CapSense® Sigma-Delta 数据表 CSD V 1.50

Copyright © 2007-2011 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。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	0	951	28	2-5
带有 1 个传感器、PRS8-based 的用户模块	1	1	0	923	26	2-5
带有 1 个传感器、基于预分频器的 PRS8	2	1	0	935	26	2-5
每个附加的 Cap-Sense 按键	-	-	-	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	-	-	-	584	79	-
每个附加的滑条元件	-	-	-	11	2	1
使用滑条双工时，静态代码和 RAM 增加	-	-	-	14	-	-

有关使用此用户模块的一个或多个完整配置的实用型示例项目，请转到 www.cypress.com/psocexampleprojects.

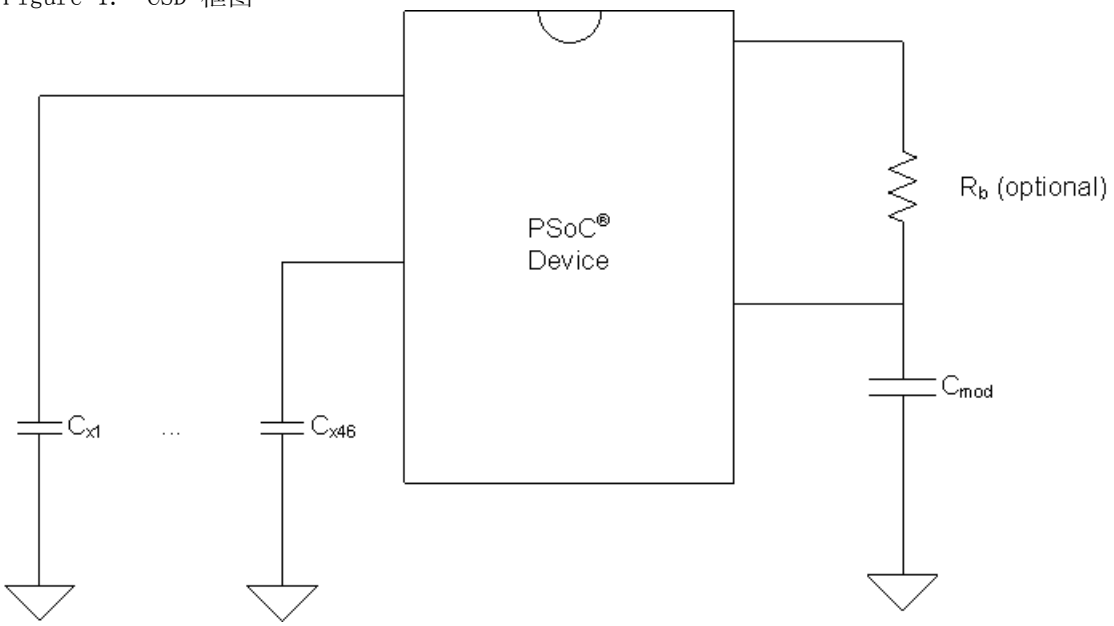
特性和概述

- 扫描 1 ～ 46 个电容传感器。
- 感应能力最多可穿透 15 mm 的玻璃覆盖层。
- 使用线缆传感器时，检测距离接近 20 cm。
- 对交流电源噪声、EMC 噪声和电源电压变化，具有极强的抗干扰能力。
- 支持独立传感器和滑条电容传感器的任意组合。
- 通过双工，使滑条传感器的物理分辨率增加一倍。
- 利用内插法，提高滑条传感器的分辨率。
- 带有两个滑条传感器的触摸板支架。
- 感应能力通过高阻抗性传导材质 （例如 ITO 薄膜）实现。
- 即使存在水膜或水滴的情况下，屏蔽电极仍可为可靠的运行提供保证。
- 通过 CSD 向导完成传感器和引脚分配。
- 使用集成基准线值更新算法处理温度、湿度和静电释放 （ESD） 事件。

- 可轻松调整的操作参数。
- PC GUI 应用程序支持实时原始数据监控和参数优化。

CSD（使用 Sigma-Delta 调制器的电容式感应技术）通过开关电容技术提供 CapSense® 功能，该技术通过 Sigma-Delta 调制器可以将感应开关电容电流转换为数字代码。

Figure 1. CSD 框图



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 CSD 用户模块。
3. 在工作区浏览器中右键单击 CSD 用户模块，以访问 CSD 向导 （数据表中稍后将介绍该向导）。
4. 设置所需的传感器、滑条或旋转滑条的数量。
5. 设置每个传感器的传感器设置。
6. 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。
7. 生成应用，并切换到 “ 应用编辑器 ”。
8. 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。
9. 将 I^2C-USB 桥连接至目标电路板，并观察信号。
10. 更改 CSD 参数以优化设置和重建应用。
11. 对 PSoC 设备进行编程，验证模块运行。调整 CSD 参数以实现 *Signal-to-Noise Ratio Requirements for CapSense Applications*（CapSense 应用的信噪比要求）中所述的 5:1 信噪比要求 - AN2403.

如果遇到任何问题，请参见*附录*中的*故障排除*部分。

[+] Feedback

功能说明

电容式传感器阵列包含独立传感器、滑条传感器以及触摸板，触摸板部署为一对互相垂直的滑条。 高层级的决策逻辑可为温度、湿度和电源电压变化等环境因素提供补偿。 独立的屏蔽电极可用于屏蔽传感器阵列以降低杂散电容。 从而在有水膜或水滴的情况下提供更可靠的操作。

高层级软件功能可提供滑条双工，以便在两个物理位置可以使用一个电气传感器，用以提高分辨率。 通过这些功能，还可以在物理传感器位置之间进一步插补解析传感器位置。

电容式传感器由物理、电气和软件组件构成：

- 物理组件
  - 物理传感器本身通常是一个在连接到 PSoC 的 PCB 上制作的导电图形，其显示屏带有绝缘封面、柔性薄膜或透明外覆层。
- 电气组件
  - 一种将传感器电容转换为数字格式的方法。 转换系统包括感应开关电容器、Sigma-Delta 调制器和基于计数器的数字过滤器，可将调制器输出位流转换为可读的数字格式。
- 软件组件
  - 检测和补偿软件算法可将计数值转换为传感器检测决策。
  - 在连续情况下，将会提供相关传感器 （例如滑条和触摸板）API，以插入分辨率比传感器的物理间距更高的位置。 例如，您可以使用 10 个传感器创建音量滑条，并使用所提供的固件将音量数字提高到 100。 或者，通过相同的 API，您也可以使用逐渐变细彼此插入对方的两个电容式传感器，并确定二者之间某个导电物体 （例如手指）的位置。

测量电容的方法有许多种，此用户模块中使用的方法是将开关电容与一个 Delta-Sigma 调制器组合在一起。

首次使用 CSD 用户模块之前，建议阅读下列文档。

- *CapSense Best Practices*（CapSense 最佳实践） - AN2394
- *Signal-to-Noise Ratio Requirements for CapSense Applications*（CapSense 应用的信噪比要求） - AN2403
- *Charting Tool to Debug CapSense Applications*（调试 CapSense 应用的制表工具） - AN2397
- *EMC Design Considerations for PSoC CapSense Applications*（PSoC CapSense 应用的 EMC 设计注意事项） - AN2318
- *Power Consumption and Sleep Considerations in Capacitive Sensing Applications*（电容式感测应用的功耗和睡眠注意事项） - AN2360
- *Layout Guidelines for PSoC CapSense*（PSoC CapSense 布局指南） - AN2292
- *Software Implementation of a Universal Asynchronous Transmitter*（通用异步发射器的软件实现） - AN2399
- *Waterproof Capacitance Sensing*（防水电容式感测） - AN2398

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。

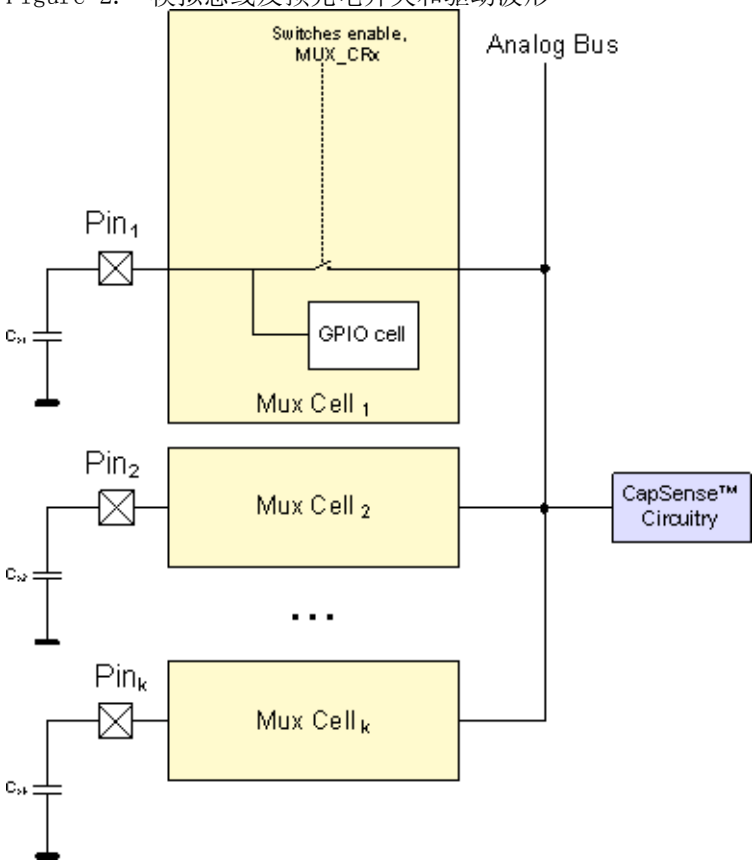
[+] Feedback

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。 CSD 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，并在相位 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 2. 模拟总线及预充电开关和驱动波形

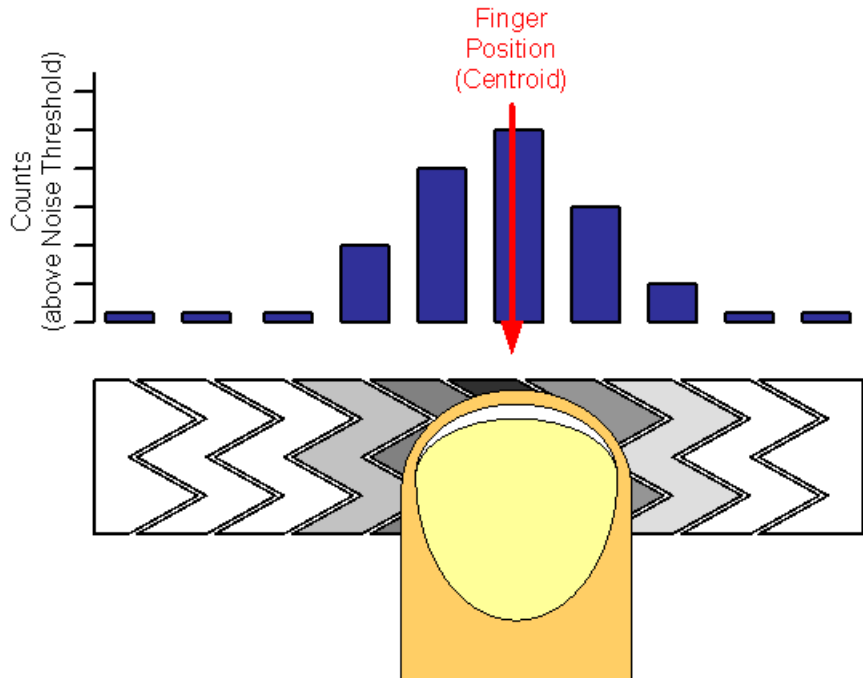


[+] Feedback

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。 一个传感器的启动将致使物理位置上相邻的传感器部分启动。 通过计算已激活传感器组的质心位置，可以确定滑条的实际位置。 通过建立一些组（每组滑条都有特定的顺序）可在 CSD 向导中对滑条进行调整。 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 设备提供的传感器位置的数值。

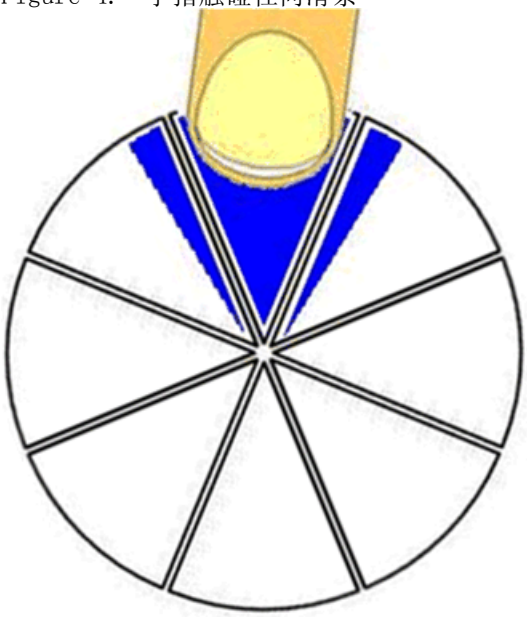
Figure 3. 对物理传感器位置排序



滑条一半内强烈信号的靠近会导致相同的电平混叠到上半部，但结果是离散的。 感应算法通过搜索强烈的相邻信号组指明解析出的滑条位置。

径向滑条

Figure 4. 手指触碰径向滑条



[+] Feedback

对于 CSD UM，可采用线性和径向这两类滑条。 径向滑条与线性滑条类似。 但是线性滑条有起始点和结束点，径向滑条却没有。 发生碰触时，质心计算算法将考虑传感器开关切换到电流开关左右两侧的次数。 径向滑条未采用双工。

CSD UM 包含两个支持径向滑条的 API 函数。 第一个函数 CSD_wGetRadiaPos() 返回质心位置，第二个函数 CSD_wGetRadialInc() 则返回以分辨率为单位的手指位移。 当手指以顺时针方向移动时，会产生正的偏移。

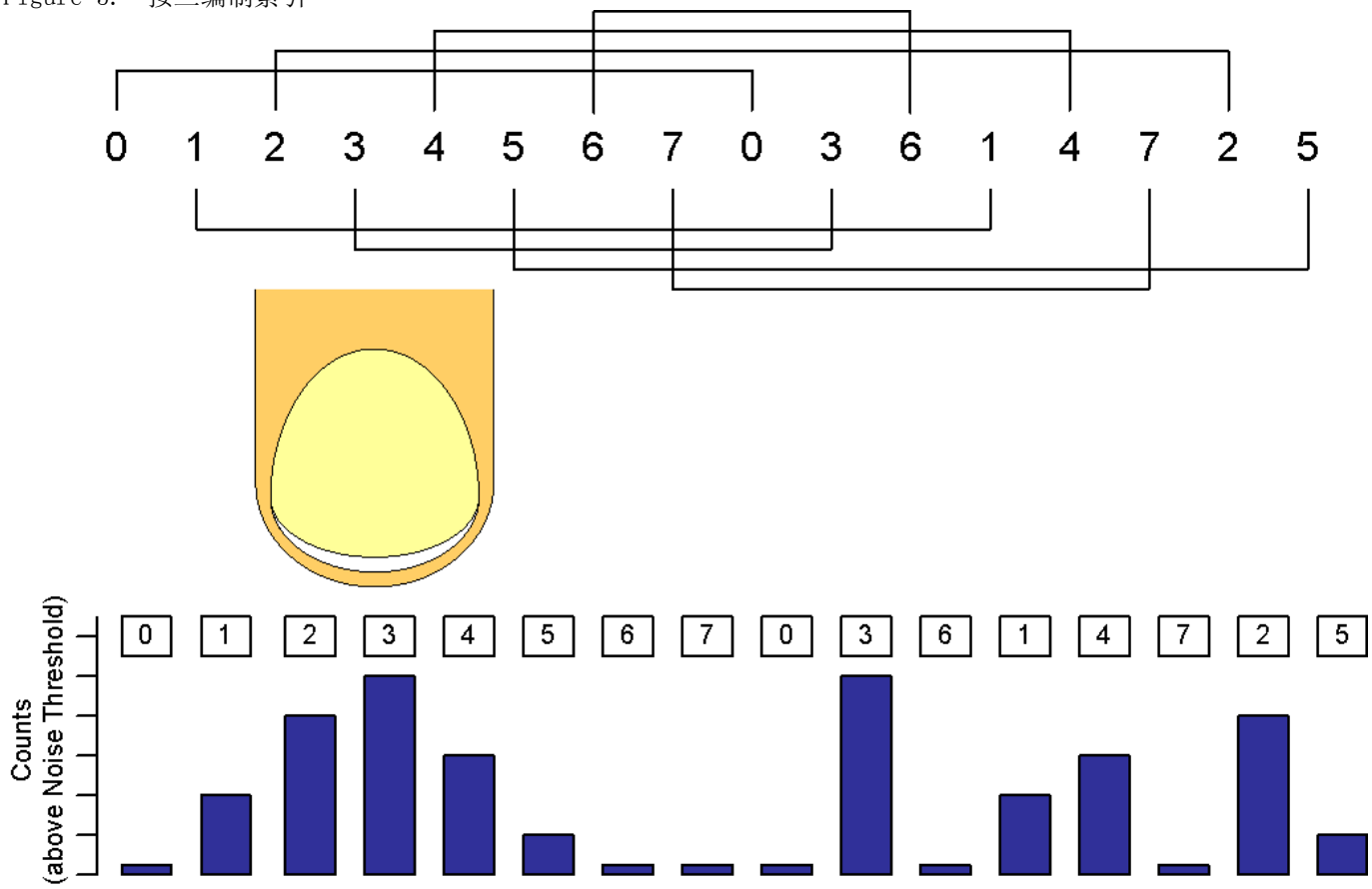
参考点 (0) 位于第一个传感器的中心。 线性和径向滑条的分辨率均限制为 3000。

双工

滑条中的每个 PSoC 传感器连接都会映射到滑条传感器阵列中的两个物理位置上。 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 CSD 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，并在包括文件中列出。 确立顺序时要做到让相邻传感器在某一半的启动不会导致另一半的邻近传感器启动。 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 5. 按三编制索引



