CapSense® Sigma-Delta 调制和 ADC 数据手册 CSDADC V 1.40

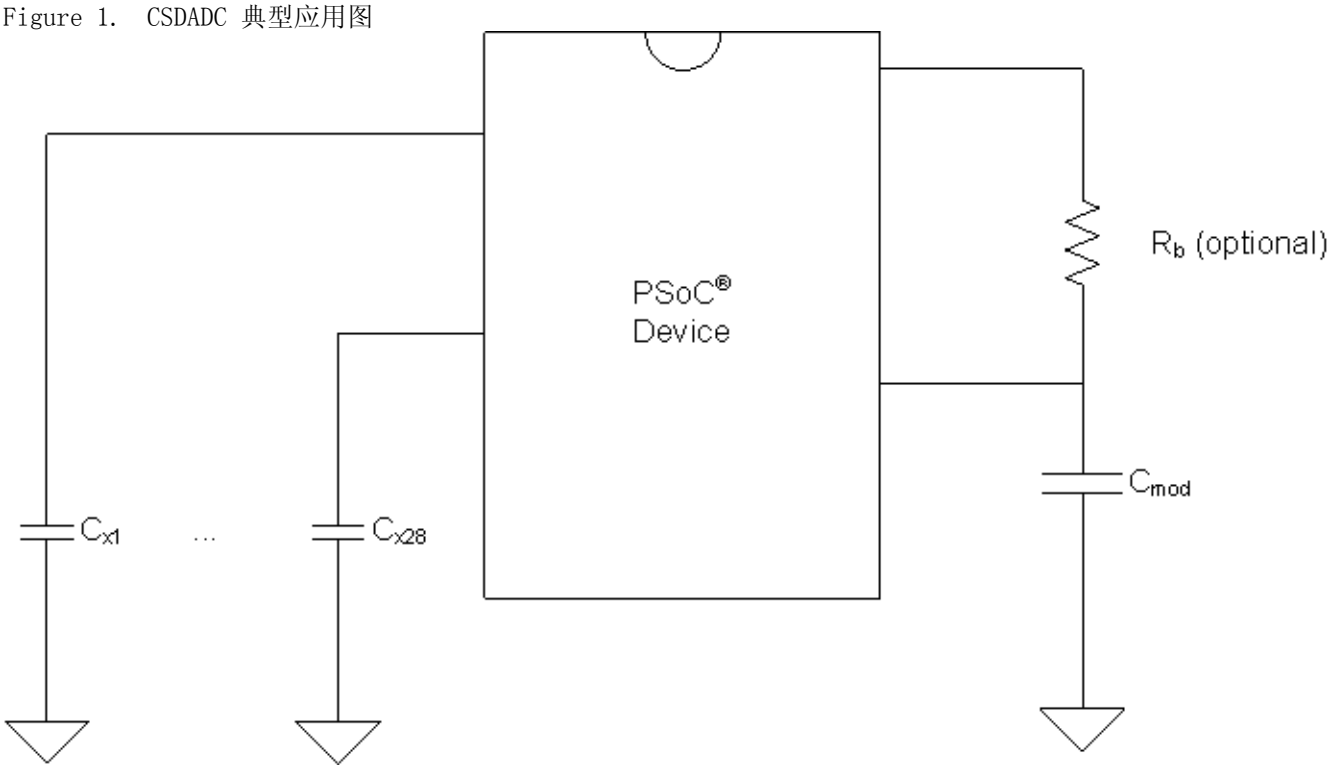
Copyright © 2011-2012 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节） 典型值		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C21x34, CY8CLED04. 闪存、RAM 和引脚的使用因传感器数量和配置而异。						
带有 1 个传感器、基于 PRS16 的用户模块	3	2	1	1491	36	2-5
带有 1 个传感器、基于 PRS8 的用户模块	2	2	1	1401	36	2-5
带有 1 个传感器、基于预分频器的用户模块	2	2	1	1426	36	2-5
有 1 个传感器的 VC2 时钟源用户模块	1	2	1	1351	36	2-5
每个附加 CapSense® 按钮	-	-	-	10	2	1
使用带有五个元件的电容式滑条时，静态代码和 RAM 增加	-	-	-	79	583	5
每个附加滑条元件	-	-	-	10	2	1
当使用滑条复用时，静态代码和 RAM 增加	-	-	-	滑块 *2	0	-
Die 温度测量	-	-	-	380	2	-

功能和概述

- 允许扫描 CapSense 传感器与测量输入电压，无需单独使用可加载配置
- ADC 特性：
  - 绝对和比例 ADC 输入模式并能在运行时进行模式转换，可让您轻松扫描不同的传感器类型
  - 用于绝对电压输入模式的单斜 ADC
  - 单斜 ADC 的内置校准机制
  - 用于比例输入模式的积分增量型 ADC
  - 允许粗略温度测量：范围为 -40° C 至 +125° C，± 40° C 精度，± 2° C 分辨率
- CapSense® 特性：
  - 基于强大的 CSD 方法
  - 扫描 1 到 28 个电容式传感器
  - 可感应高达 15-mm 厚的玻璃外覆层
  - 使用线缆传感器时，接近检测范围可达 20 cm
  - 对交流电源噪声、EMC 噪声和电源电压变动有较强的抗干扰能力
  - 支持独立按键和滑条电容式传感器的不同组合
  - 可使用复用法将滑条传感器的物理分辨率翻倍
  - 可使用插值方法增加滑条传感器的分辨率
  - 可通过两个滑条传感器提供触摸板支持
  - 感应能力通过高阻抗性传导材质 （例如 ITO 薄膜）实现
  - 具有屏蔽电极支持，可在存在水膜或水滴的情况下可靠工作
  - 可使用 CSDADC 向导实现传感器和引脚分配
  - 集成了用于处理温度、湿度和静电释放 (ESD) 事件的基准线更新算法
  - 具有可轻松调整的运行参数
  - 具有 PC GUI 应用程序支持，用于实时原始数据监控和参数优化

Figure 1. CSDADC 典型应用图



快速启动

1. 如果使用，选择并放置需要专用引脚 （例如 I2C 和 LCD）的用户模块。 根据需要分配端口和引脚。
2. 选择并放置 CSDADC 用户模块。
3. 右键单击 CSDADC 用户模块以访问 CSDADC 向导。
4. 设置 CapSense 传感器个数、配置以及引脚分配。
5. 设置引脚和全局参数。 阅读所有参数说明，遵守各种要求和相关指南。
6. 生成应用，并切换到应用编辑器。
7. 根据需要调整采样代码，以部署独立传感器、滑条传感器或触摸板。
8. 将 RS232 级转换器或 USB-I^{2}C 桥接器连接到目标电路板，并使用 GUI 优化这些参数。
9. 更改 CSDADC 参数并重建应用。
10. 对 PSoC 器件编程并验证模块操作。 按照 中的讨论，调整 CSDADC 参数达到 5:1 信噪比要求 CY8C21x34/B CapSense 设计指南 .

功能描述

CSDADC 将电容式感应与 ADC 功能结合来测量电压，无需单独使用可加载配置。 它通过重复使用 CSD 和 ADC 的共同模块来节省代码空间。 当您同时需要电容式感应与 ADC 功能时，您应使用 CSDADC。 只用其中一个或另一个功能的应用则应使用 CSD 用户模块，或使用 ADC8 或 ADC10。

CSDADC 使用开关电容技术以和 sigma-delta 调制器 (CSD) 来实现电容检测功能，该调制器可以把开关电容的电流值转换成数字量。 由于 CSDADC 可以同时使用绝对和比例输入 ADC 模式，并且可以在运行时转换模式，因此其可以轻松与各类传感器交互。 例如，使用比例模式利用热敏电阻测量温度，并使用绝对电压测量来测量电池电压。

CSDADC 用户模块在绝对输入电压模式下使用单斜 ADC 来实现，在比例模式下通过增量型 ADC 实现。

本用户模块手册提供有关 CSD 和 ADC 的基本信息。 有关参数设置、使用提示和故障排除的详细理论与建议，请参阅位于以下位置的 CSD 和 ADC10 数据手册及相关应用笔记： www.cypress.com. 如果您是首次实施电容式感应应用或使用 ADC，您应在开始操作前阅读这些文档。

请在首次使用 CSDADC 用户模块前阅读下列文档：

- 《*CY8C21x34Series PSoC Mixed Signal Array Technical Reference Manual*》，应重点阅读以下章节：
  - 两列限制模拟
  - 数字时钟
  - IO 模拟复用器
- CSD 用户模块数据手册
- ADC10 用户模块数据手册

建议在阅读 CSD 用户模块数据表后阅读以下设计指南。 这些文件在赛普拉斯网站上提供 www.cypress.com:

- CapSense 入门
- CY8C20xx6A/H CapSense 设计指南
- CY8C21x34/B CapSense 设计指南
- CY8C20x34CapSense 设计指南
- CY8CMBR2044 CapSense® 设计指南

ADC 操作说明

CSDADC 支持两类 ADC：

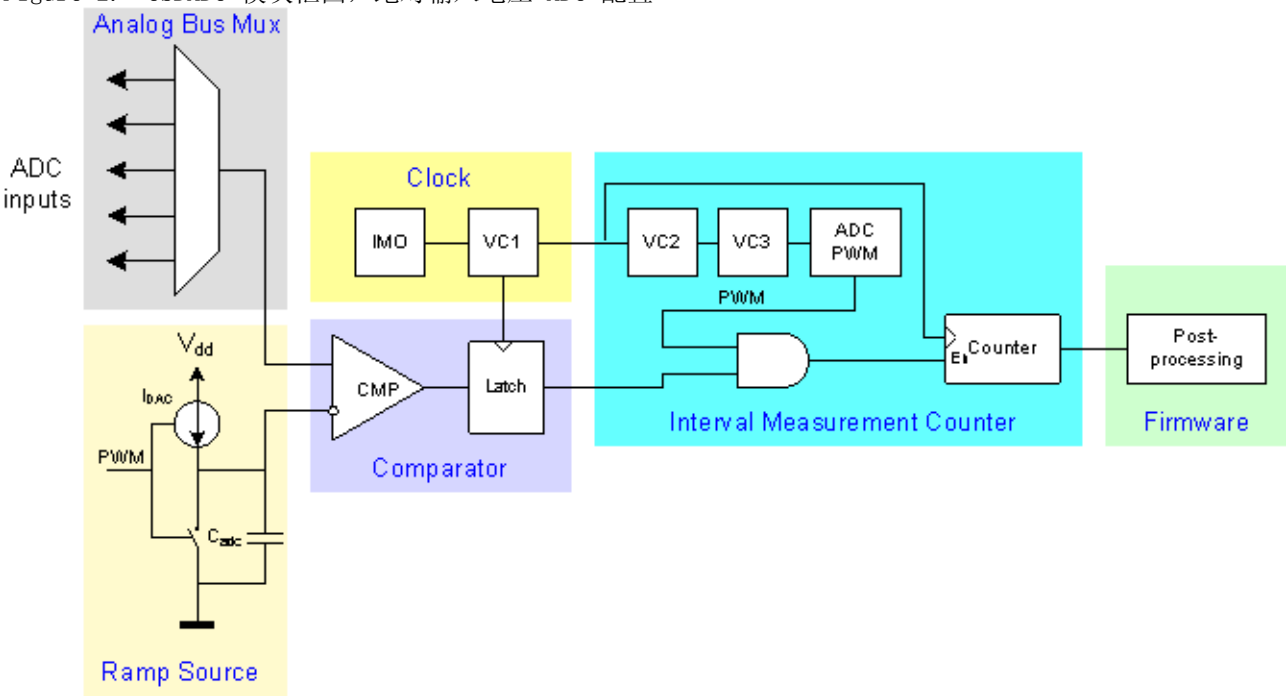
- 单斜。 单斜 ADC 支持绝对输入电压模式，此模式下 ADC 读取与输入信号成正比，不依赖于 PSoC 电源电压。
- 增量型。 增量型 ADC 支持比例操作模式，此模式下 ADC 读取与输入电压成正比，与 PSoC 电源电压成反比。

在 ADC 采样机制方面二者之间存有差异。 单斜 ADC 在采样时即会转换信号值，比例 ADC 因其积分性质在整个转换间隔期间转换信号。 所以，如果输入信号嘈杂，应该使用增量型 ADC。

单斜 ADC 操作

单斜 ADC 的配置与 ADC10 用户模块相同。

Figure 2. CSDADC 模块框图，绝对输入电压 ADC 配置



单斜 A/D 转换器可生成高达 12-bit 的全量程输出 （数量范围为 0 到 4095）。 采样率取决于您的时钟源配置和选定的参数。

ADC 由以下 PSoC 资源构成：

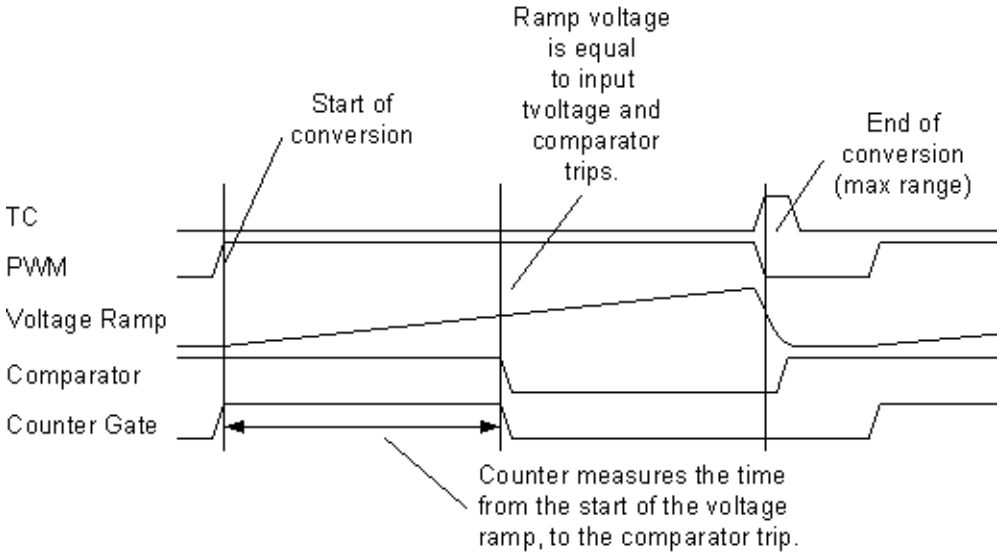
- 配置为比较器的模拟 CT 模块。
- 配置为使用内部电容和电流源的斜坡发生器的模拟 SC 模块。
- 配置为 8 位计数器的数字模块。
- 专门用于 ADC 时序的 PWM。

图 2 中显示的是单斜 ADC 实现的模块框图。 转换算法的核心包括电流源、积分电容和比较器。 电流源启动后，电容中将生成线性电压斜坡。 电容电压是模拟比较器电路的一个输入，另一个是要进行转换的模拟输入电压。 在斜坡电压超过输入电压之前，比较器一直处于高电平，之后将切换为低电平。 计数器会追踪记录斜坡开始到比较器被激发之间的时间。

基本转换波形如图 3 所示。 在 12-bit 采样的情况下，将对 PWM 的高电平时间进行设定，以使计数器达到 4096。 PWM 的低电平时间设计为允许电容放电并处理相关应答。 PWM 的终止计数用作读取转换结果的

