双 CapSense® Sigma-Delta 数据手册 CSD2X V 2.10

Copyright © 2009-2014 Cypress Semiconductor Corporation. All Rights Reserved.

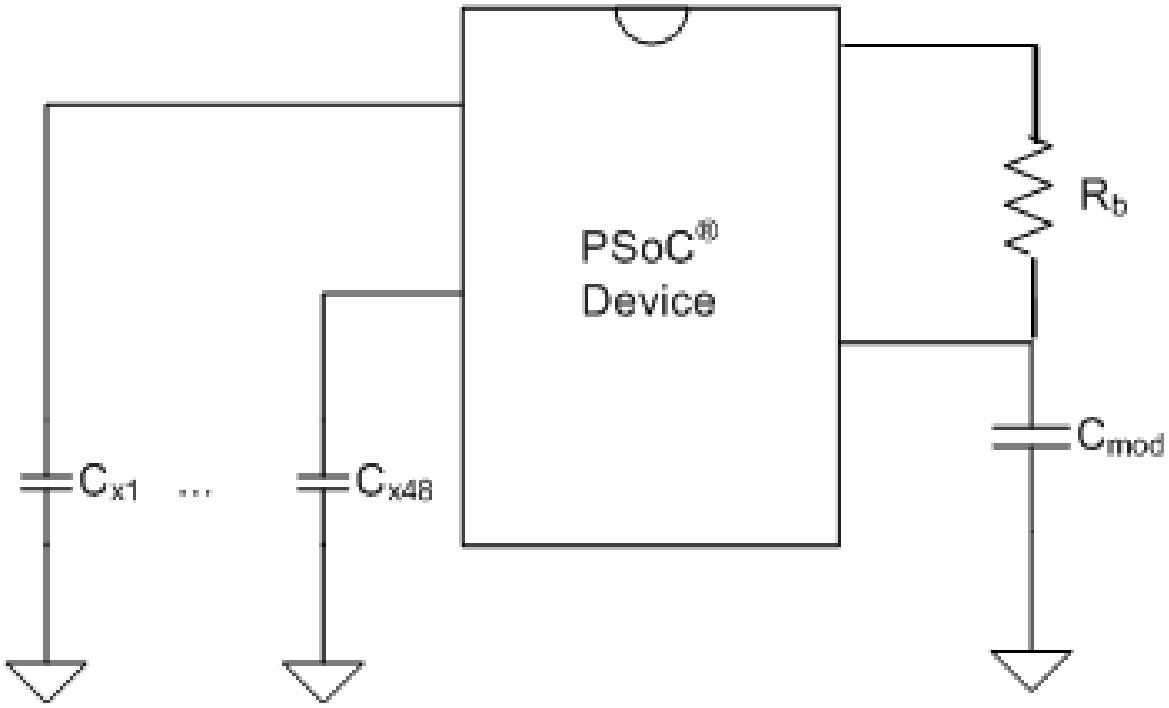
资源	PSoC® 模块				API 存储器		（所需外部 I/O 的）引脚数量
	抽取滤波器	I²C/SPI	数字	模拟	Flash	RAM	
CY8C28x45、 CY8C28x52、 CY8C28x13、 CY8C28x33							
带 IDAC 的一阶调制器	2	–	0..2	4	–	–	2
带 Rb 电阻的一阶调制器	2	–	0..2	4	–	–	4
带 IDAC 的二阶调制器	2	–	0..2	4	–	–	2
带 Rb 电阻的二阶调制器	2	–	0..2	4	–	–	4

特性与概述

- 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。
- 由于使用了线缆传感器，接近感应的检测距离可达 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传感器的各种组合。
- 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，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 CSD2X 向导完成传感器和引脚分配。
- 使用集成基准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。
- 支持 FMEA 特性，如短路测试、 Cmod 测试以及 Cmod_Rb 测试
- 背景扫描
- PC GUI 应用支持实时原始数据监控和参数优化。

CSD2X 用户模块 （使用 Sigma-Delta 调制器的电容式感应）结合使用开关电容技术和 Sigma-Delta 调制器来将感应开关电容电流转换为数字代码，从而提供电容感应功能。 CSD2X 用户模块可支持通过一阶和二阶 Sigma Delta 调制器进行的双通道 CapSense 扫描。

图 1. CSD2X 典型应用电路



快速入门

1. 如果使用，请选择并放置要求专用引脚的用户模块 （如 I2C 或 LCD）。根据需要分配端口和引脚。
2. 选择并放置 CSD2X 用户模块。
3. 在工作区浏览器中右键单击 CSD2X 用户模块，以访问 CSD2X 向导 （在本数据手册中将详细介绍该向导）。
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，以执行独立传感器、滑条传感器或触摸板。
9. 将 I2C-USB 桥连接至目标电路板，并观察信号。
10. 更改 CSD2X 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 CSD2X 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，并通过使用绝缘层、软膜或透明覆盖层与覆盖在显示屏隔。
- **电气**：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 调制器和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**：
  - 检测和补偿软件算法可以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，本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。

传感器阵列由独立传感器、滑条传感器以及做为一对互相垂直的滑条实现的触摸板组合而成。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。单独的屏蔽电极可用于屏蔽传感器阵列，以减少杂散电容，从而在水膜或水滴存在的情况下更可靠地运行。

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。

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》*中的相关章节：
  - CapSense 系统

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。这些文件可在赛普拉斯网站 www.cypress.com 上获取：

- 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

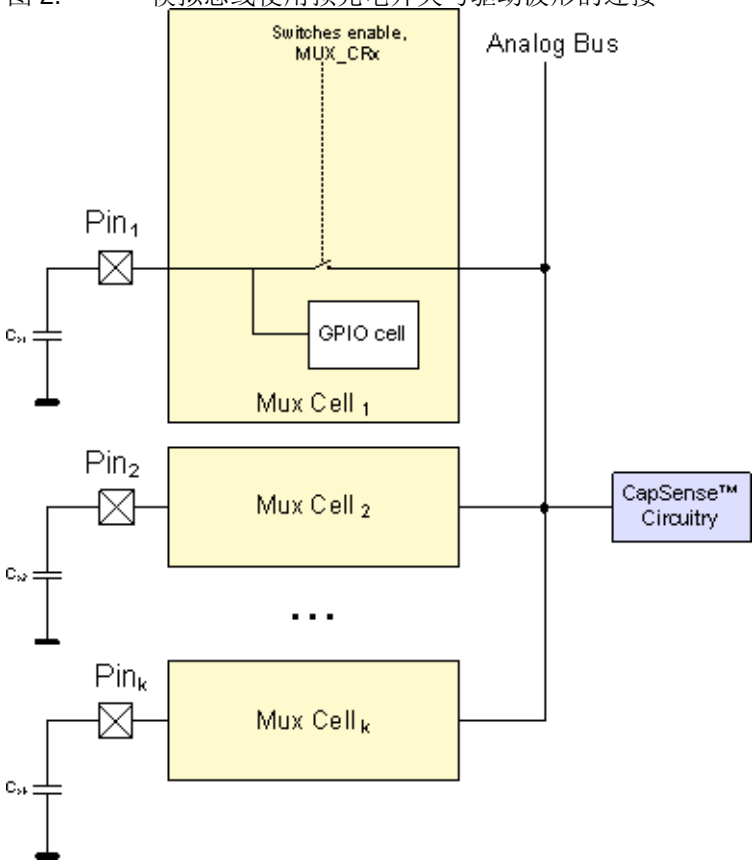
决策逻辑可通过固件实现。通过固件分析电容的测量，跟踪电容因环境因素造成的缓慢变化，并运行决策逻辑，以检测按键触摸变化以及计算滑条位置。

扫描传感器阵列

CY8C28x45 系列器件具有内置的模拟总线。可以使电容传感器连接到任意 PSoC 引脚。 CSD2X 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 调制器的调制电容以及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。

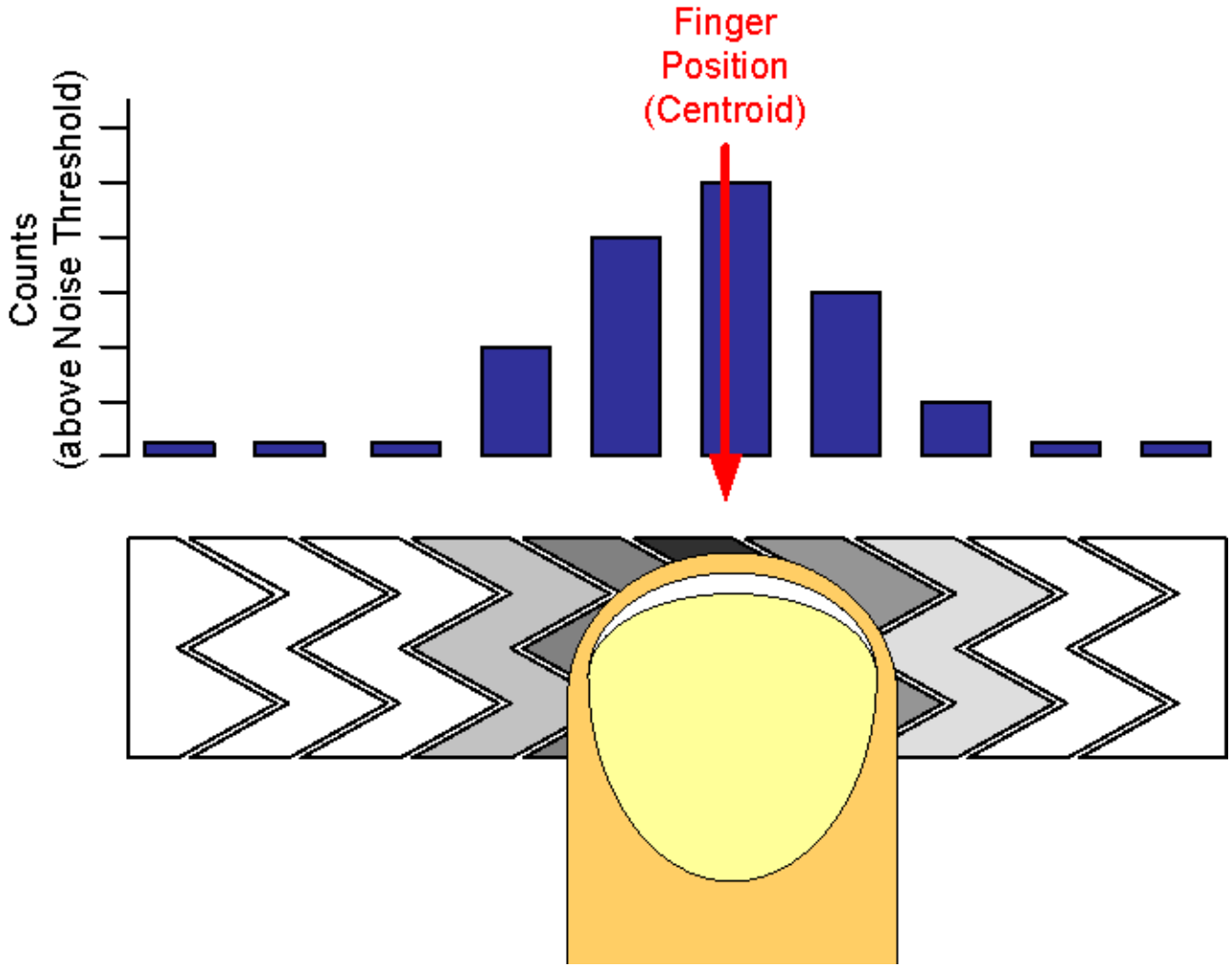
图 2. 模拟总线使用预充电开关与驱动波形的连接



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。某个传感器执行操作会使邻近的其他传感器也执行部分操作。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 CSD2X 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，上限值仅是所选 PSoC 器件提供的传感器位置的数值。

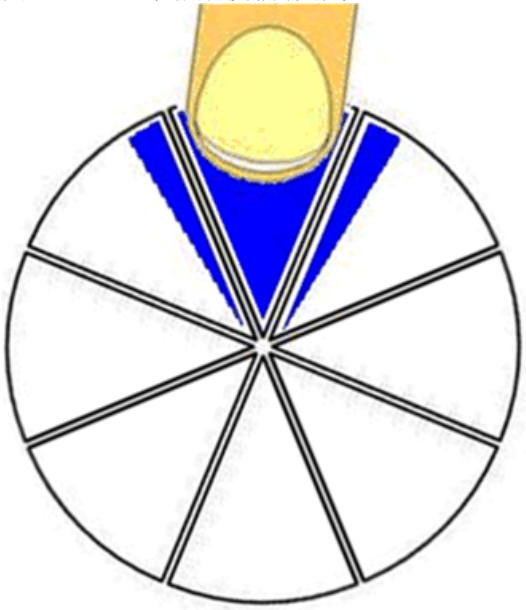
图 3. 对物理传感器位置的排序



越接近滑条一半部分的强信号，将导致最上部分产生相同程度的伪信号，但输出是杂散信号。感应算法搜索相邻强大的信号组，以确定解析的滑条位置。

辐射滑条

图 4. 手指触摸辐射滑条



CSD2X 用户模块有两个滑条类型：线性滑条和辐射滑条。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。

CSD2X 用户模块包含两个支持辐射滑条的 API 函数。第一个函数 CSD2X_wGetRadiaPos() 返回中心位置，第二个函数 CSD2X_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。

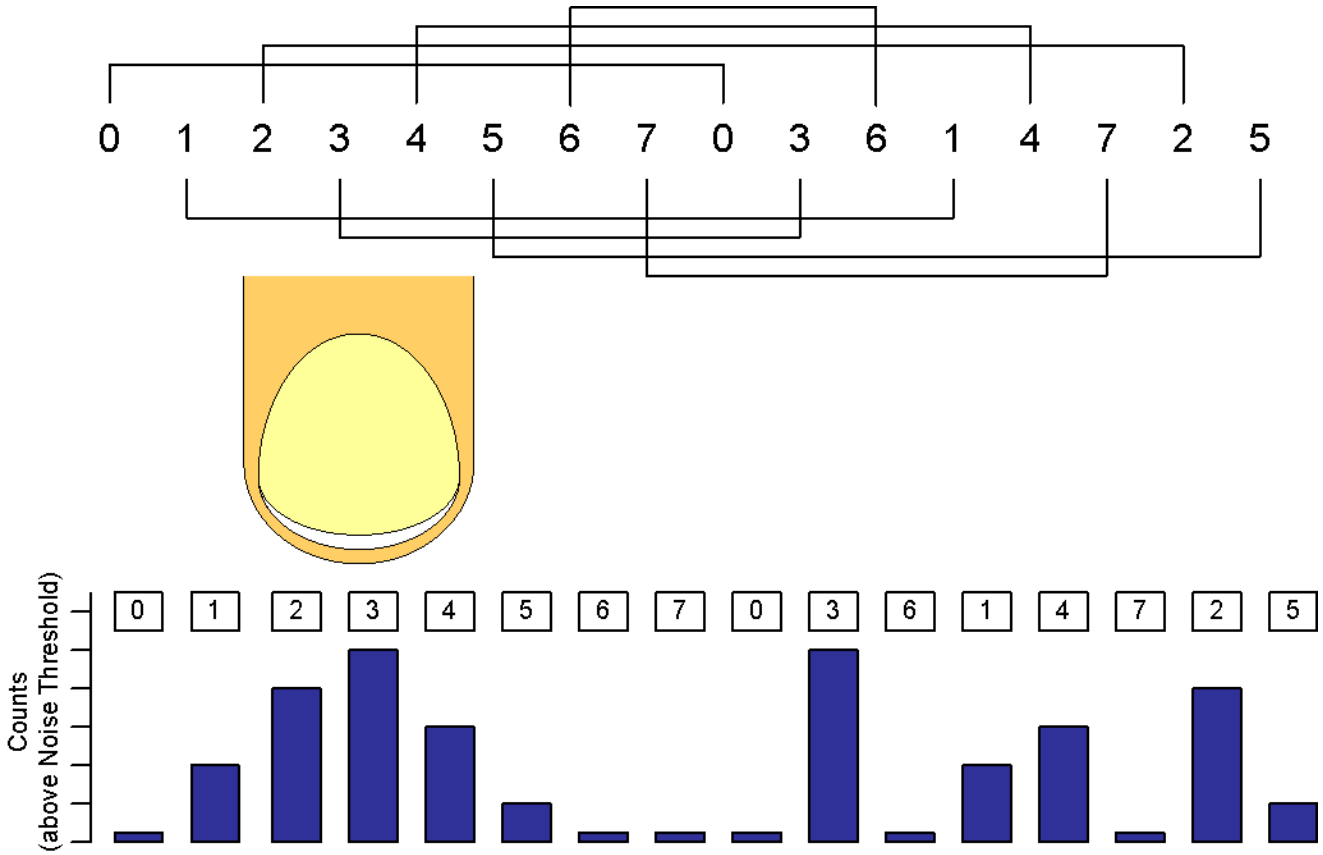
参考点 （0）位于第一个传感器的中心。线性滑条和辐射滑条的分辨率都受限制，其限制为 （传感器所用的引脚数量 - 1) x 2^8 - 1 ；对于双工型滑条，该值为 (2 x 传感器所用的引脚数量 - 1) x 2^8 - 1。

