CapSense® Sigma-Delta Plus 数据手册 CSDPLUS V 1.20

Copyright © 2012-2014 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块				API 存储器 （字节）		每个传感器所使用的引脚数
	CapSense®	I²C/SPI	定时器	比较器	闪存	RAM	
CY8C20xx7/S、 CY8C20055							
用户模块	1	–	1	1	1203	42	1
滑条 API	–	–	–	–	1863	94	0
每个传感器	–	–	–	–	5	13	1

如需一个或多个使用此用户模块且完全配置的功能性示例工程，点击这里。

特性与概述

- CY8C20xx7/S 系列使用了最新的 “sigma-delta plus” （CSDPLUS）实现 CapSense 电容式感应
- 可配置的系统参数可以在各种应用 （如接近感应、按键或滑条）中优化调试的性能
- 支持高达 31 个电容式传感器和 6 个滑块
- 能够检测低至 0.1 pF 的触摸；因此，在多达 15 mm 厚的玻璃或 5mm 厚的塑料的情况下，仍然可以检测到手指
- CSDPLUS 对更大的寄生电容作出补偿
- CSDPLUS 改善了灵敏度并扩大了接近感应的检测范围
- CSDPLUS 改善了噪声性能
- 新的驱动屏蔽电极可用于多个 GPIO （GPIO 的数量取决于封装，最多可达 5 个）
- 驱动屏蔽电极改善了对高传感器寄生电容的性能。
- 驱动屏蔽电极可确保在有水的环境中器件仍能稳定工作，并增加了接近感应的检测范围
- 提高了对 GPIO 电流瞬变和 VDD 波动导致的噪声的抵抗能力
- 稳健进入和退出 I2C 睡眠模式 , 并可以从硬件地址匹配唤醒
- 支持将电容式传感器配置为独立按键或作为相关阵列以形成滑条或上述两者
- 通过传感器复用技巧，滑条单元的有效数目能达到所使用的 I/O 引脚数两倍
- 通过内插法可将滑条的分辨率高于物理间距
- 通过使用 CSDPLUS 向导完成传感器和引脚分配
- CY8C20055 系列不支持滑条

CSDPLUS 用户模块是基于差分电容式感应的方法的。该用户模块通过使用模拟复用器总线将电容式感应模拟电路连接到任意一个 PSoC 引脚。 CSDPLUS 用户模块可将活动传感器连接到模拟复用器总线，从而允许 CapSense 电路可以测量它的传感器电容，并将该电容值转换为数字代码。通过对 MUX_CRx 寄存器中的相应位依次进行设置，固件可以连续扫描各个传感器。

快速入门

1. 如果被使用，请选择并放置需要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 CSDPLUS 用户模块。
3. 在工作区浏览器中右键单击 CSDPLUS 用户模块，以访问 CSDPLUS 向导 （稍后将在本数据手册中加以介绍）。
4. 对传感器、滑条或旋转滑条的所需数量进行设置。
5. 对于滑条，输入特定于滑条的参数。
6. 将每个传感器分配到一个未使用的引脚。
7. 在 CSDPLUS 向导内输入连接外部调制电容引脚名称。
8. 输入用于屏蔽传感器 （若适用于用户模块参数列表）的引脚名称 （请参考下面部分中的说明）。
9. 生成应用，并切换到应用编辑器。
10. 根据需要调整示例代码，以实现独立传感器、滑条传感器和 / 或触摸板。
11. 通过使用 PSoC Designer™ 所生成的 HEX 文件来对目标电路板上的 PSoC 进行编程。

简介

CapSense 是一种人机界面技术，通过使用由导电表面构成 （通常是蚀刻在 PCB 上的衬垫）的传感器来检测人体的电容。由于 CapSense 检测人体电容，所以它能够透过塑料或玻璃等绝缘外覆层来检测。这些外覆层通常构成了设备的外型。这些特性使 CapSense 成为按键和电位器等机械输入器件的完美替代品。CapSense 的主要优势包括：

- 更清洁，更美观的设计。
- 可在低端产品提供更小的外形。
- 高级的用户接口特性，如 LED 效果和接近传感。
- 提高可靠性，因为各组件不磨损或具有有限的使用周期。
- 由於没有机械接口穿透，可改善泄漏电阻。
- 因为不需要机械输入器件的穿透或其他机械特性，可降低工具成本。

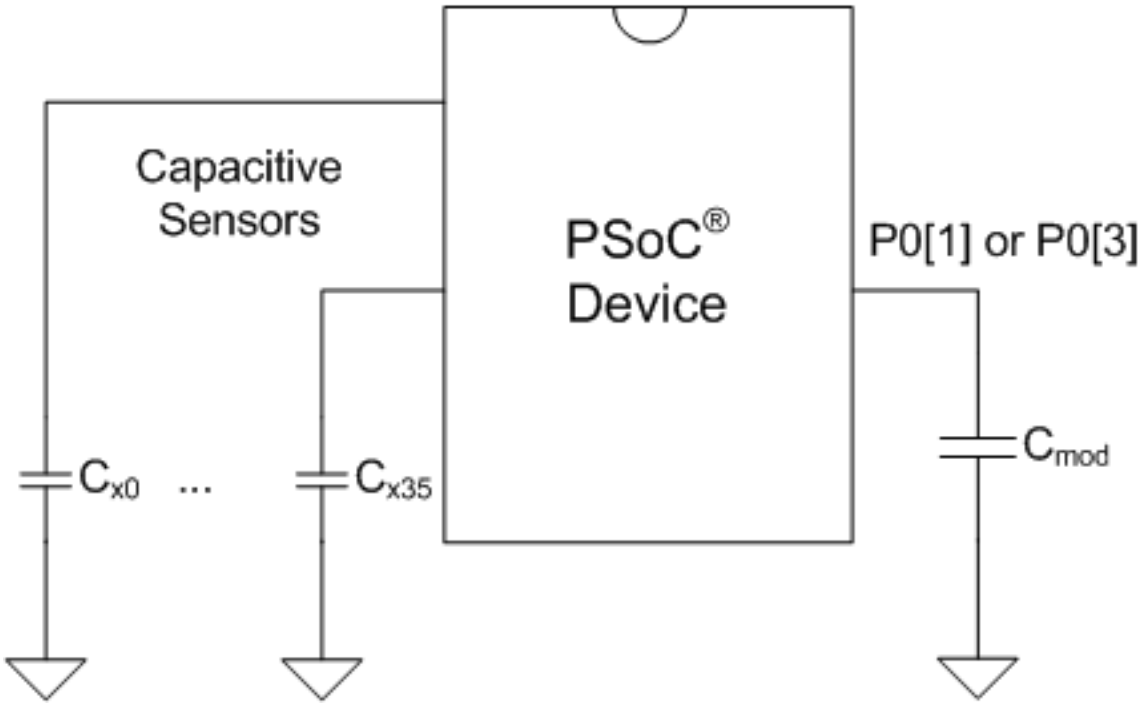
CSDPLUS 使用的电路会将 sigma-delta 电容转换成数字代码。电路的重要属性包括低 EMC 辐射和卓越的抗电磁干扰性能。

CSDPLUS 用户模块由 PCB 等级、 IC 等级以及软件组件构成，下面各节会这些组件进行说明。

PCB 等级

图 1 显示的是 CSDPLUS 的原理图。物理传感器通常是一个导电的图案，连接在与 PSoC I/O 引脚相连的 PCB 上，并且它上面带有一个绝缘覆盖层。有关 PCB 等级 CapSense 实现的详细信息，请参考赛普拉斯指南 CapSense 入门。

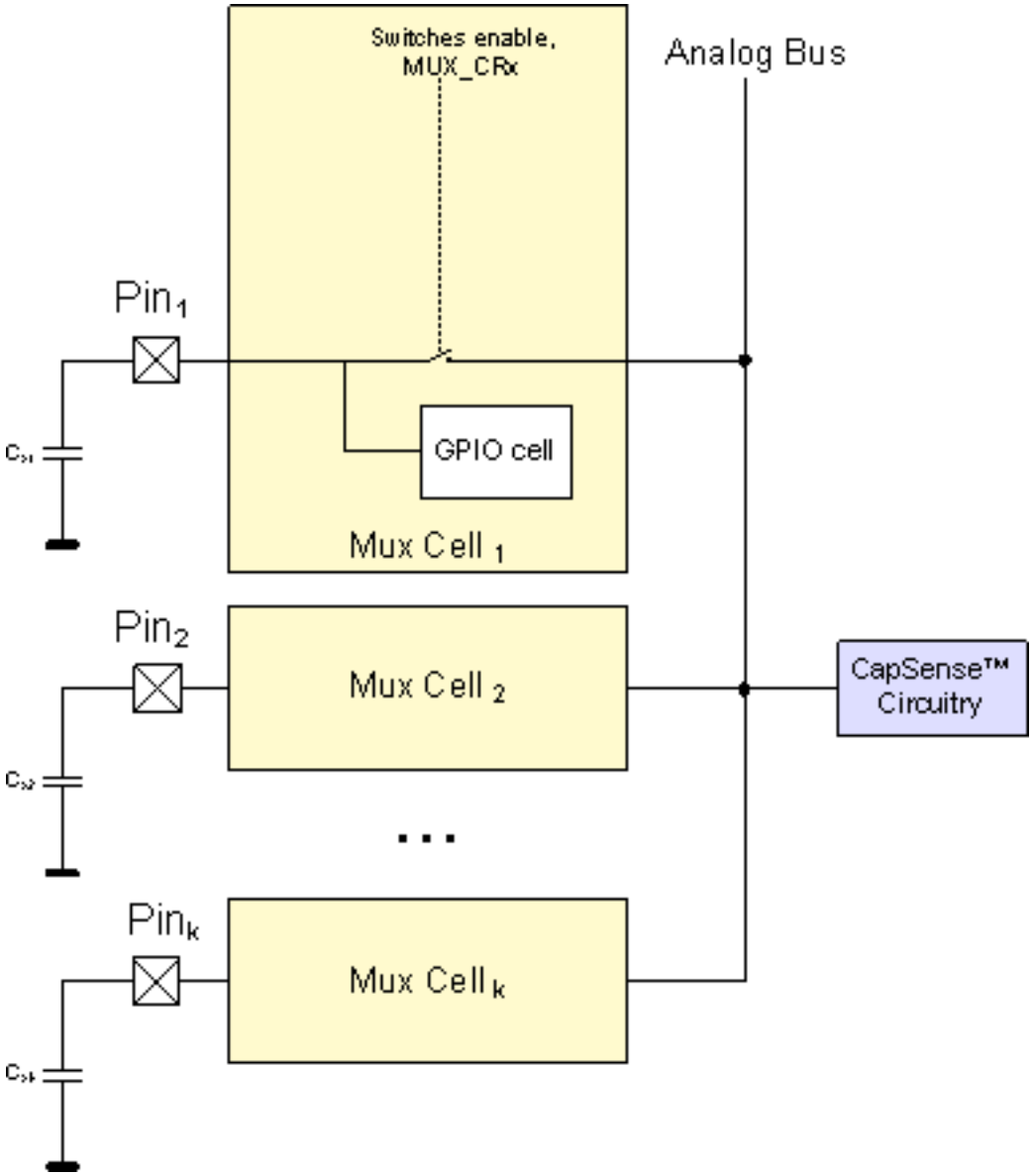
图 1. CSDPLUS 原理图



IC 等级

通过 CY8C20xx7/S 器件的模拟复用器总线可以将电容感应模拟电路连接到任何 PSoC 引脚。 CSDPLUS 用户模块可将活动传感器连接到模拟复用器总线，从而允许 CapSense 电路可以测量它的传感器电容，并将该电容值转换为数字代码。通过对 MUX_CRx 寄存器中的相应位依次进行设置，固件可以连续扫描各个传感器，如图 2 显示。

图 2. CY8C20xx7/S AMUX 框图



软件

CSDPLUS 软件组件具有以下特性：

- 通过可调试系统的配置参数可以在各种应用中优化调试的性能。
- 运行时， API 函数对电容转换电路中的原始计数值进行分析，以确定传感器状态并补偿环境变化。
- 对于连续、互相关联的传感器（例如：滑条和触摸板），会提供 API 函数，以便计算出一个分辨率高于传感器物理分辨率的位置。
- 高层软件函数提供滑条双工复用，使得单 I/O 引脚能够路由到两个物理传感器。因此，特定数量的滑条元件所消耗的 I/O 数量会节省一半。