中断。 如果斜坡电压从未超过输入值，则将 ADC_CR 的第 7 位设置为高。 PWM 发生器仅从 VC3 计时，且对高电平和低电平时间只能进行有限选择。

Figure 3. 单斜 ADC 转换时序图



ADC 输入是比较器输入的采样保持。 采样和保持电路由 PWM 控制。 这可以保证发生转换时信号保持不变。

单斜 ADC 的校准

在使用 ADC 前您必须将其校准。 为此，每个模拟列都具备一个 ADC 电容调整寄存器。 此寄存器对决定 ADC 电压斜坡斜率的电容值加以控制。 ADC_TR 寄存器的 CAPVAL [7:0] 位用来进行校准。 此校准过程的目标是对升降时间 （斜坡）进行调整，从而使全量程 ADC 输入值生成全量程 ADC 代码。 通过将升降时间与所需全量程转换周期进行匹配来实现此目标。 如果 PSoC 器件采用的电源稳压不良，则该校准过程应该定期运行，这是因为 DAC 电流受电源电压影响。

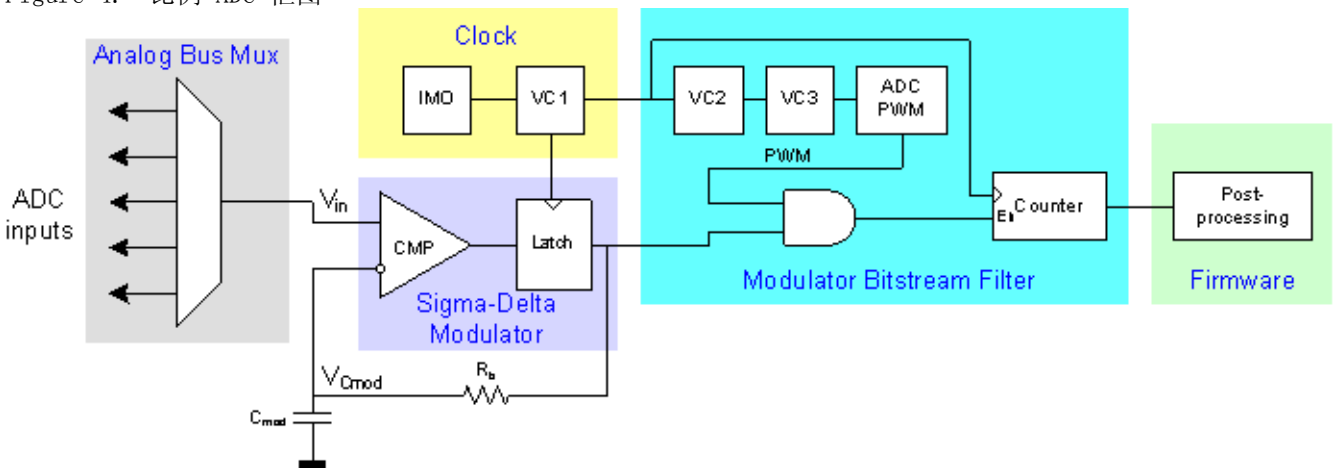
用户模块 API 包括一个称为 CSDADC_iCal 的例程，该例程执行的校准算法会基于校准电压和校准计数值对 ADC_TR 寄存器进行调整。

CSDADC_iCal 例程的返回值是在校准信号上运行 ADC 的结果。 理想情况下，该值应该与预期的校准计数值相同。 如果不同，可以通过软件方法来保证其精确性。

增量型 ADC

增量型 ADC 操作基于一阶 sigma-delta 调制器和基于计数器的调制器位流数字滤波器的组合。 图 4 显示了此 ADC 类型的模块框图。

Figure 4. 比例 ADC 框图



此类型 ADC 由以下部分组成：

- Sigma-delta 调制器
- 调制器位流滤波器
- 时钟源

此调制器由比较器、比较器锁存器、调制电容 (C_{mod}) 和反馈电阻 (R_b) 构成。 如果调制电容电压低于输入电压 V_{in}，则比较器输出为高 V_{dd} 电平，并为调制器电容充电。 当调制器电容电压超过输入电压时，比较器输出切换到低 Vss 电平。 这将引起调制器电容电压开始下降。 当调制器电压再次超过输入电压，调制器电容电压将重新开始上升，并重复调制电容的充电 / 放电周期。 锁存器使比较器操作与时钟 VCI 信号保持同步，并限制电容的最小充电 / 放电间隔。 sigma-delta 调制器通过对调制器电容 C_{mod} 充电或放电，使电压 V_{Cmod} 接近于平均输入电压 V_{in}。

调制器的数学原理很简单。 假设调制器位流占空比为 d_{mod}。 调制器参考电压如下：

Equation 1

$$V_{Cmod} = V_{dd} \cdot d_{mod}$$

考虑到 V_{Cmod} = 平均 V_{in} ，调制器占空比可从以下公式中得出：

Equation 2

$$d_{mod} = \frac{V_{in}}{V_{dd}}$$

ADC 调制器位流滤波器和时钟源

调制器将输入电压转换为输出位流占空比。 占空比使用数字滤波器进行测量。 由于 CY8C21x34 器件没有指定的抽取滤波器，因此，CY8C21x34 器件的实现使用基于计数器的滤波器。 该过滤器由 8-bit 硬件计数器和测量间隔形成电路组成。 为了节省硬件资源，软件里采用了 16 位的计数器，防止计数结束时出现溢

出。 该计数器通过调制器使用通用的 VCI 时钟信号。 当调制器输出为低时，计数器保持原有状态；当调制器输出为高时，计数器逐个减少。

形成测量间隔的电路采用周期中断来读取计数器值，并通过关闭计数器使能输入来阻止计数器的值在读期间发生变化。 内部主振荡器 (IMO) 中的占空比测量间隔 N_m 取决于 VC1、VC2、VC3 和 ADCPWM 值，按以下公式计算得出：

Equation 3

$$N_m = VC1 \cdot VC2 \cdot VC3 \cdot N_{ADCPWM}$$

VC1、VC2、VC3、N_{ADCPWM} - 分频器值。

数字滤波器在每个 N_m IMO 周期建立一个采样，该滤波器采样值与总感应电容值 C_S 成线性比例。 最大采样值由 N_{max} 决定：

Equation 4

$$N_{max} = VC2 \cdot VC3 \cdot N_{ADCPWM}$$

Equation 5

$$N_s = d_{mod} N_{max}$$

Equation 6

$$N = \frac{V_{in}}{V_{dd}} \cdot VC2 \cdot VC3 \cdot N_{ADCPWM}$$

此用户模块使用两个中断：一个是计数器溢出中断，用来扩展 8-bit 硬件计数器的最大计数，另一个是 ADCPWM 中断。 ADCPWM 中断路由到比较器 0 总线中断。 即使只开启一个比较器，也会使用两根比较器总线。

Die 温度测量

ADC 提供片上 die 温度电 路输出信号的测量。 信号作为连续时间模块 ACE01 的 PMux 模拟输入提供。此信号是名义上约为 1.0 V，范围约为 0.7 V 至 1.3 V 之间。输出电压为 die 温度的函数。

电容测量操作

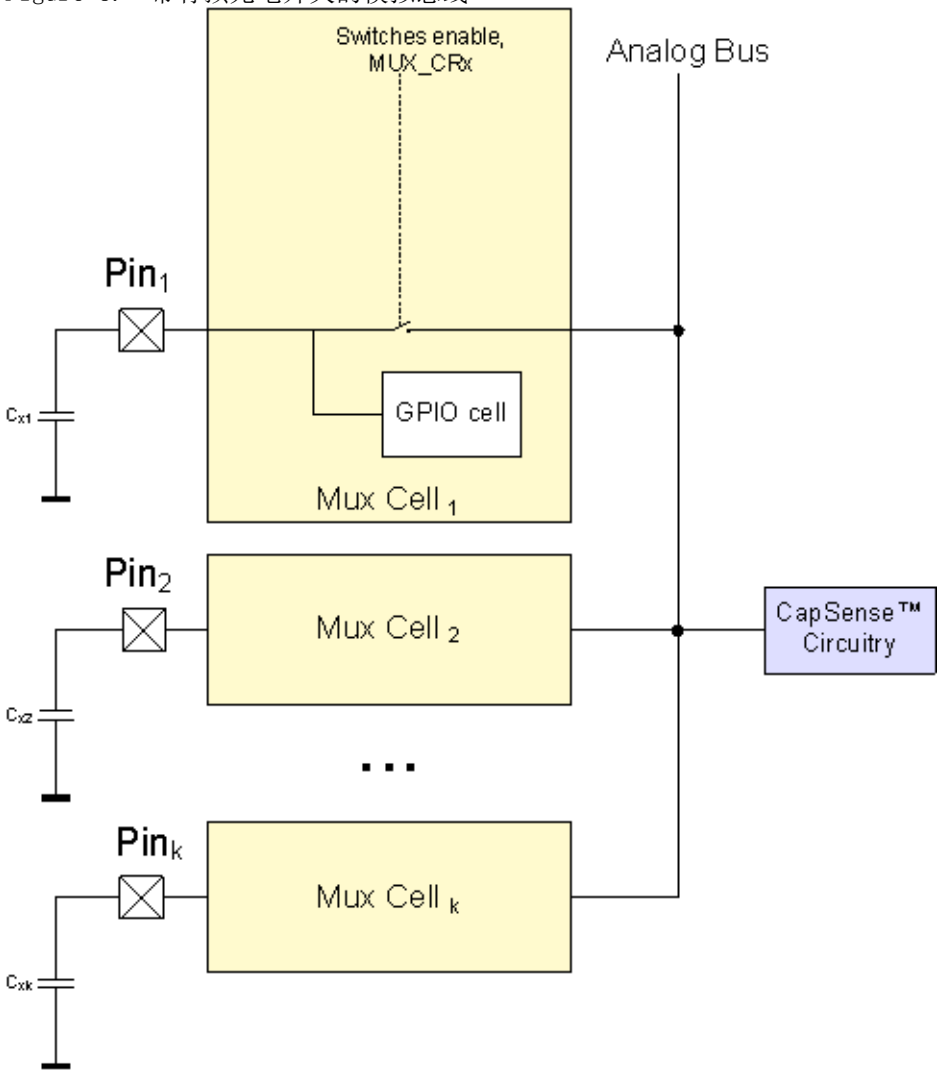
决策逻辑通过固件实现。 通过软件分析测量出的电容量，结合环境因素造成的电容缓慢变化情况，再运行决策逻辑，就可以检测出按键是否被触摸，并计算滑条位置。

扫描传感器阵列

CY8C21x34 系列器件具有内置模拟总线， 可以使电容式传感器连接到任意 PSoC 引脚。 CSDADC 用户模块使用内部预充电开关，在时钟信号相位 Ph_1 为活动传感器充电，在相位 Ph_2 将模拟总线连接到传感器。 sigma-delta 调制器的调制电容和比较器的输入端与模拟总线始终相连。

固件通过在 MUX_CRx 寄存器中设置相应的位来连续执行传感器扫描。

Figure 5. 带有预充电开关的模拟总线

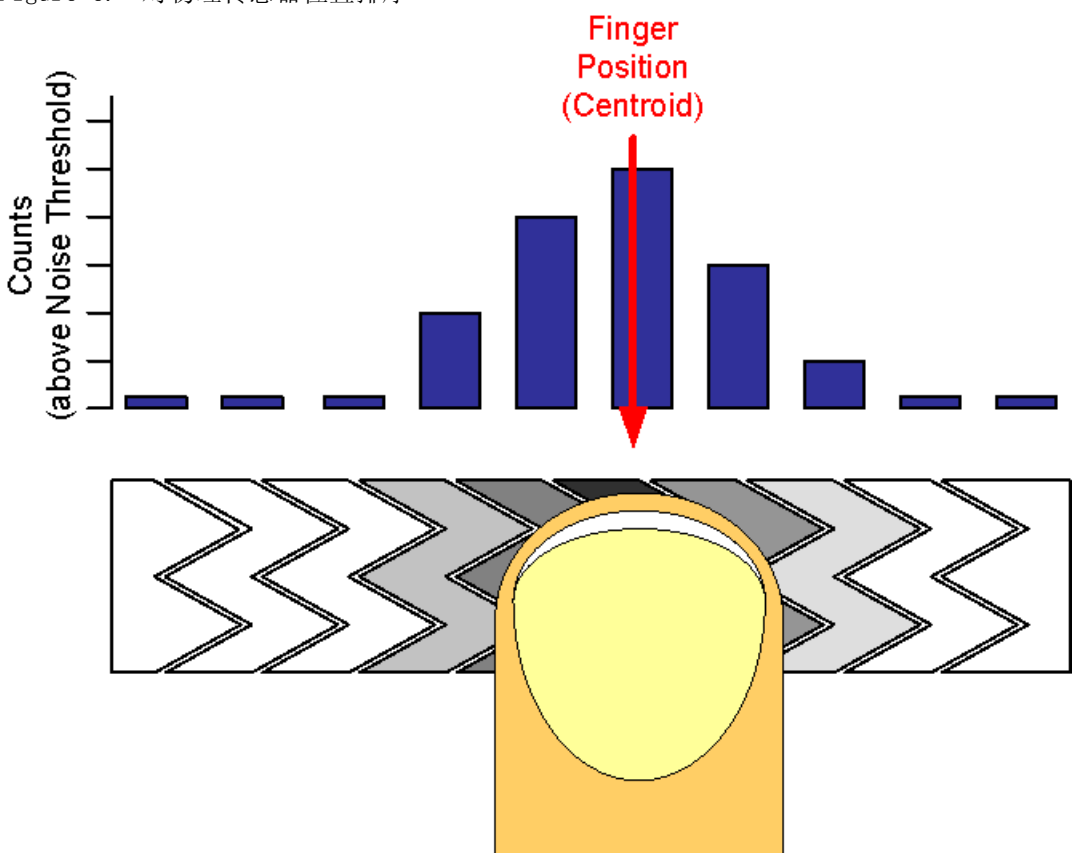


滑条

滑条适用于需要渐进式调节的控件。 示例包括照明控件 （调光器）、音量控件、图示均衡器和速度控件。 这些传感器在布局上彼此相邻。 某个传感器的动作会导致邻近的其他传感器的部分动作。 通过计算活动传

感器组的中心位置，可以确定滑条的实际位置。 CSDADC 向导可以提供滑条，方法是建立若干组，每个滑条组有特定的次序。 滑条传感器的实际下限数量是五，上限就是所选 PSoC 器件上可用的传感器位置数。

