Preliminary

自动调校式 CapSense® Sigma-Delta 数据表 CSDAUTO V 1.1

Copyright © 2009-2010 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块				API 存储器		引脚（每个外部 IO）
	CapSense	I²C/SPI	定时器	比较器	闪存	RAM	
CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CY7C645xx、CY7C643/4/5xx、CY7C60424、CY7C6053x、CYONS2110、CYONS21L1T 和 CYONSFN2162							
	1		1	1	1540	35	0

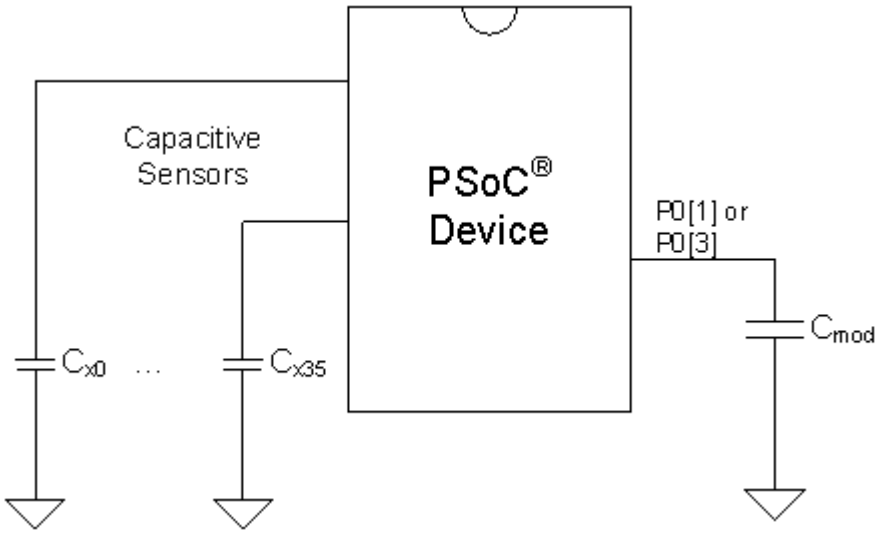
特性与概述

- 自动调校算法可在运行时根据每个传感器的寄生电容优化操作参数。
- 扫描 1 到 36 个电容式传感器。
- 只要遵循了应用笔记“电容式感测 – PSoC CapSense 布局指南” AN2292 中说明的布局准则，就能够以最大 50 pF 的寄生传感器电容 (Cp) 检测到 0.1 pF 的碰触。
- 感应能力最多可以穿透 15 mm 的玻璃外覆层。
- 对交流电源噪声、其他 EMI 和电源电压变化的抗干扰能力较强。
- 支持将电容式传感器配置为独立按键、接近传感器和 / 或作为相关阵列以形成滑条。 此测试版中并不支持滑条和接近传感器。
- 可使用双工法将滑条元件的有效数量设置为专用 IO 引脚数的两倍。
- 支持通过插值将滑条的分辨率提高到物理间距以上。
- 可按互相垂直的交织滑条实现触摸板。
- 利用屏蔽电极能够以较高的寄生电容和 / 或在有水膜的情况下进行可靠的操作。
- 使用 CSDAUTO 向导可实现引导式传感器和引脚分配。
- 支持利用 PC GUI 应用程序实时监控原始数据。

CSDAUTO（使用 Sigma-Delta 调制器自动调校式 CapSense®）用户模块可通过 Sigma-Delta 调制器使用开关电容技术将传感器电容转换为数字代码，从而进行电容式感测。

Note 该用户模块仅支持 C 语言项目， 不支持 ASM （汇编语言）项目。

Figure 1. CSDAUTO 典型应用



功能说明

电容式传感器由物理、电气和软件组件构成：

- 物理
  - 物理传感器本身在典型情况下是一个在连接到 PSoC 的印制电路板上制作的导电图形；这种印制电路板通常有一层绝缘罩、挠性薄膜或透明外覆层覆盖在显示屏上。
- 电气
  - 一种将传感器电容转换为数字格式的方法。 转换系统包括感应开关电容器、Sigma-Delta 调制器和基于计数器的数字过滤器，可将调制器输出位流转换为可读的数字格式。
- 软件
  - 检测和补偿软件算法可将计数值转换为传感器检测决策。
  - 在连续情况下，将会提供相关传感器 （例如滑条和触摸板）API，以插入分辨率比传感器的物理间距更高的位置。 例如，您可以使用 10 个传感器创建音量滑条，并使用所提供的固件将音量数字提高到 100。 或者，通过相同的 API，您也可以使用逐渐变细彼此插入对方的两个电容式传感器，并确定二者之间某个导电物体 （例如手指）的位置。
  - 高层级的决策逻辑可为温度、湿度和电源电压变化等环境因素提供补偿。 独立的屏蔽电极可用于屏蔽传感器阵列以降低杂散电容，从而在有水膜或水滴的情况下实现更可靠的操作。
  - 高层级的软件功能可适应滑条双工法，从而使单个 I/O 引脚能够连接到两个物理传感器上，以便将针对给定数量的滑条元件消耗的 IO 数减半。

在使用 CSDAUTO 用户模块实施 CapSense 设计之前，建议您阅读以下文档：

- *PSoC® CY8C20x66*、*CY8C20x66A*、*CY8C20x46/96*、*CY8C20x46A/96A*、*CY8C20x36*、*CY8C20x36A*、*CY8CTMG20x*、*CY8CTMG20xA*、*CY8CTST200 和 CY8CTST200A 技术参考手册* (TRM) 部分：
  - CapSense 系统

[+] Feedback

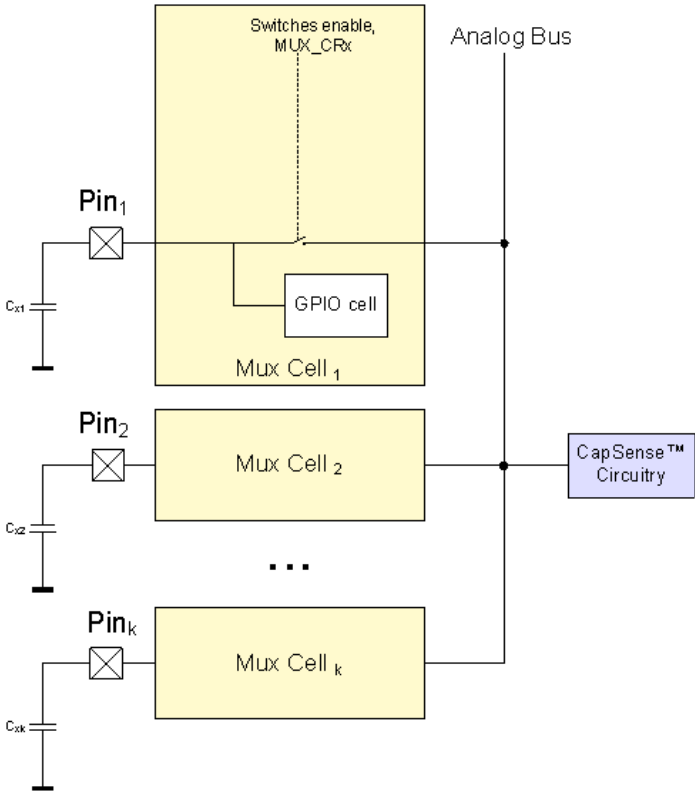
在阅读 CSD 用户模块文档之后，建议您阅读以下应用笔记。 要找到相应的应用笔记，请访问赛普拉斯半导体公司网站 www.cypress.com：

- *CapSense 最佳实践* – AN2394
- *CapSense 应用的信噪比要求* – AN2403
- *设计辅助 – CapSense 数据查看工具* – AN2397
- *PSoC CapSense 应用的 EMC 设计注意事项* – AN2318
- *电源和睡眠注意事项* – AN2360
- *PSoC CapSense 布局指南* – AN2292
- *通用异步发射器的软件实现* – AN2399
- *防水电容式感测* – AN2398

电容式感测操作

CY8C20x66 系列器件包括一个模拟复用器总线。 该总线允许将 CapSense 电路连接到任意 PSoC 引脚上。 CSDAUTO 用户模块可将活动传感器连接到模拟复用器总线，从而使始终连接的 CapSense 电路可以测量传感器电容，并将该电容值转换为数字代码。 固件通过在 MUX_CRx 寄存器中设置相应的位来连续执行传感器扫描。

Figure 2. CSDAUTO 框图



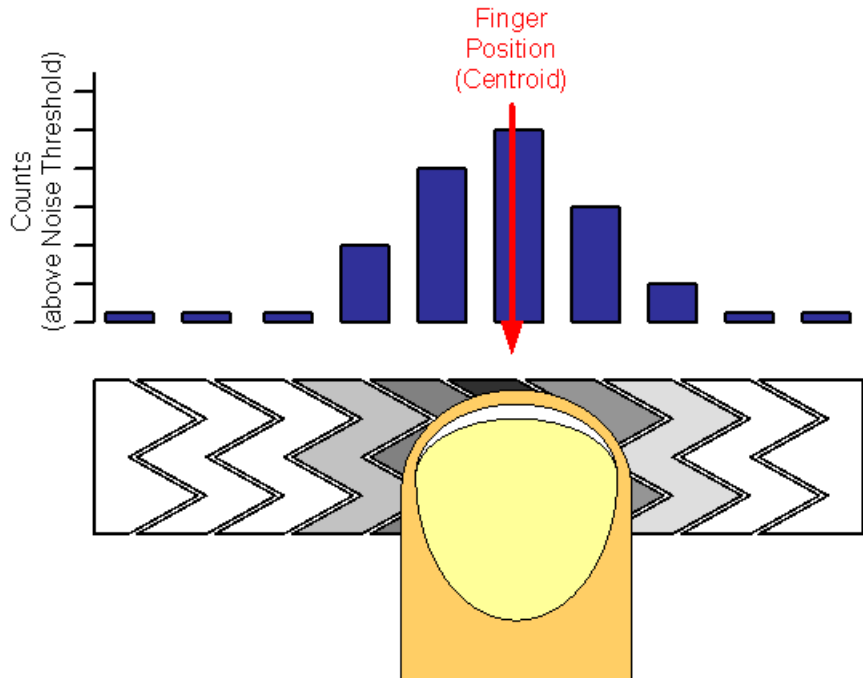
[+] Feedback

滑条

Note 此测试版中并不支持滑条和接近传感器。

滑条适用于需要渐进式调节的控件。 示例包括照明控件 （调光器）、音量控件、图形均衡器和速度控件。 构成滑条的传感器彼此相邻。 一个传感器的启动将致使物理位置上相邻的传感器部分启动。 通过计算活动传感器组的质心位置得出在滑条中的实际位置。 通过建立一些组 （每组滑条都有特定的顺序）可在 CSDAUTO 向导中对滑条进行调整。 传感器的实际下限数字是五，上限就是所选 PSoC 器件上可用的传感器位置数。

