CapSense® Sigma-Delta 数据手册 CSD V 1.90

Copyright © 2007-2014 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块				API 存储器		（所需外部 I/O 的）引脚数量
	抽取滤波器	I²C/SPI	数字	模拟	闪存	RAM	
CY8C28x45、 CY8C28x52、 CY8C28x13、 CY8C28x33、 CY8CLED04							
带 IDAC 的一阶调制器	1	–	0...2	2	–	–	1
带 IDAC 的二阶调制器	1	–	0...2	2	–	–	1

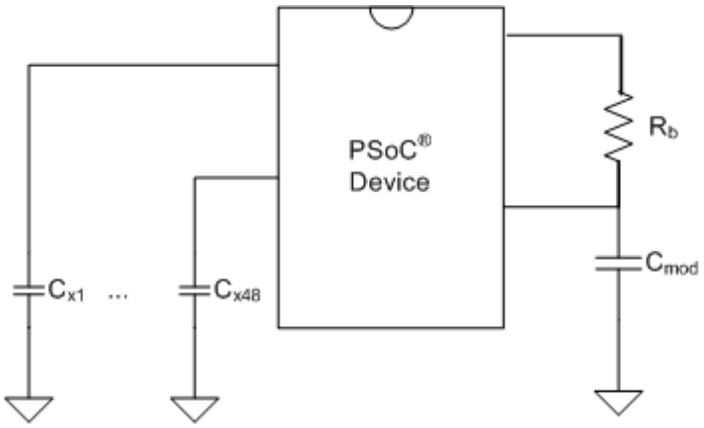
若需要一个或多个使用此用户模块的完全配置功能性示例项目，请访问 www.cypress.com/psocexampleprojects。

功能和概述

- 根据器件引脚数量，最多可以扫描 41 个电容式传感器。
- 可以感应厚度达 15 mm 的玻璃外覆层。
- 使用线缆传感器时，接近感应的检测范围可达 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式传感器的任意组合。
- 通过双工法，可以使滑条传感器的物理分辨率增加一倍。
- 利用内插法，可提高滑条传感器的分辨率。
- 支持带有两个滑条传感器的触摸板。
- 通过高阻抗导电材料 （如 ITO 薄膜）提供感应支持。
- 即使存在水膜或水滴的情况下，屏蔽电极仍可保证可靠地运行。
- 通过 CSD 向导进行传感器和引脚的分配。
- 使用集成基线更新算法来处理温度、湿度和静电放电 （ESD）事件。
- 可轻松调整各操作参数。
- PC GUI 应用支持实时的原始数据监控和参数优化。

使用 Sigma-Delta 调制 （CSD）用户模块的电容式感应采用开关电容技术提供电容感应功能，该技术可以通过Sigma-Delta调制器将感应开关电容电流转换为数字代码。CSD用户模块可支持通过一阶和二阶Sigma Delta 调制器进行的单通道 CapSense 扫描。

图 1. CSD 典型应用



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 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（稍后导向将在本数据手册中加以介绍）。
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，以执行独立的传感器、滑条传感器或触摸板。
9. 将 I^2C-USB 桥接器连接至目标电路板，并观察信号。
10. 更改 CSD 参数，以优化您的设置并重新编译应用。
11. 对 PSoC 器件进行编程并验证模块操作。调整 CSD 参数，以满足 5:1 信噪比的要求，如 CY8C21x34/B CapSense 设计指南中所述。

如果遇到任何问题，请参见*附录*中的*故障排除*部分。

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。高级决策逻辑提供了对环境因素（如温度、湿度）和电源电压变化的补偿。独立的屏蔽电极可用于屏蔽传感器阵列，以降低杂散电容。这样，可以在存在水膜或水滴的条件下更可靠地运行。

高级软件功能可提供滑条双工法，以便在两个位置中可以使用一个电气传感器来提高分辨率。通过这些功能，还可以在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、电气和软件组件组成：

- **物理**：即物理传感器本身，通常是安装在与 PSoC 相连的 PCB 上的传导模型，并通过绝缘层、软膜或透明覆盖层与显示屏隔离开。
- **电气**：是用于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、sigma-delta 调制器以及基于计数器的数字滤波器，用以将调制器输出的位流转换为可读取的数字格式。
- **软件**：是指检测和补偿软件算法，用于将计数值转换为传感器检测结果。对于连续的附属型传感器（如滑条和触摸板），将提供 API 函数，以便插入一个分辨率高于传感器物理分辨率的位置。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级别扩展为 100。另外，通过相同的 API，可以使用两个电容式传感器，以凸凹咬合方式排列，用于确定它们之间的导电物体（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 CSD2X 用户模块之前，建议阅读下列文档。

- *《CY8C28X45 和 CY8C21345 PSoC 可编程片上系统技术参考手册》*中的 "CapSense 系统" 章节。

建议在阅读 CSD 用户模块数据手册后应阅读下面的设计指南。可从赛普拉斯网站 www.cypress.com 上获取这些文档：

- Capsense 入门手册
- CY8C20xx6A/H CapSense 设计指南
- CY8C21x34/B CapSense 设计指南
- CY8C20x34 CapSense 设计指南
- CY8CMBR2044 CapSense 设计指南

电容测量操作

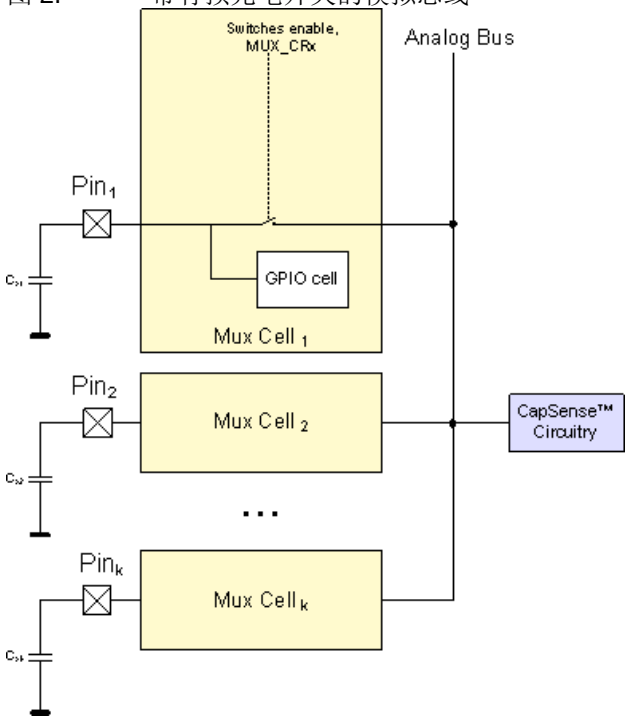
可通过固件实现决策逻辑。通过固件分析电容的测量，跟踪电容因环境因素造成的缓慢变化，并运行决策逻辑，以检测按键触摸变化以及计算滑条位置。

扫描传感器阵列

CY8C28x45 系列器件都具有内置模拟总线，可以使电容式传感器连接到任意 PSoC 引脚上。 CSD 用户模块使用内部预充电开关，以在时钟信号相位 Ph_1 充电给活动传感器，并在相位 Ph_2 将模拟总线连接至传感器。 Sigma-delta 调制器的调制电容和比较器的输入端始终与模拟总线相连接。

通过设置 MUX_CRx 寄存器中的相应位，固件可以连续执行传感器扫描。

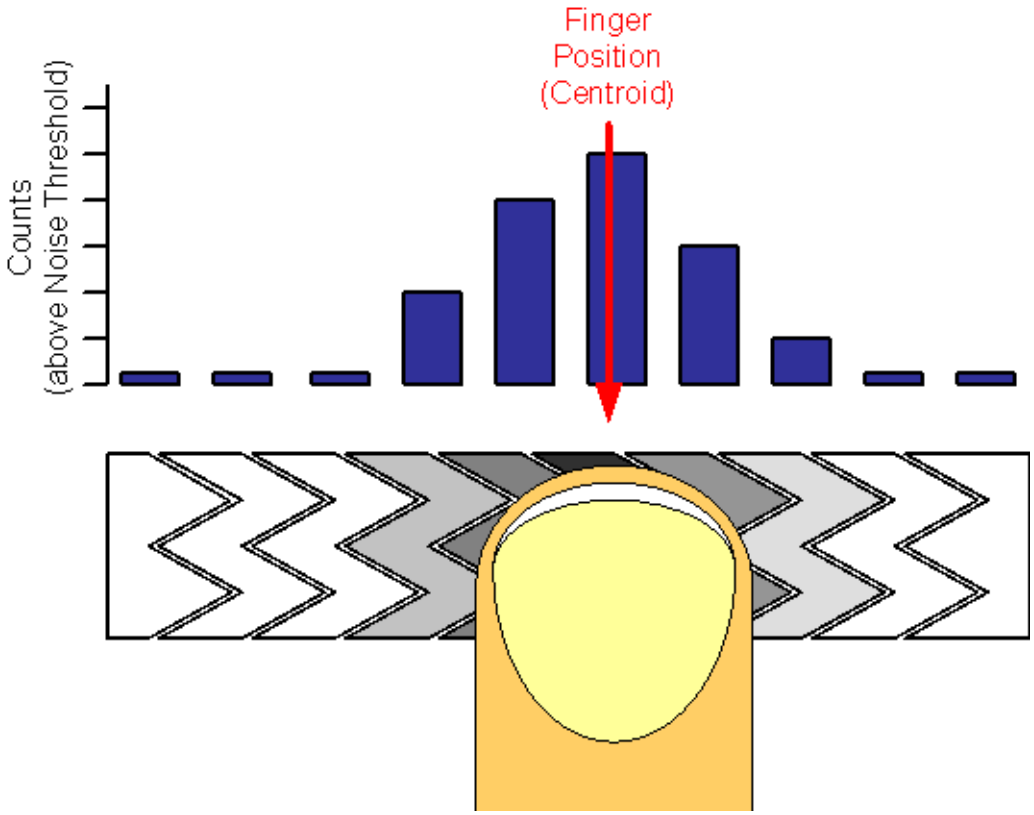
图 2. 带有预充电开关的模拟总线



滑条

滑条适用于需要渐进式调节的控制应用。示例包括照明控制 （调光器）、音量控制、图示均衡器和速度控制。这些传感器在布局上彼此相邻。某个传感器进行的动作会引起其他相邻传感器部分动作。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。滑条可在 CSD 向导中设置，即建立多组滑条，每一组都有一个特定的顺序。传感器滑条数量的实际下限值是 5，上限值是所选 PSoC 器件提供的传感器位置的数量。

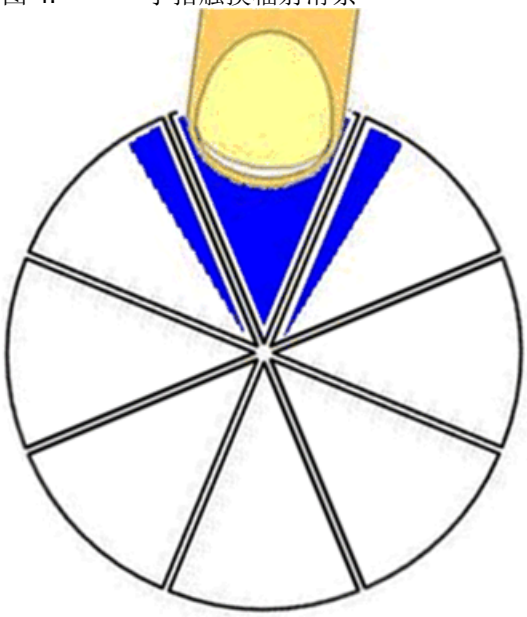
图 3. 对物理传感器位置排序



越接近滑条一半部分的强信号，将导致前半部分产生相同程度的伪信号，但最终是杂散信号。感应算法搜索相邻最强的一组信号，以确定解析的滑条位置。

辐射滑条

图 4. 手指触摸辐射滑条



CSD 用户模块支持两种滑条类型：线性和辐射滑条。辐射滑条类似于线性滑条。但线性滑条有起点和终点，而辐射滑条却没有。当发生触摸时，中心计算算法将考虑到当前开关左右两侧的传感器的开关数量。辐射滑条未采用双工法。

CSD 用户模块包含两个支持辐射滑条的 API 函数。第一个函数 CSD_wGetRadiaPos() 返回质心位置，第二个函数 CSD_wGetRadialInc() 则返回以分辨率单位表示的手指移位。当手指以顺时针方向移动时，它是正的偏移。

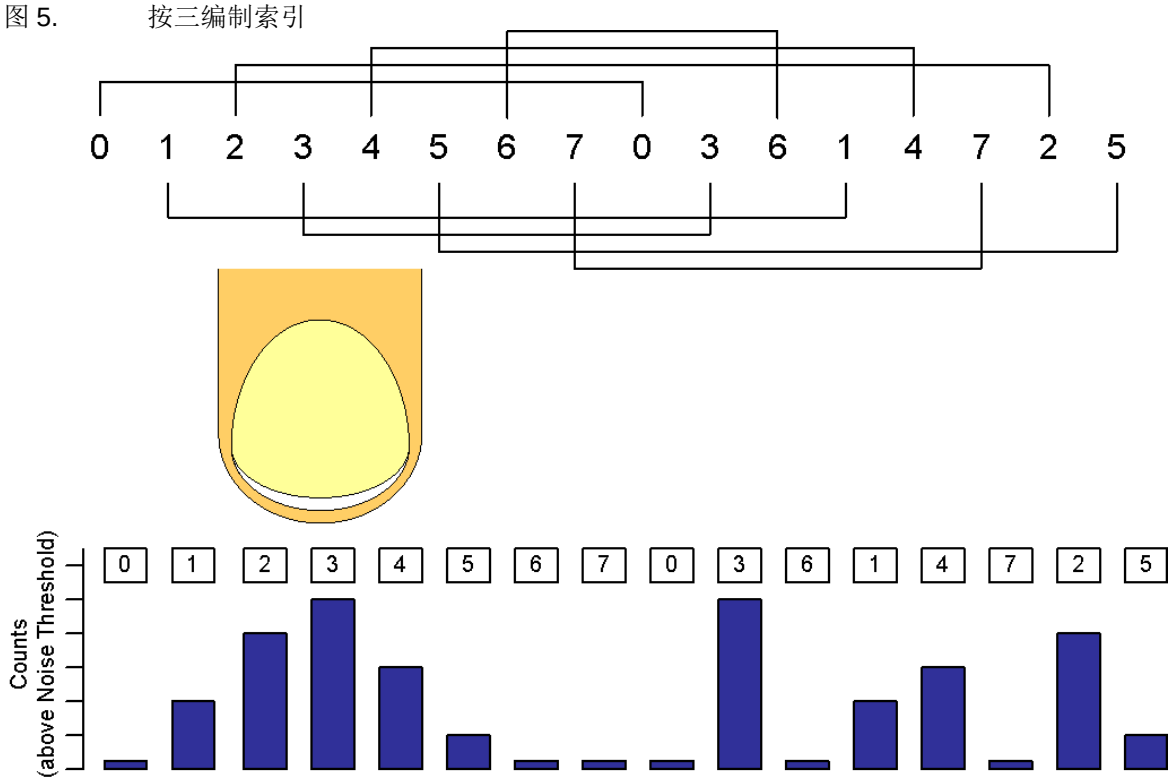
参考点 （0）位于第一个传感器的中心。线性滑条和辐射滑条的分辨率都受限制，其限制为 (传感器所用的引脚数量 - 1) x 2^8 - 1 ；对于双工型滑条，该值为 (2 x 传感器所用的引脚数量 - 1) x 2^8 - 1。