使滑条中的传感器电容均衡。 根据传感器或 PCB 设计，某些传感器对可能需要更长的路由。 当选择双工时，双工传感器索引表由 CSD 向导自动生成。 下表展示了不同滑条段计数的双工序列。

不同滑条段计数的双工序列

[+] Feedback

滑条段总计数	段序列
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
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

[+] Feedback

滑条段总计数	段序列
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
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

[+] Feedback

滑条段总计数	段序列
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

插值和定标

在滑条传感器和触摸板应用中，经常有必要在具体传感器的本质间距以上确定出手指 （或其他电容性物体）的位置。 手指在滑条传感器或触摸板上的接触区域通常大于任何单个的传感器。

为了采用一个质心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。 这要求一定数量的相邻传感器信号大于某一噪声阈值。 在找到最为强烈的信号后，此信号和那些大于噪声阈值的邻近信号均用于计算质心。 按以下形式计算质心时，最少要用两个，最多要用八个 （典型情况下）传感器来计算质心位置：

Equation 1

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

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质心 （例如对于 12 个传感器为 0 到 100 的范围），要将质心值乘以计算得出的标量。 将插值和定标操作结合到同一项计算内并在想要的标量内直接报告其结果，可以达到更高的效率。 这一过程可以在高层级 API 内进行处理。

滑条传感器计数和分辨率在 CSD 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。

质心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

[+] Feedback

使用以下公式计算分辨率：

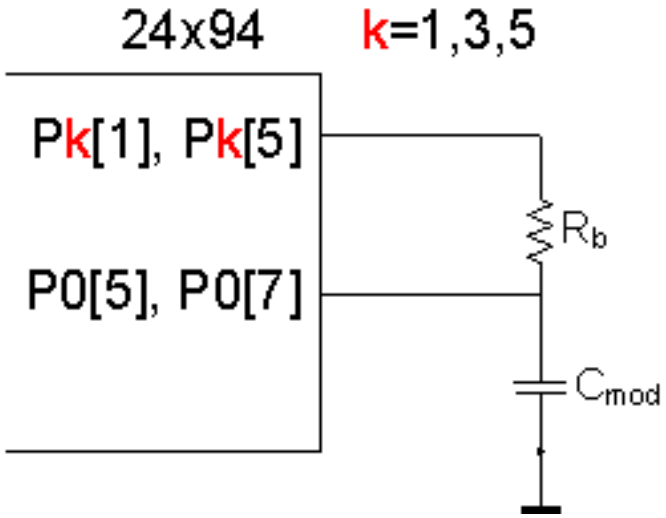
分辨率 = （传感器数 – 1）x 乘数

质心值限制为 24-bit 的无符号整数，而且其分辨率是传感器数量和乘数的函数。

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。

Figure 6. 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。 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。 通过以下方法选择电阻值：

- 监控不同传感器触摸的原始计数。
- 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%。 当电阻值升高时，原始计数增大。

典型值为 500Ω – 10 kΩ，具体取决于传感器电容和预充电开关操作频率。 如果使用 CY3214 评估板，则可以从 2.0kΩ 开始。

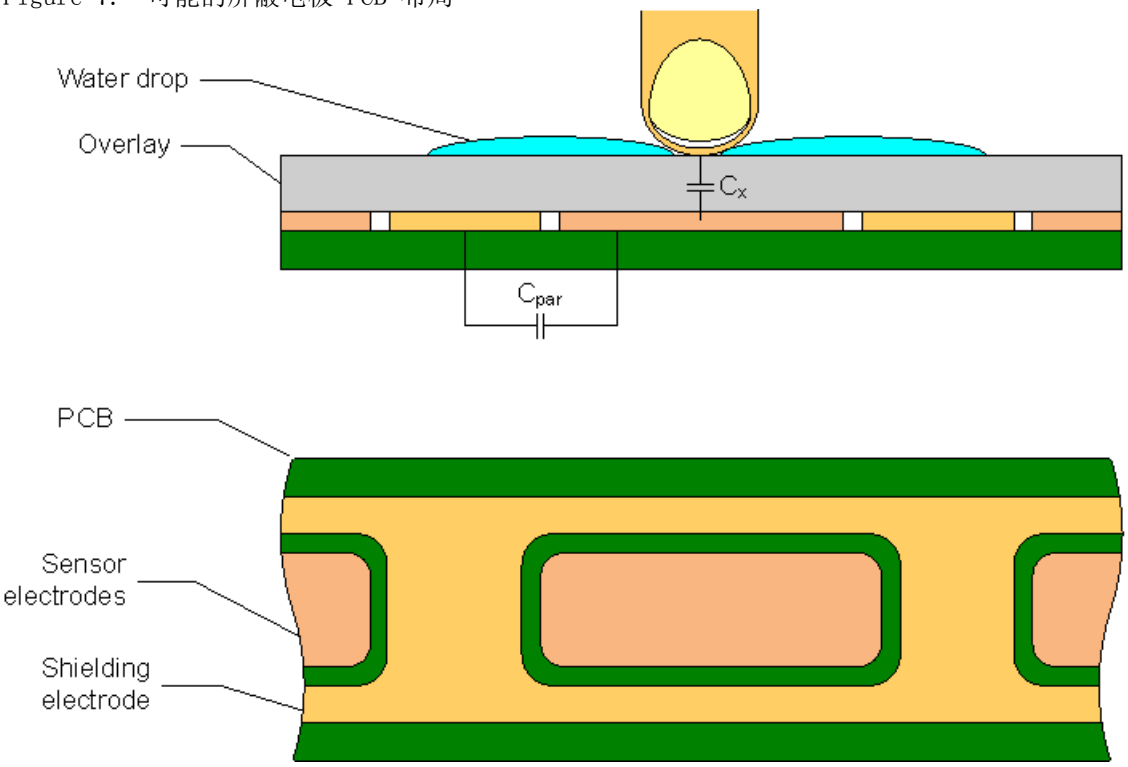
[+] Feedback

屏蔽电极

某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。 该电极位于感测电极的后方或外侧。 如果设备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。 屏蔽电极有助于减小寄生电容的影响，从而使处理感测电容变化的范围更加动态。

在某些应用场合，选择屏蔽电极信号及其相对于感测电极的位置十分有用，这样，增强两个电极之间的耦合就会造成感测电极电容测量的触摸变化减弱。 这样可以简化高层级软件 API 的工作。 CSD 用户模块支持屏蔽电极的单独输出。

Figure 7. 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。 也可将屏蔽电极置于 PCB 层的相对面，包括按键下方的面。在这种情况下推荐使用填充样式，填充率约为 30% 至 40%。 这时无需额外的接地层。

当屏蔽电极和感测电极之间出现水珠时，C_{par} 会增加而调制器电流会减弱。 在实际测试中，调制器参考电压可能会由 API 增加，因此水珠引发的原始计数增加量可能接近于零或出现较小的负值。 您可通过选择适当的调制器参考值来达到此目标。

屏蔽电极可以连接到任何空闲行输出总线。 对于行 LUT 函数，您应选择 **A**。

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 设备与屏蔽电极之间连接上升限制电阻。

[+] Feedback

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三个选项作为预充电开关的时钟源：

- 16-bit 伪随机序列发生器 (PRS16)
- 8-bit PRS 源
- 带有预分频器的 8-bit PRS 源

当第一次选择用户模块时，应当选择必需的配置。 稍后可以通过右键单击 “ 互连视图 ” 中的 “CSD 用户模块 ” 图标并选择**用户模块选择选项**来更改此选择。

PRS16 配置将 PRS16 模块用作时钟源。 PRS16 源提供扩频操作，确保很好地抵抗外部噪声源的噪声。 另外，带有扩频时钟的设计能够达到较低的电磁辐射级别。 如果应用的目标是通过 EMC/EMI 测试或者必须在嘈杂环境下提供可靠操作，则建议使用 PRS16 配置。 下表对三种配置进行了比较：

配置	操作频率	抗 EMC 噪声能力
PRS16	扩频，平均值为 $F_{IMO}/4$，峰值为 $F_{IMO}/2$	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	扩频，平均值为 $F_{IMO}/4$，峰值为 $F_{IMO}/2$	中等。 由于 PRS 重复周期较短，有更多敏感点。
带预分频器的 PRS8	可调整扩频， 平均值为 $F_{IMO}/8$ - $F_{IMO}/1024$， 峰值为 $F_{IMO}/4$ - $F_{IMO}/512$	中等。 由于 PRS 重复周期较短，有更多敏感点。

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。 调制器参考是使用电阻分频器形成的，位于连续时间模块中。 可以将参考值作为 UM 参数进行更改或使用 API 调用进行更改。

直流和交流电气特性

Table 1. 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	

[+] Feedback

Table 2. 噪声

参数[a]	最小值	典型值	最大值	单位	测试条件（Vdd = 3.3V、SysClk = 24 MHz、CPU 时钟 = 6 MHz、基准线值 >= 分辨率最大计数的 70%）
噪声计数，峰 – 峰		0.2		%（噪声计数）/（基准线值计数）	分辨率 = 16
噪声计数，峰 – 峰		1		%（噪声计数）/（基准线值计数）	分辨率 = 14
噪声计数，峰 – 峰		10		%（噪声计数）/（基准线值计数）	分辨率 = 10

a. 当扫描速度减慢且基准计数增加时，信噪比提高。

Table 3. 功耗

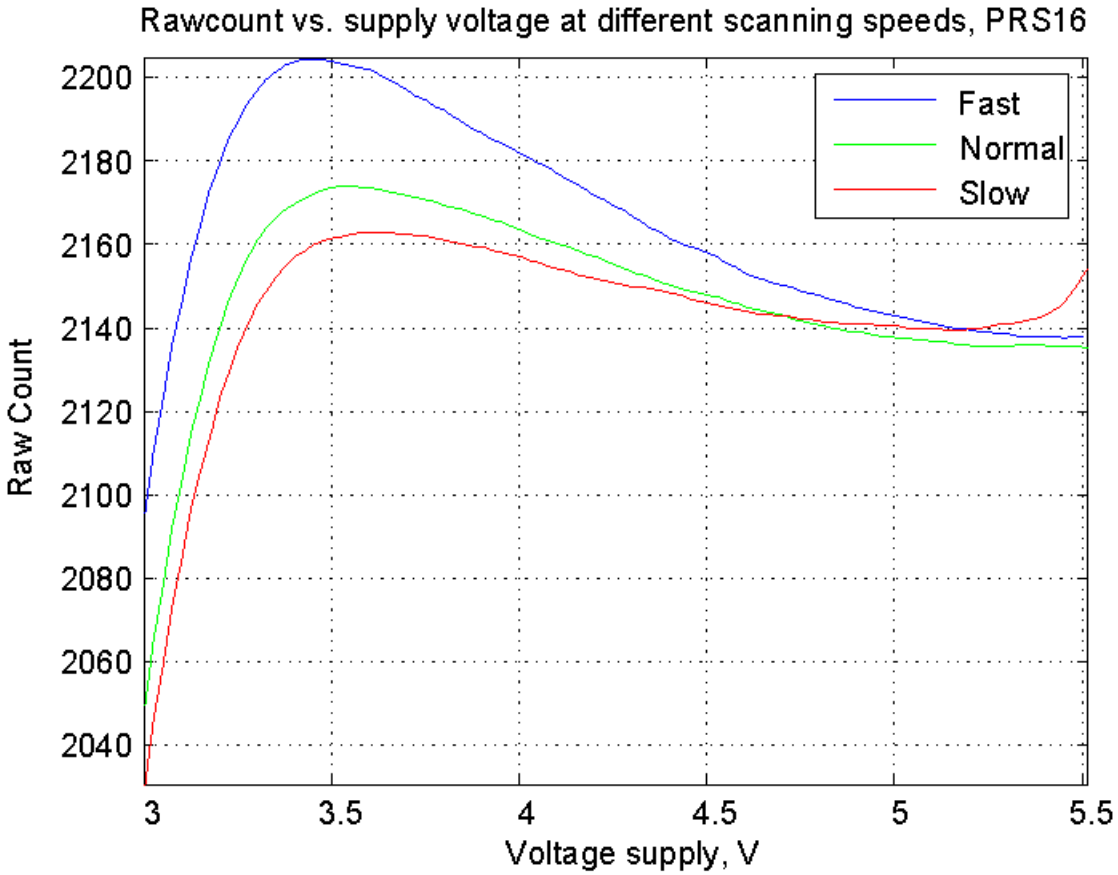
供电电压	最小值	典型值	最大值	单位	测试条件和注释
有功电流		10		mA	扫描期间平均电流，8 个传感器
待机电流		250		µA	扫描速度 = 超快，分辨率 = 9,100 ms 报告速率，8 个传感器
		1.6		mA	扫描速度 = 快速，分辨率 = 12,100 ms 报告速率，8 个传感器
睡眠 / 唤醒电流		10		µA	1s 报告速率，1 个传感器

[+] Feedback

特征图

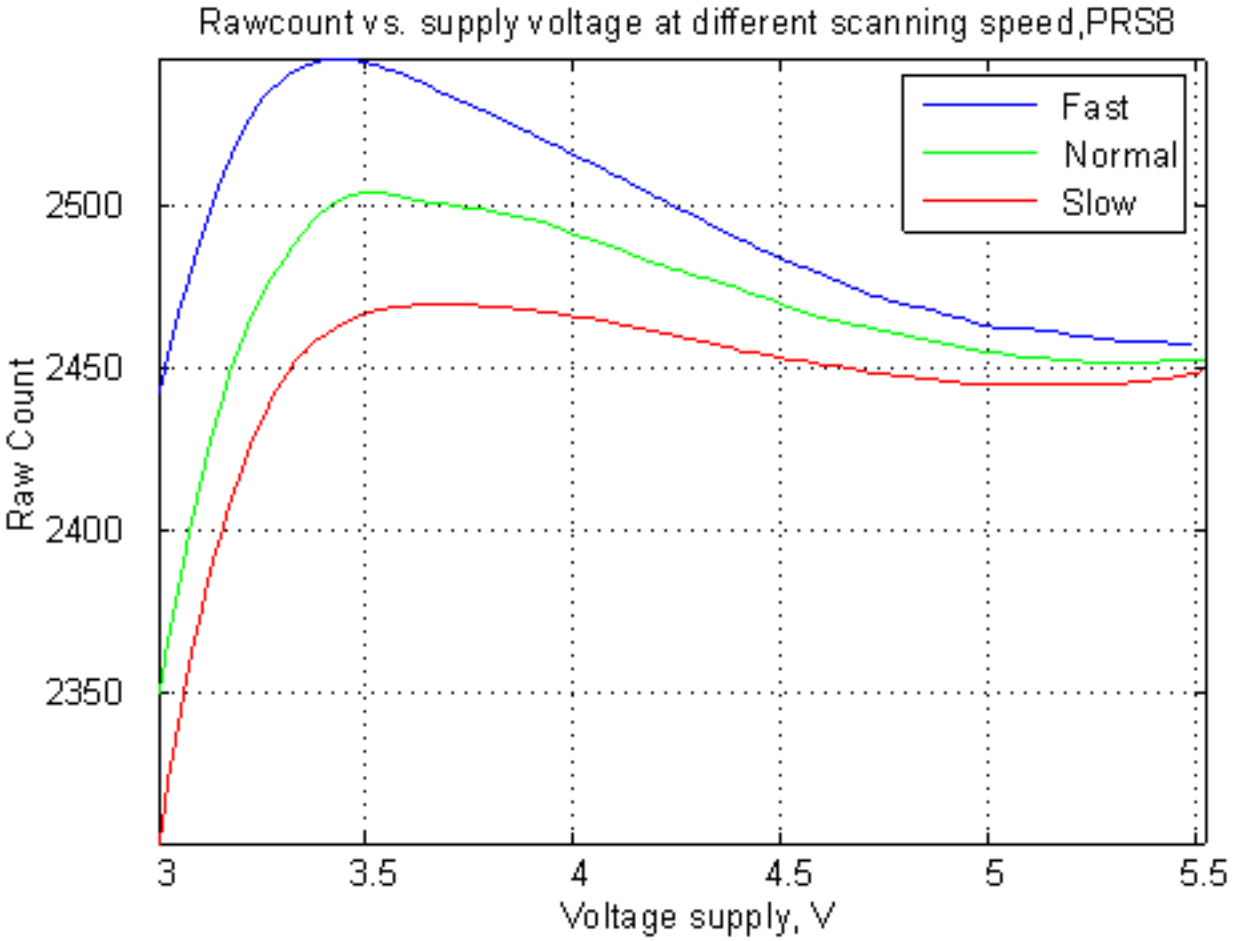
测试条件： 分辨率设置为 12 位，使用 CY3214 修订版收集数据。 一块评估板。 有关详细信息，请参见注释。