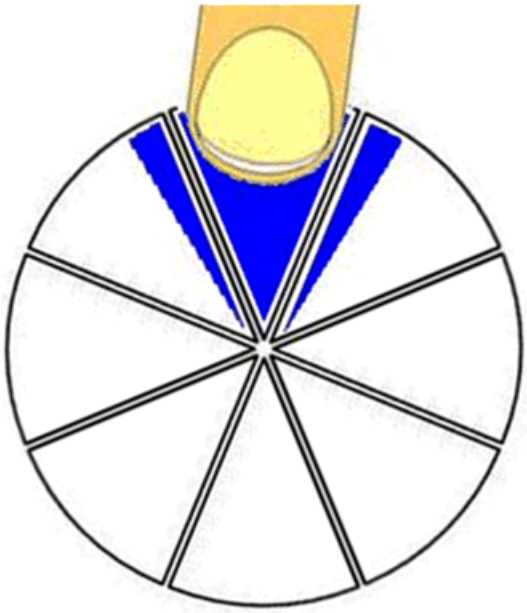
Figure 3. 滑条上手指的插值质心位置



[+] Feedback

径向滑条

Figure 4. 手指触碰径向滑条



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

CSDAUTO UM 包含两个支持径向滑条的 API 函数。 第一个函数 CSDAUTO_wGetRadiaPos() 返回质心位置，第二个函数 CSDAUTO_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。

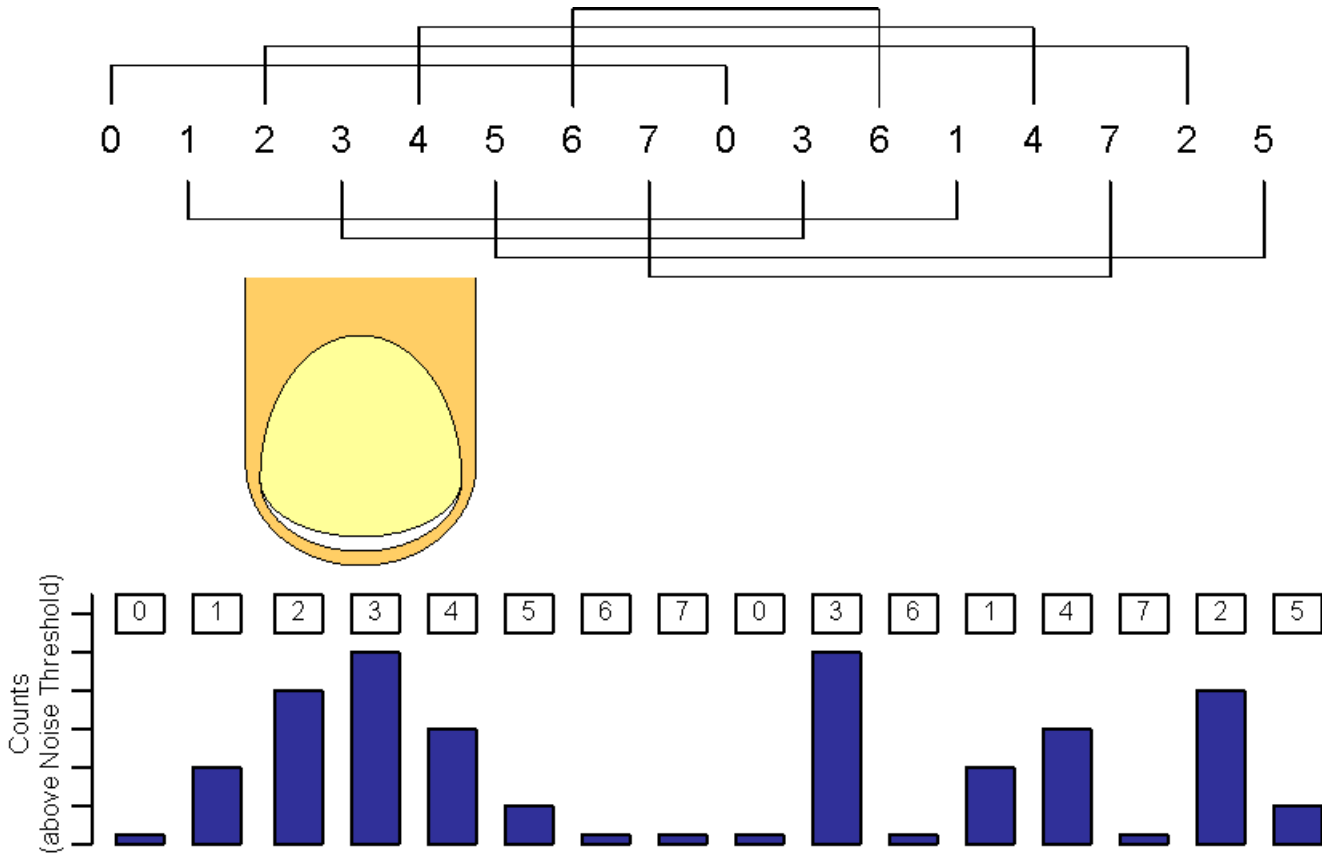
参考点 (0) 位于第一个传感器的中心。 线性和径向滑条的分辨率均限制为 3000。

[+] Feedback

双工法

使用双工法时，滑条中的每个 PSoC 传感器连接都会映射到滑条传感器阵列中的两个物理位置上。 物理位置的前 （或数值较小的）半部分将通过设计人员使用 CSDAUTO 向导分配的端口引脚按顺序映射到所分配的基准传感器上。 物理传感器位置的后 （或上）半部分将通过算法自动映射，如下图所示。

Figure 5. 由 CSDAUTO 进行的双工滑条阵列索引编制



滑条一半内强烈信号的靠近会导致相同的电平混叠到上半部，但结果是离散的。 感应算法通过搜索强烈的相邻信号组指明解析出的滑条位置。 确立顺序时要做到让相邻传感器在某一半的启动不会导致另一半的邻近传感器启动。

通过练习确保传感器与印制电路板引脚的映射符合用户模块所用的 3 倍索引模式。 双工滑条中传感器对的电容应该合理匹配 （不超过 10 pF）。

当您选择双工时，CSDAUTO 向导将自动生成双工传感器索引表。 此表说明了不同滑条段计数的双工序列：

Table 1. 不同滑条段计数的双工序列

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

[+] Feedback

滑条段总计数	段序列
22	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8
24	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11
26	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 3, 6, 9, 12, 1, 4, 7, 10, 2, 5, 8, 11
28	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 3, 6, 9, 12, 1, 4, 7, 10, 13, 2, 5, 8, 11
30	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 3, 6, 9, 12, 1, 4, 7, 10, 13, 2, 5, 8, 11, 14
32	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 2, 5, 8, 11, 14
34	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14
36	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 0, 3, 6, 9, 12, 15, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14, 17
38	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 2, 5, 8, 11 , 14, 17
40	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17
42	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19 , 2, 5, 8, 11, 14, 17, 20
44	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10, 13 , 16, 19, 2, 5, 8, 11, 14, 17, 20
46	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10 , 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20
48	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7 , 10, 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20, 23
50	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 0, 3, 6, 9, 12, 15, 18, 21, 24 , 1, 4, 7, 10, 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20, 23
52	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 0, 3, 6, 9, 12, 15, 18, 21 , 24, 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23
54	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 0, 3, 6, 9, 12, 15, 18 , 21, 24, 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23, 26
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

[+] Feedback

插值和定标

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

为了采用一个质心来计算插值后的位置，首先对阵列进行扫描以验证所给定的传感器位置是否有效。 这要求一定数量的相邻传感器信号大于某一噪声阈值。 在找到最为强烈的信号后，此信号和那些大于噪声阈值的邻近信号均用于计算质心。 按以下形式计算质心时，最少要用两个，最多要用八个 （典型情况下）传感器来计算质心位置：

Equation 1

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

计算得出的值通常是分数。 为了能够采用某一特定分辨率来报告质心 （例如对于 12 个传感器为 0 到 100 的范围），要将质心值乘以计算得出的标量。 将插值和定标操作结合到同一项计算内并在想要的标量内直接报告其结果，可以达到更高的效率。 这一过程可以在高层 API 内进行处理。