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 CSD2X 向导将端口引脚分配给这些传感器。物理传感器位置的另一部分 （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，一部分中相邻的传感器动作则不会导致另一部分中相邻的传感器动作。小心地确定此次序，并将其映射到印刷电路板上。

有许多方法可对物理传感器位置的第二部分进行排序。最简单的方法是对数值较高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 3 编制索引。

图 5. 按 3 编制索引



应当使滑条中的传感器电容均衡。根据传感器或 PCB 布局，某些传感器对可能需要更长的布线。当选择双工时， CSD2X 向导会自动生成双工传感器的索引表。此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

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

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。

为了采用一个中心值来计算插值后的位置，首先扫描传感器阵列以验证所给定的传感器位置是否有效。要求提供一定数量的相邻传感器信号，且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中心。使用从 2 到 8 个 （通常使用）传感器，通过下列公式计算中心：

公式 1

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

计算得出的值通常是分数。为了能够采用某一特定分辨率来报告中心 （例如对于 12 个传感器，分辨率范围为 0 到 100），要将中心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 CSD2X 向导中进行设置。比例值由向导计算出，并以分数值的形式进行存储。

中心分辨率的乘法器占用三个字节，相应位的定义如下：

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：

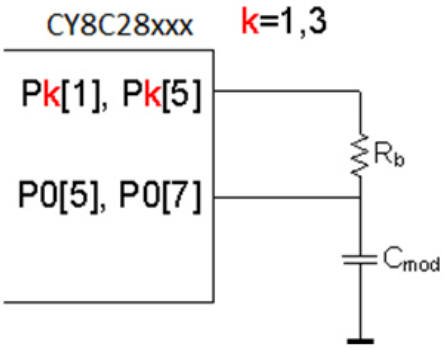
分辨率 = (传感器数量 – 1) x 乘数

中心值以 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} 和调制器反馈电阻 R_b。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。反馈电阻 R_b 可以连接到 P1[0]、 P1[1]、 P1[4]、 P1[5]、 P3[0]、 P3[1]、 P3[4] 或 P3[5] 端口引脚和电容引脚。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

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。温度电容系数并不重要。电阻值取决于总传感器电容 C_S。应通过以下方法选择电阻值：

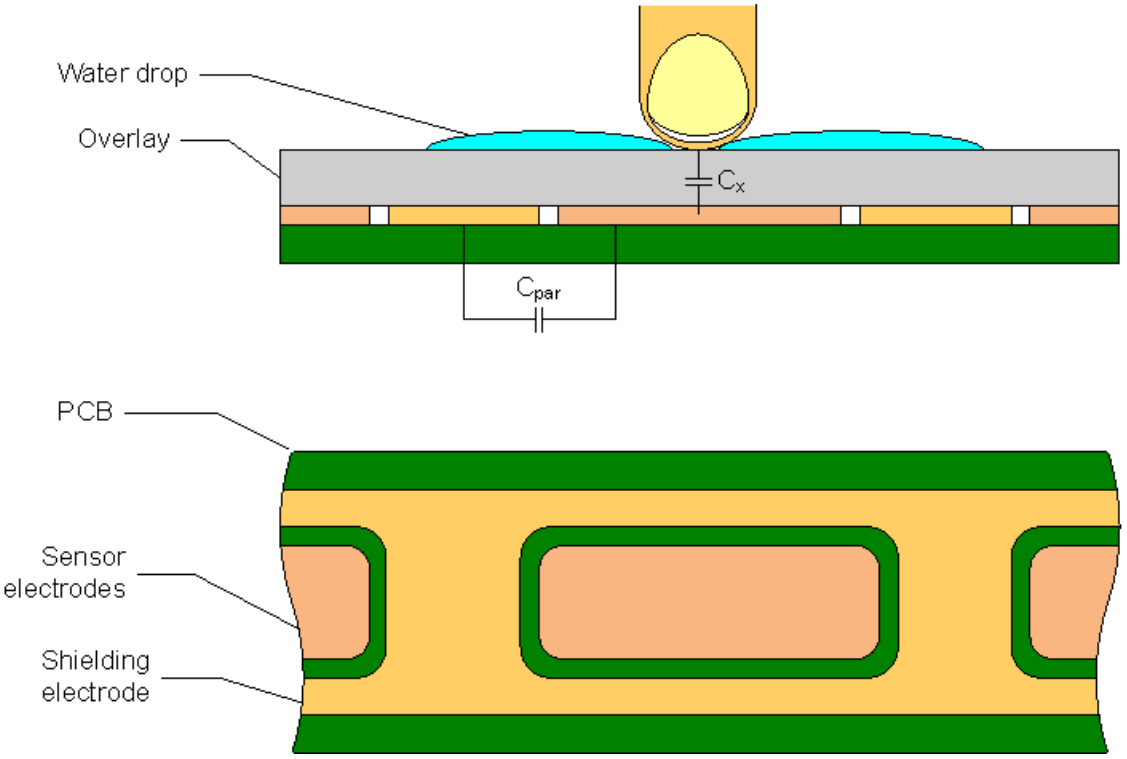
- 监控不同传感器触摸的原始信号。
- 在选定扫描分辨率的情况下，选择提供的最大读取值比全量程读取值大约低 30% 的电阻值。电阻值升高时，原始计数会增加。

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。 CSD2X 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，同时也可以减少杂散电容。

在此示例中，屏蔽电极板覆盖了按键。作为另一种替代方法，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，无需额外的接地层。

如果屏蔽电极与感应电极间存在水滴， C_{par} 将增加，调制器电流则会下降。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。

屏蔽电极可以连接到可路由到的任何 PSoC 引脚。将驱动模式设置为**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。

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形成原则。

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的选项：

类型	外部组件	UM 选择	使用场合
带隙参考	无操作	VBG	读取值与电源电压成比例。仅在电源电压稳定时才使用
模拟调制器	无操作	ASE11	建议用于大多数应用。尝试从此选项开始测试。
外部电阻电压分频器	2	模拟列（AnalogColumn）输入选择	读取值较小程度依赖于电源电压。建议 R1 = 10k； R2 = 3.6k
PRSPWM 或预分频器 -PWM 信号的外部 RC 滤波器	2	模拟列（AnalogColumn）输入选择	如果使用其他参考选择，会引起过大噪声。

您可以仅使用带隙参考或模拟调制器。外部电阻式电压分频器适用于特殊场合。

时钟源

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

- 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，应当选择必需的配置。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** （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，使用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。

PRS8 配置将 PRS8 模块用作时钟源。 PRS8 的时钟由 IMO 直接提供。 PRS8 使用较短的伪随机发生器序列，来保存数字模块。因此，使用 PRS8 配置的 CapSense 模块对于外部噪声信号的抵抗效果就较差。

带有预分频器的 PRS8 配置将 8 位计数器用作 PRS8 时钟源。该计数器源自 IMO 时钟。您可以通过更改预分频器的计数器周期，来轻松调整工作频率。预分频器配置的主要应用领域是使用高阻抗材料的电容式感应，如通过在双层触摸板器件的显示屏上使用薄而透明的 ITO 薄膜，来提供感应支持。

下表对这四种配置进行了比较：

配置	工作频率	使用的数字模块	抗 EMC 噪声能力
PRS16	扩频，平均值为 $F_{IMO}/4$，峰值为 $F_{IMO}/2$	3	高。敏感点是 PRS 序列重复周期和 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	中等。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，$F_{IMO}/4$ – $F_{IMO}/512$	1	中等。由于 PRS 重复周期较短，因此有更多敏感点。
VC2	固定， IMO/ （VC_1xVC_2）	0	在操作频率及其谐波下，器件对 EMC 信号敏感。建议仅在未计划认证 EMC/EMI 测试时使用。

直流和交流电气特性

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 = 3.3 V， SysClk = 24 MHz， CPU 时钟 = 6 MHz，基准线 >= 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） /（基准线计数）	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	测试条件和注释
活动电流		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 = 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

放置

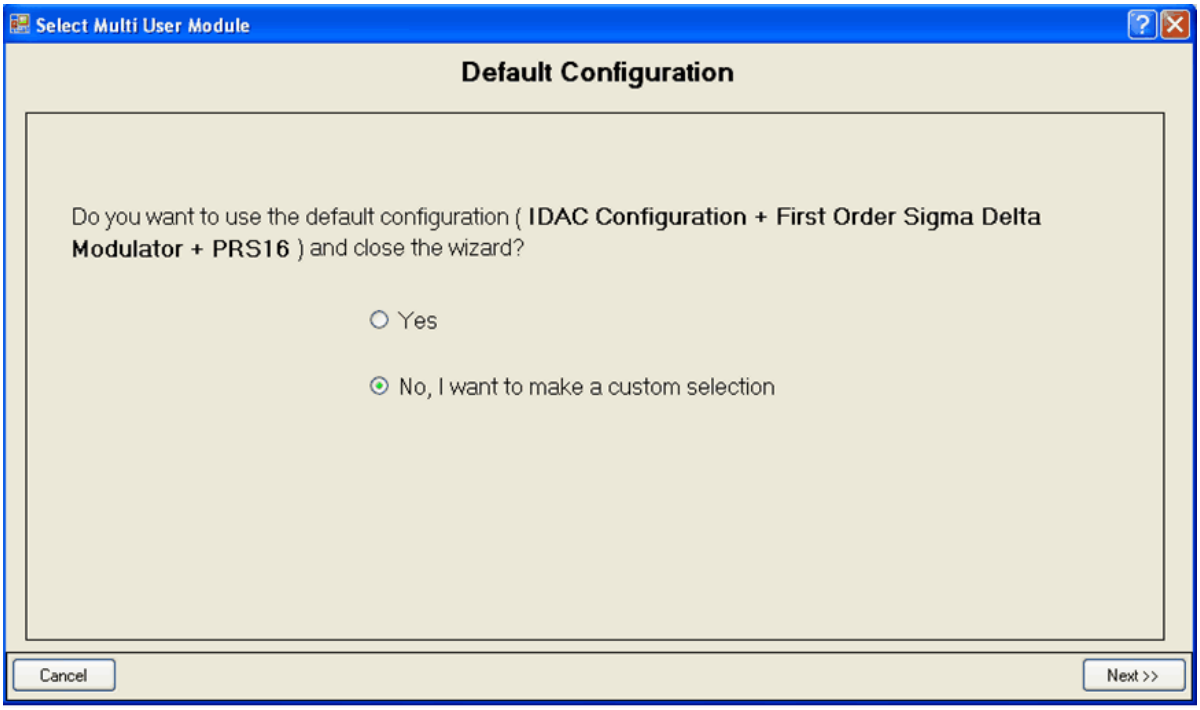
使用用户模块时，其各个模块将被自动放置。其它布置方式仅适用于单通道配置。同时使用 CSD2X 用户模块和占用同一个模拟复用总线的用户模块 （如 AMuxN）时，他们之间可能发生冲突。如果需要同步操作，则用户模块需要使用其它模拟复用总线的引脚。

