Delta Sigma ADC 数据表 DelSigPlus V 1.0

资源	PSoC® 模块				API 存储器 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	抽取滤波器列	闪存	RAM	
CYC8C24x94、CY8CLED0xD、CY8CLED0xG、CY8C28x45、 CY8C28x43							
6，1st 阶，32	0	1	0	1	84	2	1
7.5，1st 阶，64	0	1	0	1	88	2	1
9，1st 阶，128	0	1	0	1	107	3	1
10.5，1st 阶，256	0	1	0	1	109	3	1
8，2nd 阶，32	0	2	0	1	99	2	1
10，2nd 阶，64	0	2	0	1	118	3	1
12，2nd 阶，128	0	2	0	1	118	3	1
14，2nd 阶，256	0	2	0	1	118	3	1

Note “ 抽取滤波器列 ” 资源只适用于 CY8C28x45 器件。

请参见应用笔记 “Analog - ADC Selection” （模拟 - ADC 选择） AN2239 获取有关其他转换器的信息。

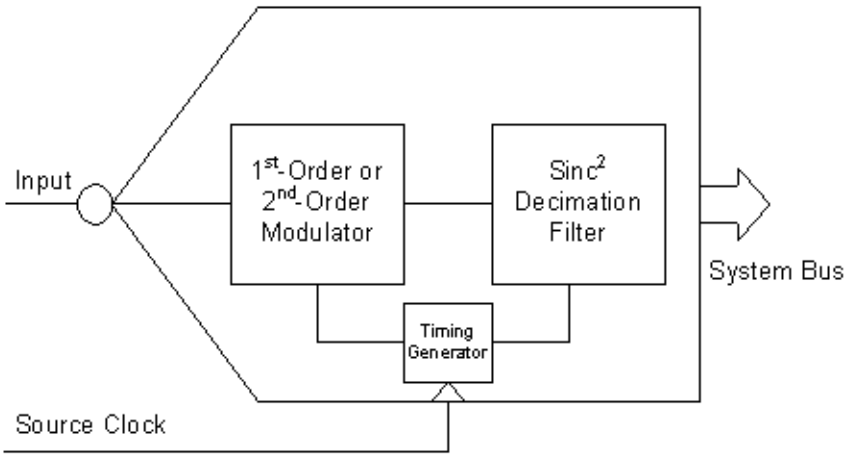
如需一个或多个使用此用户模块的完全配置的功能性示例工程，请转到 www.cypress.com/psocexampleprojects.

特性与概述

- 6-bit 到 14-bit 分辨率
- 无符号或有符号 2 的补码格式的数据
- 在 6 比特分辨率情况下最大采样率为 65,500 sps，在 14-bit 分辨率情况下最大采样率为 7812 sps
- 硬件中完全实施了 $Sinc^2$ 滤波器，降低了 CPU 开销和抗锯齿要求
- 用户可选择 1st 阶或 2nd 阶调制器以提高信噪比
- 内部和外部参考选项定义的输入范围
- 不需要数字模块

DelSigPlus 用户模块是积分转换器，需要 32 到 256 个积分周期才能生成单个输出采样。 更改复用输入会使得更改后的开始两个采样失效。 在模块放置之前请查看 “ 参数 ” 部分。