Figure 6. 对物理传感器位置排序



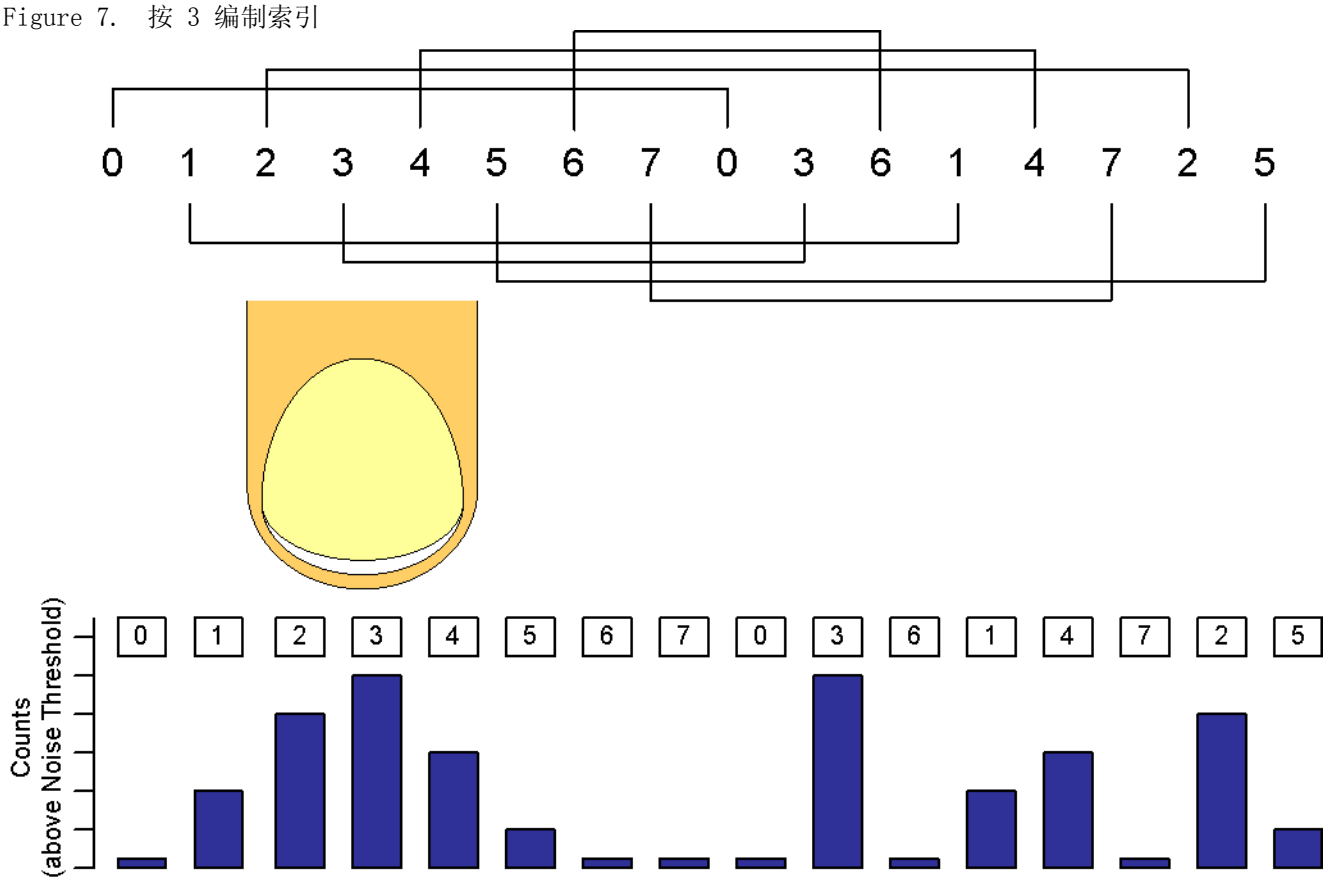
越接近滑条一半部分的强信号，将导致另一半上面产生相同程度的伪信号，但结果是发散的。 感应算法搜索相邻最强的一组信号，以确定解析的滑条位置。

复用

滑条中的每个 PSoC 传感器连接都映射到滑条传感器阵列中的两个物理位置。 物理位置的第一部分 （较低数值部分）按顺序映射到基部分配的传感器上，端口引脚由设计人员使用 CSDADC 向导分配。 物理传感器位置的另一部分 （较高数值部分）由向导中的算法自动映射，并在 include 文件中列出。 一旦创建好次序，某一部分中相邻的传感器动作则不会导致另一部分中相邻的传感器动作。 小心地确定此次序，将其映射到印刷电路板上。

有许多方法可以确定物理传感器位置另一部分的次序。 最简单的方法是对第一部分中的传感器编制索引，先对所有偶数传感器编制索引，然后是所有奇数传感器。 其他方法包括按相关值编制索引。 此用户模块选择的方法是按三编制索引。

Figure 7. 按 3 编制索引



应当使滑条中的传感器电容均衡。 根据传感器或 PCB 布局，某些传感器对可能需要更长的路由。 当选择复用方式时，CSDADC 向导将自动生成复用传感器索引表。 下表说明了不同滑条段数的复用后的顺序。

Table 1. 不同滑条段数的复用顺序

滑条段总数	段顺序
10	0, 1, 2, 3, 4, 0, 3, 1, 4, 2
12	0, 1, 2, 3, 4, 5, 0, 3, 1, 4, 2, 5
14	0, 1, 2, 3, 4, 5, 6, 0, 3, 6, 1, 4, 2, 5
16	0, 1, 2, 3, 4, 5, 6, 7, 0, 3, 6, 1, 4, 7, 2, 5
18	0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 3, 6, 1, 4, 7, 2, 5, 8
20	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 3, 6, 9, 1, 4, 7, 2, 5, 8
22	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8
24	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11
26	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 3, 6, 9, 12, 1, 4, 7, 10, 2, 5, 8, 11
28	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 3, 6, 9, 12, 1, 4, 7, 10, 13, 2, 5, 8, 11

滑条段总数	段顺序
30	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 3, 6, 9, 12, 1, 4, 7, 10, 13, 2, 5, 8, 11, 14
32	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 2, 5, 8, 11, 14
34	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14
36	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14, 17
38	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14 , 17
40	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8 , 11, 14, 17
42	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17, 20
44	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10, 13, 16 , 19, 2, 5, 8, 11, 14, 17, 20
46	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10, 13 , 16, 19, 22, 2, 5, 8, 11, 14, 17, 20
48	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10 , 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20, 23
50	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 0, 3, 6, 9, 12, 15, 18, 21, 24, 1, 4, 7, 10, 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20, 23
52	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 0, 3, 6, 9, 12, 15, 18, 21, 24 , 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23
54	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 0, 3, 6, 9, 12, 15, 18, 21 , 24, 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23, 26
56	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 0, 3, 6, 9, 12, 15, 18 , 21, 24, 27, 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23, 26

复用滑条的滑条段选择指南

选择滑条所需的段数取决于滑条的物理长度。 然而，确定复用滑条的段数时必须特别小心。

在复用方式的滑条设计中，一个传感器用于两个不同的物理滑条段，以增加滑条的物理长度。 手指触摸完全覆盖的段数必须小于从同一传感器派生出的两段之间的传感器数量。 这样可确保复用方式下滑条的正常工作。

例如，在 10-segment 滑条 （5 个传感器）的情况下，从传感器 3 中派生出的两个滑条仅被两个传感器分隔 （传感器 4 和 0）。 在这种情况下，手指触摸不得完全覆盖两个以上的传感器段，以确保滑条正常工作。

对于一个 12-segment 滑条，一个手指触摸不得覆盖超过 3 段。 同样，对于一个 18-segment 滑条，一个手指不得完全覆盖超过 4 段。

插值和定标

在滑动传感器和触摸板应用中，通常需要更精确地分辨手指 （或其他电容物体）的位置，而非单个传感器本身的分辨率。 手指接触滑条传感器或触摸板的面积通常大于任何一个传感器。

为了采用一个中心位置来计算插值后的位置，首先对阵列进行扫描以验证所给定的传感器位置是否有效。要求提供一定数量的相邻传感器信号，且高于噪声阈值。 如果发现最强信号，将使用此信号和那些大于噪声阈值的连续信号计算中心位置值。 使用少至两个、多至 （通常）八个传感器，利用下列公式计算中心位置值：

Equation 7

$$N_{Cent} = \frac{n_{i-1}(i-1) + n_i i + n_{i+1}(i+1)}{n_{i-1} + n_i + n_{i+1}}$$

计算得出的值通常是分数。 为了能够采用某一特定分辨率来报告中心位置 （例如对于 12 个传感器为 0 到 100 的范围），要将中心位置值乘以计算得出的标量。 另一种更有效的方法是将内插和按比例计算的方法统一到单一计算中，按所需的量级直接报告结果。 这一过程可以在高层 API 内进行处理。

滑条传感器总数和分辨率在 CSDADC 向导中进行设置。 向导将计算出标量数值并以分数值的形式进行存储。

位置分辨率的乘数包含在三个字节中，且具有以下位定义：

分辨率乘数最高有效位								
位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
乘数	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
分辨率乘数中等有效位								
乘数	128	64	32	18	16	8	4	2
分辨率乘数最低有效位								
乘数	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256

使用以下公式计算分辨率：

分辨率 = （传感器数量 - 1）× 乘数

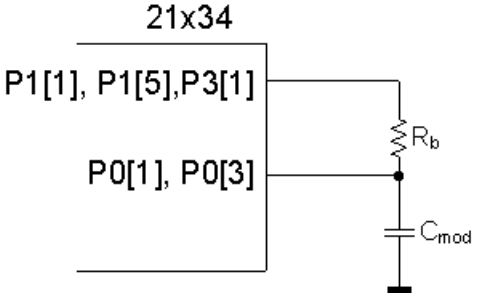
中心位置以 24-bit 无符号整数保存，其分辨率是传感器和乘数的函数。

反馈组件选择指南

用户模块需要外部调制电容 C_{mod} 和调制器反馈电阻 R_b。 电容可以连接到 P0[1]、P0[3] 端口引脚和 Vss 地。 反馈电阻 R_b 可以连接到端口引脚 P1[1]、P1[5]、P3[1] 和电容引脚。 通过用户模块参数设置可以选择引脚。 请勿将为连接调制器组件选择的引脚用于任何其他目的。 必须在建立 CSDADC 用户模块的端口

引脚连接之前，放置使用特定引脚资源 （包括 LCD 和 I2CHW）的用户模块。 当打开向导时，向导中会显示配置选择。

Figure 8. 外部组件连接



调制电容的建议值为 4.7 ～ 47 nF。 可通过实验选择最佳电容值，以获得最大信噪比。 在大多数情况下，对于 PRS16 和 PRS8 配置，5.6 ～ 10 nF 电容值具有较好的效果。 如果选择带有预分频器的配置，则建议的积分电容值为 22 ～ 47 nF。 选择反馈电阻后，可以试验几个电容值，以获得最佳的信噪比。 应使用陶瓷电容。 温度电容系数并不重要。 电阻值取决于总传感器电容 C_S。 应通过以下步骤选择电阻值：

- 监控不同传感器触摸的原始数据。
- 在选定扫描分辨率的情况下，选择的电阻值所提供的最大读数比全量程读数大约低 30%。 电阻值升高时，原始数据会增加。

根据传感器的电容，典型的值为 500Ω 至 10 kΩ 。 如果使用 CY3213 评估板，则可以从 2 kΩ 开始。

屏蔽电极

有些应用场合要求在有水膜或水滴的情况下也能进行可靠的操作。 白色家电、汽车应用、各种工业应用及其他领域需要电容式传感器不因为水、冰和湿度变化而出现误触发情况。 在这种情况下可以使用单独的屏蔽电极。 此电极位于感应电极之后或其外侧。 如果器件绝缘外覆层表面有水膜，则屏蔽和感应电极之间的耦合程度会加剧。 屏蔽电极有助于降低寄生电容的影响，为处理传感电容的变化提供了更具动态性的数值范围。

在某些应用情况下，选择屏蔽电极信号以及屏蔽电极相对于感应电极的放置位置是非常有用的，这样可增加两种电极之间的耦合程度，使感应电极电容测量的触摸变化朝着相反方面改变。 这就简化了高层级的软件 API 工作。 CSDADC 用户模块支持屏蔽电极的单独输出。

Figure 9. 可能的屏蔽电极 PCB 布局

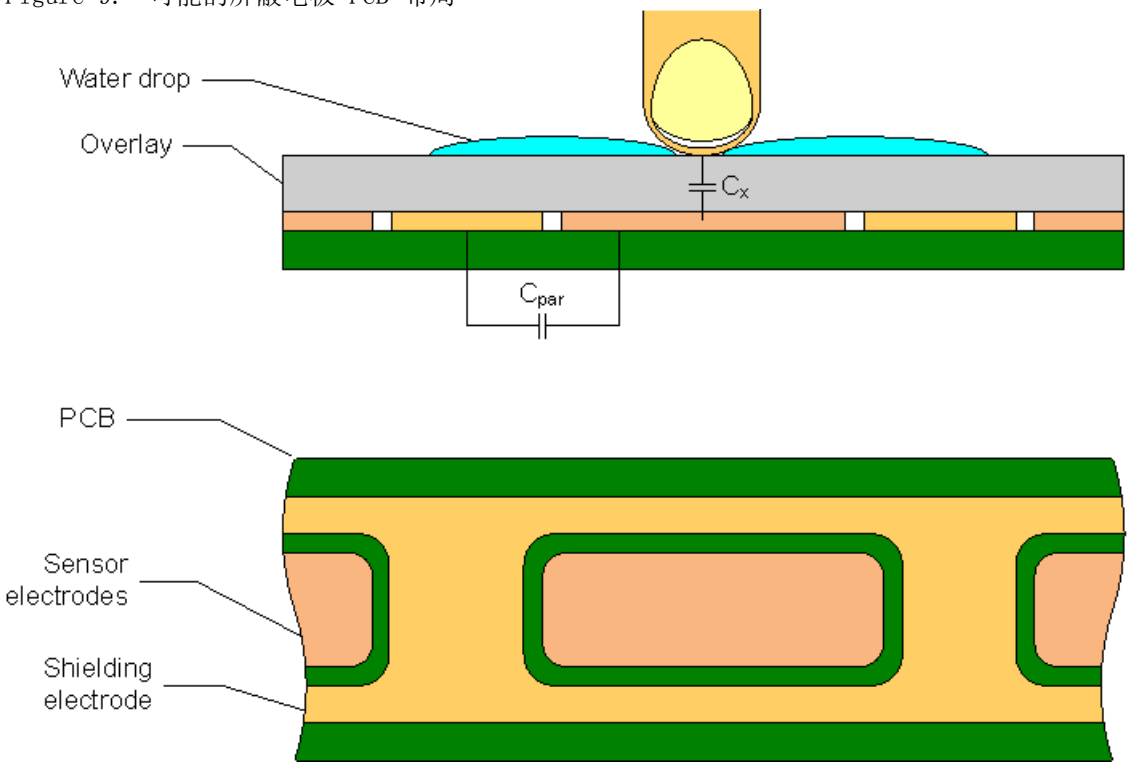


图 9 提供一种按钮屏蔽电极的可能布局配置。 屏蔽电极尤其适用于透明的 ITO 触摸板设备，在这种设备中，它不但可阻止 LCD 驱动电极的噪声影响，同时可减少杂散电容。

在本例中，按键上覆盖有一层屏蔽电极面。 作为另一替代方法，屏蔽电极可以安装在相对的 PCB 层上，其中包括按钮下面的平板。 对于这种情况，建议使用填充模式，填充率约为 30 ～ 40%。 这时无需额外的接地层。

屏蔽电极可以连接到它可以路由到的任何 PSoC 引脚。 将驱动模式设置为**很慢**，可以降低接地噪声和辐射。 另外，可以在 PSoC 器件与屏蔽电极之间连接上升限制电阻。

时钟源

时钟源用于控制感应电容上的开关。 用户模块支持将下列四个选择选项作为预充电开关的时钟源。 下表比较了这几种配置：

配置	工作频率	使用的数字模块	抗 EMC 噪声能力
PRS16	扩频，平均值为 $F_{IMO}/4$，峰值为 $F_{IMO}/2$	3	高。 敏感点是 PRS 序列重复周期的倍数和 PRS 基频 F_{IMO} 的倍数。
PRS8	可调整扩频，$F_{IMO}/4$ - $F_{IMO}/512$	2	中等。 由于 PRS 重复周期较短，因此有更多敏感点。 如果需要较好的抗 EMI 能力，请使用 PRS16 配置。
预分频器	固定，IMO/（周期 +1）	2	在操作频率及其谐波下，器件对 EMC 信号敏感。 建议仅用于通过高电阻材料进行的操作。
VC_2	固定，IMO/(VC_1xVC_2)	1	在操作频率及其谐波下，器件对 EMC 信号敏感。 建议仅在未计划认证 EMC/EMI 测试时使用。

调制器参考源

比较器参考源用于形成比较器参考电压。 该参考电压值决定了灵敏度。

下表汇总了各参考选择选项：

外部组件	UM 选择	使用场合
无	VBG	读数与电源电压成比例。 仅在电源稳压良好时使用
无	ASE11	建议应用于大多数场合。 尝试从此选项开始测试。
2	模拟列 (AnalogColumn) 输入选择	读数与电源相关性较低。 建议 R1 = 10k; R2 = 3.6k
2	模拟列 (AnalogColumn) 输入选择	如果使用其他参考选择，会有过大噪声。