双工

滑条中的每个 PSoC 传感器连接都映射到滑条传感器阵列中的两个物理位置。物理位置的前半部分 （较低数值部分）顺序被映射到基本分配的传感器内，设计师使用 CSD 向导将端口引脚分配给这些传感器。物理传感器位置的后半部分 （数值较高的部分）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，一半相邻的传感器动作则不会使另一半相邻传感器的动作。请小心地确定此次序，并将其映射到印刷电路板上。

有许多方法可以确定后半部分物理传感器位置次序。最简单的方法是对前部分中的传感器编制索引，首先是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。

图 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 布局，某些传感器对可能需要更长的布线。当选择双工时， CSD 向导会自动生成双工传感器的索引表。下表说明了不同滑条段计数的双工序列。

表 1. 不同滑条段计数的双工序列

滑条段总数	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

滑条段总数	段序列
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

双工滑条的滑条段选择指南

选择滑条所需的段数主要取决于滑条的物理长度。然而，确定双工滑条的段数时必须特别小心。

在双工滑条设计中，一个传感器将作为两个不同的物理滑条段使用，以增加滑条的长度。手指触摸完全覆盖的段数必须小于从同一传感器派生出的两段之间的传感器数量。这样可确保双工滑条能正常工作。

例如，在 10 段滑条（5 个传感器）的情况下，从传感器 3 中派生出的两个滑条段仅由两个传感器（传感器 4 和 0）分隔。此时，手指触摸不得完全覆盖两个以上的传感器段，以确保滑条能正常工作。

对于一个 12 段滑条，一个手指触摸不得覆盖 3 个段以上。同样，对于一个 18 段滑条，一个手指触摸不得完全覆盖 4 个段以上。

插值和比例因子

在滑条传感器和触摸板的应用中，通常需要确定手指 （或其他电容物体）的位置，以达到高于各传感器本身间距的分辨率。手指在滑条传感器或触控板上的触摸面积往往大于任何一个传感器。

要想计算使用质心的插值位置，首先需要扫描传感器阵列，以确定已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，且这些信号要高于噪声阈值。如果发现最强信号，将使用此信号和那些大于噪声阈值的连续信号计算质心。通常使用 2 到 8 个传感器，通过下列公式计算质心：

公式 1

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

计算出的值通常是分数。要想将质心报告给特定的分辨率（例如：对于 12 个传感器，范围值为 0 ～ 100），需要将质心数值乘以计算得出的标量。另一种更有效的方法是将内插法和按比例计算的方法统一到一个计算中，并按所需的比例因子直接报告结果。这是通过高级 API 实现的。

滑条传感器计数和分辨率都在CSD向导中进行设置。比例值由向导将计算出，并以分数值的形式进行存储。

质心分辨率的乘数被包含在三个字节中，并且具有以下的位定义：

分辨率乘数最高有效位 （MSB）								
位	7	6	5	4	3	2	1	0
乘数	2^{15}	21^{4}	21^{3}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
分辨率乘数中等有效位 （ISB）								
乘数	128	64	32	18	16	8	4	2
分辨率乘数最低有效位 （LSB）								
乘数	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256

使用以下公式计算分辨率：

分辨率 = (传感器数量 – 1) x 乘数

质心值以 24 位无符号的整数保存，其分辨率是传感器和乘数的函数。

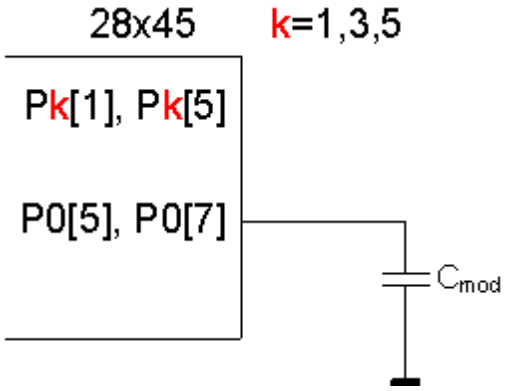
反馈组件选择指南

用户模块需要外部调制电容 C_{mod}。它还支持可选的屏蔽电极。本节介绍如何选择外部组件。

调制电容

该电容的一端可以连接到 P0[5] 或 P0[7] 端口引脚，另一端接地 Vss。通过用户模块参数设置可以选择引脚。请勿将选定给调制器组件连接的引脚用于任何其他目的。

图 6. 外部组件连接



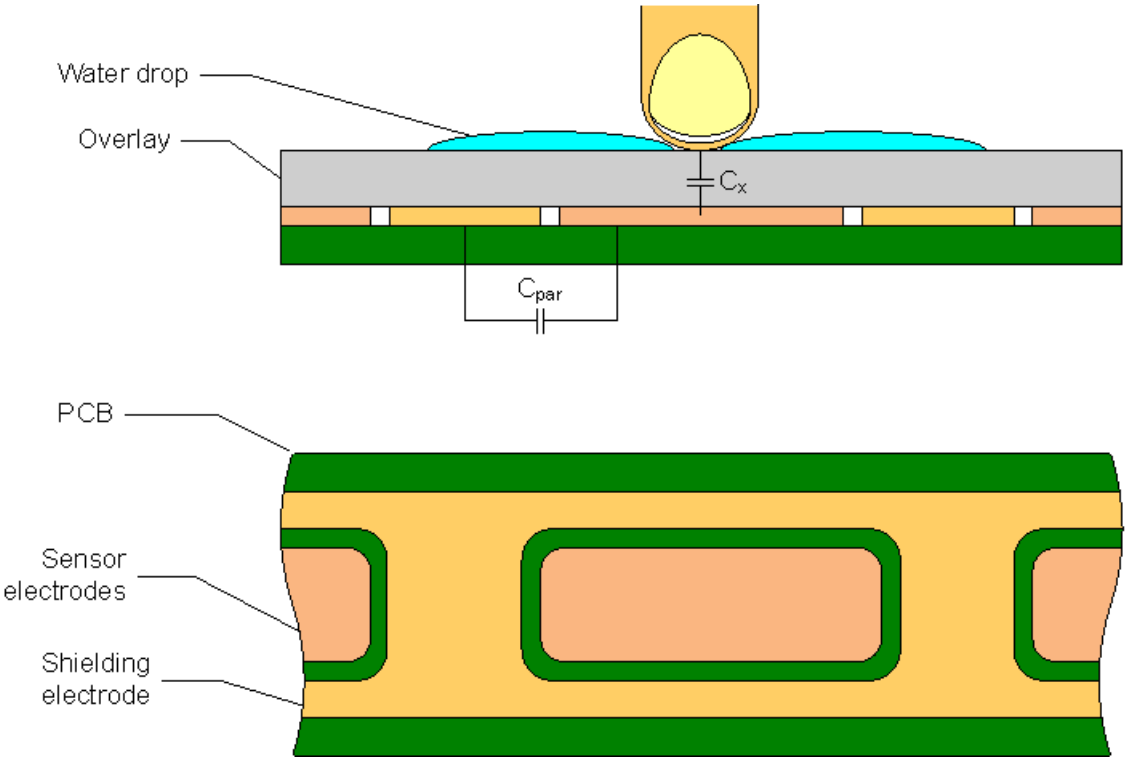
调制电容的建议值为 4.7 ～ 47 nF。可通过实验选择最佳电容值，以获得最大的信噪比。在大多数情况下，5.6 ～ 10 nF 电容值具有较好的效果。选择反馈电阻后，可以试验几个电容值，以获得最佳的信噪比。应该使用陶瓷电容。电容的温度系数并不重要。

屏蔽电极

有些应用要求即使存在水膜或水滴，也要能够可靠地运行。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会因为水、冰和湿度变化而出现误触发情况。对于这种情况，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上有水膜，则屏蔽和感应电极之间的耦合程度会增加。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大的动态数值范围。

在某些应用中，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。 CSD 用户模块支持屏蔽电极的单独输出。

图 7.　　可能的屏蔽电极 PCB 布局



上图显示了一种可能适用于按键屏蔽电极的布局配置。屏蔽电极尤其适用于透明的 ITO 触摸板器件。在这种器件中，它不但可阻止 LCD 驱动电极的噪声影响，而且还可以减少杂散电容。

在此示例中，屏蔽电极板覆盖了按键。作为另一替代方法，屏蔽电极可以安装在相对的 PCB 层上，其中包括按键下面的平板。在这种情况下，推荐使用填充模式，填充率约为 30 到 40%。这时，无需附加的接地层。

如果屏蔽电极与感应电极间存在水滴， C_{par} 将增加，调制器电流则会下降。在实际测试中，通过 API 可以增大调制器参考电压，以便使因水滴引起的原始计数增加值能够接近于零或略呈负值。可以通过选择适当的调制器参考值来实现此目的。

在此用户模块中，还将用于预充电时钟的同一信号提供给屏蔽电极。屏蔽电极可以连接到任何空闲行输出总线。将驱动模式设置为 **Strong Slow** （慢速强驱动）可以降低接地噪声和辐射。另外，可以在 PSoC 器件与屏蔽电极之间连接上升限制电阻。对于行 LUT 函数，您应选择 **A**。

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项作为预充电开关的时钟源：

- 16 位伪随机序列发生器 （PRS16）
- 8 位 PRS 源
- 带有预分频器的 8 位 PRS 源
- VC2

当第一次选择用户模块时，应当选择需要的配置。要于稍后更改此选择，请右键单击 “ 互连视图 ” 中的 “CSD 用户模块 ” 图标，并选择 **User Module Selection Options** （择用户模块选择选项）。

PRS16 配置将 PRS16 模块作为时钟源使用。PRS16 源提供扩频操作，确保能够良好地抵抗外部噪声源的噪声。另外，带有扩频时钟的设计具有更低的电磁辐射级别。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。下表比较了四种配置：

配置	工作频率	使用的数字模块	抗 EMC 噪声能力
PRS16	扩频，平均值为 $F_{IMO}/4$，峰值为 $F_{IMO}/2$	3	高。敏感点是 PRS 序列重复周期和 PRS 基频 F_{IMO} 的倍数。
PRS8	扩频，平均值为 $F_{IMO}/4$，峰值为 $F_{IMO}/2$	2	中。由于 PRS 重复周期较短，因此有更多敏感点。
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 $F_{IMO}/4 – F_{IMO}/512$	1	中。由于 PRS 重复周期较短，因此有更多敏感点。
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。

比较器参考源

比较器参考源用于生成比较器的参考电压。参考电压值决定了灵敏度。

对于一阶和二阶配置，用户模块使用不同的参考源。

对于一阶配置，用户模块支持参考源的下列多种选择：

- 带隙参考
- 模拟调制器，由 PRSPWM 或预分频器 - PWM 信号驱动
- 外部电阻电压分频器
- PRSPWM 或预分频器 - PWM 信号的外部 RC 滤波器

下表汇总了各参考选择选项：

外部组件	用户模块选择	使用场合
无	VBG	读取次数与电源电压成比例。仅在电源电压稳定时才使用
无	ASE11	建议用于大多数应用。尝试从此选项开始测试。
2	模拟列 （AnalogColumn） 输入选择	读取次数较少，与电源相关。建议 R1 = 10k； R2 = 3.6k
2	模拟列 （AnalogColumn） 输入选择	如果使用其他参考选择，会引起过大的噪声。

只能使用参考选择带隙 （VBG）或模拟调制器 （ASE11）。

对于第二阶配置，使用位于连续时间模块中的电阻分频器来构成调制器参考源。通过用户模块参数或 API 条用可以更改参考值。

直流和交流电气特性

表 2. 电源电压

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

表 3. 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 = 6MHz、基线 ≥ 分辨率最大计数的 70%)
噪声计数， 峰 - 峰值		0.2		% （噪声计数） / （基准线计数）	分辨率 = 16
噪声计数， 峰 - 峰值		1		% （噪声计数） / （基准线计数）	分辨率 = 14
噪声计数， 峰 - 峰值		10		% （噪声计数） / （基线计数）	分辨率 = 10

a. 当扫描速度减慢且基准线计数增加时，信噪比参数将提高。

表 4. 功耗

供电电压	最小值	典型值	最大值	单位	测试条件和注意
有功电流		10		mA	扫描期间的平均电流， 8 个传感器
待机电流		250		µA	扫描速度 = 快速，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

表 5. 功耗

供电电压	最小值	典型值	最大值	单位	测试条件和注意
有功电流		10		mA	扫描期间的平均电流， 8 个传感器
待机电流		250		µA	扫描速度 = 快速，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

表 6. 5.0 V PGA 直流电气特性

参数	典型值	限制	单位	条件和注意
增益值与额定值之间的偏差				
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 Vdd	V	
漏电流 [1]	1	--	nA	
输入电容 [1]	3	--	pF	
输出摆动	0.05 到 Vdd-0.05	--	V	
PSRR	73	--	dB	

表 7. 5.0 V PGA 交流电器特性

参数	典型值	限制	单位	条件和注意
斜率 （20% 到 80%） [2]				
低功耗	0.6	--	V/µs	
中功耗	2.5	--	V/µs	
高功耗	9.5	--	V/µs	
建立时间				
低功耗	13	--	µs	
中功耗	4	--	µs	
高功耗	1	--	µs	
噪声 [2]				参考输入
中功耗	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
高功耗	100		nV/√Hz	

表 8. 3.3 V PGA 直流电气特性

参数	典型值	限制	单位	条件和注意
增益值与额定值之间的偏差				
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 Vdd	V	
漏电流 [1]	1	--	nA	
输入电容 [1]	3	--	pF	
输出摆动	0.05 到 Vdd-0.05	--	V	
PSRR	68	--	dB	

参数	典型值	限制	单位	条件和注意
工作电流				
低功耗	130	--	µA	
中功耗	520	--	µA	
高功耗	2000	--	µA	

表 9. 5.0 V 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	复用器的参考电压 = Vdd/2 ± Vdd/2
输入电容	3	---	pF	包括 I/O 引脚
输入阻抗	1/(C*clk)	---	Ω	
有效分辨率：按 64 抽取、按 128 抽取、按 256 抽取	---	10、 12、 14	位	
采样率：按 64 抽取、按 128 抽取、按 256 抽取	---	31250、 15625、 7812	sps	数据时钟 8 MHz
直流精度				
DNL：按 64 抽取、按 128 抽取、按 256 抽取	<1 <1 <1 0.6	---	LSB	源时钟 1.5 MHz
偏移误差	13	---	mV	
增益误差	2		% FSR	包括参考增益误差
数据时钟	---	0.032 到 8.0	MHz	数字模块和模拟列时钟的输入

