CapSense® Sigma-Delta 调制和 ADC 数据手册 CSDADCV 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	
CY8C24x94、CY8CLED0xD、CY8CLED0xG、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	1	2	1257	34	2-5
带有 1 个传感器、PRS8-based 的用户模块	1	1	2	1185	32	2-5
有 1 个传感器的预分频器时钟源	1	1	2	1185	32	2-5
有 1 个传感器的 VC2 时钟源用户模块	0	1	2	1170	1170	1
每个附加 CapSense® 按钮	–	–	–	2	10	1
使用带有五个元件的电容式滑条时，静态代码和 RAM 增加	–	–	–	1197	79	5
每个附加滑条元件	–	–	–	2	10	1
当使用滑条复用方式时，静态代码和 RAM 增加	–	–	–	0	滑条 *2	–

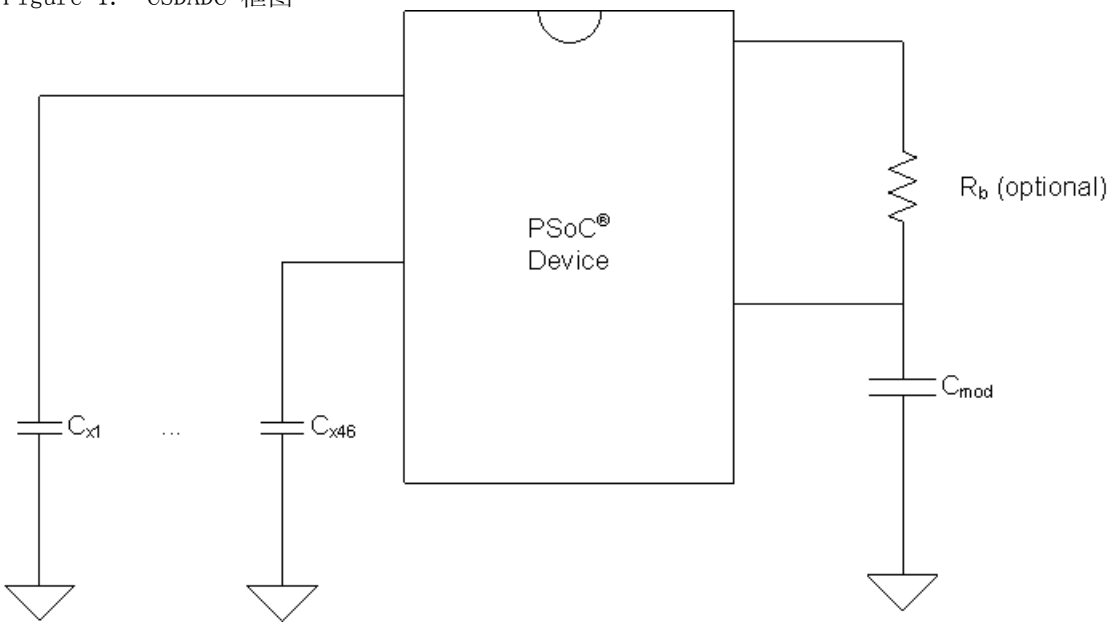
功能和概述

- 允许扫描 CapSense 传感器与测量输入电压，无需使用不同的可加载配置
- 完全使用硬件方式实现 $Sinc^N$ 滤波器从而减少 CPU 开销和降低防锯齿要求
- 支持不使用数字模块的配置
- ADC 特性：
  - 有二阶调制器的 Sigma-delta ADC
  - 数据为无符号或有符号 2 的补码格式
  - 分辨率动态更改为 10,12, 到 14 比特
  - 在 10-bit 分辨率情况下最大采样率为 31250 sps，在 14-bit 分辨率情况下最大采样率为 7812 sps
  - 内部和外部参考选项所定义的输入范围
  - 具有可配置增益和参考设置的内置可编程增益放大器
- CapSense 特性：

- 基于强大的 CSD 方法
- 二阶调制器可提供较好的信噪比性能。
- 可扫描 1 到 46 个电容式传感器
- 可感应厚达 25 mm 的玻璃外覆层
- 使用有线传感器的接近检测范围可达 30 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
- 支持独立和滑条电容式传感器的不同组合
- 可采用复用方式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
- 可通过两组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 向导设置传感器和分配引脚
- 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
- PC GUI 应用支持原始数据实时监控和参数优化。

CSDADC 模块利用开关电容技术和一个 sigma-delta 的调制器来检测电容量，其中调制器可以把检测到的开关电容的电流转换成数字代码。

Figure 1. CSDADC 框图



快速启动

1. 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. 单击右键 CSDADC 用户模块以访问 CSDADC 向导。
4. 设置传感器数量、配置和引脚分配。
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 级别转换器或者 I2C-USB 桥接器连接到目标板，并使用 GUI 优化参数。
9. 更改 CSDADC 参数并重新编译整个工程。
10. 为 PSoC 器件编程并验证模块操作。调整 CSD 参数以满足 5:1 的 SNR 要求，如 CapSense 应用的信噪比要求 – 中所述 CY8C21x34/B CapSense 设计指南 .

功能描述

CSDADC 包含了电容电测和 ADC 的功能，ADC 本身就可以测量电压，并且不需要调用另外的配置。 它通过重复使用 CSD 和 ADC 模块中的共同部分节省代码空间。 如果**同时**需要电容感应和 ADC 功能，则必须使用 CSDADC。 需要其中一种或者另一种功能的应用则必须使用 CSD 或者 DELSIG ADC。

CSDADC 利用开关电容技术和一个 sigma-delta 调试器 (CSD) 完成电容检测，其中调制器可以把检测到的开关电容的电流转换成数字代码。 CSDADC 通过二阶调制器和内置预放大器实现 sigma-delta ADC。 使用二阶调制器可产生更好的信噪比。 本用户模块数据手册提供了有关 CSD 和 ADC 操作的基本信息。 有关详细的信息、设置 CSD 参数的建议、CSD 使用提示以及 CSD 疑难解答，请参阅 CSD、DELSIG 和 PGA 数据手册及相关应用笔记。 如果之前从未使用过 CSD 和 ADC 用户模块，则必须在阅读过这些文档后再尝试在实际项目中使用这两种用户模块。

首次使用 CSDADC 用户模块之前，建议阅读下列文档：

- *CY8C24x94 Series PSoC Programmable System-on-Chip Technical Reference Manual*（系列 PSoC 可编程片上系统技术参考手册），章节：
  - 两列限制模拟
  - 数字时钟
  - IO 模拟复用器
- *了解开关电容模拟模块* – AN2041

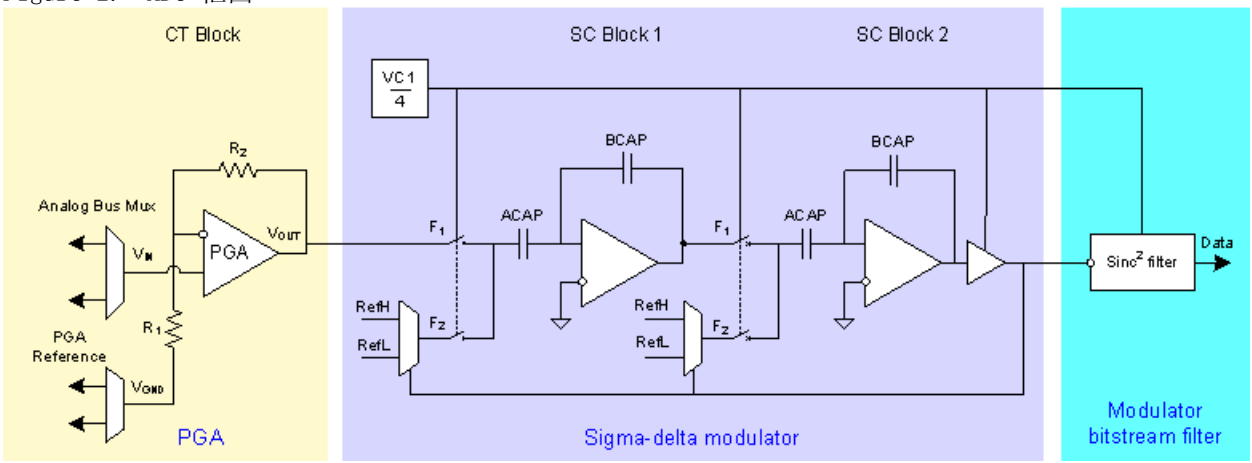
建议在阅读 CSDADC 用户模块数据手册后阅读以下设计指南。 这些文件在赛普拉斯网站上提供 www.cypress.com:

- 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

CSDADC 使用有内置 PGA 的 DELSIG ADC。 DELSIG 用户模块是积分转换器，需要 64 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前两个采样失效。 DELSIG ADC 由三个主要模块组成：PGA、二阶调制器和 $Sinc^2$ 抽取滤波器。

Figure 2. ADC 框图



可编程增益放大器

可编程增益放大器 (PGA) 先于 ADC。 此放大器允许将输入信号范围和 ADC 信号范围进行匹配，以提供更好的 ADC 动态范围利用率。 PGA 输入来自模拟总线，这使其可以与外部源进行连接。 它也可以进行重新配置以支持内部信号源。

PGA 输入信号的参考源可以是内部模拟接地、V_{SS} 或者选定的参考。 可编程增益放大器的增益的设置方法为，对电阻阵列中的可选抽头和持续时间模拟 PSoC 模块的反馈抽头进行编程。 增益和参考可在器件编辑器中设置。 放大器有以下传输函数：

Equation 1

$$V_{OUT} = (V_{IN} - V_{GND}) \cdot \left(1 + \frac{R_2}{R_1}\right) + V_{GND}$$