您很有可能仅使用 VBG 和 ASE11 参考选择。 其他两种选择在一些特殊的应用场合会大有用途。

直流和交流电气特性

Table 2. 电源电压

参数	最小值	典型值	最大值	单位	测试条件和注释
值	2.7	5.0	5.25	V	

Table 3. 电容式感应噪声

参数[a]	最小值	典型值	最大值	单位	测试条件 (Vdd = 5V, SysClk = 24 MHz, CPU 时钟 = 12MHz, 基准 >= 分辨率最大计数的 70%), PRS16
噪声数值		4		峰至峰	分辨率 = 14
噪声数值		7		峰至峰	分辨率 = 12
噪声数值		12		峰至峰	分辨率 = 10

a. 当扫描速度减慢且基准线数值增加时，信噪比提高。

Table 4. 电容式感应噪声

参数[a]	最小值	典型值	最大值	单位	测试条件 (Vdd = 3.3V, SysClk = 12 MHz, CPU 时钟 = 6 MHz, 基准 >= 分辨率最大计数的 70%)
噪声数值, 峰 - 峰		0.2		%, (噪声数值) / (基准线数值)	分辨率 >= 14
噪声数值, 峰 - 峰		1		%, (噪声数值) / (基准线数值)	分辨率 = 12
噪声数值, 峰 - 峰		10		%, (噪声数值) / (基准线数值)	分辨率 = 10

a. 当扫描速度减慢且基准线数值增加时，信噪比提高。

Table 5. 电容式感应噪声

参数[a]	最小值	典型值	最大值	单位	测试条件 (Vdd = 2.7V, SysClk = 12 MHz, CPU 时钟 = 6 MHz, 基准 >= 分辨率最大计数的 70%)
噪声数值		9		峰至峰	分辨率 = 12

a. 当扫描速度减慢且基准线数值增加时，信噪比提高。

Table 6. 电容扫描期间的功耗

参数	最小值	典型值	最大值	单位	测试条件 (Vdd = 2.7V，SysClk = CPU 时钟 = 6 MHz)
有功电流		1.43		mA	扫描期间平均电流，8 个传感器
待机电流		40		µA	扫描速度 = 超快，分辨率 = 9，100ms 报告速率，8 个传感器
		250		µA	扫描速度 = 快，分辨率 = 12 100 ms 报告速率，8 个传感器
睡眠 / 唤醒电流		5		µA	1s 报告速率，1 个传感器

Table 7. 电容扫描期间的功耗

参数	最小值	典型值	最大值	单位	测试条件 (Vdd = 3.3V，SysClk = CPU 时钟 = 6 MHz)
有功电流		1.9		mA	扫描期间平均电流，8 个传感器
待机电流		50		µA	扫描速度 = 超快，分辨率 = 9，100ms 报告速率，8 个传感器
		300		µA	扫描速度 = 快，分辨率 = 12 100 ms 报告速率，8 个传感器
睡眠 / 唤醒电流		6		µA	1s 报告速率，1 个传感器

Table 8. 电容扫描期间的功耗

参数	最小值	典型值	最大值	单位	测试条件 (Vdd = 5.0V，SysClk = CPU 时钟 = 6 MHz)
有功电流		3.18		mA	扫描期间平均电流，8 个传感器
待机电流		90		µA	扫描速度 = 超快，分辨率 = 9，100ms 报告速率，8 个传感器
		550		µA	扫描速度 = 快，分辨率 = 12 100 ms 报告速率，8 个传感器
睡眠 / 唤醒电流		7		µA	1s 报告速率，1 个传感器

Table 9. 增量型模数转换器特性

参数	典型值	限制	单位	条件与说明
工作电流		3.18	mA	5V 电源。 CPU 6MHz
噪声，比率计		3	LSB	
分辨率，比率计		9 至 16	位	
分辨率，绝对		12	位	
采样率，比率计		0.04 至 16.6	Ksps	取决于分辨率和扫描速度

参数	典型值	限制	单位	条件与说明
采样率，绝对		0.58 至 4.6	Ksps	
输入				
输入电压范围	-	Vss+0.3 至 Vdd-0.3	V	
输入电容		3	pF	

放置

用户模块所用的各模块将在用户模块实例化后自动放置，而没有提供备选放置方式。CSDADC 使用 1 到 3 个数字模块，具体取决于配置。 CSDADC 的模拟部分布置在 ACE00、ACE01 和 ASE11 中；没有提供备选布置方式。

必须在建立 CSDADC 用户模块的端口引脚连接之前，放置使用特定引脚资源（包括 LCD 和 I2CHW）的用户模块。 当打开向导时，向导中会显示配置选择。

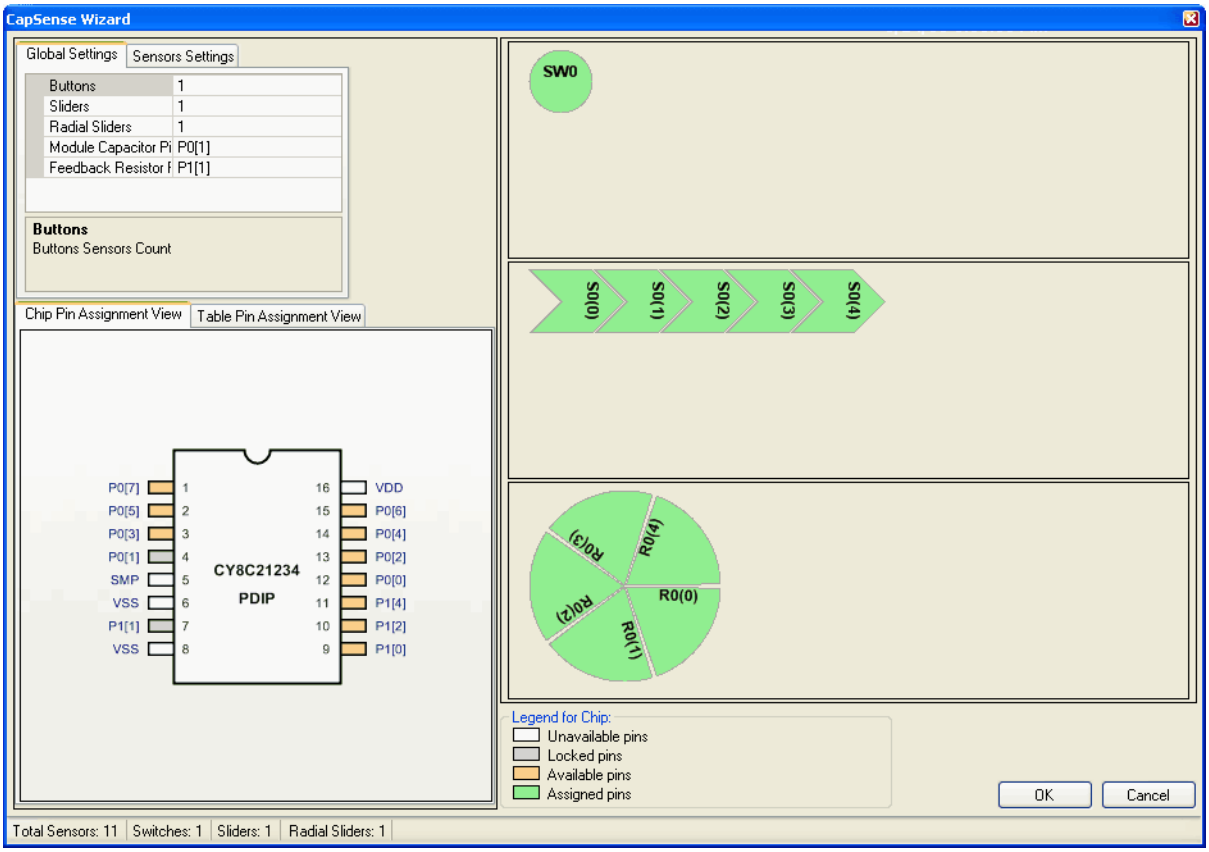
在布置电容式传感器连接时，应避免使用 P1[0] 和 P1[1]。 这些引脚用来对部件进行编程，而且有可能存在过大的布线电容值，从而会影响传感器的灵敏度和噪声。

访问向导

1. 要访问此向导，请在 “ 器件编辑器互连视图 ” 中右键单击任意 CSDADC 模块，然后选择 “CSDADC 向导 ”。



2. 向导打开，显示有传感器和滑条传感器数量的数值输入框。



向导引脚图标

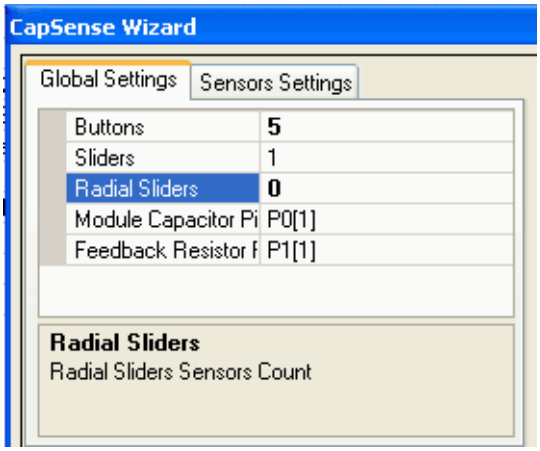
白色 - 该引脚不能用作 CapSense 输入。

灰色 - 引脚处于锁定。 这种情况有两种可能的原因。 一种可能的原因是另一个用户模块（如 LCD 或 I^2C）已占用了该引脚。 第二种可能性是引脚已更改为使用非默认名称。要恢复使用引脚的默认名称，请在 “ 引脚分布 ” 视图中展开引脚，然后从**选择** 菜单中选择**默认**。现在可以在向导中分配引脚了。

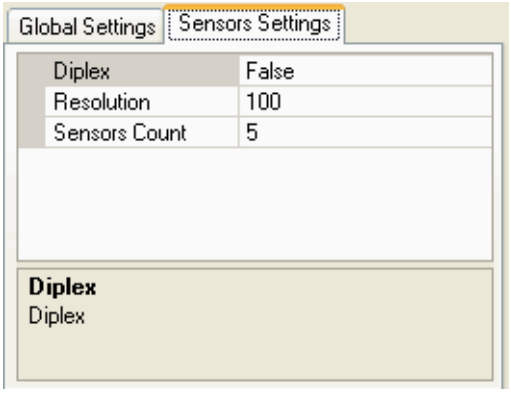
橙色 - 引脚可用于分配。

绿色 - 引脚已分配为 CapSense 输入。

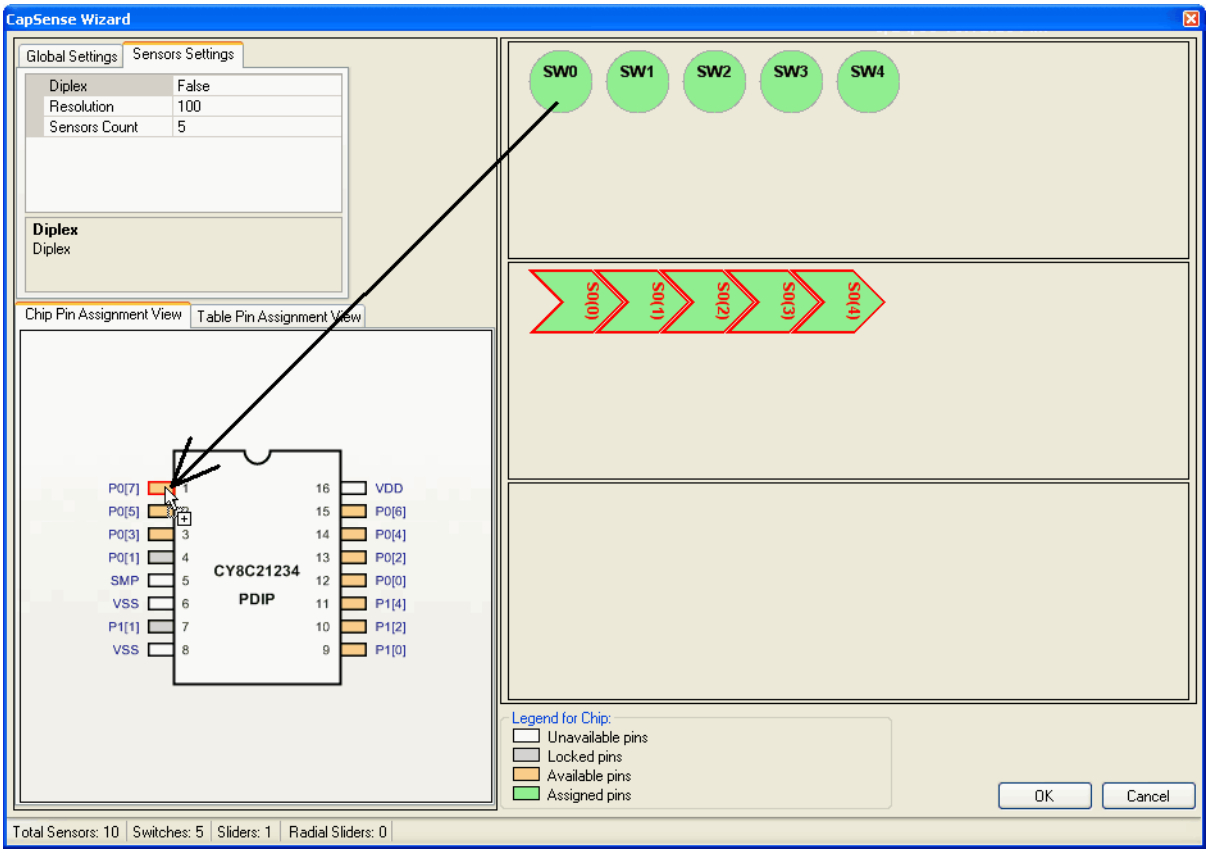
3. 键入独立按键、滑条和径向滑条的数量。 X-Y 触摸板需要两个滑条，但是选择了一个。 传感器数量限制为可用引脚的数量。