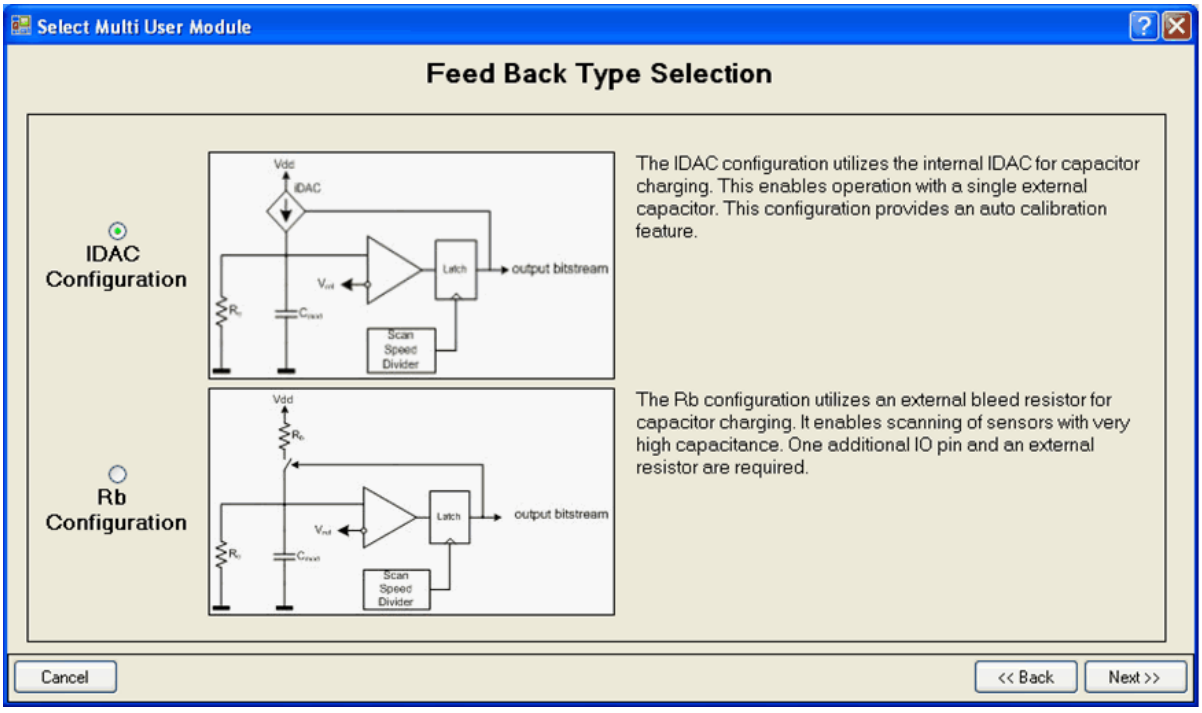
用户模块支持 16 个选项。

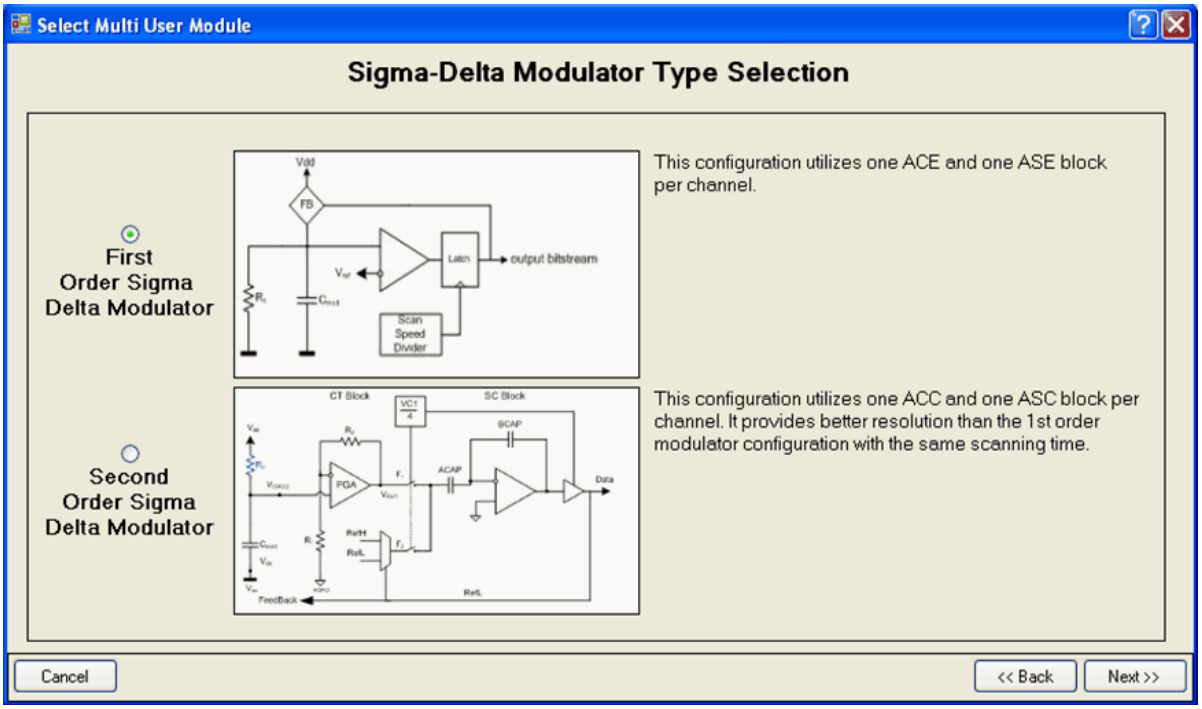
配置	缩略词
外部电阻 + 二阶调制器 + PWM8	RBCNT
外部电阻 + 二阶调制器 + PRS16	RBPRS
外部电阻 + 二阶调制器 + PRS8	RBPRS8
外部电阻 + 二阶调制器 + VC2	RBVC2
外部电阻 + 一阶调制器 + PWM8	RECNT
外部电阻 + 一阶调制器 + PRS16	REPRS
外部电阻 + 一阶调制器 + PRS8	REPRS8
外部电阻 + 一阶调制器 + VC2	REVC2
IDAC + 二阶调制器 + PWM8	IBCNT
IDAC + 二阶调制器 + PRS16	IBPRS
IDAC + 二阶调制器 + PRS8	IBPRS8
IDAC + 二阶调制器 + VC2	IBVC2
IDAC + 一阶调制器 + PWM8	IECNT
IDAC + 一阶调制器 + PRS16	IEPRS
IDAC + 一阶调制器 + PRS8	IEPRS8
IDAC + 一阶调制器 + VC2	IEVC2

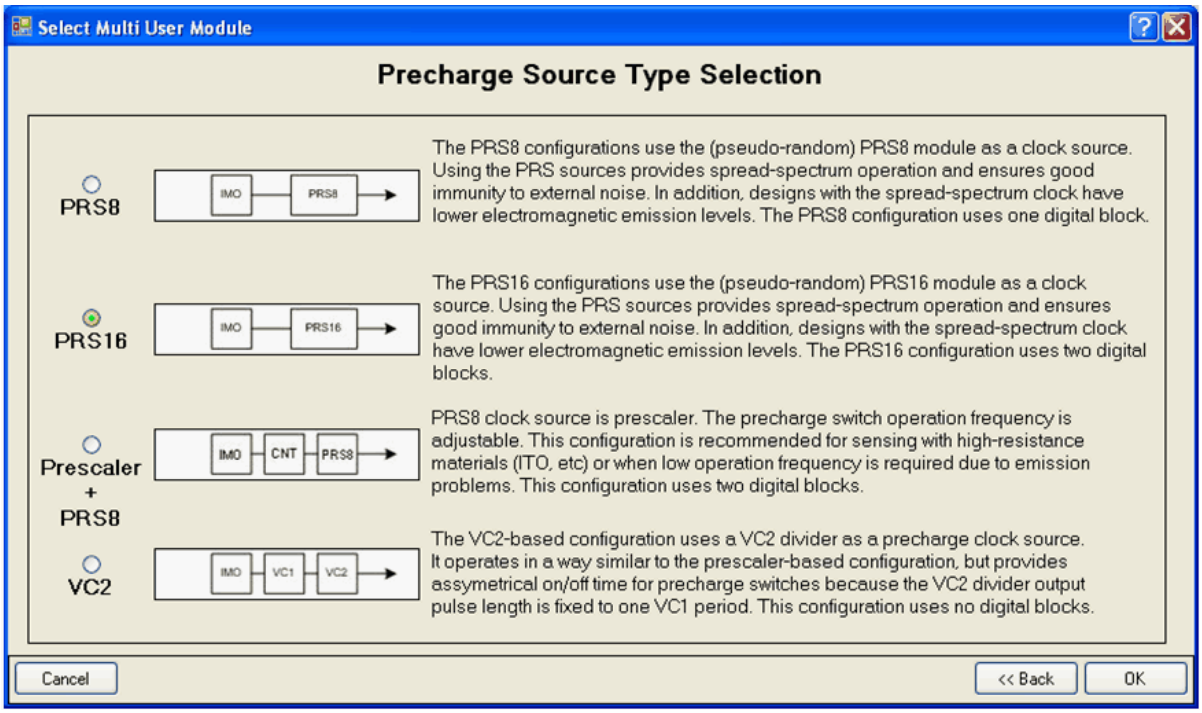
可通过 MUM 向导访问该向导。请按照向导说明进行选择。

图 8. CY8C28xxx 的 CSD2X MUM









当实例化用户模块时，会自动放置用户模块的块，其他放置仅适用于单通道的配置。所消耗的资源取决于 CSD2X 用户模块的配置情况。必须在启动 CSD2X 向导之前放置需要特定引脚资源的用户模块 （包括 LCD 和 I2CHW），以建立 CSD2X 用户模块的引脚连接。

在放置电容传感器的连接时，请勿使用 P1[0] 和 P1[1]。这些引脚用于对器件进行编程，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。

图 9. 可能存在的数字模块资源

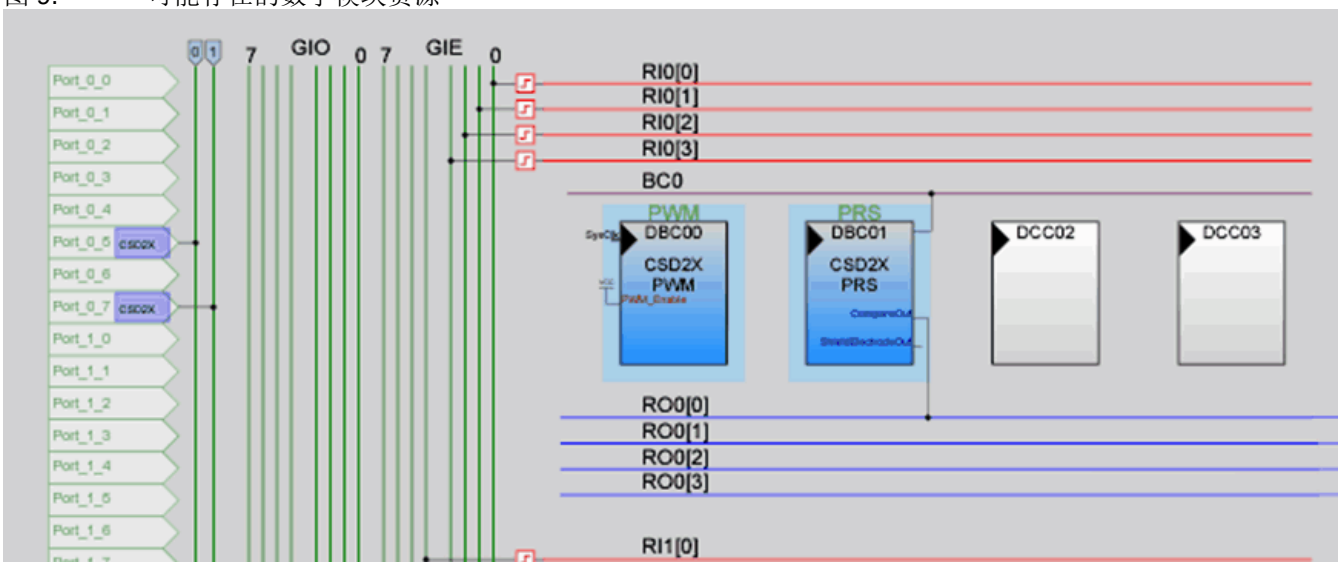


图 10. 可能存在的模拟模块资源 — 一阶调制器

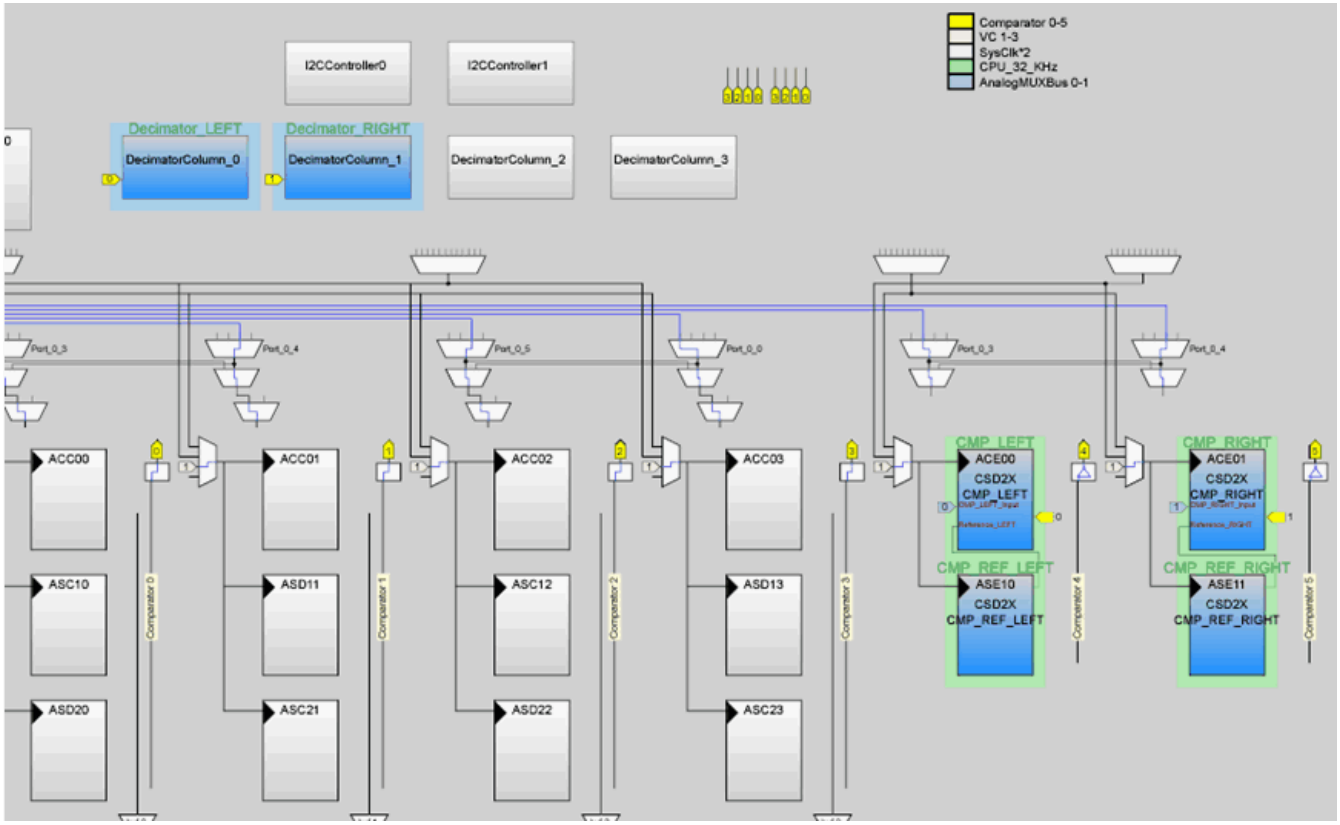
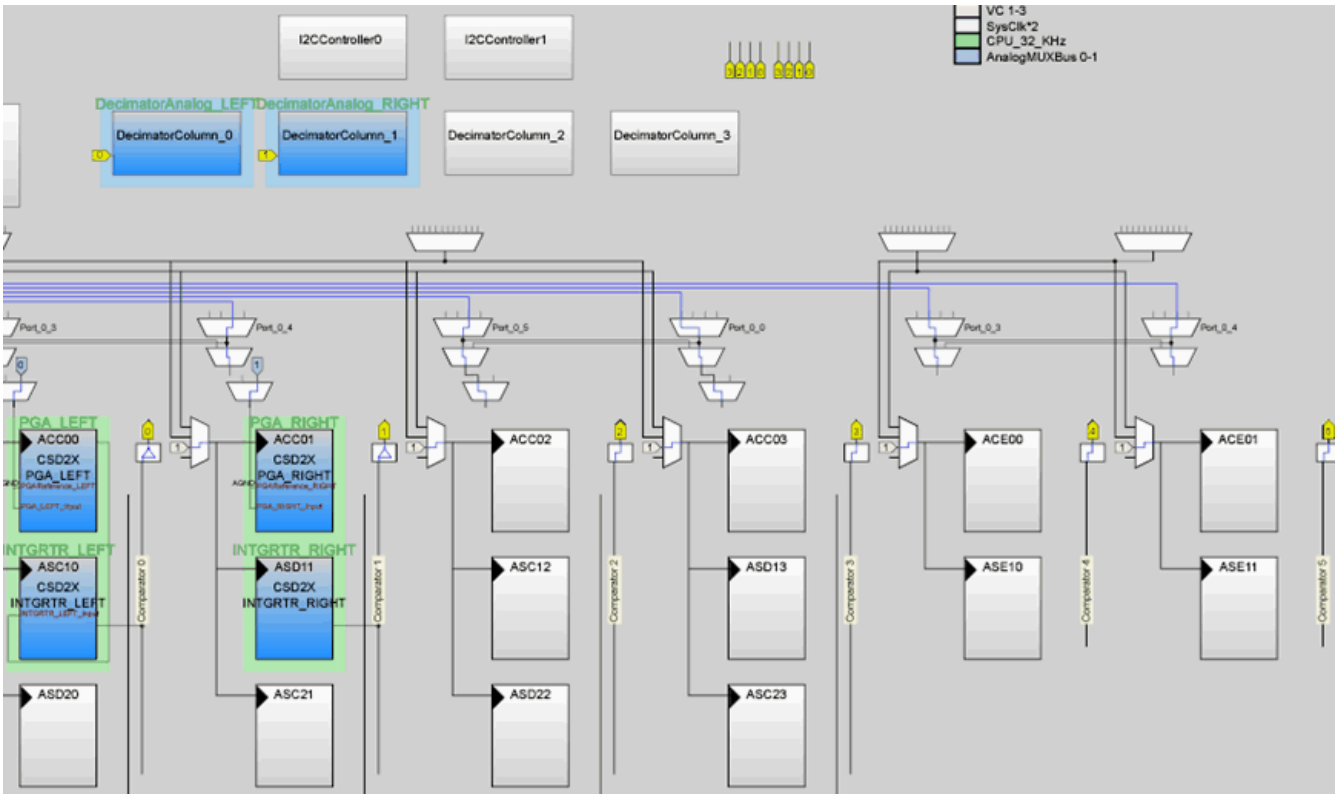