您可以将参考指定为下列中的一项：

- 派生于内部参考的固定值
- 与供电电压成比例的值
- 模拟接地
- 外部输入

放大器的输入和输出电压范围不会扩展到供电电源，即它们不是 “ 轨至轨 ” 运算放大器。 允许的输入范围以下内容的组合：

- 输入限制
- 输出限制
- 电源电压
- 模拟接地值
- 选定增益

调制器

调制器是 1-bit 过采样电路，它以所产生的 1 和 0 的密度形式表示输入电压。 调制器输出由低通抽取滤波器降低到最终采样率，该低通抽取滤波器会将多个 1-bit 样本转换为具有较高分辨率的样本。 通常，抽取速率越高 （即过采样率越高），则产生的分辨率结果越高，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 调制器会将某些量化噪声从较低频率转换为较高频率，之后这些频率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。 在最高抽取速率为 256X 时，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。 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，使第二个模块的 1-bit 比较器输出反馈回这两个模块 （如之前所示）。

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叠加放置。

DelSig ADC 的范围通过 $\pm V_{Ref}$ 建立。 必须在 PSoC Designer™ 的 “ 全局资源 ” 窗口中设置 V_{Ref}。 对于固定量程而言，V_{Ref} 设置为 $\pm V_{Bandgap}$ 或者 $\pm 1.6\ V_{Bandgap}$。 对于可调量程而言，V_{Ref} 设置为 ±Port 2[6]。 对于提供比例式量程而言，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

参考复用器设置	V_{DD} = 5 V	V_{DD} = 3.3 V
($V_{DD}/2$) ± 带隙	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
($V_{DD}/2$) ± ($V_{DD}/2$)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
带隙 ± 带隙	$0 < V_{in} < 2.6$	$0 < V_{in} < 2.6$
(1.6* 带隙) ± (1.6* 带隙)	$0 < V_{in} < 4.16$	NA
(2* 带隙) ± 带隙	$1.3 < V_{in} < 3.9$	NA
(2* 带隙) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	NA
P2[4] ± 带隙	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
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：

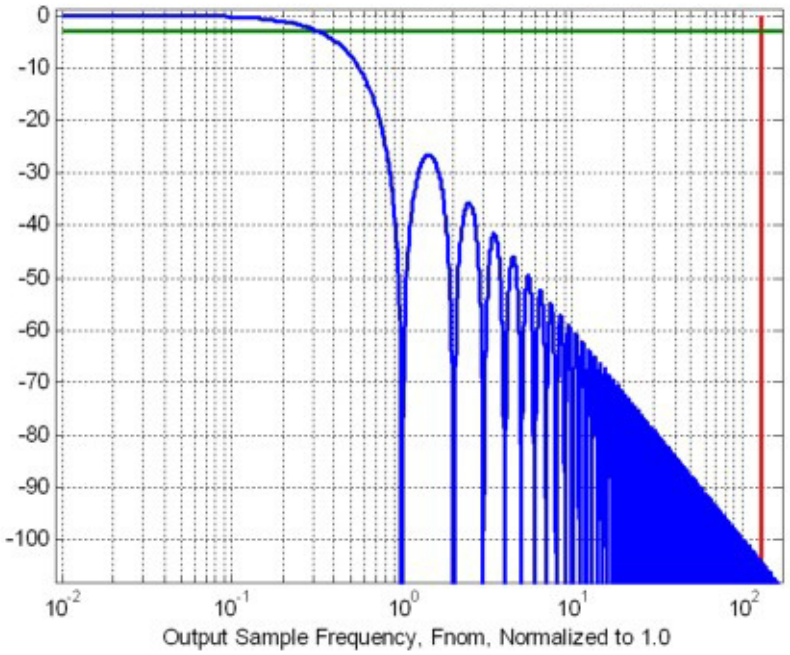
Equation 2

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

图 3 中绘制的频率域传输函数将频率标准化，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} 的每个整数倍数处。 由于 1-bit 采样率比额定输出速率高 64

到 256，奈奎斯特限制比 F_{nom} 高 5 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，抽取速率为 256 的 1-bit 奈奎斯特频率以粗垂直线显示。 虽然有可能实现较高抽取速率，但是由于存在器件本底噪声，它们几乎不会产生的影响。 对于 12-bit 拓扑，即抽取速率为 256 的二阶调制器，分辨率受信噪比限制。

Figure 3. 带 -3dB 点和奈奎斯特频率的 $Sinc^2$ 抽取滤波器量级响应



抽取滤波器以自我计数模式操作，不使用额外的定时器模块来形成抽取速率。 抽取滤波器通过以 1-bit 采样率运行的双积分器实现传输函数的分母。 分子由以额定输出采样率运行的双微分器 （第二差分运算符）实现。

电容测量操作

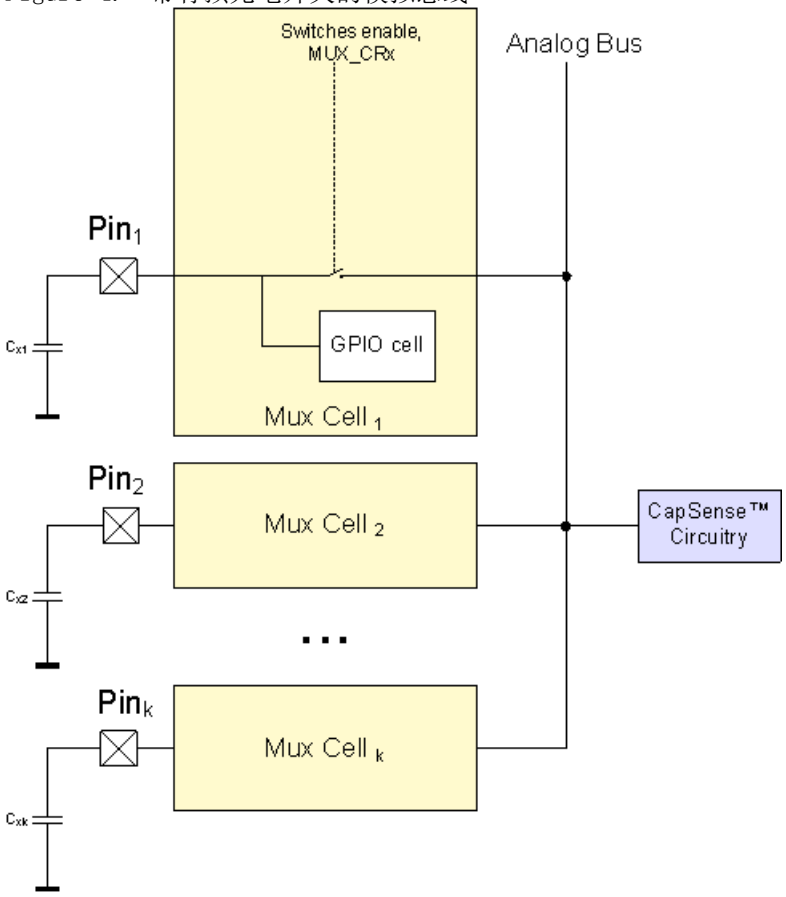
决策逻辑通过固件实现。 通过固件分析电容的测量，跟踪环境因素造成的缓慢电容变化，运行决策逻辑，以检测按键触摸变化并计算滑条位置。

扫描传感器阵列

CY8C24x94 系列设备具有两个内置模拟总线。 这两个模拟总线连接在一起，以能够扫描连接到所有引脚的传感器。 CSDADC 用户模块使用内部预充电开关在时钟信号相位 Ph_1 处为活动传感器充电，并在相位 Ph_2 处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。

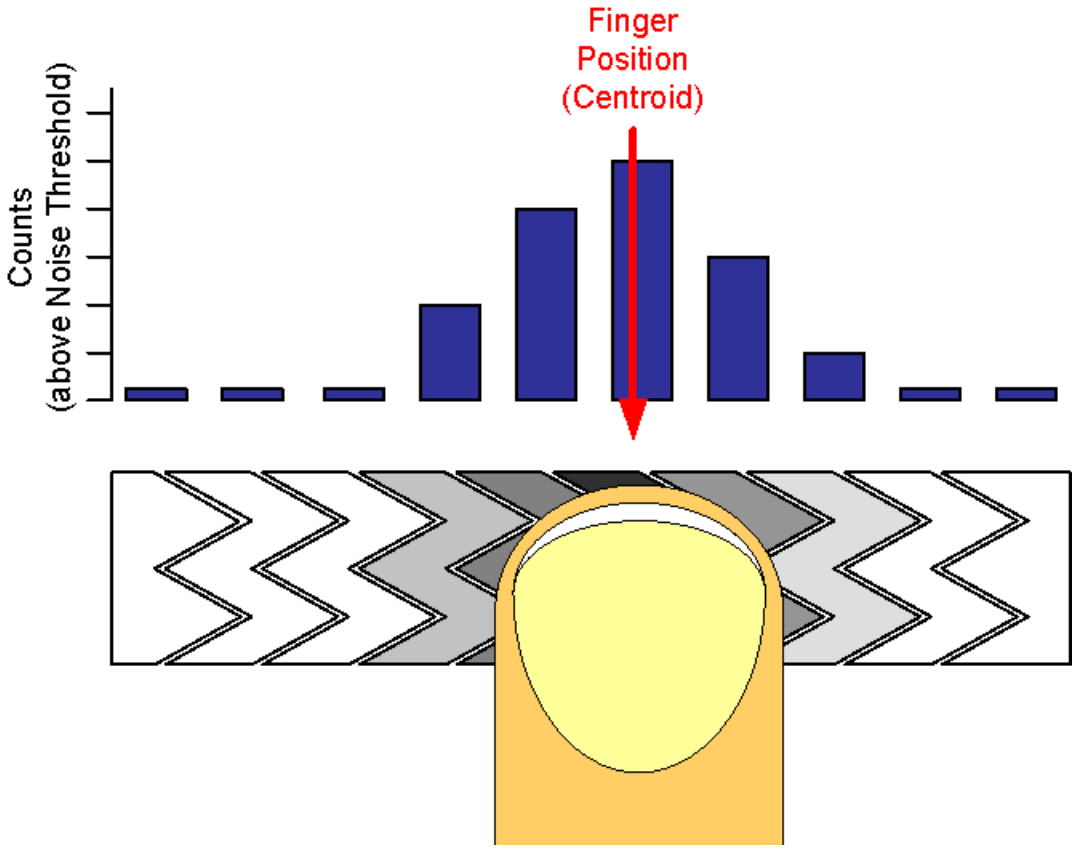
Figure 4. 带有预充电开关的模拟总线



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。 滑条可以使用 CSDADC 向导建立，方法是建立若干组，每个滑条组有特定的次序。 传感器滑条数量的实际下限值是五，上限值仅取决于所选 PSoC 器件提供的传感器位置的数值。

