初步

CapSense® Sigma-Delta 数据表 CSD V 1.50

Copyright © 2010 Cypress Semiconductor Corporation. All Rights Reserved.

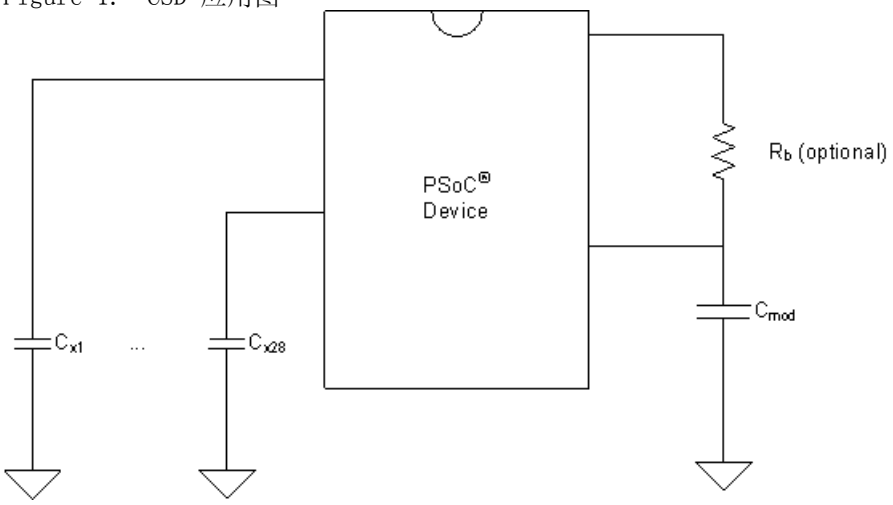
资源	CapSense 模块	API 存储器 （字节）		每个传感器所使用的引脚数
		闪存	RAM	
闪存、RAM 和引脚的使用因传感器数量和配置而异。				
带有 1 个传感器、基于 PRS 的用户模块	1	1017	26	2-5
带有 1 个传感器、基于预分频器的 PRS	1	1006	26	2-5
每个附加 CapSense 按钮		5	11	1
使用带有五段的电容式滑条时，静态代码和 RAM 增加		584	79	
每个附加滑条段		2	10	1
当使用滑条双工法时，静态代码和 RAM 增加		12		1

功能和概述

- 扫描 1 ～ 28 个电容传感器。
- 感应能力最多可穿透 15 毫米的玻璃覆盖层。
- 使用线缆传感器时，检测距离接近 20 厘米。
- 对交流电源噪声、EMC 噪声和电源电压变化，具有极强的抗干扰能力。
- 支持独立传感器和滑条电容传感器的任意组合。
- 通过双工法，使滑条传感器的物理分辨率增加一倍。
- 利用内插法，提高滑条传感器的分辨率。
- 带有两个滑条传感器的触摸板支架。
- 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，屏蔽电极仍可为可靠的运行提供保证。
- 通过 CSD 向导完成传感器和引脚分配。
- 使用集成基准线更新算法处理温度、湿度和静电释放 (ESD) 事件。
- 可轻松调整的操作参数。
- PC GUI 应用程序支持原始数据的实时监控和参数优化。

CSD（使用 Sigma Delta 调制器的电容式感应技术）采用开关电容技术来提供 CapSense® 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。

Figure 1. CSD 应用图



快速入门

1. 如果用到，请选择并布置需要专用引脚 （如 I^2C 或 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. 设置每个传感器的传感器设置。
6. 设置引脚和全局参数。 阅读参数说明，遵守各种要求和相关指南。
7. 生成应用，并切换到应用编辑器。
8. 根据需要调整采样代码，以部署独立传感器、滑条传感器或触摸板。
9. 将 I^2C-USB 桥连接至目标电路板，并观察信号。
10. 更改 CSD 参数以优化设置和重建应用。
11. 对 PSoC 器件编程并验证模块操作。 调整 CSD 参数以满足 SNR 5:1 的最低要求，如 *CapSense 应用的信噪比要求*中所述 – AN2403

如遇任何问题，请参见*附录*中的*故障排除*部分。

[+] Feedback

功能说明

电容式传感器包括物理、电气和软件组件：

- 物理组件
  - 即物理传感器本身，通常其传导模式基于与 PSoC 相连的 PCB，PSoC 是一种显示屏外覆有绝缘层、软膜或透明覆盖层的装置。
- 电气组件
  - 用于将传感器电容转换为数字格式。 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，可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。
  - 对于连续的相关传感器 （如滑条和触摸板），所提供的 API 可通过插值算法以高于传感器物理分辨率的更高分辨率进行定位。 例如，您可以创建一个包含 10 个传感器的音量滑条，并使用给定的固件将音量级别数扩展为 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，用于确定其之间的传导体 （例如手指）的位置。

测量电容有多种方法，此用户模块中采用开关电容与 delta-sigma 调制器配合使用的方法。

传感器阵列包含独立传感器、滑条传感器以及触摸板，触摸板部署为一对互相垂直的滑条。 高层决策逻辑可为环境因素 （如温度、湿度和电源电压变动等）提供补偿。 独立的屏蔽电极可用于屏蔽传感器阵列以降低杂散电容，从而在有水膜或水滴的情况下实现更可靠的操作。

高级软件功能可提供滑条双工法，以便在两个物理位置可以使用一个电气传感器，用以提高分辨率。 这些功能还可以在物理传感器位置之间对所求解的传感器位置进行进一步插值。

在首次使用 CSD 用户模块之前，建议您先阅读以下文档：

- *PSoC 器件的《技术参考手册》*

阅读 CSD 用户模块文档之后，建议阅读下列应用说明。 各应用笔记可在赛普拉斯半导体公司网站 www.cypress.com 上获取：

- *CapSense 最佳实践* – AN2394
- *CapSense 应用的信噪比要求* – AN2403
- *用于调试 CapSense 应用的绘图工具* – AN2397
- *PSoC CapSense 应用的 EMC 设计考虑事项* – AN2318
- *电容式感应应用中的能耗和睡眠考虑事项* – AN2360
- *PSoC CapSense 布线指南* – AN2292
- *通用异步发送器的软件实现* – AN2399
- *电容式感应的防水* – AN2398

[+] Feedback

电容测量操作

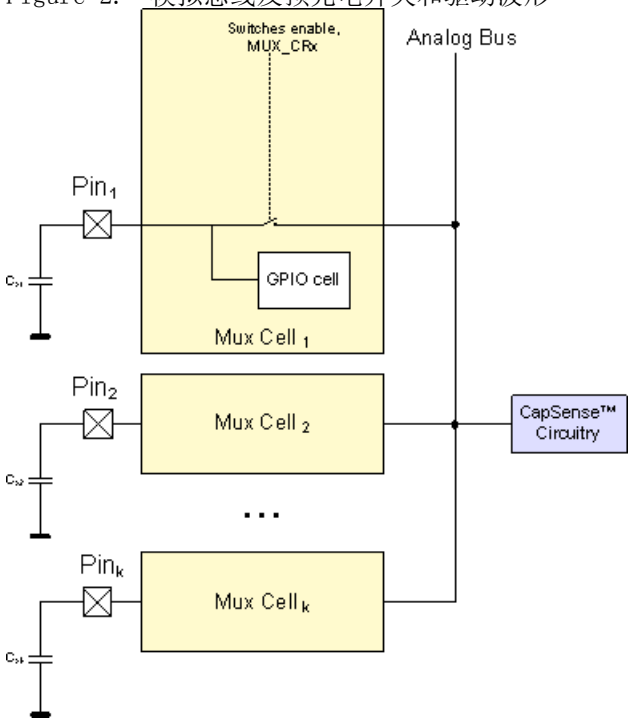
决策逻辑通过固件实现。 通过固件分析电容的测量，跟踪环境因素造成的缓慢电容变化，运行决策逻辑，以检测按钮触摸变化并计算滑条位置。

扫描传感器阵列

CY8C21x12 系列设备具有内置模拟总线， 可以使电容传感器连接到任意 PSoC 引脚。 CSD 用户模块使用内部预充电开关，在时钟信号 Ph_1 阶段为工作的传感器充电，在 Ph_2 阶段将模拟总线与传感器相连。 sigma-delta 调制器的调制电容和比较器的输入端与模拟总线始终相连。

固件通过在 MUX_CRx 寄存器中设置相应的位来执行传感器系列扫描。

Figure 2. 模拟总线及预充电开关和驱动波形

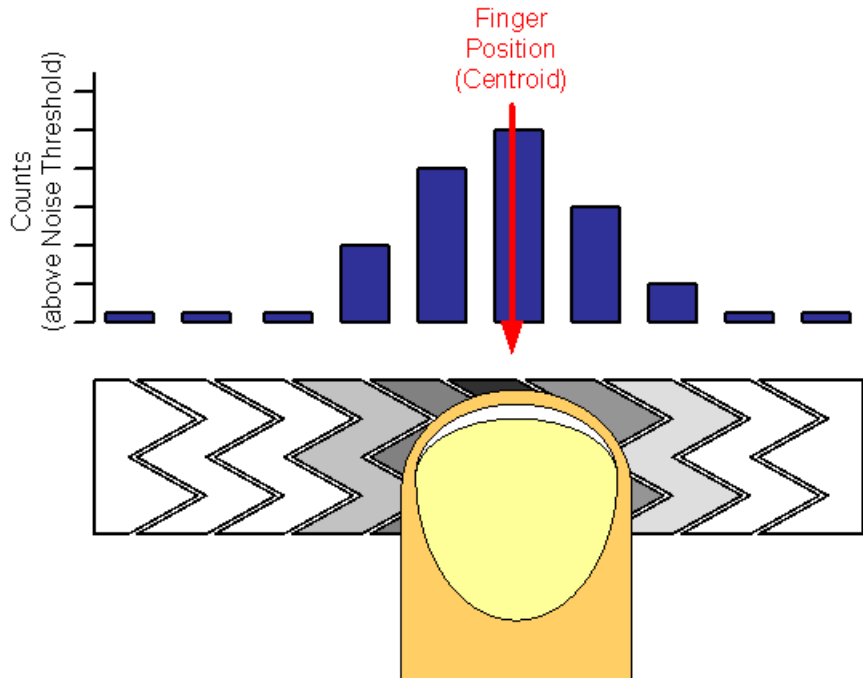


[+] Feedback

滑条

滑条用于需要渐进调整的控件。 示例包括照明控件 （调光器）、音量控件、图示均衡器和速度控件。 这些传感器在布局上彼此相邻。 某个传感器的动作会导致邻近的其他传感器的部分动作。 通过计算已激活传感器组的质心位置，可以确定滑条的实际位置。 通过 CSD 向导可以调整滑条，方法是建立若干组，每组滑条有特定的次序。 传感器滑条数量的实际下限值是五，上限值仅取决于所选 PSoC 设备提供的传感器位置的数值。

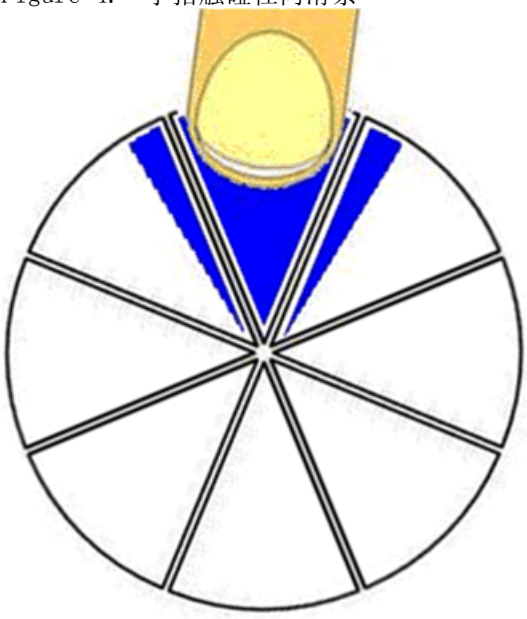
Figure 3. 对物理传感器位置排序



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。

径向滑条

Figure 4. 手指触碰径向滑条



[+] Feedback

对于 CSD 用户模块，提供两种滑条类型：线性滑条和辐射滑条。 径向滑条与线性滑条类似。 但是线性滑条有起始点和结束点，径向滑条却没有。 发生碰触时，质心计算算法将考虑传感器开关切换到电流开关左右两侧的次数。 径向滑条未采用双工法。