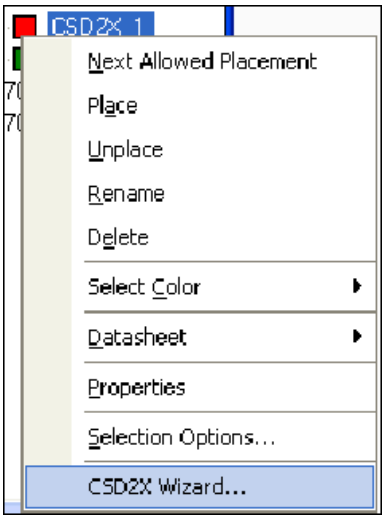
图 11. 可能的模拟模块资源 — 二阶调制器



向导

CSD2X 向导用于设置 CapSense 按键、滑条和接近感应传感器的引脚分布。可以选择所需的配置，并通过使用拖放界面来分配按键和滑条段。

1. 要访问该向导，请在工作区浏览器中右键单击用户模块，然后选择 CSD2X 向导。



2. 向导打开，其中显示了传感器数量、滑条数量和辐射状滑条传感器数量的数值输入框。向导中所显示的各个选项取决于所选配置。下面的屏幕截图显示的是不同配置的向导。

图 12. IDAC 配置的 CSD2X 向导

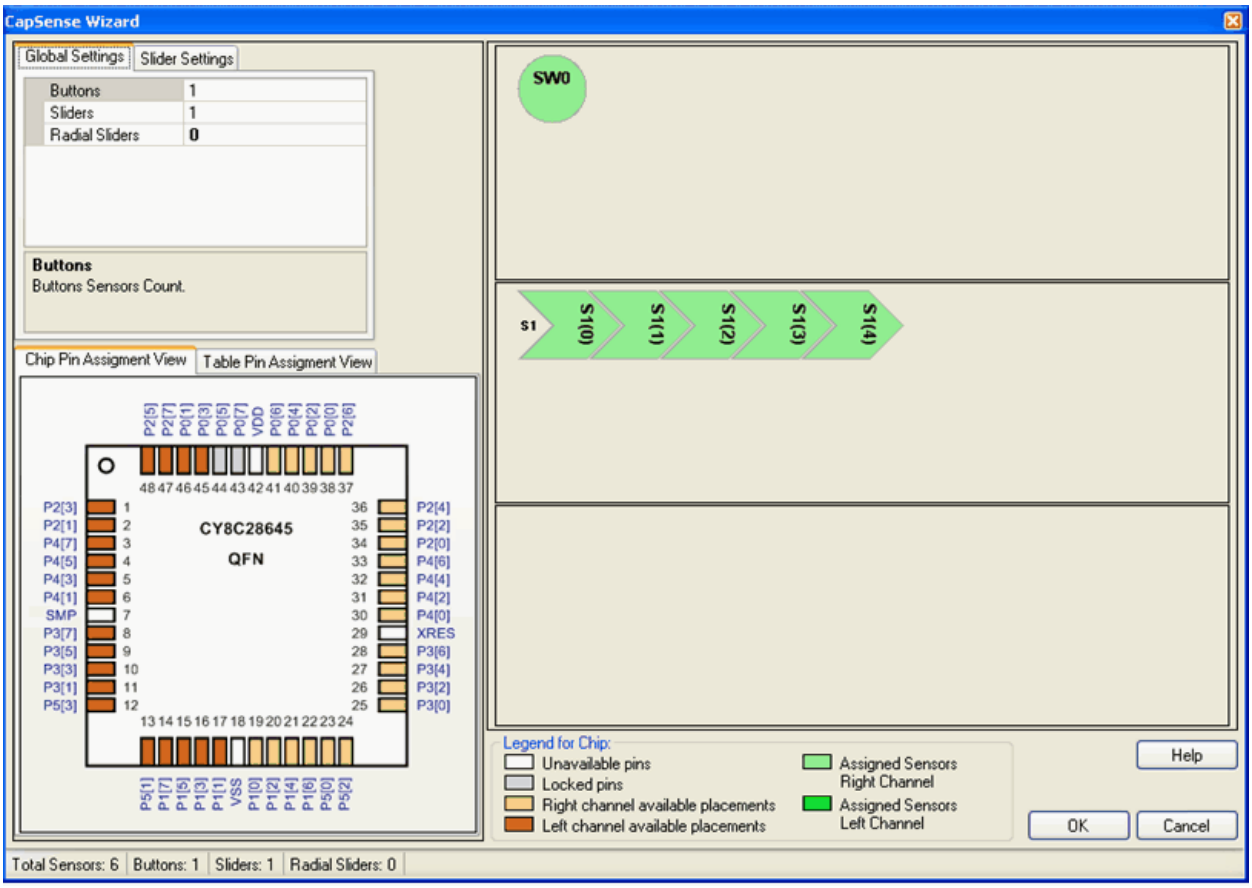
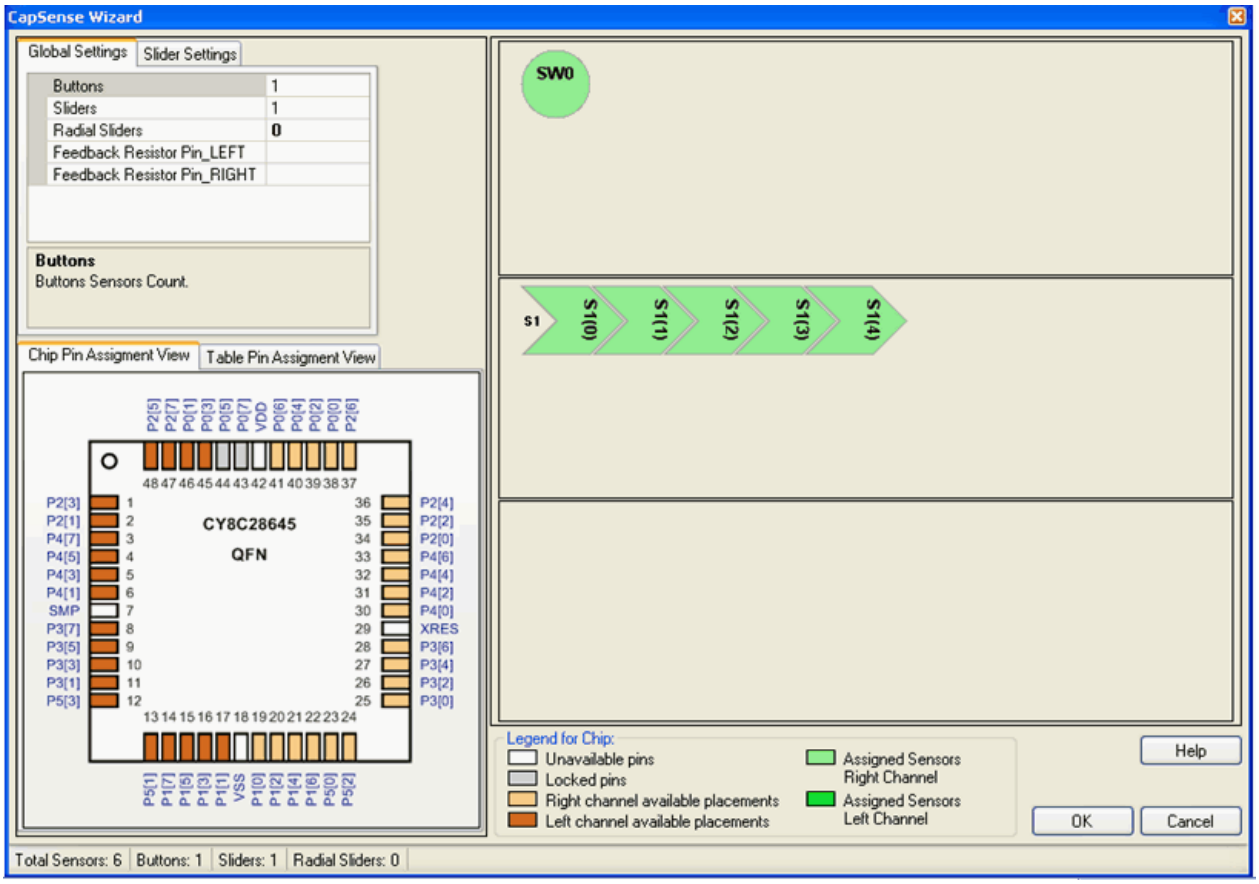


图 13. Rb 配置的 CSD2X 向导



向导引脚图标

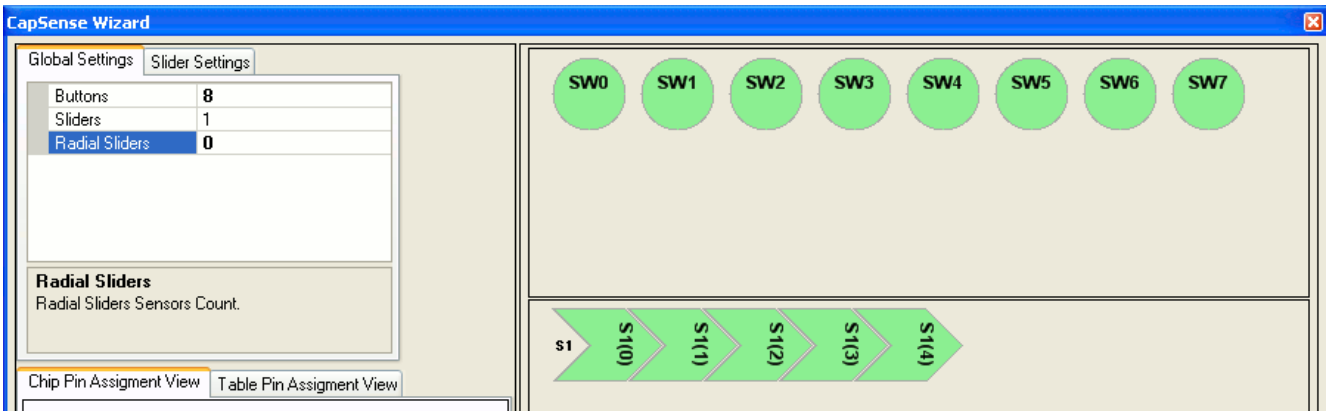
白色 — 引脚不能作为 CapSense 输入使用。

灰色 — 引脚被锁定。这种情况有两种可能的原因。一种可能的原因是另一个用户模块 （如 LCD 或 I^2C）已占用了该引脚。第二种可能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）。现在可以在向导中分配引脚了。

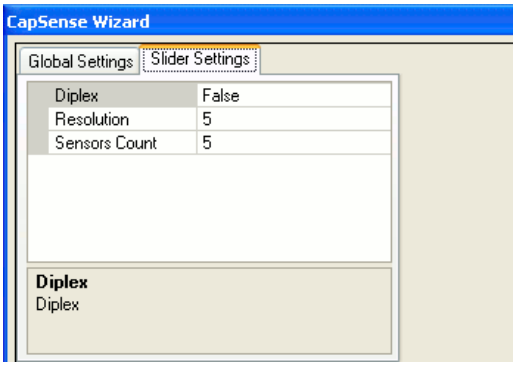
橙色 — 引脚可用于分配。

绿色 — 引脚已分配为 CapSense 输入。

3. 向导右侧区域包含三个区域，其中添加了开关、滑条和旋转滑条的表示。在相应的框中键入所需的开关、滑条和旋转滑条数量，然后按 [Enter]。显示内容随您选择的配置表示而更新。在 IDAC 配置的 CSD2X 向导中，从下拉菜单内选择反馈电阻引脚

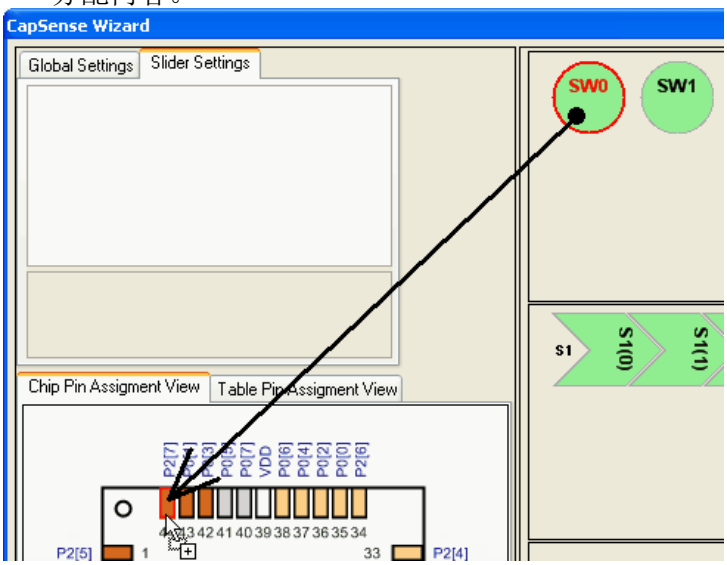


4. 切换到 “ 滑条设置 ” 选项卡，可设置滑条的参数。



5. 滑条分辨率即是滑条具有的位置数量。在滑条的多个段上给出读数的触摸值被内插到此分辨率中。如果将分辨率设置为 100，则手指在滑条上的位置报告为处于 0 到 99 之间的某个位置。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。
6. 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。
7. 要将开关映射到引脚，请左键单击开关，然后将其拖动到任意可用引脚。可以使用 “ 芯片引脚分配 ” 视图或 “ 表引脚分配 ” 视图来进行此操作。选择端口引脚后，引脚变为灰色，不再可用。通过将传感器拖离端口引脚，可更改传感器分配。如果您通过右键单击 “ 芯片引脚分配 ” 视图或 “ 表引脚分