滑条传感器数量和分辨率均在 CSDAUTO 向导中设置。 向导将计算出标量数值并以分数值的形式进行存储。

质心分辨率的乘数包含在三个字节中，且具有以下位定义：

分辨率乘数最高有效位								
位	7	6	5	4	3	2	1	0
乘数	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
分辨率乘数中等有效位								
乘数	128	64	32	18	16	8	4	2
分辨率乘数最低有效位								
乘数	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256

使用以下公式计算分辨率：

分辨率 = （传感器数量 - 1）× 乘数

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

外部组件选择指南

CSDAUTO 可使用内部或外部调制电容 C_{mod}，由 Vss 接地连接到 P0[1] 或 P0[3] 端口引脚。 通过 “ 引脚选择向导 ” 设置可选择引脚。 所选引脚不得用于其他任何用途。

除非 CSDAUTO 可以兼容内部电容，否则建议使用外部电容。 外部调制电容的建议值为 2.2 nF。 可通过实验选择最佳电容值，以获得最大信噪比。 2.2 nF 的电容值在多数情况下都能产生良好结果。 应使用陶瓷电容。 温度电容系数并不重要。

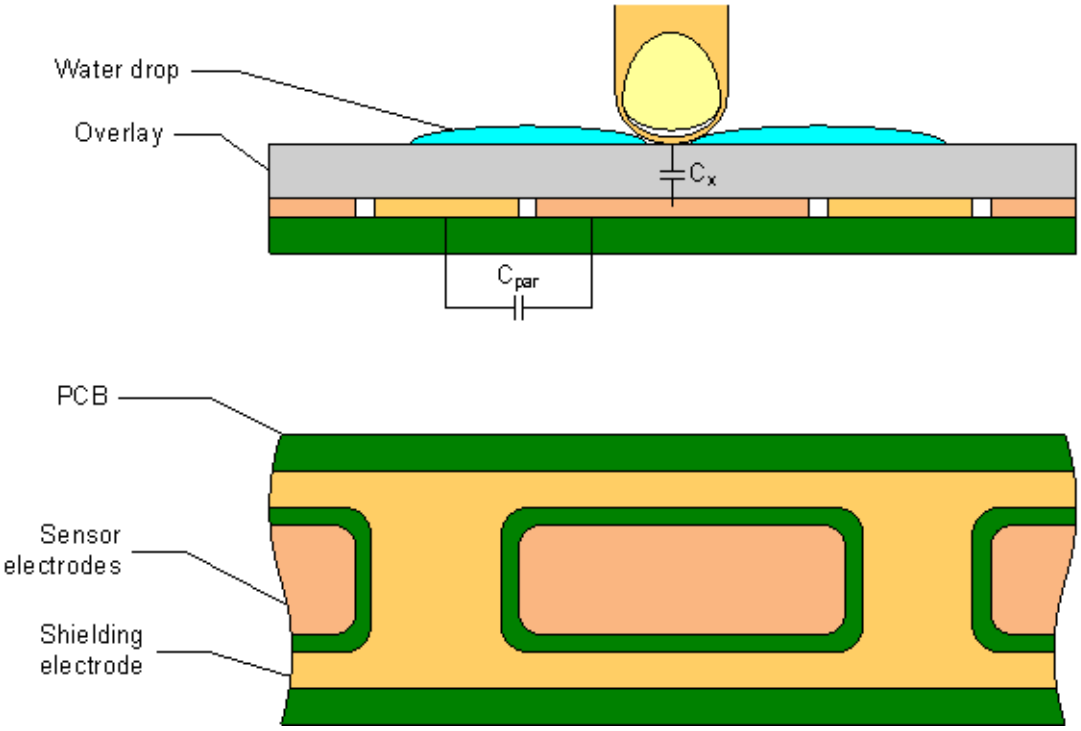
[+] Feedback

屏蔽电极

有些应用场合要求在有水膜或水滴的情况下也能进行可靠的操作。 白色家电、汽车应用、各种工业应用及其他领域需要电容式传感器不因为水、冰和湿度变化而出现误触发情况。 在这种情况下可以使用单独的屏蔽电极。 该电极位于感测电极的后方或外侧。 当绝缘外覆层表面出现水膜时，屏蔽电极和感测电极之间的耦合就会增强。 屏蔽电极有助于减小寄生电容的影响，从而使处理感测电容变化的范围更加动态。

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

Figure 6. 可能的屏蔽电极 PCB 布局



上图展示了可能为按键屏蔽电极采取的一种布局配置。 在本例中，按键上覆盖有一层屏蔽电极面。 也可将屏蔽电极置于 PCB 层的相对面，包括按键下方的面。 在这种情况下推荐使用填充样式，填充率约为 30% 至 40%。 这时无需额外的接地层。

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

CSDAUTO 使用与预充电时钟相同的信号来驱动屏蔽电极。

屏蔽电极可与专用 PSoC 引脚 （P0[7] 或 P1[2]）连接。 所选引脚的驱动模式应设为 “ 强 ”。 可在 PSoC 器件和屏蔽电极之间连接斜率限流电阻，以减少散发的电磁干扰。

[+] Feedback

直流和交流电气特性

Table 2. 电源要求

参数	最小值	典型值	最大值	单位	测试条件和注释
Vdd	1.7	5.0	5.25	V	不适用

Table 3. 各个 Cp 级别的信号和噪声

寄生电容，Cp（pF）	0.1 pF 触摸信号（计数）	噪声[a]（计数）	信噪比	扫描时间（µs）
10				
15				
20				

a. 此噪声值是在按照该文档相应指南作出的参考设计的基础上得出的典型值： AN2292

布置

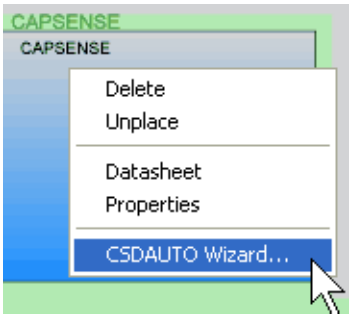
用户模块所用的模块将在用户模块实例化后自动布置，而没有提供备选布置方式。 CSDAUTO 用户模块会占用 CapSense 模块和一个定时器（定时器 1）。