推荐阅读

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

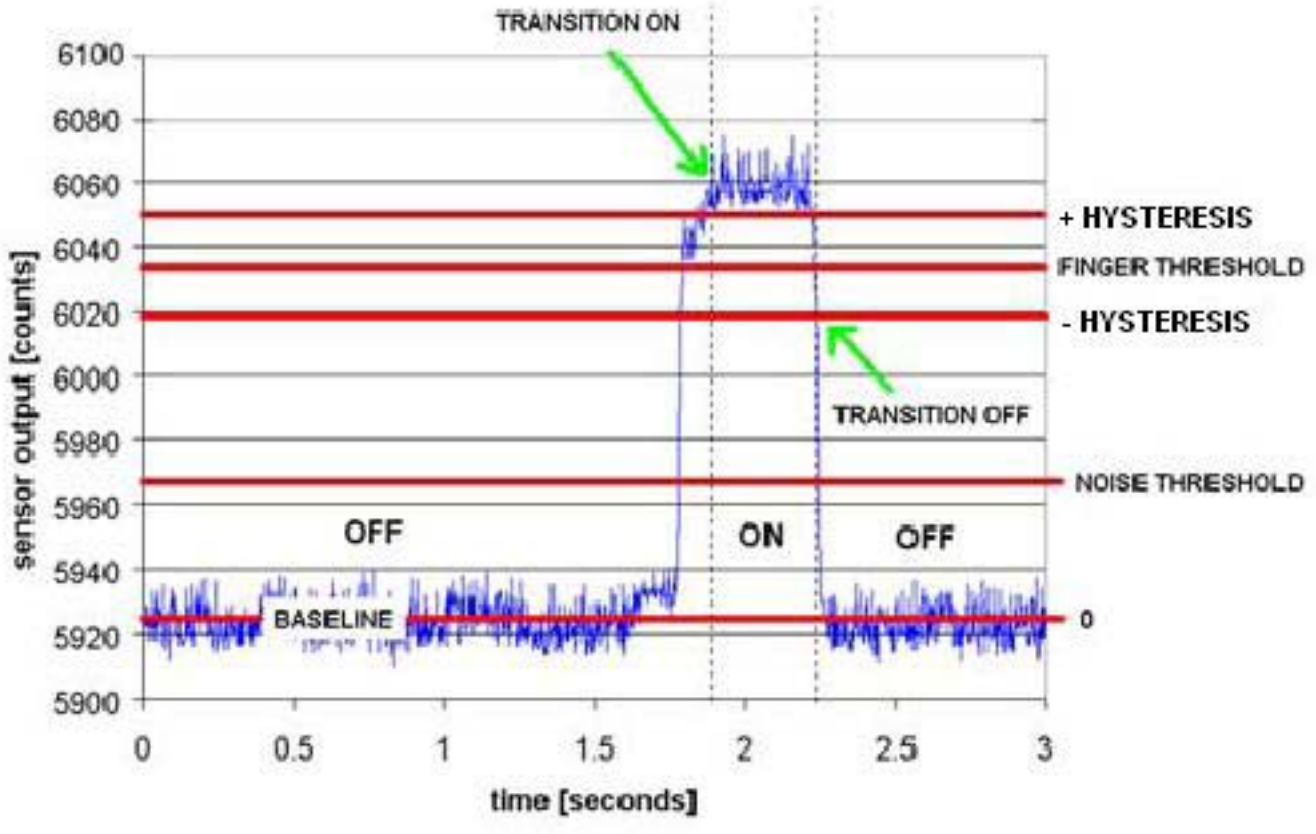
- Capsense 入门手册
- CY8C20xx6A/H/AS CapSense® 设计指南
- CY8C21x34/B CapSense® 设计指南
- CY8C20x34 CapSense® 设计指南
- CY8CMBR2044 CapSense® 设计指南

电容式感应的实现

按键

类似于机械式按键，可将 CapSense 按键用于离散控制，如打开 / 关闭开关、功能键和菜单键等。 API 函数对各传感器中的电容信号 （原始计数）进行监控并将这些值与可调试阈值参数进行比较。触摸传感器时，它的电容信号将增加；如果该增量满足 CSD 决策逻辑决定的值，会激活该传感器。图 3 显示了当传感器激活时，发出的一个典型信号 （蓝线）。可以调试阈值 （红线）以提供所需要的性能。

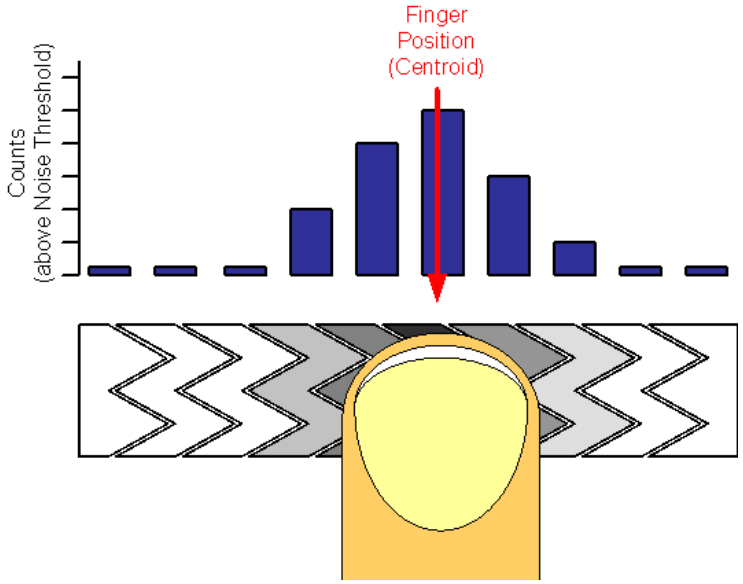
图 3. 传感器激活时的电容信号



滑条

同机械式电位器类似，可将 CapSense 滑条应用在需要连续电平的控制中。它们包括照明控件 （调光器）、音量控件、图形均衡器和速度控件。通过使用一系列相邻的电容传感器可以实现 CapSense 滑条。当手指启动某一滑条时，相邻传感器会对电容信号的增量进行记录，如图 4 所示。通过计算活动传感器组的质心位置，可以确定触摸的正确位置。实行滑条的传感器最小数量为五，最大值仅受限于 PSoC 器件上可用的 I/O 引脚数量。

图 4. 滑条上手指的插值质心位置

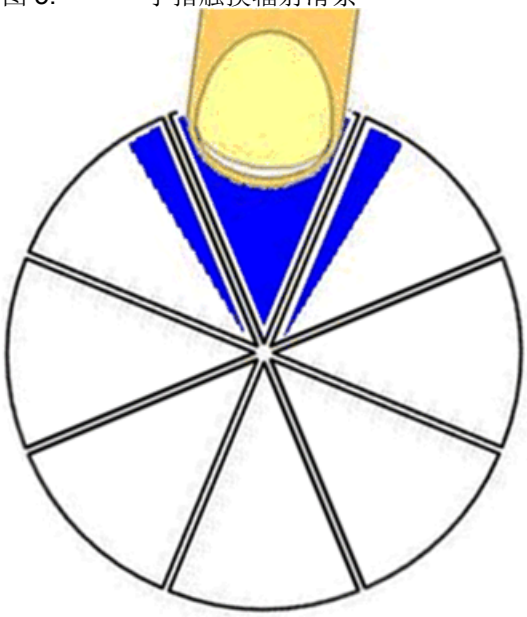


Radial Slider （辐射滑条）

CSDPLUS 支持两种滑条类型：线性和辐射。线性滑条有起点和终点，而辐射滑条却没有 （参见图 5）。在两种情况下，发生触摸时，质心算法对从相邻传感器到传感器的最大信号进行计算，以内插触摸的正确位置。辐射滑条未采用双工复用。 CSDPLUS 用户模块包含两个支持辐射滑条的特殊 API 函数。第一个函数为 CSDPLUS_wGetRadiaPos()，用于返回质心位置。第二个函数为 CSDPLUS_wGetRadialInc()，用于返回手指的偏移，它的单位是分辨率的单位。当手指以顺时针方向移动时， CSDe_wGetRadialInc() 会返回一个正偏移。参考点 （0）位于第一个传感器的中心。

线性滑条和辐射滑条解析度都受到限制，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。

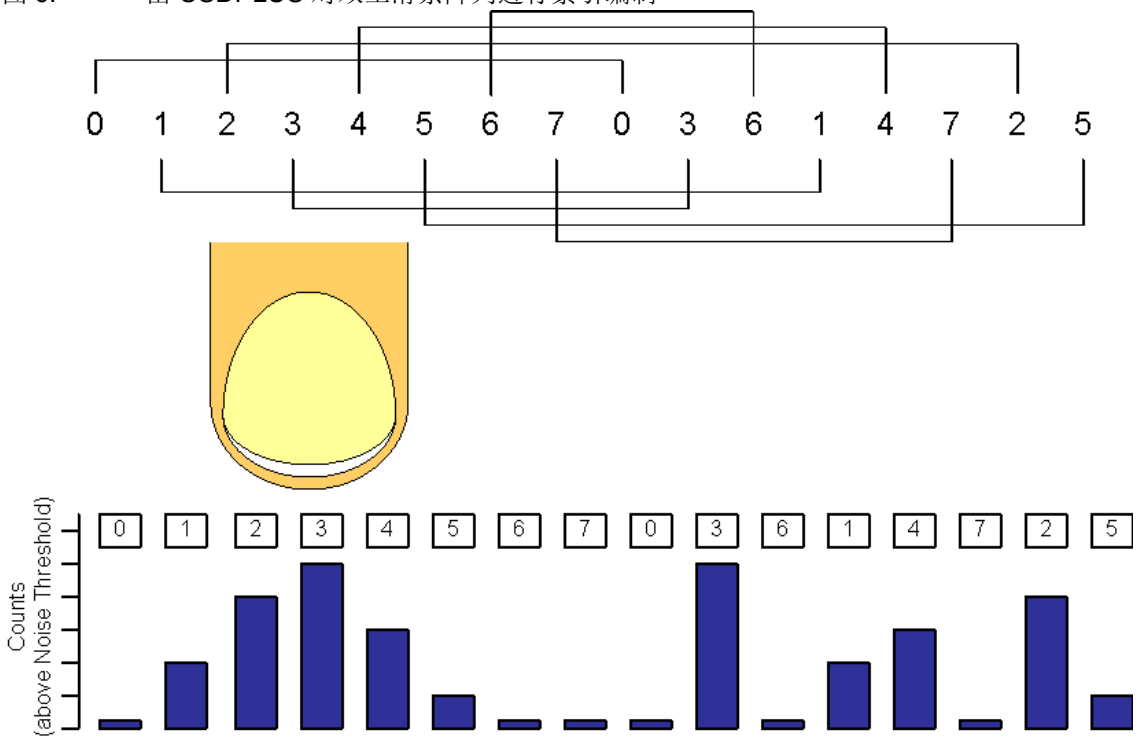
图 5. 手指触摸辐射滑条



双工

双工复用中，作为滑条元素的每个 PSoC 引脚都会映射到滑条传感器阵列中的两个物理位置上。根据 CSDPLUS 向导分配的端口引脚，物理位置的前半 （或数字上低的）部分会被映射。物理传感器位置的后半 （或数字上高的）部分将通过图 6 中所示的模式被自动映射。

图 6. 由 CSDPLUS 对双工滑条阵列进行索引编制



越接近滑条下半部分的强信号，将导致上半部分产生相同程度的伪信号。然而，在上半部分中，这些结果被分散和断连。质心算法搜索相邻最强的一组信号，以确定解析的滑条位置。映射上半传感器的格式确保该半部分中的有效信号格式不会成为剩下半部分的有效信号格式，如图 6 所示。

必须确保传感器到印制电路板 （PCB）上的引脚的映射情况符合双工算法所用的按 3 编制索引序列。双工滑条中传感器对的电容应该合理匹配 （不超过 10 pF）。当选择双工时， CSDPLUS 向导会自动生成双工传感器的索引表。表 1 显示的是双工成 28 个 PSoC I/O 引脚的 56 个滑条段的双工序列。

表 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
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

滑条段总计数	段序列
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

外部组件选择（C_{mod}）

CSDPLUS 需要一个外部调制电容（C_{mod}），该电容从 Vss 连接到一个专用 PSoC 引脚：P0[1] 或 P0[3]。可以在“Global Settings（全局设置）> Modulator Capacitor Pin”（调制电容引脚）中的 CSDPLUS 向导进行 C_{mod} 引脚分配。所选引脚不得用于其他用途。使用陶瓷电容以及外部调制电容的建议值为 2.2 nF。温度电容系数并不重要。此外，在所有 CapSense 传感器走线中使用 560 Ω 的串联电阻可抑制干扰。该电阻必须尽可能接近 PSoC。

驱动屏蔽电极

通过一个驱动屏蔽电极可以降低传感器的寄生电容（C_p）。目的在于当覆盖层上有水时会提高传感器灵敏度并防止错误传感器触摸。

屏蔽电极应该位于感应电极之后或其外侧，如图 7 所示。当水滴落到覆盖层上，而且没有驱动屏蔽电极时，各传感器和 PCB 的其他导体之间的电容耦合或寄生电容将增加。发生传感器电容信号的相应增加，这样可达到能够错误激活传感器的值。驱动屏蔽电极避免寄生电容耦合。因此水滴对传感器电容信号没有产生影响，从而防止错误激活。