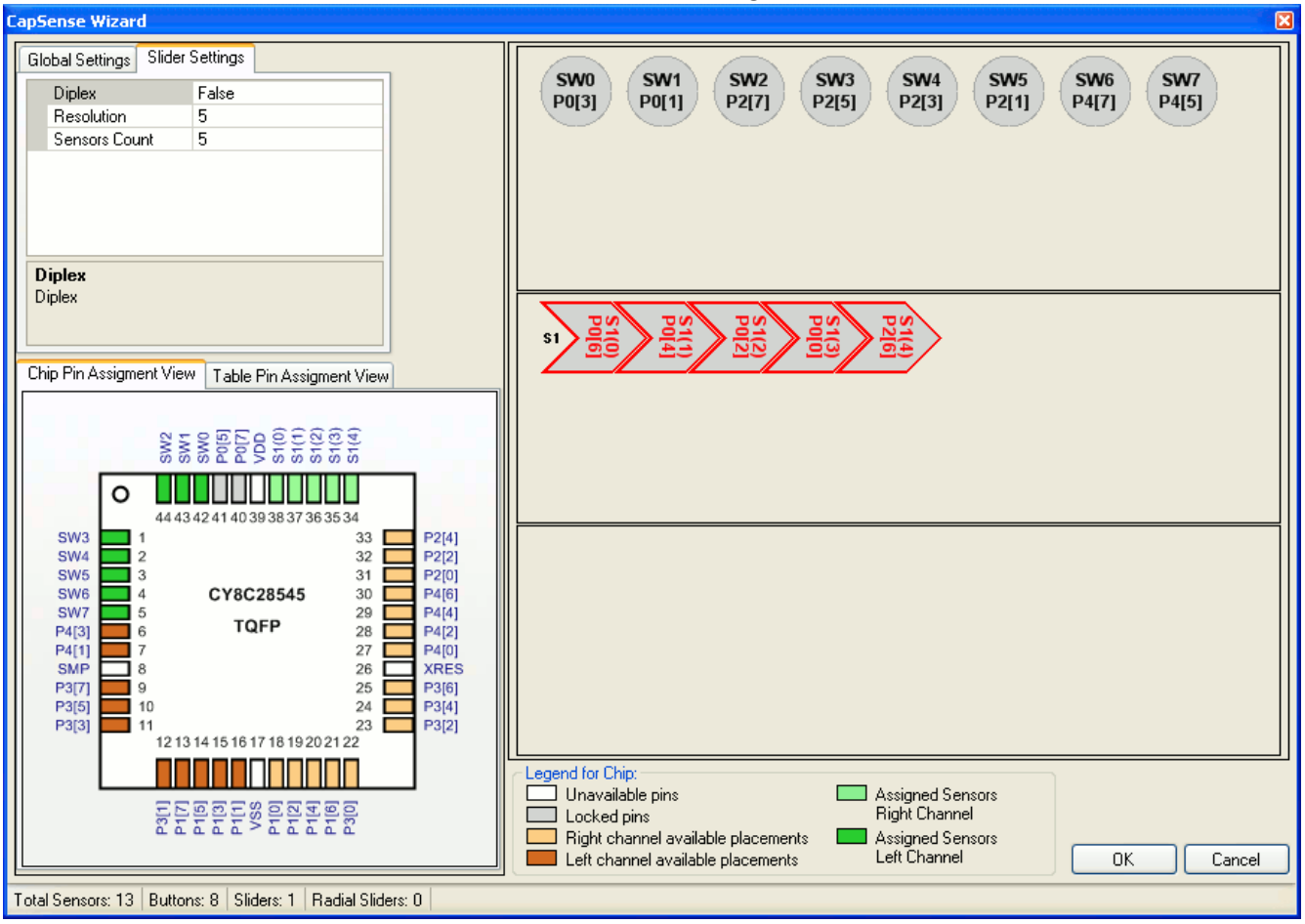
配 ” 视图，您将看到 “Clear All Pins” （移除所有引脚）选项。通过该选项，可以撤销所有引脚上的分配内容。



8. 对于其余独立传感器，重复操作即可。

9. 将滑条段映射到端口引脚与将传感器映射到端口引脚相同。

10. 单击 **OK** 按键以接受数据，然后返回到 PSoC Designer。



传感器放置现在已完成。设置用户模块参数，生成应用程序。如果需要，可以立即对示例项目进行调整。

如果要更改引脚分配，请将光标放在已分配的引脚上，单击引脚，然后将其拖放到引脚分配框外。该引脚分配被取消，然后可以进行重新分配。

向导滑条设置

双工

允许您使用一个引脚监控两个电子传感器以提高分辨率。有关更多信息，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 CSD2X_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 *CSD2X_Table.asm* 和 *CSD2X.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：

```
CSD2X_Sensor_Table_Right:
_CSD2X_Sensor_Table_Right:
      dw    0x0140  //  Port 1 Bit 6
      dw    0x0301  //  Port 3 Bit 0
      dw    0x0304  //  Port 3 Bit 2

CSD2X_Sensor_Table_Left:
_CSD2X_Sensor_Table_Left:
      dw    0x0308  //  Port 3 Bit 3
      dw    0x0302  //  Port 3 Bit 1
      dw    0x0108  //  Port 1 Bit 3
```

该表由 CSD2X_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，自由按键传感器再对应一个条目。第一个条目始终是自由传感器。每个条目为六个字节。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 表示已采用， 0 表示未采用）。第四、五、六个字节是固定点乘数，将其与所计算出的滑条中心相乘可以得出 CSD2X 向导中所需的分辨率。

```
CSD2X_Group_Table:
_CSD2X_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。该表由两个部分组成：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。下面是八传感器滑条的典型示例。

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


 CSD2X_Diplex_Table:
_CSD2X_Diplex_Table:
 db >DiplexTable_0, <DiplexTable_0
 db >DiplexTable_1, <DiplexTable_1
```

顺序表

顺序表中的每个传感器占用一个字节。该字节是与通道无关的原始计数结果阵列中的左传感器顺序。该表由向导生成，反映了传感器顺序。

```
CSD2X_1_Order_Table_Left:
_CSD2X_1_Order_Table_Left:
DB 0x01 // Position 1

CSD2X_1_Order_Table_Right:
_CSD2X_1_Order_Table_Right:
DB 0x00,0x02 // Position 0 and 2
```

参数与资源

PGAGain_RIGHT

默认值为 4.00。

PGAGain_LEFT

该参数仅适用于二阶调制器配置。为左右通道中的 ADC 设置 PGA 增益。可以独立设置两个 PGA 增益。如果使用 PSoC Designer 或 API 中所提供的 CSD2X_SetPGAGain 例程，则增益范围介于 1 到 48.00 之间。不支持小于 1 的增益设置。默认值为 4.00。

手指阈值

该阈值用于确定每个按键传感器的状态。如果有任何传感器处于活动状态， bIsAnySensorActive() 函数将返回 1。如果所有传感器均被关闭，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。取值范围为 5 到 255 ； 默认值为 40。

噪声阈值

对于单个传感器，超过该阈值的计数值不会更新基准线。对于滑条传感器，中心计算中不包括低于该阈值的计数值。取值范围为 5 到 255 ； 默认值为 20。

BaselineUpdate （基准线更新）阈值

当新的原始计数值高于当前基准线，并且差值低于噪声阈值 （“Sensors Autoreset” 参数设置为 “Disabled”）时，则当前基准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，基准线会按某个值递增，并清空该桶。此参数设置了基准线递增时桶必须达到的阈值。可能值为 0 到 255。

参数值越大，基准线的更新速度越慢。如果需要进行更加频繁的基准线更新，请减小该参数。默认值为 200。

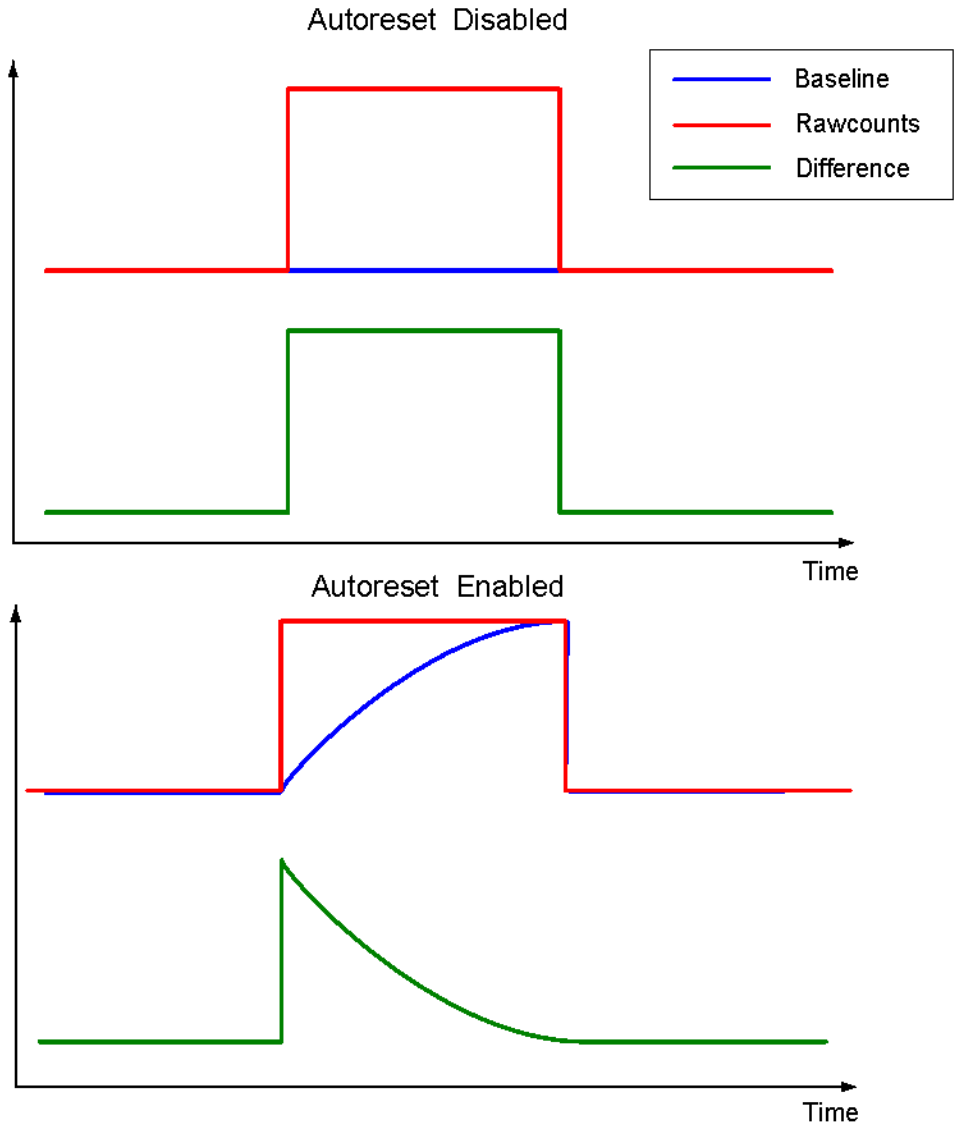
Sensors Autoreset（传感器自动复位）

该参数确定基准线是随时更新，还是仅当信号差值低于噪声阈值时才更新。当设置为 **Enabled** 时，基准线随时更新。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。原始计数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的。

如果将此参数设置为 **Disabled**（禁用），则仅当原始计数与基准线之间的差值低于噪声阈值参数时，才会更新基准线。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，否则应将该参数保持为“Disabled”（禁用）状态。默认设置为禁用状态。

下图说明了该参数对基准线更新的影响。

图 14. Sensor Autoreset （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之差。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。默认值为 10。

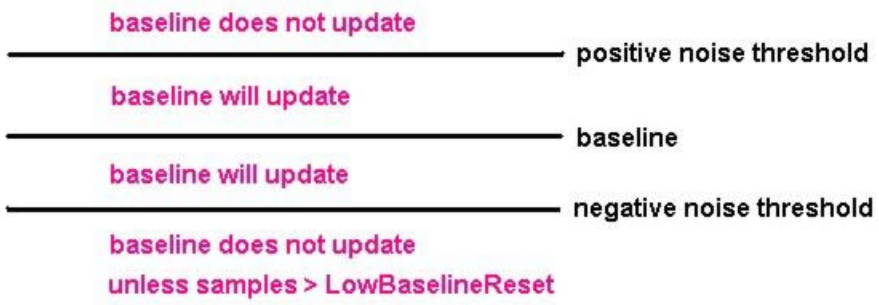
去抖动

“ 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。为了让传感器从不活动转换到活动状态，对于指定的样本数量，差计数值必须大于手指阈值与迟滞之和。防抖动计数器由 bIsSensorActive 或 bIsAnySensorActive API 函数递增。

可用的取值范围为 1 到 255 ；默认值为 3。如果将该值选择为 1，将不能实现去抖动。

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准线，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，则对基准线进行复位。默认值为 20。



LowBaselineReset （低基准线复位）

LowBaselineReset （低基准线复位）参数与 NegativeNoiseThreshold （负噪声阈值）参数配合使用。进行指定数量的样本后，如果采样计数值低于负数 NegativeNoiseThreshold 基准线，则基准线会被设置为新的原始计数值。该参数实际上计数了复位基准线所需的异常低的样本数值。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。

扫描速度

此参数影响传感器的扫描速度。选项包括 **Fast**、 **Normal** 和 **Slow**。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

分辨率 （一阶调制器）

该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计数值为 2^N-1。

增大分辨率可提高触摸检测的灵敏度和信噪比。默认值为 12。

表 5. 对于 24 MHz IMO 操作的情况，扫描时间 （以 µs 为单位）与扫描速度和分辨率的关系

分辨率 （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。

分辨率 （二阶调制器）

该参数用于确定扫描分辨率。各选项分别为 0 位、 12 位和 14 位。如果这些配置需要更高的分辨率，请使用过采样，并取几次采样结果的平均值。 N 位的扫描分辨率的最大原始计数值为 2^N-1。增大分辨率可提高触摸检测的灵敏度和信噪比。默认值为 12。

表 6. 对于 24 MHz IMO 操作的情况，扫描时间 （以 µs 为单位）与扫描速度和分辨率的关系

分辨率 （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。

IDAC 值（左）

默认值为 200。

IDAC 值（右）

该参数仅适用于 IDAC 配置。左右通道的电容测量范围取决于该参数。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。可以在运行时使用相应的 API 函数更改该参数。

取值范围为 1 到 255；默认值为 200。

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	结果
1X[a]	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

a. 不建议用于新设计。

在双通道配置中，将具有大电容的传感器连接到左通道。默认值为 x32。

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。

屏蔽电极输出

可以从备用数字行总线 （Row_0_Output_1 到 Row_0_Output_3) 中的一个选择屏蔽电极信号源。每行输出都可以连接到三个引脚的其中一个。将 ‘Row LUT Function’ （行 LUT 功能）设置为 ‘A’ 。默认值为 “None” （无）。

PRS 多项式

该参数将 PRS 多项式设置为基于 PRS 的配置。有以下两个选项：

- 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。

默认设置为 “ 短 ” 。

自动校准

使能或禁用自动校准 API 函数。

“ 自动校准 ” （Autocalibration）仅适用于 IDAC 配置。 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。默认设置为 “Enable” （使能）。

Reference_LEFT

默认值为 ASE10。

Reference_RIGHT

该参数仅适用于一阶调制器配置。该参数可为左右通道设置比较器参考值。参考值来自内部电阻电压分频器。默认值为 ASE11。

参考值

当比较器参考来自模拟调制器 （ASE11）或外部滤波 PWM/PRSPWM 信号 （来自带有 RC 滤波器的 AnalogColumn_InputSelect_1）时，该参数适用于一阶 Delta Sigma 调制器，且用于设置比较器参考值。当参考来自带隙 （VBG）或外部电压分频器 （来自带有电阻式电压分频器的 AnalogColumn_InputSelect_1）时，此值不起任何作用。

零是最小的参考值 （1/4 Vdd）。 8 是最大值 （3/4Vdd）。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。默认值为 4。

反馈电阻引脚 Pin_LEFT

反馈电阻引脚 Pin_RIGHT

该参数仅适用于 Rb 配置。该参数可为左右通道设置连接外部反馈电阻 （R_b）的引脚。可用的引脚选项分别为：P1[0]、P1[1]、P1[4]、P1[5]、P3[0]、P3[1]、P3[4]、P3[5]（左通道），以及 P1[1]、P1[2]、P1[5]、P1[6]、P3[1]、P3[2]、P3[5]、P3[6]（右通道）。对于一些器件封装或配置，某些引脚不可用。提示：如果将这些引脚中的一些使用于其他用途 （例如，分配用于传感器连接），那么不能在用户模块参数列表中选择它们。 CSD 用户模块的后来版本可能允许使用额外引脚来连接反馈电阻。这样可以在没有 P3 端口的封装中使用另一个 I^2C 端口。使用引脚 P1[5] 或 P3[1] 以避免编程问题。

BackgroundScanning （背景扫描）

通过该参数可以确定在项目中使能还是禁用背景扫描。根据选择，将生成与 API 相应的代码。

FMEA_Shorts_Test

使能或禁用 FMEA 短测试。

FMEA_Cmod_Rb_Test

使能计算 Cmod 和 Rb 的测试。

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。本节指定每个函数的接口，以及引用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 CSD2X_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，实例名称被缩写为 CSD2X。

**注意 **：在这种情况下，同所有用户模块的 API 一样， 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。尽管某些寄存器现在未被修改，但无法保证在将来的版本中是否仍不修改。

提供了进入点以初始化 CSD2X，启动其采样，并停止 CSD2X。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 CSD2X 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：

- CSD2X_waSnsBaseline[]
- CSD2X_waSnsResult[]
- CSD2X_waSnsDiff[]
- CSD2X_baSnsOnMask[]
- CSD2X_bScanComplete
- CSD2X_baSensorShortGnd[]
- CSD2X_baSensorShortVdd[]
- CSD2X_wCmod_Val
- CSD2X_wCmod_Rb_Val

CSD2X_waSnsBaseline[]：这是包含每个传感器的基准线数据的整数阵列。阵列大小等于传感器数量。通过下列函数更新 CSD2X_waSnsBaseline[] 阵列：

- CSD2X_UpdateAllBaselines();
- CSD2X_UpdateSensorBaseline();
- CSD2X_InitializeBaselines().