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() 则返回以分辨率为单位的手指位移。 当手指以顺时针方向移动时，会产生正的偏移。

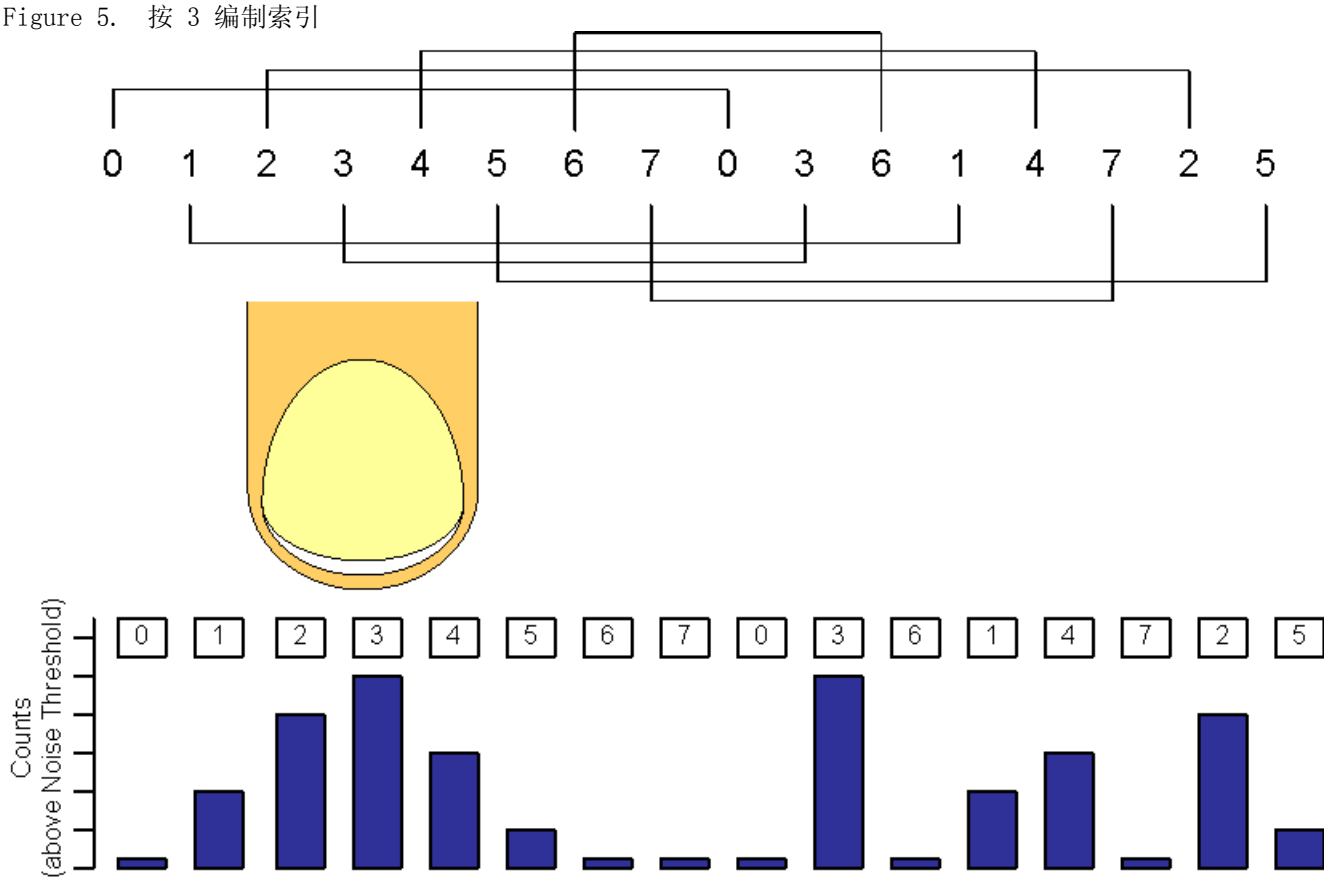
参考点 (0) 位于第一个传感器的中心。 线性和径向滑条的分辨率均限制为 3000。

双工

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

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

Figure 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 设计，某些传感器对可能需要更长的路由。 当选择双工时，双工传感器索引表由 CSD 向导自动生成。 下表列出了不同滑条段计数的双工序列：

[+] Feedback

Table 1. 不同滑条部分计数的双工序列

滑条段总计数	段序列
10	0, 1, 2, 3, 4, 0, 3, 1, 4, 2
12	0, 1, 2, 3, 4, 5, 0, 3, 1, 4, 2, 5
14	0, 1, 2, 3, 4, 5, 6, 0, 3, 6, 1, 4, 2, 5
16	0, 1, 2, 3, 4, 5, 6, 7, 0, 3, 6, 1, 4, 7, 2, 5
18	0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 3, 6, 1, 4, 7, 2, 5, 8
20	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 3, 6, 9, 1, 4, 7, 2, 5, 8
22	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8
24	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11
26	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 3, 6, 9, 12, 1, 4, 7, 10, 2, 5, 8, 11
28	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 3, 6, 9, 12, 1, 4, 7, 10, 13, 2, 5, 8, 11
30	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 3, 6, 9, 12, 1, 4, 7, 10, 13, 2, 5, 8, 11, 14
32	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 2, 5, 8, 11, 14
34	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14
36	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14, 17
38	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14, 17
40	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17
42	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17, 20
44	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17, 20
46	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10, 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20
48	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10, 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20, 23

[+] Feedback

滑条段总计数	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

内插和测量

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

要使用质心计算内插位置，首先要扫描阵列，以确定所给的传感器位置是否有效。 要求提供一定数量的相邻传感器信号，且高于噪声阈值。 如果发现最强信号，将使用此信号和那些大于噪声阈值的连续信号计算质心。 使用少至两个、多至 （通常）八个传感器，利用下列公式计算质心：

Equation 1

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

计算出的值通常是分数。 要将质心报告给特定的分辨率 （例如：对于 12 个传感器，范围值 0 ～ 100），需要将质心数值乘以计算得出的量极。 另一种更有效的方法是将内插和按比例计算的方法统一到单一计算中，按所需的量级直接报告结果。 这是通过高级 API 实现的。

滑条传感器计数和分辨率在 CSD 向导中进行设置。 比例值通过向导计算，并存储为分数值。

质心分辨率的乘法器占用三个字节，相应位的定义如下：

分辨率乘法器 MSB								
位	7	6	5	4	3	2	1	0
乘法器	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
分辨率乘法器 ISB								
乘法器	128	64	32	18	16	8	4	2
分辨率乘法器 LSB								
乘法器	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256

使用以下公式确定分辨率：

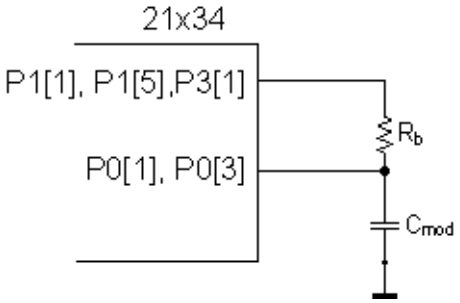
分辨率 = （传感器数 – 1）x 乘法器

质心以 24-bit 无符号整数保存，其分辨率是传感器和乘法器数值的函数。

选择反馈组件

用户模块需要外部调制电容 C_{mod} 和调制器反馈电阻 R_b。 电容可以连接到 P0[1]、P0[3] 端口引脚和 Vss 地。 反馈电阻 R_b 可以连接到端口引脚 P1[1]、P1[5]、P3[1] 和电容引脚。 通过用户模块参数设置可以选择引脚。 不要将选定用于调制器组件连接的引脚用于其他任何目的。

Figure 6. 外部组件连接



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

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

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

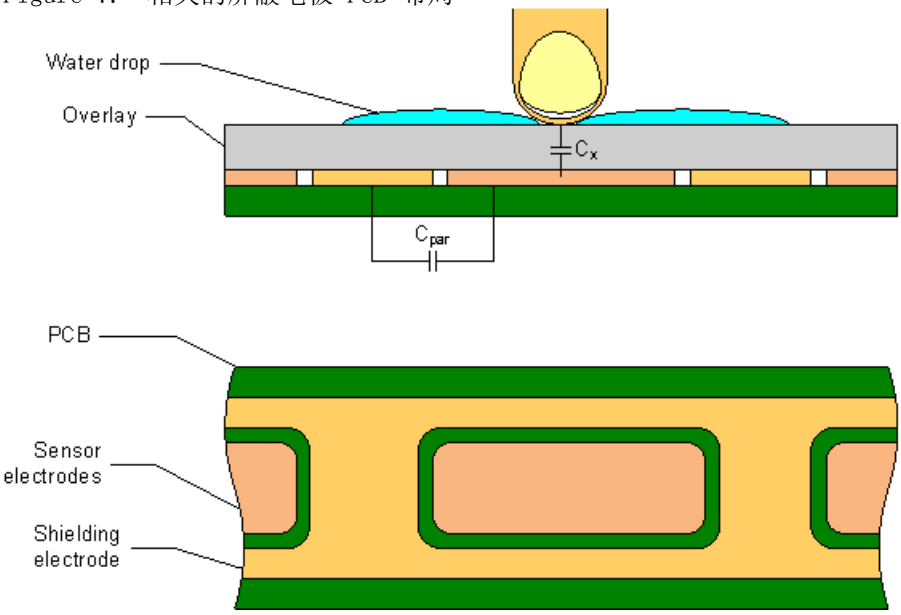
[+] Feedback

屏蔽电极

某些应用场合要求即使存在水膜或水滴，也能可靠地运行。 在白色家电、汽车应用场合、各种工业应用场合和其他场合，都需要即使有水、冰和湿度变化也不会提供假触发的电容式传感器。 对于此种情况，可以使用单独的屏蔽电极。 此电极位于感应电极之后或其外侧。 如果设备绝缘覆盖层表面有水膜，则屏蔽和感应电极之间的耦合程度会加剧。 屏蔽电极有帮于降低寄生电容的影响，为处理传感电容的变化提供了更具动态性的数值范围。

在某些应用场合，选择屏蔽电极信号及其相对于感测电极的位置十分有用，这样，增强两个电极之间的耦合就会造引起测电极电容测量的触摸变化减弱。 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。

Figure 7. 相关的屏蔽电极 PCB 布局



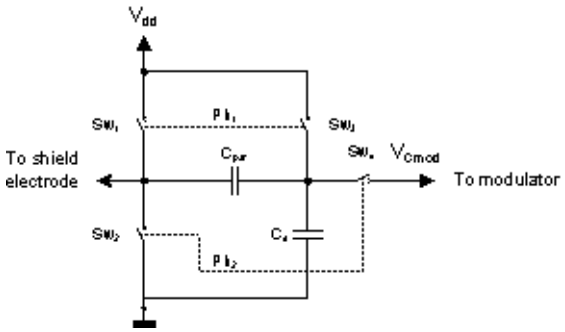
上图展示了可能为按键屏蔽电极采取的一种布局配置。 屏蔽电极尤其适用于透明的 ITO 触摸板设备，在这种设备中，它不但可阻止 LCD 驱动电极的噪声影响，同时可减少杂散电容。

在此示例中，按钮覆盖有屏蔽电极板。 作为另一替代方法，屏蔽电极可以安装在相对的 PCB 层上，其中包括按钮下面的平板。 在这种情况下，建议使用开口方案，填充率大约为 30 到 40%。 在此种情况下，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，调制器电流下降。 在实际测试中，通过 API 可以增大调制器参考电压，以便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。