需要特定引脚资源（包括 LCD 和 I2CHW）的用户模块必须在为 CSDAUTO 用户模块建立引脚连接而启动 CSDAUTO 向导前布置完毕。

在布置电容式传感器连接时，应避免使用 P1[0] 和 P1[1]。 这些引脚用来对部件进行编程，而且有可能存在过大的路由电容值，从而会影响传感器的灵敏度和噪声。

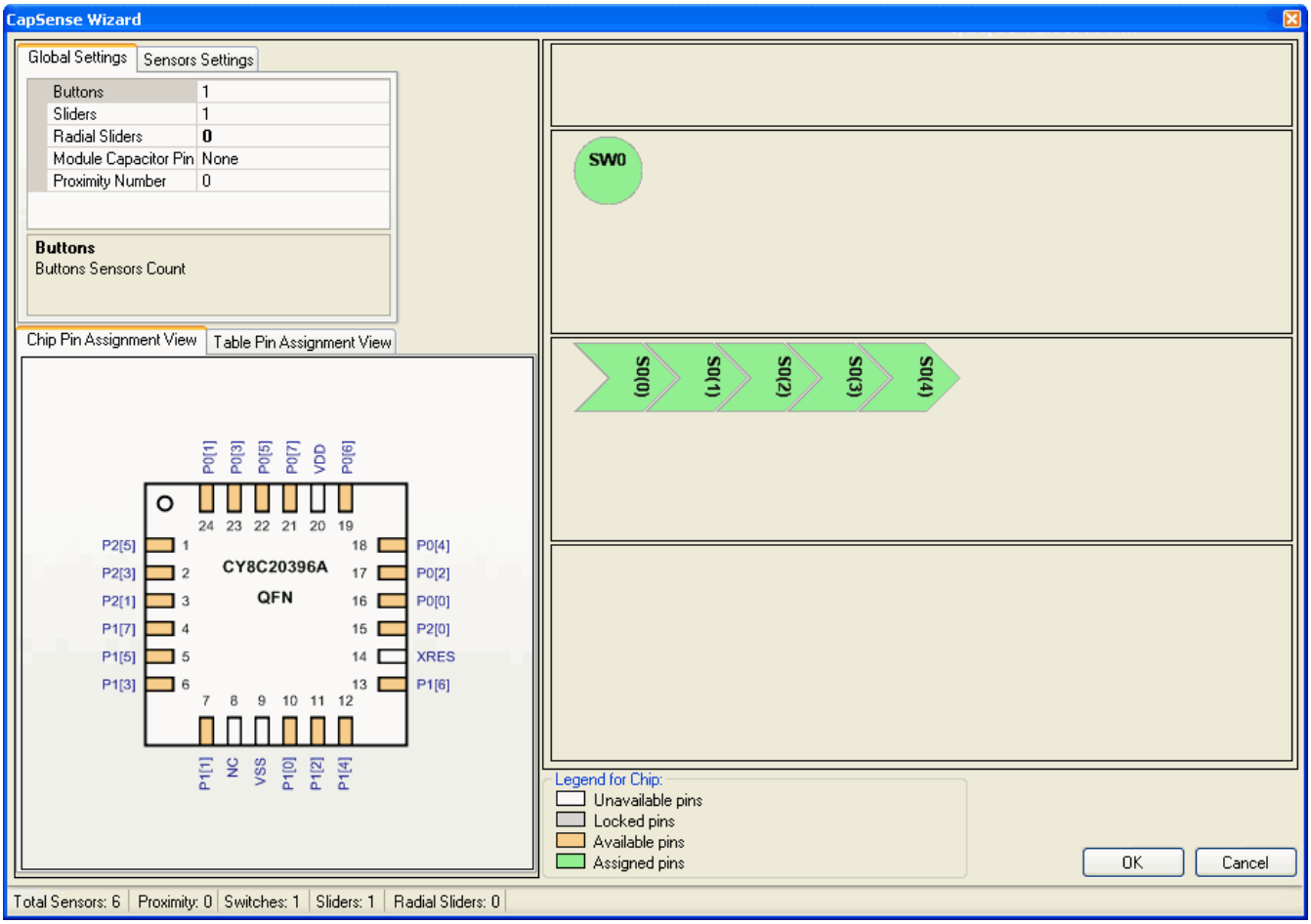
访问向导

1. 要访问向导，请右键单击器件编辑器互连视图中的任意 CSDAUTO 模块，然后左键单击以选择“CSDAUTO 向导”。



[+] Feedback

2. 打开的向导将显示数字输入框，可在其中输入传感器数量和滑条传感器数量。



向导引脚注释

白色 - 引脚不可用作 CapSense 输入端。

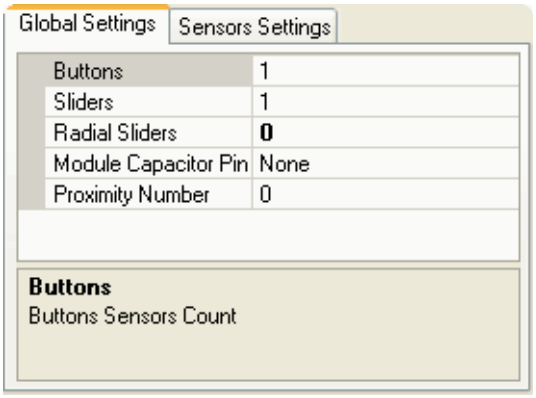
灰色 - 引脚已锁定。产生此情况的可能原因有两种。第一种可能性是另一个用户模块，例如 LCD 或 I^2C 占用了该引脚。第二种可能性是引脚已更改为使用非默认名称。要恢复使用引脚的默认名称，请在引脚分布视图中展开引脚，然后从**选择**菜单中选择**默认**。现在可在向导中分配引脚了。

橙色 - 引脚可分配。

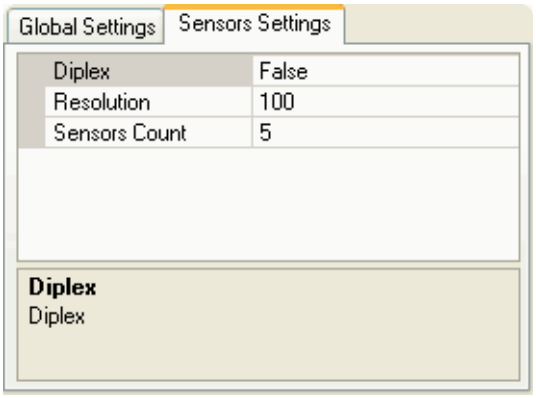
绿色 - 引脚已分配为 CapSense 输入端。

[+] Feedback

3. 键入独立按键、滑条和径向滑条的数量。 传感器 （按键加滑条元件）总数不得超过可用引脚数。 输入数据后，按 [Enter] 键更新显示屏使其显示新值。 X-Y 触摸板需要两个滑条。



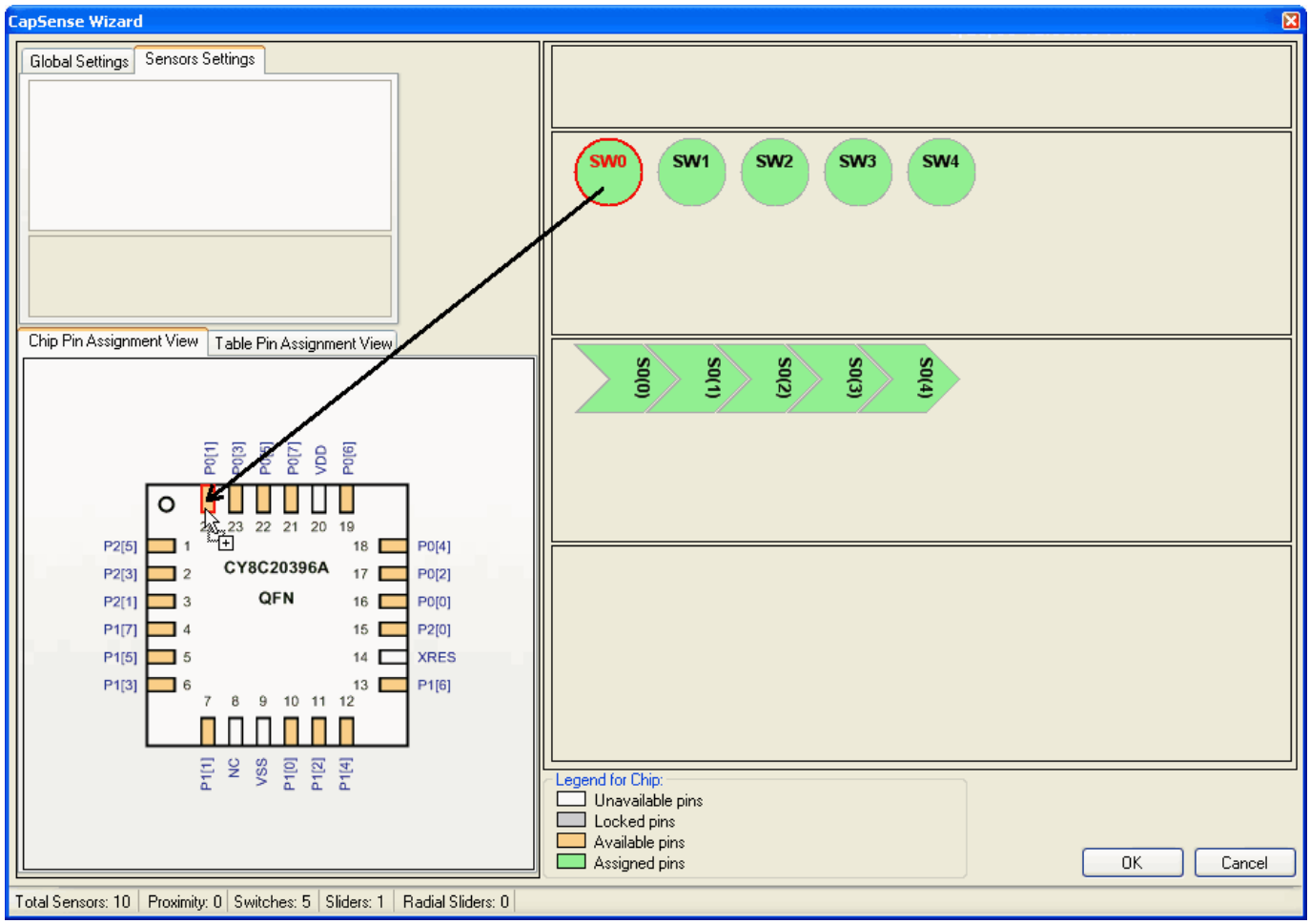
4. 选择 “ 传感器设置 ” 即可设置滑条和径向滑条。 要更改设置，请单击以激活其中一个滑条。 键入每个滑条中传感器元件的数量。 滑条传感器中的传感器实际最小数量为五，最大值受限于引脚数量。 输入数据后，按 [Enter] 键更新显示屏。 选择调制器电容 (C_{mod}) 引脚。 从可用的引脚 P0[1] 或 P0[3] 中进行选择。如果使用内部电容，则选择 “ 无 ”。 通常，使用外部 2.2 nF 电容可获得更好的信噪比。



5. 键入输出分辨率。 最小值为五。 双工滑条的最大值为 （传感器所用引脚数 - 1）× 2_{16} - 1 或 （2 x 传感器所用引脚数 - 1）× 2_{16} - 1。 软件尝试用相邻段的相对强度将触摸结果插入到指定的分辨率中。 软件报告的滑条触摸结果在零和分辨率 - 1 之间。
6. 如果需要，请选择 “ 双工 ”。 这样会将选定用于传感器的引脚数量映射成电路板上传感器位置数量的两倍。 只有这些双工传感器的前半部分会显示出来，后半部分将按照前面 “ 双工法 ” 章节所述自动映射。 查看 “ 双工 ” 章节，找到引脚连接的双工表格。