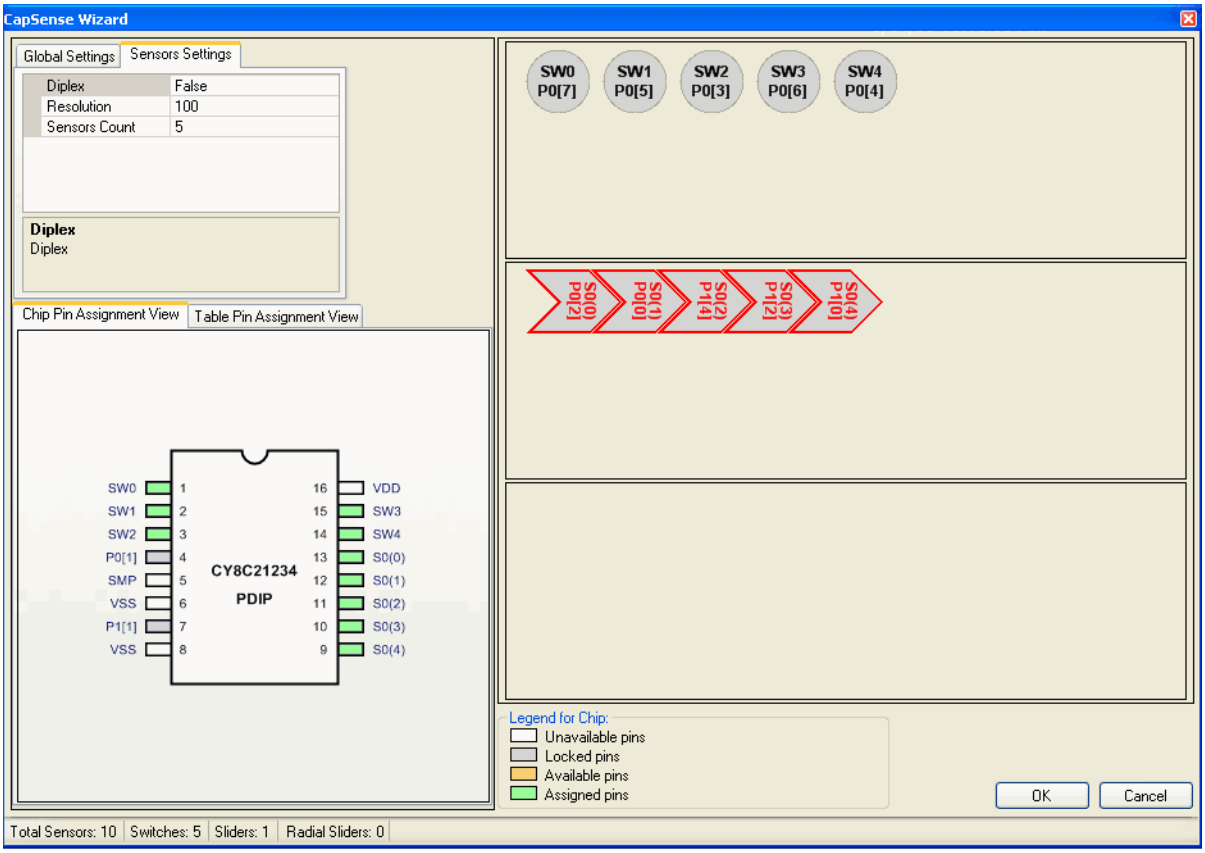
4. 选择 “ 传感器设置 ” 选项卡即可设置滑条和辐射状滑条。 键入每个滑条中传感器元件的数量。 滑条传感器中的传感器实际最小数量为五，最大值受限于引脚数量。



5. 键入输出分辨率。 最小值为 5。 对于复用型滑条，最大值为 （传感器的引脚数 - 1） x 2^8 - 1 或 （2 x 传感器引脚数 - 1） x 2^8 -1
6. 如果需要，选择 “ 复用方式 ”。 这将把为传感器选定的引脚数值映射为板上传感器位置的两倍数值。 仅显示了复用传感器的前半部分；后半部分按前面 “ 复用 ” 章节所述自动映射。 有关引脚连接的复用顺序表，请参见 “ 复用 ” 一节。
7. 单击按键，将其拖动到任意可用引脚。 选择端口引脚后，引脚变为绿色，不再可用。 通过将传感器拖离端口引脚，可更改传感器分配。



8. 对其他独立按键重复以上操作。
9. 将单个滑条传感器映射到物理端口引脚的方法与映射按键的方法相同。



10. 单击**确定**接受数据，然后返回到 PSoC Designer。

传感器放置现在已完成。 右键单击 “ 器件编辑器 ” 窗口并选择**刷新**即可更新引脚连接。

设置用户模块参数，生成应用程序。 如果需要，可以立即对示例项目进行调整。

在 CSDADC 向导中输入数值时，请首先删除原来的值，然后再输入新值。 编辑框中未显示光标。

如果想要更改引脚分配，请将光标放在分配的引脚上，单击引脚，拖放至开关框外侧。 此时引脚就会处于未分配状态，您可以重新分配。

完成向导后，单击 “ 生成应用程序 ”。 根据您输入的传感器数量、引脚分配、是否使用复用模式和分辨率，会生成一系列表格。 这些表格位于 CSDADC_Table.asm 中

传感器表

在传感器表中，每个传感器对应一个 2-byte 条目。 第一个字节是端口号，第二个字节是位的掩码 （不是位编号）。 表格中列出了所有的独立传感器，然后是按顺序排列的各个传感器。 下面是一个包含六个传感器的表的示例：

```
CSDADC_Sensor_Table:
_CSDADC_Sensor_Table:
      dw    0x0140  //  Port 1 Bit 6
      dw    0x0301  //  Port 3 Bit 0
      dw    0x0304  //  Port 3 Bit 2
      dw    0x0308  //  Port 3 Bit 3
      dw    0x0302  //  Port 3 Bit 1
      dw    0x0108  //  Port 1 Bit 3
```

该表由 CSDADC_wGetPortPin() 例程使用。

分组表

分组表定义了每个按键传感器组或滑条组。 每个滑条对应一个条目，自由按键传感器再对应一个条目。 第一个条目始终对应自由传感器。 每个条目为六个字节。 第一个字节表明传感器表中组开始的索引。 第二字节代表该组传感器的数量。 第三个字节表示是否对滑条采用了复用 （4 表示已采用复用方式，0 表示未采用）。 第四、五、六个字节是固定点乘数，将其与计算出的滑条中心位置值相乘可以获得 CSDADC 向导中所需的分辨率。

```
CSDADC_Group_Table:
_CSDADC_Group_Table:
; Group Table:
;    Origin    Count    Diplex?    DivBtwSw(wholeMSB, wholeLSB, fractByte)
 db    0x0,      0x3,      0x00,       0x00,     0x00,     0x00 ; Buttons
 db    0x3,      0x8,      0x4,        0x0,      0x0,      0x44 ; Slider 1
```

复用表

当某个组是滑条且采用复用方式时，将为该组生成复用扫描顺序，并按照该顺序扫描。 否则，就会只创建标签但不包含任何数据。 该表由两个部分组成：针对每个滑条的传感器映射，以及每个单独滑条对其表格的引用。 八个传感器滑条的典型示例为：

```
DiplexTable_0:
; This group is not a diplexed slider
DiplexTable_1:
db0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5// 8 switch slider


 CSDADC_Diplex_Table:
_CSDADC_Diplex_Table:
db >DiplexTable_0, <DiplexTable_0
db >DiplexTable_1, <DiplexTable_1
```

参数和资源

ADCEnabled

ADCEnabled 参数可以有两个值：

- 已启用 （默认值）
- 禁用

当此参数设置为 “ 启用 ” 时，ADC 例程将包含在可编译代码中并可从用户代码调用。 当此参数设置为 “ 禁用 ” 时，将不包含 ADC 例程。 如果设计中不再需要 ADC，此设置可能会有帮助。 当将此参数设置为 “ 禁用 ” 时，CSDADC 用户模块与 CSD 用户模块的作用相似。

手指阈值

此阈值用于确定每个按键传感器的状态。 如果任何传感器处于活动状态，则 bIsAnySensorActive() 函数返回 ‘1’。 如果所有传感器关闭，则 bIsAnySensorActive() 函数返回 ‘ 0’。

手指检测阈值适用于所有传感器和滑条。 对于单个传感器 （滑条组中不包含），这些阈值是变量，在 baBtnFThreshold[] 阵列中提供。 可以使用 SetDefaultFingerThresholds() 函数将阈值设置为

器件编辑器中所设置的默认值。 要调整单个传感器的灵敏度，请更改每个传感器的 baBtnFThreshold[] 值。 （此字节阵列的大小等于部署的各个传感器的数量。）

可能值的范围为从 5 到 255。 默认值为 40。

Die 温度测量

启用或禁用现有测量 VTEMP 电压并将电压转换成相应温 度 值的模拟转换器 (ADC) 功能。 该参数仅在 ADCEnabled 参数设置为 “ 已启用 ” 时可设置为 “ 启用 ”。 默认情况下，禁用此参数。

噪声阈值

对于单个传感器，此参数将设置一个计数值，超出此值时基准线值将不会更新。 对于滑条传感器，如果低于其设置的计数值，则结果将不会计入中心位置值计算。 可能值为 5 到 255。 默认值为 20。

基准线更新阈值 (BaselineUpdate Threshold)

如果新的原始计数值高于当前基准线，差值低于噪声阈值 （“ 传感器自动复位 ”(Sensors Autoreset) 参数设置为 “ 禁用 ”），则当前基准线与原始数值的差值累计到水桶中。 当水桶充满时，基准线按某个值递增，并清空水桶。 此参数用于设置为了使基准线增大 ″ 水桶 ″ 所必须达到的阈值。 可能值为 0 到 255。 参数值越大，基准线更新速度越慢。 如果需要进行更加频繁的基准线更新，请减小此参数。 默认值为 200。

低基准线复位 (LowBaselineReset)

“ 低基准线复位 ”(LowBaselineReset) 参数与 “ 负噪声阈值 ”(NegativeNoiseThreshold) 参数协同工作。 如果采样到的数值低于基准线与 “ 负噪声阈值 ”(NegativeNoiseThreshold) 之差，并且低于它的次数达到指定的采样次数，基准线会将设置为新的原始数值。 此参数实际上是对重设基准线所需的异常低的采样数值进行计数。 它通常用来纠正启动时手指已经在传感器上面的情况。 可能值为 0 到 255。 默认值为 50。

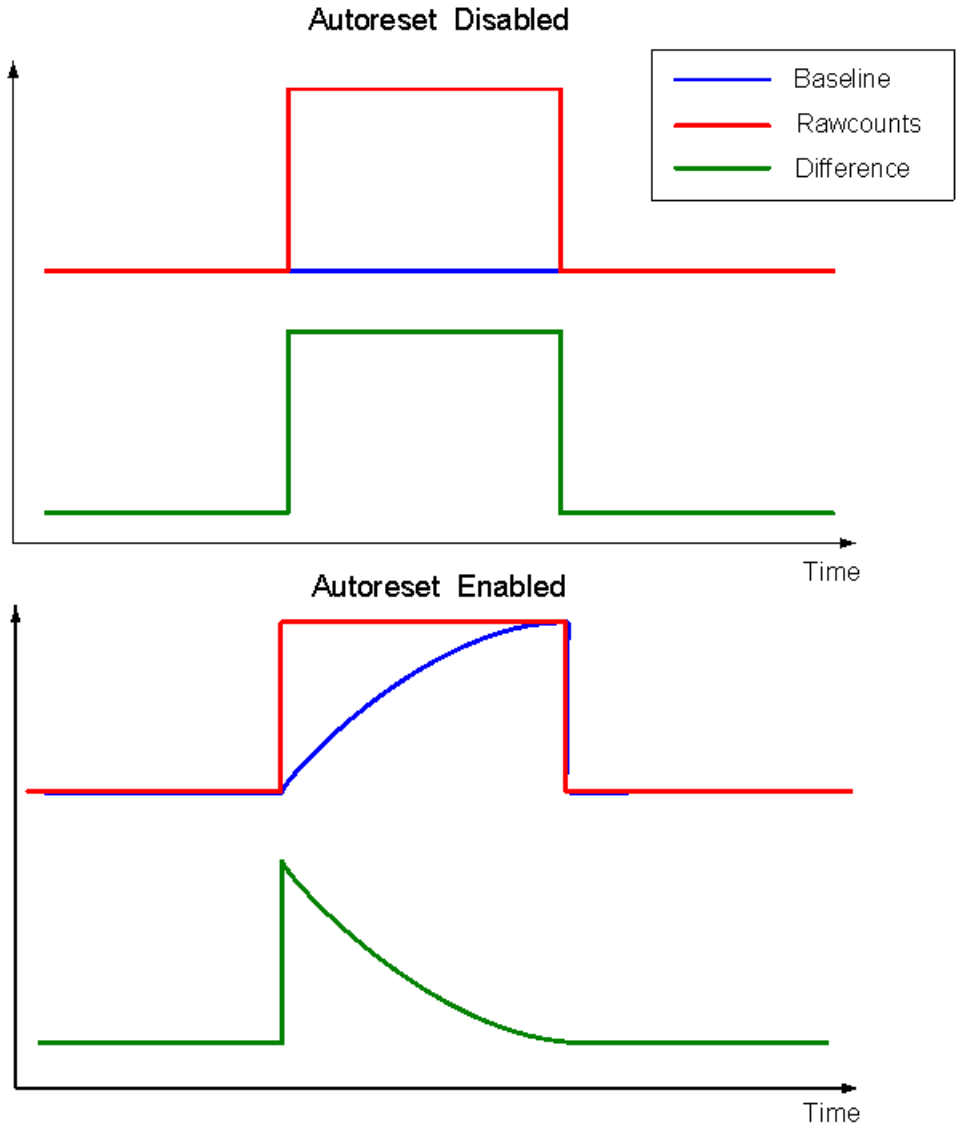
传感器自动复位

此参数确定基准线是否随时更新，还是仅当信号差值低于噪声阈值时更新。 当设置为**启用**时，基准线随时更新。 此设置限制传感器的最大持续时间 （典型值为 5 – 10s），但是当无任何物体触碰传感器而原始计数值突然上升时，可以阻止传感器始终打开。 较大的电源电压波动、高能射频噪声源或极快的温度变化都会导致这种突然升高。 此参数是默认为启用。

当此参数设置为**禁用**，则仅当原始数值与基准线的差低于噪声阈值参数时基准线才进行更新。 除非需要长期将传感器的状态保留为开，否则应将此参数设置为 “ 启用 ”。

图 10 说明了此参数对基线更新的影响。

Figure 10. 传感器自动复位参数



迟滞

“ 迟滞 ” 参数根据传感器当前是处于活动还是非活动，来增大或减小手指阈值。 如果传感器处于非活动状态，则差值必须大于手指阈值与迟滞的和。 如果传感器处于活动状态，则差值必须低于手指阈值与迟滞的差。 此参数用于增加手指检测算法的平稳性和牢固性。 当调用 bIsSensorActive() 或 bIsAnySensorActive() 时，计算带有迟滞的阈值。 可以用 bIsSensorActive() 或 baSnsOnMask[] 阵列的返回值监控传感器状态。 可能的值为 0 到 255，但是必须小于 “ 手指阈值 ”(Finger Threshold) 参数设置。 默认值为 10。

只有正确选择高级决策逻辑参数，才能高效补偿环境温度因数 （温度、湿度变化等），抑制噪声信号（ESD 和电源尖峰脉冲），并在各种使用情况下提供可靠触摸检测。

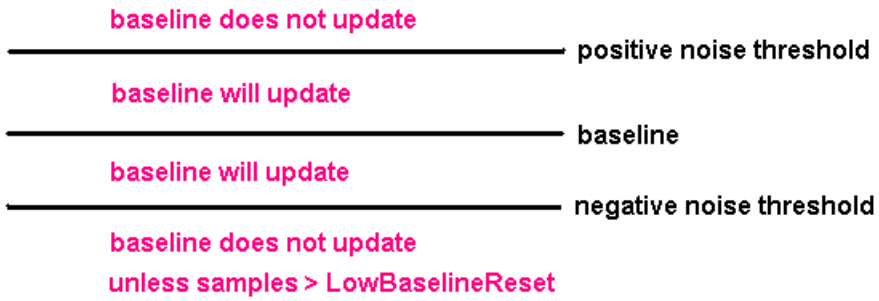
防反跳

“ 防反跳 ” 参数为传感器活动的瞬变增加了防反跳计数器。 为了让传感器能够从不活动状态切换到活动状态，在规定的样本数量内，差异值必须保持在手指阈值加迟滞值之上。 防反跳计数器按 bIsSensorActive 或 bIsAnySensorActive API 函数递增。

可能值为 1 到 255。 设置为 1 则不提供防反跳。 默认值为 3。

负噪声阈值 (NegativeNoiseThreshold)

“ 负噪声阈值 ”(NegativeNoiseThreshold) 参数增加负的差值阈值。 如果当前原始数值低于基准线且二者的差值大于此阈值，则不会更新基准线。 但是，如果当前原始数值一直保持低状态 （差值大于阈值），并且次数超过 “ 低基准线复位 ”(LowBaselineReset) 参数指定的采样次数，基准线会被重置。 可能值为 0 到 255。 默认值为 20。