表 10. 3.3 V ADC 调制器直流和交流电器特性

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	复用器的参考电压 = Vdd/2 ± Vdd/2
输入电容	3	---	pF	包括 I/O 引脚
输入阻抗	1/(C*clk)	---	Ω	
有效分辨率：按 64 抽取、按 128 抽取、按 256 抽取	---	10、 12、 14	位	
采样率：按 64 抽取、按 128 抽取、按 256 抽取	---	31250、 15625、 7812	sps	数据时钟 8 MHz

参数	典型值	限制	单位	条件和注释
DC 精度				
DNL：按 64 抽取、按 128 抽取、按 256 抽取	<1 <1 0.5	---	LSB	数据时钟 1.5 MHz
偏移误差	13	---	mV	
增益误差	2		% FSR	包括参考增益误差
数据时钟	---	0.032 到 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，但不提供其他放置方式。所有配置都使用 ACC 和 ASC/ASD 或 ACE/ASE 模拟模块。不同的用户模块配置使用 0 ～ 2 数字模块。同时使用 CSD2X 用户模块和占用同一个模拟复用总线的用户模块 （如 AMuxN）时，它们之间可能发生冲突。如果需要同时操作，那么用户模块要使用其它模拟复用总线的引脚。

下表汇总了使用的数字资源。

配置	所使用的数字模块
PRS16	两个数字模块
PRS8	一个数字模块
带预分频器的 PRS8	两个数字模块
VC2	没有使用的数字模块

未使用的模拟模块和数字模块可供您自己使用。所有用户模块配置均使用硬件抽取滤波器。

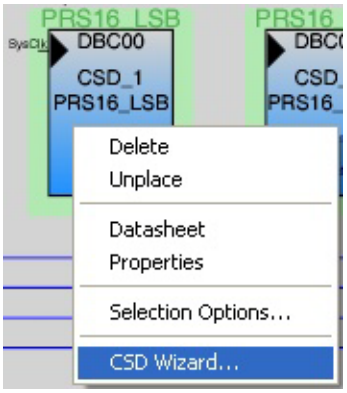
在建立 CSD 用户模块的端口引脚的连接之前，必须先放置使用特定引脚资源（包括 LCD 和 I2CHW）的用户模块。打开向导时，向导中会显示各配置选项。

在放置电容传感器的连接时，请勿使用 P1[0] 和 P1[1]。这些引脚用于对芯片进行编程，而且可能存在过大的走线电容值，这样会影响传感器的灵敏度和噪声。

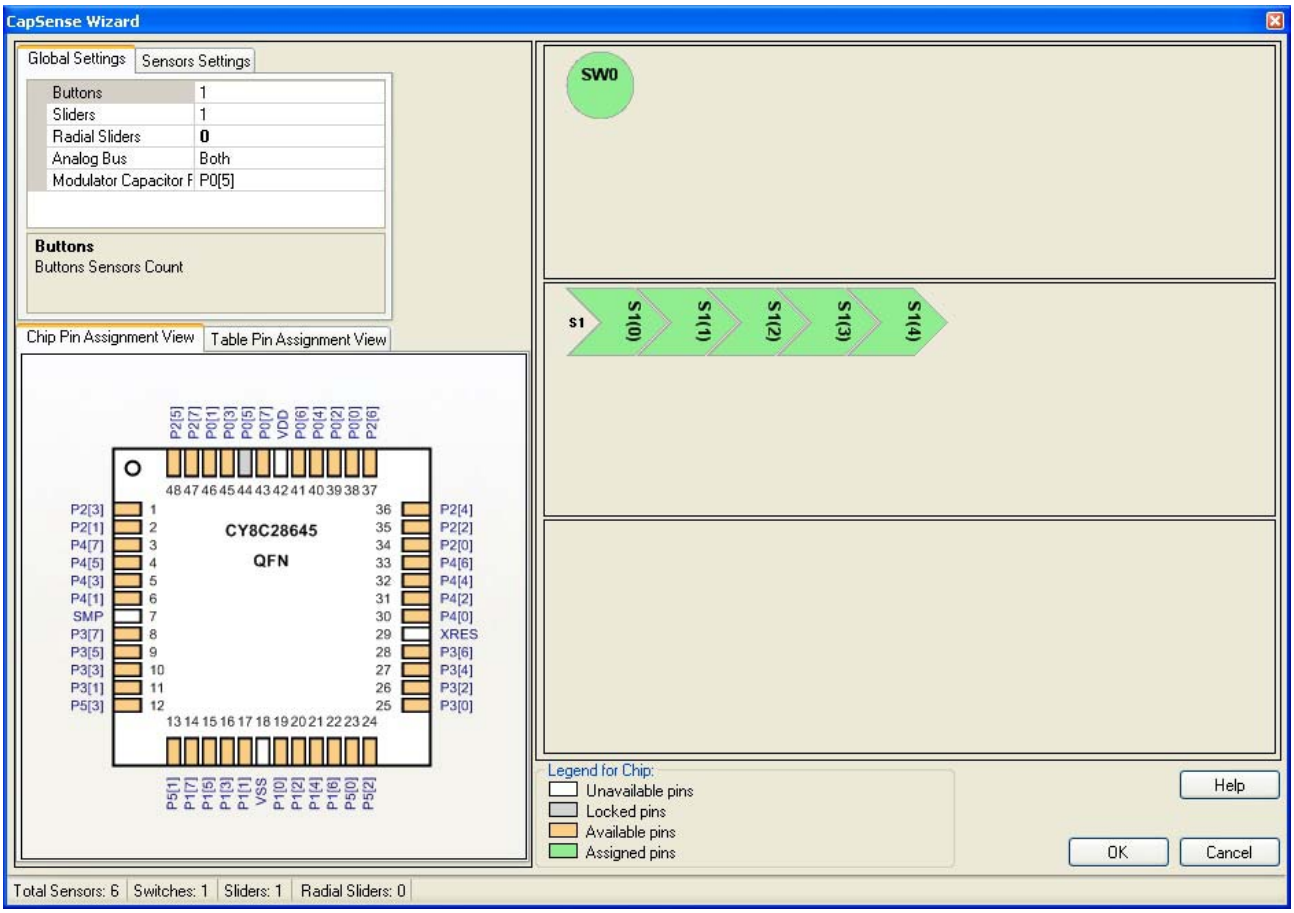
向导

CSD 向导用于设置 CapSense 按键和滑条的引脚分布。可以选择所需的配置，并通过使用拖放界面来分配按键和段。

1. 要访问向导，请在 “Device Editor Interconnect View” （器件编辑器互连视图）中右键单击任意 CSD 模块，然后左键单击选择 “CSD Wizard” （CSD 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。



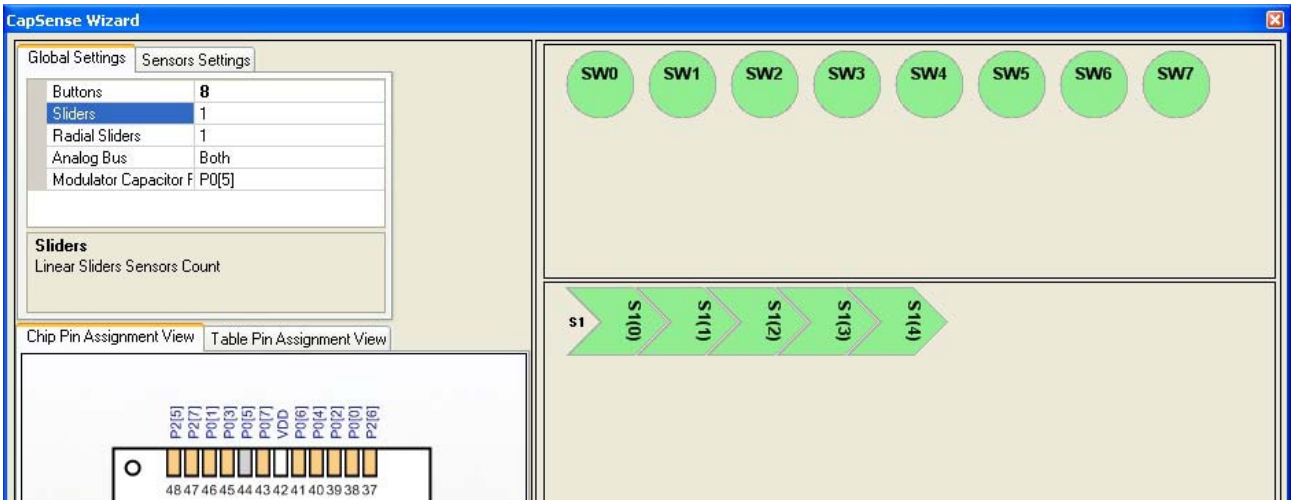
向导引脚图标

- 白色 — 引脚不能作为 CapSense 输入使用。
- 灰色 — 引脚被锁定。这种情况有两种可能的原因。第一种可能是另一个用户模块（如 LCD 或 I^2C）已占用了该引脚。第二种可能是更改了引脚的默认名称。要恢复到其默认的名称，请在 “

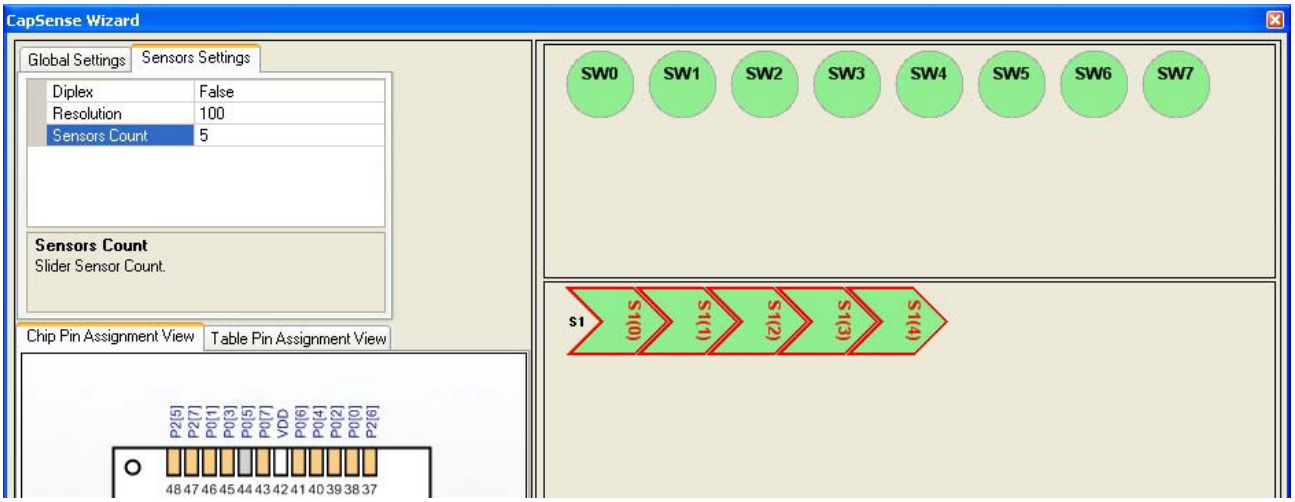
引脚分布 ” 视图中展开引脚，然后从 **Select** （选择）菜单中选择 **Default** （默认）。现在可在该向导中分配该引脚。

- 橙色 — 引脚可用于分配。
- 绿色 — 引脚已作为 CapSense 输入进行分配。

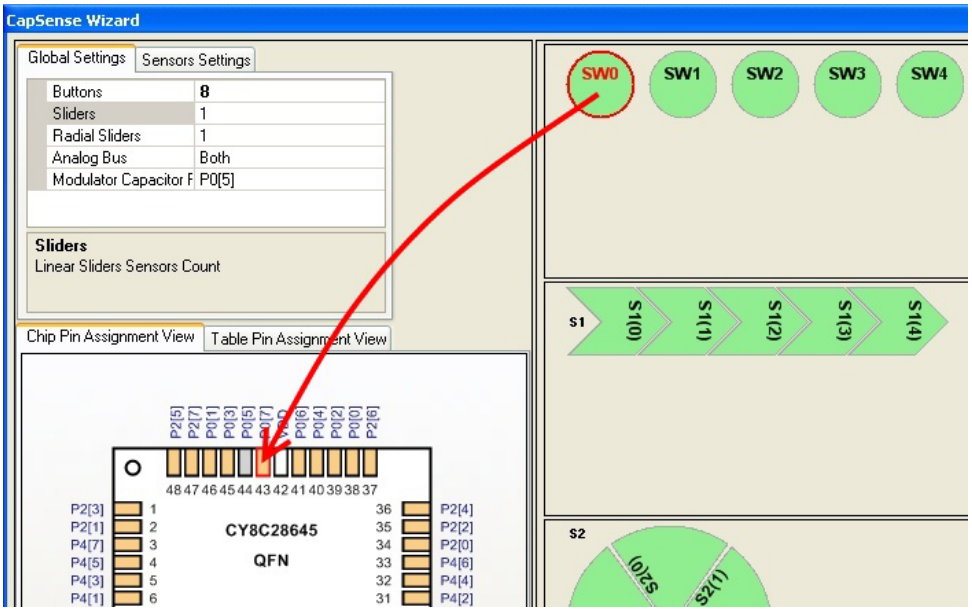
3. 键入独立传感器的数量。传感器数量被限制为可用引脚的数量。
4. 键入滑条的数量。



5. 单击滑条以启用传感器设置。选择 **Sensor Settings**（传感器设置）引脚。键入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 5，最大值受限于引脚数量。