Figure 8. 配置为 PRS16 时，在不同扫描速度下，原始计数与供电电压的关系



[+] Feedback

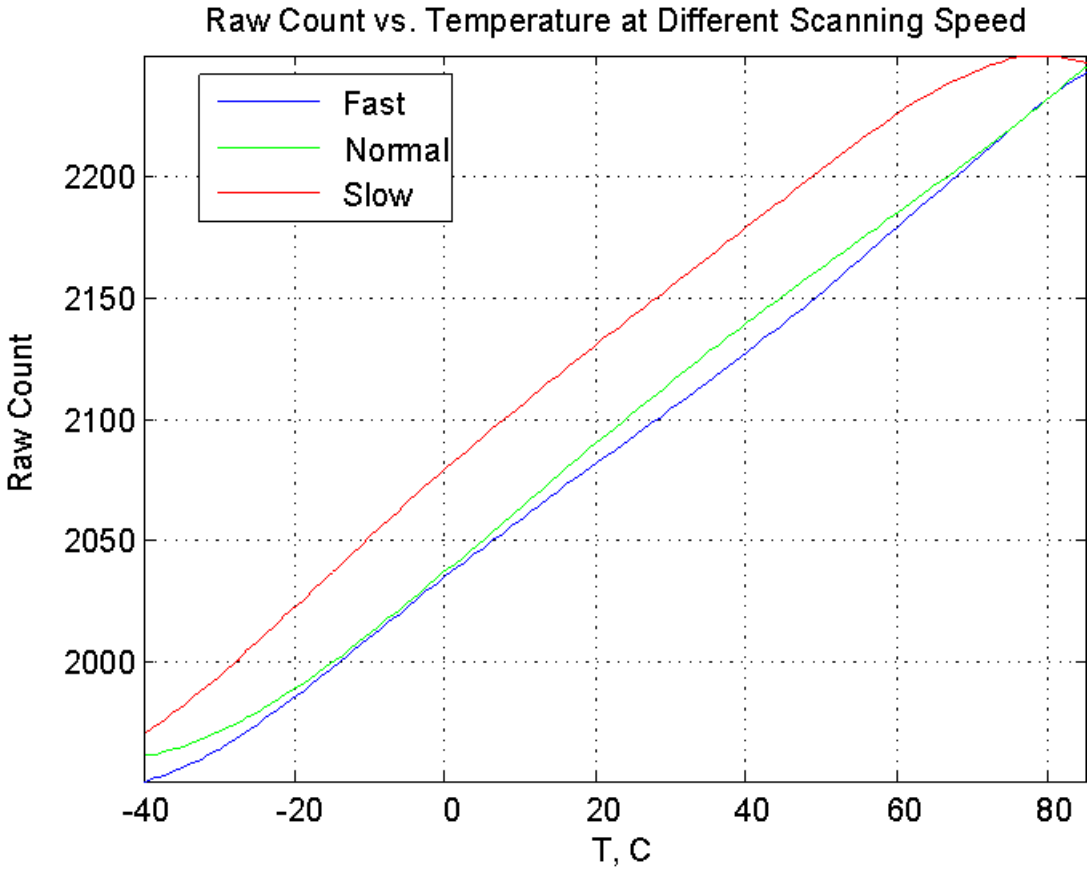
Figure 9. 配置为 PRS8 时，在不同扫描速度下，原始计数与供电电压的关系



注：通过比较曲线可以发现，在扫描速度较慢时原始计数变化较。 对于在多种供电电压范围下要求操作稳定的应用，推荐使用慢速扫描。

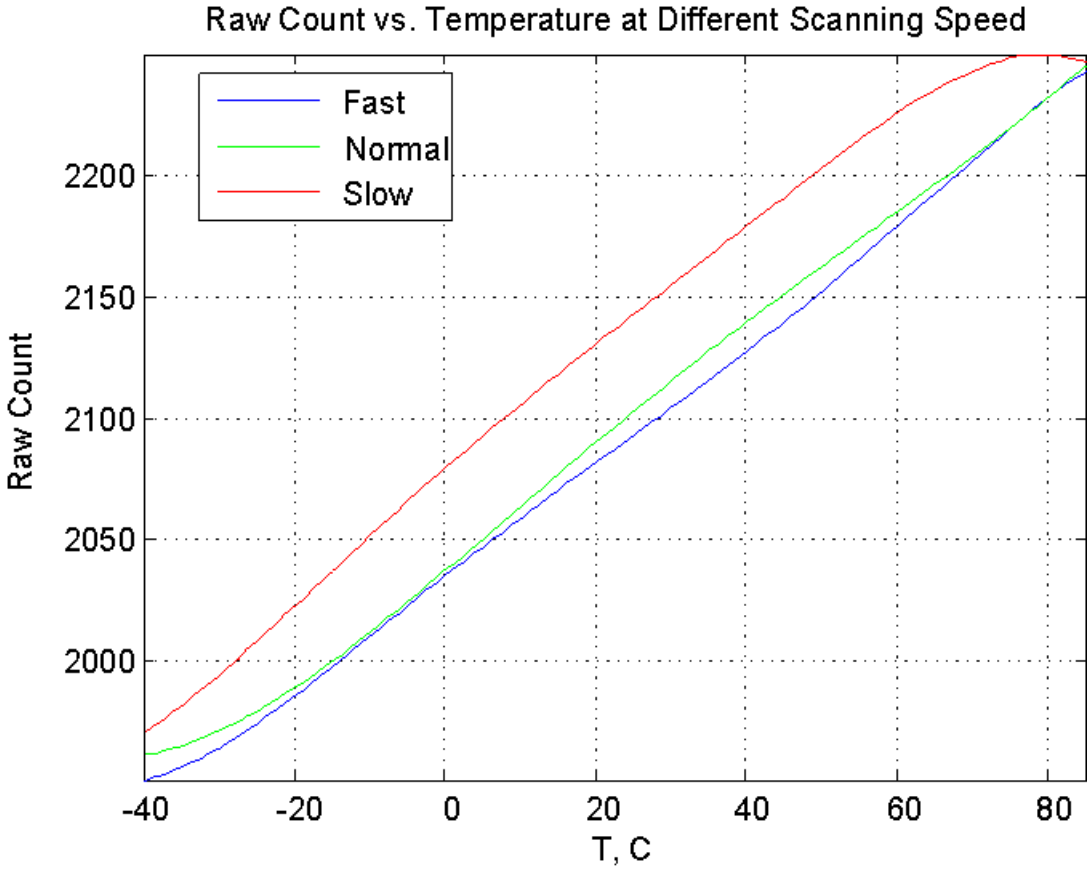
[+] Feedback

Figure 10. 配置为 PRS16 时，在不同扫描速度下，原始计数与温度的关系



[+] Feedback

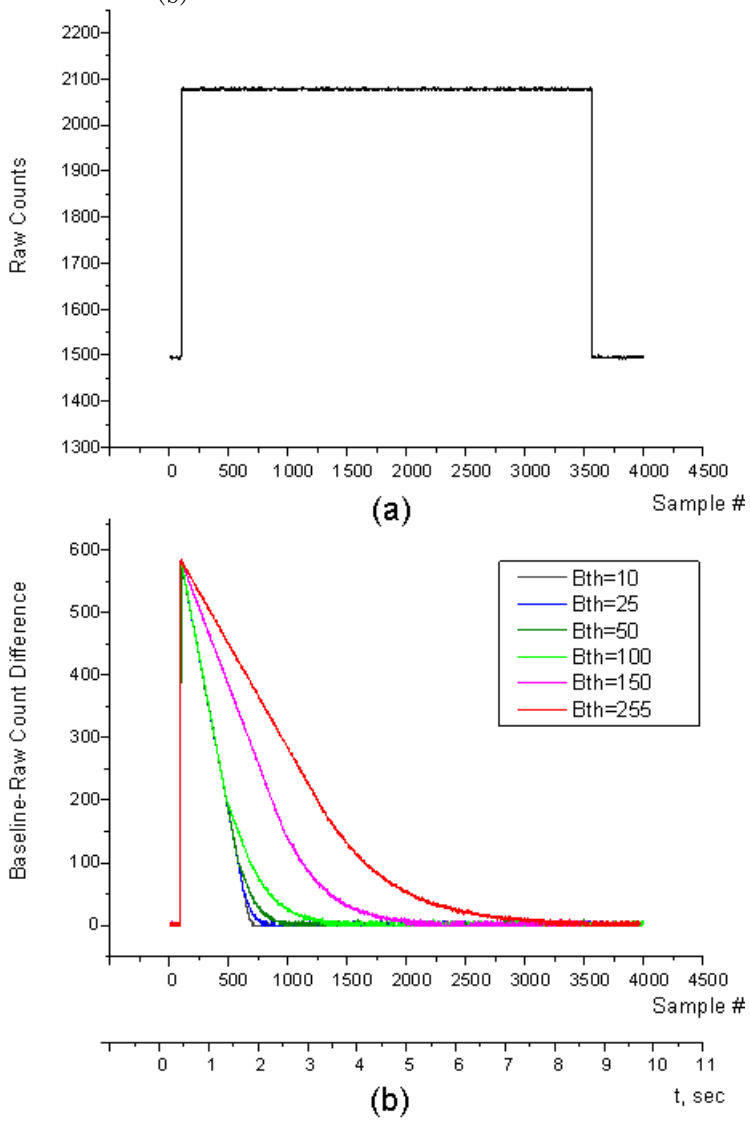
Figure 11. 配置为 PRS8 时，在不同扫描速度下，原始计数与温度的关系



[+] Feedback

测试条件：12-bit 分辨率，电源电压为 5.0V。在测试过程中将带有传感器的完整电路板 （CY3214 PSoC 评估板）置于温控室中。

Figure 12. 原始计数步长变化 (a) 以及不同 “BaselineUpdate 阈值 ”(Bth) 参数值的原始计数与基准线值之差 (b)

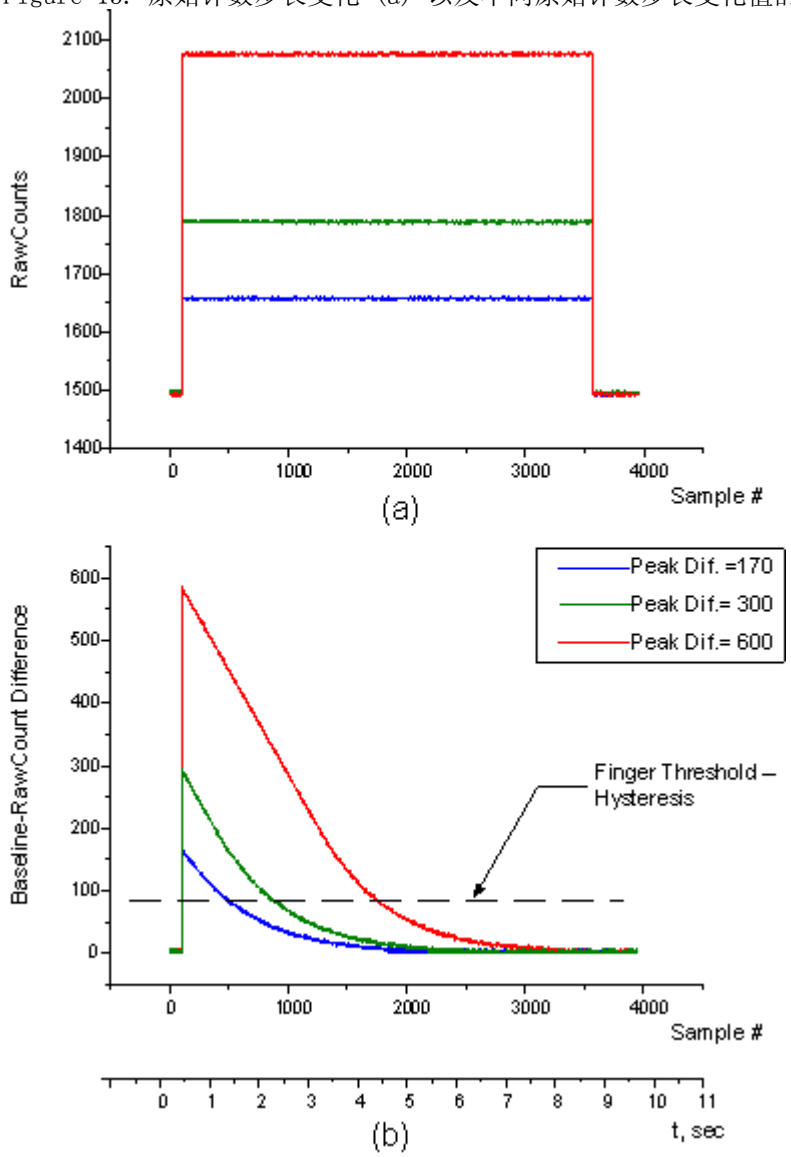


测试条件：12-bit 分辨率，SensorsAutoreset = 已启用，传感器阵列扫描和数据传输时间总计大约为 2.5 ms。

[+] Feedback

说明：增大 “BaselineUpdate 阈值 ” 会降低差的减小程度，使最大按键触摸检测时间加长。

Figure 13. 原始计数步长变化 (a) 以及不同原始计数步长变化值的原始计数与基准之差 (b)。



测试条件：12-bit 分辨率，SensorsAutoreset = 已启用，传感器阵列扫描和数据传输时间总计大约为 2.5 ms。 “BaselineUpdate 阈值 ” 参数设置为 255。

注：原始计数步长值越大，传感器自动复位启动所需的时间越长，从而需要更长时间才能将差降低到 FingerThreshold-Hysteresis 值之下。

[+] Feedback

放置

用户模块所用的模块将在用户模块实例化后自动布置，而没有提供备选布置方式。 调制器比较器位于 ACB01 连续时间模块。 不同的 UM 配置使用 1-3 个数字模块。 下表汇总了使用的数字资源。

配置	使用的数字模块
PRS16	DBB00、DBB01 和 DCB0
PRS8	DBB00
带预分频器的 PRS8	DBB00, DBB01

未使用的模拟和数字模块可自行支配。 所有的 UM 配置均使用硬件抽取滤波器。

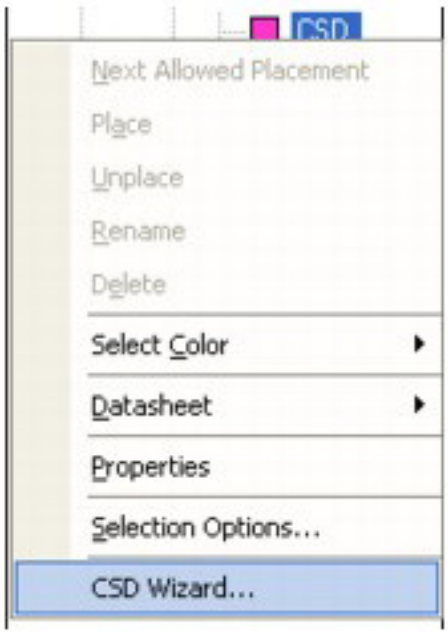
必须在建立 CSD 用户模块的端口引脚连接之前，放置使用特定引脚资源 （包括 LCD 和 I2CHW）的用户模块。 当打开向导时，向导中会显示配置选择。

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路由电容值，从而会影响传感器的灵敏度和噪声。

向导

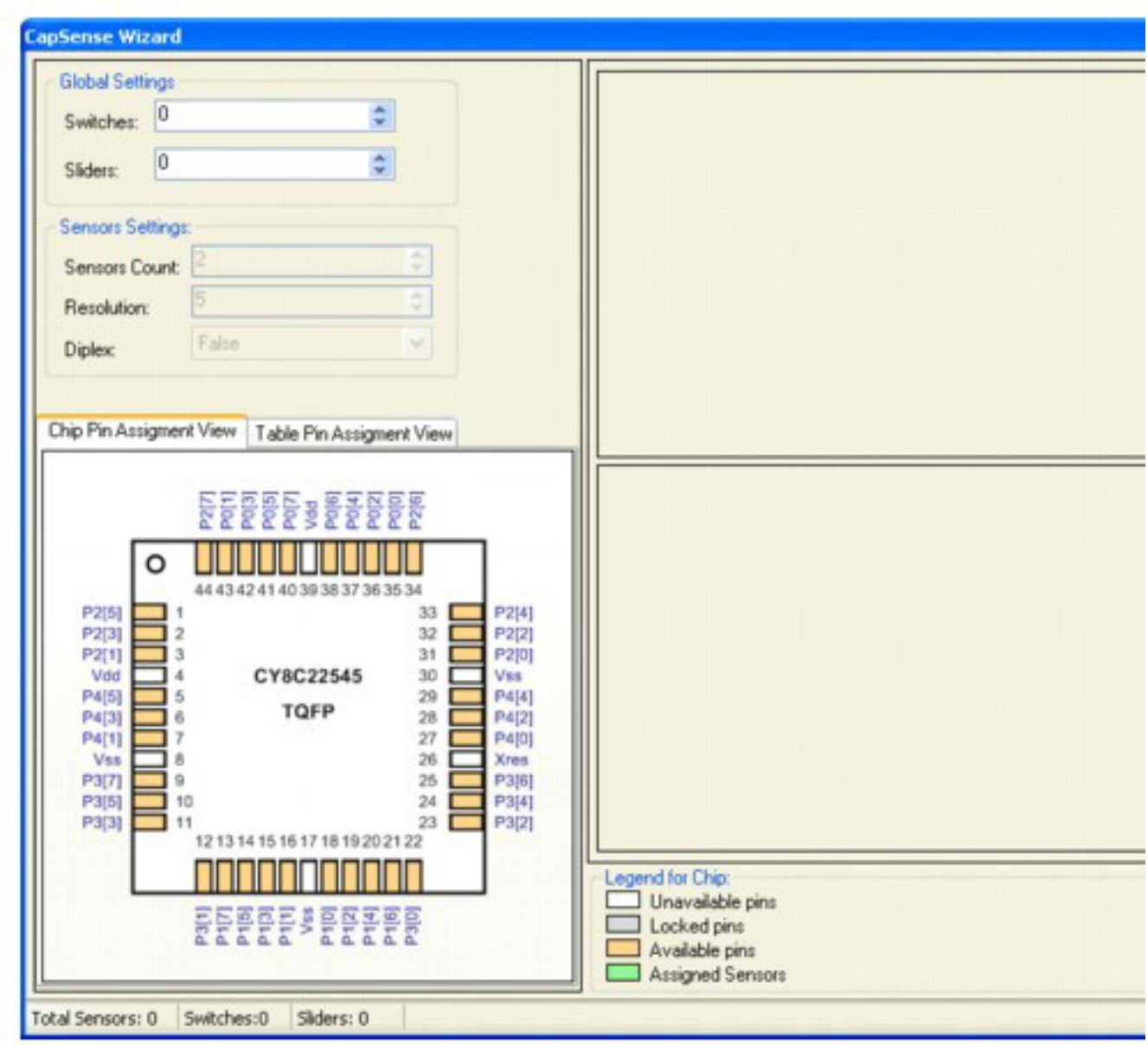
CSD 向导用来设置 CapSense 按键、滑条和接近传感器的引脚输出。 可以选择所需的配置，使用拖放界面分配按键和区段。