低基准线复位 (LowBaselineReset)

“ 低基准线复位 ”(LowBaselineReset) 参数与 “ 负噪声阈值 ”(NegativeNoiseThreshold) 参数协同工作。 如果采样到的数值低于基准线与 “ 负噪声阈值 ”(NegativeNoiseThreshold) 之差，并且低于它的次数达到指定的采样次数，基准线会将设置为新的原始数值。 此参数实际上是对重设基准线所需的异常低的采样数值进行计数。 它通常用来纠正启动时手指已经在传感器上面的情况。 可能值为 0 到 255。 默认值为 50。

扫描速度 (Scanning Speed)

此参数影响传感器的扫描速度。 可选择的速度包括： **超快、快、标准和慢**。 默认值为 “ 正常 ”。 较慢的扫描速度具有以下优势：

- 信噪比提高
- 更好地应对电源和温度的变化
- 很少需要系统中断延迟；可以处理较长的中断。

有关中断延迟的更多信息，请参见 “ 警告 ” 一节。

分辨率

此参数确定扫描分辨率 （以 “ 位 ” 为单位）。 可以用 9 到 16 位的分辨率来扫描传感器。 默认分辨率为 12 位。N 位扫描分辨率的最大原始计数为 2^N-1。

增大分辨率可提高触摸检测的灵敏度和信噪比。 对于接近检测，请使用高分辨率。 通过 16-bit 分辨率、慢速扫描模式和一根 20 cm 导线，可以在 20 cm 或更远距离检测到人手。

扫描速度和分辨率通过以下方式影响 VC1、VC2、VC3 和 ADCPWM 分频器：

扫描速度 (Scanning Speed)	VC1
超快	1
快速	2
正常	4
慢速	8

分辨率，位	VC2	VC3	ADCPWM
9	8	16	4
10	8	32	4
11	8	64	4
12	8	128	4
13	8	256	4
14	8	256	8
15	8	256	16
16	8	256	32

VC1 分频器仅取决于扫描速度。 VC2、VC3 和 ADCPWM 仅取决于分辨率。

Table 10. 对于 24 MHz IMO 操作、PRS16 配置的情况，扫描时间 （以 µs 为单位）与扫描速度和分辨率的关系

分辨率，位	扫描速度			
	超快	快速	正常	慢速
9	75	110	170	300
10	110	170	300	510
11	170	300	510	1010
12	300	510	1010	2030
13	510	1010	2030	4060
14	850	1690	3380	6760
15	1520	3040	6080	12200
16	2880	5720	11500	23200

Table 11. 对于 24 MHz IMO 操作、带有预分频器配置的 PRS8 的情况，扫描时间 （以 µs 为单位）与扫描速度和分辨率的关系

分辨率，位	扫描速度			
	超快	快速	正常	慢速
9	60	85	150	255
10	85	150	255	510
11	150	255	510	1020
12	255	510	1020	2040
13	510	1020	2040	4080
14	845	1700	3380	6760
15	1530	3060	6120	12100
16	2880	5800	11500	23000

Note 扫描时间是按两次传感器扫描的时间间隔测量的。 此时间包括传感器设置时间、调制器稳定延迟、采样转换间隔和数据预处理时间。

调制器电容引脚 (Modulator Capacitor Pin)

此参数设置引脚以连接外部调制器电容 (C_{mod})。 从可用引脚 P0[1]、P0[3] 中选择。默认引脚为 P0[1]。

反馈电阻引脚 (Feedback Resistor Pin)

此参数设置引脚以连接外部反馈电阻 (R_b)。 从以下可用引脚中选择： P1[1]、P1[5] 和 P3[1]。在某些器件封装中，部分引脚不可用。 提示：如果这些引脚中的一些引脚用于其他用途 （例如，分配用于传感器连接），它们在用户模块参数列表中不可选择。 CSDADC 用户模块的将来版本可允许使用附加引脚来连接反馈电阻。 此将允许在没有 P3 端口的封装中使用另一个 I^2C 端口。 使用引脚 P1[5] 或 P3[1] 是为了避免编程问题。 默认引脚为 P1[1]。

参考

此参数设置比较器参考源。 更多信息，请参见以下 Ref 值参数说明。 默认情况下，引用来自带隙 (VBG)。

参考值 (Ref Value)

当比较器参考值来自模拟调制器 (ASE11) 或外部滤波 PWM/PRSPWM 信号 （来自带有 RC 滤波器的 AnalogColumn_InputSelect_1）时，此参数用于设置比较器参考值。 当参考来自带隙 (VBG) 或外部电压分频器 （来自带有电阻式电压分频器的 AnalogColumn_InputSelect_1）时，此值无效。 默认 Ref 值为 0。

“0” 对应最小参考值 (1/4 Vdd)。 “8” 对应于最大参考值 (3/4Vdd)。 当参考值增大时，灵敏度下降，但是对屏蔽电极的影响增大。

如果设计使用的传感器存在显著的电容差异 （例如，传感器具有大小不同的正方形），则可以使用 API 函数为具有较大电容的传感器设置较高的参考值，来平衡原始数值。

Note 此参数在具备 VC2 时钟源配置的 CSDADC 中不可用。

预分频器周期 (Prescaler Period)

此参数设置预分频器周期寄存器，并确定预充电开关输出频率。 此参数仅可用于带预分频器的配置。 预分频器周期值的范围为 1 到 255。 默认预分频器周期为 7。

建议值为 2^n-1 以获取最大信噪比 (SNR)：

- 1
- 3
- 7
- 15
- 31
- 63
- 127
- 255

其他值会导致更多噪声，尤其是在低分辨率和高扫描速度的情况下。

屏蔽电极输出 (ShieldElectrodeOut)

可以从其中一个备用数字行总线 (Row_0_Output_0 - Row_0_Output_3). 中选择屏蔽电极信号源。每行输出都可以路由到三个引脚当中的一个。 将行 LUT 功能设置为 A。

Note 此参数仅可用在具有 PRS8/PRS16 时钟源配置的 CSDADC 中。 在具有 PWM8 时钟源配置的 CSDADC 中，屏蔽电极信号始终与 Row_0_Output_0. 连接。

应用程序编程接口

应用程序编程接口 (API) 函数作为用户模块的一部分提供，使您能够在较高的层级处理模块。 本节具体说明了每个函数对应的接口以及 include 文件所提供的相关常量。

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 CSDADC_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 CSDADC。

Note ** 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能会通过调用 API 函数而发生更改。 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。 选择这种“寄存器易失”策略是为了提高效率，并且从 PSoC Designer™ 的 1.0 版本起已开始使用。 C 编译器自动遵循此要求。 汇编语言编程人员也必须确保其代码遵守该策略。 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。

对于大型储存器模块驱动， 保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是调用程序的职责。 尽管部分寄存器现在可能不可修改，但是无法保证在将来的版本中也会如此。

提供的进入点用于初始化 CSDADC、启动其采样和停止 CSDADC。 在所有情况下，模块的实例名称会替换以下进入点中显示的 CSDADC 前缀。 未能使用正确的名称是常见的语法错误原因。

大多数 CSDADC API 函数源自 CSD 用户模块和 ADC10 API。 如果您的现有代码来自使用了两个用户模块的设计，那么该代码的更改将是最小的。 大部分函数的名称分别与两个单独的用户模块相同。 有些名称非常相似的函数在 CapSense 和 ADC 模块中所执行的功能却不同，这是函数名保持相同的一个副作用。 请认真阅读函数说明，避免出现编码错误。 新增了几个 API 例程为 ADC 功能提供支持。

API 函数使用不同的全局阵列。 不得手动更改这些阵列。 不过，您可以出于调试目的对这些值进行检查。 例如，可以使用绘图工具显示阵列的内容。 以下是几个全局阵列：

- CSDADC_waSnsBaseline[]
- CSDADC_waSnsResult[]

- CSDADC_waSnsDiff[]
- CSDADC_baSnsOnMask[]

CSDADC_waSnsBaseline[] – 这是一个整数阵列，其中包含每个传感器的基准数据。 阵列大小与电容传感器数量相等。 CSDADC_waSnsBaseline[] 阵列通过下列函数更新：

- CSDADC_UpdateAllBaselines();
- CSDADC_UpdateSensorBaseline();
- CSDADC_InitializeBaselines().

CSDADC_waSnsResult[] – 这是一个整数阵列，包含每个电容传感器的原始数据。 阵列大小与传感器数量相等。 CSDADC_waSnsResult[] 数据通过下列函数更新：

- CSDADC_ScanSensor();
- CSDADC_ScanAllSensors().

CSDADC_waSnsDiff [] – 这是一个整数阵列，包含每个电容传感器的原始数据与基准数据之间的差值。 阵列大小与传感器数量相等。

CSDADC_baSnsOnMask[] – 这是一个保持电容传感器开 / 关状态的字节阵列 （对于按键或滑条）。 CSDADC_baSnsOnMask[0] 包含传感器 0 到 7 的掩码位 （传感器 0 为 0 位，传感器 1 为 1 位）。 CSDADC_baSnsOnMask[1] 包含传感器 8 到 15 的掩码位 （如果需要），依此类推。 此字节阵列包含的元素数足以包含所有放置的传感器。 按键开启时位值为 1，关闭时位值为 0。 CSDADC_baSnsOnMask[] 数据由 CSDADC_blsSensorActive(BYTE bSensor) 函数或 CSDADC_bIsAnySensorActive() 子程序更新。

CSDADC_Start

说明：

初始化寄存器并启动用户模块。 此函数应当在调用任何其他用户模块函数之前调用。

C 原型：

```
void  CSDADC_Start()
```

汇编：

```
lcall  CSDADC_Start
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_Stop

说明：

停止传感器扫描仪，禁用内部中断，调用 CSDADC_ClearSensors() 以将所有传感器复位为非活动状态。

C 原型：

```
void  CSDADC_Stop()
```

汇编：

```
lcall  CSDADC_Stop
```

参数：

无

返回值：

无

副作用：

**

CSDADC_EnableADC

说明：

激活用户模块的 ADC 功能。 选择 CSDADC_ABSOLUTE 模式后，将启动单斜 ADC 模式，且需要校准 ADC。 有关详细信息，请参见 CSDADC_wCal() 例程。 如果选择 CSDADC_RATIOMETRIC，则不需要校准 CSDADC。

应当仅在调用 CSDADC_EnableADC 后再调用所有 ADC 相关的 API。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
mov A, Mode
lcall CSDADC_EnableADC
```

C 原型：

```
void  CSDADC_EnableADC (BYTE Mode);
```

参数：

模式 (Mode) - 确定 ADC 操作模式。 此参数有两个选项可供选择：

CSDADC_ABSOLUTE - 在绝对电压模式下配置 ADC；

CSDADC_RATIOMETRIC - 在比例测量模式下配置 ADC。

返回值：

无

副作用：

**

CSDADC_EnableCapsense

说明：

此子程序可启用 CapSense 操作，将硬件配置设置回 CapSense 函数。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
lcall CSDADC_EnableCapsense
```

C 原型：

```
void  CSDADC_EnableCapsense(void);
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_EnableInput

说明：

将输入端口连接到 ADC。 完成连接所使用的是模拟总线，以便所有可用端口都可以支持 ADC 输入函数。 在切换到 CSD 模式前，必须使用 CSDADC_DisableInput(). 将引脚从模拟总线断开。同样，在使用 CSDADC_EnableInput(). 设置 ADC 的另一个引脚前，也需将引脚从模拟总线断开。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
mov A, bMask
mov X, bPort
lcall CSDADC_EnableInput
```

C 语言原型：

```
void CSDADC_EnableInput(BYTE bMask, BYTE bPort);
```

参数：

bPort - 设置 ADC 输入端口

bMask - 端口位。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_DisableInput

说明：

通过断开所选端口引脚与模拟总线的连接将输入端口从 ADC 上断开。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
mov A, bMask
mov X, bPort
lcall CSDADC_DisableInput
```

C 语言原型：

```
void CSDADC_DisableInput(BYTE bMask, BYTE bPort);
```

参数：

bPort - ADC 输入端口

bMask - 端口位。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_StartADC

说明：

启动并持续运行 ADC。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
lcall CSDADC_StartADC
```

C 原型：

```
void CSDADC_StartADC(void)
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_StopADC

说明：

关闭 ADC 并立即停止转换处理。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
lcall CSDADC_StopADC
```

C 原型：

```
void CSDADC_StopADC(void)
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_fIsDataAvailable

说明：

检查 ADC 的状态。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
lcall CSDADC_fIsDataAvailable
; Return value will be in A
```

C 原型：

```
BYTE CSDADC_fIsDataAvailable(void)
```

参数：

无

返回值：

如果数据已转换并已准备读取，则返回一个非零值。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_wGetData

说明：

返回转换的数据。在验证数据采样是否就绪前，应当调用 CSDADC_fIsDataAvailable()。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
lcall CSDADC_wGetData
;Data will be in A (LSB) and X(MSB) upon return
```

C 原型：

```
WORD CSDADC_wGetData(void)
```

参数：

无

返回值：

通过 A 和 X 寄存器以无符号整数格式返回转换的数据采样。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_wGetDataClearFlag

说明：

获取数据并将数据可用标志复位。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

汇编：

```
lcall CSDADC_wGetDataClearFlag
;Data will be in A (LSB) and X(MSB) upon return
```

C 原型：

```
WORD CSDADC_wGetDataClearFlag(void)
```

参数：

无

返回值：

通过 A 和 X 寄存器以无符号整数格式返回转换的数据采样。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_wCal

说明：

通过调整 SC 模块中的电容来校准 ADC，以此来更改转换中所使用的参考斜坡的斜率。 必须将 ADC 输入设置为用户选定的固定参考电压，以进行校准。 此函数在调用 CSDADC_Start 后和调用 CSDADC_StartADC. 前调用。

仅当在用户模块参数中启用 ADC 时，此函数才可用。

C 原型：

```
int   CSDADC_wCal(WORD wVal)
```

汇编：

```
mov   A, [wVal+1]
mov   X, [wVal]
lcall  CSDADC_wCal
;Data will be in A (LSB) and X(MSB) upon return
```

参数：

wVal: 此数字是指定参考电压下的预期值，以此可设定输入值。 CSDADC_wCal() 例程会对 SC 模块中的电容进行调整，直至所得结果等于或尽可能接近 wVal。

返回值：

返回一个无符号整数，表示与可能的 wVal 最为接近的结果。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_StartTempMeasurement

Note 此 API 仅在 “Die 温度测量 ” 参数设置为 “ 启用 ” 时可用。

说明：

连接 VTEMP 传感器输出至 PMux 输入。 然后启动 ADC 测量。 如需 API 使用示例，请参见 “Die 温度测量 ” 一节。

C 原型：

```
void CSDADC_StartTempMeasurement (void)
```

汇编：

```
lcall  CSDADC_StartTempMeasurement
```

参数：

无

返回值：

无

副作用

请参见 API 章节开始部分的注释 **。

CSDADC_GetTemperature

Note 此 API 仅在 “Die 温度测量 ” 参数设置为 “ 启用 ” 时可用。

说明：

停止 ADC 测量，计算模具温度值，恢复 ADC 输入并复位数据可用标志。 CSDADC_fIsDataAvailable 必须在验证数据样本是否就绪前调用 API。 如需 API 使用示例，请参见 “Die 温度测量 ” 一节。