Figure 1. DelSigPlus 框图



功能说明

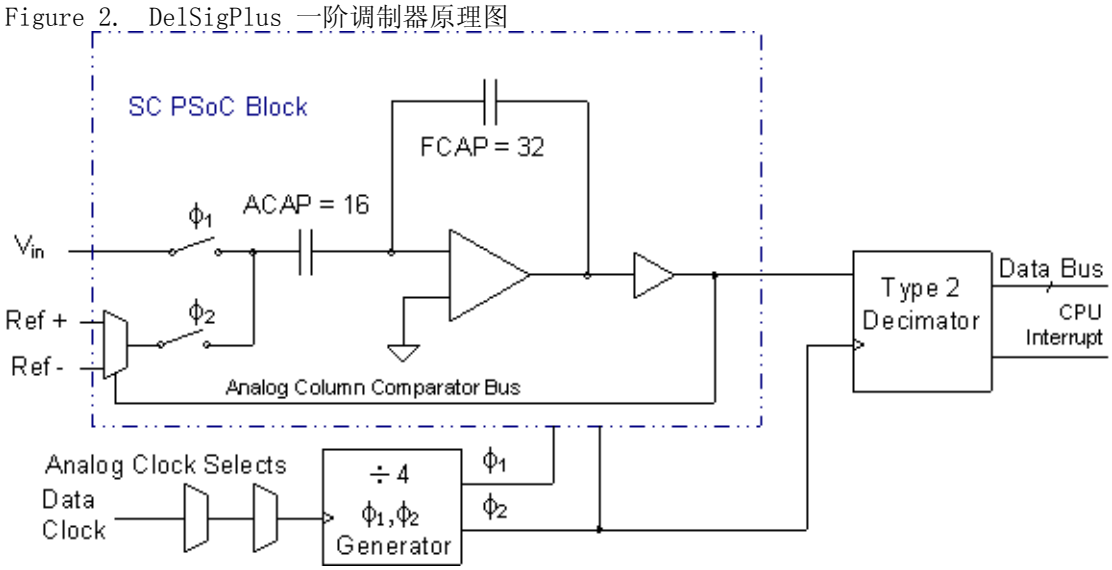
如框图所示，DelSigPlus 用户模块由三个主要功能组成：调制器、Sinc^2 抽取滤波器和定时发生器。 每个组件提供一些选项，可以调整这些选项来实现给定应用场合中性能与资源利用之间的最佳平衡。

调制器

调制器是 1-bit 过采样电路，它以所产生的密度形式 （1 和 0）表示输入电压。 通过低通抽取滤波器将多个 1-bit 样本转换为具有较高分辨率的样本，调制器输出降低到最终的采样率。 通常，抽取速率越高（即过采样率越高），则分辨率结果越高，但是其他因素 （例如调制器的阶）也会影响分辨率结果。

Delta-Sigma 转换器的主要优点是调制器可提供 “ 噪声整形 ”。 通常，信号采样中固有的量化噪声是一种大致均匀分布的噪声 （白噪声），其频率介于 “DC” 与采样频率一半 （即奈奎斯特频率）之间。 简单而言，delta-sigma 调制器将某些量化噪声从较低频率转换为较高频率，之后会由抽取滤波器进行衰减。 二阶调制器需要两个开关电容模拟 PSoC 模块，它对噪声整形的效果要好于仅需要一个模拟 PSoC 模块的一阶调制器。 由于最高抽取速率为 256X，因此与一阶调制器相比，二阶调制器将有效分辨率提高了 3.5-bit。

Figure 2. DelSigPlus 一阶调制器原理图



模拟模块配置为积分器。 电压比较器的输出极性对参考复用器进行配置，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到零。 一位电压比较器输出也会馈送到 $sinc^2$ 抽取滤波器中。

请注意，1-bit 过采样率由四分发生器确定，四分发生器生成控制开关电容 (SC) PSoC 模块的 Φ1 和 Φ2 时钟。 输出速率的确定方法是：将数据除以 4 以获得 1-bit 过采样率，接着再除以抽取速率以获得最终采样率：

Equation 1

$$SampleRate = \frac{DataClockFrequency}{4 \times DecimationRate} \text{ samples per second}$$