CSD2X_waSnsResult[]：这是包含每个传感器的原始信号的整数阵列。阵列大小等于传感器数量。通过下列函数更新 CSD2X_waSnsResult[] 数据：

- CSD2X_ScanSensor();
- CSD2X_ScanAllSensors().

CSD2X_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线数据之间的差值。阵列大小等于传感器数量。

CSD2X_baSnsOnMask[]：这是一个字节阵列，用于保持按键或滑条传感器的开 / 关状态。CSD2X_baSnsOnMask[0] 包含传感器 0 到 7 的掩码位 （传感器 0 是位 0，传感器 1 是位 1）。CSD2X_baSnsOnMask[1] 包含传感器 8 到 15 的掩码位 （如果需要），依次类推。此字节阵列包含的元素数足以包含所有被放置的传感器。按键开启时位值为 1，关闭时位值为 0。 CSD2X_baSnsOnMask[] 数据可通过 CSD2X_blsSensorActive(BYTE bSensor) 函数或 CSD2X_bIsAnySensorActive() 子程序更新。

CSD2X_bScanComplete[] ：只有背景扫描性能被使能时，该变量才有效。完成传感器扫描时，将设置该变量，并在 CSD2X 中断中更新它。

掩码位	数值	说明
CSD2X_SCAN_COMPLETE	0X01	扫描所有传感器完成标志
CSD2X_SCAN_1COMPLETE	0X02	扫描一个传感器完成标志
CSD2X_SCAN_ALLSENSORS	0X04	扫描所有传感器标志

CSD2X_baSensorShortGnd[] ：只有 FMEA 性能被使能时，该字节阵列才有效。CSD2X_baSensorShortGND[0] 包含传感器 0 到 7 的掩码位 （传感器 0 是位 0，传感器 1 是位 1）。CSD2X_baSensorShortGND[1] 包含传感器 8 到 15 的掩码位 （如果需要），依次类推。该字节阵列所包含的元素足以包含所有被放置的传感器。 CSD2X_baSensorShortGnd[] 数据通过 CSD2X_bFMEA_CheckGndShort() 函数更新。

CSD2X_baSensorShortVdd[]：仅在使能 FMEA 性能时，该字节阵列才有效。CSD2X_baSensorShortVdd[0] 包含传感器 0 到 7 的掩码位（传感器 0 是位 0，传感器 1 是位 1）。CSD2X_baSensorShortVdd[1] 包含传感器 8 到 15 的掩码位（如果需要），依次类推。该字节阵列包含的元素数足以包含所有被放置的传感器。 CSD2X_baSensorShortGnd[] 数据通过 CSD2X_bFMEA_CheckVddShort() 函数更新。

CSD2X_wCmod_Val[]：仅在 FMEA 性能被使能，并且选定 IDAC 方法时，该字变量才有效。该变量包含 CSD2X_FMEA_Cmod_Check 函数期间中所计算的 Cmod 值。

CSD2X_wCmod_Rb_Val：仅在使能 FMEA 性能并选定 Rb 方法时，该字变量才有效。该变量包含 CSD2X_FMEA_Cmod_Rb_Check 函数期间中所计算的 Cmod 与 Rb 的乘积。

以下变量组可用于参数存储：

- bNoiseThreshold — 存储 NoiseThreshold （噪声阈值）参数值；
- bNegativeNoiseThreshold — 存储 NegativeNoiseThreshold （负噪声阈值）参数值；
- bBaselineUpdateThreshold — 存储 BaselineUpdateThreshold （基准线更新阈值）参数值；
- bHysteresis — 存储 Hysteresis （迟滞）参数值；
- bDebounce — 存储 Debounce （去抖动）参数值；
- bLowBaselineReset — 存储 LowBaselineReset （低基准线复位）参数值；
- baBtnFThreshold — 为每个传感器存储默认手指阈值的值。可以通过 CSD2X_SetDefaultFinger-Thresholds() 函数初始化该变量。

API	IBCNT	IBPRS	IBPRS8	IBVC2	IECNT	IEPRS	IEPRS8	IEVC2	RBCNT	RBPRS	RBPRS8	RBvC2	RECNT	REPRS	REPRS8	REVC2
CSD2X_Start()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_Stop()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_ScanSensor()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_ScanAllSensors()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_ClearSensors()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_wReadSensor()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_wGetPortPinLeft()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_wGetPortPinRight()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_EnableSensor()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_DisableSensor()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_SetScanMode()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_SetPGAGainLeft()	•	•	•	•					•	•	•	•				
CSD2X_SetPGAGainRight()	•	•	•	•					•	•	•	•				

API	IBCNT	IBPRS	IBPRS8	IBVC2	IECNT	IEPRS	IEPRS8	IEVC2	RBCNT	RBPRS	RBPRS8	RBvC2	RECNT	REPRS	REPRS8	REVC2
CSD2X_SetPGARefLeft()	•	•	•	•					•	•	•	•				
CSD2X_SetPGARefRight()	•	•	•	•					•	•	•	•				
CSD2X_SetRefValue()					•	•	•						•	•	•	
CSD2X_Calibrate()	•	•	•	•	•	•	•	•								
CSD2X_UpdateSensorBaseline()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_bIsSensorActive()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_bIsAnySensorActive()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_SetDefaultFingerThresholds()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_InitializeSensorBaseline()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_InitializeBaselines()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_UpdateAllBaselines()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_wGetCentroidPos()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_wGetRadialPos()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_wGetRadialInc()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_baSensorShortGnd()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_baSensorShortVdd()	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CSD2X_bFMEA_Left_Cmod_Check()	•	•	•	•	•	•	•	•								
CSD2X_bFMEA_Right_Cmod_Check()	•	•	•	•	•	•	•	•								
CSD2X_bFMEA_Left_Cmod_Rb_Check()									•	•	•	•	•	•	•	•
CSD2X_bFMEA_Right_Cmod_Rb_Check()									•	•	•	•	•	•	•	•

CSD2X_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  CSD2X_Start()
```

汇编：

```
lcall  CSD2X_Start
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_Stop

说明：

停止传感器扫描，禁用内部中断，并调用 CSD2X_ClearSensors() 以将所有传感器复位为非活动状态。

C 原型：

```
void  CSD2X_Stop()
```

汇编：

```
lcall  CSD2X_Stop
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_Resume

说明：

调用 CSD2X_Stop 之后恢复用户模块操作。

C 原型：

```
void  CSD2X_Resume()
```

汇编：

```
lcall  CSD2X_Resume
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_SetPGAGainLeft

说明：

为左侧 PGA 模块设置增益，将会覆盖在器件编辑器中所设置的值。

C 原型：

```
void  CSD2X_SetPGAGainLeft(byte bGainSetting)
```

汇编：

```
mov   A, bGainSetting
lcall  CSD2X_SetPGAGainLeft
```

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
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：

**

CSD2X_SetPGAGainRight

说明：

为右侧 PGA 模块设置增益，将会覆盖在器件编辑器中所设置的值。

C 原型：

```
void  CSD2X_SetPGAGainRight(byte bGainSetting)
```

汇编：

```
mov   A, bGainSetting
lcall  CSD2X_SetPGAGainRight
```

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：

**

CSD2X_ScanSensor

说明：

扫描选定的传感器对。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。

C 原型：

```
void CSD2X_ScanSensor(BYTE bSensorLeft, byte bSensorRight);
```

汇编：

```
mov A,  bSensorLeft
mov X,  bSensorRight
lcall CSD2X_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

**

CSD2X_ScanAllSensors

说明：

通过调用每个传感器索引的 CSD2X_ScanSensor()，扫描所有已配置的传感器。

C 原型：

```
void CSD2X_ScanAllSensors();
```

汇编：

```
lcall CSD2X_ScanAllSensors
```

参数：

无

返回值：

无

其他影响

**

CSD2X_SetScanMode

说明：

根据的所需扫描速度和分辨率设置时钟。

C 原型：

```
void CSD2X_SetScanMode(BYTE bSpeed, BYTE bResolution);
```

汇编：

```
mov X, bResolution
mov A, bSpeed
lcall CSD2X_SetScanMode
```

参数：

bSpeed：扫描速度

下面给出了 bSpeed 参数的常量：

常量	数值
CSD2X_FAST_SPEED	0x01
CSD2X_NORMAL_SPEED	0x02
CSD2X_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
CSD2X_8_BIT_RESOLUTION	8
CSD2X_10_BIT_RESOLUTION	10
CSD2X_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
CSD2X_10_BIT_RESOLUTION	10
CSD2X_12_BIT_RESOLUTION	12
CSD2X_14_BIT_RESOLUTION	14

返回值：

无

其他影响

**

CSD2X_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准线。此基准线使用 “ 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 CSD2X_UpdateSensorBaseline() 时，通过从原始计数值中减去以前的基准线来计算差值计数。此差值存储在 CSD2X_waSnsDiff[] 阵列中，用户可以使用该值。

2. 如果禁用了传感器自动复位 （Sensors Autoreset），则每次调用 CSD2X_UpdateSensorBaseline() 时，差值会与噪声阈值进行比较。如果差值低于噪声阈值，将被累加到虚拟的桶中。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. 虚拟水桶中的累计差值达到 “ 基准线更新阈值 ” （BaselineUpdateThreshold）后，基准线按 1 递增，且桶形裕量复位为 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 CSD2X_UpdateSensorBaseline(BYTE bSensor)
```

汇编：

```
mov   A,   bSensor
lcall  CSD2X_UpdateSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

其他影响：

**

CSD2X_UpdateAllBaselines

说明：

使用 CSD2X_bUpdateSensorBaseline() 函数更新所有传感器的基准线。

C 原型：

```
void  CSD2X_UpdateAllBaselines()
```

汇编：

```
lcall CSD2X_UpdateAllBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_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。该函数还可更新 CSD2X_baSnsOnMask[] 阵列中传感器的位。

C 原型：

```
BYTE  CSD2X_bIsSensorActive(BYTE bSensor)
```

汇编：

```
mov  A,  bSensor
lcall CSD2X_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：

**

CSD2X_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 CSD2X_bIsSensorActive()，以便在调用此函数后更新 CSD2X_baSnsOnMask[] 阵列。

C 原型：

```
BYTE  CSD2X_bIsAnySensorActive()
```

汇编：

```
lcall CSD2X_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：表示没有任何传感器处于活动状态。

其他影响：

**

CSD2X_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 CSD2X 向导中指定的分辨率计算中心位置。仅在滑条是由 CSD2X 向导定义时，该函数才可用。

C 原型：

```
WORD  CSD2X_wGetCentroidPos(BYTE bSnsGroup)
```

汇编：

```
mov  A,  bSnsGroup
lcall CSD2X_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 是按键传感器组。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，则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，该函数会将数值从零恢复为 CSD2X 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）。如果在执行中心 / 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 CSD2X_blsSensorActive() 子程序确定触摸了哪些滑条段。

注意：如果滑条段的噪声计数值大于噪声阈值，则该子例程可能生成假的中心结果。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中心。

CSD2X_wGetRadialPos

说明：

检查中心的差值阵列。如果存在，则根据 CSD2X 向导中指定的分辨率计算该中心位置。此函数仅适用与 CSD2X 向导定义的辐射滑条。

C 原型：

```
WORD  CSD2X_wGetRadialPos(BYTE bSnsGroup)
```

汇编：

```
mov  A, bSnsGroup
lcall CSD2X_wGetRadialPos
```

参数：

bSnsGroup A => 滑条编号

该参数是正在使用的辐射滑条的编号。可以通过 CSD2X 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，以避免负差值和基准线更新。如果应用监控差值信号，则在差值计数数据传输后调用该子程序。

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 CSD2X 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）。

注意：如果滑条段的噪声计数值大于噪声阈值，则该子例程可能生成假的中心结果。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中心。

CSD2X_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。该函数与 CSD2X_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  CSD2X_wGetRadialInc(BYTE bSnsGroup)
```

汇编：

```
mov  A, bSnsGroup
lcall CSD2X_wGetRadialInc
```

参数：

bSnsGroup A => 滑条编号

该参数是正在使用的辐射滑条的编号。可以通过 CSD2X 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 CSD2X_wGetRadialPos() 返回 -1 (FFFFh)）或者当前没有任何触摸 （此时 CSD2X_wGetRadialPos() 返回 -1 (FFFFh)）

其他影响：

仅在调用 CSD2X_wGetRadialPos() API 之后，才能调用该子程序。因为它使用由 CSD2X_wGetRadialPos() 设置的内部数据 CSD2X_waSliderPrevPos 和 CSD2X_waSliderCurrPos。

CSD2X_InitializeSensorBaseline

说明：

将扫描所选的传感器，将初始值加载到 CSD2X_waSnsBaseline[bSensor] 阵列元素中。原始计数值被复制到所选传感器的基准线阵列元素中。该函数可用于复位单个传感器的基准线。

C 原型：

```
void CSD2X_InitializeSensorBaseline(BYTE bSensor)
```

汇编：

```
mov A, bSensor
lcall CSD2X_InitializeSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

其他影响：

**

CSD2X_InitializeBaselines

说明：

通过扫描每个传感器，将其初始值加载到 CSD2X_waSnsBaseline[] 阵列中。原始计数值被复制到每个传感器的基准线阵列中。

C 原型：

```
void CSD2X_InitializeBaselines()
```

汇编：

```
lcall CSD2X_InitializeBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_SetDefaultFingerThresholds

说明：

将手指阈值的参数值加载到 CSD2X_baBtnFThreshold[] 阵列内。如果没有将自定义值手动加载到 CSD2X_baBtnFThreshold[] 阵列，则必须在扫描之前调用该函数。

C 原型：

```
void CSD2X_SetDefaultFingerThresholds()
```

汇编：

```
lcall CSD2X_SetDefaultFingerThresholds
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_SetPGARefLeft

说明：

该函数将会覆盖左 PGA 参考值的用户模块参数设置。如果需要用不同的参考设置扫描某些传感器，则使用该函数。该函数可与 CSD2X_ScanSensor() 函数结合使用。该函数适用于 Rb 配置。

C 原型：