Figure 5. 对物理传感器位置排序



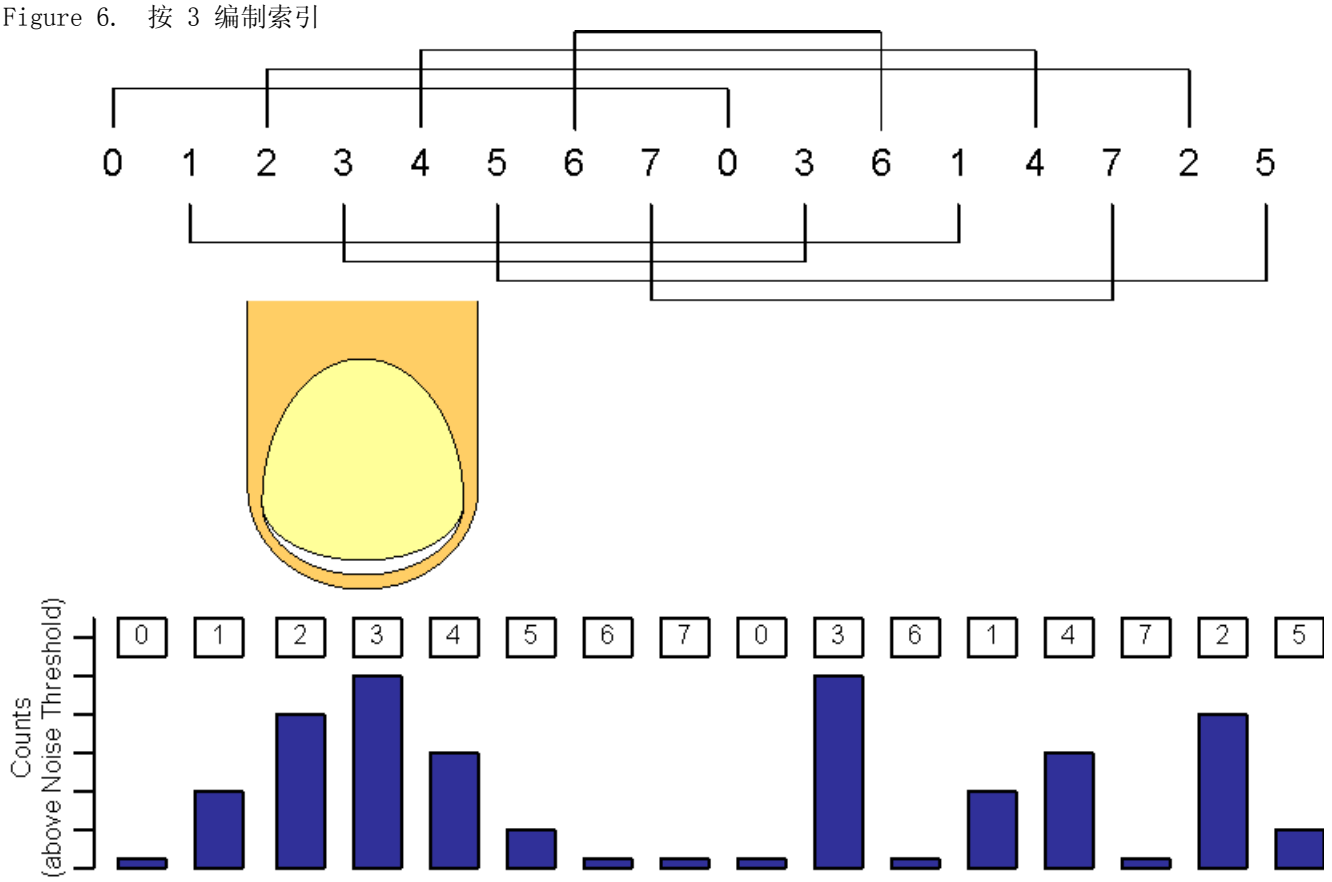
靠近滑条一半产生的强信号，将导致滑条另一半产生相同程度的伪信号，但扫描的结果是发散的。 感应算法搜索最强的一组相邻的信号，以确定滑条的位置。

复用

滑条中的每个 PSoC 传感器连接并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）按顺序映射到基本的传感器上，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 6. 按 3 编制索引



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。 当选择双工时，CSDADC 向导将自动生成传感器复用索引表。 此表说明了不同滑条段计数的复用顺序：

Table 2. 不同滑条段计数的复用顺序

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
58	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, 28, 0, 3, 6, 9, 12, 15, 18, 21, 2, 27, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 2, 5, 8, 11, 14, 17, 20, 23, 26
60	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, 28, 29, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29
62	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, 28, 29, 30, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29
64	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, 28, 29, 30, 31, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29
66	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, 28, 29, 30, 31, 32, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32
68	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, 28, 29, 30, 31, 32, 33, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32

滑条段总数	段顺序
70	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, 28, 29, 30, 31, 32, 33, 34, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32
72	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, 28, 29, 30, 31, 32, 33, 34, 35, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35
74	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35
76	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35
78	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38
80	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38
82	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38
84	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41
86	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41
88	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41
90	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44
92	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, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44

复用滑条的滑条段选择指南

选择滑条所需段数主要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。

在滑条复用设计中，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。 被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。

要使用中心算法计算出内插后的位置，首先要扫描阵列，以确定所给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，将使用此信号以及和它相邻的那些大于噪声阈值的信号计算中心位置。 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：

Equation 3

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

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一个计算出的系数。 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。 这是通过高级 API 实现的。

滑条传感器数量和分辨率在 CSDADC 向导中进行设置。 向导将计算出（换算的）刻度值并以分数值的形式进行存储。

中心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：

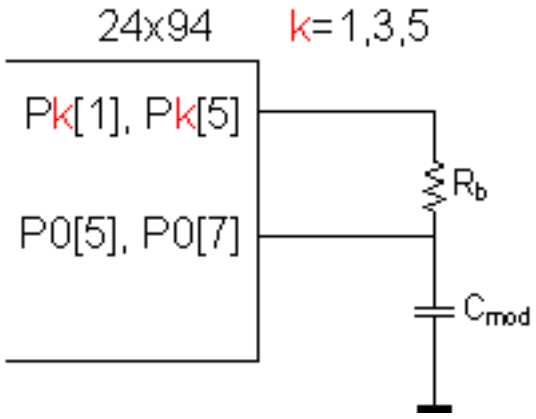
分辨率 = （传感器数 - 1）x 乘数

中心位置值以 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[5]、P0[7] 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]、P3[5]、P5[1]、P5[5] 和电容引脚。 通过用户模块参数设置可以选择引脚。不要将选定用于调制器组件连接的引脚用于其他任何目的。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 7. 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，5.6 – 10 nF 的电容值能产生良好效果。 **注**：如果已使用带有预分频器的配置，则需要更大的调制器电容。 大多数情况下，100 nF 的优质电容已足够。 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。 必须使用陶瓷电容。 温度电容系数并不重要。 电阻值取决于总传感器电容 C_S。 电阻值必须通过以下方法选择：

- 监控不同传感器触摸的原始数值。
- 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，原始数值会增加。

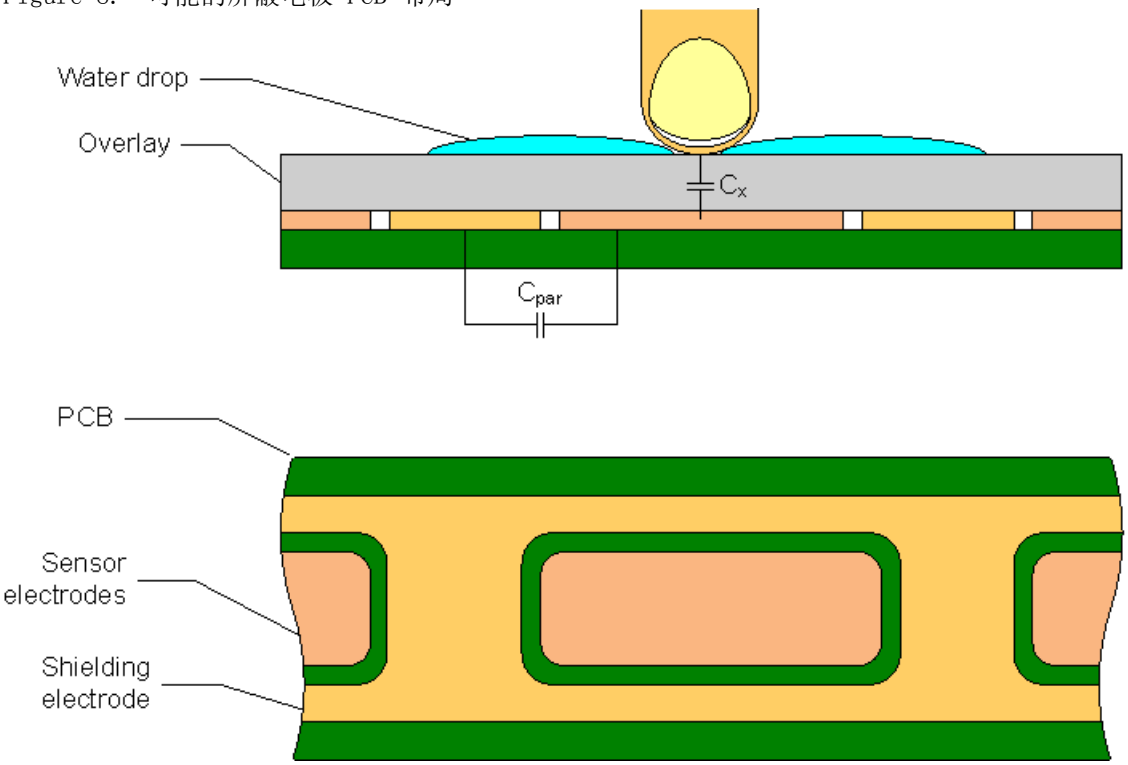
根据传感器的电容和预充电开关的工作频率，典型的值为 500Ω – 10 kΩ。 如果使用 CY3214 评估板，则可以从 2.0 kΩ 开始。