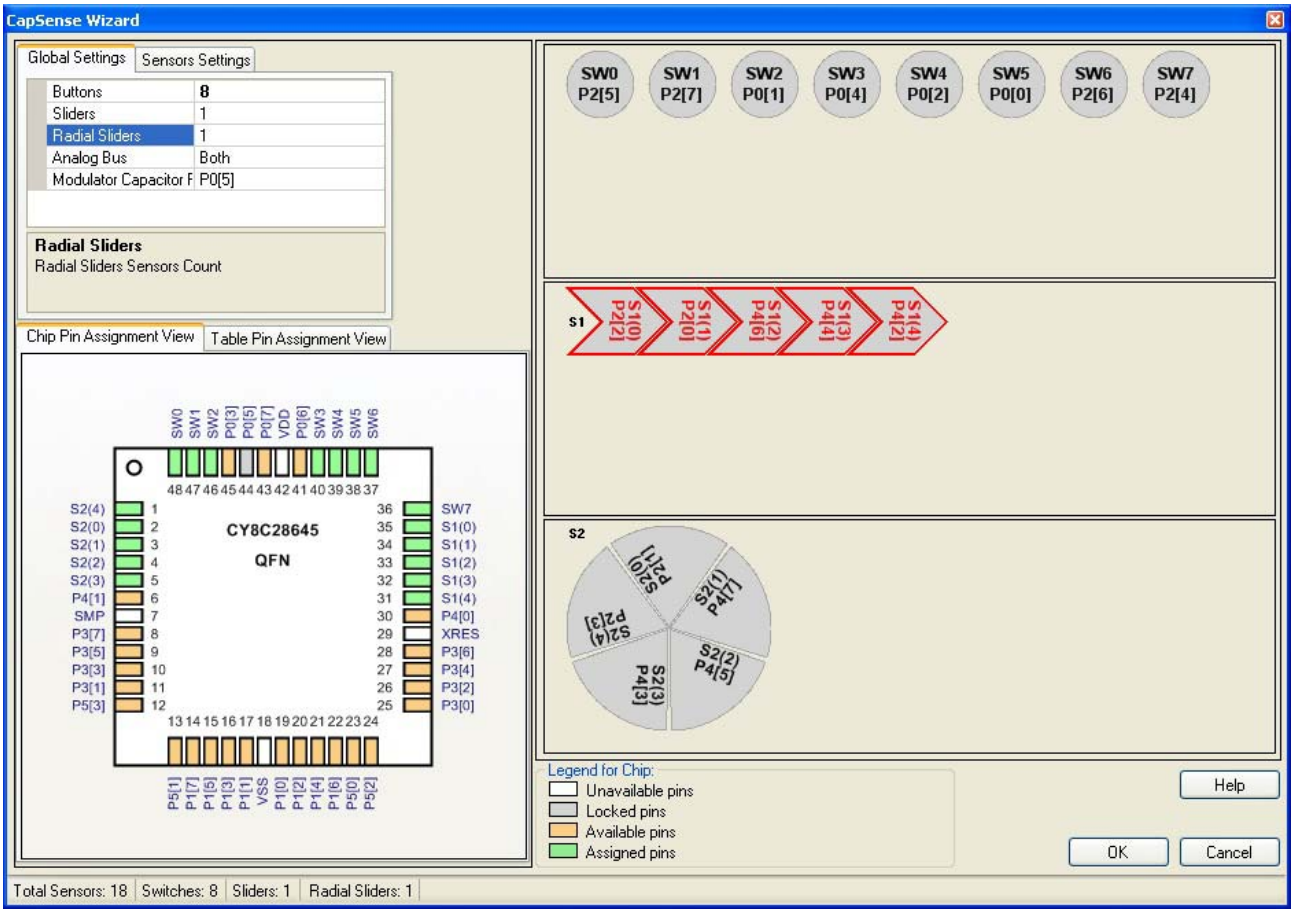
6. 键入输出分辨率。最小值为 5。最大值为 (传感器所用的引脚数量 – 1) x 2^8 – 1，对于双工滑条此值为 (2 x 传感器所用的引脚数量 – 1) x 2^8 -1。
7. 如果需要，请选择 “ 双工 ”。这样会把为传感器选定的引脚数量映射为板上传感器位置数量的两倍数值。仅显示了双工传感器的前半部分；后半部分按前面 “ 双工 ” 一节所述的内容自动映射。有关引脚连接的双工表，请参见 “ 双工 ” 一节。
8. 左键单击传感器，将其拖动到任意可用引脚。端口引脚在选定后将变为绿色，无法再用于分配。通过将传感器拖离端口引脚，可更改传感器分配。



9. 对于其余独立传感器，重复操作即可。

10. 将单个滑条传感器映射到物理端口引脚的操作与单个传感器的映射操作相同。

11. 单击 **OK** 按键以接收数据，然后返回到 PSoC Designer。



传感器放置现在已经完成。在器件编辑器窗口中右键单击，选择 **Refresh** 以更新引脚连接。

设置用户模块参数，并生成应用。如果需要，可以立即对示例项目进行调整。

要在 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。

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。

向导滑条设置

双工

仅适用于滑条。允许您使用一个引脚监控两个电子传感器以提高分辨率。更多有关信息，请参阅 “ 双工 ” 一节。

滑条分辨率

对于滑条和旋转滑条，该值设置了 CSD_wGetCentroidPos API 所返回值的范围。如果某个滑条传感器处于活动状态，该函数会将数值从零恢复为 CSD 向导中设置的分辨率值。 CapSense 算法根据相邻传感器的读数，将中心位置内插到此分辨率。

向导生成的表

向导完成后，单击 “Generate Application” （生成应用）。根据您所键入的传感器数量、引脚分配、双工和分辨率，会生成一组表格。这些表格位于 CSD_Table.asm 中。

传感器表

传感器表中每个传感器条目包括两个字节。第一个字节是端口号，第二个字节是位的掩码（不是位编号）。有两个表，分别用于左右通道。表格中列出了所有的独立传感器，然后按顺序排列每个传感器。下面的示例表格包含六个传感器：

```
CSD_Sensor_Table_Right:
_CSD_Sensor_Table_Right:
      dw    0x0140  //  Port 1 Bit 6
      dw    0x0301  //  Port 3 Bit 0
      dw    0x0304  //  Port 3 Bit 2

CSD_Sensor_Table_Left:
_CSD_Sensor_Table_Left:
      dw    0x0308  //  Port 3 Bit 3
      dw    0x0302  //  Port 3 Bit 1
      dw    0x0108  //  Port 1 Bit 3
```

该表由 CSD_wGetPortPin() 子程序使用。

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，自由按键传感器对应一个条目。第一个条目始终是自由传感器。每个条目为 6 个字节。第一个字节表示传感器表中组开始的索引。第二个字节表示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法（4：已采用， 0：未采用）。第四、五、六个字节是固定点乘数，将其与所计算出的滑条质心相乘可以得出 CSD 向导中所需的分辨率。

```
CSD_Group_Table:
_CSD_Group_Table:
```

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

双工表

当某一组是滑条且采用了双工法时，将为该组生成双工表扫描顺序数据。否则，将创建标签而不放置数据。该表由两部分组成：每个滑条的传感器映射，以及每个单独滑条对其表格的引用。八个传感器滑条的典型示例：

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


 CSD_Diplex_Table:
_CSD_Diplex_Table:
db >DiplexTable_0, <DiplexTable_0
db >DiplexTable_1, <DiplexTable_1
```

参数和资源

PGA Gain （PGA 增益）

该参数仅适用于二阶调制器配置。为 ADC 设置 PGA 增益。如果使用 PSoC Designer 或 API 中所提供的 CSD_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。该参数仅适用于二阶调制器配置。默认值为 4.00。

Finger Threshold （手指阈值）

该阈值用于确定每个按键传感器的状态。如果任一个传感器处于活动状态， bIsAnySensorActive() 函数将返回 1。如果所有传感器均是关闭状态，则 bIsAnySensorActive() 函数将返回 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 （滑条组中不包含在内），这些阈值都是变量，并提供在 baBtnFThreshold[] 阵列中。可以使用 SetDefaultFingerThresholds() 函数将各阈值设置为器件编辑器中所设定的默认值。要调整单个传感器的灵敏度，请更改每个传感器的 baBtnF-Threshold[] 值。（此字节阵列的大小等于已实现的各个传感器的数量。）

取值范围为 5 到 255 ； 默认值为 40。

Noise Threshold （噪声阈值）

对于单个传感器，超过此阈值的计数值不会更新基线。对于滑条传感器，质心计算时不考虑低于此阈值的计数值。取值范围为 5 到 255 ； 默认值为 20。

BaselineUpdate Threshold （基线更新阈值）

当新的原始计数值高于当前基线，并且差值低于噪声阈值 （“Sensors Autoreset” 参数设置为“Disabled”）时，则当前基线与原始计数之间的差值被累计到一个桶中。当该桶充满时，基线会按某个值递增，并清空该桶。此参数设置了基线递增时桶必须达到的阈值。取值范围为 0 到 255。参数值越大，基线的更新速度越慢。如果需要更频繁的基线更新，请降低此参数。默认值为 200。

Sensors Autoreset （传感器自动复位）

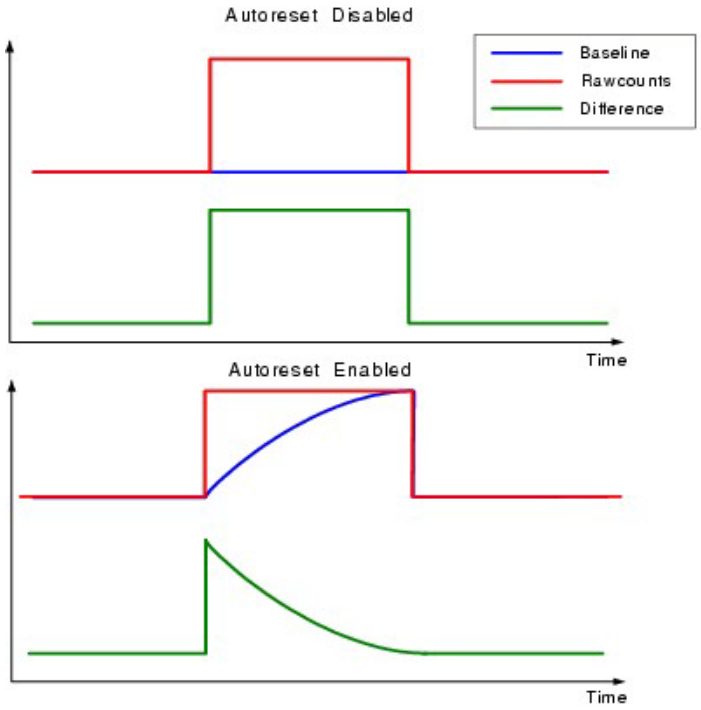
该参数确定基线是随时更新，还是仅当信号差值低于噪声阈值时才更新。当设置为 **Enabled** 时，基线随时更新。该设置限制传感器的最大持续时间 （典型值为 5 ～ 10s），但是当无任何物体触摸传感器

而原始计数突然上升时，可以阻止传感器始终打开。原始信号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。

如果将此参数设置为 **Disabled**（禁用），则仅当原始计数与基线之间的差值低于噪声阈值参数时，才会更新基线。除非是遇到在无任何物体触摸传感器而原始计数突然上升时传感器始终打开的问题，否则应将此参数保持为 “Disabled” 状态。默认设置为禁用状态。

下图说明了此参数对基线更新的影响。

图 8. 传感器自动复位参数



Hysteresis （迟滞）

“ 迟滞 ” 参数会增大或减小手指阈值，具体取决于传感器当前处于活动还是非活动状态。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之和。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才被认为是有效的状态转换条件。该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，会评估带有迟滞的阈值。可以使用 bIsSensorActive() 或 baSnsOnMask[] 阵列的返回值监控传感器状态。取值范围为 0 到 255，但是必须小于 “ 手指阈值 ” 参数设置的值。

只有正确选择高级决策逻辑参数，才能高效补偿环境因数 （温度、湿度变化等） ，抑制嘈杂信号（ESD，电源尖峰脉冲），并在各种情况下提供可靠触摸检测。默认设置为 10。

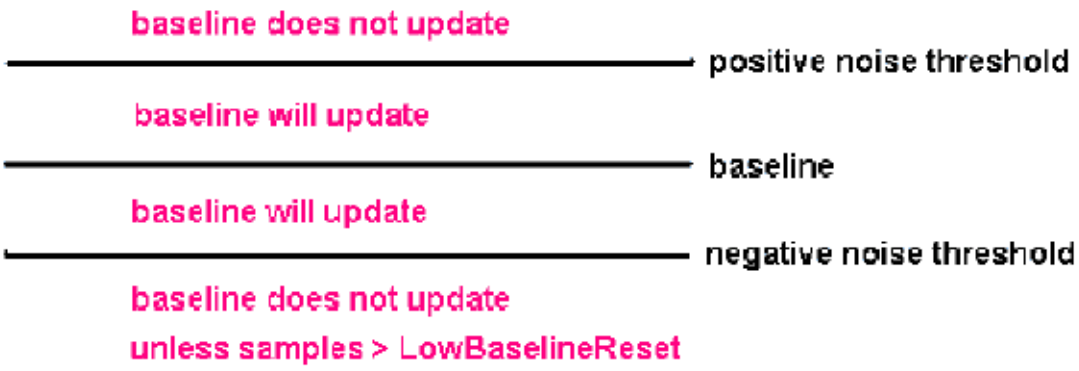
Debounce （去抖动）

“ 去抖动 ” 参数为传感器活动状态的切换添加了一个去抖动计数器。传感器要想从非活动状态切换到活动状态，差值必须在指定的采样数内保持超过手指阈值与迟滞之和。防抖动计数器由 bIsSensorActive 或 bIsAnySensorActive API 函数递增。

取值范围为 1 到 255。如果将该参数设置为 1，则不提供防抖动。默认设置为 3。

NegativeNoiseThreshold （负噪声阈值）

“ 负噪声阈值 ” 参数会添加一个负的差值计数阈值。如果当前的原始计数低于基线，且二者之差大于此阈值，则不更新基线。但是，如果当前的原始计数处于较低状态（差值大于阈值）以获得 LowBaselineReset 参数所指定的样本数量，则对基线进行复位。默认设置为 20。



LowBaselineReset （低基线复位）

LowBaselineReset （低基线复位）参数与 NegativeNoiseThreshold （负噪声阈值）参数配合使用。如果采样计数值低于基线与指定数量的样本的 NegativeNoiseThreshold 之差，则基线会设置为新的原始计数值。此参数实际上是对复位基线所需的异常低的样本数值进行了计数操作。它通常用来纠正启动时手指已放在传感器上面的情况。默认值为 50。

Scanning Speed （扫描速度）

此参数影响传感器的扫描速度。选项包括 **Fast**（快速）、**Normal**（正常）和 **Slow**（慢速）。默认设置为 Normal。较慢的扫描速度具有以下优势：

- 提高信噪比
- 更好地应对电源和温度的变化
- 需要较少的系统中断延迟；可以处理更长的中断

扫描速度和分辨率在以下方面影响 VC1 分频器：

扫描速度	VC1	
	二阶 Delta-Sigma 调制器 ACC+ASD 模块	一阶 Delta-Sigma 调制器 ACE+ASE 模块
快速	3	2
正常	4	4
慢速	8	8

Resolution （分辨率）（一阶调制器）

此参数用于确定扫描分辨率（以 “ 位 ” 为单位）。可能的选项包括 8 位、10 位和 12 位。N 位的扫描分辨率的最大原始信号为 2^N-1。

增大分辨率可提高触摸检测的灵敏度和信噪比。默认设置为 12。

表 11. 对于 24 MHz IMO 操作，扫描时间 （单位为 µs）与扫描速度和分辨率的关系

分辨率 （单位为 “ 位 ”）	扫描速度		
	快速	正常	慢速
8	85	130	260
10	130	260	510
12	260	510	1020

注意： 扫描时间是两个传感器扫描之间的时间间隔。此时间包括传感器的设置时间、调制器稳定延迟、样本转换间隔以及数据预处理时间。

Resolution （分辨率）（二阶调制器）

此参数用于确定扫描分辨率（以 “ 位 ” 为单位）。可能的选项包括 10 位、12 位和 14 位。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 位的扫描分辨率的最大原始信号为 2^N-1。增大分辨率可提高触摸检测的灵敏度和信噪比。默认设置为 12。

表 12. 对于 24 MHz IMO 操作的情况，扫描时间 （单位 µs）与扫描速度和分辨率的关系

分辨率 （单位为 “ 位 ”）	扫描速度		
	快速	正常	慢速
10	124	136	296
12	220	220	548
14	428	552	1060

注意： 扫描时间是两个传感器扫描之间的时间间隔。此时间包括传感器的设置时间、调制器稳定延迟、样本转换间隔以及数据预处理时间。

Analog Bus （模拟总线）

该参数指定要使用一个还是两个模拟复用器总线。如果使用 AnalogMUXbus_0，只有连接至奇数引脚（除引脚 P0[7] 外）的传感器被扫描。默认设置是都使用这两个模拟总线。

Modulator Capacitor Pin （调制器电容引脚）

此参数设置引脚，以将其连接至外部调制器电容 （C_{mod}）。从以下可用引脚选择。默认设置为 P0[5].

Compensation IDAC （补偿 IDAC）

该参数仅适用于 IDAC 配置。取值范围为 0 至 255 （默认值为 0）。设置 0 值时将禁用补偿 IDAC。

IDAC

该参数仅适用于 IDAC 配置，电容测量范围取决于此参数。该值越高，范围便越大。调整 IDAC 值，以获取大约为整个范围的 50-70% 的原始计数。通过使用 CSD_SetIDACValue API 函数，可以在运行时更改该参数。

取值范围为 1 到 255 ；默认值为 200。

IDAC Range （IDAC 范围）

该参数仅适用于 IDAC 配置，用于设置 IDAC 的当前乘数。该设置的结果不同于双通道配置的结果。下面显示了双通道配置结果：

设置	作用
1X	最大 IDAC 电流为 19.92 µA
4X	最大 IDAC 电流为 91.03 µA
16X	最大 IDAC 电流为 318.75 µA
32X	最大 IDAC 电流为 637.50 µA

在双通道配置中，将具有大电容的传感器连接到左通道。默认设置为 x32.

Prescaler Period （预分频器周期）

该参数仅适用于预分频器 + PRS8 配置。该参数用于设置预分频器周期寄存器，并确定预充电开关的输出频率。该参数仅适用于带预分频器的配置。预分频器周期值的范围为 1 到 255。

建议使用值 2n – 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。默认值为 7。

Shield Electrode Out （屏蔽电极输出）

当预先源为 VC2 时，该参数不可用。可以从其中一个备用数字行总线（Row_0_Output_0 到 Row_0_Output_3) 中选择屏蔽电极信号源。所选的走线将连接到任意的三个引脚。将 ‘Row LUT Function’ （行 LUT 功能）设置为 ‘A’。默认值为 “None” （无）。

PRS Polynomial （PRS 多项式）

当预先源为 VC2 时，该参数不可用。该参数将 PRS 多项式设置为基于 PRS 的配置。有以下两个选项：

- 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，因此终端器件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。

默认设置为 “ 较短 ”。

Auto Calibration （自动校准）

该参数仅适用于 IDAC 配置，使能或禁用自动校准 API 函数。默认设置为禁用状态。

Autocalibration 自动选择可能的 IDAC 值以获取分辨率范围的一半的原始计数。这会降低 CapSense 算法的整体灵敏度，但是它允许开始调试过程时快速获取可读范围中的原始计数。 Autocalibration 使用 ROM 和 RAM 资源，增加了启动时间。如果校准后的原始计数值小于分辨率范围的一半，则应当增大 IDAC 范围或降低预充电频率。 Autocalibration 用于部分提高功能配置。

Reference （参考）

该参数仅适用于一阶调制器配置。此参数设置比较器参考值。参考值来自内部电阻电压分频器。默认设置为 ASE10。

Ref Value （参考值）

该参数仅适用于一阶调制器配置。此参数设置比较器参考值。参考值来自内部电阻电压分频器。零表示最小参考值 （1/4 Vdd）。八表示最大参考值 （3/4 Vdd）。参数增加时，参考电压将呈线性增加。当参考值增大时，灵敏度会下降，但是对屏蔽电极的影响增大。

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

当 Vref = ASExx 时，有效值范围为 0 至 15。默认值为 4。

Feedback Resistor Pin （反馈电阻引脚）

该参数仅适用于外部泄露电阻（Rb）配置。此参数设置引脚，以将其连接至外部反馈电阻（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。

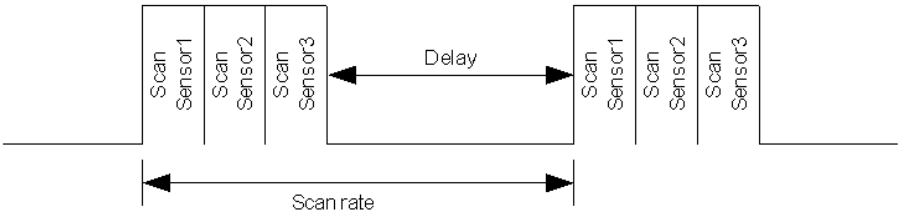
提示：如果这些引脚中的一些引脚用于其他用途 （例如，分配用于传感器连接），它们在用户模块参数列表中不可选择。

该参数取决于比较器的放置。默认引脚为 P1[1]。

传感器扫描速率选择的指南

扫描速率被定义为指传感器被扫描的速率。下图显示的是一个 3 按键设计的示例。设计中的所有传感器按顺序进行扫描，而且在各次扫描之间存在延迟。

图 9. 典型传感器扫描



为了确保基线能正常运行，推荐保持设计中的扫描速率至少为 15 ms。这意味着具有较少传感器的设计必须添加延迟，以使传感器扫描速率等于或大于 15 ms。具有更多传感器的设计不需要任何延迟，因为扫描所有传感器本身可消耗 15 mS。良好的设计会使 CapSense 控制器（而不是使固件延迟子程序）进入睡眠模式，以节能。

应用编程接口 （API）

应用编程接口（API）函数作为用户模块的一部分提供，使您能够更高级别处理该模块。本部分指定每个函数的接口，以及包含文件所提供的相关常量。

只能将此用户模式的一个实例放置在项目中，且它也应用于可加载的配置。每次放置用户模块时，都会为其分配一个实例名称。默认情况下， PSoC Designer 向给定项目中此用户模块的第一个实例分配 CSD_1。可将该值更改为符合标识符语法规则的任意唯一值。分配的实例名称成为每个全局函数名称、变量和常量符号的前缀。在下面的说明中，为了简单起见，实例名称缩写为 CSD。

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，则调用函数要保留在调用前的 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 版本起已强制使用。 C 编译器自动遵循此要求。汇编语言编程人员也必须保证他们的代码遵守该策略。虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们在将来是否也被保留。

对于大型储存器模型器件，调用程序需要保留 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 等寄存器中的所有值。尽管一些寄存器现在未被修改，但是无法保证在将来的版本中也会如此。

提供了进入点以初始化 CSD，启动其采样，并停止 CSD。在所有情况下，模块的实例名称会替换下列进入点中显示的 CSD 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。但是，可以出于调试目的检查这些值。例如，可以使用绘图工具来显示阵列的内容。下面显示的是几个全局阵列：

- 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数据。阵列大小与传感器数量相等。通过下列函数更新 CSD_waSnsBaseline[] 阵列：

- 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信号。阵列大小等于传感器数量。通过下列函数更新 CSD_waSnsResult[] 数据：

- CSD_ScanSensor() ；
- CSD_ScanAllSensors()。

CSD_waSnsDiff []：这是一个整数阵列，其中包含每个传感器中原始数据与基准数据之间的差值。阵列大小与传感器数量相等。

CSD_baSnsOnMask[]：这是一个字节阵列，用于保持传感器的开 / 关状态 （针对按键或滑条传感器）。CSD_baSnsOnMask[0] 包含传感器 0 到 7 的掩码位 （传感器 0 的是位 0，传感器 1 的是位 1）。CSD_baSnsOnMask[1] 包含传感器 8 到 15 的掩码位 （如果需要），等等。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，关闭时位值则为 0。CSD_baSnsOnMask[] 数据由 CSD_blsSensorActive(BYTE bSensor) 函数或 CSD_bIsAnySensorActive() 子程序更新。

表 13. API 与用户模块配置

	IBCNT	IBPRS	IBPRS8	IBVC2	IECNT	IEPRS	IEPRS8	IEVC2	RBCNT	RBPRS	RBPRS8	RBVC2	RECNT	REPRS	REPRS8	REVC2
CSD_Start()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_Stop()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_ScanSensor()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_ScanAllSensors()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_ClearSensors()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_wReadSensor()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_wGetPortPin()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_EnableSensor()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_DisableSensor()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_SetScanMode()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_SetPGAGain()	*	*	*	*					*	*	*	*				
CSD_Calibrate()	*	*	*	*	*	*	*	*								
CSD_SetIdacValue()	*	*	*	*	*	*	*	*								
CSD_SetRefValue()					*	*	*						*	*	*	
CSD_UpdateSensorBaseline()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_bIsSensorActive()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_bIsAnySensorActive()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_SetDefaultFingerThresholds()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_InitializeSensorBaseline()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_InitializeBaselines()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_UpdateAllBaselines()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_wGetCentroidPos()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_wGetRadialPos()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
CSD_wGetRadialInc()	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

CSD_Start

说明：

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

C 原型：