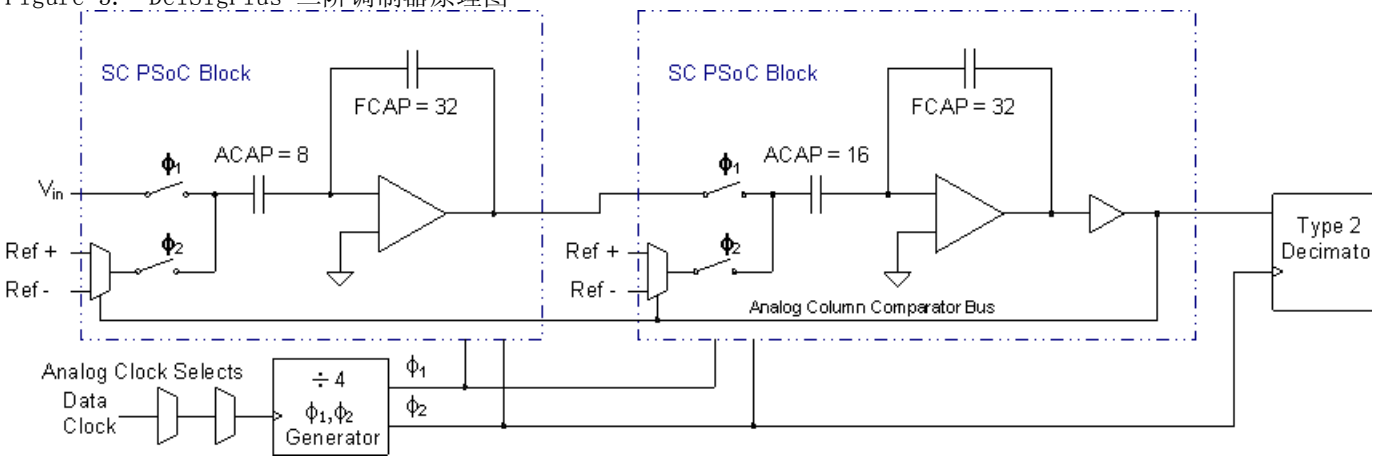
下面的规格表中给出了可以使用的最高数据时钟频率。 对于 8MHz 的数据时钟和 256 的抽取速率，采样率是：

Equation 2

$$8 \times 10^6 / (4 \times 256) = 7812.5 sps$$

二阶调制器的构造方法是：将一阶调制器的模拟输出馈送到类似的 PSoC 模块中，并修改反馈排列，以便第二个模块的 1-bit 电压比较器输出反馈到这两个模块，如下图所示：

Figure 3. DelSigPlus 二阶调制器原理图



由于模拟电压比较器总线在模拟 PSoC 模块阵列的列中垂直运行，二阶调制器的模块必须一个摞一个地放置。

DelSigPlus 的范围是通过 $\pm V_{Ref}$ 建立的。 您可以在 PSoC Designer “ 全局资源 ” 窗口中设置 V_{Ref}。对于固定量程，V_{Ref} 设置为 $\pm V_{Bandgap}$，对于 CY8C29x66 系列 PSoC 器件，设置为 $\pm 1.6\ V_{Bandgap}$。 对于可调整量程，V_{Ref} 设置为 ±Port 2[6]。 要提供比率计量程，则 V_{Ref} 设置为 $\pm V_{DD}/2$。 下表给出了完整选项列表。

Table 1. 针对 Ref Mux 全局参数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]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