图 7.　驱动屏蔽电极 PCB 布局

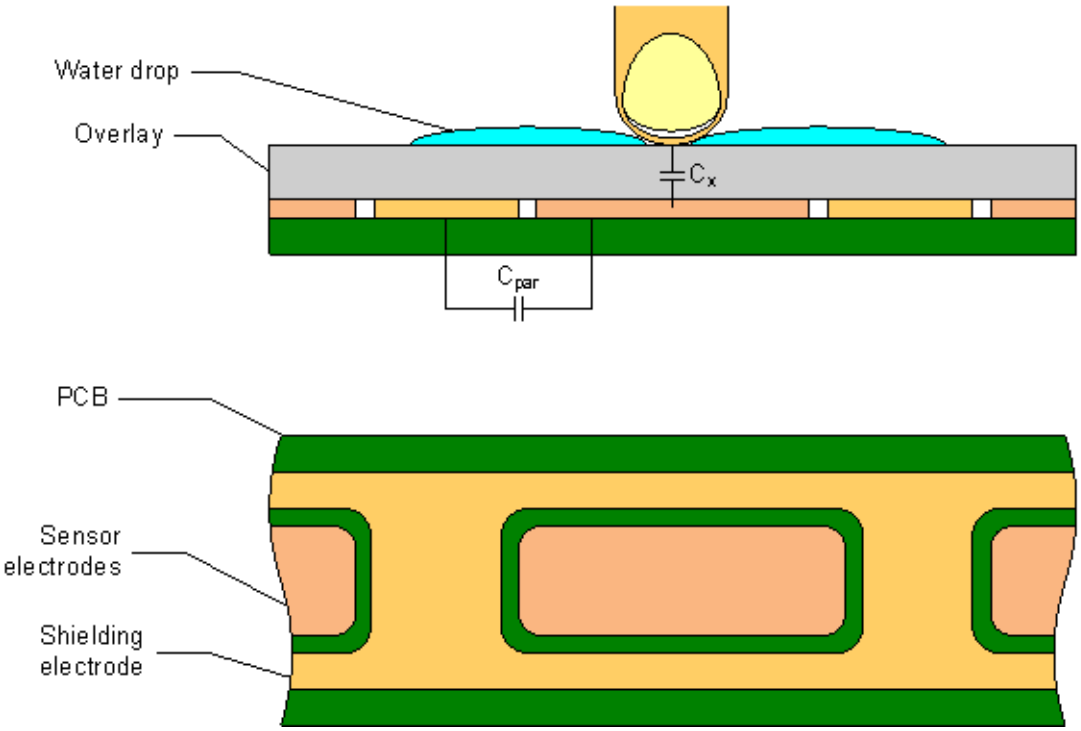
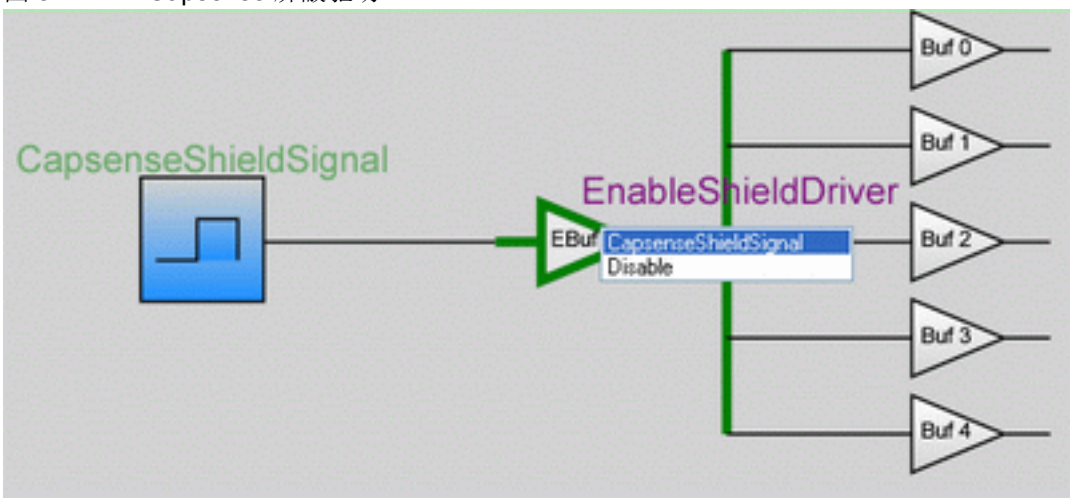


图 7 显示的是按键的驱动屏蔽电极。作为另一替代方法，屏蔽电极可以安装在相对的 PCB 层上，其中包括按键下面的平板。对于这种情况，建议使用填充率约为 30 至 40% 的填充模式。不需要其他接地层。

屏蔽电极必须连接到任何专用的 PSoC 引脚：P2[4]、P2[2]、P0[2]、P0[0] 或 P1[2]。将选定引脚的驱动模式设置为**强**。可在 PSoC 器件和屏蔽电极之间连接 560 Ω 上升限制电阻，以减少散发的电磁干扰（EMI）。

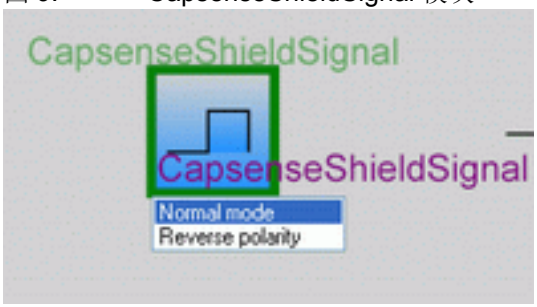
Capsense 屏蔽信号可以通过 5 个 ShieldBuffer （屏蔽缓冲区）路由到 P2[4]、 P2[2]、 P0[2]、 P0[0] 或 P1[2] 引脚，并由 PSoC Designer 芯片编辑器中的 EnableShieldDriver （使能屏蔽驱动）模块使能。

图 8.　Capsense 屏蔽驱动



屏蔽驱动器可在下面两种模式下驱动屏蔽信号：正常模式和反向极性模式。当选择反向极性模式时，会反转屏蔽驱动时钟。

图 9. CapsenseShieldSignal 模块



电源要求

表 2. CSDPLUS 电源要求

参数	最小值	典型值	最大值	单位	测试条件和注释
V_{DD}	1.71[a]	–	5.50	V	如果 V_{DD} 下降率超过基本 V_{DD} 的 5%， V_{DD} 下降和恢复的比率不能超过 200 mV/s。

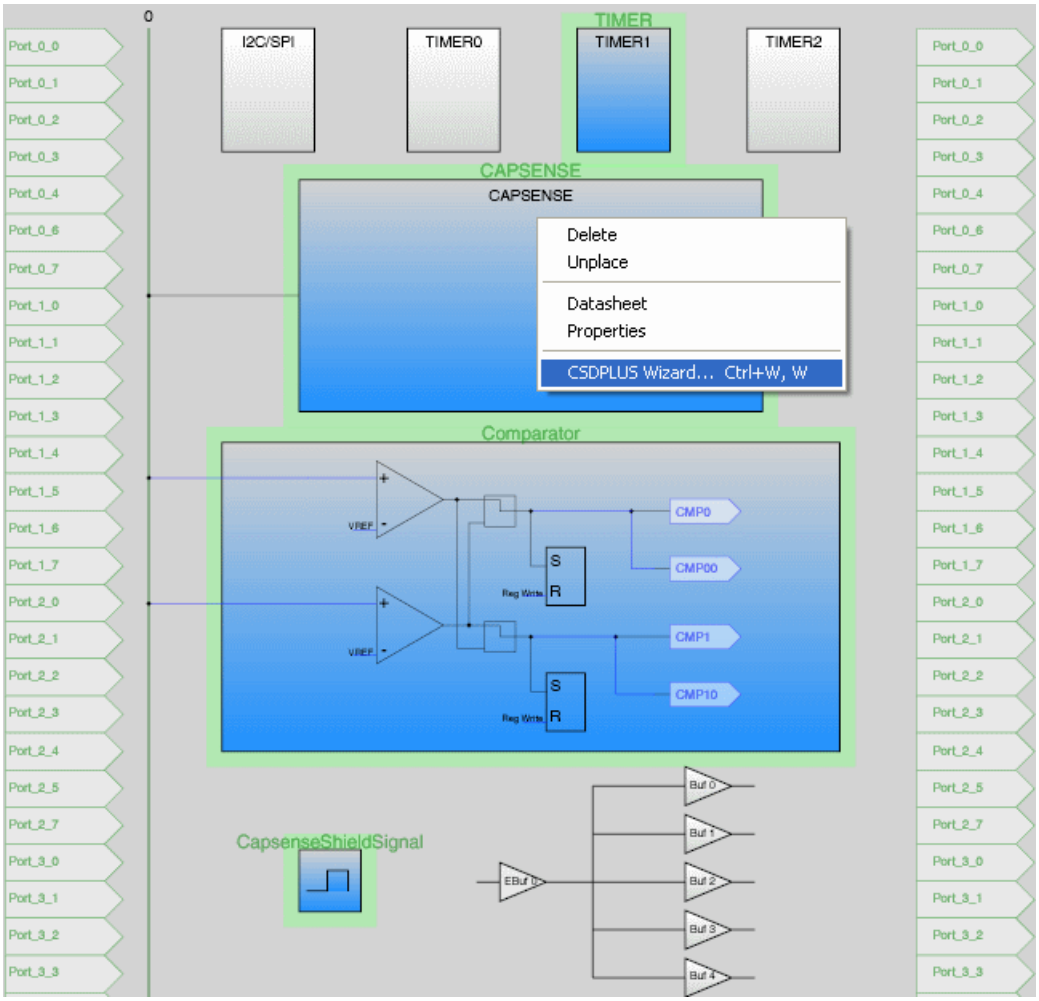
a. 最小绝对 V_{DD} 规格为 1.8 V。V_{DD} 下降到 1.8 V 以下会引入过多噪声。

放置

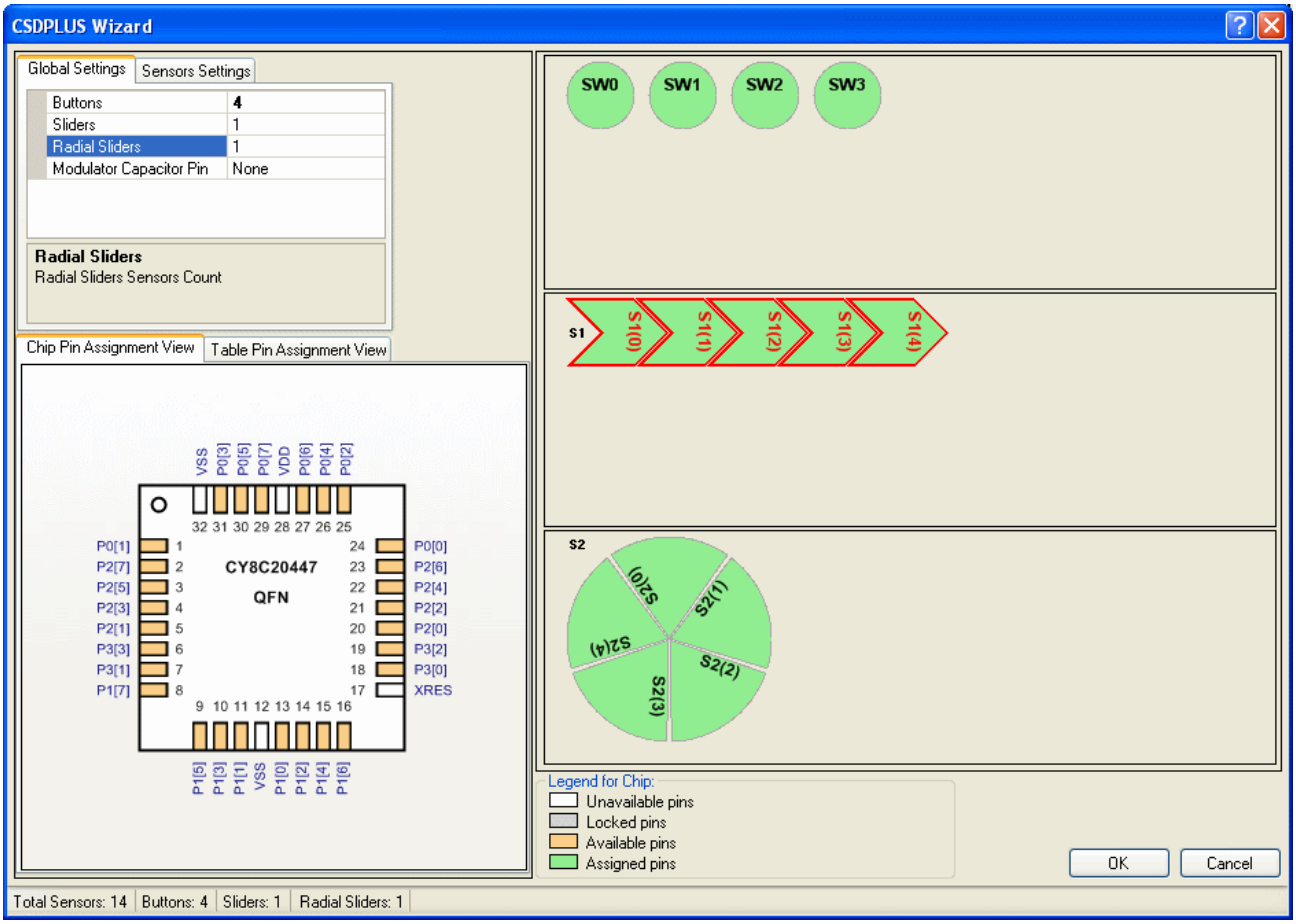
当实例化用户模块时、定时器 1、 CapSense、比较器和 CapSense 屏蔽信号模块将被分配到 CSDPLUS。备用放置不可用。在启动 CSDPLUS 向导之前必须对需要专用引脚资源 （包括 LCD 和 I2CHW）的用户模块进行放置。这样，在传感器映射到 CSDPLUS 向导中的 I/O 引脚时，专用引脚将被保留而且不能作为传感器无意激活。在放置电容传感器连接时，请勿使用 P1[0] 和 P1[1]。这些引脚用来对器件进行编程，而且有可能存在过大的布线电容值，从而会影响传感器的灵敏度。