```
void  CSD_Start()
```

汇编：

```
lcall  CSD_Start
```

参数：

无

返回值：

无

其他影响：

**

CSD_Stop

说明：

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

C 原型：

```
void  CSD_Stop()
```

汇编：

```
lcall  CSD_Stop
```

参数：

无

返回值：

无

其他影响：

**

CSD_Resume

说明：

调用 CSD_Stop 之后，恢复用户模块操作。

C 原型：

```
void  CSD_Resume()
```

汇编：

```
lcall  CSD_Resume
```

参数：

无

返回值：

无

其他影响：

**

CSD_SetPGAGain

说明：

设置 PGA 模块的增益，覆盖 “ 器件编辑器 ” 中所设置的值。

C 原型：

```
void  CSD_SetPGAGain(byte bGainSetting)
```

汇编：

```
mov   A, bGainSetting
lcall  CSD_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。

符号名称	数值
PGA_G48_0	0x0C
PGA_G24_0	0x1C
PGA_G16_0	0x08
PGA_G8_00	0x18
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

符号名称	数值
PGA_G1_23	0xC8
PGA_G1_14	0xD8
PGA_G1_06	0xE8
PGA_G1_00	0xF8

仅用于二阶配置。

返回值：

无

其他影响：

**

CSD_ScanSensor

说明：

扫描所选的传感器。每个传感器在传感器阵列中只有唯一一个编号。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。对于双通道配置，传感器编号的范围是从 0 到最大通道传感器编号。 0xFF 表示跳过这个通道的扫描过程。

在单通道配置中：

C 原型：

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

汇编：

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

参数：

A => 传感器编号

返回值：

无

其他影响

**

在双通道配置中：

C 原型：

```
void CSD_ScanSensor(BYTE bSensorLeft, byte bSensorRight)
```

汇编：

```
mov A,  bSensorLeft
mov X,  bSensorRight
lcall CSD_ScanSensor
```

参数：

A => 左通道传感器编号

X => 右通道传感器编号

返回值：

无

其他影响

**

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 原型：

```
void CSD_ScanAllSensors()
```

汇编：

```
lcall CSD_ScanAllSensors
```

参数：

无

返回值：

无

其他影响

**

CSD_UpdateSensorBaseline

说明：

针对每个传感器独立计算得出的历史计数值被称为这个传感器的基线。此基线使用 “ 桶形裕量方法 ” 进行更新。

“ 桶形裕量方法 ” 使用以下算法。

1. 每次调用CSD_UpdateSensorBaseline()时，通过从原始计数值中减去以前的基线来计算差值计数。此差值存储在 CSD_waSnsDiff[] 阵列中，用户可以使用此值。

2. 如果禁用了传感器自动复位 （Sensors Autoreset） ，则每次调用 CSD_UpdateSensorBaseline() 时，将对差值与噪声阈值进行比较。如果差值低于噪声阈值，将被累加到虚拟的水桶中。如果差值高于噪声阈值，则不更新该桶。如果使能了传感器自动复位 （Sensors Autoreset），则无论噪声阈值参数如何，差值都会累加到虚拟桶中。

3. 一旦虚拟水桶中的累计差值计数达到 BaselineUpdateThreshold，基线会按 1 递增，且桶形裕量将复位为 0。

4. 如果差值计数低于噪声阈值，则保留在 waSnsDiff[] 阵列中的值将复位为 0。因此，此阵列不包含数值大于 0 但低于噪声阈值的元素。

C 原型：

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

汇编：

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

参数：

A => 传感器编号

返回值：

无

其他影响：

**

CSD_UpdateAllBaselines

说明：

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

C 原型：

```
void  CSD_UpdateAllBaselines()
```

汇编：

```
lcall CSD_UpdateAllBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSD_bIsSensorActive

说明：

检查给定传感器与手指阈值之间的差值计数阵列。迟滞也被计算在内。根据传感器当前是否开启，对手指阈值增加或减去迟滞值。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，则增大该阈值。此函数还可更新 CSD_baSnsOnMask[] 阵列中传感器的位。

C 原型：

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

汇编：

```
mov  A,  bSensor
lcall CSD_bIsSensorActive
```

参数：

bSensor A => 传感器编号

返回值：

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

A => 1：表示选定的传感器处于活动状态， 0：表示所选传感器处于非活动状态。

其他影响：

**

CSD_bIsAnySensorActive

说明：

检查所有传感器与手指阈值之间的差值计数阵列。针对每个传感器调用 CSD_bIsSensorActive()，以便在调用此函数后更新 CSD_baSnsOnMask[] 阵列。

C 原型：