该 Die 温度可通过以下等式计算得出：

DieTemp = M * wADCCount * wCalibrationVoltage / wCalibrationCount - B

在此等式中：

wADCCount - ADC VTEMP 电压测量结果；

wCalibrationVoltage - API 参数；

wCalibrationCount - API 函数的 CSDADC_wCal(WORD wVal) wValAPI 参数的内部存储值；

M、B - 存储在闪存中的系数。

C 原型：

```
CHAR CSDADC_GetTemperature(WORD wCalibrationVoltage)
```

汇编：

```
mov A, [wCalibrationVoltage+1] ; LSB of API parameter
mov X, [wCalibrationVoltage] ; MSB of API parameter
lcall  CSDADC_GetTemperature
```

参数：

WORD wCalibrationVoltage: 用于 ADC 校准的校准电压 （计量单位为 mV）。 此电压对应 CSDADC_wCal API 函数的 wVal 参数 （计数）。

返回值：

“A” 包含 Die 温度 （摄氏度）。

副作用

请参见 API 章节开始部分的注释 **。

CSDADC_ScanSensor

说明：

扫描选定的电容式传感器。 每个传感器在传感器阵列中有唯一编号。 此编号由 CSDADC 向导按顺序分配。 Sw0 为传感器 0，Sw1 为传感器 1，依此类推。

C 原型：

```
void CSDADC_ScanSensor(BYTE bSensor);
```

汇编：

```
mov A,  bSensor
lcall CSDADC_ScanSensor
```

参数：

寄存器 A 在调用前包含传感器编号

返回值：

无

副作用

请参见 API 章节开始部分的注释 **。

CSDADC_ScanAllSensors

说明：

通过调用每个传感器索引的 CSDADC_ScanSensor()，扫描所有已配置的电容传感器。

C 原型：

```
void CSDADC_ScanAllSensors();
```

汇编：

```
lcall CSDADC_ScanAllSensors
```

参数：

无

返回值：

无

副作用

请参见 API 章节开始部分的注释 **。

CSDADC_UpdateSensorBaseline

说明：

针对每个电容式传感器独立计算得出的历史数值称为这个传感器的基准线。 此基准线使用 “ 桶形电极方法 ” 进行更新。

“ 桶形电极方法 ” 使用以下算法。

1. 每次调用 CSDADC_UpdateSensorBaseline() 时，通过从原始数值中减去以前的基准线来计算差值。 此差值存储在 CSDADC_waSnsDiff[] 阵列中向您提供。

2. 如果禁用传感器自动复位，则每次调用 CSDADC_UpdateSensorBaseline() 时，差值会与噪声阈值进行比较。 如果差值低于噪声阈值，将被累加到虚拟水桶中。 如果差值高于噪声阈值，则不更新水桶。 如果启用传感器自动复位，则无论噪声阈值参数如何，差值都将累加到虚拟水桶中。

3. 虚拟水桶中的累加差值达到 BaselineUpdateThreshold （基准线更新阈值）后，基准线按 1 递增，水桶复位为 0。

4. 如果差值低于噪声阈值，则保留在 waSnsDiff[] 阵列中的值复位为 0。 因此，此阵列不包含值大于 0 但低于噪声阈值的元素。

C 原型：

```
void CSDADC_UpdateSensorBaseline(BYTE bSensor)
```

汇编：

```
mov   A,   bSensor
lcall  CSDADC_UpdateSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_UpdateAllBaselines

说明：

使用 CSDADC_bUpdateSensorBaseline() 函数可更新所有电容式传感器的基准线

C 原型：

```
void  CSDADC_UpdateAllBaselines()
```

汇编：

```
lcall CSDADC_UpdateAllBaselines
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_bIsSensorActive

说明：

与手指阈值进行比较，检查给定传感器的差值阵列。 将迟滞考虑在内。 根据传感器当前是否开启，对手指阈值加减迟滞值。 如果传感器处于活动状态，则降低该阈值。 如果传感器处于非活动状态，则提高该阈值。 此函数还可更新 CSDADC_baSnsOnMask[] 阵列中传感器的位。

C 原型：

```
BYTE  CSDADC_bIsSensorActive(BYTE bSensor)
```

汇编：

```
mov  A,  bSensor
lcall CSDADC_bIsSensorActive
```

参数：

bSensor A => 传感器编号

返回值：

如果传感器处于活动状态，则返回值为 “1” ；如果传感器处于非活动状态，则返回值为 “0”。

A => “1” - 所选传感器处于活动状态 |，“0” - 所选传感器处于非活动状态。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_bIsAnySensorActive

说明：

与手指阈值进行比较，检查所有电容式传感器的差值阵列。 针对每个传感器调用 CSDADC_bIsSensorActive()，以便在调用此函数后 CSDADC_baSnsOnMask[] 阵列为最新。

C 原型：

```
BYTE  CSDADC_bIsAnySensorActive()
```

汇编：

```
lcall CSDADC_bIsAnySensorActive
```

参数：

无

返回值：

如果传感器处于活动状态，则返回值为 1；如果传感器处于非活动状态，则返回值为 0。

A => 1 - 一个或多个传感器处于活动状态，0 - 没有传感器处于活动状态。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_wGetCentroidPos

说明：

为了计算中心位置检查差值阵列。 如果存在，则偏移和长度存储在临时变量中，并且中心位置根据 CSDADC 向导中指定的分辨率进行计算。

只有滑条由 CSDADC 向导定义时，此函数才可用。

C 原型：

```
WORD  CSDADC_wGetCentroidPos(BYTE bSnsGroup)
```

汇编：

```
mov  A,  bSnsGroup
lcall CSDADC_wGetCentroidPos
```

参数：

bSnsGroup A => 组编号

此参数可引用作为滑条的一组特定的传感器。 组 0 用于按键。 滑条包含在组 1 和更高的组中。

返回值：

滑条的位置数值、A 中的 LSB 和 X 中的 MSB。

副作用：

此例程通过减去噪声阈值来修改差值。 此例程在每次扫描后只能调用一次，以避免得到负的差值。 如果应用程序监控差值信号，则在差值数据传输后调用此例程。

如果有滑条传感器处于活动状态，则函数返回的值在零到 CSDADC 向导中设置的分辨率值之间。 如果没有传感器处于活动状态，则该函数返回 -1 (FFFFh)。 如果在执行中心位置值 / 复用算法时出现错误，则该函数返回 -1 (FFFFh)。 如果需要，可以使用 CSDADC_blsSensorActive() 例程确定触摸了哪些滑条段。

如果滑条段的噪声数值大于噪声阈值，此例程可能会生成错误的中心位置结果。 噪声阈值应小心设置 （高出噪声水平足够程度），以确保噪声不会生成错误的中心位置值。

CSDADC_InitializeSensorBaseline

说明：

通过扫描选定的传感器为 CSDADC_waSnsBaseline[bSensor] 阵列加载初始值。 原始数值将复制到所选传感器的基准线阵列元素中。 此函数可用于复位单个传感器的基准线值。

C 原型：

```
void CSDADC_InitializeSensorBaseline(BYTE bSensor)
```

汇编：

```
mov A, bSensor
lcall CSDADC_InitializeSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_InitializeBaselines

说明：

通过扫描每个传感器，加载含有初始值的 CSDADC_waSnsBaseline[] 阵列。 原始数值将复制到每个传感器的基准线阵列中。

C 原型：

```
void CSDADC_InitializeBaselines()
```

汇编：

```
lcall CSDADC_InitializeBaselines
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_SetDefaultFingerThresholds

说明：

通过 “ 手指阈值 ”(FingerThreshold) 参数值加载 CSDADC_baBtnFThreshold[] 阵列。 如果 CSDADC_baBtnFThreshold[] 阵列不是通过自定义值手动加载，则必须在扫描之前调用此函数。

C 原型：

```
void CSDADC_SetDefaultFingerThresholds()
```

汇编：

```
lcall CSDADC_SetDefaultFingerThresholds
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_SetScanMode

说明：

设定电容式传感器的扫描速度和分辨率，并设定增量型模数转换器的扫描速度和分辨率。 此函数可以在运行时调用来更改扫描速度和分辨率。 此函数会覆盖用户模块参数设置。 当某些传感器需要用不同的扫描速度和分辨率进行扫描时 （例如：常用按键和接近检测器），此函数非常有效。 常用按键可以用 9-bit 分辨率和 300 µ? 扫描时间进行扫描。对于长距离的检测，接近检测器经常采用低于 16-bit 分辨率的值进行扫描，扫描时间大于 12 ms。 此函数可以与 CSDADC_ScanSensor() 函数一起使用。

C 原型：

```
void CSDADC_SetScanMode(BYTE bSpeed, BYTE bResolution);
```

汇编：

```
mov     A, bSpeed
mov     X, bResolution
lcall    CSDADC_SetScanMode
```

参数：

bSpeed: 扫描速度 (Scanning Speed)

下面给出了 bSpeed 参数的常量：

常量	值
CSDADC_ULTRA_FAST_SPEED	0x00
CSDADC_FAST_SPEED	0x01
CSDADC_NORMAL_SPEED	0x02
CSDADC_SLOW_SPEED	0x03

bResolution: 扫描分辨率。 将此值设置为所需的分辨率位数。 此参数值不得小于 9 或大于 16。
提供 bResolution 参数的以下可能常数:

常量	值
CSDADC_9_BIT_RESOLUTION	9
CSDADC_10_BIT_RESOLUTION	10
CSDADC_11_BIT_RESOLUTION	11
CSDADC_12_BIT_RESOLUTION	12
CSDADC_13_BIT_RESOLUTION	13
CSDADC_14_BIT_RESOLUTION	14
CSDADC_15_BIT_RESOLUTION	15
CSDADC_16_BIT_RESOLUTION	16

返回值:

无

副作用:

请参见 API 章节开始部分的注释 **。

CSDADC_SetRefValue

说明:

设置电容式传感器的扫描参考值。 仅当参考是从模拟调制器 (“ 参考 ”(Reference) 参数中的 ASE11)或外部滤波 PWM/PRS 信号提供时有效。 接受的值为 0..8。 值 0 对应于提供最大灵敏度的最小参考电压。 值 8 设置最大参考电压，因而灵敏度较低。 此函数可以与 CSDADC_ScanSensor(). 一起使用。

此函数在带 VC2 时钟源配置的 CSDADC 中不可用。

C 原型:

```
void CSDADC_SetRefValue(BYTE bRefValue);
```

汇编:

```
mov     A, bRefValue
lcall    CSDADC_SetRefValue
```

参数:

bRefValue - 设置扫描参考值。 接受的值为 0..8。

返回值:

无

副作用:

请参见 API 章节开始部分的注释 **。

CSDADC_ClearSensors

说明：

通过针对每个传感器按顺序调用 CSDADC_wGetPortPin() 和 CSDADC_DisableSensor()，将所有电容式传感器清除为非采样状态。

C 原型：

```
void  CSDADC_ClearSensors()
```

汇编：

```
lcall  CSDADC_ClearSensors
```

参数：

无

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_wReadSensor

说明：

返回 A (LSB) 和 X (MSB) 中的关键原始扫描值。

C 原型：

```
WORD  CSDADC_wReadSensor(BYTE bSensor)
```

汇编：

```
mov A, bSensor
lcall CSDADC_wReadSensor
```

参数：

A => 传感器编号

返回值：

传感器的扫描值、A 中的 LSB 和 X 中的 MSB。

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_wGetPortPin

说明：

返回指定传感器的端口号和引脚掩码。传递的参数对 CSDADC_Sensor_Table[]. 中的数据编制索引并进行选择。返回值可以传递给 CSDADC_EnableSensor()、CSDADC_DisableSensor().。

C 原型：

```
WORD  CSDADC_wGetPortPin(BYTE bSensorNum)
```

汇编：

```
mov  A,  bSensorNumber
```

```
lcall     CSDADC_wGetPortPin
;Data will be in A (LSB, Sensor Bitmap) and X(MSB, Port Number) upon return
```

参数：

bSensorNumber - 范围为 0 到 (n - 1)，其中 n 是 CSDADC 向导中设置的传感器总数与滑条中包括的传感器数量之和。 CSDADC_wGetPortPin() 使用传感器编号来确定所选活动传感器的端口和位掩码。

返回值：

A => 传感器位图

X => Port 编号

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_EnableSensor

说明：

配置所选传感器以便在下一测量周期中进行测量。 可以使用 CSDADC_wGetPortPin() 函数选择端口和传感器，端口号和传感器位掩码分别加载到 X 和 A 中。 驱动模式会修改以便将选中的端口和引脚设定为模拟 High Z 模式，并启用正确的模拟复用器总线输入。 这还可以启用比较器功能。

C 原型：

```
void  CSDADC_EnableSensor(BYTE bMask, BYTE bPort)
```

汇编：

```
mov  X,  bPort
mov  A,  bMask
lcall  CSDADC_EnableSensor
```

参数：

A => 传感器位图

X => 端口编号

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

CSDADC_DisableSensor

说明：

禁用 CSDADC_wGetPortPin() 函数选择的传感器。 驱动模式更改为 “ 强 (001)” 。 这可以将传感器有效接地。 端口引脚与 “ 模拟复用器总线 ”(AnalogMuxBus) 的连接关闭。 函数参数由 CSDADC_wGetPortPin() 函数返回。

C 原型：

```
void  CSDADC_DisableSensor(BYTE bMask, BYTE bPort)
```

汇编：

```
mov  X,  bPort
```

```
mov  A,  bMask
lcall  CSDADC_DisableSensor
```

参数：

A => 传感器位图

X => 端口编号

返回值：

无

副作用：

请参见 API 章节开始部分的注释 **。

Die 温度测量

Die 温度仅可在 “Die Temp Measurement” 参数设置为 “ 已启用 ” 时测量。

代码示例：

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

void main(void)
{
    CHAR cTemp;
    WORD wVoltage;

    M8C_EnableGInt;
    CSDADC_Start(); // Start the User Module
    CSDADC_EnableADC (CSDADC_ABSOLUTE);

// Calibrate the user module using the 1.3 V internal BandGap reference source
ACE01CR1 &= ~0x04;           // Connect Vref to PMux input of ACE01 Analog Block
    CSDADC_wCal(0x428);        // Calibrate ADC
    ACE01CR1 |= 0x04;               // Restore PMux connection to Analog MUX Bus

    while (1)
    {
        CSDADC_EnableInput(0x01, 0x00); // use P0[0]
        CSDADC_StartADC();
        while (!CSDADC_fIsDataAvailable());
        wVoltage = CSDADC_wGetDataClearFlag();
        CSDADC_StopADC();
        CSDADC_DisableInput(0x01, 0x00); // required for normal CSD operation

CSDADC_StartTempMeasurement();
        while (!CSDADC_fIsDataAvailable());
        cTemp = CSDADC_GetTemperature(1300);

/* Add user code here to display the results of
        voltage and temperature measurement */
    }
}
```

Note

1. 调用 CSDADC_StartTempMeasurement API 前必须校准 ADC。
2. CSDADC_StartTempMeasurement API 调用 CSDADC_StartADC API；作为一个结果，没有必要在开始温度测量前调用 API。
3. CSDADC_GetTemperature API 停止 ADC 的转换。 如果需要在 Die 温度测量后测量电压，必须调用 CSDADC_StartADC API。

固件源代码示例

此代码启动用户模块并连续扫描传感器。

因为支持比例和绝对输入 ADC 模式，以及运行时可能的模式转换，可轻松与不同类型的传感器交互。

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

//#define ACD_KIND  CSDADC_RATIOMETRIC
#define ACD_KIND  CSDADC_ABSOLUTE

WORD wCal;
WORD wResult;

void main(void)
{
  M8C_EnableGInt;
  CSDADC_Start();
  CSDADC_SetDefaultFingerThresholds();
CSDADC_InitializeBaselines();

#if (ACD_KIND==CSDADC_ABSOLUTE)
  CSDADC_EnableADC(ACD_KIND);
  wCal = CSDADC_wCal(1000);
#endif

  while (1) {
    CSDADC_EnableCapsense();
    CSDADC_ScanAllSensors();
    CSDADC_UpdateAllBaselines();