```
void CSD2X_SetPGARefLeft(BYTE bRefValue);
```

汇编：

```
mov     A, bRefValue
lcall    CSD2X_SetPGARefLeft
```

参数：

bRefValue — 设置参考值。接受的值为 1 （AGND）或 2 （VSS）。

返回值：

无

其他影响：

**

CSD2X_SetPGARefRight

说明：

该函数将会覆盖 Rb 配置中右 PGA 参考值的用户模块参数设置。如果需要用不同的参考设置扫描某些传感器，则使用该函数。该函数可与 CSD2X_ScanSensor() 函数结合使用。该函数适用于 Rb 配置。

C 原型：

```
void CSD2X_SetPGARefRight(BYTE bRefValue);
```

汇编：

```
mov     A, bRefValue
lcall    CSD2X_SetPGARefRight
```

参数：

bRefValue — 设置参考值。接受的值为 1 （AGND）或 2 （VSS）。

返回值：

无

其他影响：

**

CSD2X_Calibrate

说明：

该函数将会覆盖 DAC 值的用户模块参数设置，以移动一半范围内的原始计数。需要在基线初始化前调用该函数。该函数不适用于 Rb 配置。

C 原型：

```
void CSD2X_Calibrate(void);
```

汇编：

```
lcall    CSD2X_Calibrate
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_ClearSensors

说明：

通过为每个传感器依次调用 CSD2X_wGetPortPinLeft()/CSD2X_wGetPortPinRight() 和 CSD2X_DisableSensor()，将所有的传感器清除为非采样状态。

C 原型：

```
void  CSD2X_ClearSensors()
```

汇编：

```
lcall  CSD2X_ClearSensors
```

参数：

无

返回值：

无

其他影响：

**

CSD2X_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  CSD2X_wReadSensor(BYTE bSensor)
```

汇编：

```
mov A, bSensor
lcall CSD2X_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：

**

CSD2X_wGetPortPinLeft

说明：

返回指定传感器的端口号和引脚掩码。传递的参数对 CSD2X_Sensor_Table_Left[] 中的数据编制索引并进行选择。返回值可以传递给 CSD2X_EnableSensor() 和 CSD2X_DisableSensor()。该函数仅适用于单通道配置。

C 原型：

```
WORD  CSD2X_wGetPortPinLeft(BYTE bSensor)
```

汇编：

```
mov  A,  bSensor
lcall     CSD2X_wGetPortPinLeft
```

参数：

bSensor — 传感器编号，范围是 0 到 （n - 1），其中 “n” 是左通道 CSD2X 向导中设置的传感器总数（包括连接至左通道的滑条段数量）。CSD2X_wGetPortPinLeft() 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：

**

CSD2X_wGetPortPinRight

说明：

返回连接到右通道的给定传感器的端口号和引脚掩码。传递的参数对 CSD2X_Sensor_Table_Right[] 中的数据编制索引并进行选择。返回值可以传递给 CSD2X_EnableSensor() 和 CSD2X_DisableSensor()。该函数仅适用于双通道配置。

C 原型：

```
WORD  CSD2X_wGetPortPinRight(BYTE bSensor)
```

汇编：

```
mov  A,  bSensor
lcall     CSD2X_wGetPortPinRight
```

参数：

bSensor — 传感器编号，范围是 0 到（n - 1），其中 “n” 是右通道 CSD2X 向导中设置的传感器总数（包括连接至右通道的滑条段数量）。 CSD2X_wGetPortPinRight() 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：

**

CSD2X_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。可以使用 CSD2X_wGetPortPinLeft() 或 CSD2X_wGetPortPinRight() 函数选择端口和传感器，端口编号和传感器位掩码分别加载到 X 和 A 中。修改驱动模式，以便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  CSD2X_EnableSensor(BYTE bMask, BYTE bPort)
```

汇编：

```
mov  X,  bPort
mov  A,  bMask
lcall  CSD2X_EnableSensor
```

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

CSD2X_DisableSensor

说明：

禁用 CSD2X_wGetPortPinLeft() 或 CSD2X_wGetPortPinRight() 函数所选的传感器。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。

C 原型：

```
void  CSD2X_DisableSensor(BYTE bMask, BYTE bPort)
```

汇编：

```
mov  X,  bPort
mov  A,  bMask
lcall  CSD2X_DisableSensor
```

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

CSD2X_SetRefValue

说明：

通过调整 PRS 或 PWM 的比较值来设置参考电压。

C 原型：

```
void CSD2X_SetRefValue(BYTE bRefValue);
```

汇编：

```
mov A, bRefValue
lcall CSD2X_SetRefValue
```

参数：

bRefValue — 参考电压代码，其范围从 0 到 8 （0 -> Ref = 1/4 * Vcc， 8 -> Ref = 3/4 * Vcc）

返回值：

无

其他影响

**

用于背景扫描和 FMEA 的 API：

BYTE CSD2X_bIsScanComplete(void)

说明

检查 bScanComplete 变量中的扫描完成标志，然后返回 TRUE 或 FASLE 值给用户。该 API 在调用后还可复位该标志。

C 原型：

```
BYTE CSD2X_bIsScanComplete(void);
```

汇编

```
lcall  CSD2X_bIsScanComplete
```

参数

无

返回

该参数将返回 1（表示活动状态）或 0（表示非活动状态）。如果 A => 1，则表示扫描完成；如果 A = 0，表示扫描未完成。

BYTE CSD2X_bFMEA_CheckGndShort(void)

说明

该 API 用于检查是否有任何传感器（若使能屏蔽电极，它也被包括在内）短路接地。

C 原型：

```
BYTE CSD2X_bFMEA_CheckGndShort(void);
```

汇编

```
lcall  CSD2X_bFMEA_CheckGndShort
```

参数

无

返回

如果有任何传感器短接 GND，则该参数将返回 1。如果没有任何传感器短接 GND，将返回 0。

BYTE CSD2X_bFMEA_CheckVddShort(void)

说明

该 API 用于检查是否存在传感器（若使能屏蔽电极，它也被包括在内）短接 Vdd。

C 原型：

```
BYTE CSD2X_bFMEA_CheckVddShort(void);
```

汇编

```
lcall  CSD2X_bFMEA_CheckVddShort
```

参数

无

返回

如果有任何传感器被短接 Vdd，该参数将返回 1。如果没有任何传感器短接 Vdd，将返回 0。

BYTE CSD2X_bFMEA_CheckSensorShort(void)

说明

该 API 用于检查传感器之间 （若使能屏蔽电极，它也被包括在内）是否存在短接情况。

C 原型：

```
BYTE CSD2X_bFMEA_CheckSensorShort(void);
```

汇编

```
lcall  CSD2X_bFMEA_CheckSensorShort
```

参数

无

返回

如果传感器之间发生短接，该参数将返回 1。如果传感器之间未发生短接，则返回 0。

如果存在传感器短接 V_{DD}，则 CSD2X_bFMEA_CheckSensorShort API 将不会准确检查传感器之间的短接情况。使用 CSD2X_bFMEA_CheckVddShort() 和 CSD2X_bFMEA_CheckGndShort() API 来检查 Gnd 和 Vdd 的短接情况。

BYTE CSD2X_bFMEA_Left_Cmod_Check(void)

BYTE CSD2X_bFMEA_Right_Cmod_Check(void)

说明

该 API 仅适用于 IDAC 配置。该 API 将检查 C_{mod} 的短接情况，另外它还会检查是否处于有效范围内。所推荐的最小和最大的 C_{mod} 值被定义为它的有效范围，其容差为 20%。最小的 C_{mod} 值为 1 nF，最大的 C_{mod} 值为 20 nF。 Cmod 范围的最大和最小值被定义在 CSD2X.inc 文件中，用户可以在 PSoC Designer 的生成 / 编译期间修改这些值。

分别使用 CSD2X_bFMEA_Left_Cmod_Check() 和 CSD2X_bFMEA_Right_Cmod_Check() 函数更新 CSD2X_wCmod_L_Val 和 CSD2X_wCmod_R_Val WORD 变量。这些变量包含被放大 100 倍的 C_{mod} 值 （其单位为 nF）。例如， CSD2X_wCmod_Val = 2200，表示的是 22 nF （或 0.022 uF）。仅在 CSD2X_bFMEA_Left_Cmod_Check() 或 CSD2X_bFMEA_Left_Cmod_Check() API 返回 bRetVal.4 时，该值才有效。

C 原型：

```
BYTE CSD2X_bFMEA_Left_Cmod_Check(void);
BYTE CSD2X_bFMEA_Right_Cmod_Check(void)
```

汇编

```
lcall  CSD2X_bFMEA_Left_Cmod_Check
lcall  CSD2X_bFMEA_Right_Cmod_Check
```

参数

无

返回

bRetVal.0：C_{mod} 与 GND 短接
bRetVal.1：C_{mod} 与 Vdd 短接
bRetVal.4：Cmod 的偏差不超过 +20%
bRetVal.8：Cmod 低于有效范围或 Cmod 被断开连接
bRetVal.16：C_{mod} 高于有效范围

常量	数值	说明
CSD2X_LEFT_CMOD_SHORTED_TO_GND	0	左通道的 Cmod 与 GND 短接
CSD2X_ LEFT_CMOD_SHORTED_TO_VDD	1	左通道的 Cmod 与 Vdd 短接
CSD2X_ LEFT_CMOD_WITHIN_20	4	左通道的 Cmod 的偏差不超过 +20%
CSD2X_ LEFT_CMOD_IS_LOW	8	左通道的 Cmod 超出范围
CSD2X_ LEFT_CMOD_IS_HIGH	16	左通道的 Cmod 超出范围
CSD2X_RIGHT_CMOD_SHORTED_TO_GND	0	右通道的 Cmod 与 GND 短接
CSD2X_RIGHT_CMOD_SHORTED_TO_VDD	1	右通道的 Cmod 与 Vdd 短接
CSD2X_RIGHT_CMOD_WITHIN_20	4	右通道的 Cmod 的偏差不超过 +20%
CSD2X_RIGHT_CMOD_IS_LOW	8	右通道的 Cmod 超出范围
CSD2X_RIGHT_CMOD_IS_HIGH	16	右通道的 Cmod 超出范围

注意：

1. 调用 CSD2X_bFMEA_Left_Cmod_Rb_Check() 和 CSD2X_bFMEA_Right_Cmod_Rb_Check() API 前，需要完成背景扫描。
2. 检查 Cmod 范围时，全局中断被禁用。

BYTE CSD2X_bFMEA_Left_Cmod_Rb_Check(void)

BYTE CSD2X_bFMEA_Right_Cmod_Rb_Check(void)

说明

仅在选择 Rb 配置并使能 FMEA 特性时，该 API 才可用。该 API 将检查 C_{mod} 和 Rb 值的短接情况，另外还检查这两个值的乘积是否处于有效范围。有效范围被定义为容差为 40% 的 C_{mod} 与 Rb 的乘积。最小值为 $3.3*10^{-6}$ 秒，最大值为 $700*10^{-6}$ 秒。

分别使用 CSD2X_bFMEA_Left_Cmod_Rb_Check() 和 CSD2X_bFMEA_Right_Cmod_Rb_Check() 更新 WORD CSD2X_wCmod_Rb_L_Val 和 CSD2X_wCmod_Rb_R_Val 变量。这些变量包含 CSD2X_bFMEA_Left_Cmod_Rb_Check() 和 CSD2X_bFMEA_Right_Cmod_Rb_Check() 函数期间计算得出的 C_{mod} 与 Rb 的乘积 （单位为 µsec）。例如， CSD2X_wCmod_Rb_Val 包含值 120，对应 120 10-6 秒 （或 120 usec）。仅在 CSD2X_bFMEA_Left_Cmod_Rb_Check() （或 CSD2X_bFMEA_Right_Cmod_Rb_Check()） API 返回 bRetVal.4 值时，该值才有效。

C 原型：

```
BYTE CSD2X_bFMEA_Left_Cmod_Rb_Check(void);
BYTE CSD2X_bFMEA_Right_Cmod_Rb_Check(void)
```

汇编

```
lcall  CSD2X_bFMEA_Left_Cmod_Rb_Check
lcall  CSD2X_bFMEA_Right_Cmod_Rb_Check
```

参数

无

返回

bRetVal.0：C_{mod} 与 GND 短接
bRetVal.1：C_{mod} 与 Vdd 短接
bRetVal.2：Rb 与 Gnd 短接
bRetVal.3：Rb 与 Vdd 短接
bRetVal.4：Cmod 与 Rb 的乘积的偏差不超过 +40%
bRetVal.8：Cmod 与 Rb 的乘积低于有效范围，或 Cmod 被断开连接
bRetVal.16：Cmod 与 Rb 的乘积高于有效范围或 Rb 被断开连接

常量	数值	说明
CSD2X_LEFT_CMOD_SHORTED_TO_GND	0	左通道的 Cmod 与 GND 短接
CSD2X_ LEFT_CMOD_SHORTED_TO_VDD	1	左通道的 Cmod 与 Vdd 短接
CSD2X_LEFT_RB_SHORTED_TO_GND	2	左通道的 Rb 与 GND 短接
CSD2X_LEFT_RB_SHORTED_TO_VDD	3	左通道的 Rb 与 Vdd 短接
CSD2X_LEFT_CMODRB_WITHIN_40	4	左通道的 Cmod 与 Rb 的乘积的偏差不超过 +40%
CSD2X_LEFT_CMODRB_IS_LOW	8	左通道的 Cmod 与 Rb 的乘积超出范围
CSD2X_LEFT_CMODRB_IS_HIGH	16	左通道的 Cmod 与 Rb 的乘积超出范围
CSD2X_RIGHT_CMOD_SHORTED_TO_GND	0	右通道的 Cmod 与 GND 短接
CSD2X_RIGHT_CMOD_SHORTED_TO_VDD	1	右通道的 Cmod 与 Vdd 短接
CSD2X_RIGHT_RB_SHORTED_TO_GND	2	右通道的 Rb 与 GND 短接

常量	数值	说明
CSD2X_RIGHT_RB_SHORTED_TO_VDD	3	右通道的 Rb 与 Vdd 短接
CSD2X_RIGHT_CMODRB_WITHIN_40	4	右通道的 Cmod 与 Rb 的乘积的偏差不超过 +40%
CSD2X_RIGHT_CMODRB_IS_LOW	8	右通道的 Cmod 与 Rb 的乘积超出范围
CSD2X_RIGHT_CMODRB_IS_HIGH	16	右通道的 Cmod 与 Rb 的乘积超出范围

注意

1. 调用 CSD2X_bFMEA_Left_Cmod_Rb_Check() 和 CSD2X_bFMEA_Right_Cmod_Rb_Check() API 前，需要完成背景扫描。
2. 如果 Rb 小于 6 千欧姆，则 API 将返回 “Cmod 被短接地 ”，而不是 “Rb 被短接地 ”。需要在 Cmod 上增加一个上拉电阻，以便区分 Rb 与 GND 短接和 Cmod 与 GND 短接。
3. 如果 Rb 小于 6 千欧姆，则 API 将返回 “Cmod 与 Vdd 短接 ”，而不是 “Rb 与 Vdd 短接 ”。需要在 Cmod 上增加一个下拉电阻，以便区分 Rb 与 Vdd 短接和 Cmod 与 Vdd 短接。
4. 检查 Cmod 和 Rb 范围时，全局中断被禁用。

CSD2X_wCmod_Rb_L_Val 和 CSD2X_wCmod_Rb_R_Val 变量包含 Rb*Cmod 值。对于接近有效范围下限的 Rb*Cmod 值，其偏差最大可达 80% （在 Rb 配置下）。

示例固件源代码

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 CSD2X 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;
    CSD2X_Start();
    CSD2X_InitializeBaselines() ; //scan all sensors first time, init baseline
    CSD2X_SetDefaultFingerThresholds() ;
    //
    // Loop Forever
    //
    while (1)
    {
        CSD2X_ScanAllSensors();    //scan all sensors in array (buttons and sliders)
        CSD2X_UpdateAllBaselines(); //Update all baseline levels;
```

```
        //detect if any sensor is pressed
        if(CSD2X_bIsAnySensorActive())
        {
        // Add user code here to proceed with sensor touching
        }

        // now we are ready to send all status variables to chart program
        // communication here
        //
        // OUTPUT CSD2X_waSnsResult[x] <- Raw Counts
        // OUTPUT CSD2X_waSnsDiff[x] <- Difference
        // OUTPUT CSD2X_waSnsBaseline[x] <- Baseline
        // OUTPUT CSD2X_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 CSD2X module
//-------------------------------------------------------------------------

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

void main(void)
{
    M8C_EnableGInt;

    CSD2X_Start(); // Start CSD2X UM
    CSD2X_SetDefaultFingerThresholds(); // Set default thresholds for buttons
    // Initialize baseline for sensor number "3"
    CSD2X_InitializeSensorBaseline(3);

    while (1)
    {
        // Scan continuously sensor number "3" which is connected
        CSD2X_ScanSensor(0xFF, 3);
        CSD2X_UpdateSensorBaseline(3); // Update Baseline for sensor 3
        if(CSD2X_bIsSensorActive(3)) // check if sensor 3 is touched
        {
        // Add user code here to proceed with 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，此代码非常有用。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD2X 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;

    CSD2X_Start();
    CSD2X_InitializeBaselines();

    // set finger threshold for sensor "0"
    CSD2X_baBtnFThreshold[0] = 10;
    // set finger threshold for sensor "1"
    CSD2X_baBtnFThreshold[1] = 20;
    // set finger threshold for sensor "2"
    CSD2X_baBtnFThreshold[2] = 30;
    // set finger threshold for sensor "3"
    CSD2X_baBtnFThreshold[3] = 40;
    // set finger threshold for sensor "4"
    CSD2X_baBtnFThreshold[4] = 50;
    // set finger threshold for sensor "5"
    CSD2X_baBtnFThreshold[5] = 255;
    // set finger threshold for sensor "6"
    CSD2X_baBtnFThreshold[6] = 200;

        while (1) {
        // Scan continuously all sensors
        CSD2X_ScanAllSensors();
        CSD2X_UpdateAllBaselines();
        //detect if any sensor is pressed
        if(CSD2X_bIsAnySensorActive())
        {
        // Add user code here to proceed the buttons pressing
        }
    }
}
```