[+] Feedback

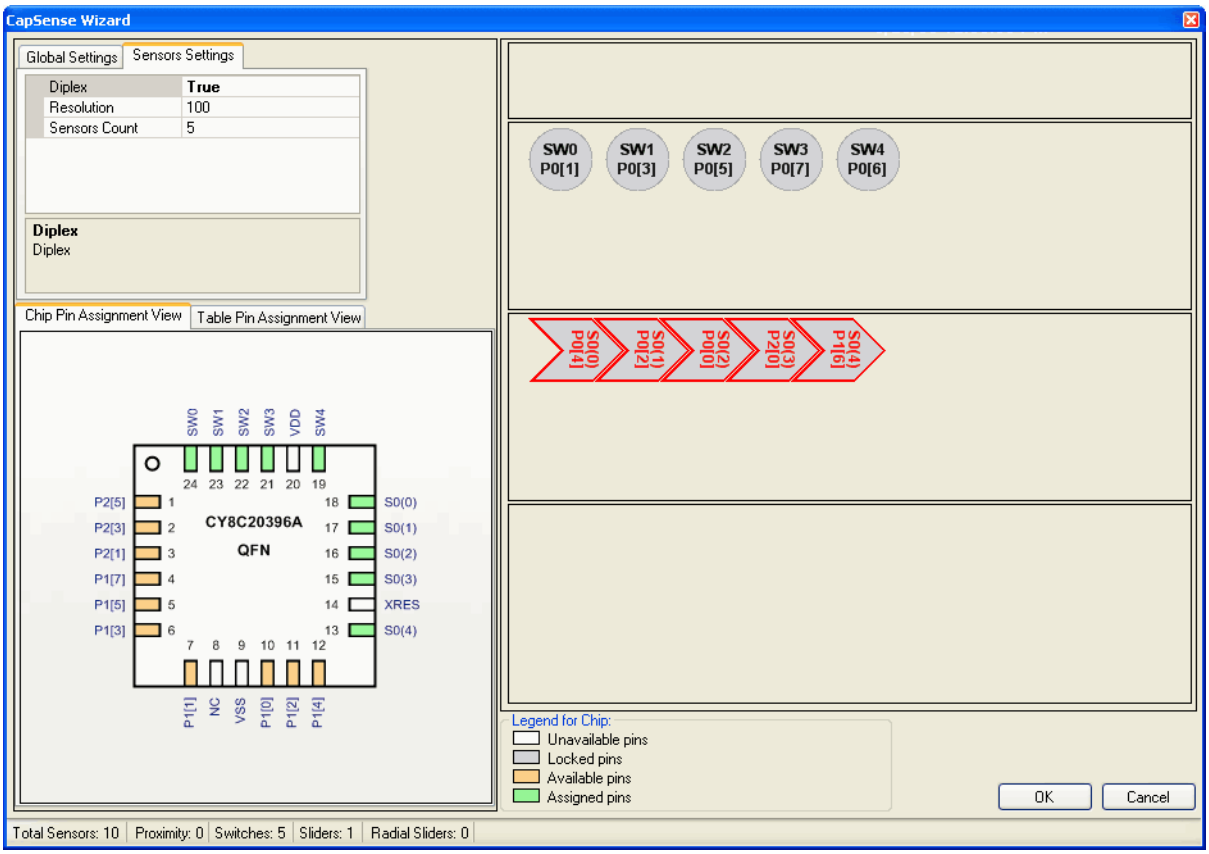
7. 在引脚分配视图中将开关或传感器拖到引脚上，这样就可以将开关或传感器分配到各个引脚。 您可以选择在芯片引脚分配视图或表格引脚分配视图中将开关或传感器分配到各个引脚。 端口引脚在选定后将变为绿色，无法再用于分配。 通过将端口引脚拖回未指定的表格，可改变传感器的分配情况。 确保避免选择已经指定用于其他用户模块的引脚。



[+] Feedback

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

这时就完成了传感器布置操作。 右键单击器件编辑器窗口并选择**刷新**即可更新引脚连接。



设置用户模块参数并生成应用程序。 如果愿意，您现在可以调整示例项目。

要更改引脚分配情况，请将光标置于已分配的引脚上，单击此引脚，然后将其拖到开关框外。 此时引脚就会处于未分配状态，您可以重新分配。

完成向导后，单击 “ 生成应用程序 ”。 根据您输入的传感器数量、引脚分配、双工和分辨率，会生成一系列表格。 表格位于 CSDAUTO_Table.asm 中

[+] Feedback

传感器表格

传感器表格包含每个传感器的 2-byte 条目。 第一个字节是端口编号，第二个字节是位的位掩码 （而非位的编号）。 表格中列出了所有的独立传感器，然后是按顺序排列的各个传感器。 以下是包含六个传感器的示例表格：

```
CSDAUTO_Sensor_Table:
_CSDAUTO_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
```

此表格由 CSDAUTO_wGetPortPin() 子程式使用。

组表格

组表格定义了每个按键传感器组或滑条组。 每个滑条对应一个条目，自由按键传感器再对应一个条目。 第一个条目始终对应自由传感器。 每个条目有六字节。 第一个字节是传感器表格中该组起始处的索引。 第二个字节是该组中的传感器总数。 第三个字节表示是否对滑条采用了双工法 （4 表示已采用双工法，0 表示未采用）。 第四个、第五个和第六个字节是固定点乘数，需要与滑条计算所得的质心值相乘，以获得 CSDAUTO 向导中所需的分辨率。

```
CSDAUTO_Group_Table:
_CSDAUTO_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
```

双工表格

双工表格扫描顺序数据用于已启用双工的滑条组。 否则，就会只创建标签但不包含任何数据。 此表格包括两部分：每个滑条的传感器映射，以及每个单独滑条与对应表格的参照。 此处展示了一个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


 CSDAUTO_Diplex_Table:
_CSDAUTO_Diplex_Table:
 db >DiplexTable_0, <DiplexTable_0
 db >DiplexTable_1, <DiplexTable_1
```


[+] Feedback

参数和资源

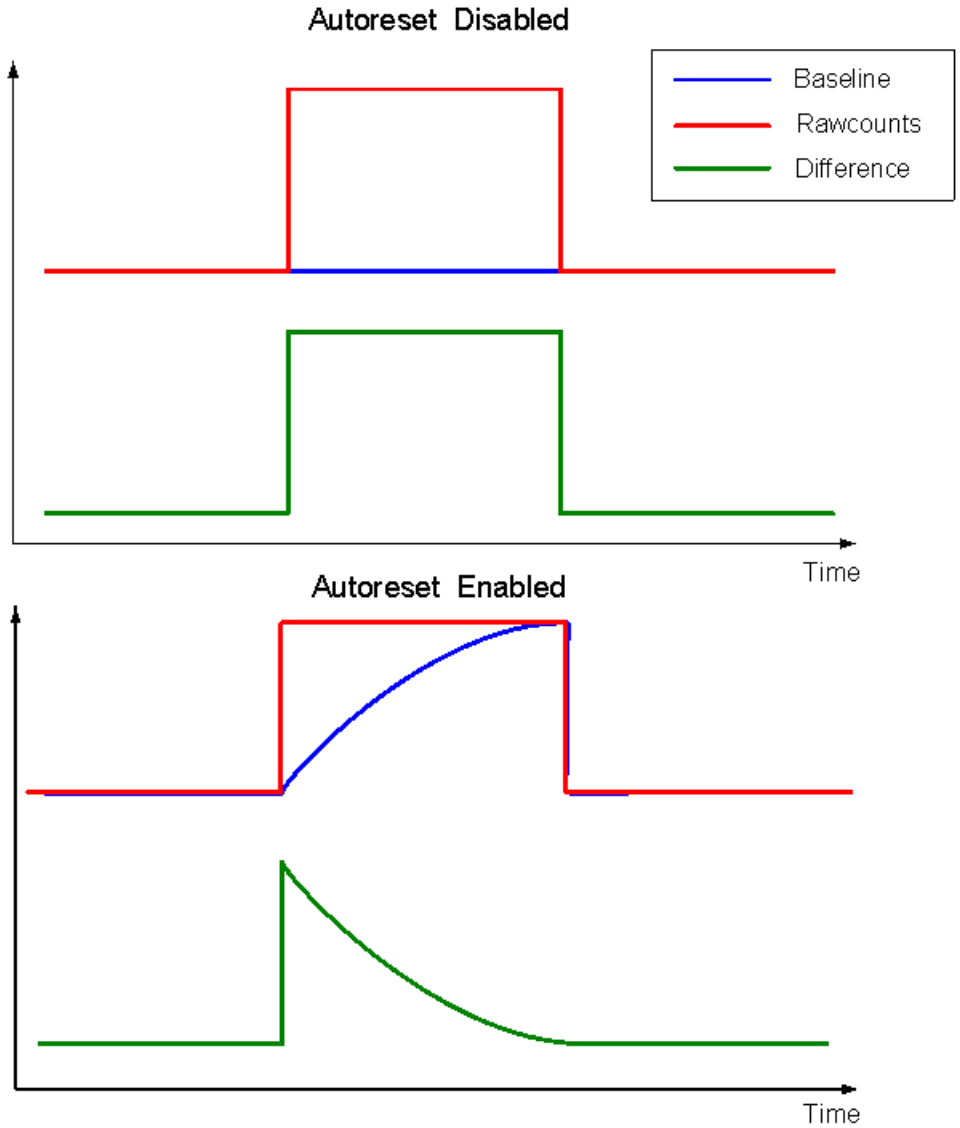
传感器自动复位

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

在此参数设置为**禁用**时，基线只在原始计数与基线的差值低于噪声阈值参数时更新。 除非在没有任何物体触碰到传感器时原始计数突然上升导致传感器保持永久开启状态，否则应当保持此参数的禁用状态。

下图说明了此参数对基线更新的影响。

Figure 7. 传感器自动复位参数



[+] Feedback

去抖动

去抖动参数添加了一个用于传感器活动状态切换的去抖动计数器。 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。 去抖动计数器的值随着 bIsSensorActive 或 bIsAnySensorActive 函数递增。

可能的值为 1 到 255。 设置为 1 时不提供去抖动功能。

ShieldElectrodeOut

屏蔽电极信号源可以路由到 P0[7] 或 P1[2]。

应用程序编程接口

应用程序编程接口 (API) 函数作为用户模块的一部分提供，使您能够在较高的层级处理模块。 本节具体说明了每个函数对应的接口以及 include 文件所提供的相关常量。

每次布置用户模块时，都会为其分配一个实例名称。 默认情况下，PSoC Designer 将会为给定项目中此用户模块的第一个实例分配 CSDAUTO_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。 为简单起见，在后面的介绍中我们将实例名称缩短为 CSDAUTO。

注 ** 此时，如同所有用户模块 API 中，A 和 X 寄存器的值可能会通过调用 API 函数而发生更改。 调用函数的职责是，如果调用后需要用到 A 和 X 的值，则在调用前需要保留这些值。 为实现高效，选择了这项 “ 寄存器为易失性 ” 的策略，该策略从 PSoC Designer 1.0 版本开始实施。 C 语言编译器将自动满足此项要求。 汇编语言程序员也必须确保其代码遵守该策略。 尽管有些用户模块的 API 函数会保留 A 和 X 不变，但并不保证在未来也将如此。

输入点用来初始化 CSDAUTO、启动采样和停止 CSDAUTO。 在所有情况下，模块的实例名称将取代以下输入点中显示的 CSDAUTO 前缀。 未能使用正确的实例名称是产生语法错误的常见原因。

API 函数使用不同的全局数组。 您不能手动更改这些数组， 但可以在需要调试时检查这些值。 例如，您可以使用制图工具显示数组内容。 有多种全局数组：

- CSDAUTO_waSnsBaseline[]
- CSDAUTO_waSnsResult[]
- CSDAUTO_waSnsDiff[]
- CSDAUTO_baSnsOnMask[]

CSDAUTO_waSnsBaseline[] - 这是包括每个传感器的基线数据的整数数组。 数组大小与传感器数量相等。 CSDAUTO_waSnsBaseline[] 数组由以下函数更新：

- CSDAUTO_UpdateAllBaselines();
- CSDAUTO_UpdateSensorBaseline();
- CSDAUTO_InitializeBaselines().

CSDAUTO_waSnsResult[] - 这是包括每个传感器的原始数据的整数数组。 数组大小与传感器数量相等。 CSDAUTO_waSnsResult[] 数据由以下函数更新：

- CSDAUTO_ScanSensor();
- CSDAUTO_ScanAllSensors().

CSDAUTO_waSnsDiff [] - 这是包括每个传感器的原始数据和基线数据之间差值的整数数组。 数组大小与传感器数量相等。

[+] Feedback

CSDAUTO_baSnsOnMask[] - 这是保持传感器开启或关闭状态 （按键或滑条）的字节数组。
CSDAUTO_baSnsOnMask[0] 包括传感器 0 到 7 的掩码位 （传感器 0 即位 0，传感器 1 即位 1）。
CSDAUTO_baSnsOnMask[1] 包括传感器 8 到 15 的掩码位 （如果需要），依此类推。 该字节数组含有足够多的元素，可以包括所布置的全部传感器。 按键开启时位值为 1，关闭时位值为 0。
CSDAUTO_baSnsOnMask[] 数据由 CSDAUTO_blsSensorActive(BYTE bSensnor) 函数或 CSDAUTO_bIsAnySensorActive() 子程式更新。

CSDAUTO_Start

说明：

初始化寄存器并启动用户模块。 应该在调用其他任何用户模块函数前调用此函数。

C 语言原型：