```
BYTE  CSD_bIsAnySensorActive()
```

汇编：

```
lcall CSD_bIsAnySensorActive
```

参数：

无

返回值：

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

A => 1：表示一个或多个传感器处于活动状态， 0：表示没有传感器处于活动状态。

其他影响：

**

CSD_wGetCentroidPos

说明：

检查中心的差值阵列。如果存在，则偏移和长度都存储为临时变量，并根据 CSD 向导中指定的分辨率计算质心位置。仅在滑条是由 CSD 向导定义时，此函数才可用。

C 原型：

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

汇编：

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

参数：

bSnsGroup A => 组编号

该参数是作为滑条的特定传感器组的参考。组 0 用于按键。则组 1 和更高的组是滑条传感器组。

返回值：

滑条的位置， LSB 位于 A 中、 MSB 位于 X 中。

其他影响：

该子程序通过减去噪声阈值来修改差值计数。每次扫描后只能调用该子程序一次，以避免负差值。如果应用监控差值信号，在差值计数数据传输后调用该子程序。

如果某个滑条传感器处于活动状态，此函数会将数值从零恢复为 CSD 向导中设置的分辨率值。如果没有任何传感器处于活动状态，此函数将返回 –1（FFFFh）。如果在执行质心 / 双工算法时出现了错误，该函数将返回 –1（FFFFh）。若需要，可以使用 CSD_blsSensorActive() 子程序确定触摸了哪些滑条段。

注意：当滑条段的噪声计数大于噪声阈值时，此子例程可能会生成假的质心结果。设置噪声阈值时请务必小心 （应使之比噪声级别足够高），使得噪声不会产生假的质心。

CSD_wGetRadialPos

说明：

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

C 原型：

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

汇编：

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

参数：

bSnsGroup A => 组编号

该参数是正在使用的辐射滑条的编号。可以通过CSD用户模块向导从辐射滑条表示法的左侧获取其编号 （例如：s2 的辐射滑条编号为 2）。

返回值：

辐射滑条的位置， LSB 位于 A 中和 MSB 位于 X 中。

其他影响：

该子程序在每次扫描后只能调用一次，以避免负差值和基线更新。如果应用监控差值信号，则在差值计数数据传输后调用此子程序。

如果某个滑条传感器处于活动状态，此函数会将数值从零恢复为 CSD 向导中设置的分辨率值。如果没有任何传感器被激活，则函数返回 -1 （FFFFh）。

注意：如果滑条段的噪声计数值大于噪声阈值，则此子例程可能生成假的质心结果。设置噪声阈值时请务必小心 （应使之比噪声级别足够高），以便噪声不会产生假的质心。

CSD_wGetRadialInc

说明：

返回实际手指移位情况，即手指的当前位置与先前位置之间的差值。此函数与 CSD_wGetRadialPos() 配对使用，并采用后者生成的数据 （数据保存在内部变量中）。

C 原型：

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

汇编：

```
mov  A,  bSnsGroup
```

```
call CSD_wGetRadialInc
```

参数：

bSnsGroup A => 组编号

此参数是您使用的辐射滑条的编号。可以通过CSD用户模块向导从辐射滑条表示法的左侧获取其编号（例如：s2 的辐射滑条编号为 2）。

返回值：

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

手指移位值是手指的当前位置与先前位置之间的差值。如果在先前的扫描期间未发生触摸，倒数第二次 CSD_wGetRadialPos() 将返回 -1 （FFFFh）；如果当前没有任何触摸，则此时 CSD_wGetRadialPos() 会返回 -1 （FFFFh）

其他影响：

仅在调用CSD_wGetRadialPos() API之后，才能调用此子程序，因为它使用由CSD_wGetRadialPos()设置的内部数据 CSD_waSliderPrevPos 和 CSD_waSliderCurrPos。

CSD_InitializeSensorBaseline

说明：

通过扫描所选的传感器，加载含初始值的 CSD_waSnsBaseline[bSensor] 阵列元素。原始计数值将复制到所选传感器的基线阵列元素中。该函数可用于复位单个传感器的基线。

C 原型：

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

汇编：

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

参数：

A => 传感器编号

返回值：

无

其他影响：

**

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。原始计数值将复制到每个传感器的基线阵列中。

C 原型：

```
void CSD_InitializeBaselines()
```

汇编：

```
lcall CSD_InitializeBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSD_SetDefaultFingerThresholds

说明：

通过 FingerThreshold （手指阈值）参数值加载 CSD_baBtnFThreshold[] 阵列。如果尚未将自定义值手动地加载到 CSD_baBtnFThreshold[] 阵列中，则必须在扫描之前调用此函数。

C 原型：

```
void CSD_SetDefaultFingerThresholds()
```

汇编：

```
lcall CSD_SetDefaultFingerThresholds
```

参数：

无

返回值：

无

其他影响：

**

CSD_SetIDACValue

说明：

此函数将覆盖用户模块参数设置中的 iDAC 值。如果需要用其他 iDAC 设置值来扫描某些传感器，则使用此函数。此函数可与 CSD_ScanSensor() 函数结合使用。该函数不适用于 Rb 配置。

C 原型：

```
void CSD_SetIdacValue(BYTE bCompensationIdacValue, BYTE bIdacValue)
```

汇编：

```
mov A, bCompensationIdacValuemov X, bIdacValue
lcall    CSD_SetIDACValue
```

参数：

bCompensationIdacValue — 设置 IDAC 补偿值。取值范围为 1 至 255。

bIdacValue — 设置 iDAC 值。取值范围为 1 至 255。

返回值：

无

其他影响：

**

CSD_SetScanMode

说明：

设置扫描速度和分辨率。可以在运行时调用此函数来更改扫描速度和分辨率。此函数会覆盖用户模块参数的设置。当某些传感器需要以不同的扫描速度和分辨率进行扫描时（例如：常用按键和接近感应的检测器），此函数非常有效。常用按键可以在 9 位的分辨率和 300 µs 的扫描时间进行扫描。对于长距离的检测，接近感应的检测器经常采用低于 16 位的分辨率和长于 12 ms 的扫描时间进行扫描。此函数可与 CSD_ScanSensor() 函数结合使用。

C 原型：

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

汇编：

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

参数：

bSpeed：扫描速度

下面列出了 bSpeed 参数的常量：

常量	数值
CSD_FAST_SPEED	0x01
CSD_NORMAL_SPEED	0x02
CSD_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。

下面提供了一阶调制器的 bResolution 参数的可能常量：

常量	数值
CSD_8_BIT_RESOLUTION	8
CSD_10_BIT_RESOLUTION	10
CSD_12_BIT_RESOLUTION	12

下面提供了二阶调制器的 bResolution 参数的可能常量：

常量	数值
CSD_10_BIT_RESOLUTION	10
CSD_12_BIT_RESOLUTION	12
CSD_14_BIT_RESOLUTION	14

返回值：

无

其他影响：

**

CSD_SetRefValue

说明：

设置扫描参考值。仅在参考电压来自模拟调制器（参考参数中的 ASE11）或来自外部滤波后的 PWM/PRSPWM 信号时，此参数才有效。取值范围为 0 至 8。0 对应于提供最大灵敏度的最小参考电压。8 则用于设置最大参考电压，因而使灵敏度较低。此函数可与 CSD_ScanSensor() 函数结合使用。

C 原型：

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

汇编：

```
mov     A, bRefValue
lcall    CSD_SetRefValue
```

参数：

bRefValue — 设置扫描参考值。取值范围为 0 至 8。

返回值：

无

其他影响：

**

CSD_Calibrate

说明：

该函数将会覆盖 DAC 值的用户模块参数设置，以提取一半部分的原始计数。需要在基线初始化前调用该函数。

C 原型：

```
void CSD_Calibrate(void)
```

汇编：

```
lcall    CSD_Calibrate
```

参数：

无

返回值：

无

其他影响：

**

CSD_ClearSensors

说明：

通过在每个传感器中按顺序调用 CSD_wGetPortPin() 和 CSD_DisableSensor()，可以将其设置为非采样状态。

C 原型：

```
void  CSD_ClearSensors()
```

汇编：

```
lcall  CSD_ClearSensors
```

参数：

无

返回值：

无

其他影响：

**

CSD_wReadSensor

说明：

返回 A 和 X 中的关键原始扫描值 （分别为 LSB 和 MSB）。

C 原型：

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

汇编：

```
mov A, bSensor
lcall CSD_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。

其他影响：

**

CSD_wGetPortPin

说明：

返回指定传感器的端口号和引脚掩码。这些参数对 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()、CSD_DisableSensor()。该函数仅适用于单通道配置。

C 原型：

```
WORD  CSD_wGetPortPin(BYTE bSensorNum)
```

汇编：

```
mov  A,  bSensorNumber
```

```
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数量与滑条中包括的传感器数量之和。 CSD_wGetPortPin() 使用传感器编号来确定所选的活动传感器的端口和位掩码。

返回值：

A => 传感器位图

X => 端口编号

其他影响：

**

CSD_EnableSensor

说明：

配置所选的传感器，以便在下个测量周期中进行测量。通过 CSD_wGetPortPin() 函数可以选择端口和传感器，其中端口编号和传感器位掩码分别被加载到 X 和 A 中。驱动模式被修改，以便使所选的端口和引脚进入模拟高阻态模式并使能正确的模拟复用器总线输入。这样还可以使能比较器的功能。

C 原型：

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

汇编：

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

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

CSD_DisableSensor

说明：

禁用 CSD_wGetPortPin() 函数选择的传感器。驱动模式更改为 “Strong” （强驱动）（即 001）。这样可以将传感器有效地接地。端口引脚与 “ 模拟复用器总线 ”（AnalogMuxBus）的连接被断开。函数参数由 CSD_wGetPortPin() 函数返回。

C 原型：

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

汇编：

```
mov  X,  bPort
mov  A,  bMask
```

```
lcall  CSD_DisableSensor
```

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

固件源代码示例

示例 1. 此代码用于启动用户模块，并连续扫描传感器。可以使用通信部分将值传递给 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
}
}
```

示例 2. 下面的代码演示了两个传感器在用户模块向导中配置时的一个传感器用途示例。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
//-------------------------------------------------------------------------

#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(); // Start CSD UM
     CSD_SetDefaultFingerThresholds();  // Set default thresholds for buttons
     // Initialize baseline for sensor number "3"
     CSD_InitializeSensorBaseline(3);

     while (1)
{
          // Scan continuously sensor number "3" which is connected
CSD_ScanSensor(3);
         CSD_UpdateSensorBaseline(3); // Update Baseline for sensor 3
         if(CSD_bIsSensorActive(3)) // check if sensor 3 is touched
         {
               // Add user code here to proceed the buttons pressing
         }
     }
}
```