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。 对于此种情况，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。 如果器件绝缘覆盖层表面有水膜，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。 这样可以简化高级软件 API 的工作。 CSDADC 用户模块支持屏蔽电极的单独输出。

Figure 8. 可能的屏蔽电极 PCB 布局



上图展示了可能为按键屏蔽电极采取的一种布局配置。 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。

如果屏蔽电极与感应电极间存在水滴，C_{par} 将增加，调制器电流下降。 在实际测试中，调制器参考电压可能会由 API 增加，因此水滴引发的原始数值增加量必须接近于零或为较小的负值。 可以通过选择适当的调制器参考值来实现此目的。

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 器件和屏蔽电极之间连接上升限流电阻。

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：

下表对这四种配置进行了比较：

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。 如果要求良好的抗 EMI 干扰的能力，则使用 PRS16 配置。
预分频器	固定，IMO/（周期 +1）	1	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。
VC2	固定，$IMO/(VC_1 x VC_2)$	0	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。

直流和交流电气特性

Table 3. 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	3.0	5.0	5.25	V	

Table 4. 典型噪声

参数[a]	快速	正常	慢速	单位	测试条件（V_{DD} = 5V，SysClk = 24 MHz，CPU 时钟 = 12 MHz，基准线值 >= 分辨率最大计数的 70%）增益 = 4
噪声数值，峰 - 峰	1	1	1	峰至峰	分辨率 = 14
噪声数值，峰 - 峰	1	2	2	峰至峰	分辨率 = 12
噪声数值，峰 - 峰	3	3	4	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 5. 功耗

供电电压	最小值	典型值	最大值	单位	测试条件和注释
有功电流		10		mA	扫描期间平均电流，8 个传感器
待机电流		250		µA	扫描速度 = 超快，分辨率 = 9,100 ms 报告速率，8 个传感器
		1.6		mA	扫描速度 = 快速，分辨率 = 12，100 ms 报告速率，8 个传感器
睡眠 / 唤醒电流		10		µA	1s 报告速率，1 个传感器

Table 6. 5.0V PGA 直流电气特性

参数	典型值	限制	单位	条件和注释
额定增益差				
G=48.00	3.0	--	%	
G=24.00	2.2	--	%	
G=16.00	1.5	--	%	
G=4.00	0.7	--	%	
G=1.0	0.5	--	%	
输入				
输入偏移电压	4.5	--	mV	
输入电压范围	--	Vss 到 V_{DD}	V	
漏电流 [1]	1	--	nA	
输入电容 [1]	3	--	pF	
输出摆动	0.05 到 V_{DD}-0.05	--	V	
PSRR	73	--	dB	

Table 7. 5.0V PGA 交流电气特性

参数	典型值	限制	单位	条件和注释
斜率 （20% 到 80%） [2]				
功耗	0.6	--	V/µs	
功耗	2.5	--	V/µs	
功耗	9.5	--	V/µs	
建立时间				

参数	典型值	限制	单位	条件和注释
功耗	13	--	µs	
功耗	4	--	µs	
功耗	1	--	µs	
噪声 [2]				参照输入
功耗	110		nV/√Hz	除高功耗情况，其他情况下运算放大器偏压较低。 参考输入设置为 AGND。
功耗	100		nV/√Hz	

Table 8. 3.3V PGA 直流电气特性

参数	典型值	限制	单位	条件和注释
额定增益差				
G=48.00	4.0	--	%	
G=24.00	2.2	--	%	
G=16.00	1.2	--	%	
G=4.00	0.6	--	%	
G=1.0	0.3	--	%	
输入				
输入偏移电压	3.5	--	mV	
输入电压范围	--	Vss 到 V_{DD}	V	
漏电流 [1]	1	--	nA	
输入电容 [1]	3	--	pF	
输出摆动	0.05 到 V_{DD}-0.05	--	V	
PSRR	68	--	dB	
工作电流				
功耗	130	--	µA	
功耗	520	--	µA	
功耗	2000	--	µA	

Table 9. 5.0V ADC 调制器直流和交流电气特性

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 V_{DD}	V	Ref Mux = $V_{DD}/2 \pm V_{DD}/2$
输入电容	3	---	pF	包括 I/O 引脚。
输入阻抗	1/(C*clk)	---	W	
有效分辨率				
按 64 抽取	---	10	位	
按 128 抽取		12		
按 256 抽取		14		
采样率				
按 64 抽取	---	31,250	sps	数据时钟 8 MHz
按 128 抽取		15625		
按 256 抽取		7812		
直流精度				
DNL				
按 64 抽取	<1	---	LSB	源时钟 1.5 MHz
按 128 抽取	<1			
按 256 抽取	0.6			
增益错误	13	---	mV	
增益错误	2		% FSR	包括参考增益误差
数据时钟	---	0.032 to 8.0	MHz	数字模块和模拟列时钟的输入

Table 10. 3.3V ADC 调制器直流和交流电气特性

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 Vdd	V	Ref Mux = $V_{DD}/2 \pm V_{DD}/2$
输入电容	3	---	pF	包括 I/O 引脚。
输入阻抗	1/(C*clk)	---	W	
有效分辨率				
按 64 抽取	---	10	位	
按 128 抽取		12		
按 256 抽取		14		

参数	典型值	限制	单位	条件和注释
采样率				
按 64 抽取	---	31,250	sps	数据时钟 8 MHz
按 128 抽取		15625		
按 256 抽取		7812		
直流精度				
DNL				
按 64 抽取	<1	---	LSB	数据时钟 1.5 MHz
按 128 抽取	<1			
按 256 抽取	0.5			
增益错误	13	---	mV	
增益错误	2		% FSR	包括参考增益误差
数据时钟	---	0.032 to 8.0	MHz	数字模块和模拟列时钟的输入

放置

当实例化用户模块时，会自动放置适用于用户模块的模块，不提供其他放置方式。 调制器比较器位于 ACB01 连续时间模块和 ASD11/ASD21 开关电容模块中。 不同的 UM 配置使用 0-3 个数字模块。

下表汇总了所使用的数字资源。

配置	使用的数字模块
PRS16	3
PRS8	2
预分频器	1
VC2 时钟源	0

未使用的模拟模块和数字模块可供您自己使用。 所有 UM 配置均使用硬件十进制转换器。

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]。 这些引脚用来对芯片进行编程，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。

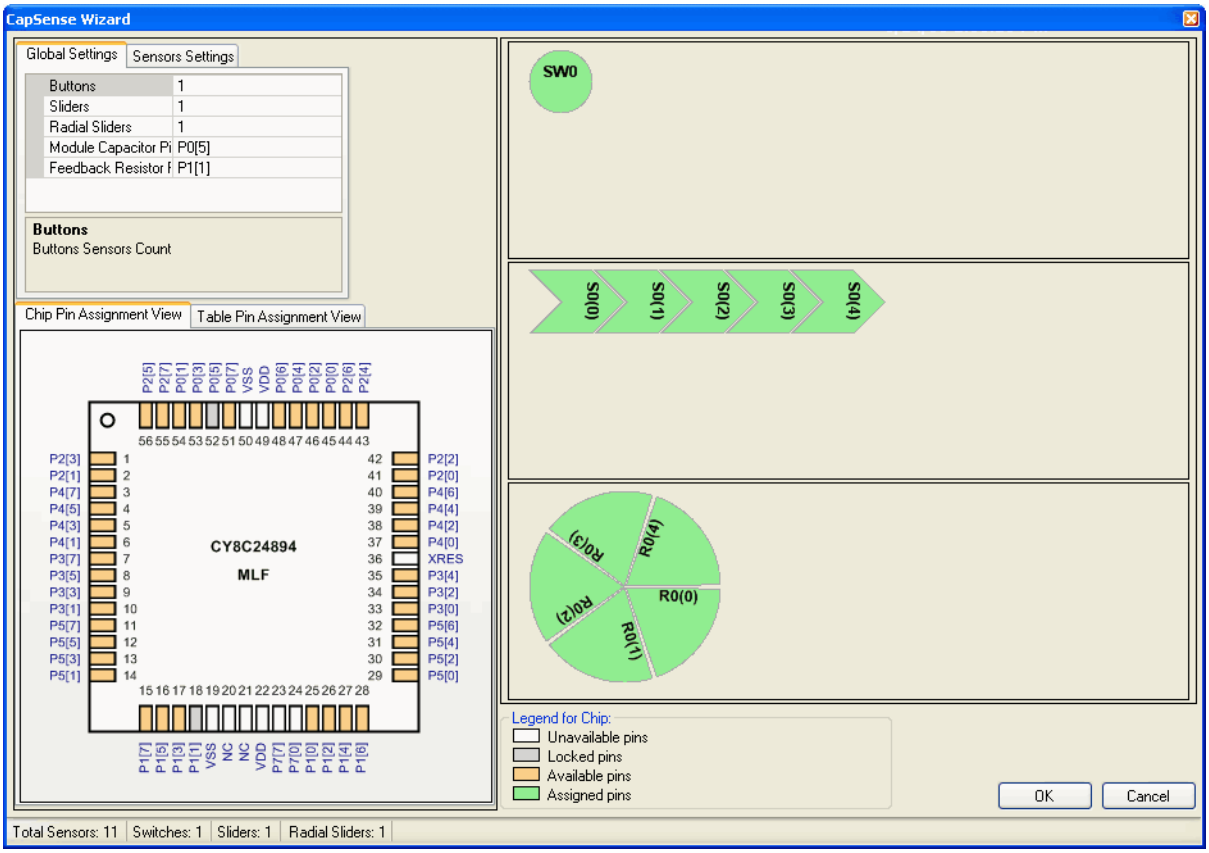
向导

CSDADC 向导用于设置 CapSense 按键、滑条和接近传感器的引脚分布。 可以选择所需的配置，使用拖放界面分配按钮和滑条的段。

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，然后单击鼠标左键选择 “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

