT1: 寄存器输入

位	7	6	5	4	3	2	1	0
值	数据				时钟			

“ 数据 ”(Data) 选择放置 ADC 模块的列比较器。 “ 时钟 ”(Clock) 从 16 个源中选择一个时钟输入，此参数在器件编辑器中设置。

Table 37. 模块 CNT1: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

[+] Feedback

Table 38. 模块 CNT1: 寄存器 DR0

位	7	6	5	4	3	2	1	0
值	计数值							

Table 39. 模块 CNT1: 寄存器 DR1

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 40. 模块 CNT1: 寄存器 DR2

位	7	6	5	4	3	2	1	0
值	数据输出							

“ 数据输出 ”(Data Out) 由 API 用于获取计数器值。

Table 41. 模块 CNT1: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	使能

若设置了 “ 使能 ”(Enable)，则 CNT 处于使能状态。 可使用 TriADC API 对其进行修改和控制

Table 42. 模块 CNT2: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 43. 模块 CNT2: 寄存器输入

位	7	6	5	4	3	2	1	0
值	数据				时钟			

“ 数据 ”(Data) 选择放置 ADC 模块的列比较器。 “ 时钟 ”(Clock) 从 16 个源中选择一个时钟输入，此参数在器件编辑器中设置。

Table 44. 模块 CNT2: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

[+] Feedback

Table 45. 模块 CNT2: 寄存器 DR0

位	7	6	5	4	3	2	1	0
值	计数值							

Table 46. 模块 CNT2: 寄存器 DR1

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 47. 模块 CNT2: 寄存器 DR2

位	7	6	5	4	3	2	1	0
值	数据输出							

“ 数据输出 ”(Data Out) 由 API 用于获取计数器值。

Table 48. 模块 CNT2: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	使能

若设置了 “ 使能 ”(Enable)，则 CNT 处于使能状态。 可使用 TriADC API 对其进行修改和控制。

Table 49. 模块 CNT3: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 50. 模块 CNT3: 寄存器输入

位	7	6	5	4	3	2	1	0
值	数据				时钟			

“ 数据 ”(Data) 选择放置 ADC 模块的列比较器。 “ 时钟 ”(Clock) 从 16 个源中选择一个时钟输入，此参数在器件编辑器中设置。

Table 51. 模块 CNT3: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

[+] Feedback

Table 52. 模块 CNT3: 寄存器 DR0

位	7	6	5	4	3	2	1	0
值	计数值							

Table 53. 模块 CNT3: 寄存器 DR1

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 54. 模块 CNT3: 寄存器 DR2

位	7	6	5	4	3	2	1	0
值	数据输出							

“ 数据输出 ”(Data Out) 由 API 用于获取计数器值。

Table 55. 模块 CNT3: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	使能

若设置了 “ 使能 ”(Enable)，则 CNT 处于使能状态。 可使用 TriADC API 对其进行修改和控制。

Table 56. 寄存器 INT_MSK1

位	7	6	5	4	3	2	1	0
值								

在此处设置对应于 PWM 模块和 CNT 模块的掩码位，以便启用各自的中断。 实际掩码值取决于各个模块的放置位置。

[+] Feedback

版本历史记录

版本	创作者	说明
2.1	DHA	添加 DRC 以检查是否： 1. 数字资源和模拟资源中的源时钟不同。 2. ADC 时钟高于 CPU 时钟。

Note PSoC Designer 5.1 在所有用户模块数据表中提供版本历史记录。 本部分记录了当前和先前用户模块版本之间区别的高级描述。

Copyright © 2001-2010 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.