[+] Feedback

在此用户模块中，用于预充电时钟的同一信号还提供给屏蔽电极。 具体操作请见下图：



C_s - 传感器总电容

C_{par} - 屏蔽电极与感应电极之间的电容

开关 Sw_1 和 Sw_3 在 Ph_1 阶段接通，开关 Sw_2 和 Sw_4 在相位 Ph_2 阶段接通。 C_{par} 在 Ph_1 阶段放电，在 Ph_2 阶段充电。 调制器电流是 C_s 和 C_{par} 电流的代数和，可以通过以下等式计算：

Equation 2

$$I_{mod} = I_C - I_{Cpar} = f_s C_s (V_{dd} - V_{Cmod}) - f_s C_{par} V_{Cmod}$$

另外，降低调制器参考电压可降低寄生电容 C_{par} 对调制器总电流的影响。 因此，在防水感应技术中，可以最大程度地降低调制器参考的最佳值。

如等式 2 中所示，随着电极之间的耦合增加，调制器电流会下降。 这样可以区分开由于水和手指产生的信号，对于固件处理很有帮助。

屏蔽电极可以连接到任何空闲行输出总线。

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两个选择选项作为预充电开关的时钟源：

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

首次选择用户模块时，您应该选择必需的配置。 以后可以通过右键单击 CSD 用户模块并选择 “ 用户模块选择选项 ” 来更改此选择。

PRS 配置将 PRS16 模块用作时钟源。 PRS 源提供扩频操作，确保很好地抵抗外部噪声源的噪声。 另外，带有扩频时钟的设计能够达到较低的电磁辐射级别。 如果应用的目标是通过 EMC/EMI 测试或者必须在嘈杂环境下提供可靠操作，则建议使用 PRS16 配置。 下表比较了这两种配置：

[+] Feedback

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

比较器参考源

比较器参考源用来构成比较器参考电压。 该参考电压值决定了灵敏度。 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 - 脉冲宽度调制信号驱动
- 外部电阻电压分频器
- PRSPWM 或预分频器 - 脉冲宽度调制信号的外部 RC 滤波器

下表汇总了参考选择选项：

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

只能使用参考选择带隙 (VBG) 或模拟调制器 (ASE11)。 其他两种选择用于解决具体问题。

[+] Feedback

直流和交流电气特性

Table 2. 电源电压

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

Table 3. 噪声

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

a. 当扫描速度减慢且基线计数增加时，信噪比提高。

Table 4. 功耗

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

放置

当实例化用户模块时，会自动放置适用于用户模块的模块，不提供其他放置方式。 CSD 用户模块使用 3 个数字块 (DBB00、DBB01 和 DCB02)。 这些 DCB03 数字块是可供用户使用。

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

当放置电容式传感器连接时，避免使用 P1[0] 和 P1[1]。 这些引脚用来对部件进行编程，可能具有过多影响传感器灵敏度和噪声的路由电容。

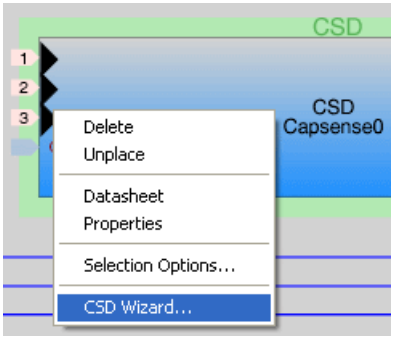
[+] Feedback

CSD 向导

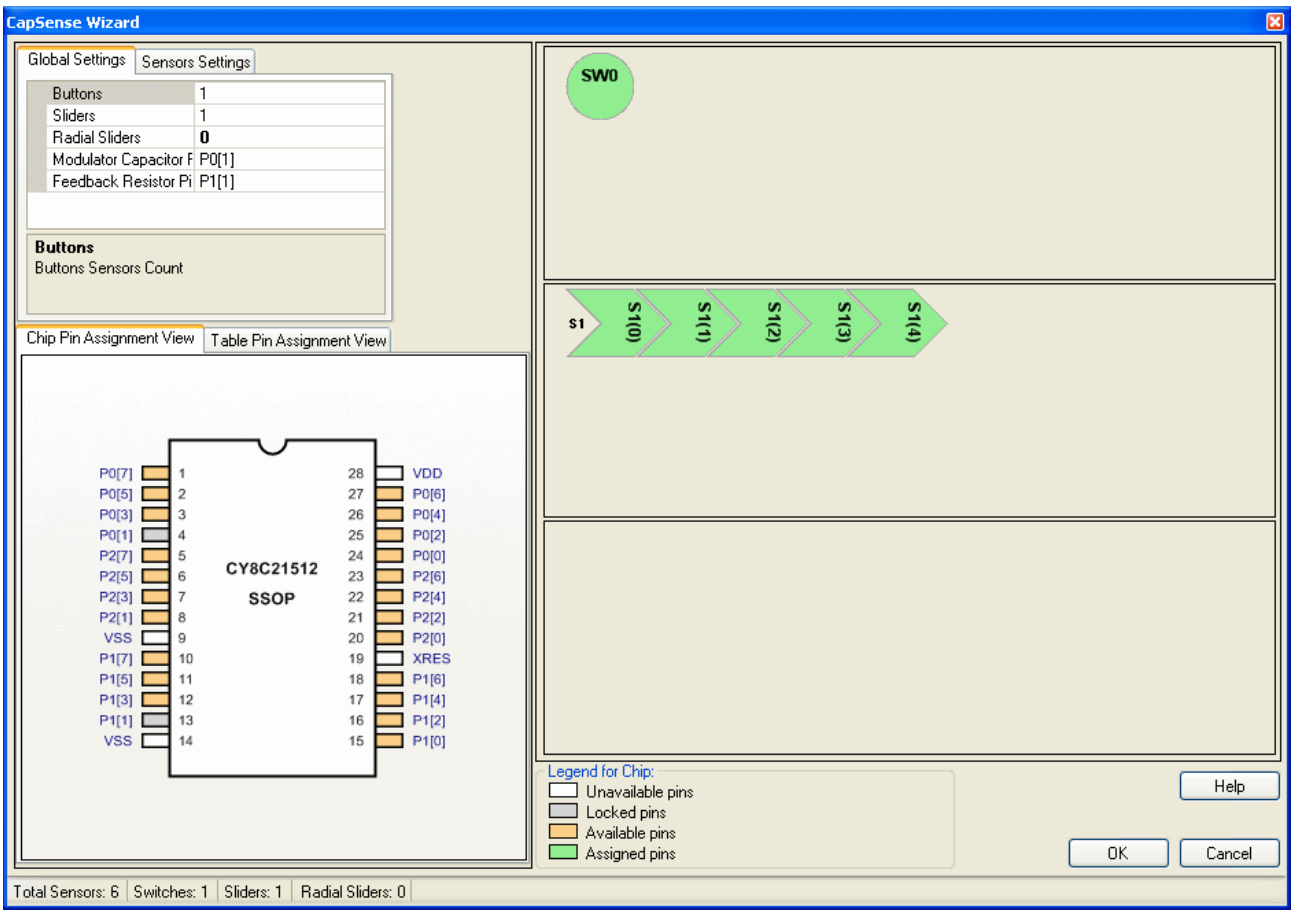
CSD 向导用来设置 CapSense 按钮、滑条和接近传感器的引脚输出。 您可以使用一个拖放界面选择所需配置并分配按钮和段。

访问向导

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



2. 打开的向导将显示数字输入框，可在其中输入传感器数量和滑条传感器数量。



[+] Feedback

向导引脚图例

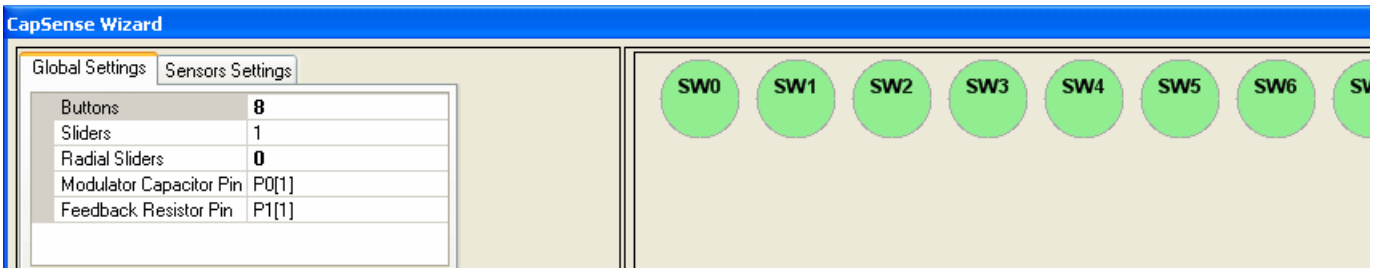
白色 – 该引脚不能用作 CapSense 输入。

灰色 – 该引脚处于锁定状态。 这可能是由于两个原因。 第一种可能性是另一个用户模块（如 LCD 或 I^2C）已经声明使用了该引脚。 第二种可能的原因是引脚名称已更改，不再是默认值。 要恢复使用引脚的默认名称，请在 “ 引脚分布 ” 视图中展开该引脚，然后选择 **默认值** ，该选项在 **选择** 菜单中。 这时该引脚在向导中便可用于分配。