1. 要访问向导，请在 “ 器件编辑器互连视图 ” 中右键单击任意 CSD 块，然后通过鼠标左键单击选择 “CSD 向导 ”。



[+] Feedback

2. 打开的向导将显示数字输入框，可在其中输入传感器数量和滑条传感器数量。



向导引脚注释

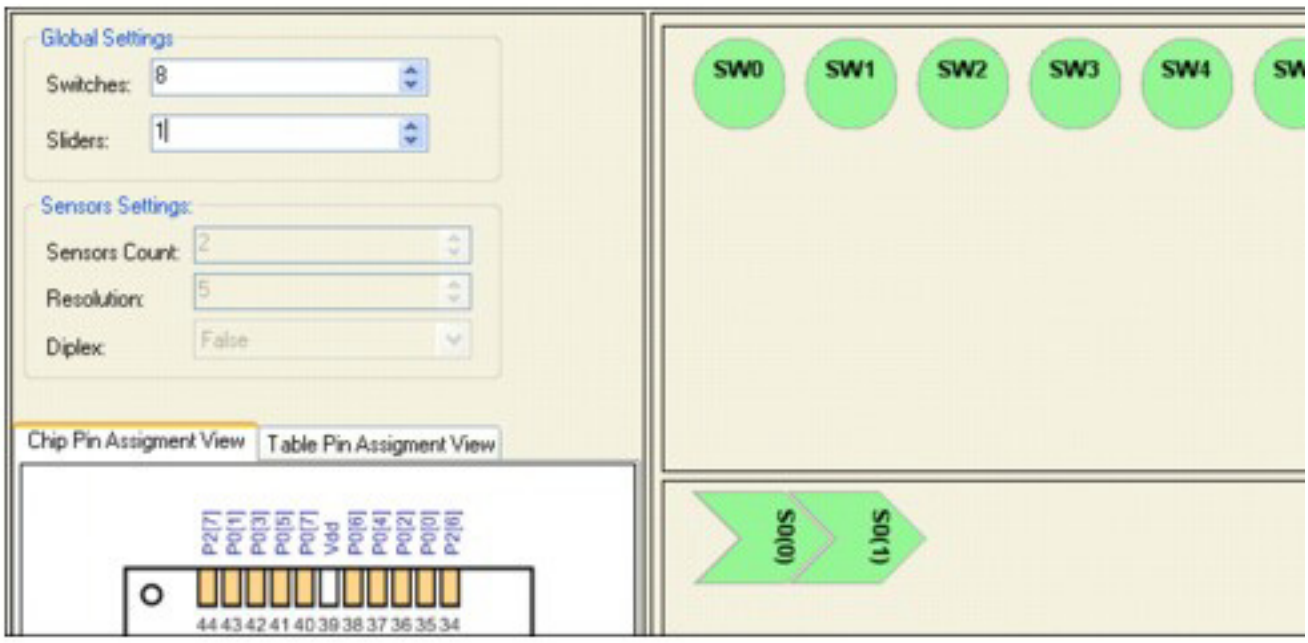
- 白色 – 引脚不可用作 CapSense 输入端。
- 灰色 – 引脚已锁定。产生此情况的可能原因有两种。第一种可能性是另一个用户模块，例如 LCD 或 I^2C 占用了该引脚。第二种可能性是引脚已更改为使用非默认名称。要恢复使用引脚的默认名称，请在 “ 引脚分布 ” 视图中展开引脚，然后从**选择**菜单中选择**默认**。现在可在向导中分配引脚了。
- 橙色 – 引脚可分配。
- 绿色 – 引脚已分配为 CapSense 输入端。

[+] Feedback

3. 键入独立传感器的数量。 传感器数量限制为可用引脚的数量。

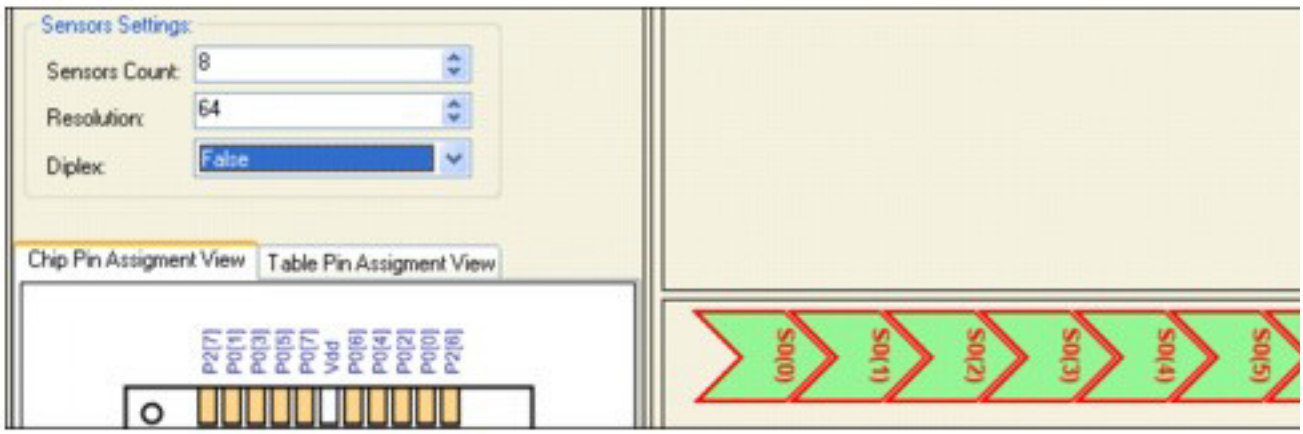


4. 键入滑条的数量。

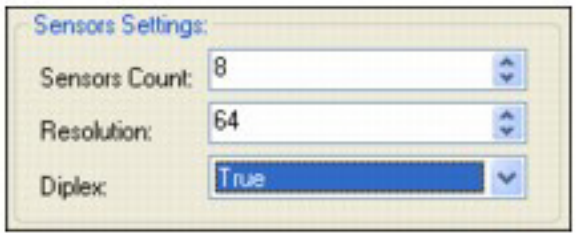


[+] Feedback

5. 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。



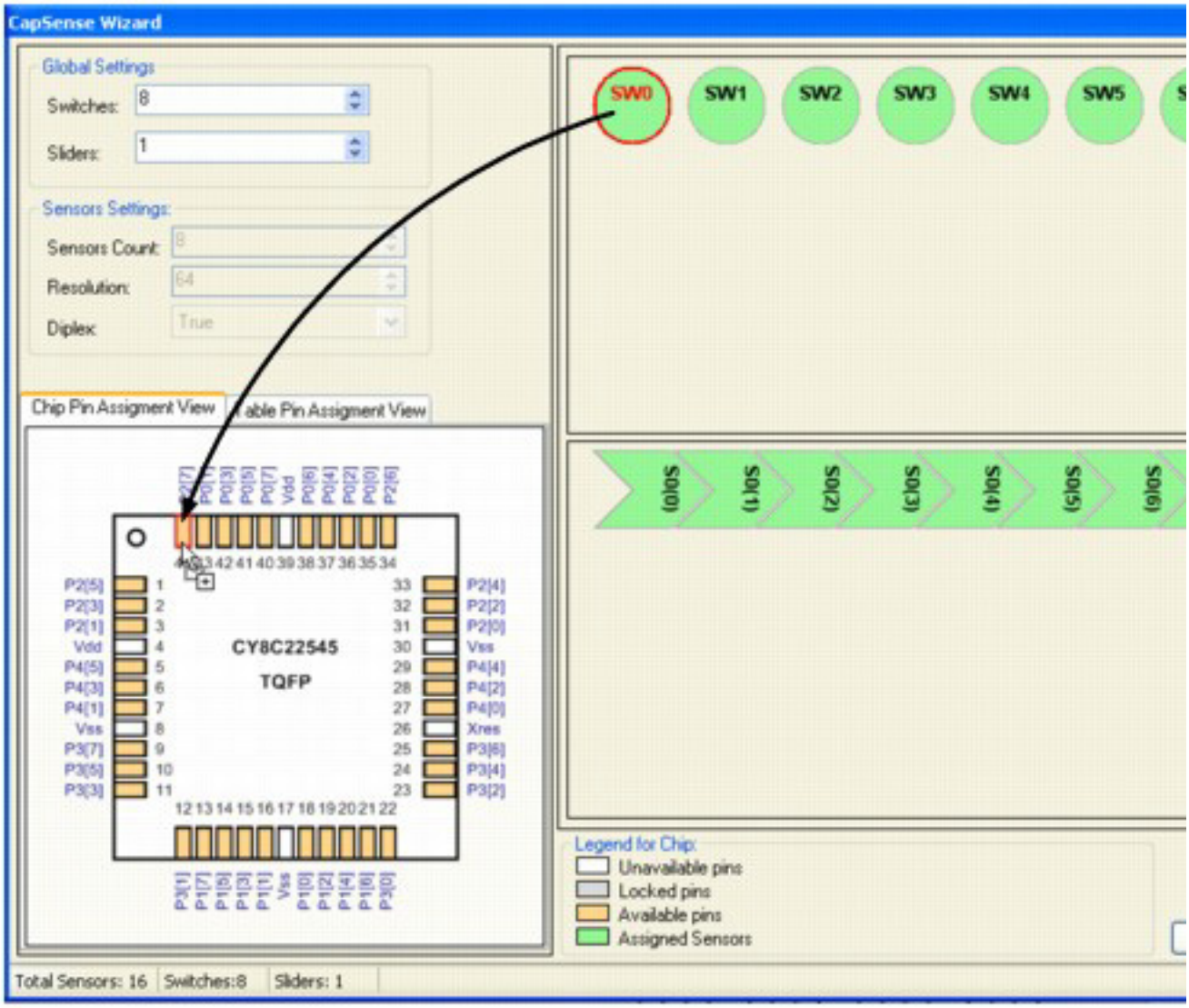
6. 键入输出分辨率。 最小值为五。 对于双工型滑条，最大值为（传感器的引脚数 - 1）x 2^{16} - 1 或（2 x 传感器的引脚数 - 1）x 2^{16} -1。



7. 如果需要，请选择 “ 双工 ”。 这样会将选定用于传感器的引脚数量映射成电路板上传感器位置数量的两倍。 仅显示了双工传感器的前一半部分；后一半部分按前面 “ 双工 ” 章节所述自动映射。 有关引脚连接的双工表，请参见 “ 双工 ” 一节。

[+] Feedback

8. 左键单击传感器，将其拖动到任意可用引脚。 端口引脚在选定后将变为绿色，无法再用于分配。 通过将传感器拖离端口引脚，更改传感器分配。



9. 对其他独立传感器重复以上操作。
10. 将单个滑条传感器映射到物理端口引脚的方法与映射单个传感器的方法相同。
11. 单击**确定**接受数据，然后返回到 PSoC Designer。

这时就完成了传感器布置操作。 在 “ 器件编辑器 ” 窗口中右键单击，选择**刷新**更新引脚连接。

设置用户模块参数并生成应用程序。 如果愿意，您现在可以调整示例项目。

在 CSD 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。

要更改引脚分配情况，请将光标置于已分配的引脚上，单击此引脚，然后将其拖到开关框外。 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、双工和分辨率，会生成一系列表格。 表格位于 CSD_Table.asm 中

[+] Feedback

传感器表格

传感器表格包含每个传感器的 2-byte 条目。 第一个字节是端口编号，第二个字节是位的位掩码 （而非位的编号）。 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：

```
CSD_Sensor_Table:
_CSD_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 CSD_wGetPortPin() 子程序使用。

组表格

组表格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。 每个条目有六字节。 第一个字节是传感器表格中该组起始处的索引。 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采用了双工 （4 表示已采用双工，0 表示未采用）。 第四、五、六个字节是固定值乘数，将其与计算出的滑条质心相乘可以获得 CSD 向导中所需的分辨率。

```
CSD_1_Group_Table:
_CSD_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
```

双工表格

当某个组是滑条且采用双工时，将为该组生成双工表格扫描顺序数据。 否则，就会只创建标签但不包含任何数据。 此表格包括两部分：每个滑条的传感器映射，以及每个单独滑条与对应表格的参照。 此处展示了一个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


 CSD_1_Diplex_Table:
_CSD_1_Diplex_Table:
    db >DiplexTable_0, <DiplexTable_0
    db >DiplexTable_1, <DiplexTable_1
```


[+] Feedback

参数和资源

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 ” 中默认值设置。 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单个传感器的数量。）

可能值的范围为 5 到 255。

噪声阈值

对于单个传感器，高于此阈值的计数值不会使基准线值更新。 对于滑条传感器，质心计算中不考虑低于此阈值的计数值。 可能值为 5 到 255。

基准线值更新阈值

如果新的原始计数值高于当前基准线值，差值低于噪声阈值 （传感器自动复位参数设置为 “ 禁用 ”），则当前基准线值与原始计数的差值累计到桶形变数 (″水桶″) 中。 当水桶充满时，基准线值按某个值递增，并清空水桶。 此参数用于设置为了使水桶必须达到基准线值而要增大的阈值。 可能值为 0 到 255。 参数值越大，基准线值更新速度越慢。 如果需要更频繁的基准线值更新，请减小此参数。

LowBaselineReset

LowBaselineReset 参数与 NegativeNoiseThreshold 参数协同工作。 对于指定数量的样品，如果样品计数值低于基准线值与 NegativeNoiseTreshold 的差值，则基准线值设置为新的原始计数值。 此参数实际上是对需要复位基准线值的异常低的样品数值进行计数。 它通常用于更正启动时手指位置的情况。

传感器自动复位

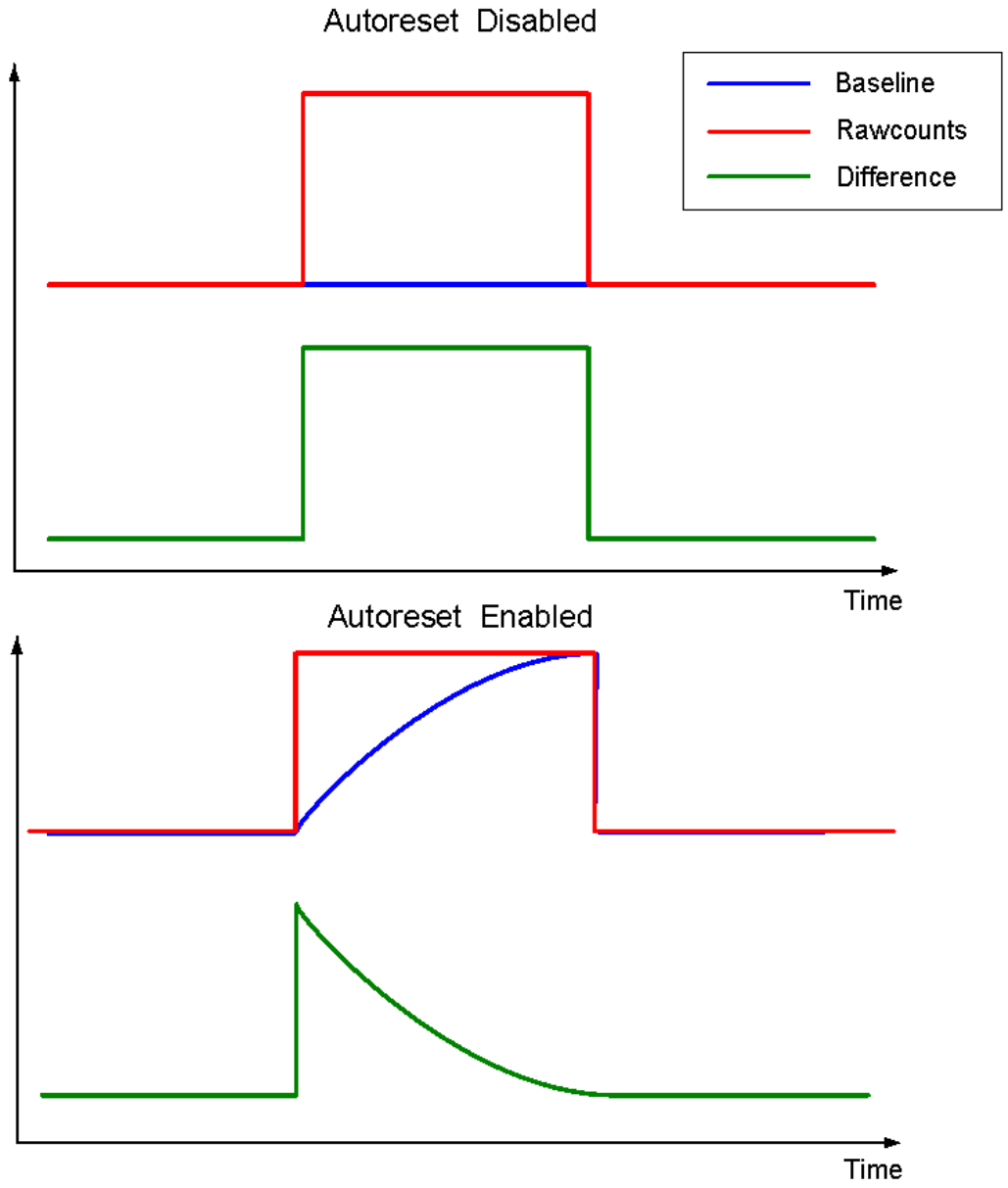
此参数决定了基准线值是随时更新，还是只在信号差值低于 “ 噪声阈值 ” 时更新。 设置为**启用**时，基准线值将持续更新。 此设置限制传感器的最大持续时间 （典型值为 5 - 10s），但它防止了没有任何物体触碰到传感器时原始计数突然上升所导致的传感器永久开启的现象。 这种突然上升的原因可能是较大的电源电压波动、高能量的射频噪声源或极快的温度变化。

在此参数设置为**禁用**时，基准线值只在原始计数与基准线值的差值低于 “ 噪声阈值 ” 参数时更新。 除非要求将传感器长时间保持在启用状态，否则您应当使此参数处于 “ 禁用 ” 状态。