CSDPLUS 向导

1. 要访问 CSDPLUS 向导，请在 “ 芯片编辑器 ” 中右键单击任意用户模块的子模块，然后通过鼠标左键单击选择 “CSDPLUS 向导 ”。



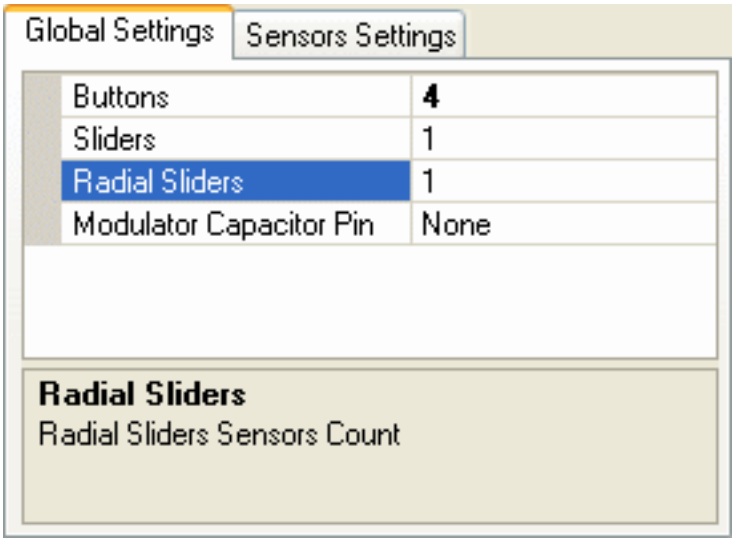
2. 向导打开，其中显示了传感器数、滑条数和辐射状滑条传感器数的数值输入框。



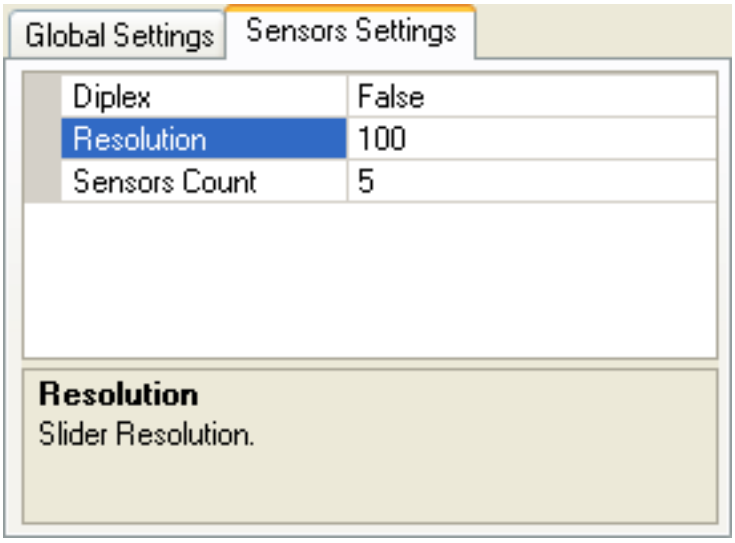
向导引脚图标

- 白色 — 该引脚不能作为 CapSense 输入使用。
- 灰色 — 由于两种原因中的一种造成引脚被锁定。第一种可能是另一个用户模块 （如 LCD 或 I2C）已占用了该引脚。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 ‘pinout view’ 视图中展开引脚，然后从 ‘Select’ 菜单中选择 ‘Default’ 。现在即可在向导中分配引脚。
- 橙色 — 可以分配引脚。
- 绿色 — 引脚已被分配为 CapSense 输入。

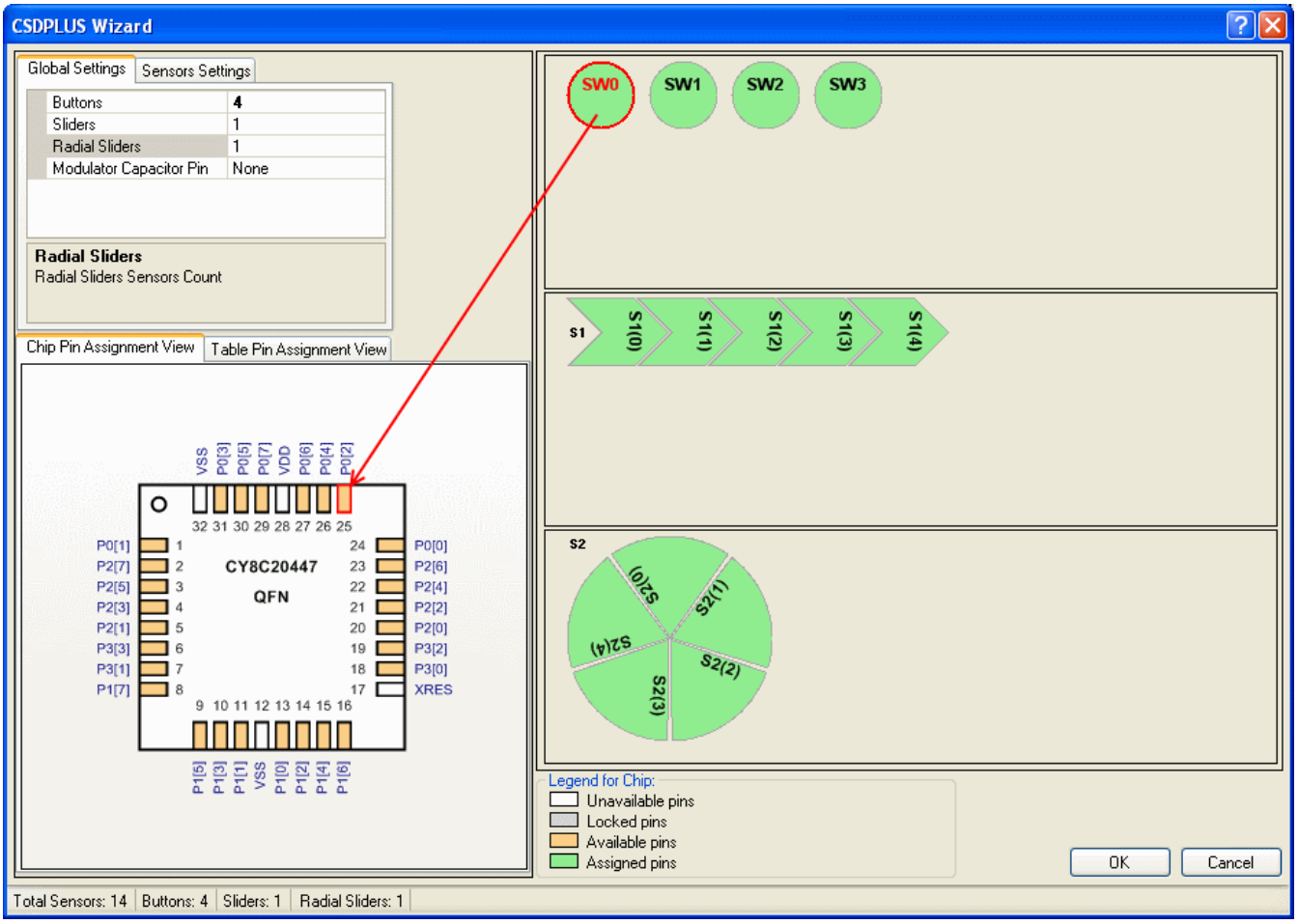
3. 选择 ‘Global Settings’ 选项卡以输入独立按键、滑条和径向滑条的数量。传感器 （按键和滑条元件）总数不得超过可用引脚数。输入数据后，按 [Enter] 键更新显示屏使其显示新值。
4. 选择调制器电容 （C_{mod}）。您可以选择 P0[1] 或 P0[3]。



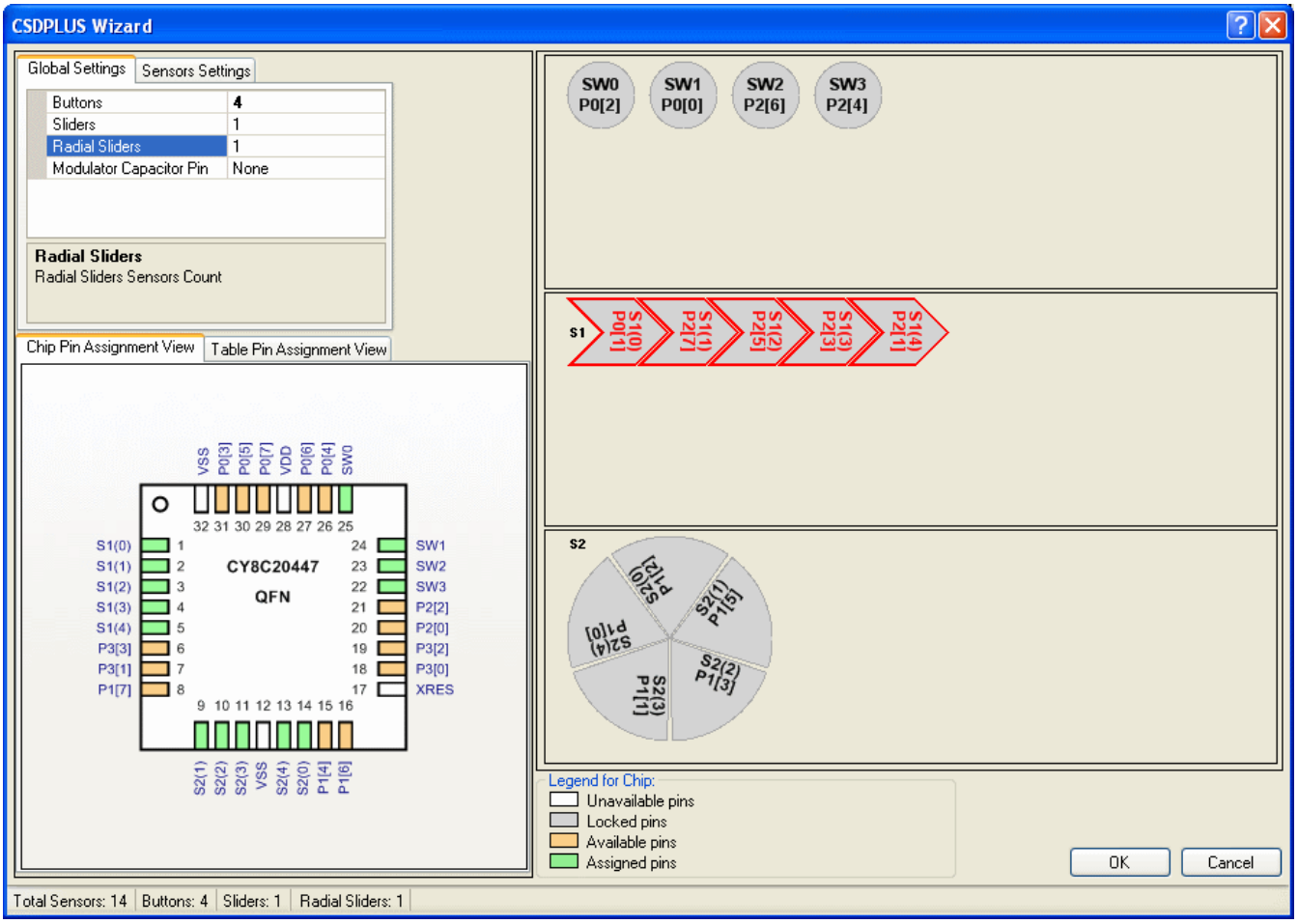
5. 选择 ‘Sensors Settings’ 选项卡即可设置滑条和辐射状滑条。要更改设置，请单击其中一个滑条以激活它。输入每个滑条中传感器元件的数量。滑条传感器中的传感器实际最小数量为五，最大值仅受限于引脚数量。输入数据后，按 [Enter] 键更新显示屏。