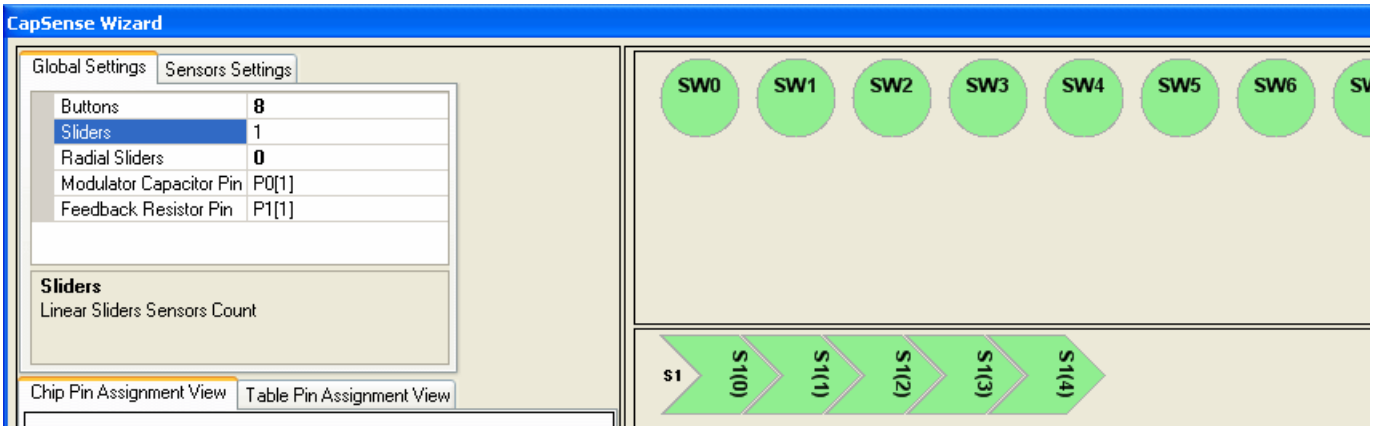
橙色 – 该引脚可用于分配。

绿色 – 该引脚已经分配为一个 CapSense 输入。

3. 键入独立传感器的数量。 传感器数受可用引脚数限制。

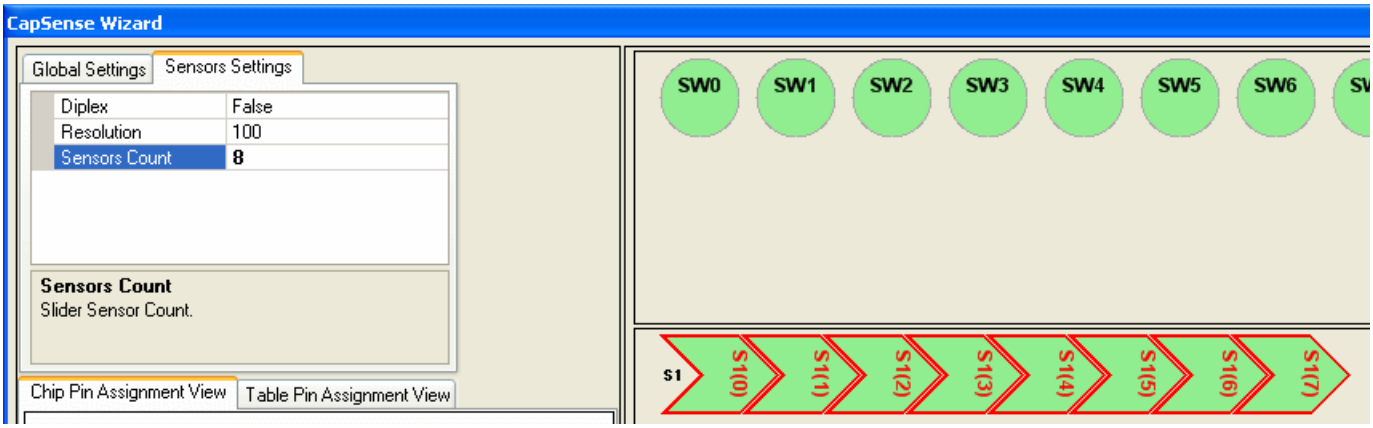


4. 键入滑条的数量。 X-Y 触摸板需要两个滑条 （下面仅选择了一个）。

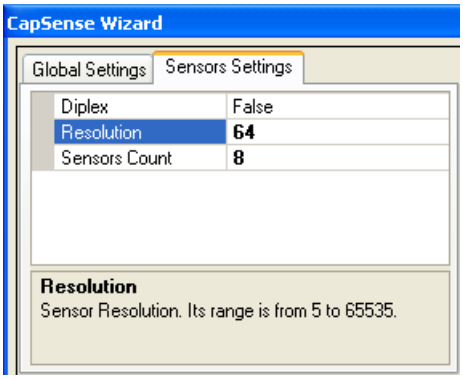


[+] Feedback

5. 选择滑条以输入该滑条的设置。 键入滑条中传感器元件的数量。 滑条传感器中传感器的实际最小数值是五个。 最大值受引脚数量的限制。



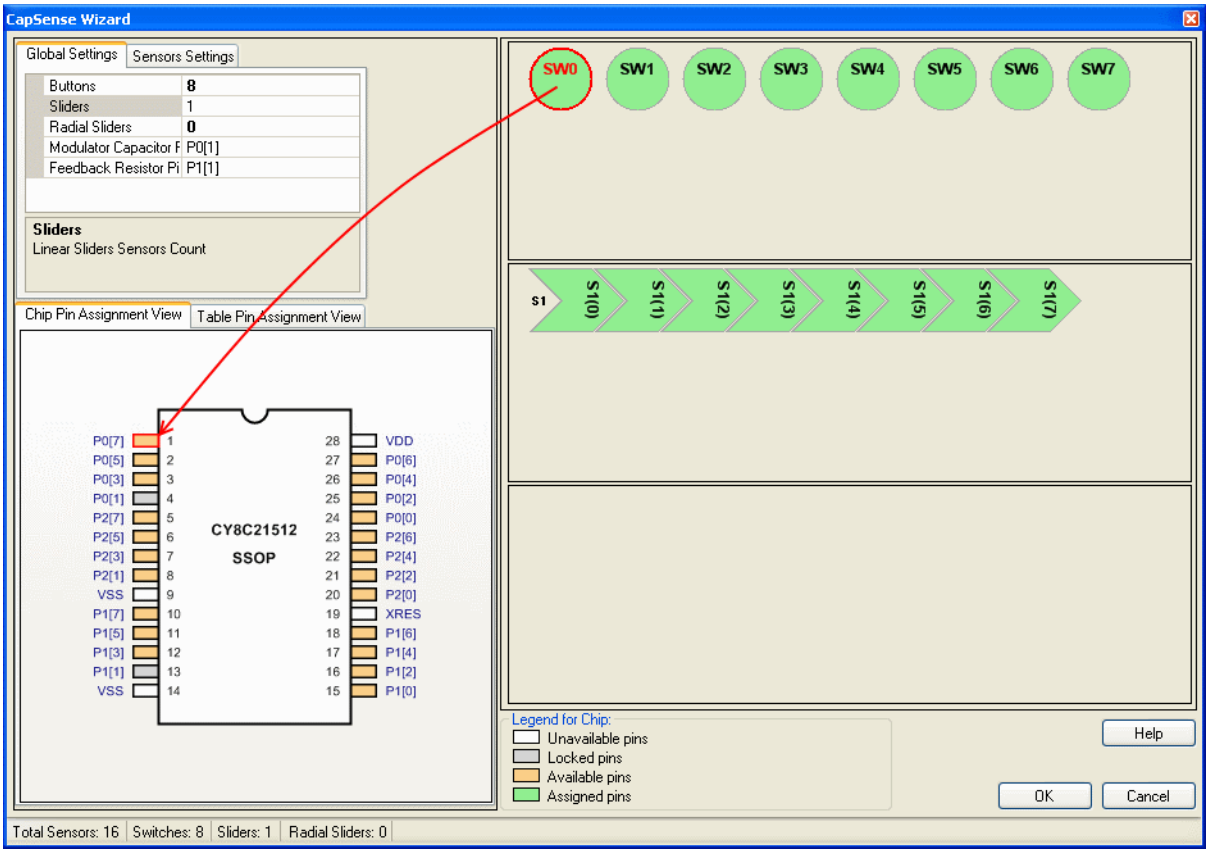
6. 键入输出分辨率。 最小值为 5。 对于双工型滑条，最大值为（传感器的引脚数 - 1）x 2^{16} - 1 或（2 x 传感器的引脚数 - 1）x 2^{16} - 1。 软件根据相邻滑条传感器上的相对信号强度，尝试更改此分辨率的触摸情况。



7. 如果需要，请选择 “ 双工 ”。 这会将选择用于传感器的引脚数映射到板上两倍数量的传感器位置，仅显示了双工传感器的前一半部分；后一半部分按前面 “ 双工 ” 章节所述自动映射。 有关引脚连接的双工表，请参见 “ 双工 ” 一节。

[+] Feedback

8. 左键单击传感器，将其拖动到任意可用引脚。 选定后，端口引脚将变为绿色，且不能再用于分配。 通过拖动端口引脚的传感器，可以更改传感器分配。



9. 对其他独立传感器重复以上操作。
10. 将单个滑条传感器映射到物理端口引脚的操作与单个传感器相同。
11. 单击 **“ 确定 ”(OK)** 以接受数据并返回到 PSoC Designer。

至此就完成了传感器的布置。 在 “ 设备编辑器 ”(Device Editor) 窗口中右键单击并选择 **“ 刷新 ”(Refresh)** 以更新引脚连接。

设置用户模块参数并生成应用程序。 如果愿意，现在您可以改编一个样本项目。

要更改引脚分配，请将光标放在已分配的引脚上，单击该引脚，然后将其拖放至开关框外面。 此引脚现在处于未分配状态，您可以重新分配。

完成向导后，单击 “ 生成应用程序 ”(Generate Application)。 根据您输入的传感器个数、引脚分配、复用和分辨率，将生成一组表。 这些表位于 CSD_Table.asm 中。

[+] Feedback

传感器表

在传感器表中，每个传感器对应一个 2-byte 条目。 第一个字节是端口号，第二个字节是位的位掩码 （而不是位编号）。 该表包含所有独立传感器，然后按次序列出每个传感器。 下面是一个包含六个传感器的示例表：

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

该表由 CSD_wGetPortPin() 例程使用。

组表

组表对每一组按钮传感器或滑条进行定义。 每个滑条对应一个条目，另外还有一个条目用于备用按钮传感器。 第一个条目始终是备用传感器。 每个条目为六字节。 第一个字节是传感器表中组开始的索引。 第二个字节是该组中的传感器数量。 第三个字节指示滑条是否为双工 （4 为双工，0 为非双工）。 第四、五、六个字节是固定值乘数，将其与计算出的滑条质心相乘可以获得 CSD 向导中所需的分辨率。

```
CSD_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

当某个组是滑条且采用双工时，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。 该表包含两部分：每个滑条的传感器映射，以及每个单独滑条对其表的引用。 此处展示了一个八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
```


[+] Feedback

参数和资源

手指阈值

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

手指检测阈值适用于所有传感器和滑条。 对于单个传感器 （不包含在滑条组中），这些阈值是变量，在 baBtnFThreshold[] 阵列中提供。 可以使用 SetDefaultFingerThresholds() 函数将阈值设置为设备编辑器中默认值设置。 要调整单个传感器的灵敏度，请更改每个传感器的 baBtnFThreshold[] 值。 （此字节阵列的大小等于部署的单个传感器的数量。）

可能值的范围为从 5 到 255。

噪音阈值

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

基准更新阈值

如果新的原始计数值高于当前基准线，差值低于噪声阈值 （传感器自动复位参数设置为 “ 禁用 ”），则当前基准线与原始计数的差值累计到桶形变量（″水桶″）中。 当桶形电极充满时，基准线按某个值递增，并清空水桶。 此参数设置了基准线递增时水桶必须要达到的阈值。 可能值为 0 到 255。 参数值越大，基准线更新速度越慢。 如果需要更频繁的基准线更新，将减小此参数。

LowBaselineReset

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

传感器自动复位

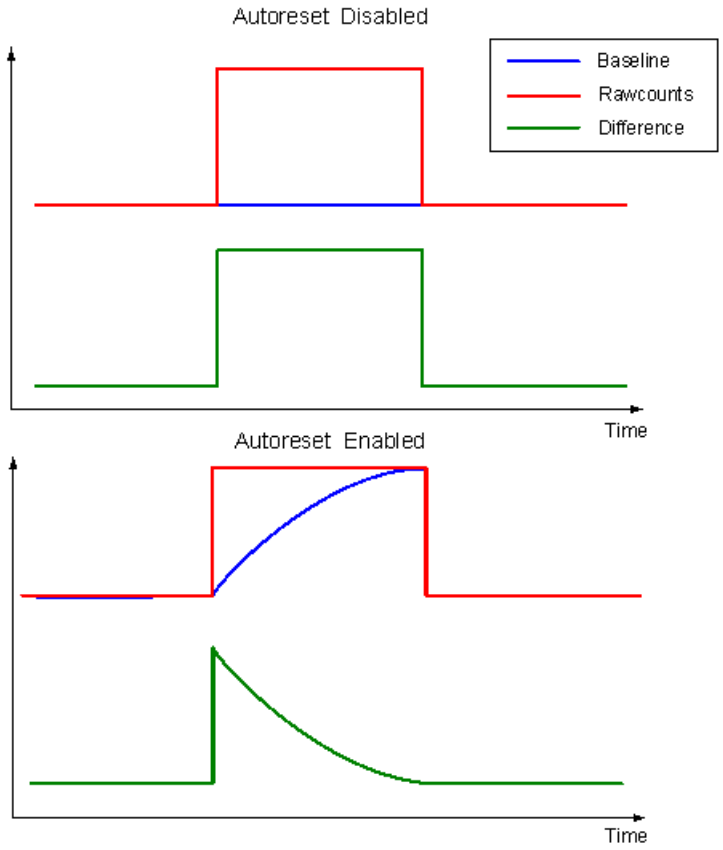
此参数决定是在所有情况下都更新基准线还是仅当信号差低于噪音阈值时才更新基准线。 如果设置为 **Enabled**，基准线将持续更新。 该设置限定了传感器的最大持续时间长度（典型值为 5 - 10s），它会防止当没有任何东西触摸传感器而原始计数突然升高时传感器一直保持开启状态。 较大的供电电压波动、高能射频噪音源或极快的温度变化都会导致这种突然升高。

如果该参数设置为 **Disabled**，则仅当原始数值与基准线之差低于 “ 噪音阈值 ” 参数时才会更新基准线。 除非是遇到在任何物体未触碰传感器而原始计数突然上升时传感器始终打开的问题，否则应将此参数保留为 “ 禁用 ”。