CSDADC_EnableADC(ACD_KIND);
    CSDADC_EnableInput(0x01, 0x02); // use P2[0]

CSDADC_StartADC();

while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();

CSDADC_StopADC();

CSDADC_DisableInput(0x01, 0x02); // required for normal CSD operation
  }
}
```

同一工程用汇编语言表示为：

```
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

; ACD_KIND: EQU  CSDADC_RATIOMETRIC
  ACD_KIND: EQU  CSDADC_ABSOLUTE

_main:

    M8C_EnableGInt

lcall  CSDADC_Start
lcall  CSDADC_SetDefaultFingerThresholds
    lcall  CSDADC_InitializeBaselines

IF (ACD_KIND & CSDADC_ABSOLUTE)
    mov    A, CSDADC_ABSOLUTE
    lcall  CSDADC_EnableADC

mov    A, <1000
    mov    X, >1000
    lcall  CSDADC_wCal
    ;calibration data are located in A (LSB) and X(MSB)
ENDIF

loop:
    lcall  CSDADC_EnableCapsense
    lcall  CSDADC_ScanAllSensors
    lcall  CSDADC_UpdateAllBaselines

mov    A, ACD_KIND
    lcall  CSDADC_EnableADC

mov    A, 0x01
    mov    X, 0x02
lcall  CSDADC_EnableInput  ; use P2[0]

lcall  CSDADC_StartADC

.scan:
lcall  CSDADC_fIsDataAvailable
    cmp    A, 0
    jz     .scan
    lcall  CSDADC_wGetDataClearFlag

; The ADC rezult is stored in A (LSB) and X(MSB)

lcall  CSDADC_StopADC

    mov    A, 0x01
    mov    X, 0x02
    lcall  CSDADC_DisableInput  ; required for normal CSD operation
```

```
    jmp     loop
```

配置寄存器

带 PRS16 时钟源配置寄存器的 CSDADC

Table 12. 模块 CMP，寄存器： ACE_CONTROL1 (ACE01CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	1	参考			1	1	1

参考值与参考参数保持一致。

Table 13. 模块 CMP，寄存器： ACE_CONTROL2 (ACE01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	功耗	

电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 14. 模块 CMP_REF，寄存器： ASE_CONTROL (ASE11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 15. 模块 CMP_REF，寄存器： ADC_CONTROL (ADC1_CR)，组 0

位	7	6	5	4	3	2	1	0
值	CMPST	1	1	0	0	自动	0	ADCEN

CMPST 是只读模式，供与 ADC 相关的 API 使用。 自动位由与 ADC 相关的 API 进行维护。 ADCEN 启用 ADC 操作，并由 CSDADC API 进行维护。

Table 16. 模块 CMP_REF，寄存器： ADC_TRIM (ADC1_TR)，组 1

位	7	6	5	4	3	2	1	0
值	ADC 预设值 (ADCTrimValue)							

“ADC 预设值 ”(ADCTrimValue) 由 CSDADC API 控制。 包括针对单斜 ADC 的 ADC 预设值和针对 CSD 操作的零值。

Table 17. 模块 CNT，寄存器： Function (DxBxxFN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 18. 模块 CNT，寄存器： Input (DxBxxIN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 19. 模块 CNT，寄存器： Output (DxBxxOU)，组 1

位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	0	0	0

Table 20. 模块 CNT，寄存器： Control (DxBxxCR0)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

启用位启用计数器操作，并由 CSDADC API 进行维护。

Table 21. 模块 CNT，寄存器： 周期 (DxBxxDR0)，组 0

位	7	6	5	4	3	2	1	0
值	“计数器值”(CounterValue) 无法直接读写							

Table 22. 模块 CNT，寄存器： Period (DxBxxDR1)，组 0

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 23. 模块 CNT，寄存器： Compare (DxBxxDR2)，组 0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 24. 模块 PRS，寄存器： Function (DxBxxFN)，组 1

位	7	6	5	4	3	2	1	0
LSB	0	0	0	0	1	0	1	0
MSB	0	1	1	0	1	0	1	0

Table 25. 模块 PRS，寄存器： Input (DxBxxIN)，组 1

位	7	6	5	4	3	2	1	0
LSB	0	0	0	0	0	0	0	0
MSB	0	0	1	1	0	0	0	0

Table 26. 模块 PRS，寄存器： Output (DxBxxOU)，组 1

位	7	6	5	4	3	2	1	0
LSB	1	1	0	0	0	0	0	0
MSB	1	1	1	0	0	屏蔽电极输出 (ShieldElectrodeOut)		

ShieldElectrodeOut 可将屏蔽电极的输出信号启用位行输出。 这由具有相同名称的用户模块参数控制。

Table 27. 模块 PRS，寄存器： Control (DxBxxCR0)，组 1

位	7	6	5	4	3	2	1	0
LSB	0	0	0	0	0	0	0	启用
MSB	0	0	0	0	0	0	0	0

启用位可启用 PRS 模块，由 CSDADC API 保持

Table 28. 模块 PRS，寄存器： Shift (DxBxxDR0)，组 0

位	7	6	5	4	3	2	1	0
LSB	PRS 移位寄存器 (LSB) - 无直接访问							
MSB	PRS 移位寄存器 (MSB) - 无直接访问							

Table 29. 模块 PRS，寄存器： Polynomial (DxBxxDR1)，组 0

位	7	6	5	4	3	2	1	0
LSB	PRS 多项式 (LSB)							
MSB	PRS 多项式 (MSB)							

PRS 多项式由 CSDADC API 根据 ScanSpeed 和分辨率参数保持。

Table 30. 模块 PRS，寄存器： Seed (DxBxxDR2)，组 0

位	7	6	5	4	3	2	1	0
LSB	PRS 种子 / 比较寄存器 (LSB)							
MSB	PRS 种子 / 比较寄存器 (MSB)							

此寄存器的值由 API 根据几乎所有的参数值保持。

带有 PWM8 时钟源配置寄存器的 CSDADC

Table 31. 模块 CMP，寄存器： ACE_CONTROL1 (ACE01CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	1	参考			1	1	1

由同一名称的用户模块参数对参考值进行维护。

Table 32. 模块 CMP，寄存器： ACE_CONTROL2 (ACE01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	功耗	

电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 33. 模块 CMP_REF，寄存器： ASE_CONTROL (ASE11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 34. 模块 CMP_REF，寄存器： ADC_CONTROL (ADC1_CR)，组 0

位	7	6	5	4	3	2	1	0
值	CMPST	1	1	0	0	自动	0	ADCEN

CMPST 是只读模式，供与 ADC 相关的 API 使用。 自动位由与 ADC 相关的 API 进行维护。 ADCEN 使能 ADC 操作，并由 CSDADC API 进行维护。

Table 35. 模块 CMP_REF，寄存器： ADC_TRIM (ADC1_TR)，组 1

位	7	6	5	4	3	2	1	0
值	ADC 预设值 (ADCTrimValue)							

“ADC 预设值 ”(ADCTrimValue) 由 CSDADC API 控制。 包括针对单斜 ADC 的 ADC 预设值和针对 CSD 操作的零值。

Table 36. 模块 CNT，寄存器： Function (DxBxxFN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 37. 模块 CNT，寄存器： Input (DxBxxIN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 38. 模块 CNT，寄存器： Output (DxBxxOU)，组 1

位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	0	0	0

Table 39. 模块 CNT，寄存器： Control (DxBxxCR0)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

“Enable” 位启用计数器操作，并由 CSDADC API 维护。

Table 40. 模块 CNT，寄存器： 周期 (DxBxxDR0)，组 0

位	7	6	5	4	3	2	1	0
值	“ 计数器值 ”(CounterValue) 无法直接读写							

Table 41. 模块 CNT，寄存器： Period (DxBxxDR1)，组 0

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 42. 模块 CNT，寄存器： Compare (DxBxxDR2)，组 0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 43. 模块 PWM，寄存器： Function (DxBxxFN)，组 1

位	7	6	5	4	3	2	1	0
值	0	1	1	0	0	0	0	1

Table 44. 模块 PWM，寄存器： Input (DxBxxIN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	1	0	0	0	0

Table 45. 模块 PWM，寄存器： Output (DxBxxOU)，组 1

位	7	6	5	4	3	2	1	0
值	1	1	0	0	0	1	0	0

Table 46. 模块 PWM，寄存器： Control (DxBxxCR0)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

启用位可启用 PRS 模块，由 CSDADC API 保持

Table 47. 模块 PWM，寄存器： Count (DxBxxDR0)，组 0

位	7	6	5	4	3	2	1	0
值	计数寄存器 - 无直接访问							

Table 48. 模块 PWM，寄存器： Period (DxBxxDR1)，组 0

位	7	6	5	4	3	2	1	0
值	PrescalerPeriod							

PrescalerPeriod 由具有相同名称的用户模块参数控制。

Table 49. 模块 PWM，寄存器： Compare (DxBxxDR2)，组 0

位	7	6	5	4	3	2	1	0
值	比较							

由 API 根据参考用户模块参数保持该寄存器的值。

带 PRS8 时钟源配置寄存器的 CSDADC

Table 50. 模块 CMP，寄存器： ACE_CONTROL1 (ACE01CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	1	参考			1	1	1

由同一名称的用户模块参数对参考值进行维护。

Table 51. 模块 CMP，寄存器： ACE_CONTROL2 (ACE01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	功耗	

电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 52. 模块 CMP_REF，寄存器： ASE_CONTROL (ASE11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 53. 模块 CMP_REF，寄存器： ADC_CONTROL (ADC1_CR)，组 0

位	7	6	5	4	3	2	1	0
值	CMPST	1	1	0	0	自动	0	ADCEN

CMPST 是只读模式，供与 ADC 相关的 API 使用。 自动位由与 ADC 相关的 API 进行维护。 ADCEN 使能 ADC 操作，并由 CSDADC API 进行维护。

Table 54. 模块 CMP_REF，寄存器： ADC_TRIM (ADC1_TR)，组 1

位	7	6	5	4	3	2	1	0
值	ADC 预设值 (ADCTrimValue)							

“ADC 预设值 ”(ADCTrimValue) 由 CSDADC API 控制。 包括针对单斜 ADC 的 ADC 预设值和针对 CSD 操作的零值。

Table 55. 模块 CNT，寄存器： Function (DxBxxFN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 56. 模块 CNT，寄存器： Input (DxBxxIN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 57. 模块 CNT，寄存器： Output (DxBxxOU)，组 1

位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	0	0	0

Table 58. 模块 CNT，寄存器： Control (DxBxxCR0)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

“Enable” 位启用计数器操作，并由 CSDADC API 维护。

Table 59. 模块 CNT，寄存器： 周期 (DxBxxDR0)，组 0

位	7	6	5	4	3	2	1	0
值	“ 计数器值 ”(CounterValue) 无法直接读写							

Table 60. 模块 CNT，寄存器： Period (DxBxxDR1)，组 0

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 61. 模块 CNT，寄存器： Compare (DxBxxDR2)，组 0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 62. 模块 PRS，寄存器： Function (DxBxxFN)，组 1

位	7	6	5	4	3	2	1	0
值	0	1	1	0	1	0	1	0

Table 63. 模块 PRS，寄存器： Input (DxBxxIN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 64. 模块 PRS，寄存器： Output (DxBxxOU)，组 1

位	7	6	5	4	3	2	1	0
值	1	1	1	0	0	屏蔽电极输出 (ShieldElectrodeOut)		

ShieldElectrodeOut 可将屏蔽电极的输出信号启用位行输出。 这由具有相同名称的用户模块参数控制。

Table 65. 模块 PRS，寄存器： Control (DxBxxCR0)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

启用位可启用 PRS 模块，由 CSDADC API 保持。

Table 66. 模块 PRS，寄存器： Shift (DxBxxDR0)，组 0

位	7	6	5	4	3	2	1	0
值	PRS 移位寄存器 - 无直接访问							

Table 67. 模块 PRS，寄存器： Polynomial (DxBxxDR1)，组 0

位	7	6	5	4	3	2	1	0
值	PRS 多项式							

PRS 多项式由 CSDADC API 根据 ScanSpeed 和分辨率参数保持。

Table 68. 模块 PRS，寄存器： Seed (DxBxxDR2)，组 0

位	7	6	5	4	3	2	1	0
值	PRS 种子 / 比较寄存器							

此寄存器的值由 API 根据几乎所有的参数值保持。

带 VC2 时钟源配置寄存器的 CSDADC。

Table 69. 模块 CMP，寄存器： ACE_CONTROL1 (ACE01CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	1	参考			1	1	1

由同一名称的用户模块参数对参考值进行维护。

Table 70. 模块 CMP，寄存器： ACE_CONTROL2 (ACE01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	功耗	

电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 71. 模块 CMP_REF，寄存器： ASE_CONTROL (ASE11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 72. 模块 CMP_REF，寄存器： ADC_CONTROL (ADC1_CR)，组 0

位	7	6	5	4	3	2	1	0
值	CMPST	1	1	0	0	自动	0	ADCEN

CMPST 是只读模式，供与 ADC 相关的 API 使用。 自动位由与 ADC 相关的 API 进行维护。 ADCEN 启用 ADC 操作，并由 CSDADC API 进行维护。

Table 73. 模块 CMP_REF，寄存器： ADC_TRIM (ADC1_TR)，组 1

位	7	6	5	4	3	2	1	0
值	ADC 预设值 (ADCTrimValue)							

“ADC 预设值”(ADCTrimValue) 由 CSDADC API 控制。 包括针对单斜 ADC 的 ADC 预设值和针对 CSD 操作的零值。

Table 74. 模块 CNT，寄存器： Function (DxBxxFN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 75. 模块 CNT，寄存器： Input (DxBxxIN)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 76. 模块 CNT，寄存器： Output (DxBxxOU)，组 1

位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	0	0	0

Table 77. 模块 CNT，寄存器： Control (DxBxxCR0)，组 1

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

“使能”位启用计数器操作，并由 CSDADC API 维护。

Table 78. 模块 CNT，寄存器： 周期 (DxBxxDR0)，组 0

位	7	6	5	4	3	2	1	0
值	“计数器值”(CounterValue) 无法直接读写							

Table 79. 模块 CNT，寄存器： Period (DxBxxDR1)，组 0

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 80. 模块 CNT，寄存器： Compare (DxBxxDR2)，组 0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

版本历史记录

版本	创作者	说明
1.1	DHA	更新了错误消息。
1.20	DHA	1. 将 DisableInt 宏调用从轮循移动到 ISR （PRS16 CSD 模式）中。 2. 模拟比较器列 (AnalogComparatorColumn) 和全局输出 (GlobalOutput) 线路之间的连接显示在互连视图中。
1.20.b	DHA	在向导中添加了帮助文件。
1.30	DHA	1. 将 DiplexTable 从 “AREA UserModules” 传输到了 “AREA lit”。 2. 将默认的 “DiplexTable” 参数值设置为了 0x0112。 3. 添加了 “DiplexUsed” 参数以提高代码压缩。
1.40	DHA	1. 更新了区域声明以支持 Imagecraft 优化。 2. 为此用户模块数据手册中的分辨率参数添加了符号名。 3. 添加了 Die 温度测量功能说明，并更新了 SetScanMode() API 功能说明。 4. 更新了 CSDADC 用户模块以实现 CSDADC_StartTempMeasurement 和 CSDADC_GetTemperature 功能。 5. 更新了用户模块向导中的滑条辐射状滑条分辨率范围内计算。 6. 更新了用户模块向导帮助。 添加了一个滑条分辨率参数最大 / 最小值说明。

Note PSoC Designer 5.1 在所有用户模块数据手册中都引入了 “ 版本历史 ”。 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 © 2011-2012 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.