[+] Feedback

下图展示了此参数对基准线值更新的影响。

Figure 14. 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，则差值计数必须大于手指阈值与迟滞的和。 如果传感器处于活动状态，则差值计数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防反跳和粘着性程度。 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 ” 参数设置。

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。

[+] Feedback

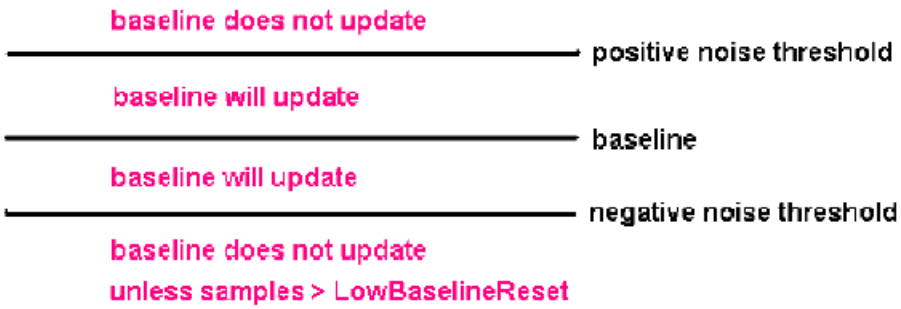
防反跳

“ 防反跳 ” 参数添加了一个用于传感器活动状态切换的防反跳计数器。 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，差计数值必须大于手指阈值与迟滞之和。 防反跳计数器的值随着 bIsSensorActive 或 bIsAnySensorActive API 函数递增。

可能值为 1 到 255。 设置为 1 时则不提供防反跳功能。

NegativeNoiseThreshold

NegativeNoiseThreshold 参数增加负的差值计数阈值。 如果当前原始计数低于基准线值且它们的差大于此阈值，则不更新基准线值。 但是，如果对于 LowBaselineReset 参数所指定的样品数量，当前原始计数处于较低状态 （差大于阈值），则对基准进行复位。



LowBaselineReset

LowBaselineReset 参数与 NegativeNoiseThreshold 参数协同工作。 对于指定数量的样品，如果样品计数值低于基准线值与 NegativeNoiseTreshold 的差值，则基准线值设置为新的原始计数值。 此参数实际上是对需要复位基准线值的异常低的样品数值进行计数。 它通常用于更正启动时手指位置的情况。

扫描速度

此参数影响传感器的扫描速度。 可用的选择有**快速**、**正常**和**慢速**。 较慢的扫描速度具有下列好处：

- 提高了信噪比
- 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）。 PRS16 配置可以使用分辨率为 9 到 16 位的扫描传感器。 带有预分频器配置的 PRS8 和 PRS8 只能使用 8、10 和 12 位的扫描传感器。 N 位的扫描分辨率最大原始计数为 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。

VC1 分频器的值取决于扫描速度。 下表显示了扫描速度如何影响 VC1 分频器：

[+] Feedback

Table 4. VC_1 分频器值与扫描速度

扫描速度	VC1
快速	2
正常	4
慢速	8

Table 5. 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 （以位为单位）	扫描速度		
	快速	正常	慢速
9	260	510	1030
10	425	850	1710
11	770	1550	3080
12	1470	2940	5840
13	2840	5680	11400
14	5560	11200	22200
15	11100	22000	44000
16	21800	44400	88000

Table 6. 对于 24 MHz IMO 操作、PRS8 或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 （以位为单位）	扫描速度		
	快速	正常	慢速
8	85	130	260
10	130	260	510
12	260	510	1020

注：扫描时间按两次传感器扫描之间的时间间隔来测量。 此时间包括传感器设置时间、调制器稳定延迟、样品转换间隔和数据预处理时间。

调制器电容引脚

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。

反馈电阻引脚

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。

[+] Feedback

参考值

此参数设置比较器参考值。 参考来自内部电阻式电压分频器。 零对应于最小参考值 (1/4 Vdd)。 八对应于最大参考值 (3/4 Vdd)。 参数增加时，参考电压将呈线性增加。 参考值增加时，灵敏度将减小，但是对屏蔽电极的影响将变大。

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状），则可以使用 API 功能为具有较大电容的传感器设置较高的参考值，以平衡原始计数。

预分频器周期

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。

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，在低分辨率和高扫描速度情况下尤其如此。

PRS 多项式

此参数设置 PRS8 和带有预分频器配置的 PRS8 中的 PRS 多项式。 以下有两个选择：

- 短 - 短多项式设置可产生更好的信噪比，但由于重复周期较短，终端器件可能会更容易受外部噪声源的影响。
- 长 - 长多项式设置产生的信噪比较差，但是器件受噪声信号影响较小。

ShieldElectrodeOut

屏蔽电极信号源可从空闲的数字行总线 (Row_0_Output_1 - Row_0_Output_3). 中进行选择。每行输出都可以路由到三个引脚中的一个。 将行 LUT 函数设置为 A。

[+] Feedback

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，允许在更高级别上对模块进行处理。 本节详细说明了每个函数的接口，以及包含文件中所提供的相关常量。

注 ** 如同在所有用户模块 API 中，这里 A 和 X 寄存器的值可通过调用 API 函数进行更改。 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。 选择这一 “ 寄存器易失 ” 策略是因为效率原因，自 PSoC Designer 1.0 版起已强制使用此策略。 C 语言编译器自动遵循此要求。 汇编语言编程人员也必须确保他们的代码遵守此策略。 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。

提供的进入点用于初始化 CSD、启动其采样和停止 CSD。 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 CSD 前缀。 未能使用正确的实例名称是语法错误的常见原因。

API 函数使用不同的全局阵列。 不得手动更改这些阵列。 但可以因为调试而对这些值进行检查。 例如，可以使用绘图工具显示阵列内容。 有多种全局阵列：

- CSD_waSnsBaseline[]
- CSD_waSnsResult[]
- CSD_waSnsDiff[]
- CSD_baSnsOnMask[]

CSD_waSnsBaseline[] - 这是一个整数阵列，其中包含每个传感器的基准线值数据。 阵列大小等于传感器计数。 CSD_waSnsBaseline[] 阵列由下列函数更新：

- CSD_UpdateAllBaselines();
- CSD_UpdateSensorBaseline();
- CSD_InitializeBaselines().

CSD_waSnsResult[] - 这是一个整数阵列，其中包含每个传感器的原始数据。 阵列大小等于传感器计数。 CSD_waSnsResult[] 数据由下列函数更新：

- CSD_ScanSensor();
- CSD_ScanAllSensors().

CSD_waSnsDiff [] - 这是一个整数阵列，其中包含每个传感器原始数据与基准线值数据之间的差值。 阵列大小等于传感器计数。

CSD_baSnsOnMask[] - 这是一个保持传感器打开或关闭状态的字节阵列 （用于按键或滑条）。 CSD_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）。 CSD_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量足以包含所有放置的传感器。 如果按键开启，则位的值为 1；如果按键关闭，则位的值为 0。 CSD_baSnsOnMask[] 数据由 CSD_blsSensorActive(BYTE bSensor) 函数或 CSD_bIsAnySensorActive() 子程序更新。

[+] Feedback

CSD_Start

说明：

初始化寄存器并启动用户模块。 此函数应在调用其他任何用户模块函数前调用。

C 语言原型：

```
void  CSD_Start()
```

汇编语言：

```
lcall  CSD_Start
```

参数：

无

返回值：

无

副作用：

**

CSD_Stop

说明：

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

C 语言原型：

```
void  CSD_Stop()
```

汇编语言：

```
lcall  CSD_Stop
```

参数：

无

返回值：

无

副作用：

**

CSD_ScanSensor

说明：

扫描所选传感器。 每个传感器在传感器阵列中都有唯一编号。 此编号由 CSD 向导按顺序分配。 Sw0 为传感器 0，Sw1 为传感器 1，依此类推。

C 语言原型：

```
void CSD_ScanSensor(BYTE bSensor)
```

汇编语言：

```
mov A,  bSensor
lcall CSD_ScanSensor
```


[+] Feedback

参数：

A => 传感器编号

bSensor - 传感器编号

返回值：

无

副作用

**

CSD_ScanAllSensors

说明：

通过调用每个传感器索引的 CSD_ScanSensor()，扫描所有已配置的传感器。

C 语言原型：

```
void CSD_ScanAllSensors()
```

汇编语言：

```
lcall CSD_ScanAllSensors
```

参数：

无

返回值：

无

副作用

**

CSD_UpdateSensorBaseline

说明：

针对每个传感器独立计算的历史计数值称为传感器的基准线值。 此基准线值使用 “ 水桶方法 ” 进行更新。

“ 水桶方法 ” 使用下列算法：

1. 每次调用 CSD_UpdateSensorBaseline() 时，通过从原始计数值中减去以前的基准来计算差值计数。 此差值存储在 CSD_waSnsDiff[] 阵列中并提供给您。

2. 如果禁用 “ 传感器自动复位 ”，则每次调用 CSD_UpdateSensorBaseline() 时，差值计数与噪声阈值进行比较。 如果差值低于噪声阈值，则将被累计到虚拟水桶中。 如果差值高于噪声阈值，则不更新水桶。 如果 “ 传感器自动复位 ” 已启用，则无论噪声阈值参数如何，差值都将累计到虚拟水桶中。

3. 虚拟水桶中的累计差值计数达到 BaselineUpdateThreshold 后，基准线值加一，水桶复位为 0。

4. 如果差值计数低于噪声阈值，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 NoiseThreshold 的元素。

C 语言原型：

```
void CSD_UpdateSensorBaseline(BYTE bSensor)
```


[+] Feedback

汇编语言：

```
mov   A,   bSensor
lcall  CSD_UpdateSensorBaseline
```

参数：

A => 传感器编号

bSensor - 传感器编号

返回值：

无

副作用：

**

CSD_UpdateAllBaselines

说明：

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

C 语言原型：

```
void  CSD_UpdateAllBaselines()
```

汇编语言：

```
call CSD_UpdateAllBaselines
```

参数：

无

返回值：

无

副作用：

**

CSD_bIsSensorActive

说明：

与手指阈值进行比较，检查给定传感器的差值计数阵列。 迟滞考虑在内。 根据传感器当前是否开启，手指阈值中将添加或减去迟滞值。 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。 此函数还会更新 CSD_baSnsOnMask[] 阵列中的传感器位。

C 语言原型：

```
BYTE  CSD_bIsSensorActive(BYTE bSensor)
```

汇编语言：

```
mov  A,  bSensor
call CSD_bIsSensorActive
```

参数：

bSensor - 传感器编号

bSensor A => 传感器编号

[+] Feedback

返回值：

处于活动状态时返回值为 1；处于非活动状态时返回值为 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：

**

CSD_bIsAnySensorActive

说明：

与手指阈值进行比较，检查所有传感器的差值计数阵列。 针对每个传感器调用 CSD_bIsSensorActive()，以便在调用此函数后 CSD_baSnsOnMask[] 阵列为最新状态。

C 语言原型：

```
BYTE  CSD_bIsAnySensorActive()
```

汇编语言：

```
lcall CSD_bIsAnySensorActive
```

参数：

无

返回值：

处于活动状态时返回值为 1；处于非活动状态时返回值为 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：

**

CSD_wGetCentroidPos

说明：

检查质心的差值阵列。 如果存在，则偏移和长度存储在临时变量中，根据 CSD 向导中指定的分辨率计算质心位置。 只有当滑条由 CSD 向导定义时，此函数才可用。

C 语言原型：

```
WORD  CSD_wGetCentroidPos(BYTE bSnsGroup)
```

汇编语言：

```
mov  A,  bSnsGroup
lcall CSD_wGetCentroidPos
```

参数：

bSnsGroup - 传感器组

传感器组 = 0（独立传感器组）

传感器组 = 1（第一个滑条组）

传感器组 = 2（第二个滑条组）

bSnsGroup A => 组编号

此参数引用的是用作滑条的特定传感器组。 组 0 用于按键。 滑条包含在组 1 和更高的组中。

[+] Feedback

返回值：

滑条的位置数值、A 中的 LSB 和 X 中的 MSB。

副作用：

此子程序通过减去噪声阈值来修改差值计数。 此子程序应在每次扫描后只调用一次，以避免得到负的差值。 如果应用监控差值计数信号，则在差值计数数据传输后调用此子程序。

如果有任何滑条传感器处于活动状态，则函数返回的值在零到 CSD 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，则函数返回 -1 (FFFFh)。 如果在执行质心 / 双工算法时发生错误，则函数返回 -1 (FFFFh)。 如果需要，您可以使用 CSD_blsSensorActive() 子程序确定被触摸的滑条段。

注：如果滑条段的噪声计数大于噪声阈值，则此子程序可能会生成假质心结果。 此噪声阈值应仔细设置 （足够高于噪声水平），以防止噪声生成假质心。

CSD_wGetRadialPos

说明：

检查质心的差值阵列。 如果存在质心位置，则根据 CSD 向导中指定的分辨率计算该质心位置。 此函数仅 CSD 向导定义的径向滑条可用。

C 语言原型：

```
WORD  CSD_wGetRadialPos(BYTE bSnsGroup)
```

汇编语言：

```
mov  A,  bSnsGroup
call CSD_wGetRadialPos
```

参数：

bSnsGroup A => 组编号

此参数是正在使用的径向滑条的编号。 该编号可以通过 CSD UM 向导从径向滑条表示法的左侧获取（例如：对于 s2，径向滑条编号为 2）。

返回值：

径向滑条的位置值、A 中的 LSB 和 X 中的 MSB。

副作用：

此子程序应在每次扫描后只调用一次，以避免得到负的差值和基准线值更新。 如果应用监控差值计数信号，则在差值计数数据传输后调用此子程序。

如果有任何滑条传感器处于活动状态，则函数返回的值在零到 CSD 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，则函数返回 -1 (FFFFh)。

注：如果滑条段的噪声计数大于噪声阈值，则此子程序可能会生成假质心结果。 此噪声阈值应仔细设置 （足够高于噪声水平），以防止噪声生成假质心。

CSD_wGetRadialInc

说明：

返回实际的手指移位以及手指当前位置与先前位置的差值。 此函数与 CSD_wGetRadialPos() 结合使用，采用后者生成的数据 （数据保存在内部变量中）。

C 语言原型：

```
WORD  CSD_wGetRadialInc(BYTE bSnsGroup)
```


[+] Feedback

汇编语言：

```
mov  A,  bSnsGroup
call CSD_wGetRadialInc
```

参数：

bSnsGroup A => 组编号

此参数是正在使用的径向滑条的编号。 该编号可以通过 CSD UM 向导从径向滑条表示法的左侧获取（例如：对于 s2，径向滑条编号为 2）。

返回值：

手指移位值 （顺时针为正，逆时针为负）、A 中的 LSB 和 X 中的 MSB。

手指移位值是手指当前位置与先前位置之间的差值。 如果在先前的扫描过程中无触摸 （倒数第二次 CSD_wGetRadialPos() 返回 -1 (FFFFh)）或者当前无触摸 （此时 CSD_wGetRadialPos() 返回 -1 (FFFFh)）

副作用：

此子程序应仅在 CSD_wGetRadialPos() API 之后调用。 因为它使用由 CSD_wGetRadialPos(). 设置的内部数据 CSD_waSliderPrevPos 和 CSD_waSliderCurrPos

CSD_InitializeSensorBaseline

说明：

通过扫描所选传感器，加载含有初始值的 CSD_waSnsBaseline[bSensor] 阵列元素。 原始计数值复制到所选传感器的基准线值阵列元素中。 此函数可用于复位单个传感器的基准线值。

C 语言原型：

```
void CSD_InitializeSensorBaseline(BYTE bSensor)
```

汇编语言：

```
mov A, bSensor
lcall CSD_InitializeSensorBaseline
```

参数：

bSensor - 传感器编号

A => 传感器编号

返回值：

无

副作用：

**

CSD_InitializeBaselines

说明：

通过扫描每个传感器，加载含有初始值的 CSD_waSnsBaseline[] 阵列。 原始计数值复制到每个传感器的基准线值阵列中。

C 语言原型：

```
void CSD_InitializeBaselines()
```


[+] Feedback

汇编语言：

```
lcall CSD_InitializeBaselines
```

参数：

无

返回值：

无

副作用：

**

CSD_SetDefaultFingerThresholds

说明：

加载含有 FingerThreshold 参数值的 CSD_baBtnFThreshold[] 阵列。 如果 CSD_baBtnFThreshold[] 阵列未通过自定义值手动加载，则必须在扫描之前调用此函数。

C 语言原型：

```
void CSD_SetDefaultFingerThresholds()
```

汇编语言：

```
lcall CSD_SetDefaultFingerThresholds
```

参数：

无

返回值：

无

副作用：

**

CSD_SetScanMode

说明：

设置扫描速度和分辨率。 此函数可以在运行时调用以更改扫描速度和分辨率。 此函数将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 µs 扫描时间进行扫描。 接近检测器可以以较低频率扫描，使用 16-bit 分辨率和 12 ms 的扫描时间进行远程检测。 此函数可以与 CSD_ScanSensor() 函数一起使用。

C 语言原型：

```
void CSD_SetScanMode(BYTE bSpeed, BYTE bResolution)
```

汇编语言：

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

参数：

bSpeed - 扫描速度代码，在 0 到 3 之间

[+] Feedback

bResolution - 分辨率 [9..14]

返回值：

无

副作用：

**

CSD_SetRefValue

说明：

设置扫描参考值。仅在这些情况下有效：提供的参考来自模拟调制器（“参考”参数中的 ASE11）或外部滤波后的 PWM/PRSPWM 信号。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 设置的是最大参考电压，因而灵敏度较低。此函数可以与 CSD_ScanSensor(). 一起使用

C 语言原型：

```
void CSD_SetRefValue(BYTE bRefValue)
```

汇编语言：

```
mov     A, bRefValue
lcall    CSD_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：

**

CSD_ClearSensors

说明：

通过按顺序为每个传感器调用 CSD_wGetPortPin() 和 CSD_DisableSensor()，将所有的传感器清除到非采样状态。

C 语言原型：