[+] Feedback

下图说明了此参数对基准线更新的影响。

Figure 8. 传感器自动复位参数



迟滞

“ 迟滞 ” 参数会与手指阈值相加或相减，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的和。 如果传感器处于活动状态，则差值计数必须低于手指阈值与迟滞的差。 此参数用于增加手指检测算法的防反跳和粘着性程度。 当调用 bIsSensorActive() 或 bIsAnySensorActive() 时，计算带有迟滞的阈值。 可以用 bIsSensorActive() 或 baSnsOnMask[] 阵列的返回值监控传感器状态。 可能的值为 0 到 255，但是必须小于 “ 手指阈值 ” 参数设置。

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

反跳

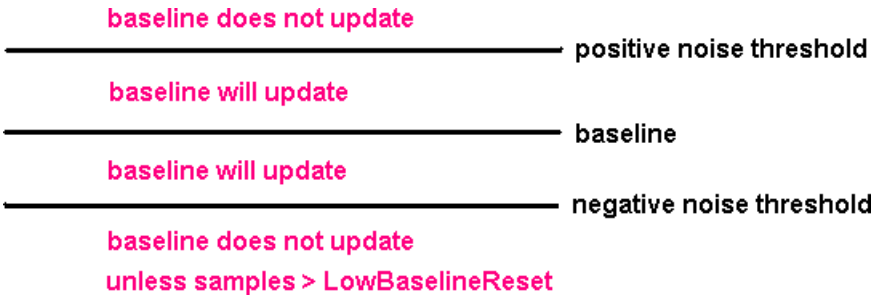
“ 反跳 ” 参数为传感器的活动状态切换添加了一个反跳计数器。 为了让传感器能够从不活动状态切换到活动状态，在规定的样本数量内，差异计数值必须保持在手指阈值加迟滞值之上。 防反跳计数器按 bIsSensorActive 或 bIsAnySensorActive API 函数递增。

可能值为 1 到 255。 设置为 1 则不提供防反跳。

NegativeNoiseThreshold

NegativeNoiseThreshold 参数会添加一个负的差值阈值。 如果当前原始计数低于基准线且它们的差大于此阈值，则不更新基准线。 但是，如果当前原始数值一直保持低状态 （差值大于阈值），并且次数超过 LowBaselineReset 参数指定的采样次数，基准线会被重置。

[+] Feedback



扫描速度

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

- 信噪比提高
- 更高的抗电源和温度变化能力
- 很少需要系统中断延迟；可以处理较长的中断

有关中断延迟的更多信息，请参见 “ 警告 ” 一节。

分辨率

此参数确定扫描分辨率 （以 bit 为单位）。 可以用 9 到 16 位的分辨率来扫描传感器。 N 位的扫描分辨率最大原始计数为 2^N-1。

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

扫描速度和分辨率通过下表所示方式影响 VC1、VC2、VC3 和 ADCPWM 分频器：

扫描速度	VC1
超快	1
快速	2
正常	4
慢速	8

分辨率 （以 bit 为单位）	VC2	VC3	ADCPWM
9	8	16	4
10	8	32	4
11	8	64	4
12	8	128	4

[+] Feedback

分辨率 （以 bit 为单位）	VC2	VC3	ADCPWM
13	8	256	4
14	8	256	8
15	8	256	16
16	8	256	32

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

Table 5. 对于 24 MHz IMO 操作、PRS16 配置的情况，扫描时间 （以 µs 为单位）与扫描速度和分辨率的关系

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

Table 6. 对于 24 MHz IMO 操作、带有预分频器配置的 PRS8 的情况，扫描时间 （以 µs 为单位）与扫描速度和分辨率的关系

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

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

[+] Feedback

调制器电容引脚

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

反馈电阻引脚

此参数设置引脚以连接外部反馈电阻 (R_b)。 从以下可用引脚选择： P1[1]、P1[5] 和 P3[1]。在某些设备封装中，某些引脚不可用。

提示：如果这些引脚中的部分引脚用于其他用途 （例如，用于传感器连接），则不在 UM 参数列表中提供选择。 CSD 用户模块的将来版本可允许使用附加引脚来连接反馈电阻。 此将允许在没有 P3 端口的封装中使用另一个 I^2C 端口。 使用引脚 P1[5] 或 P3[1] 是为了避免编程问题。

参考

此参数设置比较器参考源。 有关更多信息，请参见 “ 参考值 ”。

参考值

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

零是最小参考值 (1/4 Vdd)。 8 是最大值 (3/4Vdd)。 当参考值增大时，灵敏度下降，但是对屏蔽电极的影响增大。

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

预分频器周期

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

建议使用值 2^n-1 以获取最大信噪比 (SNR)：

1

3

7

15

31

63

127

255

其他值会导致更多噪声，在低分辨率和高扫描速度情况下尤其如此。

ShieldElectrodeOut

可以从其中一个备用数字行总线 (Row_0_Output_1 - Row_0_Output_3). 中选择屏蔽电极信号源。每行输出都可以路由到三个引脚当中的一个。 将行 LUT 功能设置为 A。

[+] Feedback

应用程序编程接口

应用程序编程接口 (API) 函数作为用户模块的一部分提供，使您能够以更高级别处理模块。 本节指定每个函数的接口，以及内置文件所提供的相关常量。

注意 ** 此种情况如同所有用户模块的 API，A 和 X 寄存器的值可以通过调用 API 函数来更改。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 此 “ 寄存器易失 ” 策略是针对提高效率的目的选择，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言编程人员也必须确保他们的代码符合此策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

提供了进入点以初始化 CSD、启动其采样和停止 CSD。 在所有情况下，模块的实例名称会替换在下列进入点中显示的 CSD 前缀。 未能使用正确的实例名称是造成语法错误的一个常见原因。

API 函数使用不同的全局阵列。 您不能手动改变这些数组。 但是，可以出于调试目的检查这些值。 例如，可以使用绘图工具显示阵列的内容。 下面是一些全局阵列：

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

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

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

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

- CSD_ScanSensor();
- CSD_ScanAllSensors().

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

CSD_baSnsOnMask[] - 这是一个保持传感器开或关状态的字节阵列 （对于按钮或滑条）。
CSD_baSnsOnMask[0] 包含传感器 0 到 7 的掩码位 （传感器 0 为 0 位，传感器 1 为 1 位）。
CSD_baSnsOnMask[1] 包含传感器 8 到 15 的掩码位 （如果需要），依此类推。 该字节数组包含囊括所有布置的传感器所需的元素。 如果按钮开启，则位的值为 1；如果按钮关闭，则位的值为 0。
CSD_baSnsOnMask[] 数据通过 CSD_bIsSensorActive(BYTE bSensor) 函数或 CSD_bIsAnySensorActive() 例程更新。

CSD_Start

说明：

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

C 原型：

```
void  CSD_Start()
```

汇编：

```
lcall  CSD_Start
```


[+] Feedback

参数：

无

返回值：

无

副作用：

CSD_Stop

说明：

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

C 原型：

```
void  CSD_Stop()
```

汇编：

```
lcall  CSD_Stop
```

参数：

无

返回值：

无

副作用：

CSD_ScanSensor

说明：

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

C 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

副作用

[+] Feedback

CSD_ScanAllSensors

说明：

通过为每个传感器索引调用 CSD_ScanSensor() 扫描所有配置的传感器。

C 原型：

```
void CSD_ScanAllSensors()
```

汇编：

```
lcall CSD_ScanAllSensors
```

参数：

无

返回值：

无

副作用

CSD_UpdateSensorBaseline

说明：

为每个传感器单独计算的历史数据称为传感器的基准线。 该基准线值使用 “ 水桶方法 ” 进行更新。

桶形电极方法使用下列算法：

1. 每次调用 CSD_UpdateSensorBaseline() 时，通过从原始数值中减去前一个基准线值，得到差值。此差值存储在 CSD_waSnsDiff[] 阵列中向您提供。

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

3. 虚拟水桶中的累计差值计数达到 BaselineUpdateThreshold 后，基准线按 1 递增，水桶复位为 0。

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

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

副作用：

[+] Feedback

CSD_UpdateAllBaselines

说明：

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

C 原型：

```
void  CSD_UpdateAllBaselines()
```

汇编：

```
lcall CSD_UpdateAllBaselines
```

参数：

无

返回值：

无

副作用：

CSD_bIsSensorActive

说明：

检查指定传感器（上的电容变化量）与其手指阈值相比的差值。 其中会考虑 “ 迟滞 ”。 根据传感器当前是否开启，对手指阈值加减迟滞值。 如果传感器处于活动状态，则降低阈值。 如果处于非活动状态，则升高阈值。 此函数还更新传感器 CSD_baSnsOnMask[] 阵列中的位。

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：

如果处于活动状态，则返回值为 1，如果处于非活动状态，则返回值为 0。

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

副作用：

CSD_bIsAnySensorActive

说明：

检查所有传感器（采到的变化量）与其手指阈值相比的差值。 为每个传感器调用 CSD_bIsSensorActive() 以便在调用此函数之后使 CSD_baSnsOnMask[] 数组保持最新状态。

C 原型：

```
BYTE  CSD_bIsAnySensorActive()
```


[+] Feedback

汇编：

```
lcall CSD_bIsAnySensorActive
```

参数：

无

返回值：

如果处于活动状态，则返回值为 1，如果处于非活动状态，则返回值为 0。

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

副作用：

CSD_wGetCentroidPos

说明：

检查差值数组的中心。 如果存在，则偏移和长度存储在临时变量中，根据 CSD 向导中指定的分辨率计算质心位置。 只有当滑条是由 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

此参数可引用作为滑条的一组特定的传感器。 组 0 用于按钮。 滑条包含在组 1 及更高编号的组中。

返回值：

滑条的位置值，LSB 位于 A 中，MSB 位于 X 中。

副作用：

该例程通过减去噪音阈值来修改差值。 该例程只应在每次扫描后调用一次以避免得到负差值。 如果您的应用要监控差值信号，请在传输差值数据之后调用此例程。

如果有滑条传感器处于活动状态，则该函数将值从零恢复为 CSD 向导中设置的分辨率值。 如果没有传感器处于活动状态，则该函数返回 -1 (FFFFh)。 如果在执行质心 / 双工算法时出现错误，则该函数返回 -1 (FFFFh)。 如果需要，可以使用 CSD_blsSensorActive() 例程确定触摸了哪些滑条段。

注意：如果滑条段的噪声计数大于噪声阈值，则此子程序可能生成假的质心结果。 噪音阈值应小心设置 （高出噪音水平足够程度），以确保噪音不会生成错误的中心位置。

CSD_wGetRadialPos

说明：

检查差值数组的中心。 如果存在，则根据 CSD 向导中指定的分辨率计算质心位置。 此函数仅用于 CSD 向导定义的辐射滑条。

C 原型：

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


[+] Feedback

汇编：

```
mov  A,  bSnsGroup
lcall CSD_wGetRadialPos
```

参数：

bSnsGroup A => 组编号

此参数是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：对于 s2，辐射滑条编号为 2）。

返回值：

辐射滑条的位置数值、A 中的 LSB 和 X 中的 MSB。

副作用：

此例程在每次扫描后只能调用一次，以避免得到负的差值和基准更新。 如果您的应用要监控差值信号，请在传输差值数据之后调用此例程。

如果有滑条传感器处于活动状态，则该函数将值从零恢复为 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，则该函数返回 -1 (FFFFh)。

注意：如果滑条段的噪声计数大于噪声阈值，则此子程序可能生成假的质心结果。 噪音阈值应小心设置 （高出噪音水平足够程度），以确保噪音不会生成错误的中心位置。

CSD_wGetRadialInc

说明：

返回实际的手指移动情况、手指当前位置与先前位置的差值。 此函数与 CSD_wGetRadialPos() 成对使用，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
lcall CSD_wGetRadialInc
```

参数：

bSnsGroup A => 组编号

此参数是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：

手指移动数值 （顺时针为正，逆时针为负）、A 中的 LSB 和 X 中的 MSB。

手指移动数值是手指当前位置与先前位置之间的差值。 如果在先前的扫描期间未发生触摸 （倒数第二次 CSD_wGetRadialPos() 返回 -1 (FFFFh)）或者当前没有任何触摸（此时 CSD_wGetRadialPos() 返回 -1 (FFFFh)）

副作用：

只能在 CSD_wGetRadialPos() API 之后调用该例程。 因为它使用由 CSD_wGetRadialPos(). 设置的内部数据 CSD_waSliderPrevPos 和 CSD_waSliderCurrPos

[+] Feedback

CSD_InitializeSensorBaseline

说明：

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

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

副作用：

CSD_InitializeBaselines

说明：

通过扫描每个传感器为 CSD_waSnsBaseline[] 数组加载初始值。 原始计数值将复制到每个传感器的基准阵列中。

C 原型：