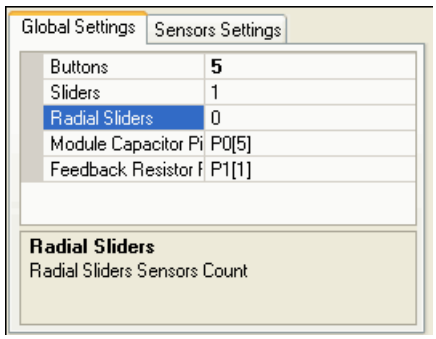
白色 - 该引脚不能用作 CapSense 输入。

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。

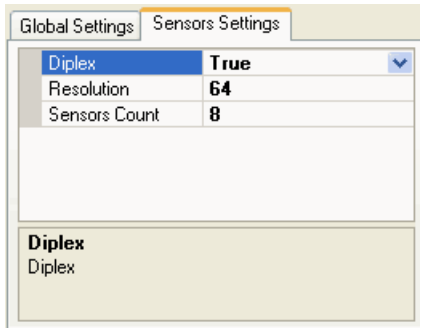
橙色 - 引脚可用于分配。

绿色 - 引脚已分配为 CapSense 输入。

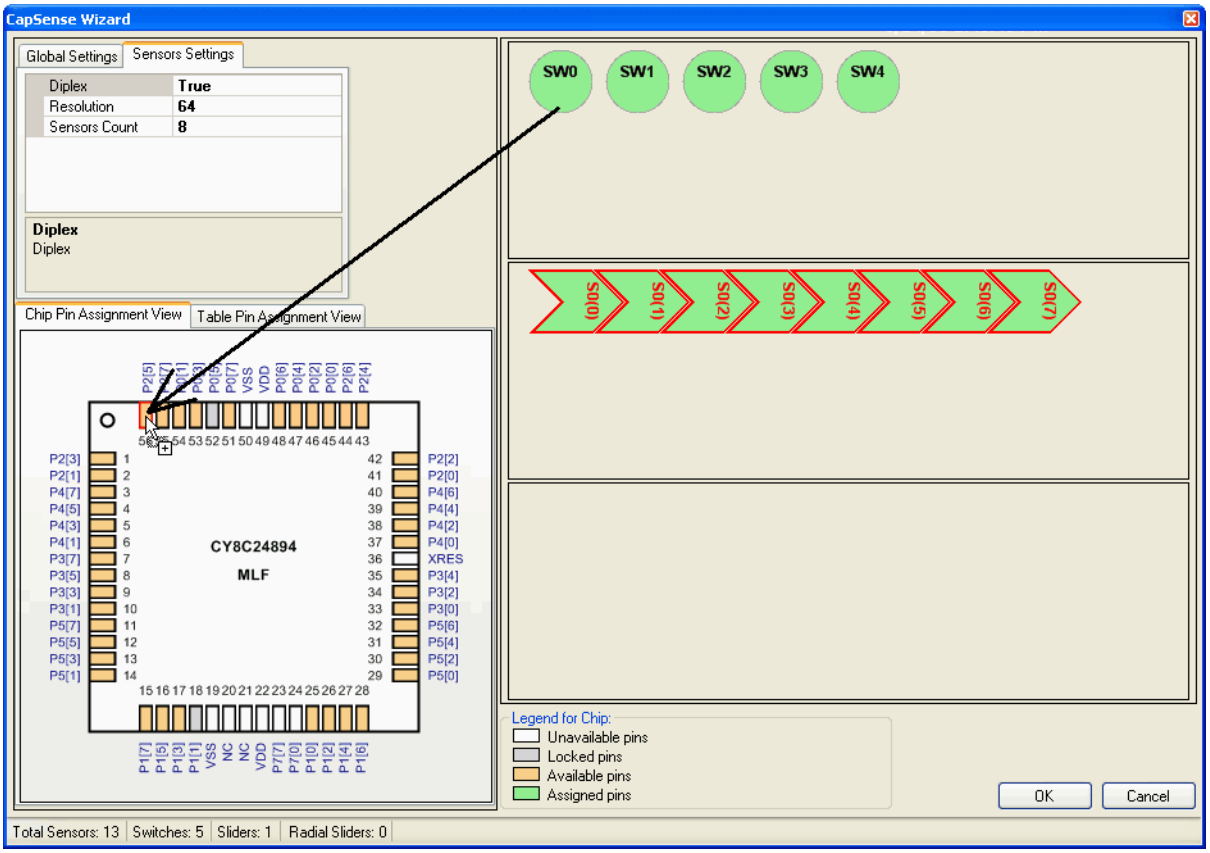
3. 键入独立按键、滑条和径向滑条的数量。 传感器总数量限制为可用引脚的数量。 X-Y 触摸板需要两个滑条，但是选择了一个。



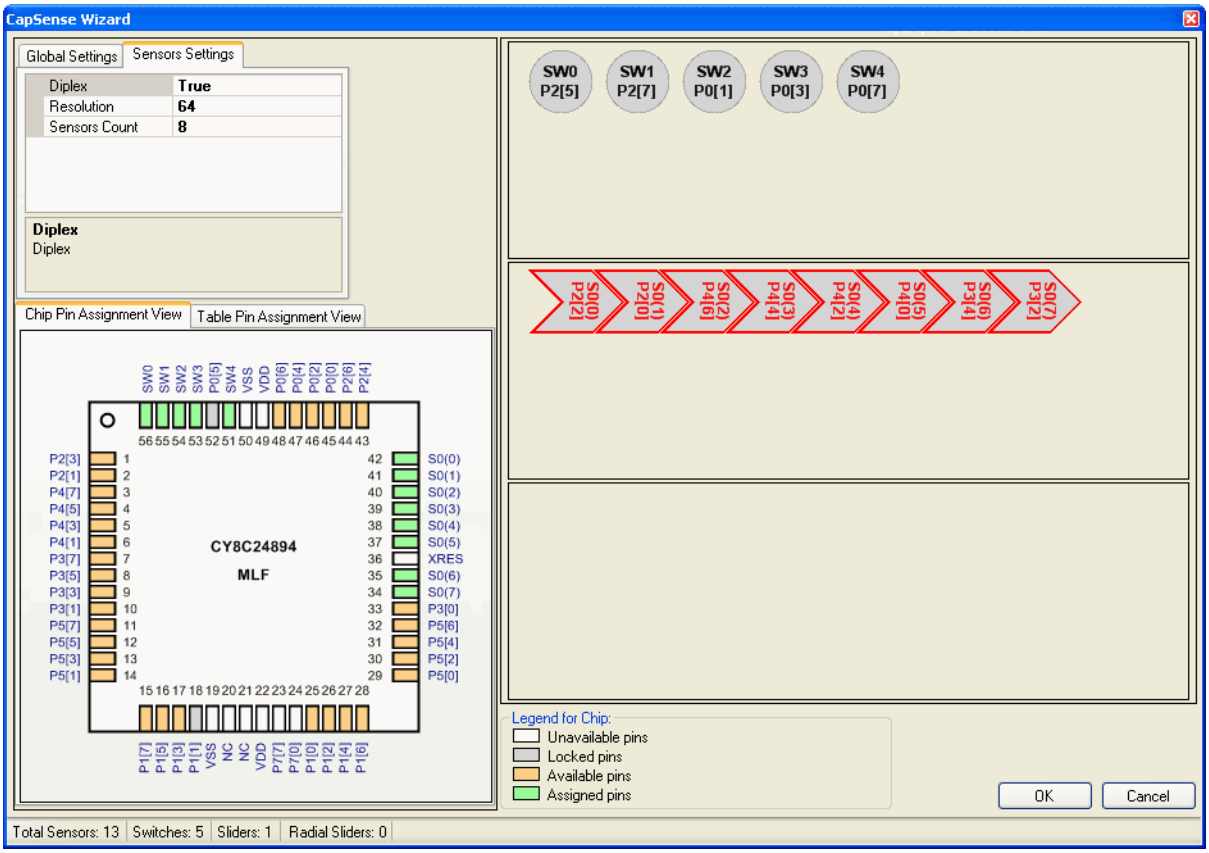
4. 选择 **传感器设置** 选项卡来设置传感器总数以及滑条和径向滑条的其他设置。 键入每个滑条中传感器元件的数量。 滑条传感器中的传感器实际最小数量为五，最大值受限于引脚数量。



5. 键入输出分辨率。 最小值为五。 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 ” 。 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 “ 复用滑条 ” 章节所述自动映射。 有关引脚连接的复用分配表，请参见 “ 复用滑条 ” 一节。
7. 左键单击按键，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。 通过将按键拖离端口更改传感器的分配。



8. 对其他按键重复以上操作。
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，先删除旧值再输入新值。 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。 该引脚分配取消，然后可以将其重新分配。

完成向导后，单击 “ 生成应用程序 ”。 根据传感器数输入、引脚分配、是否复用和分辨率，生成一组表。 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。 以下是包含十个传感器的示例表格：

```
CSDADC_Sensor_Table:
_CSDADC_Sensor_Table:
    dw    0x0001    // Port 0 Bit 0
    dw    0x0002    // Port 0 Bit 1
    dw    0x0004    // Port 0 Bit 2
    dw    0x0008    // Port 0 Bit 3
    dw    0x0010    // Port 0 Bit 4
    dw    0x0101    // Port 1 Bit 0
    dw    0x0102    // Port 1 Bit 1
    dw    0x0104    // Port 1 Bit 2
    dw    0x0108    // Port 1 Bit 3
    dw    0x0110    // Port 1 Bit 4
```