```
void  CSD_ClearSensors()
```

汇编语言：

```
lcall  CSD_ClearSensors
```

参数：

无

返回值：

无

副作用：

**

CSD_wReadSensor

说明：

返回 A (LSB) 和 X (MSB) 中的关键原始扫描值。

C 语言原型：

```
WORD  CSD_wReadSensor(BYTE bSensor)
```

汇编语言：

```
mov A, bSensor
lcall CSD_wReadSensor
```

参数：

bSensor - 传感器编号

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：

**

CSD_wGetPortPin

说明：

返回给定传感器的端口号和引脚掩码。 传递的参数对 CSD_Sensor_Table[]. 中的数据编制索引并进行选择。返回值可以传递给 CSD_EnableSensor() 和 CSD_DisableSensor().

C 语言原型：

```
WORD  CSD_wGetPortPin(BYTE bSensor)
```

汇编语言：

```
mov  A,  bSensor
lcall     CSD_wGetPortPin
```

参数：

bSensorNumber - 范围为 0 到 (n - 1)，其中 n 是 CSD 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。 CSD_wGetPortPin() 使用传感器数量来确定所选活动传感器的端口和位掩码。

返回值：

A => 传感器位掩码

X => 端口号

副作用：

**

CSD_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 CSD_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。 驱动模式将被修改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。 这还会启用比较器功能。

[+] Feedback

C 语言原型：

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

汇编语言：

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

参数：

A => 传感器位掩码

X => 端口号

bSensorMask - 给定传感器的位掩码

bPort - 给定键的端口号

返回值：

无

副作用：

**

CSD_DisableSensor

说明：

禁用 CSD_wGetPortPin() 函数选择的传感器。 驱动模式更改为 “ 强 (001)” 并设置为零。 这可以将传感器有效接地。 端口引脚与 AnalogMuxBus 的连接关闭。 函数参数由 CSD_wGetPortPin() 函数返回。

C 语言原型：

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

汇编语言：

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

参数：

A => 传感器位掩码

X => 端口号

bSensorMask - 给定传感器的位掩码

bPort - 给定键的端口号

返回值：

无

副作用：

**

[+] Feedback

固件源代码示例

示例 1. 此代码将启动用户模块并连续扫描传感器。 通信部分可用于将值传递给 PC 绘图工具。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

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

void main(void)
{
  M8C_EnableGInt;
  CSD_Start();
  CSD_InitializeBaselines() ; //scan all sensors first time, init baseline
  CSD_SetDefaultFingerThresholds() ;
  //
  // Loop Forever
  //
  while (1) {
    CSD_ScanAllSensors(); //scan all sensors in array (buttons and sliders)
    CSD_UpdateAllBaselines(); //Update all baseline levels;

//detect if any sensor is pressed
if(CSD_bIsAnySensorActive()){
// Add user code here to proceed the sensor touching
}

    // now we are ready to send all status variables to chart program
    // communication here
    //
    // OUTPUT CSD_waSnsResult[x] <- Raw Counts
    // OUTPUT CSD_waSnsDiff[x] <- Difference
    // OUTPUT CSD_waSnsBaseline[x] <- Baseline
    // OUTPUT CSD_baSnsOnMask[x] <- Sensor On/Off
  }
}
```


[+] Feedback

示例 2. 下面的代码演示的是如何能够并联多个传感器并通过调用 CSD_ScanSensor() 函数同时扫描这些传感器。 当需要在不分隔已触摸传感器的情况下检测传感器触摸时，此示例非常有用。 可以使用此示例进行器件唤醒检测和最大程度地缩短扫描时间以节省电池能量。 如果检测到唤醒触摸，可以将每个传感器分别返回到常规扫描。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Scan several sensors in parallel
//-------------------------------------------------------------------------

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

void main(void)
{
     M8C_EnableGInt;

     CSD_Start();
     CSD_SetDefaultFingerThresholds();

     // Enable the sensor connected to P1[4]
     CSD_EnableSensor(0x10, 1);
     // Enable the sensor connected to P1[6]
     CSD_EnableSensor(0x40, 1);
     // Enable the sensor connected to P3[0]
     CSD_EnableSensor(0x01, 3);

     // Initialize baseline for sensor number "3"
     CSD_InitializeSensorBaseline(3);

     while (1) {
          // Scan continuously sensor number "3" which is connected
          //in parallel to the enabled above sensors
          CSD_ScanSensor(3);
          CSD_UpdateSensorBaseline(3);
                if(CSD_bIsSensorActive(3)){
     // Add user code here to proceed the buttons pressing
                }
     }
}
```


[+] Feedback

示例 3. 下面的示例演示的是如何能够用 CSD_SetScanMode() 函数，以不同的扫描参数扫描不同的传感器。 当需要执行按键触摸检测和接近检测时这非常有用。 扫描按键时采用低分辨率以减少扫描时间，扫描接近情况时采用较高分辨率以获取最大灵敏度。 代码可进行调整，降低接近情况的扫描频率，以及只有当未检测到按键触摸时才进行接近情况扫描。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Scanning sensors with different scanning speed and resolution
//-------------------------------------------------------------------------

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

void main(void)
{

     M8C_EnableGInt;

     CSD_Start();
     CSD_SetDefaultFingerThresholds();

     // Set Fast, 9-bit resolution mode for baseline calculations
     CSD_SetScanMode(1, 9);

     // Initialize baselines for all of the sensors which operate in
     // Fast mode and 9-bit resolution
     CSD_InitializeSensorBaseline(0);
     CSD_InitializeSensorBaseline(1);
     CSD_InitializeSensorBaseline(2);

     // Set Slow, 14-bit resolution mode for baseline calculations
     CSD_SetScanMode(3, 14);
     // Initialize baselines for all of the sensors which operate in
     // Slow mode and 14-bit resolution
     CSD_InitializeSensorBaseline(3);

     while (1) {
          // Set Fast, 9-bit resolution mode for the following buttons
          CSD_SetScanMode(1, 9);
          // Scan sensor number "0"
          CSD_ScanSensor(0);
          // Scan sensor number "1"
          CSD_ScanSensor(1);
          // Scan sensor number "2"
          CSD_ScanSensor(2);

          // Set Slow, 14-bit resolution mode for the following sensor
          CSD_SetScanMode(3, 14);
          // Scan sensor number "3"
          CSD_ScanSensor(3);

          CSD_UpdateAllBaselines();
          //detect if any sensor is pressed
          if(CSD_bIsAnySensorActive()){
               // Add user code here to proceed the buttons pressing
```


[+] Feedback

```
            }
        }
}
```

示例 4. 下面的示例演示的是如何能够为每个传感器设置不同的手指阈值级别。 当多个传感器放置在不同位置上且其中一些传感器比其他更灵敏时，这非常有用。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Set individual finger threshold parameter for each sensor
//-------------------------------------------------------------------------

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

void main(void)
{
M8C_EnableGInt;

CSD_Start();
CSD_InitializeBaselines();

// set finger threshold for sensor "0"
CSD_baBtnFThreshold[0] = 10;
// set finger threshold for sensor "1"
CSD_baBtnFThreshold[1] = 20;
// set finger threshold for sensor "2"
CSD_baBtnFThreshold[2] = 30;
// set finger threshold for sensor "3"
CSD_baBtnFThreshold[3] = 40;
// set finger threshold for sensor "4"
CSD_baBtnFThreshold[4] = 50;
// set finger threshold for sensor "5"
CSD_baBtnFThreshold[5] = 255;
// set finger threshold for sensor "6"
CSD_baBtnFThreshold[6] = 200;