```
void CSD_InitializeBaselines()
```

汇编：

```
lcall CSD_InitializeBaselines
```

参数：

无

返回值：

无

副作用：

CSD_SetDefaultFingerThresholds

说明：

为 CSD_baBtnFThreshold[] 数组加载 FingerThreshold 参数值。 如果没有为 CSD_baBtnFThreshold[] 数组手动加载定制值，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

副作用：

CSD_SetScanMode

说明：

设置扫描速度和分辨率。 此函数可以在运行时调用来更改扫描速度和分辨率。 此函数覆盖用户模块参数设置。 当某些传感器需要用不同的扫描速度和分辨率进行扫描时 （例如：常用按钮和接近检测器），此函数非常有效。 常用按钮可以用 9-bit 分辨率和 300 µ? 扫描时间进行扫描。 对于长距离的检测，接近检测器经常采用低于 16-bit 分辨率的值进行扫描，扫描时间大于 12 ms。 此函数可以与 CSD_ScanSensor() 函数一起使用。

C 原型：

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

汇编：

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

参数：

bSpeed

bResolution

返回值：

无

副作用：

CSD_SetRefValue

说明：

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

C 原型：

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

汇编：

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


[+] Feedback

参数：

bRefValue - 设置扫描参考值。 接受的值为 0..8。

返回值：

无

副作用：

CSD_ClearSensors

说明：

通过针对每个传感器按顺序调用 CSD_wGetPortPin() 和 CSD_DisableSensor()，将所有传感器清除为非采样状态。

C 原型：

```
void  CSD_ClearSensors()
```

汇编：

```
lcall  CSD_ClearSensors
```

参数：

无

返回值：

无

副作用：

CSD_wReadSensor

说明：

在 A (LSB) 和 X (MSB) 中返回关键原始扫描值。

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

副作用：

[+] Feedback

CSD_wGetPortPin

说明：

返回指定传感器的端口号和引脚掩码。 传递的参数对 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().

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总数与滑条中包括的传感器数量之和。 CSD_wGetPortPin() 使用传感器数量来确定所选活动传感器的端口和位掩码。

返回值：

A => 传感器位图

X => 端口号

副作用：

CSD_EnableSensor

说明：

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

C 原型：

```
void  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 => 端口号

返回值：

无

副作用：

[+] Feedback

CSD_DisableSensor

说明：

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

C 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 => 端口号

返回值：

无

副作用：

固件源代码示例

示例 1。 此代码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
```


[+] Feedback

```
}

    // 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。下面的代码演示了如何能够并行连接多个传感器并通过调用 CSD_ScanSensor() 函数同时扫描它们。 当您需要在未区分已触摸哪些传感器的情况下检测传感器触摸时，此示例非常有用。 可以使用示例进行设备唤醒检测和最大程度地减少扫描时间以节省电池能量。 如果检测到唤醒触摸，则可以分别将每个传感器返回到常规扫描。

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

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

void main(void)
{
     M8C_EnableGInt;

     CSD_Start();
     CSD_SetDefaultFingerThresholds();

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

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

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


[+] Feedback

示例 3。 下面的示例演示如何能够使用 CSD_SetScanMode() 函数，用不同的扫描参数扫描不同的传感器。当需要执行按钮触摸检测和接近检测时非常有用。 扫描按钮时采用低分辨率以减少扫描时间，扫描接近情况时采用较高分辨率以获取最大灵敏度。 您可以调整此代码，以便降低扫描接近情况的频率，只有当不进行按钮触摸检测时才扫描接近情况。

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

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

void main(void)
{

     M8C_EnableGInt;

     CSD_Start();
     CSD_SetDefaultFingerThresholds();

     // Set UltraFast, 9-bit resolution mode for baseline calculations
     CSD_SetScanMode(0, 9);