Sinc^2 抽取滤波器

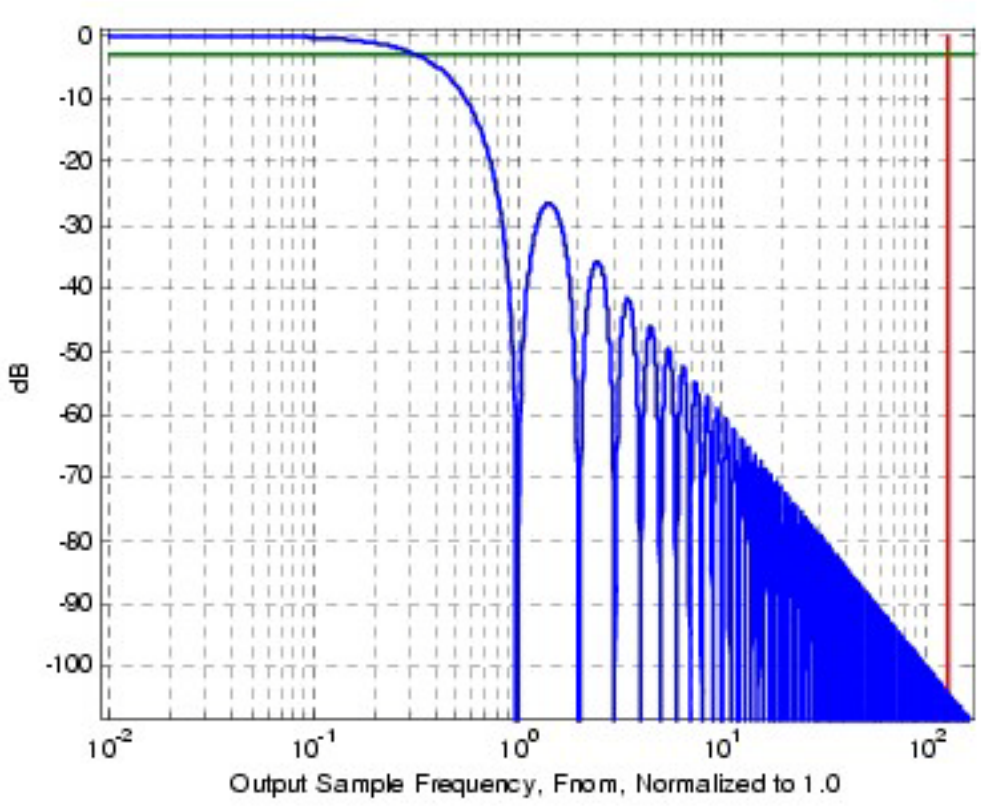
抽取滤波器的响应由下列 z 域关系提供：

Equation 3

$$H(z) = \left[\frac{1 - z^{-n}}{1 - z^{-1}}\right]^2, \text{ where n is the decimation level.}$$

下面绘制的频率域传输函数将频率标准化，以使输出采样率 F_{nom} 等于 1.0。 -3 dB 点出现在紧靠 $0.318 \times F_{nom}$ 上方，函数的零点出现在 F_{nom} 的每个整数倍处。 由于 1-bit 采样率比额定输出速率高 32 到 256，奈奎斯特限制为比 F_{nom} 高 4 到 7 个八度，因而极大降低了对防锯齿滤波器的要求。 在图形右侧，用粗垂直线显示了抽取速率为 256 的 1-bit 奈奎斯特频率。 虽然实现较高抽取速率是可能的，但是由于器件本底噪声的存在，它们带来的好处有限。 对于 14-bit 拓扑、抽取速率为 256 的二阶调制器，分辨率受信噪比限制。 要在 DC 或慢速移动信号测量中获取可重复的 14-bit 分辨率，需要对多个输出采样值求平均值，或者应用更复杂的信号处理技术。

Figure 4. Sinc^2 抽取滤波器幅度响应，包含 -3dB 点和奈奎斯特频率



与早先的 DELSIG8 和 DELSIG11 不同，此用户模块在硬件中完整实施传输函数的分子和分母。 这需要改进的 “ 类型 2” 抽取滤波器。 该滤波器既可用于一阶调制器拓扑，也可用于二阶调制器拓扑。 抽取滤波器通过在 1-bit 采样率下运行的双积分器实现传输函数的分母。 分子由在额定输出采样率下运行的双微分器 （第二差分运算符）实现。 DelSigPlus 用户模块使用的 CPU 开销和中断延迟限制为大约 80 周期或更少，以便从 I/O 空间中的抽取滤波器寄存器检索采样数据。 类型 2 抽取滤波器实际上是为 n 位转换器生成从 0 到 2^n-1 的无符号值。 中断服务子程序可以配置为将此转换为从 -2^{n-1} 到 $+2^{n-1}-1$ 的 2 的补码值。

Table 2. Delta Sigma ADC 特性表

特性	Delta Sigma ADC		
	DELSIG8, DELSIG11	DelSig	DelSigPlus
分辨率	8，11	6-14	6-14
数字模块	1	1-2	0
模拟模块	1-2	1-2	1-2
支持的部件	CY8C24/27/29，而非 CY8C24x94	CY8C24x94, CY8C29xxx	CY8C24x94
CPU 开销和中断延迟	高	低	低