while (1) {
// Scan continuously all sensors
CSD_ScanAllSensors();
CSD_UpdateAllBaselines();
//detect if any sensor is pressed
if(CSD_bIsAnySensorActive()){
// Add user code here to proceed the buttons pressing
}
}
}
```


[+] Feedback

配置寄存器

Table 7. 模块 CMP，寄存器： 控制 0

位	7	6	5	4	3	2	1	0
值	0	0	1	1	1	0	1	0

Table 8. 模块 CMP，寄存器： 控制 1

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

寄存器 PRS8 配置

Table 9. 模块 CMP，寄存器： 控制 2

位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	0	0	0

Table 10. 模块 CMP，寄存器： 控制 3

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 11. 模块 PRS，寄存器： 控制

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 12. 模块 PRS，寄存器： 多项式

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 13. 模块 PRS，寄存器： 种子

模式 / 位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 14. 模块 PRS，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	1	1	0	1	0	1	0

Table 15. 模块 PRS，寄存器： 输入

模式 / 位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 16. 模块 PRS，寄存器： 输出

模式 / 位	7	6	5	4	3	2	1	0
值	1	1	0	0	0	0	0	0

Table 17. 模块 CMP，寄存器： 控制 0

位	7	6	5	4	3	2	1	0
值	0	0	1	1	1	0	1	0

[+] Feedback

Table 18. 模块 CMP，寄存器： 控制 1

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

寄存器 PRS16 配置

Table 19. 模块 CMP，寄存器： 控制 2

位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	0	0	0

Table 20. 模块 CMP，寄存器： 控制 3

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 21. 模块 CNT，寄存器： 控制

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 22. 模块 CNT，寄存器： 周期

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 23. 模块 CNT，寄存器： 比较

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 24. 模块 PRS16_LSB，寄存器： 控制

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 25. 模块 PRS16_LSB，寄存器： 多项式

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 26. 模块 PRS16_LSB，寄存器： 种子

模式 / 位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 27. 模块 PRS16_MSB，寄存器： 控制

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 28. 模块 PRS16_MSB，寄存器： 多项式

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

[+] Feedback

Table 29. 模块 PRS16_MSB，寄存器： 种子

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 30. 模块 CNT，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 31. 模块 CNT，寄存器： 输入

模式 / 位	7	6	5	4	3	2	1	0
值	0	0	0	1	0	1	1	0

Table 32. 模块 CNT，寄存器： 输出

模式 / 位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	0	0	0

Table 33. 模块 PRS16_LSB，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	0	0	1	0	1	0

Table 34. 模块 PRS16_LSB，寄存器： 输入

模式 / 位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 35. 模块 PRS16_LSB，寄存器： 输出

模式 / 位	7	6	5	4	3	2	1	0
值	1	1	0	0	0	0	0	0

Table 36. 模块 PRS16_MSB，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	1	1	0	1	0	1	0

Table 37. 模块 PRS16_MSB，寄存器： 输入

模式 / 位	7	6	5	4	3	2	1	0
值	0	0	1	1	0	0	0	0

Table 38. 模块 PRS16_MSB，寄存器： 输出

模式 / 位	7	6	5	4	3	2	1	0
值	1	1	0	0	0	0	0	0

[+] Feedback

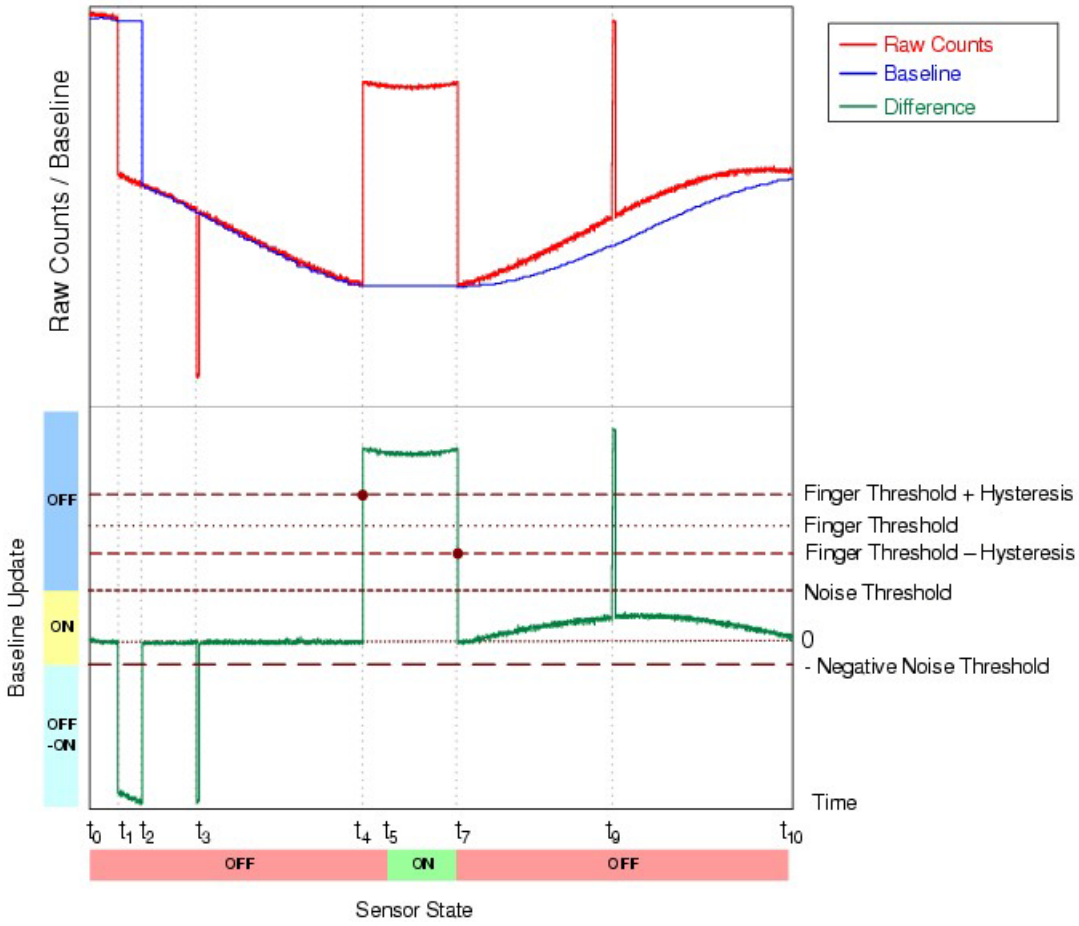
附录

以下部分介绍用户模块数据手册中通常没有包含的信息。 该详细信息由赛普拉斯工程师编写，以帮助用户成功设计 CapSense 应用。 部分信息将在以后转为应用注解。

CSD 参数的交互

下面的两幅图说明的是基准线值更新和决策逻辑操作，可帮助更好地理解如何设置 UM 参数以获得最佳性能。 第一幅图说明的是当 “ 传感器自动复位 ” 参数设置为**禁用**时的系统操作。 第二幅图说明的是 “ 传感器自动复位 ” 参数设置为**启用**时的情况。 图中显示了手指阈值、噪声阈值、迟滞、负噪声阈值以及差值信号 （原始计数 － 基准线值）。 数据是在一些人工测试中收集的，这些测试演示了原始计数慢速和快速变化时的系统操作。慢速变化会由温度或湿度变化引起，快速变化会因为传感器触摸、ESD 事件或强射频场的影响而触发。

Figure 15. “ 传感器自动复位 ” 设为 “ 禁用 ” 时原始计数、基准线值、差值信号变化示例



在 t_0 处，原始计数接近于基准线值水平，然后因为湿度或温度变化，开始缓慢下降。 因为两次连续转变之间的原始计数变化未超过 NegativeNoiseThreshold 参数 （绝对值），所以基准线值通过跟踪原始计数最小值进行更新，保留原始计数信号的较小值。

在 t_1 处，原始计数快速下降，负差值超过 NegativeNoiseThreshold。 如果在手指位于传感器上时器件加电，过一段时间后手指移开，则会发生这种情况。 此时，基准线值更新机制冻结，内部超时计数器激活。 当差值信号低于 LowBaselineReset 示例的 NegativeNoiseThreshold 时，基准线值复位。 这发生在 t_2 处。

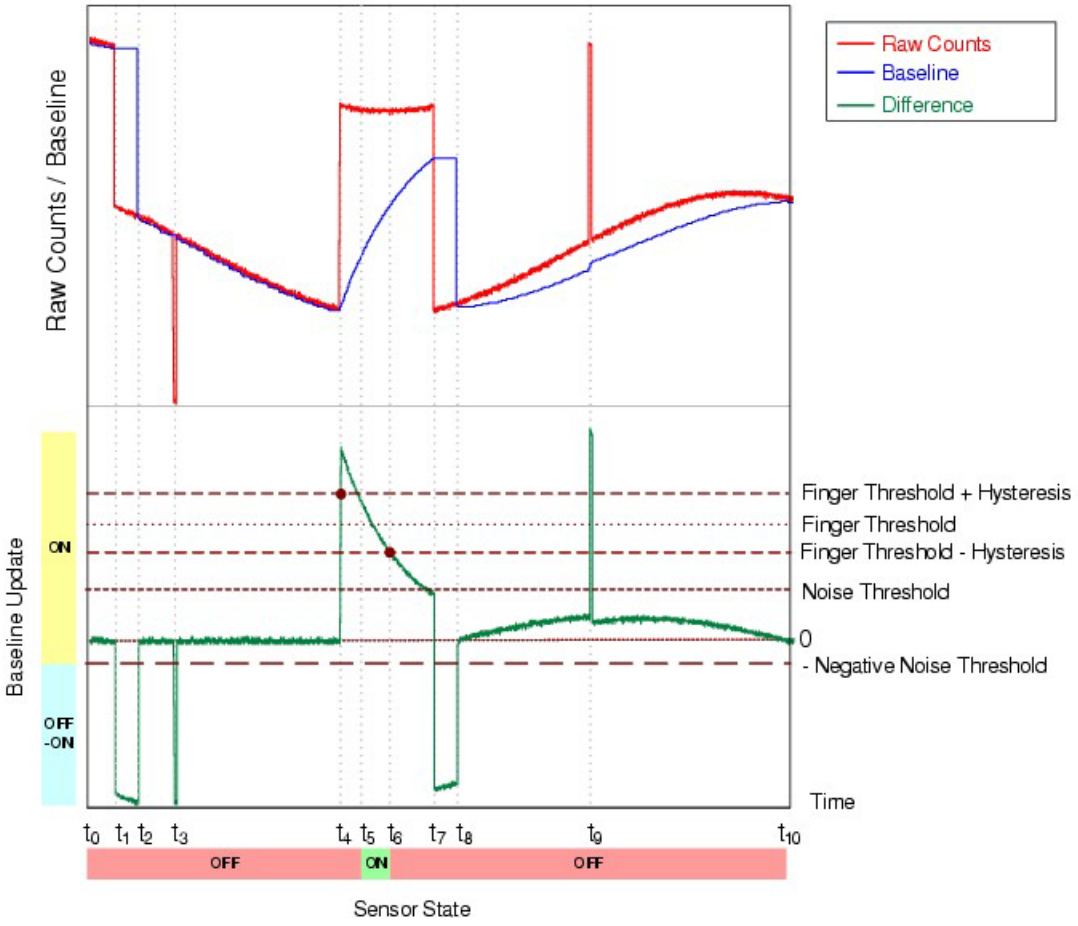
[+] Feedback

第二大负差值信号尖峰脉冲发生在 t_3 处，此尖峰脉冲可能由 ESD 事件等触发。 因为计数示例中的尖峰脉冲持续时间小于 LowBaselineReset 参数，所以基准线值保持不变，尖峰脉冲进行滤波。 这可以防止假基准线值复位和导致假触摸检测。

传感器在 t_4 处被触摸。 当差值信号超过 “ 手指阈值 + 迟滞 ” 值时，内部防反跳计数器激活。 如果信号超过此值的量大于防反跳示例数，则传感器状态设置为启用。 这发生 t_5 处。 当差值信号在 t_7 处下降到 “ 手指阈值 - 迟滞 ” 级别之下时，传感器立即恢复为关闭状态。 t_9 处短暂的正尖峰脉冲被防反跳计数器滤波，这是因为样品单元中的尖峰脉冲持续时间未超过防反跳值。

原始计数在 t_7 与 t_{10} 之间缓慢升高。 当差值信号低于 NoiseThreshold （“ 传感器自动复位 ” 设置为 “ 禁用 ”），差值信号与漂移速率成比例时，基准线值将使用水桶算法进行更新。 基准线值更新速度可以使用 BaselineUpdate 阈值参数控制。 参数值越低，基准线值更新速度越快。

Figure 16. 在 “ 传感器自动复位 ” 设置为 “ 启用 ” 的情况下原始计数、基准线值、差值信号的变化示例



上图中的系统操作类似于上例中的操作，但有下列区别：

- 在 t_6 处传感器被触摸时，触摸持续时间因为活动基准线值更新算法而下降。
- 手指移开后，在短时间阻止了触摸检测的 LowBaselineReset 采样 (t_8) 后基准线值复位。 这用作额外的去抖动机制。

[+] Feedback

CSD 分步调校指南

电容式感应的成功取决于是否为给定感应电极设置了最佳参数。 影响这些设置的变量包括：

- 电极的几何尺寸
- 外覆层厚度和介电常数
- PSoC 器件的电极连接阻抗
- 最终应用状况，例如：

- 电源的存在情况
- 温度
- 湿度
- 潮气的存在情况
- ESD、EMC 或 EMI 要求

不同任务 （防水操作、使用高阻抗材料进行感应、接近检测和通过厚外覆层的操作以及有关通过认证测试的建议）的最佳做法在单独的应用注解中说明。

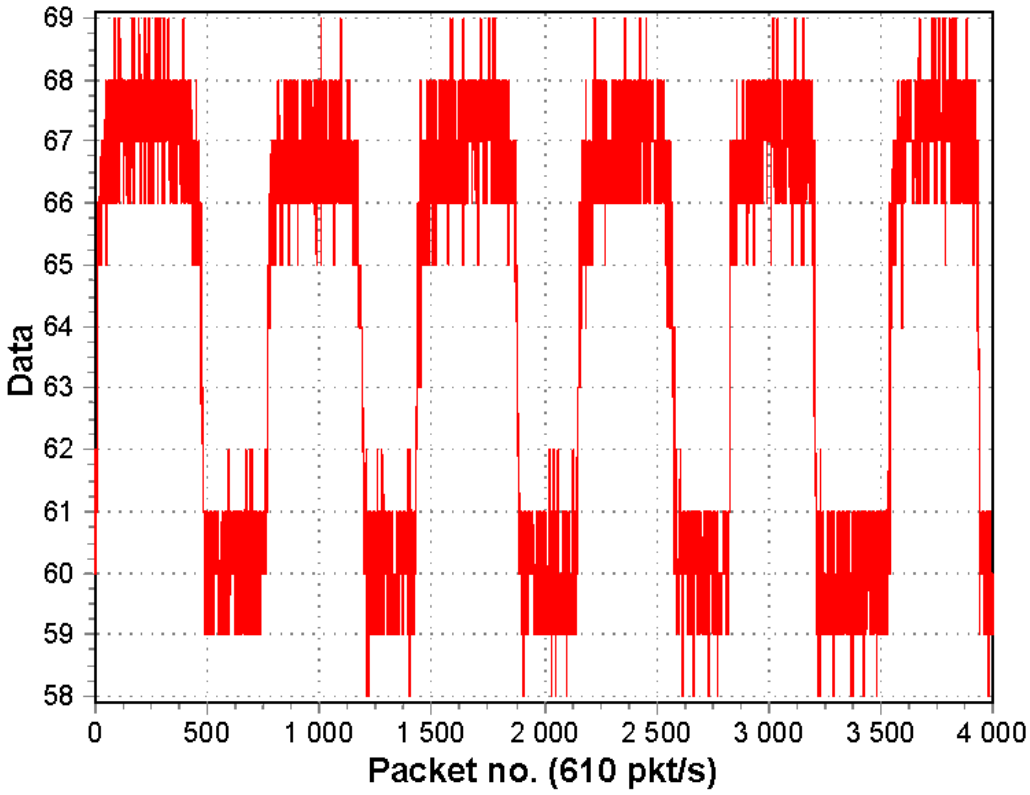
下面以 CY3214 板作为测试示例，介绍在典型 CapSense 应用中配置用户模块的基本准则。 感应区覆盖有 2 mm 的塑料外覆层。 请按下列步骤配置 CSD 用户模块参数：

1. 准备目标板。 组装目标应用 PCB ，在它上面安装外覆层。 使用胶水或特殊胶带来进行安装。 避免在 PCB 与外覆层之间留有气隙，否则会极大降低灵敏度，且由于当您触摸时气隙漂移，会导致出现许多假按键触发情况。
2. 设置用于监控数据的实时监控工具。 在 CSD 配置过程中，使用可以实时观察一个或多个数据系列的 PC 绘图工具。 在用户模块调校过程中，必须观察原始计数、基准线值和信号差值。 为此，可以使用 I^2C-USB 桥接器。 在我们的测试中，使用了一个桥接器进行原始计数数据监控。 另一个好的备选方案是使用 USBUART 用户模块，通过一个仿真串行端口发送调试信息。 不要使用 LCD 或其他任何数字显示屏来监控计数，因为它们太慢，您无法观察到数据的动态变化。
3. 设置初始配置。 此配置使用的是 16-bit PRS。 在开始测试之前，PSoC Designer 中已设置下列参数：

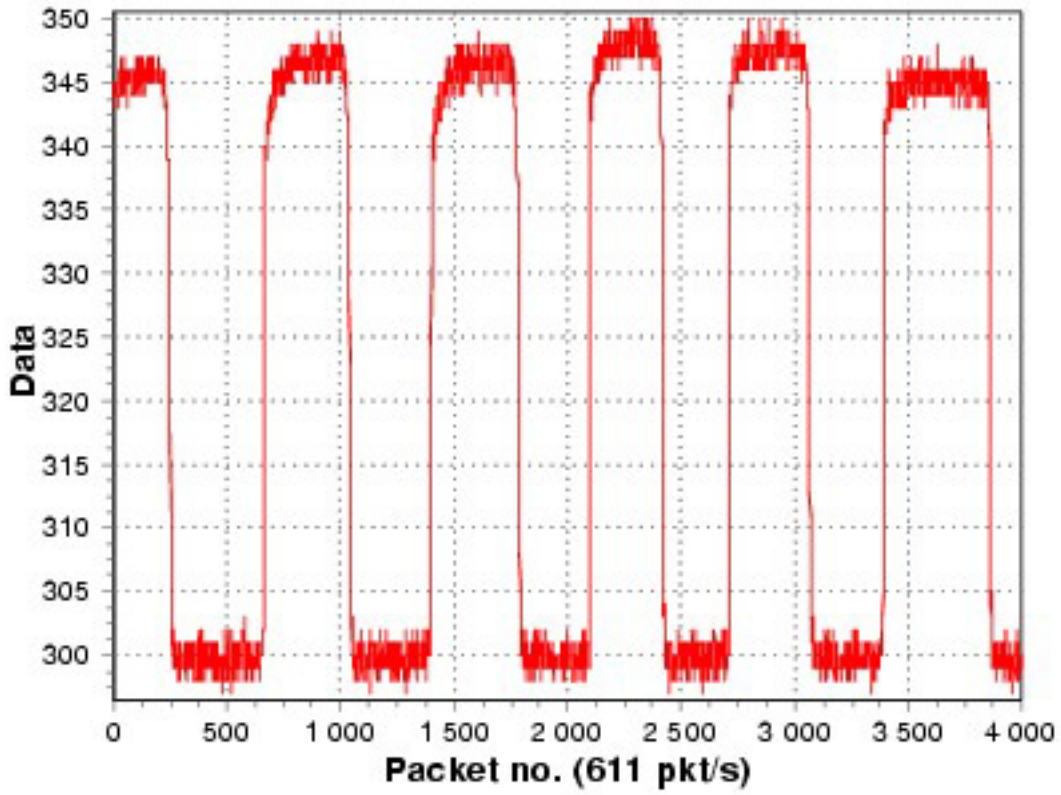
User Module Parameters	Value
FingerThreshold	40
NoiseThreshold	20
BaselineUpdateThreshold	200
Sensors Autoreset	Enabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	20
LowBaselineReset	50
Scanning Speed	Fast
Resolution	9
Modulator Capacitor Pin	P0[5]
Feedback Resistor Pin	P3[1]
Ref Value	2
ShieldElectrodeOut	None

[+] Feedback