```
void  CSDAUTO_Start()
```

汇编语言：

```
lcall  CSDAUTO_Start
```

参数：

无

返回值：

无

副作用：

CSDAUTO_Stop

说明：

停止传感器扫描、禁用内部中断并调用 CSDAUTO_ClearSensors() 以便将所有传感器复位到非活动状态。

C 语言原型：

```
void  CSDAUTO_Stop()
```

汇编语言：

```
lcall  CSDAUTO_Stop
```

参数：

无

返回值：

无

副作用：

[+] Feedback

CSDAUTO_ScanSensor

说明：

扫描所选的传感器。 每个传感器在传感器阵列中都有一个唯一编号。 此编号由 CSDAUTO 向导按序分配。 Sw0 代表传感器 0，Sw1 代表传感器 1，依此类推。

C 语言原型：

```
void CSDAUTO_ScanSensor(BYTE bSensor);
```

汇编语言：

```
mov A,  bSensor
lcall CSDAUTO_ScanSensor
```

参数：

A => 传感器编号

返回值：

无

副作用

CSDAUTO_UpdateSensorBaseline

说明：

针对每个传感器独立计算得出的历史计数值称为这个传感器的基线。 基线通过一种称为漏桶的方法得以更新。

漏桶方法采用了以下算法。

1. 每次调用 CSDAUTO_UpdateSensorBaseline() 时，通过从原始计数值中减去之前的基线，计算得出差值计数。 此差值存储在 CSDAUTO_waSnsDiff[] 数组内并提供给您。
2. 如果 “ 传感器自动复位 ” 已禁用，每次调用 CSDAUTO_UpdateSensorBaseline() 时，会将差值计数与噪声阈值进行比较。 如果差值低于噪声阈值，则累加到虚拟漏桶内。 如果差值高于噪声阈值，则不更新漏桶。 如果 “ 传感器自动复位 ” 已启用，则无论噪声阈值参数如何，都将差值累加到虚拟漏桶内。
3. 一旦虚拟漏桶内所累加的差值计数达到了 BaselineUpdateThreshold，基线就递增 1，而漏桶复位为 0。
4. 如果差值计数低于噪声阈值，则保持在 waSnsDiff[] 数组内的值将复位为 0。 因此，该数组不包含值大于 0 但低于噪声阈值的元素。

C 语言原型：

```
void CSDAUTO_UpdateSensorBaseline(BYTE bSensor)
```

汇编语言：

```
mov   A,   bSensor
lcall  CSDAUTO_UpdateSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

[+] Feedback

副作用：

CSDAUTO_bIsSensorActive

说明：

以传感器的手指阈值为基准，针对给定传感器检查差值计数数组与手指阈值的比较结果。 这里要考虑到迟滞因素。 根据传感器目前是否运行，以手指阈值为基础，加上或减去迟滞值。 如果传感器处于活动状态，则阈值降低。 如果传感器处于不活动状态，则阈值升高。 此函数还能够更新 CSDAUTO_baSnsOnMask[] 数组内的传感器位。

C 语言原型：

```
BYTE  CSDAUTO_bIsSensorActive(BYTE bSensor)
```

汇编语言：

```
mov  A,  bSensor
lcall CSDAUTO_bIsSensorActive
```

参数：

bSensor A => 传感器编号

返回值：

处于活动状态时返回值为 1，不处于活动状态则为 0。

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

副作用：

CSDAUTO_bIsAnySensorActive

说明：

以各自手指阈值为基准，针对所有传感器检查差值计数数组与手指阈值的比较结果。 针对每个传感器调用 CSDAUTO_bIsSensorActive()，让 CSDAUTO_baSnsOnMask[] 数组在调用此函数后保持最新状态。

C 语言原型：

```
BYTE  CSDAUTO_bIsAnySensorActive()
```

汇编语言：

```
lcall CSDAUTO_bIsAnySensorActive
```

参数：

无

返回值：

处于活动状态时返回值为 1，不处于活动状态则为 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：

[+] Feedback

CSDAUTO_wGetCentroidPos

说明：

检查差值数组，看是否存在质心。 如果存在，则偏移和长度存储在临时变量内，质心位置计入 CSDAUTO 向导指定的分辨率中。 此函数仅适用于 CSDAUTO 向导已定义滑条的情况。

C 语言原型：

```
WORD  CSDAUTO_wGetCentroidPos(BYTE bSnsGroup)
```

汇编语言：

```
mov  A,  bSnsGroup
lcall CSDAUTO_wGetCentroidPos
```

参数：

bSnsGroup A => 组编号

此参数是用作滑条的特定传感器组的参考值。 组 0 代表按键。 滑条包含在组 1 或更高的组中。

返回值：

滑条的位置值，最低有效位在 A 中，最高有效位在 X 中。

副作用：

该子程式通过减去噪声阈值来修改差值计数。 只能在每次扫描后调用一次此子程式，以避免得到负的差值。 如果您的应用程序监测差值计数信号，则可在差值计数数据传输后调用此子程式。

如果有任何滑条传感器处于活动状态，函数将返回从零到 CSDAUTO 向导设置的分辨率值之间的数值。 如果没有传感器处于活动状态，函数返回 -1 (FFFFh)。 如果在执行质心 / 双工算法时出错，函数将返回 -1 (FFFFh)。 如果需要，您可以使用 CSDAUTO_blsSensorActive() 子程式确定触摸的滑条段。

注： 如果滑条段上的噪声计数大于噪声阈值，那么该子程式会生成错误的质心结果。 此噪声阀值应当仔细设置 （足够高于噪声水平），以防止噪声生成虚假的质心结果。

CSDAUTO_wGetRadialPos

说明：

检查差值数组，看是否存在质心。 如果存在，那么质心位置将根据 CSDAUTO 向导中指定的分辨率进行计算。 此函数仅适用于 CSDAUTO 向导定义的放射式滑条。

C 语言原型：

```
WORD  CSDAUTO_wGetRadialPos(BYTE bSnsGroup)
```

汇编语言：

```
mov  A,  bSnsGroup
lcall CSDAUTO_wGetRadialPos
```

参数：

bSnsGroup A => 组编号

此参数是用户进行操作的放射式滑条的编号。 可通过放射式滑条图形左边的 CSDAUTO UM 向导获得其编号 （例如，s2 表示放射式滑条编号为 2）。

返回值：

放射式滑条的位置值，A 中为最低有效位 (LSB)，X 中为最高有效位 (MSB)。

[+] Feedback

副作用：

每次扫描后只能调用一次子程式，以避免获得负差值以及基线更新。 如果用户的应用程序在监测差值计数信号，则可在差值计数数据传输之后调用该子程式。

如果有滑条传感器处于活动状态，则函数将从零值返回到 CSD 向导中设置的分辨率值。 如果没有传感器处于活动状态，则函数返回 -1 (FFFFh)。

注：如果滑条段上的噪声计数大于噪声阀值，则此子程式会生成虚假的质心结果。 此噪声阀值应当仔细设置 （足够高于噪声水平），以防止噪声生成虚假的质心结果。

CSDAUTO_wGetRadialInc

说明：

返回实际触摸移位，即当前触摸位置和之前触摸位置之间的差值。 此函数与 CSDAUTO_wGetRadialPos() 结合使用，并获取后者生成的数据 （数据保存在内部变量中）。