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

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

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

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

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


[+] Feedback

```
            }
      }
}
```

示例 4。 下面的示例演示如何能够为每个传感器设置不同的手指阈值级别。 当多个传感器放置在不同位置上而其中一些传感器比其他更灵敏时，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 process button presses
}
}
}
```


[+] Feedback

配置寄存器

Table 7. 模块 CMP，寄存器： ACE_CR1

位	7	6	5	4	3	2	1	0
值	0	1	0	0	1	1	1	1

Table 8. 模块 CMP，寄存器： ACE_CR2

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	电源

电源： 0x01 打开模拟模块的电源。 0x00 关闭模拟模块的电源。

Table 9. 模块脉冲宽度调制，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 10. 模块脉冲宽度调制，寄存器： 输入

位	7	6	5	4	3	2	1	0
值	0	0	0	1	1	1	0	0

数据输入高、0x10。 选择来自振荡器输出 （通过全局总线进行路由的比较器总线）的时钟输入。

Table 11. 模块脉冲宽度调制，寄存器： 输出

位	7	6	5	4	3	2	1	0
值	0	1	0	0	0	1	0	0

Table 12. 模块脉冲宽度调制，寄存器： 周期

位	7	6	5	4	3	2	1	0
值	0	0	0	0	1	1	1	1

脉冲宽度调制除以 16，启用计数时间和计数后处理时间间隔。

Table 13. 模块脉冲宽度调制，寄存器： 比较

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

脉冲宽度调制对 “ 比较 ” 值确定的时间进行计数。 采样开始时，当读取和处理前一计数的数据时，禁用计数。

Table 14. 模块 Counter16_LSB，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	1

[+] Feedback

Table 15. 模块 Counter16_LSB，寄存器： 输入

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

数据 = 0x80 (Row_output_0)，时钟 = 0x0X （SysClk 指令，位于输出寄存器中）。

Table 16. 模块 Counter16_LSB，寄存器： 输出

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

输入时钟 = 使用 SysClk 指令。

Table 17. 模块 Counter16_LSB，寄存器： Data2

位	7	6	5	4	3	2	1	0
值	数据输出 LSB							

Table 18. 模块 Counter16_MSB，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 19. 模块 Counter16_MSB，寄存器： 输入

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

数据输入链接自 LSB、0xC0。

周期方法： 输入时钟 = 0x00、SysClk 指令。

Table 20. 模块 Counter16_MSB，寄存器： 输出

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

输入时钟 = SysClk 指令。

Table 21. 模块 Counter16_MSB，寄存器： Data2

位	7	6	5	4	3	2	1	0
值	数据输出 MSB							

[+] Feedback

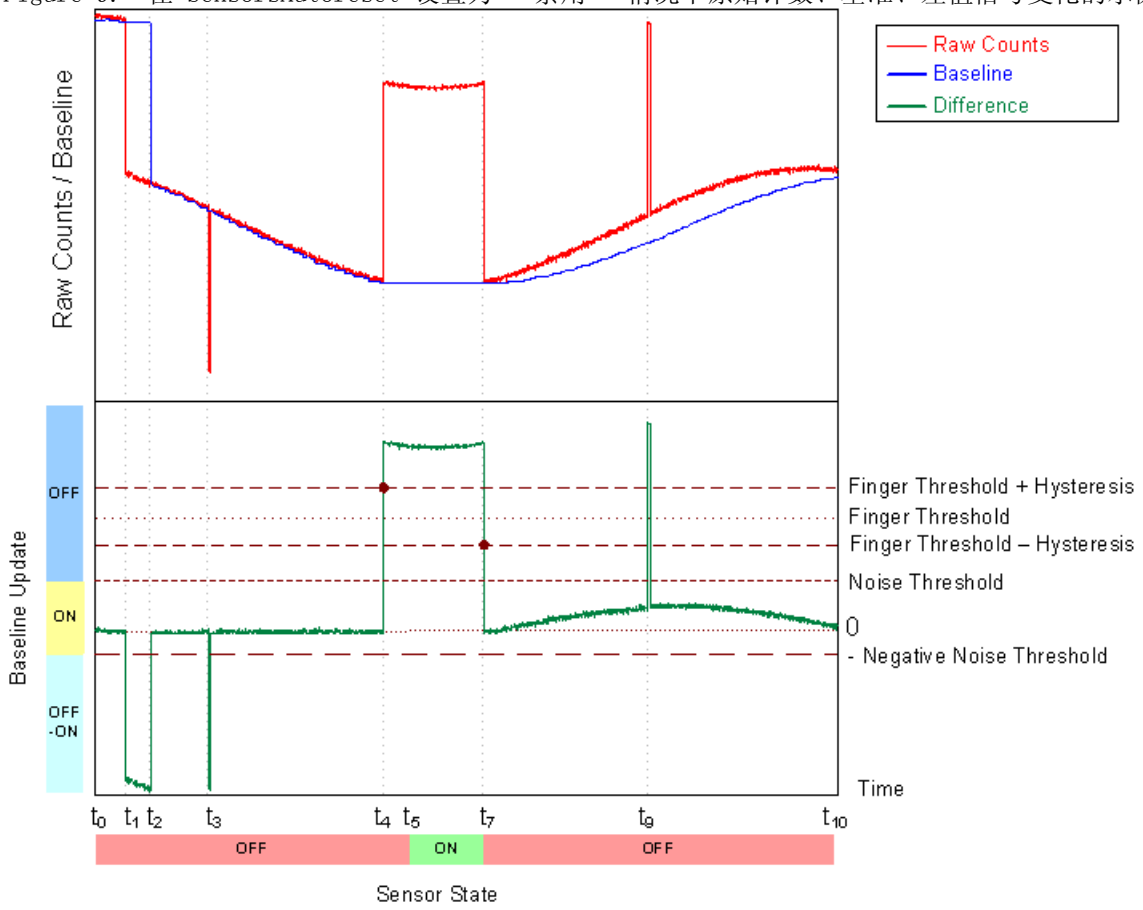
附录

以下部分介绍用户模块数据手册中通常没有包含的信息。 赛普拉斯工程师开发了详细信息来帮助您成功设计 CapSense 应用程序。 此信息的某些部分将来会移动到应用笔记中。

CSD 参数的交互

下图说明了基准线更新和决策逻辑操作。 这有助于更好地了解如何设置用户模块参数以获得最佳性能。 第一个图说明传感器自动复位参数设为**禁用**时的系统操作。 第二个图说明传感器自动复位参数设为**启用**时的系统操作。 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）。 数据通过一些人工测试收集，这些测试展示在低速和高速原始计数变化情况下的系统操作。 低速变化可由温度或湿度变化引起，高速变化可由传感器触摸、ESD 事件或强射频场的影响引起。

Figure 9. 在 SensorsAutoreset 设置为 “ 禁用 ” 情况下原始计数、基准、差值信号变化的示例



在 t_0 处，原始计数接近基线水平，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 NegativeNoiseThreshold 参数 （绝对值），因此通过跟踪原始计数最小值来更新基准，保留原始计数信号的较小值。

在 t_1 处，原始计数急剧下降，且负差值超过 “ 负噪声阀值 ”。 如果手指位于传感器上时设备加电，过一段时间后手指移开，则会发生这种情况。 此时，基准更新机制冻结，激活内部超时计数器。 当差值信号低于 LowBaselineReset 示例的 NegativeNoiseThreshold 时，基准复位。 这是在 t_2 发生的。

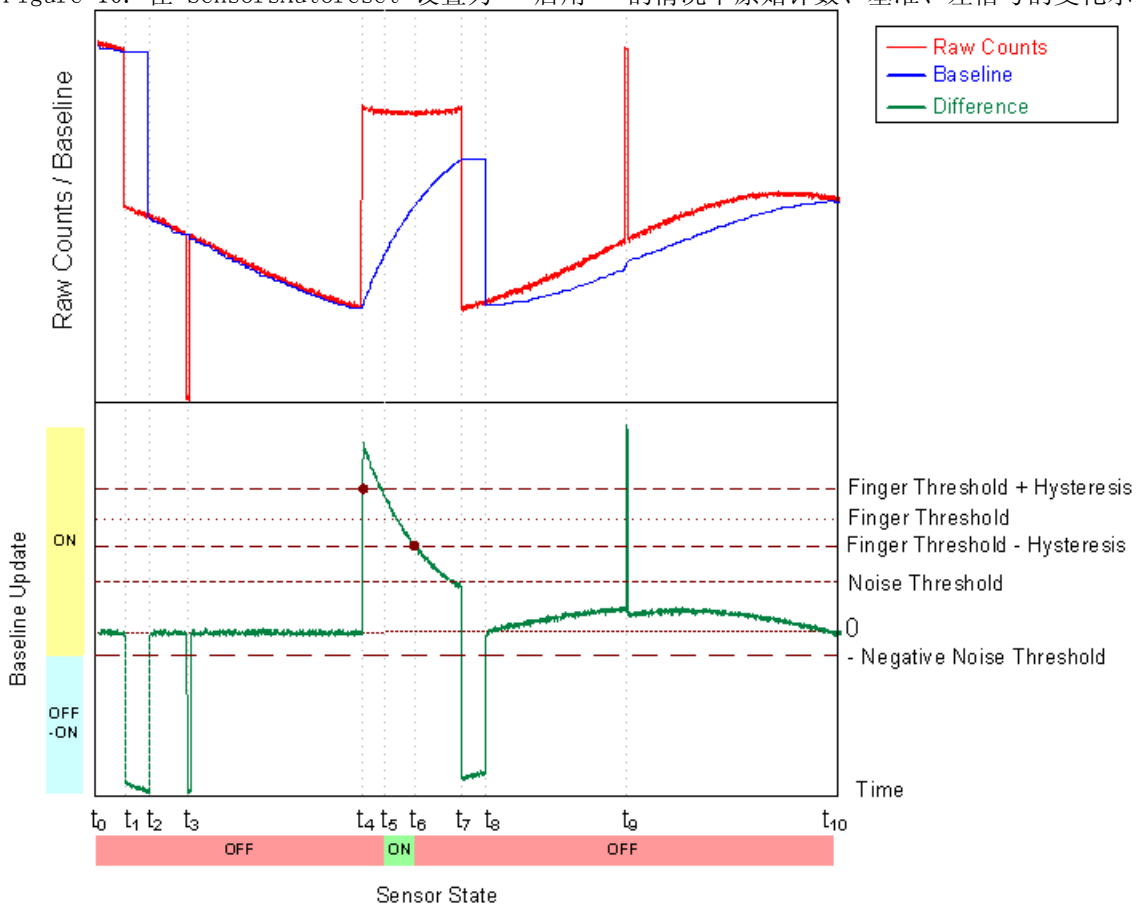
第二大负差信号尖峰位于 t_3 处。 这一尖峰可能是由 ESD 事件等引起的。 由于计数示例中的尖峰脉冲持续时间小于 LowBaselineReset 参数，因此保留基准，对尖峰脉冲进行滤波。 这可以阻止假基准复位和导致假触摸检测。

[+] Feedback

传感器是在 t_4 上被触摸的。 当差值信号超过 “ 手指阈值 + 迟滞 ” 值时，激活内部防反跳计数器。 如果信号超过此值的量大于防反跳示例，则传感器状态设置为启用。 这是在 t_5 发生的。 当差值信号在 t_7 下降到 “ 手指阈值 - 迟滞 ” 级别之下时，传感器立即恢复为关闭状态。 由于样本单位中的尖峰宽度没有超过去抖动值，因此 t_9 处短的正尖峰被去抖动计数器过滤。

原始计数在 t_7 与 t_{10} 之间缓慢升高。 当差信号低于 NoiseThreshold（SensorsAutoreset 设置为 “ 禁用 ”），差信号与漂移速率成比例时，使用水桶算法来更新基准。 可以使用 BaselineUpdate 阈值参数来控制基准更新速度。 参数值越低，基准更新速度越快。

Figure 10. 在 SensorsAutoreset 设置为 “ 启用 ” 的情况下原始计数、基准、差信号的变化示例



[+] Feedback

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 处传感器被触摸，根据活动基准更新算法，触摸持续时间下降。
- 手指移开后，在 LowBaselineReset 样品（t_8）模块触摸检测很短时间后，基准复位。 这称为防反跳机制。

分步调校指南

电容式传感的成功取决于是否为给定传感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 设备的电极连接阻抗
- 最终应用状况，例如：
  - 电源的提供
  - 温度
  - 湿度
  - 潮气的存在情况
  - ESD、EMC 或 EMI 要求

在单独的应用文件中，介绍了不同任务的最佳做法（防水操作、使用高阻抗材料的传感、接近检测和通过厚外覆层的操作以及通过认证测试的建议）。

下面以 CY3212/CY3213 板作为测试示例，介绍了在典型 CapSense 应用程序中配置用户模块的基本指南。 传感区覆盖有 2 mm 厚的塑料外覆层。 请按下列步骤配置 CSD 用户模块参数：

1. 准备目标板。 组装目标应用 PCB ，在它上面安装外覆层。 使用胶水或特殊胶带来进行安装。 避免在 PCB 与外覆层之间留有气隙，否则会极大降低灵敏度，且由于当您触摸时气隙会漂移，导致出现许多假按钮触发情况。
2. 设置用于监控数据的实时监控工具。 在 CSD 配置期间，使用 PC 绘图工具能够实时观察一个或多个数据系列。 在用户模块调校过程中，必须注意原始数、基线和信号差。 为此，可以使用 I^2C-USB 桥。 在测试中，已经使用一个 ^2C-USB 桥来监控原始计数数据。 请勿使用 LCD 或任何其他数字显示屏来监控计数，因为其速度较慢，无法帮助您观察到数据的动态变化。

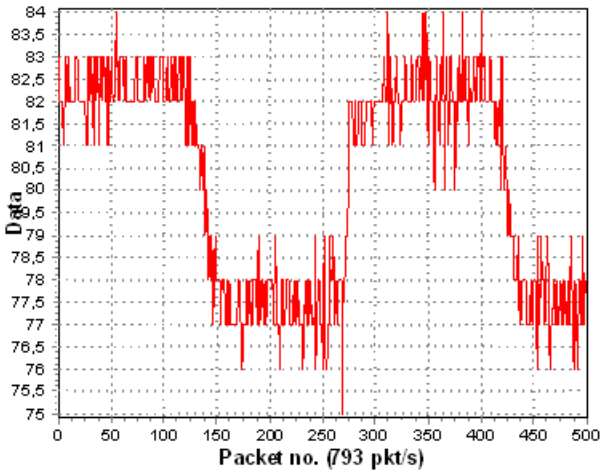
[+] Feedback

3. 设置初始配置。 此配置使用不带预分频器的 16-bit PRS。 在开始测试之前，在 PSoC Designer 中设置下列参数：

Properties - CSD_1

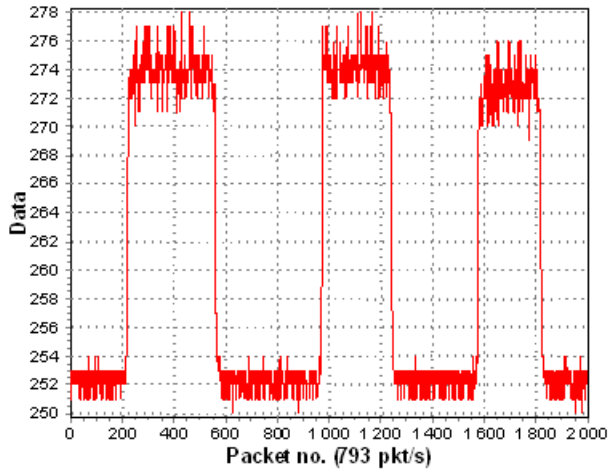
Name	CSD_1
User Module	CSD
Version	1.3
FingerThreshold	40
NoiseThreshold	20
BaselineUpdateThreshold	200
Sensors Autoreset	Enabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	20
LowBaselineReset	10
Scanning Speed	Ultra Fast
Resolution	9
Modulator Capacitor Pin	P0[1]
Feedback Resistor Pin	P1[5]
Reference	ASE11
Ref Value	2
ShieldElectrodeOut	None

4. 在 CSD 向导中分配传感器引脚 （分配传感器 P13、P31 和 P33 来进行扫描）。
5. 生成应用程序和示例代码。
6. 使用绘图工具监控传感器原始数数据，以确认用户模块可操作。 触摸传感器应导致原始数（CSD_waSnsResult 变量）从 77 更改为 82。

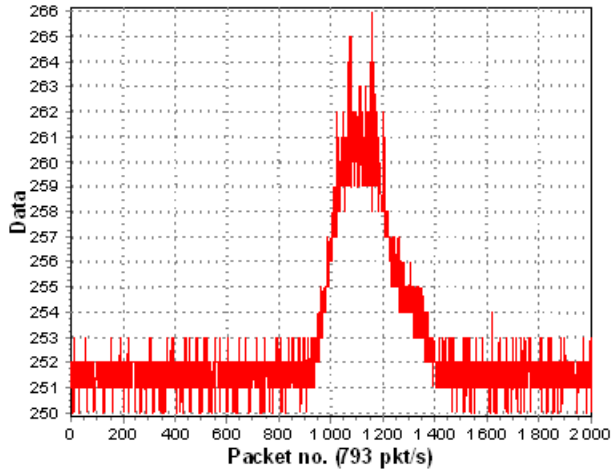


7. 调校外部组件。 赛普拉斯最初使用 5.6 nF 调制器电容 (C_{mod}) 和 1.6 kΩ 反馈电阻 R_b 。 在触摸情况下对来自不同传感器的原始数值进行观察后，赛普拉斯找到了生成最大原始数值的传感器。 上图显示了来自此传感器的信号。 下面的信号值对应于无手指触摸的情况，上面的信号值对应于触摸情况。 通过分析来自此传感器的信号，可以看到系统仅使用电容到代码转换器动态范围的 16%。 9-bit 分辨率的整个范围为 Nm = 512，最大原始数大约为 85。 这意味着通过将反馈电阻值增加到 5.1 kΩ? 可以将动态范围利用率提高到建议的 60-70%。 可以使用不同的电阻值来完成此任务，具体取决于您的原始数观察情况。 下面的手指响应是更换电阻后的结果。 增加了来自手指触摸的响应。

[+] Feedback