示例 4. 下面的示例说明的是 CSD2X_wGetRadialPos() 和 CSD2X_wGetRadialInc() API 如何工作的。您应该有一个在 CSD2X 向导中定义的辐射滑条。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD2X module
// Define one radial slider in CSD2X Wizard
//-------------------------------------------------------------------------

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

WORD Pos, Inc;
void main(void)
{
    M8C_EnableGInt;
```

```
    CSD2X_Start();

    CSD2X_InitializeBaselines() ;
    CSD2X_SetDefaultFingerThresholds();

        while (1)
        {
        CSD2X_ScanAllSensors();
        CSD2X_UpdateAllBaselines();

        Pos = CSD2X_wGetRadialPos(1);
        Inc = CSD2X_wGetRadialInc(1);
    }
}
```

示例 5. 启用 FMEA 和背景扫描时，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 CSD2X 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;
    CSD2X_Start();
    CSD2X_InitializeBaselines() ; //scan all sensors first time, init baseline
    CSD2X_SetDefaultFingerThresholds();
    if(CSD2X_bIsScanComplete())
    {
        //Short tests
        if (CSD2X_bFMEA_CheckGndShort())
        {
            // Add user code here to Sensor to Ground short alarm
            // CSD2X_baSensorShortGnd[0]
            // contains contains the masked bits for sensors 0 through 7
        }
        if (CSD2X_bFMEA_CheckVddShort())
        {
            // Add user code here to Sensor to Vdd short alarm
            // CSD2X_baSensorShortVdd[0]
            // contains contains the masked bits for sensors 0 through 7
        }
        if (CSD2X_bFMEA_CheckSensorShort())
        {
           // Add user code here to Sensor to Sensor short alarm
        }
        if (CSD2X_bFMEA_Left_Cmod_Check() != 4)
        {
            // Add user code here to alarm Cmod issues
            // CSD2X_wCmod_Val
        }
        if (CSD2X_bFMEA_Right_Cmod_Check() != 4)
        {
```

```
            // Add user code here to alarm Cmod and Rb issues
        }
    }
    //
    // Loop Forever
    //
    while (1)
    {
            CSD2X_ScanAllSensors(); //scan all sensors in array (buttons and sliders)

            if(CSD2X_bIsScanComplete())
            {
                CSD2X_UpdateAllBaselines(); //Update all baseline levels;
            }

            //detect if any sensor is pressed
            if(CSD2X_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 CSD2X_waSnsResult[x] <- Raw Counts
        // OUTPUT CSD2X_waSnsDiff[x] <- Difference
        // OUTPUT CSD2X_waSnsBaseline[x] <- Baseline
        // OUTPUT CSD2X_baSnsOnMask[x] <- Sensor On/Off

        // Do other tasks
    }
}
```

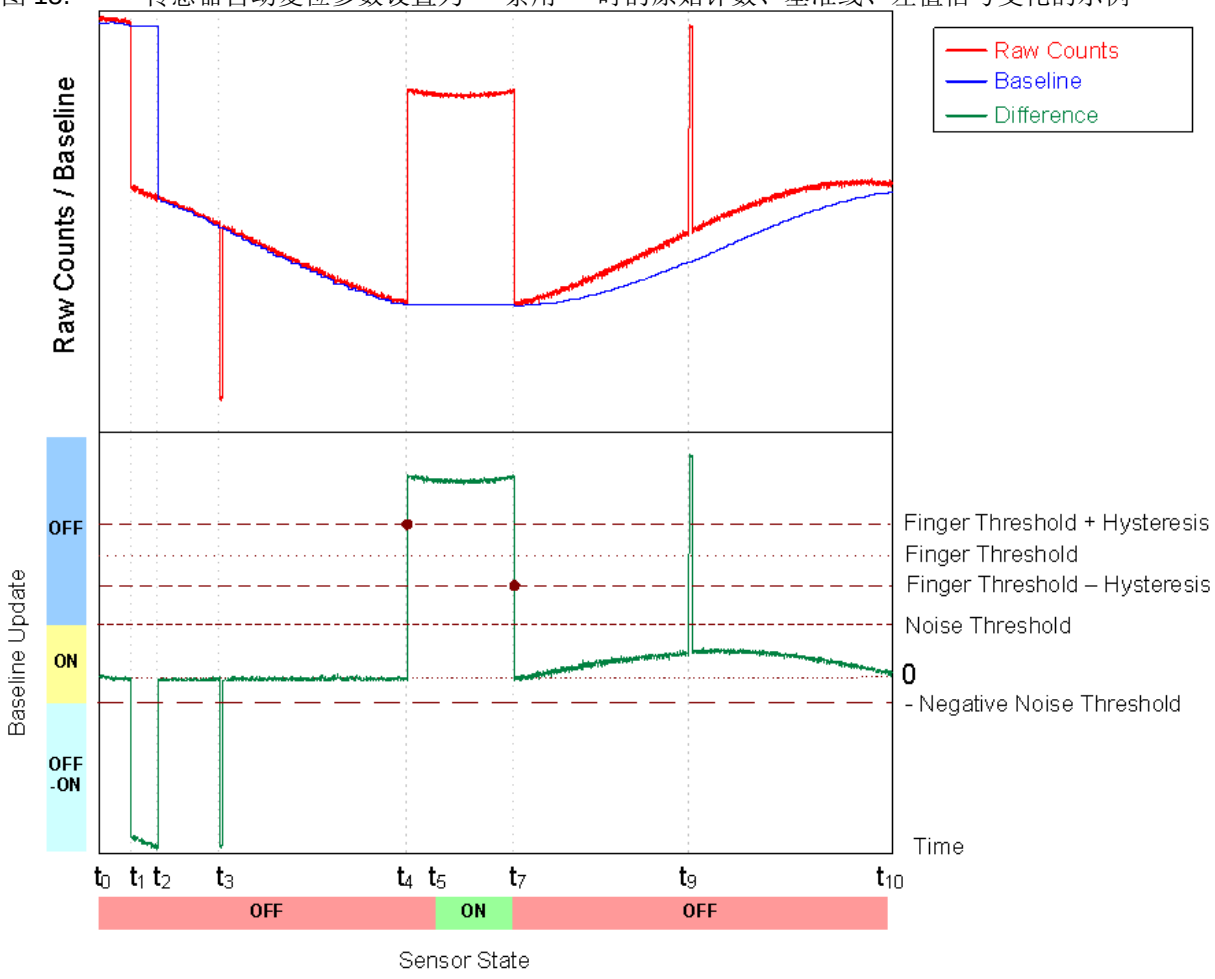
附录

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。

CSD2X 参数的交互

下图说明了基准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 15. 传感器自动复位参数设置为 “ 禁用 ” 时的原始计数、基准线、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。

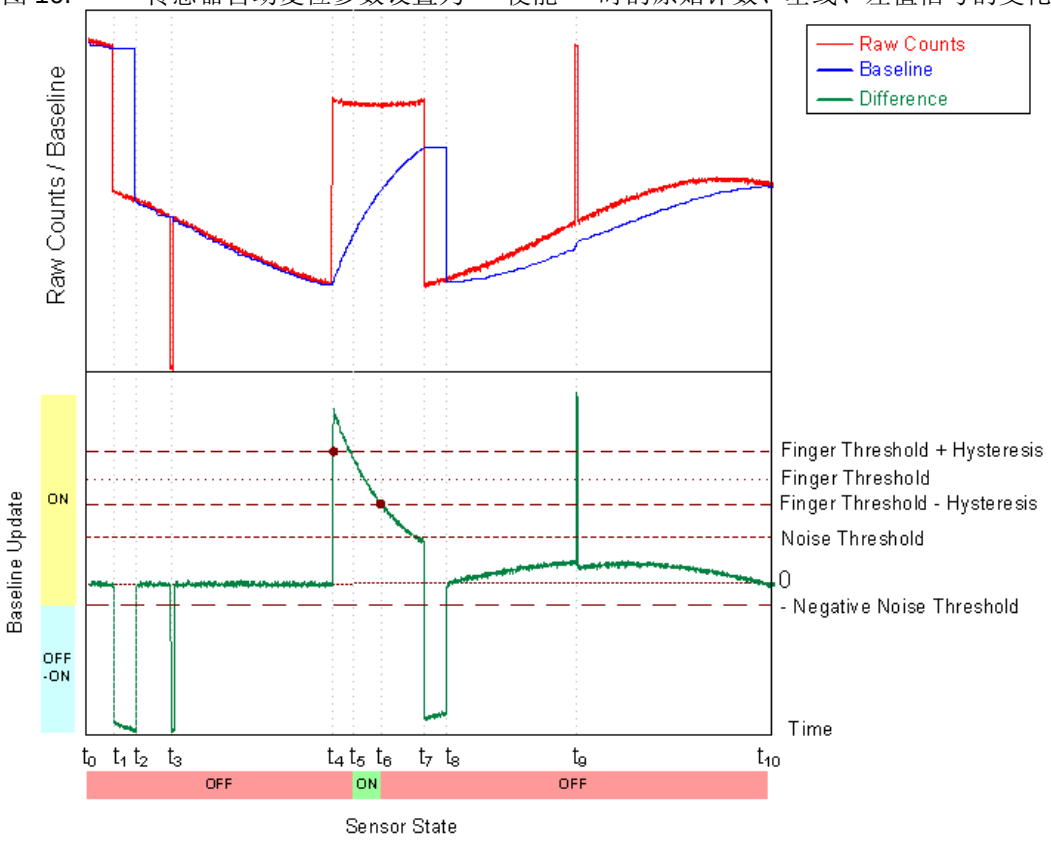
在 t_1 处，原始计数快速下降，并且负差超过 NegativeNoiseThreshold。如果器件在手指位于传感器上时被加电，过一段时间后移开了手指，则会发生这种情况。此时，基线更新机制冻结，内部超时计数器被激活。当差值信号低于 “ 低基线复位 ” 样品的 “ 负噪声阈值 ” 时，会复位基线。这是在 t_2 处发生的。

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 16. 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。这可用作附加的去抖动机制。

双通道扫描

双通道扫描是以同步模块函数形式进行的。

同步意味着同时扫描左右通道传感器，只有在完成上一对传感器的扫描之后，才能开始扫描下一对传感器。左右传感器可以具有不同的分辨率和扫描时间。在此情况下，“ 扫描传感器 ”（Scan Sensor）函数等待较慢的传感器，不对另一个传感器执行任何操作。如果左右传感器阵列由不同数量的传感器组成，则“ 扫描所有传感器 ”（ScanAllSensors）函数将会：

- 成对扫描所有可能的传感器
- 如果一侧的传感器比另一侧多，则一次扫描该通道上的剩余传感器中的一个

扫描从阵列末尾开始。在 GUI 中，会按照从左上到右下的传感器放置顺序分配传感器位置。要尽量确保您的双通道扫描高效进行，请执行下列操作：

- 在每个通道中放置相同数量的传感器
- 排列传感器，以便扫描时间较长的传感器成对放置
- 将所有滑条段放置在同一通道中
- 将大型传感器放置在左通道中
- 如果使用不同的参考，则首先放置具有较高参考值的传感器

版本历史记录

版本	创作者	说明
1.0	DHA	初始版本。
1.10	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。 2. 消除了 0.5 的移位，并添加了对负值的补偿。 3. 模拟比较器列 （AnalogComparatorColumn）和全局输出 （GlobalOutput）线路之间的连接显示在互连视图中。
1.10.b	DHA	在向导中添加了帮助文件。
1.20	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. 将 CSD2X_InitializeBaselines API 中的 “call” 指令更改为 “lcall”，以支持 CY8C28xxx 器件的 CSD2X_28IEPRS。 5. 更新了 CSD2X_wGetPortPinLeft() 和 CSD2X_wGetPortPinRight() API 函数的说明内容。

版本	创作者	说明
1.30	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. 更新了 SetScanMode() API 的说明内容。 4. 添加了滑条和辐射滑条的分辨率参数的上限值。 5. 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 / 最大值的说明。
1.30.b	DHA	增加了注意内容，建议不要将 IDAC 参数的 1X 值适用于新的设计。
2.00	DHA	1. 向用户模块增加了 FMEA 和背景扫描功能。 2. 更新了用户模块数据手册中的用户模块图。 3. 向 inc 文件中增加了 `@INSTANCE_NAME`_IDAC_RANGE。 将 “ 滑条设置 ” 选项卡的名称更改为 “ 传感器设置 ”。 5. 说明了数据手册中动态配置的限制。
2.10	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. 向中断服务子程序添加了 CUR_PP 和 IDX_PP，以便处理启用背景扫描时的场合。 5. 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。 6. 优化了 “Start User Modul+L143e” 函数代码。 7. 初始化 CY8C28xxx 系列的抽取滤波器放置。 8. 更新了用户向导滑条的设置算法，以考虑到自由引脚。

注意： PSoC Designer 5.1 在所有用户模块数据手册中引入了 “ 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 © 2009-2014 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。