定时发生器和要求

向模拟调制器提供 Φ1 和 Φ2 时钟的四分时钟发生器也会向抽取滤波器提供位时钟。 对应于输出采样率的抽取因子由字时钟确定。 字时钟由抽取滤波器内部定时器生成。

类型 2 抽取滤波器是 Sinc2 滤波器的全硬件版本。 其体系结构使您可以选择将内部定时器用于抽取和中断目的。 要计算有效分辨率，请使用下列等式：

单一调制器：(log2(DecimatorRate) - 1)*1,5

双调制器：(log2(DecimatorRate) - 1)*2

DataFormat 位可以加权为有符号 （2s 补码输出）或无符号 （偏移二进制数据）数值。

Table 3. 抽取滤波器数据输出移位

抽取速率	调制器类型	有效分辨率	移位
32	单	6	4
32	双	8	2
64	单	8 (7.5)	4
64	双	10	2
128	单	9	5
128	双	12	2
256	单	11(10.5)	5
256	双	14	2

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 = -40、25、85 和 125° C、V_{dd} = 5.0V。

Table 4. 5.0V 结果汇总

参数	典型值	极限值	单位	备注
8 位、24MHz CPU 时钟、1 MHz 数据时钟、高功率				
增益	-2.6482	2	%FSR	
偏移	-47.0072	13	mV	
微分非线性误差	0.161	<1	LSB	
积分非线性误差	0.27	--	LSB	
信噪比	45.86	--	dB	
8 位、24MHz CPU 时钟、2 MHz 数据时钟、高功率				
增益	-2.3168	2	%FSR	
偏移	-62.3507	13	mV	
微分非线性误差	0.069	<1	LSB	
积分非线性误差	0.172	--	LSB	
信噪比	45.86	--	dB	

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 = -40、25、85 和 125° C、V_{dd} = 3.3V。

Table 5. 3.3V 结果汇总

参数	典型值	极限值	单位	备注
8 位、24MHz CPU 时钟、1 MHz 数据时钟、高功率				
增益	-2.7182	2	%FSR	
偏移	-40.1334	5	mV	
微分非线性误差		<1	LSB	
积分非线性误差		--	LSB	
信噪比		--	dB	
8 位、24MHz CPU 时钟、2 MHz 数据时钟、高功率				
增益	-2.8219	2	%FSR	
偏移	-42.8073	5	mV	
微分非线性误差	0.064	<1	LSB	
积分非线性误差	0.161	--	LSB	
信噪比	46.02	--	dB	

放置

当 DelSigPlus 用户模块是在工具栏中或者是通过在选择器视图中双击其图标选择时，将打开一个提供选择适当拓扑指南的选择窗口。 稍后要更改拓扑，可以通过右键单击位置视图中的用户模块并 从上下文菜单选择 “ 用户模块选择选项 ...”。

一阶调制器设计需要一个 PSoC 模拟模块。 名为 “ADC” 的模拟模块可以放置在任何开关电容 PSoC 模块中。

二阶调制器设计使用两个开关电容 PSoC 模块：ADC1 和 ADC2。 由于连接它们的模拟电压比较器总线在模拟阵列的每列中垂直运行，开关电容 PSoC 模块必须一个摞一个地垂直放置。

虽然有大量可用于模拟模块的位置，但是 DelSigPlus 还使用 PSoC 器件的唯一硬件抽取滤波器。 抽取滤波器是在放置模拟模块时自动分配的；不需要附加操作。 因此，给定配置中只能放置 DelSigPlus 用户模块的一个实例。 通过动态重新配置，可以同时加载 （激活）多个配置，不执行阻止两个 DelSigPlus 用户模块同时运行的检查。 如果发生这种情况，则两个实例似乎都可以运作；但是，只有一个最近加载的实例会控制抽取滤波器。 两个中断仍可以运行，但是可能会有干扰。

参数和资源

一旦放置 DelSigPlus 实例后，必须配置下列参数才能正确操作：输入信号复用器选择 (Input Signal Multiplexer selection)、时钟相位 (Clock Phase) 和轮询选择 (Polling selection)

DataFormat

此参数可以采用无符号 （默认值）或有符号值。 对于 n 位分辨率，无符号数据会采用从零到 2^n-1 的数值。 有符号数据的值会在 -2^{n-1} 到 $+2^{n-1}-1$ 范围之内