6. 键入输出分辨率。最小值为 5。 CSDPLUS 通过尝试使用相邻段的相对强度将触摸结果插入到指定的分辨率中。软件在滑条报告的触摸结果是零和分辨率 –1 之间。
7. 如果需要，选择 “ 双工 ”。这将把为传感器选定的引脚数值映射为板上传感器位置的两倍数值。仅显示了双工传感器的前半部分；后半部分按前面 “ 双工 ” 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。



8. 在引脚分配视图中通过将传感器拖到引脚上来给引脚分配传感器。您可以使用 ‘Chip Pin Assignment View’ 或 ‘Table Pin Assignment View’ 来进行此操作。选择端口引脚后，它会变为绿色，且不再可用。通过将端口引脚拖回未提交的表格，可改变传感器的分配情况。避免选择已经指定用于其他用户模块的引脚。
9. 在其他传感器上重复以上操作。单击确定接受数据，然后返回到 PSoC Designer。传感器放置现在已完成。右键单击 “ 器件编辑器 ” 窗口并选择刷新以更新引脚连接。



要想更改引脚的分配，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。引脚分配将被取消。

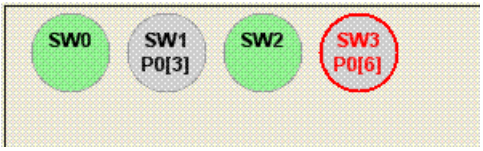
向导完成后，单击 “Generate Application” （生成应用）。根据您输入的传感器数量、引脚分配、双工和分辨率，一系列表格将被生成。

传感器的表示章节

本章节以图形方式表示在一个项目中所有可用的传感器类型。本章节描述了如何将传感器拖放到芯片和表格引脚分配视图内。所分配的传感器以灰色显示。本章节包括以下三个标题：

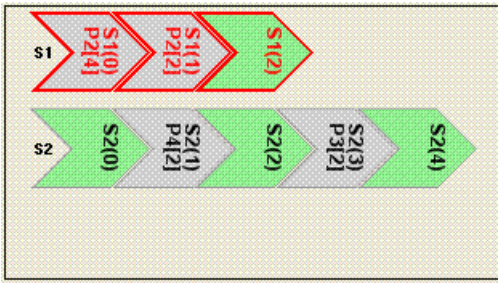
- 按键表示 — 传感器的按键会在向导中显示，如图 10 所示。每个按键传感器元件都有自己的标题。所有按键为芯片和表格引脚分配视图支持拖动和拖放功能。如果已经给按键分配，以分配的端口引脚数量在按键标题下显示。如果选择了按键传感器，将按键 widget 以红色的帧设置。

图 10. 按键表示章节



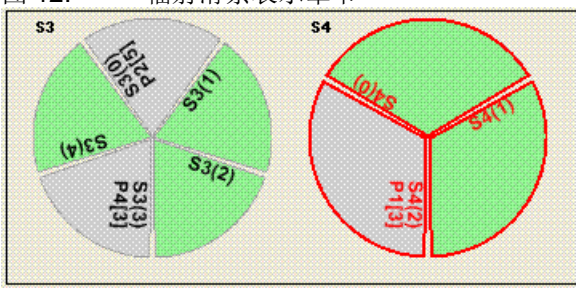
- 滑条表示 — 滑条被显示为传感器的段信号，如图 11 所示。某个滑条的所有段显示了特定传感器名称。滑条中的所有传感器均可在芯片和表格引脚分配视图中实现拖放操作。如果滑条段已被分配，每个段会显示被分配的端口引脚编号。如果已选择一个滑条，将滑条中的所有传感器以红色的帧设置。

图 11. 滑条表示章节



- 辐射滑条表示 — 辐射滑条被显示为一组段，如图 12 所示。在辐射滑条中的每个段显示了传感器名称。辐射滑条中的所有传感器均可在芯片和表格引脚分配视图中实现拖放操作。如果辐射滑条段已被分配，每个段会显示分配的端口引脚编号。如果一个辐射滑条已被选择，辐射滑条中的所有传感器以红色的帧设置。

图 12. 辐射滑条表示章节

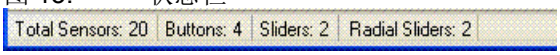


状态栏

状态栏显示有关设计的通用信息 （参见图 13）：

- 传感器总数 （Total Sensors）— 显示设计中所使用的传感器总数量。
- 按键 （Buttons）— 显示了设计中所使用的按键总数。
- 滑条 （Sliders）— 显示了设计中所使用的线性滑条总数。
- 辐射滑条 （Radial Sliders）— 显示设计中所使用的辐射滑条总数。

图 13. 状态栏



向导按键

该 CSDPLUS 用户模块向导提供了预定义功能的按键。

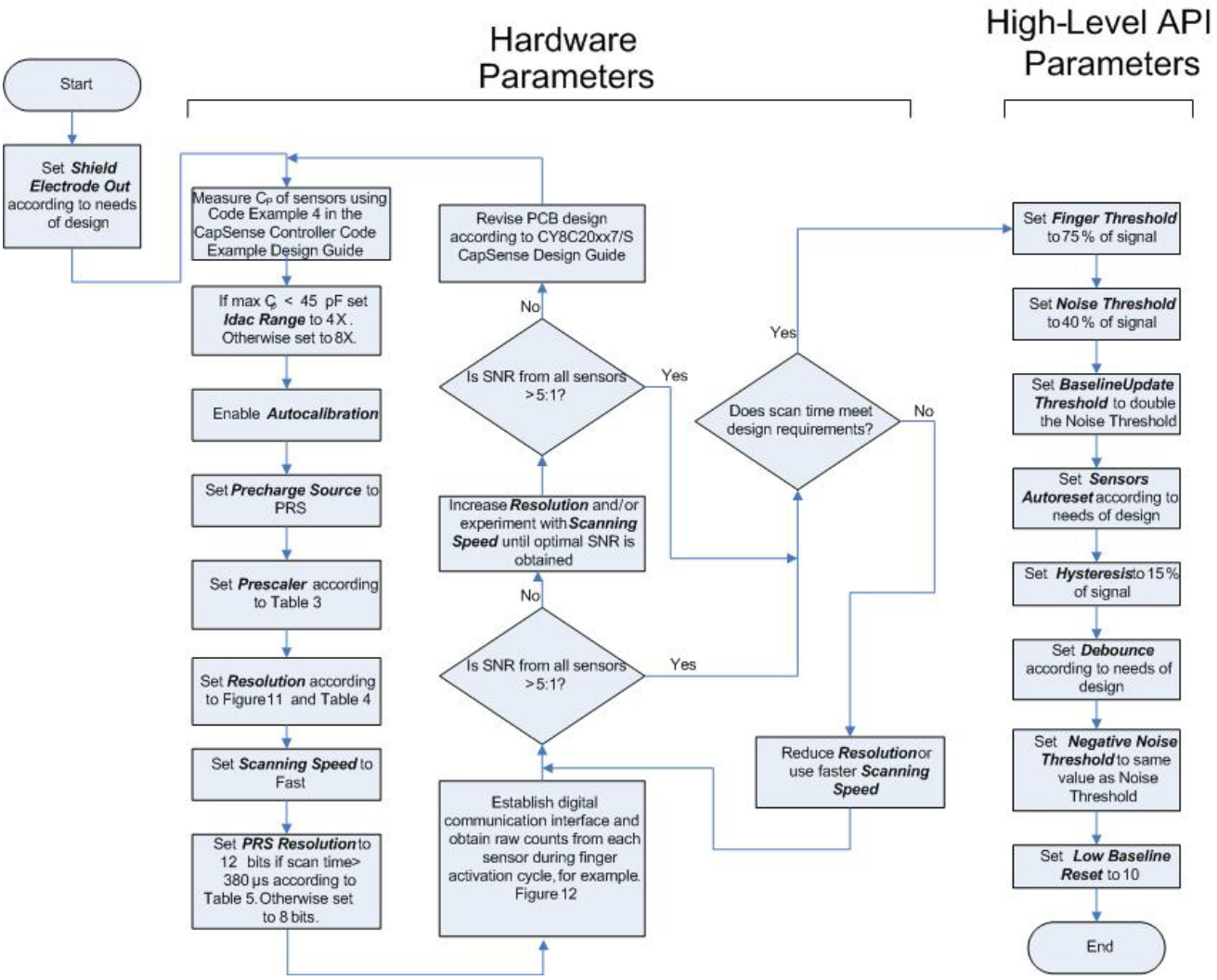
1. “OK” — 该按键检查以确定向导的参数是否正确以及是否已分配给所有传感器。如果正确，向导将保存参数并关闭；否则，它将显示了相应的警告信息，不保存参数，并保持打开状态。
2. “Cancel” — 该按键关闭向导而不会保存任何参数。
3. “Close” — 该按键是关闭窗口的标准按键，它位于向导右上角的标题栏。如果您点击关闭按键，则将关闭向导，而不会保存任何参数。
4. “Help” （帮助）— 该按键调用帮助页面以提供有关如何使用 CSDPLUS 用户模块向导的参考信息。它简要地描述 CSDPLUS 用户模块向导的特性。通过点击标准窗口帮助按键，将会打开 “Help” 页面。该按键标有问号，并位于向导右上角的标题栏。

用户模块参数 — 调试指南

在 CSDPLUS 向导中完成配置和 I/O 引脚分配后，必须设置用户模块参数。请注意：为了激活任何用户模块参数的更改，必须重新生成项目。

图 8 是展示 CSDPLUS UM 参数调试过程的流程图。这些参数可被分为两类：硬件和高级 API。这两种类别的参数以不同的方式影响电容式感应系统的性能，因此，该部分分别介绍了各个参数。然而，各个传感器的灵敏性之间存在互补关系，这取决于硬件参数设置和许多高级参数设置。更改任何硬件参数时，必须考虑这种情况，由此确保相应地调整相对的高级 API 参数。调试 CSDPLUS 用户模块参数应始终从硬件参数开始。

图 14. CSDPLUS 用户模块参数调试流程图



- 要想测量传感器 C_P 的大小，请参考 CapSense® 控制器代码示例设计指南
- 要想修改 PCB 设计，请参考 CY8C20xx7/S CapSense 设计指南

硬件参数

硬件参数配置硬件，这些硬件中，使用 CSD 方法用来把每个传感器的物理电容值转换为数字代码。此部分描述这些参数，并为如何根据硬件特征和其他参数调试提供指导。

默认情况下，硬件参数是设计中应用到所有 CapSense 传感器的全局设置。在各设计中，如果传感器（Cp）的寄生电容的总值或传感器的灵敏度在一个较大范围内浮动，全局硬件参数设置可能会不适合所有传感器。此类例子中，可以使用 SetPrescaler(i) 和 SetScanResolution(i) API 函数来为每传感器配置硬件参数，其中 (i) 是在调用 ScanSensor(i) API 函数之前的传感器索引。示例代码部分包含这样的示例。

iDAC 范围

通过该参数可以设置 iDAC 的输出范围。各选项为 4X 和 8X。对于少于 45 pF 的最大传感器 C_p 项目，可以使用 4X；否则，则使用 8X。默认值为 4X。

Autocalibration （自动校准）

该参数通过使能 iDAC 设置的逐次逼近来建立 85% 或 85% 以上的原始计数基准线。在 CSD 设计中，应将自动校准始终设置为激活状态。自动校准算法能力能否成功设置 iDAC 取决于预分频器的合理设置和 Cmod 是否处于推荐大小的状态。默认设置为 “Enable”（使能）。

iDAC 值

当自动校准禁用后，此参数将决定 iDAC 的当前输出。自动校准启用后，会使用推荐设定，该参数将被覆盖，并且失效。自动校准禁用后，提高该参数会降低原始计数基准线，反之亦然。该参数的默认设置为 20。

Compensation iDAC Value （补偿 iDAC 值）

当自动校准禁用后，此参数将决定补偿 iDAC 的当前输出。自动校准启用后，会使用推荐设定，该参数将被覆盖，并且失效。自动校准禁用后，提高该参数会降低原始计数基准线，反之亦然。默认设置为 0。