C 语言原型：

```
WORD  CSDAUTO_wGetRadialInc(BYTE bSnsGroup)
```

汇编语言：

```
mov  A,  bSnsGroup
lcall CSDAUTO_wGetRadialInc
```

参数：

bSnsGroup A => 组编号

此参数是用户进行操作的放射式滑条的编号。 可通过放射式滑条图形左边的 CSDAUTO UM 向导获得其编号 （例如，s2 表示放射式滑条编号为 2）。

返回值：

触摸移位值，顺时针为正，逆时针为负； A 中为最低有效位，X 中为最高有效位。

触摸移位值是当前触摸位置和之前触摸位置之间的差值。 如果在之前的扫描过程中没有触摸 （上一次 CSDAUTO_wGetRadialPos() 返回 -1 (FFFFh)），或者，此刻没有触摸 （这次 CSDAUTO_wGetRadialPos() 返回 -1 (FFFFh)）

副作用：

应当仅在 CSDAUTO_wGetRadialPos() API 后调用该子程式。 因为它使用的是 CSDAUTO_wGetRadialPos(). 设置的内部数据 CSDAUTO_waSliderPrevPos 和 CSDAUTO_waSliderCurrPos。

CSDAUTO_InitializeSensorBaseline

说明：

通过扫描选定的传感器为 CSDAUTO_waSnsBaseline[bSensor] 阵列元件加载初始值。 原始计数值将被复制到选定传感器的基线阵列元件中。 此函数可用来复位单个传感器的基线。

C 语言原型：

```
void CSDAUTO_InitializeSensorBaseline(BYTE bSensor)
```

汇编语言：

```
mov A, bSensor
lcall CSDAUTO_InitializeSensorBaseline
```


[+] Feedback

参数：

A => 传感器编号

返回值：

无

副作用：

CSDAUTO_InitializeBaselines

说明：

通过扫描每个传感器为 CSDAUTO_waSnsBaseline[] 阵列加载初始值。 原始计数值将被复制到每个传感器的基线阵列中。

C 语言原型：

```
void CSDAUTO_InitializeBaselines()
```

汇编语言：

```
lcall CSDAUTO_InitializeBaselines
```

参数：

无

返回值：

无

副作用：

CSDAUTO_SetScanMode

说明：

设置扫描速度和分辨率。 此函数可在运行时调用，以更改扫描速度和分辨率。 当一些传感器需要以不同的扫描速度和分辨率进行扫描时 （例如常规按键以及接近检测器），此函数非常有用。 常规按键能够以 9-bit 分辨率进行扫描。 接近检测器能够以较低的扫描频率、16-bit 分辨率以及较长的扫描时间为远程检测进行扫描。 此函数可与 CSDAUTO_ScanSensor() 函数结合使用。

C 语言原型：

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

汇编语言：

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

参数：

bSpeed、bResolution

返回值：

无

[+] Feedback

副作用：

CSDAUTO_SetIdacValue

说明：

此函数用于设置 iDAC 电流值。 如果一些传感器需要以不同的 iDAC 设置进行扫描时，可使用此函数。 此函数可与 CSDAUTO_ScanSensor(). 结合使用。

C 语言原型：

```
void CSDAUTO_SetIdacValue(BYTE bRefValue);
```

汇编语言：

```
mov     A, bIdacValue
lcall    CSDAUTO_SetIdacValue
```

参数：

bIdacValue - 设置 iDAC 值。 接受的值为 1.. 255。

返回值：

无

副作用：

CSDAUTO_SetPrescaler

说明：

此函数用于设置预分频器值。 如果一些传感器需要以预分频器设置进行扫描时，则可使用此函数。 此函数可与 CSDAUTO_ScanSensor(). 结合使用。

C 语言原型：

```
void CSDAUTO_SetPrescaler(BYTE bPrescaler);
```

汇编语言：

```
mov     A, bPrescaler
lcall    CSDAUTO_SetPrescaler
```

参数：

bPrescaler - 设置预分频器值。 接受的值在下表中列出：

名称	值	预分频器
CSDAUTO_PRESCALER_1	0x00	1
CSDAUTO_PRESCALER_2	0x01	2
CSDAUTO_PRESCALER_4	0x02	4
CSDAUTO_PRESCALER_8	0x03	8
CSDAUTO_PRESCALER_16	0x04	16

名称	值	预分频器
CSDAUTO_PRESCALER_32	0x05	32
CSDAUTO_PRESCALER_64	0x06	64
CSDAUTO_PRESCALER_128	0x07	128
CSDAUTO_PRESCALER_256	0x08	256

返回值：

无

副作用：

CSDAUTO_ClearSensors

说明：

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

C 语言原型：

```
void  CSDAUTO_ClearSensors()
```

汇编语言：

```
lcall  CSDAUTO_ClearSensors
```

参数：

无

返回值：

无

副作用：

[+] Feedback

CSDAUTO_wReadSensor

说明：

在 A （最低有效位）和 X （最高有效位）内返回关键原始扫描值。

C 语言原型：

```
WORD  CSDAUTO_wReadSensor(BYTE bSensor)
```

汇编语言：

```
mov A, bSensor
lcall CSDAUTO_wReadSensor
```

参数：

A => 传感器编号

返回值：

传感器的扫描值，最低有效位在 A 内，最高有效位在 X 内。

副作用：

CSDAUTO_wGetPortPin

说明：

返回给定传感器的端口编号和引脚掩码。 所传递的参数从 CSDAUTO_Sensor_Table[]. 中索引和选择数据。返回值可传递至 CSDAUTO_EnableSensor()、 CSDAUTO_DisableSensor().

C 语言原型：

```
WORD  CSDAUTO_wGetPortPin(BYTE bSensorNum)
```

汇编语言：

```
mov  A,  bSensorNumber
lcall     CSDAUTO_wGetPortPin
```

参数：

bSensorNumber - 此范围为 0 至 (n - 1)，其中 n 为 CSDAUTO 向导程序所设置的传感器总数加上包括在滑条内的传感器数量。 CSDAUTO_wGetPortPin() 使用传感器编号来确定选定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

副作用：

[+] Feedback

CSDAUTO_EnableSensor

说明：

配置所选定的传感器用于在下一个测量周期内执行测量操作。 端口和传感器可以使用 CSDAUTO_wGetPortPin() 函数进行选择，将端口编号和传感器位掩码分别加载至 X 和 A。 驱动模式已经修改以放置选定的端口和引脚至模拟 High-Z 模式下，并启用正确的模拟复用器总线输入。 这还会启用比较器函数。

C 语言原型：

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

汇编语言：

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

参数：

A => 传感器位图

X => 端口编号

返回值：

无

副作用：

CSDAUTO_DisableSensor

说明：

禁用 CSDAUTO_wGetPortPin() 函数所选定的传感器。 驱动模式改变为强 (001) 并设为零。 这样会让传感器有效地接地。 从端口引脚至 “ 模拟复用器总线 ” 的连接关断。 函数参数通过 CSDAUTO_wGetPortPin() 函数返回。

C 语言原型：

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

汇编语言：

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

参数：

A => 传感器位图

X => 端口编号

返回值：

无

副作用：

[+] Feedback

示例固件源代码

此代码启动用户模块，并持续扫描传感器。 通信部分可用于将值传递到 PC 制图工具。

```
//-------------------------------------------------------------------------
// Sample C code for the CSDAUTO 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)
{
  BYTE bIndex;

  M8C_EnableGInt;
  CSDAUTO_Start();
  CSDAUTO_InitializeBaselines() ; //Scan all sensors first time, init baseline

  while (1) {     //Loop forever
    for(bIndex=0; bIndex < CSDAUTO_TotalSensorCount; bIndex++) //Loop through all
sensors
  {
      CSDAUTO_ScanSensor(bIndex);      // Scan Sensors
      CSDAUTO_UpdateSensorBaseline(bIndex);  // Run baseline filter
  }

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

    // now we are ready to send all status variables to chart program
    // communication here
    //
    // OUTPUT CSDAUTO_waSnsResult[x] <- Raw Counts
    // OUTPUT CSDAUTO_waSnsDiff[x] <- Difference
    // OUTPUT CSDAUTO_waSnsBaseline[x] <- Baseline
    // OUTPUT CSDAUTO_baSnsOnMask[x] <- Sensor On/Off
  }
}
```