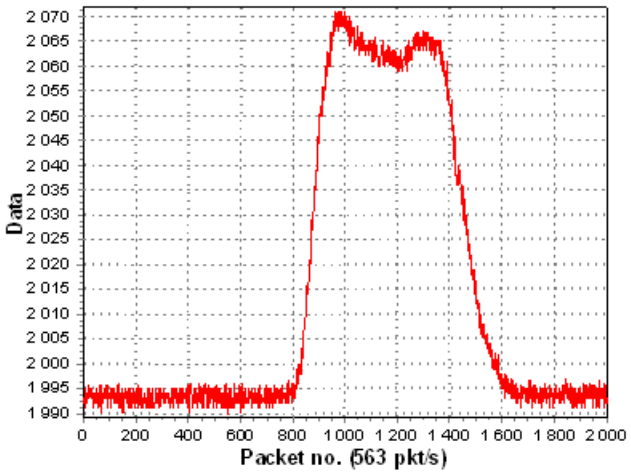


8. 对最坏情况进行调整。 使用手指模拟器确保设备在不同的情况 （例如很轻的触摸）下可靠工作。 将 10 mm 未连接线圈放在外覆层上，模拟最坏情况。 使用绝缘物体 （如火柴或牙签）将线圈移过按钮。 下图显示了结果。 如果您的板在传感器周围使用了接地层，则可以运行此测试。 如果该板被屏蔽电极而不是接地层覆盖，您可以通过用手指很轻触摸来模拟最坏情况。



[+] Feedback

9. 来自线圈的信号被识别出来，但是信噪比太小，不足以进行可靠检测。 差大约为 8 dB。 要增加灵敏度，请选择较高扫描分辨率。 在测试中，分辨率从 9 位增加到 12 位。 下面是在这些设置情况下来自线圈的信号。



10. 将扫描分辨率从 9 位增加到 12 位可以将信噪比提高到 23 dB，这对于获得最实际的应用很有帮助。来自人的手指的信号非常大。

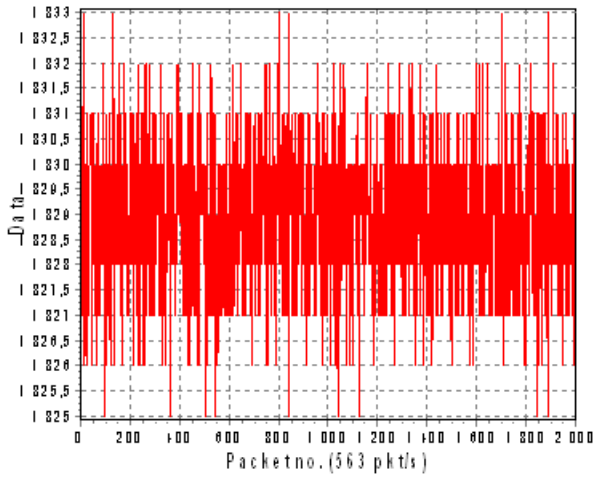
11. 设置阈值。 对用户模块参数进行以下更改：

Properties - CSD_1

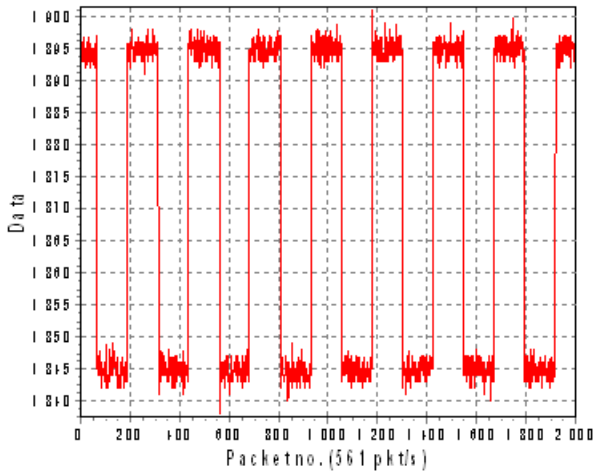
Name	CSD_1
User Module	CSD
Version	1.3
FingerThreshold	40
NoiseThreshold	20
BaselineUpdateThreshold	200
Sensors Autoreset	Enabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	20
LowBaselineReset	10
Scanning Speed	Ultra Fast
Resolution	12
Modulator Capacitor Pin	P0[1]
Feedback Resistor Pin	P1[5]
Reference	ASE11
Ref Value	2
ShieldElectrodeOut	None

12. 设置最佳扫描速度。 假设测试应用电源电压稳压不良，由于目标设备其他部分的运行，可能产生 ±5% 的快速电源波动。 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的原始数变化。 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。 下图显示了在扫描处于活动的情况下切换 LED 时的原始数。

[+] Feedback

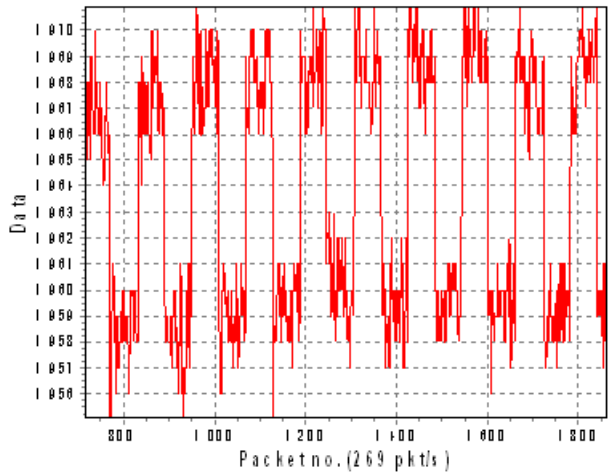


13. 如图所示，扫描处于活动情况下的 LED 开启 / 关闭对于原始数值没有明显的影响。 测试电源快速变化时 CapSense 的稳定性。 很慢的电源变化由基线更新算法来处理，在大多数情况下不会产生问题。 此测试中使用了 LM1117-ADJ 电压稳压器。 输出电压是通过反馈电阻网络变化进行调制的，该变化利用由外部信号源驱动的 MOSFET 来产生。 下图显示了当电源在 4.75V 到 5.25V 之间振荡时传感器的原始数差。



[+] Feedback

14. 如图所示，电源瞬变原始数变化（50）与阈值（35..45）接近，会导致假触摸检测。 解决方法是使用较慢的扫描速度。 下图显示了以正常扫描速度收集的原始计数数据：



15. 如图所示，降低扫描速度会降低电源电压变化对原始数的影响。 现在，瞬变差大约为 12 次。 这远低于阈值，对 CapSense 模块操作没有不利影响。
16. 调校 BaselineUpdateThreshold 参数 . 应用程序要求最大触摸检测时间少于 1 秒。 将 SensorsAutoreset 参数设置为 “ 启用 ”。 请检查 BaselineUpdateThreshold （基准线更新阈值）是否提供了适当补偿环境变化的基准线更新速度。 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，温度变化造成原始数下降。 通过将基线自动复位成原始数值，基线对此情况进行跟踪。 因此，在大多数情况下，由于环境因素造成的原始计数下降应当不是问题。 如果原始数由于温度变化而增加，可能将此变化解释为触摸而触发假触摸。 我们需要调整基线更新速度，以便温度 （或其他环境因素）对原始数与基线之差的影响远低于 “ 手指阈值 ” 值。 原始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，此参数设置为最小值 5。 这意味着预设的 BaselineUpdate 阈值参数提供了足够的基线跟踪速度，对于我们的应用程序而言，温度波动不应当是问题。
17. 设置完所有参数后，可以运行 ESD 测试。 即使 “ESD 防反跳 ” 参数设置为 “ 禁用 ”，您的应用程序也能通过这些测试，不会产生问题。 如果需要，您可以在 ESD 测试出现问题时启用 “ESD 防反跳 ” 参数。 启用此参数的代价是 RAM 缓冲区的大小增加。
18. 许多 CapSense 应用程序被要求通过各种 EMC/EMI 兼容性测试。 如果您的应用程序在 EMC/EMI 方面遇到问题，应用笔记 “*PSoC CapSense 应用程序的 EMC 设计注意事项* ”AN2318 中提供了用于解决问题的信息。 解决此问题的其他可行方法是使用较慢的 PRS 时钟减少传感器路径辐射。 您可以尝试带有预分频器的配置，或者使用较慢的 IMO 模块 （例如，SYSCLK 在 6 MHz 而不是 24 MHz 运行）。 如果 PRS 时钟频率或预分频器周期设置有任何更改，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。
19. 如果您的应用程序未通过 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，因而获得较高的抗噪声能力。 但代价是增加了传感器扫描时间。

[+] Feedback

故障排除

- 如果您在原始传感器数中看到奇怪的信号变化尖峰脉冲，它们是 256 的倍数，请降低扫描速度。 如果此操作奏效，则问题原因是缺少计数器溢出中断。 分析中断持续时间，并对它们进行优化。 从较低优先级中断重新启用全局中断。 有关详细信息，请参见下一节有关中断持续时间管理的内容。
- 您可以将预充电预分频器用作 UART 波特率时钟源。 建议的 UART 速度不应当小于 115,200 波特。 对于 24 MHz IMO 操作，预分频器周期应当设置为 25。 由于此值不是 2^N 的倍数，建议使用较慢的扫描速度来获得较高的信噪比。 请通过实验来测试。
- 如果从 CSD 开始操作，请选择模拟调制器 (ASE11) 参考。 检查原始数信噪比。 切换到带隙以进行比较。 即使在高分辨率情况下，噪声也不应大于 ±5..±7 LSB。
- 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。 此延时在测量开始之前设置调制器启动时间。 减少调制器电容 C_{mod} 也会有帮助。 产生此噪声的原因是：由于内部模拟调制器低通滤波器上的时间常数很小，在以前的测量周期中调制器电容充电为另一个电压。
- 当预分频器频率小于 100 kHz 时，由于内部低通滤波器上有大的波纹，不建议设置对模拟调制器 (ASE11) 的参考。 请转而使用带隙参考 (VBG) 或外部滤波的参考 (AnalogColumn_InputSelect1)。
- 带有预分频器的 PRS 配置为低分辨率提供了较快的扫描时间。 如果需要很快的扫描时间，请选择此配置。
- 扫描速度和分辨率影响信噪比。 较慢扫描速度和较高分辨率可获得更好的信噪比。
- 如果电极外覆层很厚，可能需要较高分辨率和较慢扫描速度。
- 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品转换周期计数。 较慢的扫描速率和较高的分辨率会产生较长的 PRS 序列，因而在 EMC 测试期间可提供更好的抗噪能力。
- 扫描速度越慢，调制器运行频率越低，读数对比较器动态特性的依赖度也越低。 如果您需要在电源波动或 PSoC 设备控制高电流负载的情况下获得良好的原始数稳定性，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 “ 正常 ” 或 “ 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 毫秒时间进行传感器扫描，可以获得最佳结果。
- 有效地使用屏蔽电极，可减少杂散电容的影响，即使在不防水的传感应用场合下。 对于这种情况，屏蔽电极可以安装于电容式传感器区域下 PCB 底层的上方。 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

小心不要更改此用户模块使用的硬件配置。 其中包括：

- 内部使用的 GlobalOutEven_0 和 Row_0_Output_0 总线。 不要将任何资源与这些总线相连。
- 不要更改 Row_0_Output_0 总线输出的 LUT 功能。 应选择为 **A**。
- 不要更改比较器总线的 LUT 功能。 比较器 Bus_0 LUT 功能应设置为 **B**，比较器 Bus_1 LUT 应设置为 **~A**。
- 模拟模块时钟源应设置为 **VC1**。
- VC1、VC2、VC3 分频器和 VC3 源可以通过用户模块在内部设置。 全局资源中输入的值在运行时会被覆盖。
- 使用屏蔽电极时，将行 LUT 功能设置为 **A**。

[+] Feedback

中断持续时间管理

当运行传感器扫描时，小心管理 “ 中断服务例程 (ISR)” 的持续时间。 8-bit 计数器的计时直接来自 VC1。 最坏情况下的 VC1 溢出时间间隔为：

Equation 3

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

F_{IMO} - IMO 频率，VC1=1,2,4,8 分别针对超快、快速、正常、慢速扫描速度。

需要特别注意异步通信例程 （例如 I2C、SPI、UART）和睡眠定时器中断。 确保其短于根据等式 3 估计的最坏溢出情况的时间。 从低优先级中断 （例如睡眠定时器和 I^2C）中重新启用全局中断，可以避免丢失计数器溢出中断。 在某些情况下，应对存在问题的中断服务例程源代码进行优化 （例如：I2CHW 和 EZI2C 存在较长的中断处理程序）。 其他中断应当通过调用 M8C_EnableGInt 宏从 I2CHW 中断中重新启用。

可能的 ISSP 引脚冲突

将低阻抗反馈电阻与 P1[1] 引脚始终连接，将导致 ISSP 编程错误。 这种情况下应使用其他引脚。 如果可用，请使用封装中的 P3[1] 引脚。 CSD 用户模块的未来版本可能允许使用附加引脚，用于连接反馈电阻，从而允许使用其他的 I^2C 端口。

版本历史记录

版本	创作者	说明
1.4	DHA	分辨率最大值为 3000。 消除了 0.5 的移位，并添加了对负值的补偿。 向导提供固定引脚列表。
1.50	DHA	增加了对 CY8C21x12 设备的支持。

Note PSoC Designer 5.1 在所有的用户模块数据表中提供版本历史记录。 本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2010 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

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

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

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

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