Precharge Source （预充源）

此参数选择传感器开关时钟源。可用选项是指预分频器，它使用分频后的 IMO 或 PRS。默认值为 PRS。PRS 可使分割的 IMO 时钟通过任意发生器，提供扩频时钟。PRS 提供高级降噪，释放更少的噪音，因而它也成为推荐预充源的默认设置。在某些情况下，预分频器预充源可提供更高的 SNR（信噪比）。然而，使用铜线路时，信噪比的提升通常比较细微，并且也不能明显看出是否益于前面提到的 PRS。

Prescaler （预分频器）

预分频器是应用于 IMO 以建立预充电时钟的分频器。合理调试 CSDPLUS 设计时，此为最重要的硬件 UM 参数。预分频器依据所选预充源、 IMO 和被扫描传感器的 Cp 值而定。表 3 显示了基于这些参数的预分频器推荐设置。默认设置为 2。

表 3. 基于预充源、 IMO 和 C_p 的预分频器设置

Cp （pF）	预充源 = PRS		
	预分频器 IMO = 24 MHz	预分频器 IMO = 12 MHz	预分频器 IMO = 6 MHz
<6	1	注意 1	注意 1
7 – 11	2	1	注意 1
12 – 15	2	1	注意 1
16 – 19	4	2	1
20 – 22	4	2	1
23 – 26	4	2	1
27 – 30	4	2	1
31 – 34	4	2	1
35 – 37	8	4	2
38 – 41	8	4	2
42 – 45	8	4	2
46 – 49	8	4	2
50 – 52	8	4	2
53 – 56	8	4	2
57 – 60	8	4	2

注意 1：不推荐使用这种预充源、预分频器和 C_p 的组合

Resolution （分辨率）

通过该参数设置 iDAC 的分辨率。可选范围为 9 到 16 位。提高分辨率会加强传感器的灵敏度，加大信噪比，并延长降噪的扫描时间。扫描分辨率为 n 时，最大原始计数值 （所有范围内）为 2^n-1。表 4 显示了基于 Cp 和手指电容值 Cf 的推荐分辨率设置。 Cf 是手指放置在传感器上时电容值的变化。C_f 值取决于外覆层厚度、传感器大小及传感器与其他大型导体的接近程度。图 9 显示了 C_f 值，它可作为外覆层厚度和圆形传感器直径的函数。该参数的默认设置为 12。

图 15. 基于外覆层厚度和圆形传感器直径的 （C_f）手指电容值

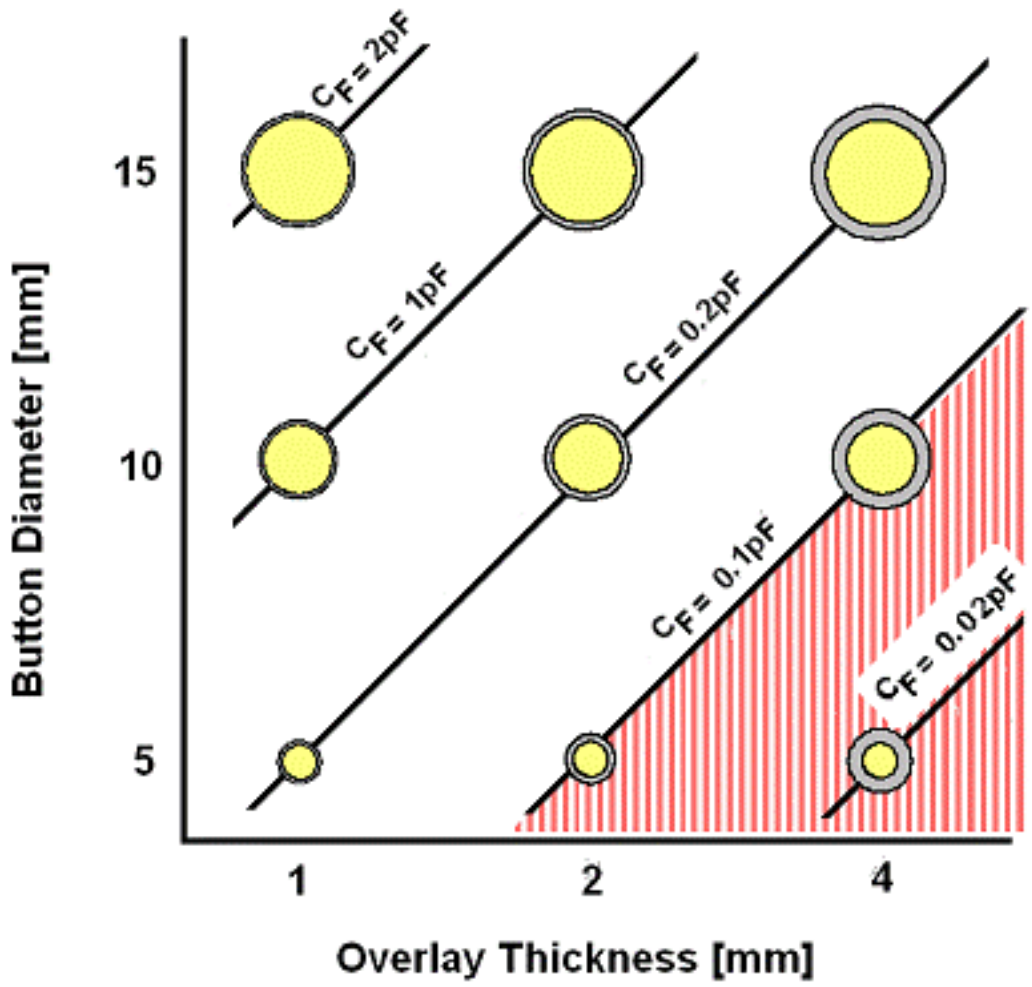


表 4. 基于手指电容值和 Cp 的分辨率设置

Cp （pF）	C_f = 0.1 pF	C_f = 0.2 pF	C_f = 0.4 pF	C_f = 0.8 pF
<6	12	11	10	9
7 - 12	13	12	11	10
13 - 24	14	13	12	11
25 - 48	15	14	13	12
>49	16	15	14	13

Scanning Speed （扫描速度）

该参数控制各个扫描结果的 LSB 积分时间。扫描速度选项包括：超快、快速、正常和慢速。建议初始值选择 “ 快速 ”。在某些情况下，较慢的扫描速度可以获得更高的信噪比，但扫描时间更长且功耗更大。该参数的默认设置为正常。表 5 显示了在不同分辨率和扫描速度下，单传感器的实际扫描时间 （单位为微秒）。

表 5. 单传感器在不同分辨率和扫描速度下的扫描时间 （µs）

分辨率 （位）	扫描速度			
	超快	快速	正常	慢速
9	110	130	175	260
10	130	175	260	430
11	175	260	430	780
12	260	430	780	1500
13	430	780	1500	2900
14	780	1500	2900	5600
15	1500	2900	5600	11250
16	2900	5600	11250	22500

PRS Resolution （PRS 分辨率）

此参数更改 PRS 序列的长度。可能的值为 8 和 12 位。相应的序列长度分别为 511 和 2047 输入时钟周期。如果需要极短的扫描时间，则必须使用 8 位 PRS 以避免过大噪声。扫描时间由分辨率 （不应该与 PRS 分辨率相互混淆）参数确定。对于扫描时间 ≤ 380 µs，将 PRS 分辨率设置为 8 位；对于扫描时间 > 380 µs，将 PRS 分辨率设置为 12 位。默认设置为 8 位。

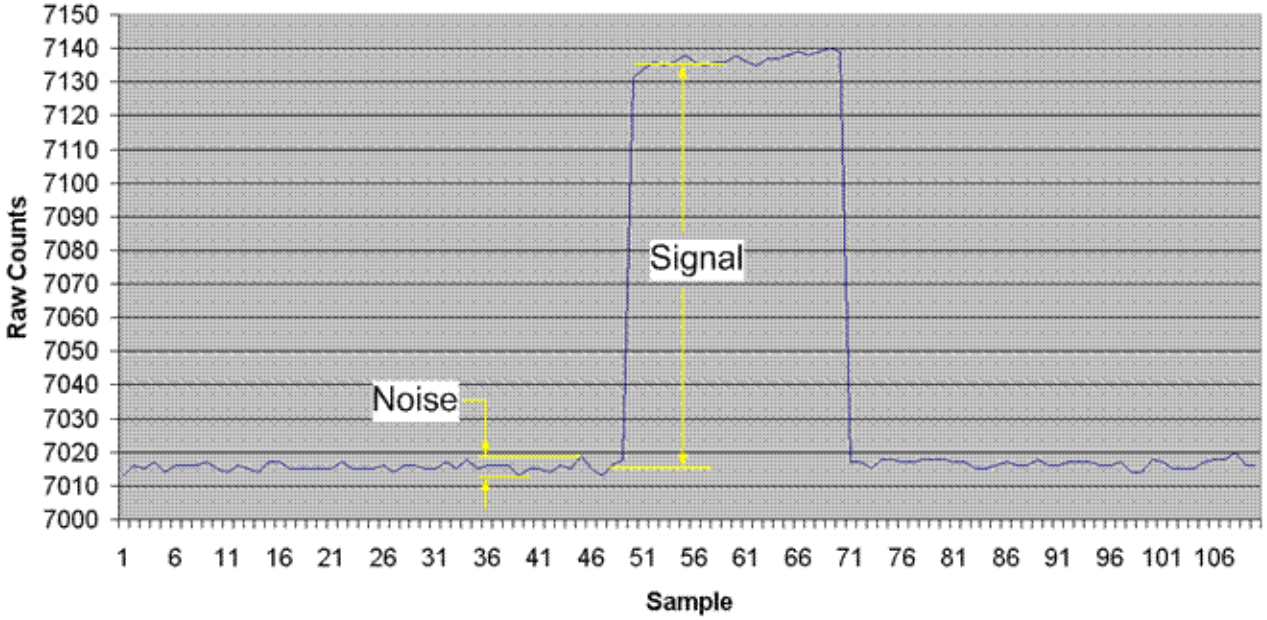
高级 API 参数

高层 API 参数决定高层固件算法行为，此算法用于区分传感器的激活与噪声，并补偿由环境因素引起的信号漂移。为了确定适当的参数取值，您必须在系统中建立起数字通信接口，以监控每个传感器手指激活事件期间的原始计数、基准线和差值。数据分别存储于如下三个数组：CSDPLUS_waSnsResult[]、CSDPLUS_waSnsBaseline[] 和 CSDPLUS_waSnsDiff[]。

如该数据所示，高层 API 参数的设置主要基于环境噪声和手指信号强度。噪声与信号强度取决于 EMI 环境、 PCB 布局、覆盖层厚度及系统的其他特性。因此，该数据作为参数设置基础，须从系统的原始位置中获取，并且该系统须处于最终装配状态。此外，您使用的 EMI 环境必须与将来实际使用的相同。

图 10 显示了传感器在一个手指激活周期中获取的典型原始计数值 （即传感器被激活再取消激活）。叠加在数据上的标签说明了如何根据原始数据计算噪声与信号。在适当情况下，下面的高层参数描述包括有关如何根据噪声和信号值进行设置每参数的信息。为实现 CapSence 系统稳定地操作，信噪比 （SNR）至少要为 5:1。若信噪比低于 5:1，则需要调整硬件参数或更改 PCB 布局，至少将信噪比提高至 5:1。

图 16. 传感器在手指激活周期中的典型原始计数值



Finger Threshold （手指阈值）

手指阈值确定原始计数的标称变化 （不包含迟滞），用于激活传感器。可能值的范围为从 5 到 255。默认情况下，通过 CSD_SetDefaultFingerThresholds() API 函数，全局手指阈值适用于所有传感器。对于单个按键 （滑条组中不包含），通过将所需要的值写入到 CSD_baBtnFThreshold[] 全局阵列中相应于传感器编号的索引位置内可以设置按键的手指阈值。为了设置该参数，您必须分析每个传感器对手指激活的原始计数响应。当极限 （最小手指）触摸该传感器时，应该将每个传感器的手指阈值设置为原始计数信号的 75% （参见图 10）。对于作为滑条元素的传感器，手指阈值没有任何意义或功能。该参数的默认设置为 60。

Noise Threshold （噪声阈值）

如果原始计数的积极变化低于该等级，它将被累加，以更新原始计数基准线。值范围为 5 到 255。对于按键传感器，当传感器被自动复位为禁用 （默认）状态时，超过该阈值的计数值不会更新基准线；而对于滑条传感器，质心计算将不包含低于该阈值的计数值。噪声阈值是可用于所有传感器的全局参数。噪声阈值的良好起点是原始计数信号的 40% （请参考图 10）。默认设置为 10。

BaselineUpdate Threshold （基准线更新阈值）

CSDPLUS 使用所谓的 “ 水桶 ” 方法来更新 CSDPLUS_UpdateSensorBaseline() API 函数中的基准线计数。当满足这些条件时，原始计数值和基准线计数值之间的半个差值被添加到 “ 水桶 ”：