[+] Feedback

寄存器配置

Table 4. 模块 CapSense，寄存器： CS_CR0

位	7	6	5	4	3	2	1	0
值	0	0	CSDAUTO_PRSCLK	0	1	0	0	EN

Table 5. 模块 CapSense，寄存器： CS_CR1

位	7	6	5	4	3	2	1	0
值	1	扫描速度		0	0	0	0	0

电源： 0x01 接通模拟模块的电源。 0x00 断开模拟模块的电源。

Table 6. 模块 CapSense，寄存器： CS_CR2

位	7	6	5	4	3	2	1	0
值	1	0	0	0	0	1	0	0

Table 7. 模块 CapSense，寄存器： CS_CR3

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

Table 8. 模块 CapSense，寄存器： CS_CNTH

位	7	6	5	4	3	2	1	0
数据输出最高有效位								

Table 9. 模块 CapSense，寄存器： CS_CNTL

位	7	6	5	4	3	2	1	0
数据输出最低有效位								

Table 10. 模块 CapSense，寄存器： PRS_CR

模式 / 位	7	6	5	4	3	2	1	0
值	1	0	8/12 位	1	预分频器			

[+] Feedback

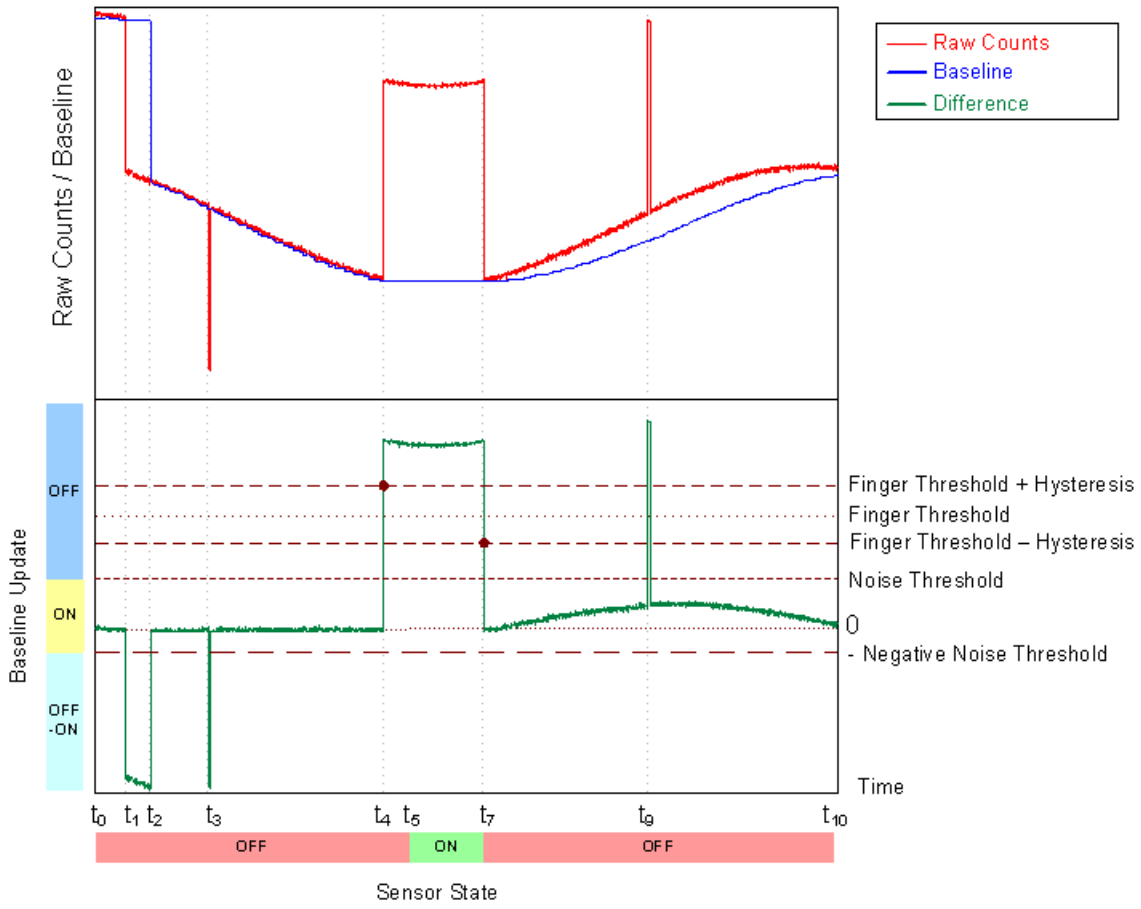
附录

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

CSDAUTO 参数的交互

下图说明了基线更新和决策逻辑操作，有助于更好地了解如何设置 UM 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。 手指阀值、噪声阀值、迟滞和负噪声阀值与差值信号一起显示 （原始计数 - 基线）。 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 8. “ 传感器自动复位 ” 设为 “ 禁用 ” 时原始计数、基线、差值信号变化示例



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。 由于两个连续转换之间的原始计数变化没有超过 “ 负噪声阀值 ” 参数 （绝对值），因此基线通过跟踪 “ 原始计数 ” 最小值进行更新，保持原始计数信号的下限值。

在 t_1 处，原始计数急剧下降，且负差值超过 “ 负噪声阀值 ”。 当手指在传感器上时器件通电，一段时间后手指移去，就会发生这种情况。 此时，基线更新机制处于冻结状态，内部超时计数器被激活。 当差值信号低于 LowBaselineReset 样本的 NegativeNoiseThreshold 时，基线复位。 这一情况发生在 t_2 处。

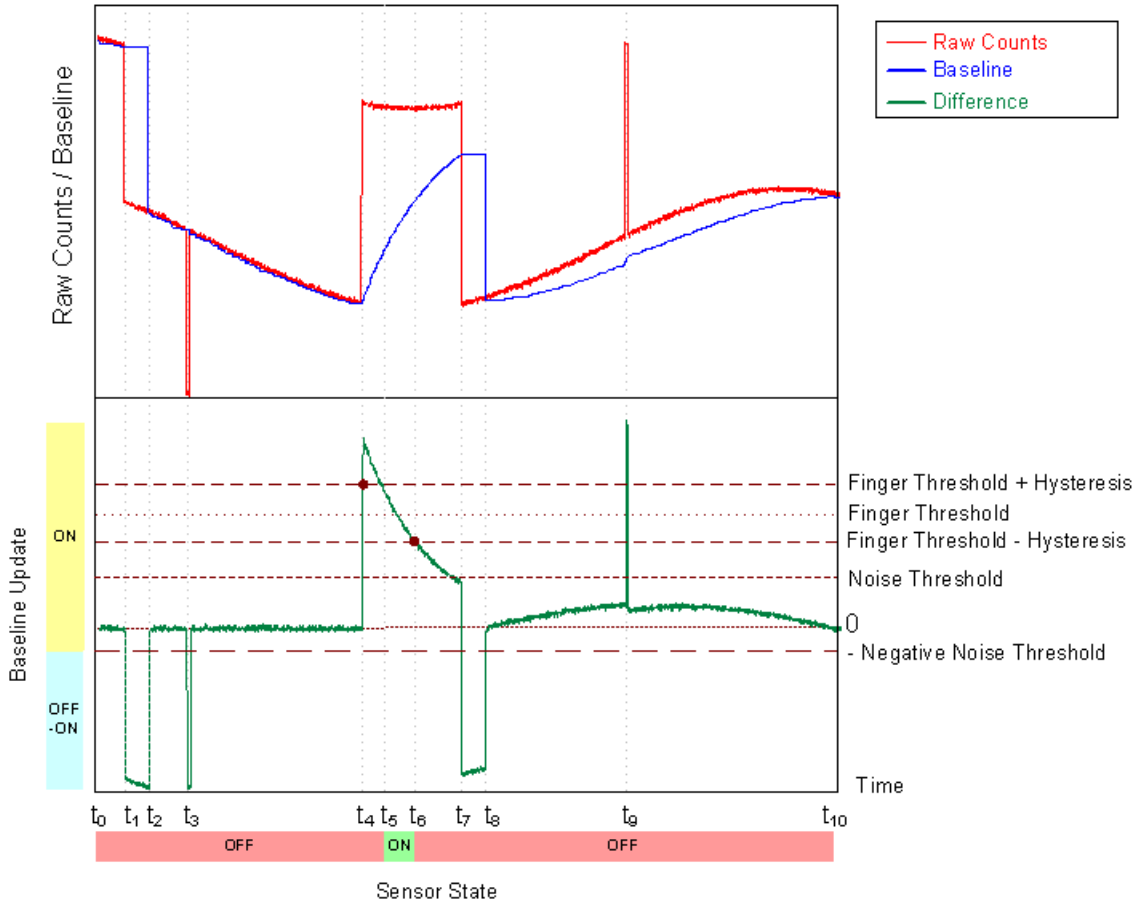
第二大负差信号尖峰位于 t_3 处，这一尖峰可能是由 ESD 事件等引起的。 由于样本计数中的尖峰宽度小于 LowBaselineReset 参数，因此基线保持挂起状态，尖峰被过滤。 这可以防止出现虚假的基线复位以及产生虚假的触摸检测。

[+] Feedback

传感器在 t_4 处被触摸。 当差值信号超过 “ 手指阀值 + 迟滞 ” 值时，内部去抖动计数器被激活。 如果信号超过该值多于去抖动样本的数量，则传感器状态设置为开启。 这一情况发生在 t_5 处。 在 t_7 处，当差值信号下降到 “ 手指阀值 - 迟滞水平 ” 以下时，传感器立即变回关闭状态。 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 和 t_{10} 之间缓慢向上漂移。 当差值信号低于 “ 噪声阀值 ” （“ 传感器自动复位 ” 设为 “ 禁用 ”），差值信号与漂移速率成比例时，基线使用漏桶算法进行更新。 可以使用 “ 基线更新阀值 ” 参数控制基线更新速度。 参数值越低，基线更新速度越快。

Figure 9. “ 传感器自动设置 ” 设为 “ 启用 ” 时原始计数、基线、差值信号变化示例



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：

- 当在 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缩短。
- 手指移去后，基线在 LowBaselineReset 样本（t_8）后复位，造成触摸检测的短时间中断。 这可以充当额外的去抖动机制。

[+] Feedback

版本历史

版本	创建者	说明
1.1	DHA	更新了无线性滑条时的放射式滑条位置。 目前最大分辨率为 3000。 消除了 0.5 的移位，并添加了对负值的补偿。 增加了添加额外滑条的功能。 将 ModCap 选择从用户模块参数移到了向导设置部分。 增加了新参数 “ 传感器灵敏度 ”。 增加了针对 TMA3xx 系列芯片的 CSD 和 CSDAUTO 支持。

Note PSoC Designer 5.1 在所有用户模块数据表中引入了版本历史。 此部分记录了当前用户模块版本和以前用户模块版本之间区别的高级描述。

Copyright © 2009-2010 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.