示例 3. 下面的示例演示了如何能够设置每个传感器的不同手指阈值级别。当多个传感器放置在不同位置上而其中一些传感器的灵敏度比其他的更高时，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
}
}
}
```

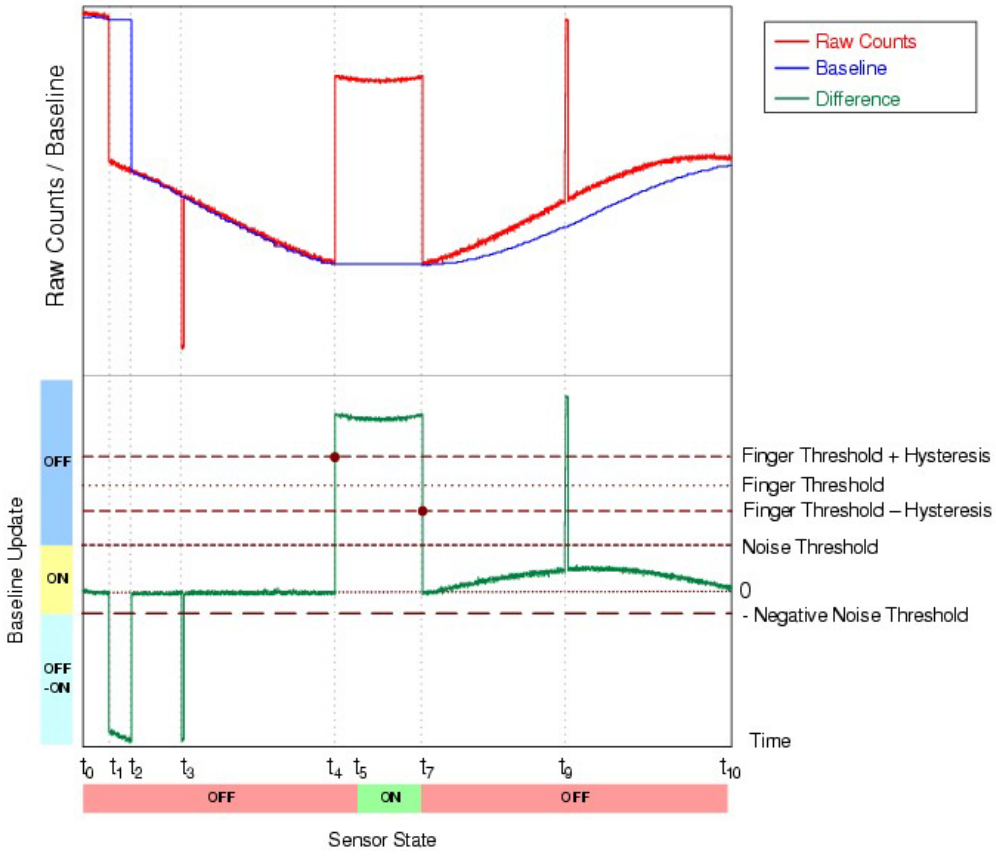
附录

下面各节介绍了用户模块数据手册中通常没有包含的信息。赛普拉斯工程师开发的详细信息，帮助您成功设计 CapSense 应用。此信息的某些部分将来会移到应用笔记中。

CSD 参数的交互

下图说明了基线更新和决策逻辑操作，对于更好地了解如何设置用户参数以获得最佳性能很有帮助。第一图说明了在 “ 传感器自动复位 ” 参数被设置为 **Disabled**（禁用）时的系统操作。第二图说明的是 “ 传感器自动复位 ” 参数设置为 **Enabled**（使能）时的情况。图中还一同显示了手指阈值、噪声阈值、迟滞和负噪声阈值与差值信号 （原始计数 — 基准）。数据是在一些人工测试中收集的，这些测试展现了原始计数慢速和快速变化时的系统操作。慢速变化可能是温度或湿度变化所致，快速变化可能是由传感器触摸、 ESD 事件或强射频场的影响触发的。

图 10. 传感器自动复位被置为禁用时原始计数、基准线值、差值信号变化的示例



在 t_0 处，原始计数接近于准水平，然后由于湿度或温度变化，开始缓慢下降。由于两次连续转变之间的原始计数变化不超过 “ 负噪声阈值 ” 参数（绝对值），因此通过跟踪原始计数最小值来更新基线，保留原始计数信号的较小值。

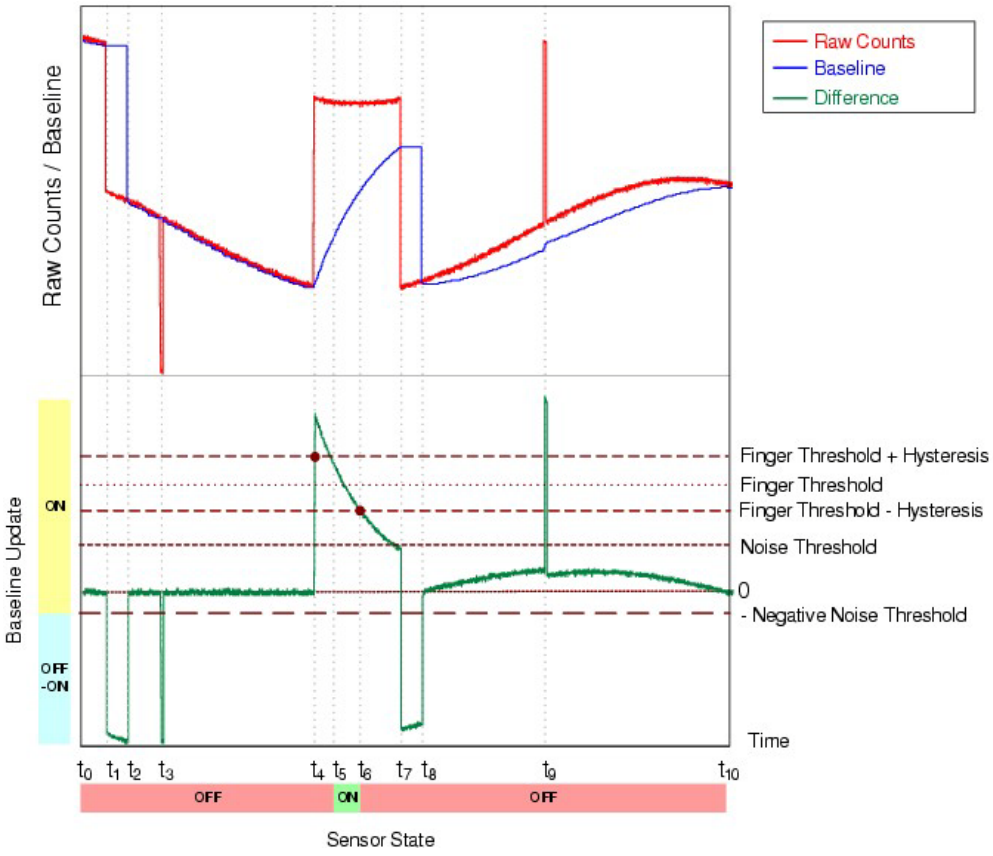
在 t_1 处，原始计数快速下降，并且负差超过 NegativeNoiseThreshold。如果器件在手指位于传感器上时被加电，过一段时间后移开了手指，则会发生这种情况。此时，基线更新机制冻结，内部超时计数器被激活。当差值信号低于 “ 低基线复位 ” 样品的 “ 负噪声阈值 ” 时，会复位基线。这是在 t_2 处发生的。

第二大负差值信号尖峰脉冲在 t_3 处发生，例如，可能已通过 ESD 事件触发此尖峰脉冲。由于采样计数中的尖峰脉冲持续时间小于 “ 低基线复位 ” 参数，因此保留基线，并对尖峰脉冲进行滤波。这样可以防止假基线复位及导致假触摸检测。

传感器是在 t_4 处被触摸的。当差值信号超过手指阈值和迟滞之和时，会激活内部防抖动计数器。如果信号超过此值的量大于防抖动样本数量，则传感器状态设置为开启。这是在 t_5 处发生的。当差值信号在 t_7 处下降到低于手指阈值和迟滞之差时，传感器将立即恢复为关闭状态。在 t_9 处短的正尖峰脉冲已被防抖动计数器滤波，这是因为样本单元中的尖峰脉冲持续时间不会超过防抖动值。

原始计数值在 t_7 与 t_{10} 之间缓慢升高。当差值信号低于噪声阈值 （传感器自动复位设置为 “ 禁用 ”），并且差值信号与漂移速率成比例时，使用桶形裕量算法来更新基线。可以使用基线更新阈值参数来控制基线更新速度。参数值越低，基线更新速会度越快。

图 11. 传感器自动复位参数设置为 “ 使能 ” 时的原始计数、基线、差值信号的变化示例



上图中的系统操作与上一实例中的操作类似，但有以下区别：

- 在 t_6 处触摸传感器时，触摸持续时间因活动基线更新算法而会减少。
- 手指抬起后，基线会在低基线复位采样 （t_8）后复位，这会短时阻止触摸检测。这用作额外的去抖动机制。

CSD 分步调试指南

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
- 外覆层厚度和绝缘常数
- PSoC 器件的电极连接阻抗
- 最终应用状况，例如：

- 电源的提供
- 温度
- 湿度
- 潮气的存在
- ESD、 EMC 或 EMI 的要求

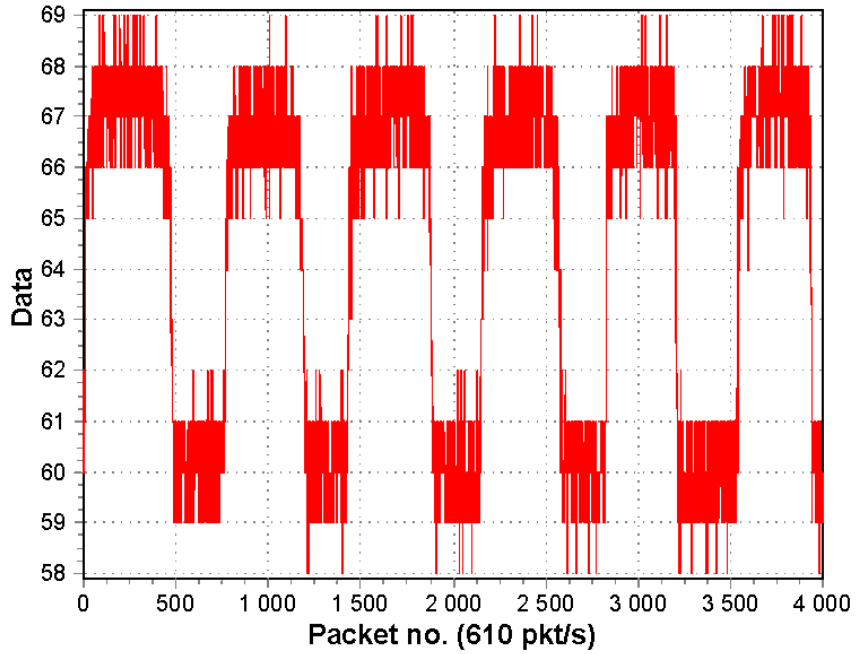
不同任务 （如：防水操作、使用高阻抗材料进行感应、接近感应的检测和通过厚外覆层的操作以及有关通过认证测试的建议）的最佳做法在单独的应用笔记中说明。

以下内容基本介绍了如何通过将 CCY3214 板作为测试示例，来配置典型 CapSense 应用中的用户模块。使用 2 mm 厚的塑料外覆层盖上感应区。请按照下面的流程配置 CSD 用户模块参数：

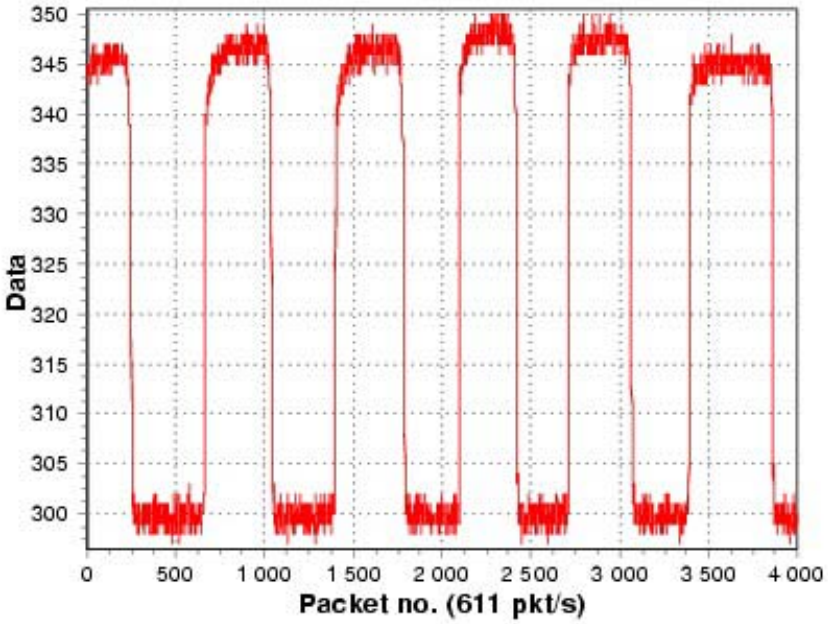
1. 准备目标电路板。组装目标应用 PCB，并在它上面固定外覆层。使用胶水或特殊胶带来进行安装。避免在 PCB 与外覆层之间留有气隙，否则会极大降低灵敏度，且由于当您触摸时气隙漂移，会导致出现许多假按键触发情况。
2. 设置用于监控数据的实时监控工具。在配置 CSD 期间，使用 PC 绘图工具以实时观察一个或多个数据系列。在用户模块调试过程中，必须注意原始计数、基线和信号差值。为此，可以使用 I^2C-USB 桥接器。在我们的测试中，使用了一个桥接器进行原始计数数据监控。另一个好的备选方案是使用 USBUART 用户模块，通过一个仿真串行端口发送调试信息。请勿使用 LCD 或其他任何数字显示屏来监控计数值，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 位 PRS。开始测试之前，在 PSoC Designer 中设置下列参数：

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

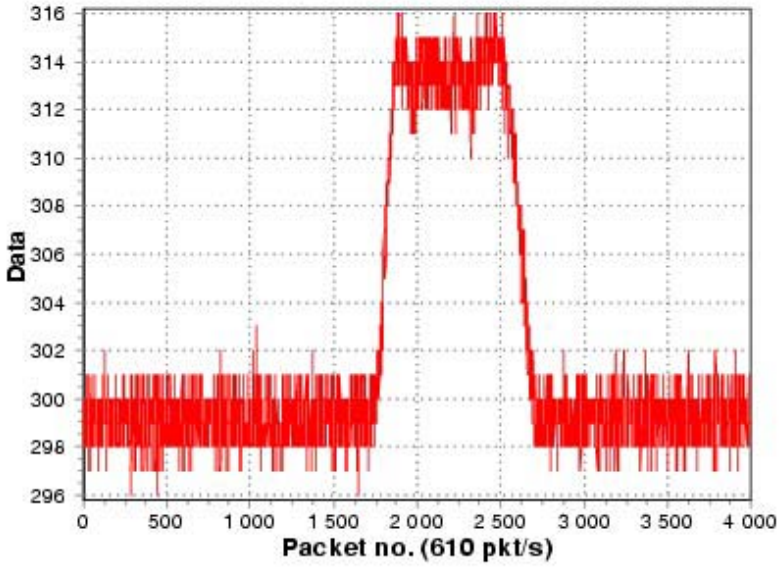
4. 在 CSD 向导中分配传感器引脚 （分配用于扫描的引脚包括 P5[7]、 P3[7] 和 P3[6]）。
5. 生成应用和示例代码。
6. 使用绘图工具监控传感器的原始计数数据，从而确认用户模块是可操作的。触摸传感器会使原始计数（CSD_waSnsResult 变量）从 59 更改为 68。



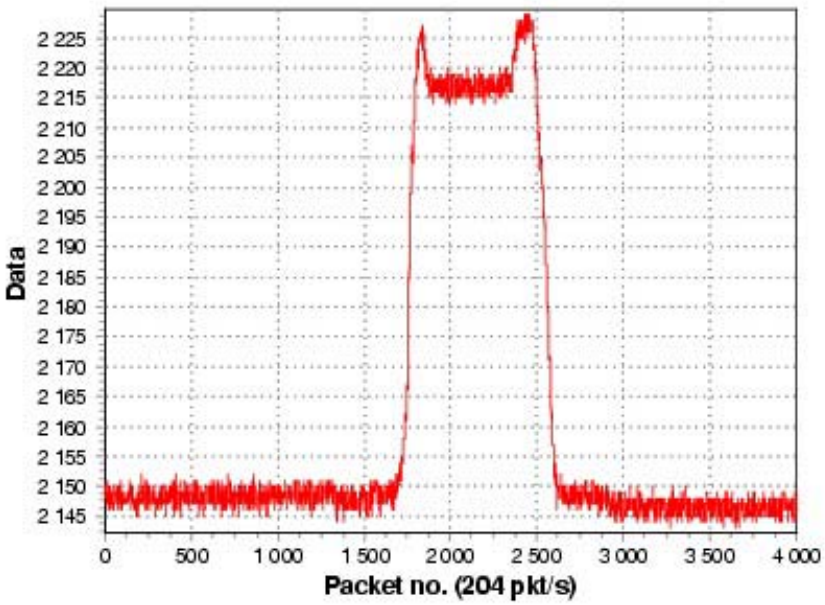
7. 调试外部组件。赛普拉斯最初使用了 5.6 nF 调制器电容 （C_{mod}）以及 1.2 kΩ 反馈电阻 R_b。在触摸情况下对来自不同传感器的原始计数值进行观察后，赛普拉斯找到了生成最大原始计数值的传感器。上图显示了来自此传感器的信号。较低的信号值对应于无手指触摸的情况，较高的信号值对应于触摸情况。通过分析来自此传感器的信号值，可以看到系统仅使用了电容代码转换器动态范围的 8%。 9 位分辨率的全范围为 Nm = 512，最大原始计数值大约为 85。这意味着通过将反馈电阻值增加到 5.1 kΩ，可以将动态范围利用率提高到建议的 60 至 70%。可以使用不同的电阻值来实现，具体取决于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。手指触摸的响应被加快。



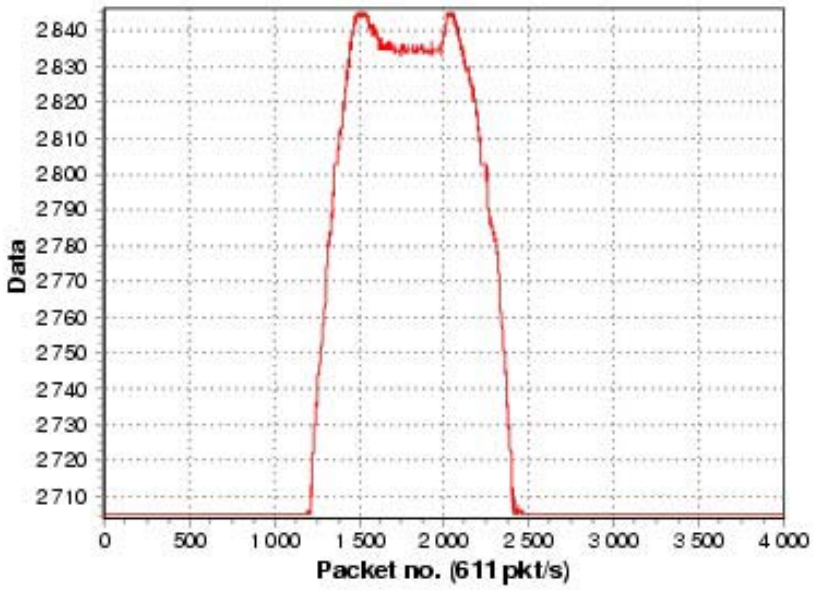
8. 对最坏情况进行调整。使用手指模拟器确保器件在不同的情况 （例如，很轻的触摸）下仍可靠工作。将 10 mm 的未连接线圈放在外覆层上，模拟最坏情况。使用绝缘物体（如火柴或牙签）将线圈移过按键。下图显示了结果。如果您的板在传感器周围使用了接地层，则可以进行此测试。如果该板是被屏蔽电极而不是接地层覆盖，可以通过用手指很轻触摸来模拟最坏情况。



9. 来自线圈的信号被识别出来，但是信噪比 （SNR）过小，不足以进行可靠的检测。差值仅大约为 9 dB。要增加灵敏度，请选择较高的扫描分辨率。在测试过程中，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。



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。以下是相同用户模式参数下来自 PRS8 配置的线圈响应 （PRS 多项式被设置为短）：

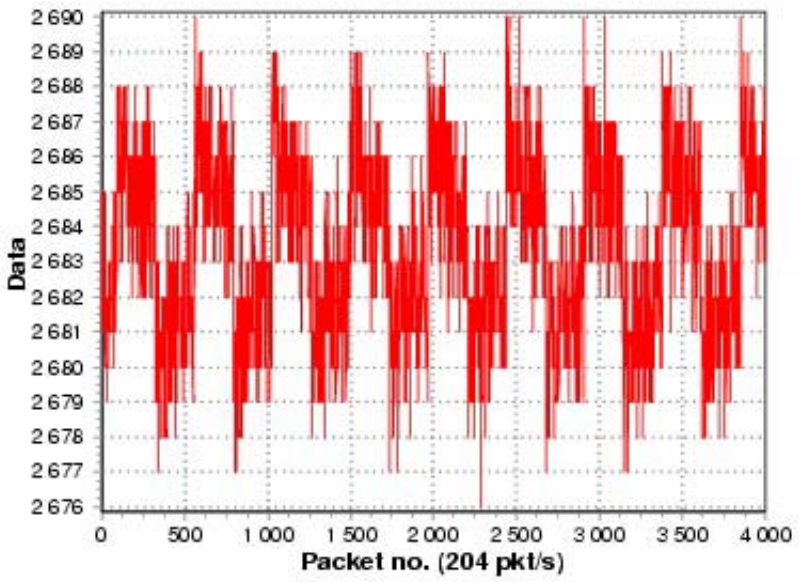


11. 与短 PRS 多项式下的 PRS8 配置相比，此配置提供更佳的噪音比。但是较短的伪随机顺序可能会导致较差的外部电气抗噪能力。

12. 设置阈值。对用户模块参数进行以下更改：

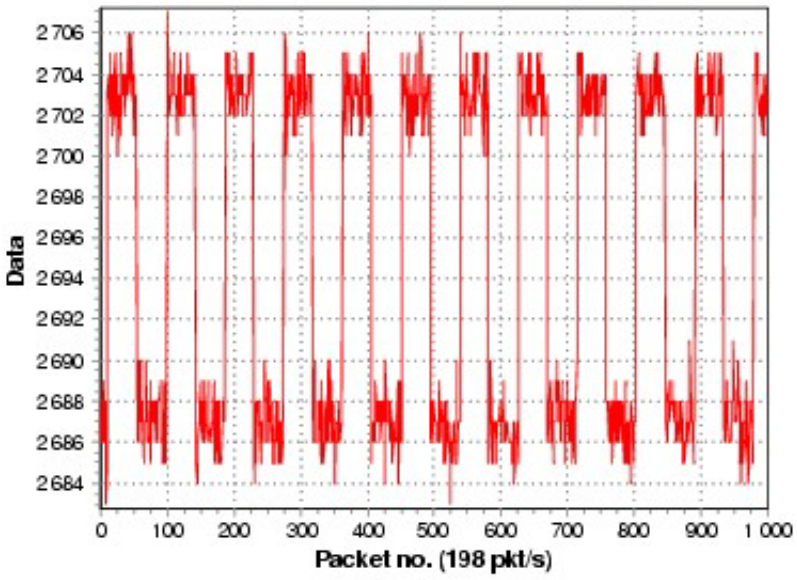
User Module Parameters	Value
FingerThreshold	40
NoiseThreshold	20
BaselineUpdateThreshold	200
Sensors Autoreset	Enabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	20
LowBaselineReset	50
Scanning Speed	Fast
Resolution	12
Modulator Capacitor Pin	P0[5]
Feedback Resistor Pin	P3[1]
Ref Value	2
ShieldElectrodeOut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

13. 设置最佳的扫描速度。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% 的尖锐电源波动。另外，假设 PSoC 器件驱动多个 10 mA 的 LED 及其 CapSense 功能。内部芯片块阻抗上的电流下降可能导致内部电源电压波动。即使存在此电压瞬变，CapSense 系统也应当继续运行。测试由于这些波动导致原始计数发生什么变化。LED 必须同时开启和关闭。睡眠定时器中断最适合执行此任务。另外，可以使用外部脉冲源模拟外部负载的开启和关闭。下图显示了在进行扫描的情况下切换 LED 时的原始计数。

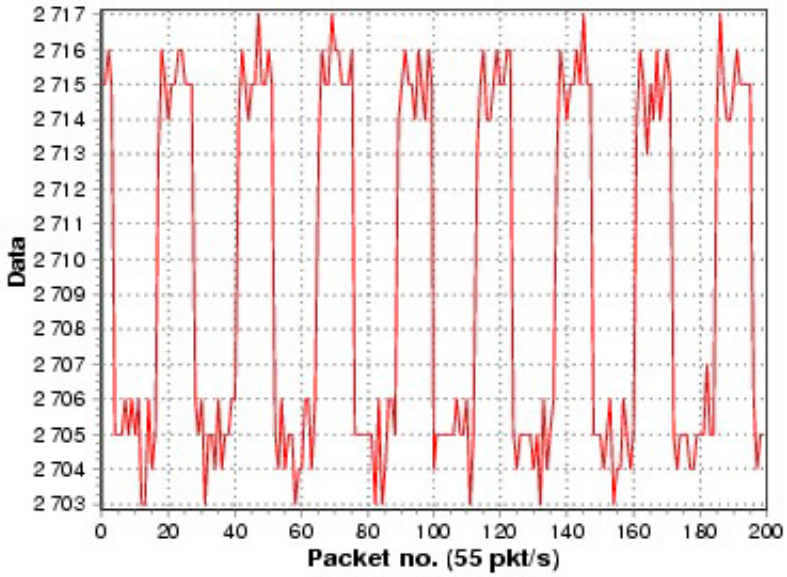


14. 如该图所示，正在进行扫描时开启 / 关闭 LED 会对于原始计数值没有明显的影响。测试电源快速变化时 CapSense 的稳定性。很慢的电源变化可由基线更新算法来处理，在大多数情况下不会产生问题。此

测试中使用了 LM1117-ADJ 电压调节器。输出电压是通过反馈电阻网络变化进行调制的，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 4.75 V 到 5.25 V 之间振荡时传感器的原始计数差值。



15. 如该图所示，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。但很轻的触摸可以导致多个触摸触发检测。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所收集的原始计数数据：