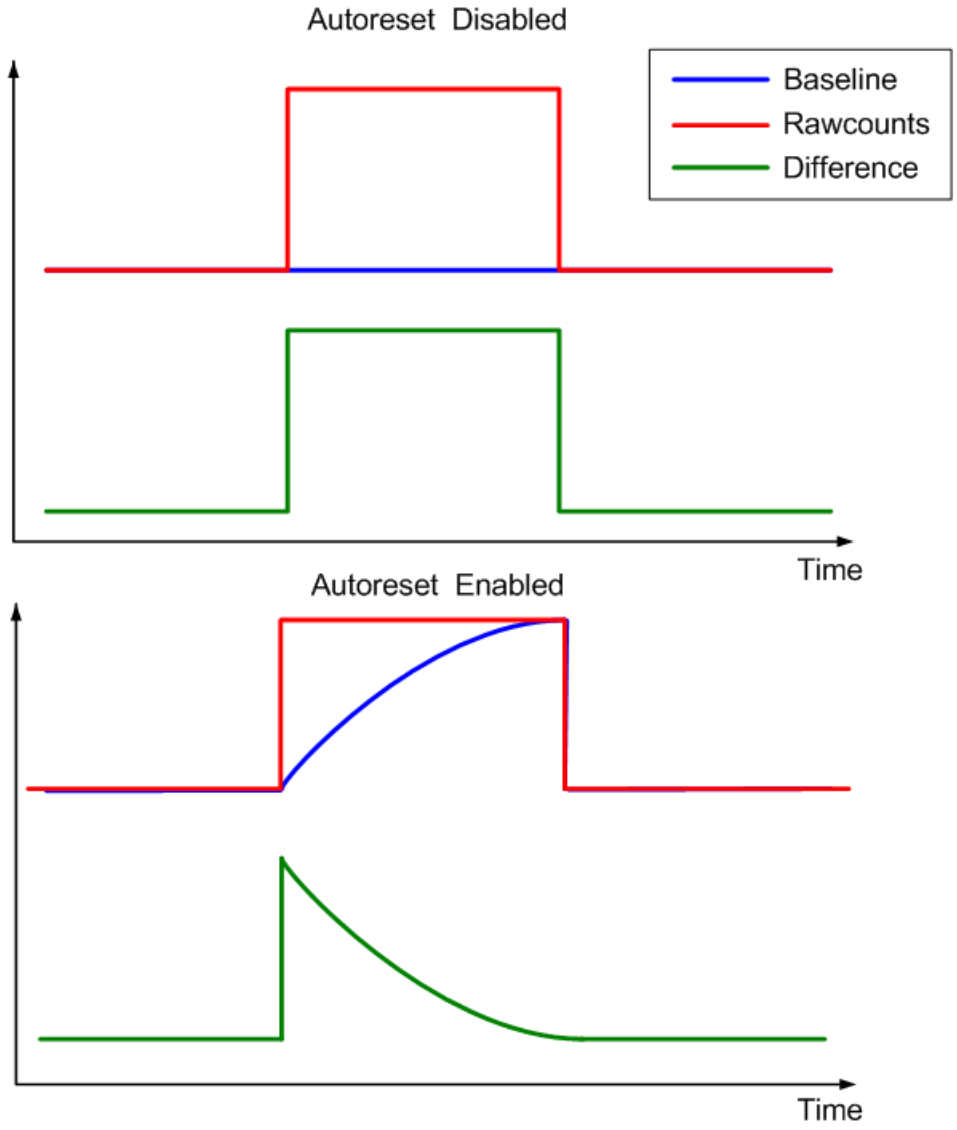
1. 从扫描返回的原始计数值高于当前基准线，与及。
2. 原始计数值和基准线之间的差值低于噪声阈值。

基准线更新阈值参数设置 “ 水桶 ” 必须达到的值，以增加基准线。该参数的良好起点是噪声阈值的两倍。取值范围为 0 到 255 ；默认值为 100。

Sensors Autoreset （传感器自动复位）

该参数确定基准线得到更新的时间：随时更新或只当信号差值低于噪声阈值时才更新。当此参数设置为 “ 禁用 ”，则仅当原始计数与基准线的差值低于噪声阈值时，基准线才进行更新。图 11 说明了此参数对基准线更新的影响。当 “Sensors Autoreset” 设置为 “Enabled” （使能）时，基准线始终被更新，无论噪声阈值如何。此设置限制传感器的最大激活时间 （通常为 5 到 10s）。但对防止传感器因非触摸引起的原始计数突然上升而被停滞带来了好处。原始计数的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。当 “Sensors Autoreset” （传感器自动复位）设置为禁用，则仅当原始计数与基准线之间的差值低于噪声阈值时，基准线才进行更新。应该将该参数设置为 “Disabled” （禁用）的默认状态。有关该参数的更多信息，请参见附件。

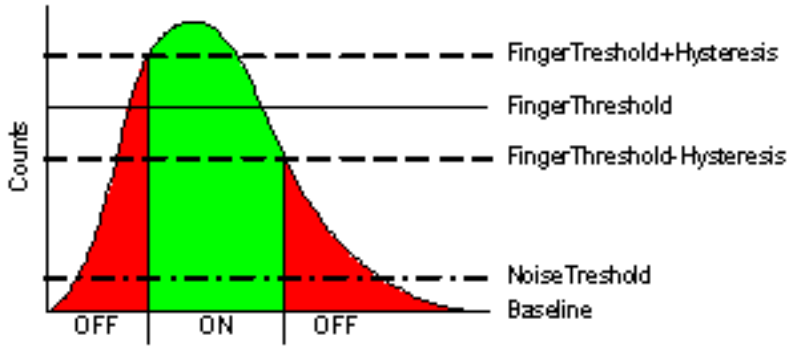
图 17.　传感器自动复位参数在基准线上的影响



Hysteresis （迟滞）

为了提高按键传感器的激活状态识别并提供更稳定的操作，迟滞将传感激活状态依次设置为 OFF — ON — OFF （请参考图 12）。计数值必须大于手指阈值 + 迟滞，以将状态从 OFF 修改为 ON。计数值必须小于手指阈值 — 迟滞，以将状态从 ON 修改为 OFF。迟滞的良好起点是原始计数信号的 15%（参考图 10）。默认设置为 10。

图 18. 在按键传感器状态中手指阈值和迟滞的关系



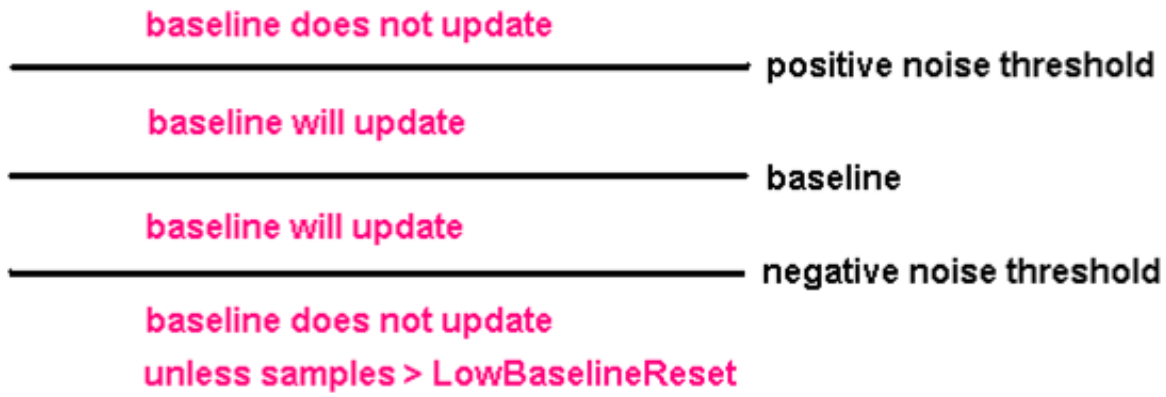
Debounce （去抖动）

该参数为传感器活动的瞬变添加了去抖动计数器。为了让传感器能够从未激活状态切换为激活状态，在该参数指定的样本数量内，差异计数值必须大于手指阈值 + 迟滞。去抖动计数器由 CSDPLUS_bIsSensorActive 或 CSDPLUS_bIsAnySensorActive API 函数递增。值范围为 1 到 255。如果设置为‘1’，则没有去抖动但会提供最快的响应。默认设置为 3。

Negative Noise Threshold （负噪声阈值）

当原始计数低于基准线的计数时，该参数被使用。负噪声阈值建立的级别与当前基准线相对应，如果超过了这个级别，则基准线将复位为原始信号值。如果原始计数低于该级别，基准线将不复位，除非达到 Low Baseline Reset （低基准线复位）参数的限制。在这种情况下，将复位基准线。图 13 显示的是噪声阈值和基准线复位之间的关系。负噪声阈值的良好起点是使用噪声阈值。默认设置为 10。

图 19. 噪声阈值和基准线复位之间的关系



Low Baseline Reset （低基准线复位）

该参数与负噪声阈值一起使用，以定义复位基准线到原始计数必须经过的采样次数，这种采样中的原始计数必须小于基准线值。对于由低基准线复位参数指定的采样数，如果原始计数值小于基准线值与负噪声阈值之差，则基准线值将下降到原始计数值。低基准线复位通常用来纠正启动时手指已经在传感器上面的情况。建议初始值选择 10 ；默认设置为 50。

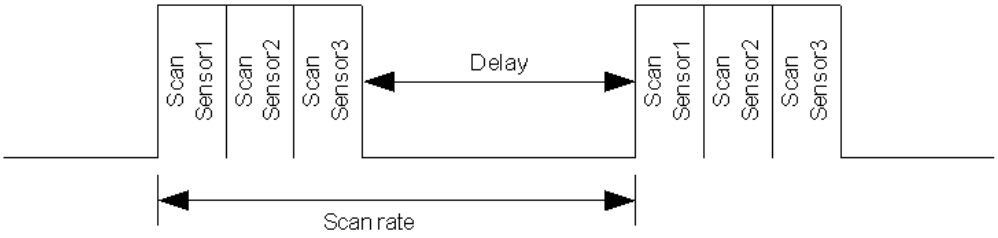
FMEA_Cp_Range_Test

通过该参数可以使能 GetSnsParasiticCapacitance API。它添加 GetSnsParasiticCapacitance、Calibrate、 div_24_16_24 和 mul_16x16_32 函数的有条件编译。

传感器扫描速率选择的指南

扫描速率是指传感器被扫描的速率。下图显示的是一个 3 按键设计的示例。设计中的所有传感器按顺序进行扫描，而且在各个扫描之间有一个延迟。

图 20. 典型传感器扫描



为了确保基准线正常运行，建议保持设计的扫描速率为 15 ms 或 15 ms 以上。这意味着具有更少传感器的设计必须添加一个延迟，以使传感器扫描速率不低于 15 ms。具有更多传感器的设计不需要任何延迟，因为扫描所有传感器本身已消耗 15 ms 了。良好的设计会使 CapSense 控制器 （而不是使固件延迟子程序）进入睡眠模式，以节能。

应用编程接口

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

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 CSDPLUS_1。可将该值更改为符合标识符语法规则的任意唯一值。分配的实例名称成为每个全局函数名称、变量和常量符号的前缀。在下面的说明中，为了简单起见，实例名称缩写为 CSDPLUS。

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 语言编译器自动处理此项要求。汇编语言编程人员也必须确保其代码遵守该策略。虽然一些用户模块 API 函数可以保持 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 CSDPLUS，启动其采样，并停止 CSDPLUS。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 CSDPLUS 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。请勿手动更改这些阵列。不过，您可以出于调试目的对这些值进行检查。例如，可以使用绘图工具显示数组的内容。以下是几个全局阵列：

- CSDPLUS_waSnsResult[]
- CSDPLUS_waSnsBaseline[]
- CSDPLUS_waSnsDiff[]
- CSDPLUS_baSnsOnMask[]
- CSDPLUS_baBtnFThreshold[]
- CSDPLUS_baDAC[]
- CSDPLUS_baCompensationDAC[]

CSDPLUS_waSnsResult[] ：该整数阵列被 CSDPLUS_ScanSensor() 函数用来储存每个实际传感器扫描的原始计数。阵列大小与传感器数量相等。

CSDPLUS_waSnsBaseline[] ：这是一个整数阵列，其中包含每个传感器的基准数据。阵列大小与传感器数量相等。通过下列函数更新 CSDPLUS_waSnsBaseline[] 阵列：

- CSDPLUS_UpdateAllBaselines() ；
- CSDPLUS_UpdateSensorBaseline() ；
- CSDPLUS_InitializeBaselines()。