时钟相位

“ 时钟相位 ”(Clock Phase) 的选择用于将一个模拟 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 （Φ1, Φ2) 获取并传输信号。 通常，DelSigPlus 的输入取样于 Φ1。 这便产生一个问题：许多用户模块在 Φ1 期间将其输出自动归零，仅在 Φ2 期间给出有效输出。 如果此类模块的输出馈送到 DelSigPlus 的输入，则 DelSigPlus 会对中间值采样。 时钟相位选择允许交换相位，因此输入信号是在 Φ2 期间获取的。

PosInput

此参数确定单端输入的信号源，或差分输入的同相输入。

NegInput 和 NegInputGain

NegInput 选择差分信号对的反相输入源。 如果使用单端输入，则可以将此参数设置为任意合法值。 通过将 NegInputGain 参数设置为 “ 断开连接 ” （零增益），可以断开其与转换器的连接。

NegInputGain 调整相对于同相输入的反相输入的增益 （请参见上述 NegInput 参数）。 对于单端输入，此参数应当采用 “ 断开连接 ” 值。 对于差分输入，NegInputGain 可以设置为 1.000。 如果需要，相对于同相输入，应用于反相输入的增益还可以在 0.0625 与 1.9375 之间按 1/16th 增量进行调整。

中断生成控制

当选中 PSoC Designer 中的 **“ 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：

- Interrupt API
- IntDispatchMode

InterruptAPI

使用 InterruptAPI 参数，可以有条件地生成用户模块中断处理程序和中断矢量表条目。 选择 “ 使能 ”(Enable) 将生成中断处理程序和中断矢量表条目。 选择 “ 禁用 ”(Disable) 将不生成中断处理程序和中断矢量表条目。

如果项目有多个程序层，由不同的程序层使用同一个模块资源，则选择时一定要特别注意。 选择以仅为实际需要它们的程序层生成中断，以节省代码空间。

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。

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 DelSigPlus_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。 为简单起见，在下面的说明中将实例名称缩短为 DelSigPlus。

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。

DelSigPlus_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

C 语言原型：

```
void  DelSigPlus_Start (BYTE bPowerSetting)
```

汇编语言：

```
mov   A, bPowerSetting
lcall  DelSigPlus_Start
```

参数：

bPowerSetting: 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 DelSigPlus 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
DelSigPlus_OFF	0
DelSigPlus_LOWPOWER	1
DelSigPlus_MEDPOWER	2
DelSigPlus_HIGHPOWER	3

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。

DelSigPlus_Stop

说明：

将开关电容 PSoC 模块的功耗水平设置为关闭。

C 语言原型：

```
void  DelSigPlus_Stop (void)
```

汇编语言：

```
lcall  DelSigPlus_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。

DelSigPlus_SetPower

说明：

设置开关电容 PSoC 模块的功耗水平。

C 语言原型：

```
void  DelSigPlus_SetPower(BYTE bPowerSetting)
```

汇编语言：

```
mov   A, bPowerSetting
lcall  DelSigPlus_SetPower
```

参数：

bPowerSetting: bPowerSetting: 用于指定功耗水平的 1 个字节。 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
DelSigPlus_OFF	0
DelSigPlus_LOWPOWER	1
DelSigPlus_MEDPOWER	2
DelSigPlus_HIGHPOWER	3

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。

DelSigPlus_StartAD

说明：

激活此用户模块的中断，开始采样。

C 语言原型：

```
void  DelSigPlus_StartAD (void)
```

汇编语言

```
lcall  DelSigPlus_StartAD
```

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。

DelSigPlus_StopAD

说明：

通过中断禁用关闭 A/D。 模拟电源仍提供给模拟模块。

C 语言原型：

```
void  DelSigPlus_StopAD(void)
```

汇编语言：

```
lcall  DelSigPlus_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。

DelSigPlus_fIsDataAvailable

说明：

检查采样数据的可用性。

C 语言原型：

```
BYTE  DelSigPlus_fIsDataAvailable (void)
```

汇编语言：

```
lcall  DelSigPlus_fIsDataAvailable
cmp   A, 0
jz    .DataNotAvailable
```