4. 在 CSD 向导中分配传感器引脚 （为扫描分配传感器 P5[7]、P3[7] 和 P3[6]）。
5. 生成应用和采样代码。
6. 使用绘图工具监控传感器原始计数数据，以确认用户模块可操作。 触摸传感器应导致原始计数（CSD_waSnsResult 变量）从 59 更改为 68。

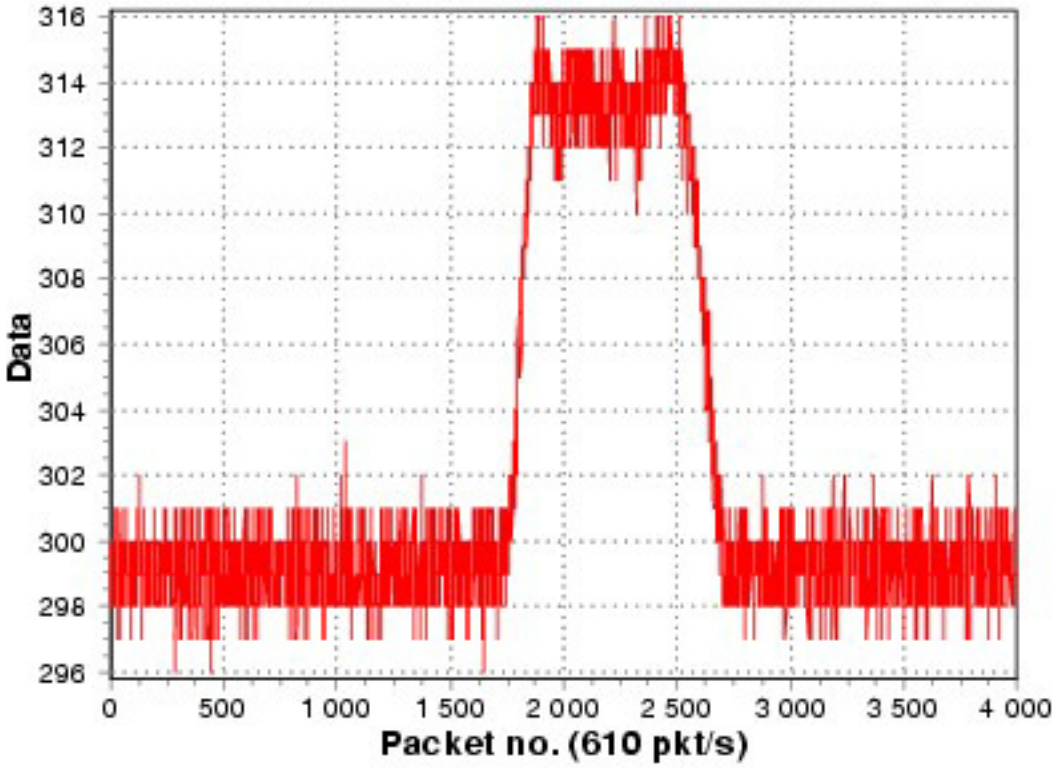


7. 调校外部组件。 赛普拉斯最初使用了 5.6 nF 调制器电容 (C_{mod}) 和 1.2 kΩ 反馈电阻 R_b 。 在触摸情况下对来自不同传感器的原始计数值进行观察后，赛普拉斯找到了产生最大原始计数值的传感器。 上图显示了来自此传感器的信号。 下面的信号值对应于无手指触摸的情况，上面的信号值对应于触摸情况。 通过分析来自此传感器的信号值，可以看到系统仅使用了电容代码转换器动态范围的 8%。 9-bit 分辨率的整个范围为 Nm = 512，最大原始计数大约为 85。 这意味着可以通过将反馈电阻值增加到 5.1 kΩ，使动态范围利用率提高到建议的 60-70%。 可以使用不同的电阻值来完成此任务，具体取决于您的原始计数观察情况。 下面的手指响应是更换电阻后的结果。 来自手指触摸的响应得到增加。



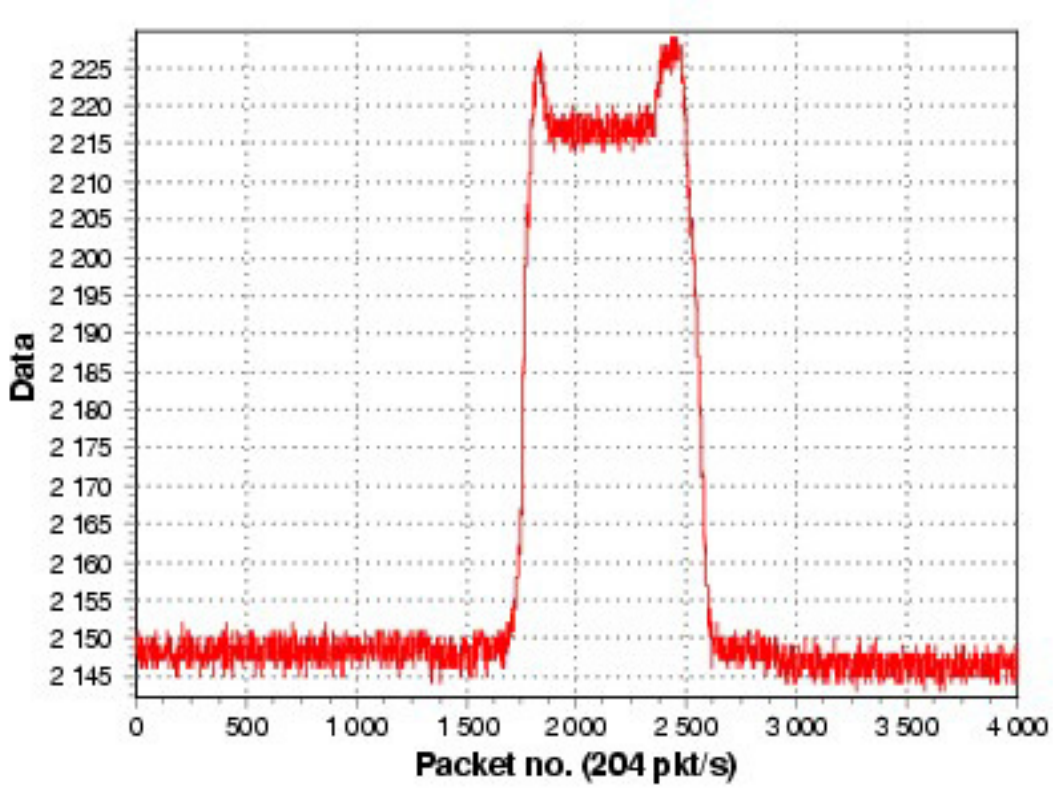
[+] Feedback

8. 针对最坏情况进行调整。 使用手指模拟器确保器件在不同的情况 （例如很轻的触摸）下都能够可靠工作。 将 10 mm 未连接线圈放在外覆层上可模拟最坏情况。 使用介电物体 （如火柴或牙签）将线圈移过按键。 结果如下图所示。 如果板在传感器周围使用了接地层，则可以运行此测试。 如果该板覆盖的是屏蔽电极而非接地层，则可以通过用手指轻触摸来模拟最坏情况。



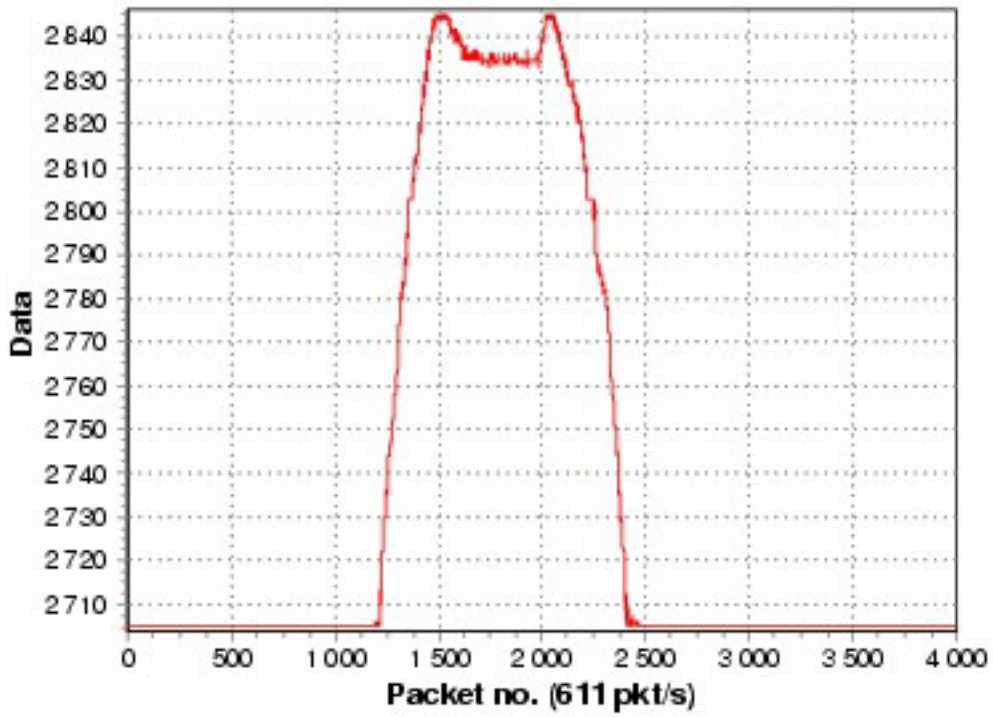
[+] Feedback

9. 来自线圈的信号得到了识别，但是信噪比太小，无法进行可靠检测。 差值只有大约 9 dB。 要增加灵敏度，请选择较高扫描分辨率。 在测试中，分辨率从 9 位增加到 12 位。 下面是在这些设置情况下来自线圈的信号。



[+] Feedback

10. 将扫描分辨率从 9 位增加到 12 位使信噪比提高到了 25 dB，这对于大多数实际的应用都很有帮助。来自人手指的信号大得多。 此操作的代价是扫描时间的增加。 如果扫描时间对于应用而言非常重要，则可以切换为 PRS8 配置。 以下是相同 UM 参数下来自 PRS8 配置的线圈响应 （PRS 多项式设置为短）：



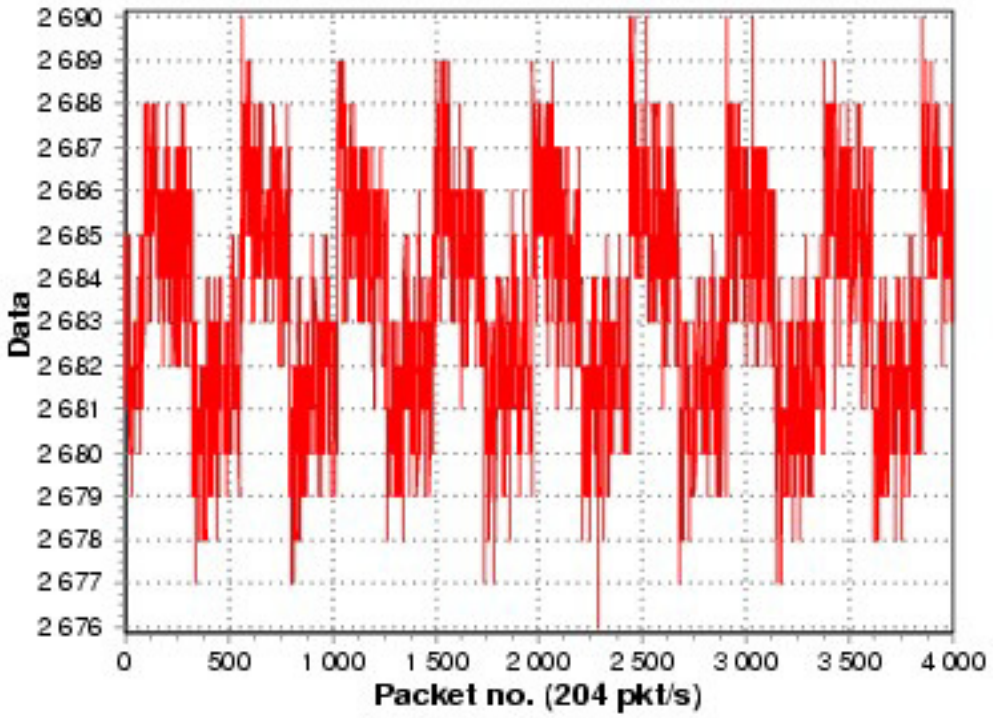
11. 与短 PRS 多项式下的 PRS8 配置相比，此配置提供的噪音比更佳。 但是较短的伪随机顺序可能会导致较差的外部电气抗噪能力。

[+] Feedback

12. 设置阈值。 对用户模块参数进行以下更改：

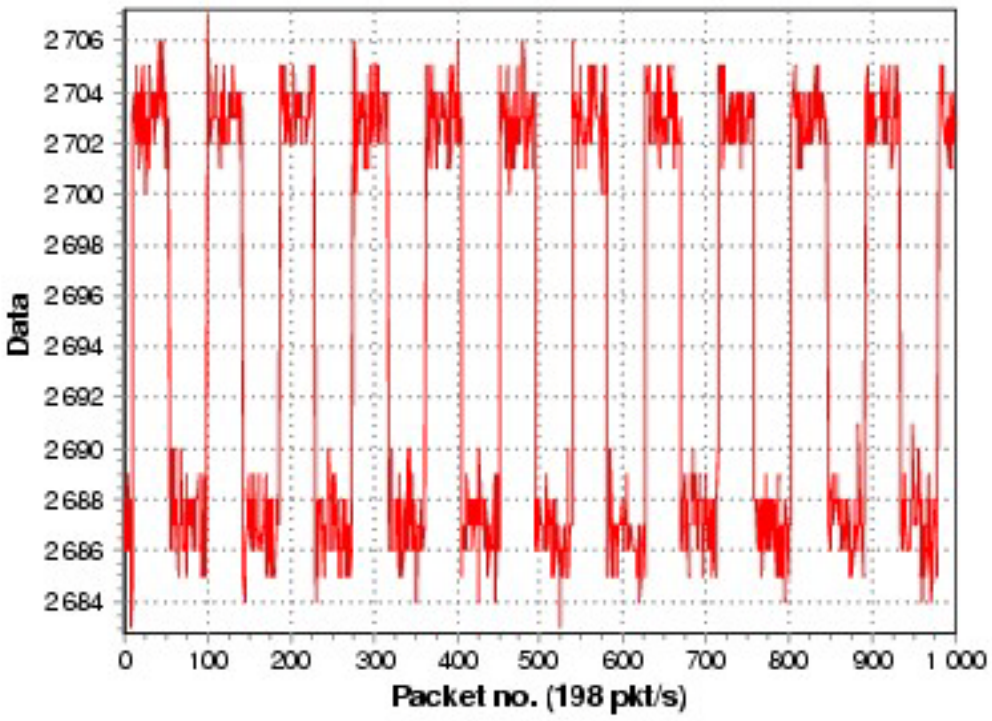
User Module Parameters	Value
FingerThreshold	40
NoiseThreshold	20
BaselineUpdateThreshold	200
Sensors Autoreset	Enabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	20
LowBaselineReset	50
Scanning Speed	Fast
Resolution	12
Modulator Capacitor Pin	P0[5]
Feedback Resistor Pin	P3[1]
Ref Value	2
ShieldElectrodeOut	None

13. 设置最佳扫描速度。 假设测试应用电源电压稳压不良，则由于目标器件其他部分的运行，可能会产生 ± 5% 的快速电源波动。 此外，假设 PSoC 器件驱动多个 10 mA LED 及其 CapSense 函数。 内部芯片块阻抗上的电流下降可能会导致内部电源电压波动。 在此电压瞬变情况下，CapSense 系统应继续运行。 测试由于这些波动导致的原始计数变化。 LED 必须同时开启和关闭。 睡眠定时器中断是执行此任务的理想选择。 此外，可以使用外部脉冲源模拟外部负载开启和关闭。 下图显示了在扫描处于活动状态时切换 LED 得到原始计数。



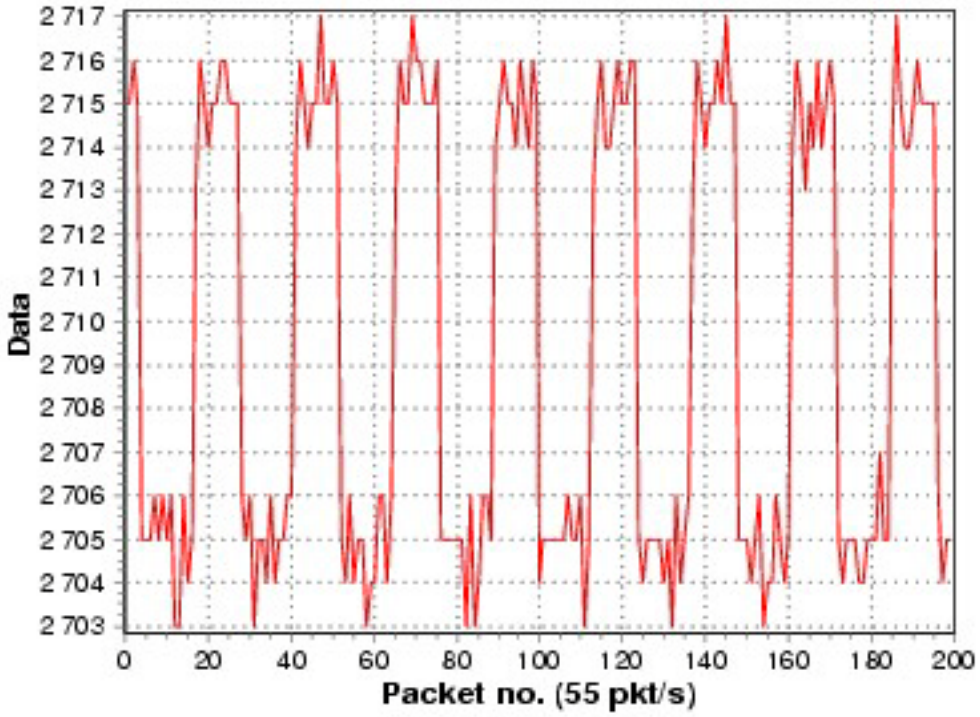
[+] Feedback

14. 如图所示，在扫描处于活动状态时 LED 开 / 关对于原始计数值没有明显的影响。 测试电源快速变化时 CapSense 的稳定性。 很慢的电源变化由基准更新算法处理，在大多数情况下不会产生问题。 此测试中使用了 LM1117-ADJ 电压稳压器。 输出电压由反馈电阻网络变化进行调制，该变化利用由外部信号源驱动的 MOSFET 来产生。 下图显示的是当电源在 4.75V 到 5.25V 之间振荡时传感器的原始计数差值。



[+] Feedback

15. 如图所示，电源瞬变原始计数变化（18）与阈值（35..45）接近，不会导致假触摸检测。 但会因为很轻的触摸导致多个触摸触发情况检测。 这一情况的解决方案是在用户模块参数中增加迟滞。 此外，通过使用较慢的扫描速度也可以降低电源波动影响。 下图显示的是较慢的扫描速度下收集的原始计数数据：



16. 如图所示，降低扫描速度降低了电源电压变化对原始计数的影响。 现在，瞬变差值大约为 10 个计数。 这远低于阈值，对 CapSense 模块操作没有不利影响。 此操作的代价是扫描速度增加了四倍，这在某些情况下是不利的。
17. 调校 BaselineUpdateThreshold 参数。 应用要求最大触摸检测时间少于 1sec。 将 SensorsAutoreset 参数设置为 “ 启用 ”。 检查 BaselineUpdateThreshold 是否提供了足够补偿环境变化的基准线值更新速度。 例如，如果应用场合是厨房，冷空气吹到板上可能导致温度快速变化，而原始计数会因温度变化而下降。 通过将基准线值自动复位成原始计数值，基准线值会对此情况进行跟踪。 因此，在大多数情况下，由于环境因素造成的原始计数下降不应是问题。 如果原始计数由于温度变化而增加，则可能会因为将此变化解释为触摸而触发假触摸。 基准线值更新速度必须进行调整，使温度（或其他环境因素）对原始计数与基准线值之差的影响远低于 “ 手指阈值 ”。 这些测试过程中对原始计数与基准线值之差进行了监控。 监控的值为 0，因而差值低于 “ 噪声阈值 ” 参数。 在这些测试过程中，此参数设置为最小值五。 这意味着预设的 BaselineUpdate 阈值参数提供了足够的基准线值跟踪速度，对于应用而言，温度波动不应是问题。
18. 设置完所有参数后，可以运行 ESD 测试。 即使 “ESD 防反跳 ” 参数设置为 “ 禁用 ”，应用也能通过这些测试，不会产生问题。 如果需要，在 ESD 测试出现问题时可以启用 “ESD 防反跳 ” 参数。 启用此参数的代价是 RAM 缓冲区大小的增加。
19. 许多 CapSense 应用需要通过多种 EMC/EMI 兼容性测试。 如果您的应用在 EMC/EMI 方面遇到问题，AN2318 *EMC Design Considerations for PSoC CapSense Applications*（PSoC CapSense 应用的 EMC 设计注意事项）中包含了解决问题的信息。 解决此问题的其他可行方法是使用较慢的 PRS 时钟，减少传感器路径辐射。 您可以尝试带有预分频器的配置，或者使用较慢的 IMO 模式（例如：以 6MHz 而非 24MHz 运行 SYSCLK）。 如果 PRS 时钟频率或预分频器周期设置有任何更改，则反馈电阻也需要进行调整，以最大程度地利用动态范围来达到最大灵敏度。

[+] Feedback

20. 如果应用未通过 EMC 测试，请尝试降低扫描速度并提高分辨率。 这将使 PRS 多项式序列变长，从而获得较高的抗噪声能力。 此操作的代价是传感器扫描时间的增加。

故障排除

- 您可以将预充电预分频器用作 UART 波特率时钟源。 建议的 UART 速度不得小于 115,200 波特。 对于 24 MHz IMO 操作，预分频器周期应当设置为 25。 由于此值不是 2^N 的倍数，建议使用较慢的扫描速度以获得更好的信噪比。 通过实验对此进行测试。
- 如果在参考设置中看到较大的周期性噪声，请尝试增大 **CSD.asm** 文件中的 CSD_DELAY 常数。 此延时设置的是测量开始之前的调制器启动时间。 减少调制器电容 C_{mod} 降低也可以提供帮助。 产生此噪声的原因是：由于内部模拟调制器低通滤波器上的时间常数很小，在之前的测量周期中调制器电容充电为另一个电压。
- 扫描速度和分辨率会影响信噪比 (SNR)。 在某些情况下，较慢的扫描速度和较高的分辨率可获得更好的信噪比。
- 如果电极外覆层较厚，可能需要较高的分辨率和较慢的扫描速度。
- PRS 多项式会自动根据扫描速度和分辨率进行调整，使 PRS 序列重复周期接近于样品转换周期计数。 较慢的扫描速率和较高的分辨率会产生较长的 PRS 序列，因而在 EMC 测试期间可提供较强的抗噪声能力。
- 扫描速度越慢，调制器运行频率越低，读数对比较器动态特性的依赖度也越低。 如果您需要在电源波动或 PSoC 器件控制高电流负载的情况下获得良好的原始计数稳定性，请使用模拟调制器在内部形成比较器参考。 这种情况下，建议的扫描速度为 “ 正常 ” 或 “ 慢速 ”。
- Sigma-Delta 调制转换方法属于积分法类别。 在较高分辨率下，它显示出具有最佳性能。 使用可能的最长扫描时间。 使用 1 ms 进行传感器扫描，以获得最佳结果。
- 可以使用屏蔽电极有效减少杂散电容影响，甚至在不需要防水的应用中也可以使用。 这种情况下，屏蔽电极可以安装在电容式传感器区域下 PCB 底层上。 在这种情况下，建议使用开口填充模式，以减小屏蔽电极的电容。

消除可能的资源使用冲突

注意不要更改此用户模块使用的硬件配置。 其中包括：

- 内部使用 GlobalOutEven_1 或者 GlobalOutEven_5 （取决于调制器反馈电阻引脚选择）总线将电压比较器输出信号传递到输出总线。 不要将任何源与这些总线连接。
- 不要更改电压比较器总线 1 LUT 函数。 比较器 Bus_1 应设置为 **~A**。
- 模拟列一时钟源应设置为 **VC1**。
- VC1 由用户模块内部设置。 输入 “ 全局资源 ” 的值在运行时覆盖。
- 使用屏蔽电极时，请将行 LUT 函数设置为 **A**。

[+] Feedback

中断持续时间管理

PRS16 配置下传感器扫描处于活动状态时，管理中断服务子程序 (ISR) 持续时间。 8-bit 定时器的定时来自 VC2。 定时器较差的溢出间隔为：

Equation 2

$$T_{owf} = VC_2 \cdot VC_1 \frac{256}{F_{IMO}}$$

F_{IMO} - IMO 频率，VC1 = 2、4 或 8，分别对应快速、正常、慢速扫描速度。

VC_2 - 此值始终设置为四。

大多数情况下，此间隔不会引起问题。 在某些情况需要对其进行检查。

可能的 ISSP 引脚冲突

低阻抗反馈电阻与 P1[1] 引脚永久连接会导致 ISSP 编程错误。 这种情况下请使用其他引脚。

时钟速度

CY8C24x94 设备的 CPU 速度应该为 SysClk/32 或者更快才能实现正常的功能。

[+] Feedback

版本历史记录

版本	创建者	说明
1.4	DHA	分辨率最大值为 3000。 删除了 0.5 移位，增加了负值补偿。 修正了向导中的引脚列表。
1.50	DHA	增加了对 CY8C21x12 设备的支持。

Note PSoC Designer 5.1 在所有用户模块数据表中都引入了 “ 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 © 2007-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

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

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

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

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