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。 第二个字节是该组中的传感器总数。 第三个字节表示是否对滑条使用了复用法 （4 表示已采用复用方式，0 表示未采用）。 第四、五、六个字节是定点乘数，将其与计算出的滑条质心相乘可以获得 CSDADC 向导中所需的分辨率。

```
CSDADC_1_Group_Table:
_CSDADC_1_Group_Table:
; Group Table:
;    Origin    Count    Diplex    DivBtwSw(wholeMSB, wholeLSB, fractByte)
 db   0x0,      0x5,     0x00,     0x00,     0x00,     0x00 ; Buttons
 db   0x5,      0x5,     0x3,      0x0,      0x0,      0x71 ; Slider 1
```

复用排列表

当某个组是滑条且采用复用方式时，将为该组生成复用表，并根据这个表顺序扫描数据。 否则，将创建标签而不放置数据。 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。 以下所示为五传感器滑条的典型示例。

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


 CSDADC_1_Diplex_Table:
_CSDADC_1_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

如果将参数设置为 “ 启用 ”，用户模块可编译代码中将包含 ADC 子程序并且可以从用户代码中调用该子程序。 如果将此参数设置为 “ 禁用 ”，则通过使用条件编译指令，用户模块可编译代码中将不包含 ADC 子程序。 如果设计不要求更长的 ADC，则将此参数设置为 “ 禁用 ” 将很有帮助，通过这种方法，CSDADC 允许使用二阶调制器在标准 CY8C24x94 CSD UM 上提升 SNR。

PGAGain

为 ADC 设置 PGA 增益。 如果使用 PSoC Designer 或 API 中所提供的 CSDADC_SetGain 例程，则增益范围介于 1 到 48.00 之间。 不支持小于 1 的增益设置。 默认值为 1.00。

PGAReference

PGA 增益参照所选的 “ 接地 ” 值。 选择包括 AGND （片上模拟接地）、V_{SS}、相邻连续时间 (CT) 和开关电容 (SC) 模块。 CT 和 SC 模块连接允许将可调偏移作为控制参考电压。 CSDADC_SetRef API 可用于在运行时更改参考电压。 默认 PGA 参考值为 AGND。

AnalogBus

PGA 模块输出可以通过本地互连的模拟 PSoC 模块阵列的网络和 / 或通过模拟输出总线进行路由。 将 PGA 用户模块 AnalogBus 参数设置为 “ 禁用 ”（默认值），可限制到本地网络的可能连接组。 如果在总线上启用 PGA AnalogBus 输出，一定要注意确保无其他用户模块驱动同一总线。 用户模块的 CSD 和 ADC 部分均使用此参数。

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。

噪声阈值

对于单个传感器，此参数将设置一个数值，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。 如果采样到的计数值低于基准线与 “ 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。

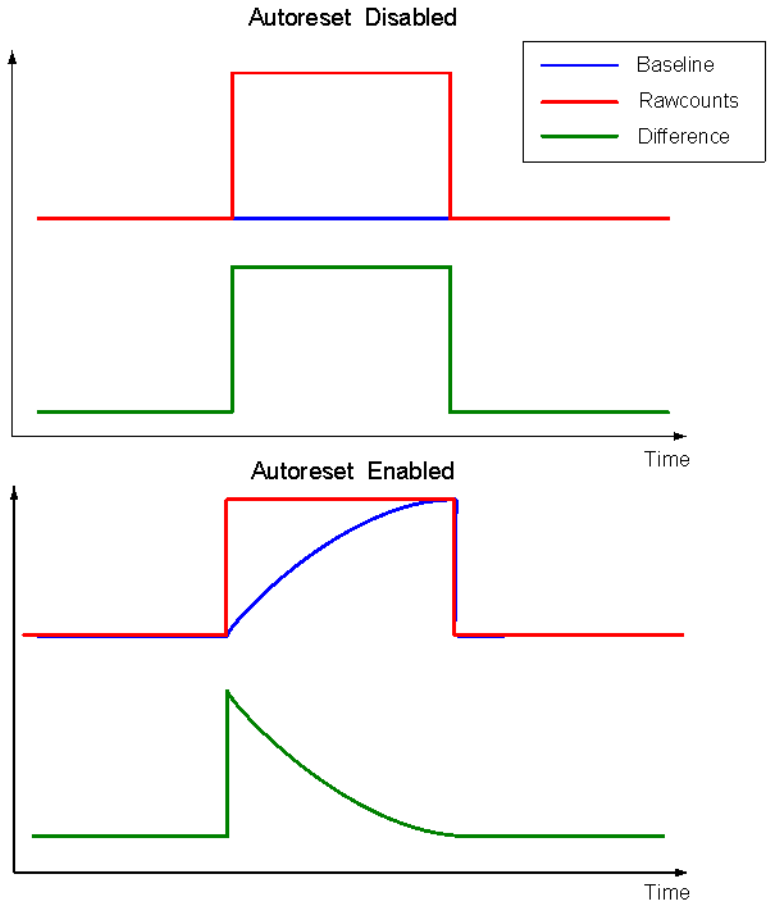
传感器自动复位

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数值突然上升时，可以阻止传感器始终打开。 原始数值突然上升可能是由电源电压剧烈波动、高能射频噪声源或温度快速变化所导致。 默认为已启用。

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。 除非是遇到在任何物体未触碰传感器而原始数值突然上升时传感器始终打开的问题，否则应将此参数保留为 “ 禁用 ”。

下图说明了此参数对基准线更新的影响。

Figure 9. 传感器自动复位 (Sensor Autoreset) 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, 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

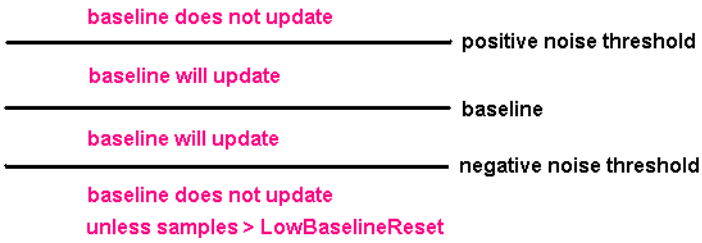
“ 防反跳 ” 参数为传感器活动的瞬变增加了防反跳计数器。 为了让传感器从不活动转换到活动状态，对于指定的样品数量，差值必须大于手指阈值与迟滞之和。 防反跳计数器按 bIsSensorActive 或 bIsAnySensorActive API 函数递增。

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。 如果当前的原始数值低于基准线且二者之差大于此阈值，则不更新基准线。 但是，如果对于低基准线复位 (LowBaselineReset) 参

数所指定的样品数量，当前原始数值处于较低状态 （差大于阈值），则对基准线进行复位。 可能值为 0 到 255。 默认值为 20。



扫描速度 (Scanning Speed)

此参数影响传感器的扫描速度。 选择包括**快速**、**正常**和**慢速**。 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

分辨率

此参数确定扫描分辨率 （以 “ 位 ” 为单位）。 PRS16 配置可以使用分辨率为 9 到 16 位的扫描传感器。 默认分辨率是 12 位。 PRS8、预分频器和 VC2 配置只允许在 10、12 和 14 位扫描传感器。 如果需要这些配置具有更高的分辨率，则使用过度采样并对几个采样求平均值。 N 位的扫描分辨率最大原始计数为 2^N-1。 ADC 分辨率设置使用同一参数值。

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。

Table 11. 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		
	快速	正常	慢速
9	424	548	990
10	680	890	1670
11	1200	1580	3040
12	2220	2880	5800
13	4280	5680	11200
14	8400	11100	22300
15	16600	22100	44200
16	33000	44000	88000

Table 12. 对于 24 MHz IMO 操作、PRS8、预分频器和 VC2 配置，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		
	快速	正常	慢速
10	124	136	296
12	220	220	548
14	428	552	1060

注： 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})。 从以下可用引脚选择。 默认引脚为 P0[5]。

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)。 从以下可用引脚选择。 由于 ISSP 编程可能出现问题，因此不建议使用 P1[1] 来连接反馈电阻。 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 UM 参数列表中不可选择。 默认引脚为 P1[1]。

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 PRS8 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。

建议使用值 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。

PRS 多项式

此参数设置的是 PRS8 配置中的 PRS 多项式。 有以下两种选择选项：

- 较短 - 较短的多项式设置会获得较好的信噪比 (SNR)，但由于重复周期较短，因此终端设备更易受到外部噪声源的影响。
- 较长 - 较长的多项式设置会获得较差的信噪比 (SNR)，但器件对于噪声信号的抵抗能力较强。 默认值。

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。默认值为 “ 无 ”。

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。

注意 ** 此种情况如同所有用户模块的 API，A 和 X 寄存器的值可以通过调用 API 函数来更改。 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。

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，其中包含每个传感器的原始数据。 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_ScanSensorsAveraging()

CSDADC_waSnsDiff [] - 这是一个整数阵列，其中包含原始数据与每个传感器基准数据之间的差值。 阵列大小与传感器数量相等。

CSDADC_baSnsOnMask[] - 这是一个保持传感器开或关状态的字节阵列 （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。 此函数必须在调用其他任何用户模块函数前调用。

C 原型：

```
void  CSDADC_Start()
```

汇编：

```
lcall  CSDADC_Start
```

参数：

无

返回值：

无

副作用：

**

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_SetRefMux

说明：

此子程序将在系统中设置 AGND、RefHi、RefLo 参考，覆盖在 PSoC Designer 的 “ 器件编辑器 ” 中设置的值

C 原型：

```
void  CSDADC_SetRefMux(BYTE bReference)
```

汇编：

```
mov A, [Ref]
lcall  CSDADC_SetRefMux
```

参数：

参考值，可以是下列值之一：

符号名	值	说明
REF_HALFVDD_BANDGAP	0x00	(Vdd/2) +/- 带隙
REF_P24_P26	0x08	P2[4] +/- P2[6]
REF_HALFVDD_HALFVDD	0x10	(Vdd/2) +/- (Vdd/2)
REF_2BANDGAP_BANDGAP	0x18	带隙 +/- 带隙
REF_2BANDGAP_P26	0x20	2Bandgap +/- P2[6]
REF_P24_BANDGAP	0x28	P2[4] +/- 带隙
REF_BANDGAP_BANDGAP	0x30	带隙 +/- 带隙
REF_16BANDGAP_16BANDGAP	0x38	1.6BandGap +/- 1.6BandGap