参数：

无

返回值：

如果已经转换好数据并已准备好读取，则返回一个非零数值。

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。

DelSigPlus_cGetData

DelSigPlus_iGetData

说明：

以有符号的 8-bit 或 16-bit 2 的补码格式返回转换的数据。 请注意，用户模块 DataFormat 参数决定了基本表示。 当基本表示是无符号数据时，调用有符号格式函数不会更改数据值。可以调用 DelSigPlus_fIsDataAvailable() 以验证数据采样已就绪。

C 原型：

```
CHAR DelSigPlus_cGetData (void)      // use for 8-bit resolution or lower
INT  DelSigPlus_iGetData (void)      // use for 9-bit resolution or higher
```

汇编语言：

```
lcall  DelSigPlus_iGetData      ; Result will be in A
- or -
lcall  DelSigPlus_iGetData      ; LSB will be in A, MSB in X upon return
```

参数：

无

返回值：

以 8-bit 或 16-bit 2 的补码格式返回转换的数据采样。

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。

DelSigPlus_bGetData

DelSigPlus_wGetData

说明：

以 8-bit 或 16-bit 无符号格式返回转换的数据。 请注意，用户模块 DataFormat 参数决定了基本表示。 当基本表示是有符号数据时，调用无符号格式函数不会更改数据值。可以调用 DelSigPlus_fIsDataAvailable() 以验证数据采样已就绪。

C 原型：

```
BYTE DelSigPlus_bGetData(void)      // use for 8-bit resolution or lower
WORD DelSigPlus_wGetData(void)      // use for 9-bit resolution or higher
```

汇编语言：

```
lcall  DelSigPlus_bGetData       ; Result will be in A
- or -
lcall  DelSigPlus_wGetData       ; LSB will be in A, MSB in X upon return
```

参数：

无

返回值：

根据该函数，以 8-bit 或 16-bit 无符号格式返回转换的数据采样。

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。

DelSigPlus_ClearFlag

说明：

复位 “ 数据可用 ”(Data Available) 标志。

C 语言原型：

```
void  DelSigPlus_ClearFlag(void)
```

汇编语言：

```
lcall  DelSigPlus_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。

DelSigPlus_cGetDataClearFlag

DelSigPlus_iGetDataClearFlag

说明：

返回转换的数据 8-bit 或 16-bit 的补码格式，并复位 “ 数据可用 ”(Data Available) 标志。 请注意，用户模块 DataFormat 参数决定了基本表示。 当基本表示是有符号数据时，调用无符号格式函数不会更改数据值。可以调用 DelSigPlus_fIsDataAvailable() 以验证数据采样已就绪。

C 语言原型：

```
CHAR DelSigPlus_cGetDataClearFlag(void) //for 8-bit resolution or lower
INT  DelSigPlus_iGetDataClearFlag(void) //for 9-bit resolution or higher
```

汇编语言：

```
lcall  DelSigPlus_cGetDataClearFlag  ;Result will be in A
- or -
lcall  DelSigPlus_iGetDataClearFlag  ;LSB will be in A, MSB in X upon return
```

参数：

无

返回值：

以 8-bit 或 16-bit 2 的补码格式返回转换的数据采样。

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。

DelSigPlus_bGetDataClearFlag

DelSigPlus_wGetDataClearFlag

说明：

以 8-bit 或 16-bit 无符号格式返回转换的数据，并复位 “ 数据可用 ”(Data Available) 标志。 请注意，用户模块 DataFormat 参数决定了基本表示。 当基本表示是有符号数据时，调用无符号格式函数不会更改数据值。可以调用 DelSigPlus_fIsDataAvailable() 以验证数据采样已就绪。

C 语言原型：

```
BYTE DelSigPlus_bGetDataClearFlag(void)  //for 8-bit resolution or lower
WORD DelSigPlus_wGetDataClearFlag(void)  //for 9-bit resolution or higher
```

汇编语言：

```
lcall  DelSigPlus_bGetDataClearFlag  ;Result will be in A
- or -
lcall  DelSigPlus_wGetDataClearFlag  ;LSB will be in A, MSB in X upon return
```

参数：

无

返回值：

以 8-bit 或 16-bit 无符号格式返回转换的数据采样。

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

此示例重复上一情形，但是使用 API 函数而不是直接引用全局变量。

下面是汇编语言示例：