16. 如该图所示，降低扫描速度也会降低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. 调试基线更新阈值参数。应用要求最大触摸检测时间少于 1 秒。将传感器自动复位参数设置为 “ 使能 ”。检查基线更新阈值是否提供了适当补偿环境变化的基线更新速度。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，从而使原始计数下降。通过将基线自动复位成原始计数值，基线对此情况进行跟踪。因此，在大多数情况下，原始计数因环境因素而下降不应当是问题。如果原始计数由于温度变化而增加，可以将此变化解释为触摸来触发假触摸。必须调整基线值更新速度，使温度（或其他环境因素）对原始计数与基线值之差的影响远低于 “ 手指阈值 ”。原始计数与基线之差是在这些测试期间监控的。监控的值为 0，因而差值低于噪声阈值参数。在这些测试中，此参数被设置为最小值 5。这意味着预设的基线更新阈值参数提供了足够的基线跟踪速度，对于我们的应用而言，温度波动不应当是问题。

18. 设置完所有参数后，就可以运行 ESD 测试了。即使 ESD 防抖动参数设置为 “ 禁用 ”，您的应用也能通过这些测试，不会产生问题。如果需要，您可以在 ESD 测试出现问题时使能 ESD 防抖动参数。但这样会增加 RAM 缓冲区的使用大小。
19. 要求许多 CapSense 应用通过各种 EMC/EMI 兼容性测试。如果您应用存在有关 EMC/EMI 的问题，请参考 CapSense 入门的设计指南。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的辐射。您可以尝试使用带有预分频器的配置，或者使用较慢的 IMO 模块（例如， SYSCLK 在 6 MHz 而不是在 24 MHz 运行）。如果 PRS 时钟频率或预分频器周期设置发生了任何更改，您还需要调整反馈电阻，以最大程度地利用动态范围来达到最大灵敏度。
20. 如果您的应用未通过 EMC 测试，请尝试降低扫描速度并提高分辨率。这会导致 PRS 多项式序列较长，因而获得的抗噪能力会较高，但会延长传感器的扫描时间。

故障排除

- 您可以将预充电预分频器作为UART波特率时钟源使用。建议的UART速度不得小于115,200波特。对于 24 MHz 的 IMO 操作，预分频器周期值应当设置为 25。由于此值不是 2^N 的倍数，推荐使用较慢的扫描速度来获得较高的信噪比。请通过实验来测试。
- 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 常量。此延迟设置了测量开始前的调制器启动时间。减少调制器电容 C_{mod} 也会有帮助。产生这种噪声的原因是：由于内部模拟调制器低通滤波器上的时间常数很小，在以前的测量周期中调制器电容被充电到另一个电压。
- 扫描速度和分辨率均会影响信噪比 （SNR）。在一些情况下，较慢的扫描速度和较高的分辨率可获得更好的信噪比。
- 如果电极外覆层很厚，可能需要较高的分辨率和较慢的扫描速度。
- PRS多项式自动根据扫描速度和分辨率进行调整，以使PRS序列重复周期接近于样本转换的周期计数。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测试期间可提供更高的抗噪能力。
- 扫描速度越慢，调制器工作频率会越低，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构成比较器参考。这种情况下，建议的扫描速度为 “Normal” （正常）或 “Slow” （慢速）。

- Sigma-Delta 转换方法属于积分法类别。它证明了在较高的分辨率情况下具有最佳的性能。请尽量使用最长的扫描时间。使用 1 ms 时间来扫描传感器，可以获得最佳结果。
- 可以有效地使用屏蔽电极来减少杂散电容影响，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

请勿更改此用户模块使用的硬件配置。其中包括：

- GlobalOutEven_1 或 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相连。
- 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。在运行时间内，全局资源所输入的值被覆盖。
- 使用屏蔽电极时，请将行 LUT 函数设置为 **A**。

中断持续时间管理

当传感器扫描可用于 PRS16 配置时，管理中断服务子程序（ISR）的持续时间。8 位定时器的定时为 VC2。定时器较差的溢出间隔为：

公式 2

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

F_{IMO} — IMO 频率， VC1=2、 4 和 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] 引脚的始终连接，可导致 ISSP 编程错误。此时，应使用其他引脚。

时钟速度

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

版本历史记录

版本	创作者	说明
1.4	DHA	1. 最大分辨率值为 （传感器所用的引脚数量 – 1）x 2^8 – 1 ；对于双工滑条，此值为 （2 x 传感器所用的引脚数量 – 1）x 2^8 – 1。消除了 0.5 的移位，并添加了对负值的补偿。 2. 修正了向导中的引脚列表。 3. 添加了 API 与用户模块配置 4. 更改了 CY8C28xxx 系列的 P0(7) 放置错误。
1.50	DHA	增加了对 CY8C21x12 器件的支持。
1.50.b	DHA	1. 更改了滑条传感器的最大分辨率。 2. 在向导中添加了帮助文件。 3. 本用户模块数据手册中已更新了下面内容： a. 添加了 CY8C28xxx 的模拟总线的说明内容。 b. 更新了各图像。
1.60	DHA	1. 将 ‘DiplexTable’ （双工表）从 “AREA UserModules” 传输到了 “AREA lit”。 2. 将默认的 “DiplexTable” 参数值设置为 0x0112。 3. 添加了 “DiplexUsed” 参数，以提高代码的压缩能力。 4. 添加了在启动 API 结束后调用 CSD_ScanAllSensors API。

版本	创作者	说明
1.70	DHA	1. 更新了区域声明以支持 Imagecraft 优化。 2. 为本用户模块数据手册中的分辨率参数添加了符号名。 3. 解决了与 CSD 和 DelSig 用户模块共存的问题。 4. 添加了预充电功能，以便能够将 Cmod 电容预充电到参考电压。 5. 添加了对 CY8C21x34 器件上 Rb 引脚 P1[0]、 P1[4] 和 P3[0] 的支持。 6. 添加了设置错误的反馈电阻参数时的设计准则检查。 7. 添加了对 CY8C24x94 上 Rb 引脚 P1[0]、 P3[0]、 P5[0]、 P1[4]、 P3[4] 和 P5[4] 的支持。 8. 添加了滑条和辐射滑条的分辨率参数的上限值。 9. 在本用户模块数据手册中更新了下面部分： SetScanMode() API 函数的说明内容 反馈电阻引脚部分 ISSP 引脚可能发生的冲突部分 Rb 引脚参考 10. 更新了用户模块向导中的滑条和辐射滑条分辨率范围内计算。 11. 更新了用户模块向导帮助。添加了一个滑条分辨率参数的最小 / 最大值说明。
1.70.b	DHA	1. 已针对 PRS8 和带预分频器的配置将频率峰值从 FIMO/4 更改为 FIMO/2。 2. 仅针对 CY8C20xx7/S 器件将 CSD_MODE 位的设置从 ScanSensor API 转移到 Start API。 3. 仅针对 CY8C20xx7/S 器件在 CSD_Start API 中将校准分辨率从 9 位改为 12 位。
1.80	DHA	1. 将默认的 “Reference” （参考）值从 VBG 更改为 ASE11。 2. 更新了用户模块框图。 3. 本用户模块数据手册中更新了 RAM 和 ROM 的用途。 4. 消除了冗余的寄存器写入，并纠正了 PRS16 配置的屏蔽信号连接。 5. 添加了对 CYRF89x35 器件的支持。 6. 移除了 CY8C24x94 芯片架构的冗余比较器总线的使用情况。 7. CSD 未被放置时，可以释放模拟复用器资源。

版本	创作者	说明
1.90	MYKZ	1. 添加了用户模块 API 的 Resume() 函数。 2. 纠正了有关保存滑条信息的问题。 3. 更新了基线算法，用于检查负差值计数。 4. 添加了当用户尝试编译项目前未先调用用户模块向导时的编译错误信息。 5. 更新了用户向导滑条的设置算法，以考虑到自由引脚。 6. 优化了启动用户模块的功能代码。 7. 删除了反馈电阻引脚的默认值。 8. 更新了 Precharge() 函数，以纠正 Cmod 与 GND 的连接。 9. 更新了 ScanSensor() 函数，以复位 PRS。

注意： PSoC Designer 版本 5.1 在所有用户模块数据手册中都介绍了 “ 版本历史记录 ”。本数据手册详细介绍当前和先前用户模块版本之间的区别。

Copyright © 2007-2014 赛普拉斯半导体公司。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不会根据专利权或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC Designer™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标， PSoC® 是赛普拉斯半导体公司的注册商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码 （软件和 / 或固件）均归赛普拉斯半导体公司 （赛普拉斯）所有，并受全球专利法规 （美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和 / 或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定用途外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对该材料提供任何类型的明示或暗示保证，包括 （但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不另行通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于合理预计可能发生运转异常和故障，并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而导致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用于赛普拉斯软件许可协议的限制。