返回值：

无

副作用：

**

CSDADC_SetPGAGain

说明：

为 PGA 模块设置增益，覆盖 “ 器件编辑器 ” 中设置的值。

C 原型：

```
void  CSDADC_SetPGAGain(BYTE bGainSetting)
```

汇编：

```
mov   A, bGainSetting
lcall  CSDADC_SetPGAGain
```

参数：

bGainSetting: 下表给出了在 C 语言和汇编语言 include 文件中提供的符号名及其相关值。 PGA 增益的设置范围是 1 到 48，不支持将其设置为 1 以下的数字。 该函数对于 ADC 和 CSD 模式很常用。首先设置 ADC 预放大器增益，然后再设置调制器增益。

符号名	值
PGA_G48_0	0x0C
PGA_G24_0	0x1C
PGA_G16_0	0x08
PGA_G8_00	0x18

符号名	值
PGA_G5_33	0x28
PGA_G4_00	0x38
PGA_G3_20	0x48
PGA_G2_67	0x58
PGA_G2_27	0x68
PGA_G2_00	0x78
PGA_G1_78	0x88
PGA_G1_60	0x98
PGA_G1_46	0xA8
PGA_G1_33	0xB8
PGA_G1_23	0xC8
PGA_G1_14	0xD8
PGA_G1_06	0xE8
PGA_G1_00	0xF8

返回值：

无

副作用：

**

CSDADC_SetPGARef

说明：

为 PGA 模块设置参考。 PGA 增益以用户所选的 “ 接地 ” 值为参考。 选择包括 AGND （片上模拟接地）、V_{SS}、相邻连续时间 (CT) 和开关电容 (SC) 模块。 CT 和 SC 模块连接允许将可调偏移作为控制参考电压。

C 原型：

```
void  CSDADC_SetPGARef(BYTE bRefSetting)
```

汇编：

```
mov   A, bRefSetting
lcall  CSDADC_SetPGARef
```

参数：

bRefSetting：此表给出了 C 语言程序和汇编语言程序内提供的符号名称及其相关值。

符号名	值
ADC00	0x00
AGND	0x01
VSS	0x02
ASC10	0x03

返回值：

无

副作用：

**

CSDADC_EnableADC

说明：

激活 UM 的 ADC 函数。 只能在此子程序调用后才允许调用 ADC 相关子程序。

C 原型：

```
void  CSDADC_EnableADC (void);
```

汇编：

```
lcall CSDADC_EnableADC
```

参数：

无

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。 现在允许在此子程序调用后调用 CapSense 相关子程序。

C 原型：

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

汇编：

```
lcall CSDADC_EnableCapsense
```

参数：

无

返回值：

无

副作用：

**

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。

C 语言原型：

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

汇编：

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

参数：

bPort - 设置 ADC 输入端口

bMask - 端口位。

副作用：

**

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。 CSDADC_ EnableInput 和 CSDADC_ DisableInput 子程序必须成对调用。

C 语言原型：

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

汇编：

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

参数：

bPort - 设置 ADC 输入端口

bMask - 端口位。

副作用：

**

CSDADC_StartADC

说明：

启动并持续运行 ADC。 ADC 运行必须启用全局中断。

C 原型：

```
void CSDADC_StartADC(void)
```

汇编：

```
lcall CSDADC_StartADC
```

参数：

无

返回值：

无

副作用：

**

CSDADC_StopADC

说明：

关闭 ADC 并立即停止转换处理。

C 原型：

```
void CSDADC_StopADC(void)
```

汇编：

```
lcall CSDADC_StopADC
```

参数：

无

返回值：

无

副作用：

**

CSDADC_fIsDataAvailable

说明：

检查 ADC 的状态。

C 原型：

```
BYTE CSDADC_fIsDataAvailable(void)
```

汇编：

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

参数：

无

返回值：

如果数据已转换且可以读取，则返回非零值。

副作用：

**

CSDADC_wGetData

说明：

返回转换的数据。必须调用 CSDADC_fIsDataAvailable() 以验证数据样本是否已就绪。

C 原型：

```
WORD CSDADC_wGetData(void)
```

汇编：

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

参数：

无

返回值：

以无符号整数格式返回转换的数据

副作用：

**

CSDADC_wGetDataClearFlag

说明：

获取数据并将数据可用标志复位。

C 原型：

```
WORD CSDADC_wGetDataClearFlag(void)
```

汇编：

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

参数：

无

返回值：

以无符号整数格式返回 ADC 转换样本

副作用：

**

CSDADC_ScanSensor

说明：

扫描所选的电容传感器。 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：

A => 传感器编号

返回值：

无

副作用

**

CSDADC_ScanSensorAveraging

说明：

扫描所选的传感器 2^{bPower} 次，然后对结果求平均值。 由于输出数据向右移动 bPower 位，因此保留了输出数据分辨率。 对多个扫描结果求平均值可以得到更高的信噪比 (SNR)。 在使用 PRS8、预分频器或 VC2 预充电时钟配置时，此函数比多次调用 CSDADC_ScanSensor() 函数更快，因为 $Sinc^2$ 滤波器需要 CSDADC_ScanSensor() 函数才能跳过前两个样本。

为给定的分辨率和扫描速度使用 CSDADC_ScanSensor() 函数时，如果 T 是扫描时间的单个传感器，则速度之差为：

- ScanSensorAveraging - $(T/3)*2^{bPower} + 2T/3$
- 多个 ScanSensor 调用 - $T*2^{bPower}$

只有当在应用程序中不能够获得良好的信噪比结果，即使经过仔细调整所述的结果，应该使用此功能 CY8C21x34/B CapSense 设计指南中所述内容仔细调整结果后，仍无法从应用中获得良好的 SNR 结果时，才能使用此函数，例如通过厚外覆层扫描时、使用 CapSense 进行接近检测时或者必须在较低频率下操作 （例如，使用 ITO 薄膜）时。

C 原型：

```
void CSDADC_ScanSensorAveraging(BYTE bSensor, BYTE bPower);
```

汇编：

```
mov A,  bSensor
mov A,  bPower
lcall CSDADC_ScanSensorAveraging
```

参数：

bSensor：要扫描的传感器数量。

bPower： 定义传感器的扫描次数。 传感器扫描次数为 2^{bPower} 次。 允许的值的范围是 0..8。

返回值：

无

副作用

**

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

**

CSDADC_UpdateSensorBaseline

说明：

单独针对每个传感器计算的历史数值称为传感器的基准线。 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 bSensorNum)
```

汇编：

```
mov   A,   bSensorNum
lcall  CSDADC_UpdateSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

副作用：

**

CSDADC_UpdateAllBaselines

说明：

使用 CSDADC_bUpdateSensorBaseline() 函数更新所有传感器的基准线

C 原型：

```
void  CSDADC_UpdateAllBaselines()
```

汇编：

```
lcall CSDADC_UpdateAllBaselines
```

参数：

无

返回值：

无

副作用：

**

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 bSensorNum)
```

汇编：

```
mov  A,  bSensorNum
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。

副作用：

**

CSDADC_bIsAnySensorActive

说明：

与手指阈值进行比较，检查所有传感器的差值阵列。 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。

副作用：

**

CSDADC_wGetCentroidPos

说明：

为了计算中心位置，检查各传感器的差值阵列。 如果存在，则偏移和长度存储在临时变量中，并且根据 CSDADC 向导中指定的分辨率计算出中心位置。 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。 每次扫描后必须调用此函数，但只能调用一次，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计算 / 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。