```
include "DelSigPlus.inc"
include "m8c.inc"

export _main

_main:
   M8C_EnableGInt                   ; enable global interrupts
   mov   A,DelSigPlus_HIGHPOWER        ; Establish power setting...
   call  DelSigPlus_Start              ; and initialize
   call  DelSigPlus_StartAD            ; Commence sampling process
mainloop:
   call  DelSigPlus_fIsDataAvailable   ; Retrieve the status byte
   cmp   A, 0
   jz    mainloop                   ; spin lock until(data is Available)
   call  DelSigPlus_iGetDataClearFlag   ; fastcall convention puts data in X, A
   call  ProcessSample             ; pass the sample to the dummy fcn
   jmp   mainloop


ProcessSample:
   ...                              ; (do something useful with the data)
   ret
```

C 语言中的等效代码：

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

void ProcessSample( int iSample )
{
    ; // (Do something useful with the data)
}

void main(void)
{
   M8C_EnableGInt;
   DelSigPlus_Start( DelSigPlus_HIGHPOWER );
   DelSigPlus_StartAD();
   while (1) {
      if ( DelSigPlus_fIsDataAvailable() ) {
         ProcessSample( DelSigPlus_iGetDataClearFlag() );
      }
   }
}
```

配置寄存器

模拟寄存器、1st 阶调制器

Table 6. “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	0	0	1	0	0	0	0
CR1	PosInput			InvertingGain				
CR2	0	1	0	0	0	0	0	0
CR3	1	1	0	0	NegInput		电源	

PosInput 选择单端输入信号或差分输入信号的同相输入。 NegInput 选择差分输入的反相输入。 只要 InvertingGain 字段设置为零，反相输入就会断开连接。 电源是通过 DelSigPlus_Start 和 DelSigPlus_SetPower API 函数设置的。

模拟寄存器、2nd 阶调制器

Table 7. “ADC1” 和 “ADC2” 模拟开关电容 PSoC 模块所使用的寄存器

寄存器	7	6	5	4	3	2	1	0
ADC1CR0	1	0	0	0	1	0	0	0
ADC1CR1	PosInput			InvertingGain				
ADC1CR2	0	1	0	0	0	0	0	0
ADC1CR3	1	1	0	0	NegInput		电源	
ADC2CR0	1	0	0	1	0	0	0	0
ADC2CR1	LinkToADC1			0	0	0	0	0
ADC2CR2	0	1	0	0	0	0	0	0
ADC2CR3	1	1	0	0	0	0	电源	

PosInput 选择单端输入信号或差分输入信号的同相输入。 NegInput 选择差分输入的反相输入。 只要 InvertingGain 字段设置为零，反相输入就会断开连接。 LinktoADC1 由模块放置确定，它将 ADC1 模块的输出连接到 ADC2 PSoC 模块的 “A” 输入电容。 电源是通过 DelSigPlus_Start 和 DelSigPlus_SetPower API 函数设置的。

抽取滤波器控制寄存器

Table 8. 抽取控制寄存器

位	7	6	5	4	3	2	1	0
DEC_CR0	0	0	0	0	0	DCol		DCLKSEL

位	7	6	5	4	3	2	1	0
DEC_CR1	0	1	0	0	0	DCLKSEL		
DEC_CR2	1	0	移位		1	DecimationRate		
DEC_DH	抽取滤波器高字节输出							
DEC_DL	抽取滤波器低字节输出							

抽取滤波器是用于实现 Sinc2 滤波器的专用硬件。 它由三个控制寄存器和两个数据输出寄存器组成。 DCol 选择将连接的列比较器。 DCLKSEL 选择用于控制抽取滤波器定时的数字时钟。 这两个参数都在设备编辑器中设置。 DEC_CR2 中的移位是根据抽取速率设置的，在 DEC_CR2 中也进行了指定，其目的是最大程度地减少软件中必须完成的对齐数据。

版本历史记录

版本	创作者	说明
1.0	DHA	初始版本

Note PSoC Designer 5.1 介绍了所有用户模块数据移位中的版本历史，以记录当前用户模块版本和以前用户模块版本之间区别的高级描述。