注意：如果滑条段的噪声数值大于噪声阈值，则此子例程可能生成错误的中心位置值。 噪声阈值必须仔细设置 （足够高于噪声水平），防止噪声生成错误中心位置。

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。

C 原型：

```
void CSDADC_InitializeSensorBaseline(BYTE bSensorNum)
```

汇编：

```
mov A, bSensorNum
lcall CSDADC_InitializeSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

副作用：

**

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：

**

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：

**

CSDADC_SetScanMode

说明：

设置扫描速度和分辨率。 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_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。 数值范围取决于用户模块配置。

以下可能的常数用于 PRS16 配置的 bResolution 参数：

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

以下给出了可能的 PRS8、预分频器、和 VC2 配置 bResolution 参数：

常量	值
CSDADC_10_BIT_RESOLUTION	10
CSDADC_12_BIT_RESOLUTION	12
CSDADC_14_BIT_RESOLUTION	14

返回值：

无

副作用：

**

CSDADC_ClearSensors

说明：

通过按顺序为每个电容传感器调用 CSDADC_wGetPortPin() 和 CSDADC_DisableSensor()，将所有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：

**

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：

**

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：

**

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。 修改驱动模式以便将所选端口和引脚置于模拟 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：

**

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：

**

固件源代码示例

此代码启动用户模块并连续扫描传感器。

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

WORD wResult;

void main(void) {
   M8C_EnableGInt;

   CSDADC_SetRefMux(CSDADC_REF_BANDGAP_BANDGAP); //BandGap +/- BandGap
   CSDADC_Start();
   CSDADC_SetDefaultFingerThresholds();
   CSDADC_InitializeBaselines();

   while (1) {
     CSDADC_SetRefMux(CSDADC_REF_BANDGAP_BANDGAP); //BandGap +/- BandGap
     CSDADC_EnableCapsense();
     CSDADC_ScanAllSensors();
     CSDADC_UpdateAllBaselines();
```

```
    //Here you can insert a code for processing of CapSence data

    CSDADC_SetRefMux(CSDADC_REF_HALFVDD_HALFVDD);//(Vdd/2) +/-(Vdd/2)
    CSDADC_SetPGAGain(CSDADC_G1_00);
    CSDADC_EnableADC();
    CSDADC_EnableInput(0x01,0x02); // use P2[0]
    CSDADC_StartADC();

    // Skip three first samples
    while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();
    while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();
    while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();

    //Here you can insert a code for processing of ADC data

    CSDADC_StopADC();
    CSDADC_DisableInput(0x01, 0x02); // ruquired 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

area bss(RAM)          ;inform assembler that variables follow
bCounter:  blk 1

area text(ROM,REL)     ;inform assembler that program code follows

_main:

    M8C_EnableGInt

mov    A, CSDADC_REF_BANDGAP_BANDGAP
    lcall  CSDADC_SetRefMux       ; BandGap +/- BandGap

lcall  CSDADC_Start
    lcall  CSDADC_SetDefaultFingerThresholds
    lcall  CSDADC_InitializeBaselines

loop:
    mov    A, CSDADC_REF_BANDGAP_BANDGAP
    lcall  CSDADC_SetRefMux       ; BandGap +/- BandGap

    lcall  CSDADC_EnableCapsense
    lcall  CSDADC_ScanAllSensors
```

```
 lcall  CSDADC_UpdateAllBaselines

    ;Here you can insert a code for processing of CapSence data

    mov    A, CSDADC_REF_HALFVDD_HALFVDD
    lcall  CSDADC_SetRefMux        ; (Vdd/2) +/-(Vdd/2)

    mov    A, CSDADC_G1_00
    lcall  CSDADC_SetPGAGain

    lcall  CSDADC_EnableADC

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

    lcall  CSDADC_StartADC

    ; Skip three first samples

mov   [bCounter], 3
.scan:
lcall  CSDADC_fIsDataAvailable
    cmp    A, 0
    jz     .scan
    lcall  CSDADC_wGetDataClearFlag
    dec    [bCounter]
    jnz    .scan

; The ADC rezult is stored in A (LSB) and X(MSB)
    ; Here you can insert a code for processing of ADC data

lcall  CSDADC_StopADC

    mov    A, 0x01
    mov    X, 0x02
    lcall  CSDADC_DisableInput

    jmp    loop
```

配置寄存器

Table 13. 模块 PGA，寄存器： ACB_CONTROL0_REG (ACB01CR0)，组 x

位	7	6	5	4	3	2	1	0
值	增益					1	参考	

增益由具有 PGA 增益名称的用户模块参数和 ADC 相关 API 保持。 参考由 PGA 参考用户模块参数保持。

Table 14. 模块 PGA，寄存器： ACB_CONTROL1 (ACB01CR1)，组 x

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

AnalogBus 由具有相同名称的用户模块参数保持。 电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 15. 模块 PGA，寄存器： ACB_CONTROL2 (ACB01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	功耗	

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

Table 16. 模块 PGA，寄存器： ACB_CONTROL3 (ACB01CR3)，组 x

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

选择了 24 或 48 的 PGA 增益即设置了 ExGain 位。

Table 17. 模块 ADC1，寄存器： AtoD1cr0 (ASD11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	0	1	0	0	0

Table 18. 模块 ADC1，寄存器： AtoD1cr1 (ASD11CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 19. 模块 ADC1，寄存器： AtoD1cr2 (ASD11CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 20. 模块 ADC1，寄存器： AtoD1cr3 (ASD11CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	1	0	功耗	

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

Table 21. 模块 ADC2，寄存器： AtoD2cr0 (ASC21CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 22. 模块 ADC2，寄存器： AtoD2cr1 (ASC21CR1)，组 x

位	7	6	5	4	3	2	1	0
值	00	0	0	0	0	0	0	0

Table 23. 模块 ADC2，寄存器： AtoD2cr2 (ASC21CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 24. 模块 ADC3，寄存器： AtoD2cr3 (ASC21CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	0	0	功耗	

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

Table 25. 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 26. 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 27. 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引入到内部的行输出。 这由具有相同名称的用户模块参数控制。

Table 28. 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 29. 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 30. 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 31. 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 PRS8 时钟源配置寄存器的 CSDADC

Table 32. 模块 PGA，寄存器： ACB_CONTROL0_REG (ACB01CR0)，组 x

位	7	6	5	4	3	2	1	0
值	增益					1	参考	

增益由具有 PGA 增益名称的用户模块参数和 ADC 相关 API 保持。 参考由 PGA 参考用户模块参数保持。

Table 33. 模块 PGA，寄存器： ACB_CONTROL1 (ACB01CR1)，组 x

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

AnalogBus 由具有相同名称的用户模块参数保持。 电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 34. 模块 PGA，寄存器： ACB_CONTROL2 (ACB01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	功耗	

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

Table 35. 模块 PGA，寄存器： ACB_CONTROL3 (ACB01CR3)，组 x

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

选择了 24 或 48 的 PGA 增益即设置了 ExGain 位。

Table 36. 模块 ADC1，寄存器： AtoD1cr0 (ASD11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	0	1	0	0	0

Table 37. 模块 ADC1，寄存器： AtoD1cr1 (ASD11CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 38. 模块 ADC1，寄存器： AtoD1cr2 (ASD11CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 39. 模块 ADC1，寄存器： AtoD1cr3 (ASD11CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	1	0	功耗	

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

Table 40. 模块 ADC2，寄存器： AtoD2cr0 (ASC21CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 41. 模块 ADC2，寄存器： AtoD2cr1 (ASC21CR1)，组 x

位	7	6	5	4	3	2	1	0
值	00	0	0	0	0	0	0	0

Table 42. 模块 ADC2，寄存器： AtoD2cr2 (ASC21CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 43. 模块 ADC3，寄存器： AtoD2cr3 (ASC21CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	0	0	功耗	

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

Table 44. 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 45. 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 46. 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 47. 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 48. 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 49. 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 50. 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。

带有 PWM8 时钟源配置寄存器的 CSDADC

Table 51. 模块 PGA，寄存器： ACB_CONTROL0_REG (ACB01CR0)，组 x

位	7	6	5	4	3	2	1	0
值	增益					1	参考	

增益由具有 PGA 增益名称的用户模块参数和 ADC 相关 API 保持。 参考由 PGA 参考用户模块参数保持。

Table 52. 模块 PGA，寄存器： ACB_CONTROL1 (ACB01CR1)，组 x

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

AnalogBus 由具有相同名称的用户模块参数保持。 电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 53. 模块 PGA，寄存器： ACB_CONTROL2 (ACB01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	功耗	

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

Table 54. 模块 PGA，寄存器： ACB_CONTROL3 (ACB01CR3)，组 x

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

选择了 24 或 48 的 PGA 增益即设置了 ExGain 位。

Table 55. 模块 ADC1，寄存器： AtoD1cr0 (ASD11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	0	1	0	0	0

Table 56. 模块 ADC1，寄存器： AtoD1cr1 (ASD11CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 57. 模块 ADC1，寄存器： AtoD1cr2 (ASD11CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 58. 模块 ADC1，寄存器： AtoD1cr3 (ASD11CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	1	0	功耗	

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

Table 59. 模块 ADC2，寄存器： AtoD2cr0 (ASC21CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 60. 模块 ADC2，寄存器： AtoD2cr1 (ASC21CR1)，组 x

位	7	6	5	4	3	2	1	0
值	00	0	0	0	0	0	0	0

Table 61. 模块 ADC2，寄存器： AtoD2cr2 (ASC21CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 62. 模块 ADC3，寄存器： AtoD2cr3 (ASC21CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	0	0	功耗	

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

Table 63. 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 64. 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 65. 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 66. 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 67. 模块 PWM，寄存器： Count (DxBxxDR0)，组 0

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

Table 68. 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 69. 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。

带有 VC2 时钟源配置寄存器的 CSDADC

Table 70. 模块 PGA，寄存器： ACB_CONTROL0_REG (ACB01CR0)，组 x

位	7	6	5	4	3	2	1	0
值	增益					1	参考	

增益由具有 PGA 增益名称的用户模块参数和 ADC 相关 API 保持。 参考由 PGA 参考用户模块参数保持。

Table 71. 模块 PGA，寄存器： ACB_CONTROL1 (ACB01CR1)，组 x

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

AnalogBus 由具有相同名称的用户模块参数保持。 电源由 CSDADC_Start() 和 CSDADC_Stop() API 保持。

Table 72. 模块 PGA，寄存器： ACB_CONTROL2 (ACB01CR2)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	功耗	

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

Table 73. 模块 PGA，寄存器： ACB_CONTROL3 (ACB01CR3)，组 x

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

选择了 24 或 48 的 PGA 增益即设置了 ExGain 位。

Table 74. 模块 ADC1，寄存器： AtoD1cr0 (ASD11CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	0	1	0	0	0

Table 75. 模块 ADC1，寄存器： AtoD1cr1 (ASD11CR1)，组 x

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 76. 模块 ADC1，寄存器： AtoD1cr2 (ASD11CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 77. 模块 ADC1，寄存器：AtoD1cr3 (ASD11CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	1	0	功耗	

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

Table 78. 模块 ADC2，寄存器：AtoD2cr0 (ASC21CR0)，组 x

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 79. 模块 ADC2，寄存器：AtoD2cr1 (ASC21CR1)，组 x

位	7	6	5	4	3	2	1	0
值	00	0	0	0	0	0	0	0

Table 80. 模块 ADC2，寄存器：AtoD2cr2 (ASC21CR2)，组 x

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

CompBus 由 CSDADC API 控制。

Table 81. 模块 ADC3，寄存器：AtoD2cr3 (ASC21CR3)，组 x

位	7	6	5	4	3	2	1	0
值	1	1	1	0	0	0	功耗	

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

版本历史记录

版本	创作者	说明
1.1	DHA	更新了错误消息。

版本	创作者	说明
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.