CSDPLUS_waSnsDiff[] ：该整数阵列包含每个传感器中原始数据与基准数据之间的差值。阵列大小与传感器数量相等。

CSDPLUS_baSnsOnMask[] ：这是一个字节阵列，用于保持传感器的开 / 关状态 （针对按键或滑条传感器）。 CSDPLUS_baSnsOnMask[0] 包含传感器 0 到 7 的掩码位 （传感器 0 的是位 0，传感器 1 的是位 1）。 CSDPLUS_baSnsOnMask[1] 包含传感器 8 到 15 的掩码位 （如果需要），等。此字节阵列包含的元素数足以包含所有放置的传感器。按键开启时位值为 1，关闭时位值为 0。 CSDPLUS_baSnsOnMask[] 数据由 CSDPLUS_bIsSensorActive(BYTE bSensor) 函数或 CSDPLUS_bIsAnySensorActive() 子程序更新。

CSDPLUS_baBtnFThreshold[] ：这是用来为每个传感器储存阈值的字节阵列。阵列大小与传感器数量相等。

CSDPLUS_baDAC[] ：这是用来为每个实际传感器储存自动校准 IDAC 值的字节阵列。每传感器的值由 CSDPLUS_CalibrateSensor() 函数设置以及由 CSDPLUS_ScanSensor() 函数使用。阵列大小与传感器数量相等。

CSDPLUS_baCompensationDAC[] ：它是一个字节阵列，用于存储同每个传感器相应的补偿 IDAC 值。阵列大小与传感器数量相等。

CSDPLUS_Start

说明：

该函数启动了 CSDPLUS 用户模块。该函数可初始化全局变量、配置 Cmod 并将它连接到 amux 总线，以及配置 CapSense 模块和相关硬件。应当在调用任何其他用户模块函数之前，先调用此函数。

依据 “ 补偿 iDAC 值 ” 中的用户模块参数，该 API 函数将使用用户键入的值填充到 CSDPLUS_baCompensationDAC[] 内。

C 原型：

```
void  CSDPLUS_Start(void)
```

汇编：

```
lcall CSDPLUS_Start
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_Stop

说明：

该函数停止传感器扫描仪、禁用内部中断并调用 CSDPLUS_ClearSensors() 以将所有传感器复位为非活动状态。

C 原型：

```
void  CSDPLUS_Stop(void)
```

汇编：

```
lcall CSDPLUS_Stop
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_Resume

说明：

调用 CSDPLUS_Stop 之后，该函数对用户模块操作进行恢复。

C 原型：

```
void  CSDPLUS_Resume(void)
```

汇编：

```
lcall CSDPLUS_Resume
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_ScanSensor

说明：

该函数扫描已选的传感器。由 CSDPLUS 向导按顺序所分配的传感器阵列中的每个传感器都有唯一的一个编号。 Sw0 为传感器 0， Sw1 为传感器 1，依此类推。

C 原型：

```
void  CSDPLUS_ScanSensor(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDPLUS_ScanSensor
```

参数：

bSensor — 范围是 0 到 （n – 1），其中 ‘n’ 是 CSDPLUS 向导中设置的传感器数量与滑条中包括的传感器数量之和。 CSDPLUS_wGetPortPin() 使用传感器编号来确定所选的活动传感器的端口和位掩码。

返回值：

无

其他影响

**

CSDPLUS_ScanAllSensors

说明：

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

C 原型：

```
void  CSDPLUS_ScanAllSensors(void)
```

汇编：

```
lcall CSDPLUS_ScanAllSensors
```

参数：

无

返回值：

无

其他影响

**

CSDPLUS_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准线。通过使用带有以下算法的水桶方法，对该基准线进行更新：

1. 每次调用 CSDPLUS_UpdateSensorBaseline()，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 CSDPLUS_waSnsDiff[] 阵列中，并为您提供的。

2. 如果禁用了 “Sensors Autoreset”，则每次调用 CSDPLUS_UpdateSensorBaseline()，将对差值与噪声阈值进行比较。如果差值低于噪声阈值，将被累加到虚拟的水桶中。如果差值高于噪声阈值，则不更新水桶。如果使能了 Sensors Autoreset，则无论噪声阈值参数如何，差值都会累加到虚拟水桶中。

3. 虚拟水桶中的累计差值计数达到 BaselineUpdateThreshold 后，基准线会按 1 递增，且水桶将复位为 0。

4. 如果差值计数低于噪声阈值，则保留在 CSDPLUS_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  CSDPLUS_UpdateSensorBaseline(BYTE bSensorNum)
```

汇编：

```
mov   A, bSensorNum
lcall CSDPLUS_UpdateSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

其他影响：

**

CSDPLUS_UpdateAllBaselines

说明：

该函数使用 CSDPLUS_bUpdateSensorBaseline() 函数以对所有传感器的基准线进行更新。

C 原型：

```
void CSDPLUS_UpdateAllBaselines(void)
```

汇编：

```
lcall CSDPLUS_UpdateAllBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_bIsSensorActive

说明：

该函数检查给定传感器与手指阈值之间的差值计数阵列。计算迟滞值：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，则提高该阈值。该函数还更新了 CSDPLUS_baSnsOnMask[] 阵列中的传感器的位。

C 原型：

```
BYTE  CSDPLUS_bIsSensorActive(BYTE bSensorNum)
```

汇编：

```
mov   A, bSensorNum
lcall CSDPLUS_bIsSensorActive
```

参数：

bSensorNum — 范围是 0 到 （n – 1），其中 ‘n’ 是 CSDPLUS 向导中设置的传感器数量与滑条中包括的传感器数量之和。 CSDPLUS_wGetPortPin() 使用传感器编号来确定所选的活动传感器的端口和位掩码。

返回值：

BYTE：1 — 已选的传感器处于活动状态； BYTE 0 — 已选的传感器处于非活动状态。

其他影响：

**

CSDPLUS_bIsAnySensorActive

说明：

该函数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 CSDPLUS_bIsSensorActive()，以便在调用该函数后更新 CSDPLUS_baSnsOnMask[] 阵列。

C 原型：

```
BYTE  CSDPLUS_bIsAnySensorActive(void)
```

汇编：

```
lcall CSDPLUS_bIsAnySensorActive
```

参数：

无

返回值：

BYTE：1 — 表示一个或多个传感器处于活动状态； BYTE 0 — 没有传感器处于活动状态。

其他影响：

**

CSDPLUS_wGetCentroidPos

说明：

该函数为质心检查差值阵列。如果存在，则偏移和长度都存储为临时变量，并根据 CSDPLUS 向导中指定的分辨率计算质心位置。仅当滑条是由 CSDPLUS 向导定义时，此函数才可用。

C 原型：

```
WORD  CSDPLUS_wGetCentroidPos(BYTE bSnsGroup)
```

汇编：

```
mov   A, bSnsGroup
lcall CSDPLUS_wGetCentroidPos
```

参数：

bSnsGroup => 组编号

该参数作为滑条的特定传感器的参考。组 0 用于按键。滑条包含在组 1 和更高的组中。

返回值：

WORD：滑条的位置，即 LSB 位于 A 中以及 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。每次扫描后必须只调用一次子程序，以避免得到负的差值。如果应用监控差值信号，则在差值计数数据传输后调用此子程序。

如果某个滑条传感器处于活动状态，此函数会将返回数值为从零到 CSDPLUS 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，该函数将返回 –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）。若需要，可以使用 CSDPLUS_blsSensorActive() 子程序确定触摸了哪些滑条段。

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

CSDPLUS_wGetRadialPos

说明：

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

C 原型：

```
WORD  CSDPLUS_wGetRadialPos(BYTE bSnsGroup)
```

汇编：

```
mov   A, bSnsGroup
lcall CSDPLUS_wGetRadialPos
```

参数：

bSnsGroup => 组编号

此参数是您正在使用的辐射滑条的编号。通过在辐射滑条表示法中左侧上的 CSDPLUS UM 向导可以找到该编号 （比如，对于 s2，辐射滑条编号为 2）。

返回值：

WORD：滑条的位置，即 LSB 位于 A 中以及 MSB 位于 X 中。

其他影响：

每次扫描后必须只调用一次子程序，以避免得到负的差值及更新基准线。如果应用监控差值信号，则在差值计数数据传输后调用此子程序。

如果某个滑条传感器处于活动状态，此函数会将返回数值为从零到 CSDPLUS 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）。

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

CSDPLUS_wGetRadialInc

说明：

该函数返回实际手指移位情况，即手指的当前位置与先前位置之间的差值。此函数与 CSDPLUS_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  CSDPLUS_wGetRadialInc(BYTE bSnsGroup)
```

汇编：

```
mov   A, bSnsGroup
lcall CSDPLUS_wGetRadialInc
```

参数：

bSnsGroup => 组编号

此参数是您正在使用的辐射滑条的编号。通过在辐射滑条表示法中左侧上的 CSDPLUS UM 向导可以找到该编号 （比如，对于 s2，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 中以及 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 CSDPLUS_wGetRadialPos() 将返回 -1 （FFFFh）；如果当前没有任何触摸，则此时 CSDPLUS_wGetRadialPos() 会返回 -1 （FFFFh）。

其他影响：

仅在调用 CSDPLUS_wGetRadialPos() API 之后，才能调用此子程序。因为它使用由 CSDPLUS_wGetRadialPos() 设置的 CSDPLUS_waSliderPrevPos 和 CSDPLUS_waSliderCurrPos 内部数据。

CSDPLUS_InitializeSensorBaseline

说明：

该函数通过扫描所选的传感器，将初始值加载到 CSDPLUS_waSnsBaseline[bSensorNum] 阵列元素中。原始计数值将复制到所选传感器的基准线阵列元素中。此函数可用于对单个传感器的基准线值进行复位。

C 原型：

```
void  CSDPLUS_InitializeSensorBaseline(BYTE bSensorNum)
```

汇编：

```
mov   A, bSensorNum
lcall CSDPLUS_InitializeSensorBaseline
```

参数：

bSensorNum — 范围是 0 到 （n – 1），其中 ‘n’ 是 CSDPLUS 向导中设置的传感器数量与滑条中包括的传感器数量之和。 CSDPLUS_wGetPortPin() 使用传感器编号来确定所选的活动传感器的端口和位掩码。

返回值：

无

其他影响：

**

CSDPLUS_InitializeBaselines

说明：

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

C 原型：

```
void  CSDPLUS_InitializeBaselines(void)
```

汇编：

```
lcall CSDPLUS_InitializeBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_SetDefaultFingerThresholds

说明：

通过 FingerThreshold （手指阈值）参数值，该函数进行加载 CSDPLUS_baBtnFThreshold[] 阵列。如果尚未将自定义值手动地加载到 CSDPLUS_baBtnFThreshold[] 阵列，则必须在扫描之前调用该函数。

C 原型：

```
void  CSDPLUS_SetDefaultFingerThresholds(void)
```

汇编：

```
lcall CSDPLUS_SetDefaultFingerThresholds
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_SetScanMode

说明：

该函数设置扫描速度和分辨率。可以在运行时调用此函数来更改扫描速度和分辨率。此函数会覆盖用户模块参数的设置。将该函数用来设置扫描速度、分辨率并可以在运行时调用此函数来更改扫描速度和分辨率。此函数会覆盖用户模块参数的设置。如果需要以不同的速度和分辨率对传感器进行扫描，

那么要使用该函数。例如：常用按键和接近感应检测器。可以用 9 位分辨率扫描常规按键。接近检测器经常可以采用比 16 位略低的分辨率进行扫描，对于大范围检测，扫描时间较长。此函数可与 CSDPLUS_ScanSensor() 函数结合使用。

C 原型：

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

汇编：

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

参数：

bSpeed — 设置扫描速度；下表列出了合适的数值。

bResolution — 设置扫描分辨率；下表列出了合适的值。

名称	数值
扫描速度值	
CSDPLUS_ULTRA_FAST_SPEED	0
CSDPLUS_FAST_SPEED	1
CSDPLUS_NORMAL_SPEED	2
CSDPLUS_SLOW_SPEED	3
分辨率值	
CSDPLUS_9_BIT_RESOLUTION	9
CSDPLUS_10_BIT_RESOLUTION	10
CSDPLUS_11_BIT_RESOLUTION	11
CSDPLUS_12_BIT_RESOLUTION	12
CSDPLUS_13_BIT_RESOLUTION	13
CSDPLUS_14_BIT_RESOLUTION	14
CSDPLUS_15_BIT_RESOLUTION	15
CSDPLUS_16_BIT_RESOLUTION	16

返回值：

无

其他影响：

**

CSDPLUS_SetSliderIdac

说明：

该函数将滑条元素的 iDAC 电流设置为每个滑条组的最高值。只有 “AutoCalibration” 用户属性被设置为 “Enabled” （使能），该函数才可用。

C 原型：

```
void  CSDPLUS_SetSliderIdac(void)
```

汇编：

```
lcall CSDPLUS_SetSliderIdac
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_SetIdacValue

说明：

此函数覆盖用户模块参数设置 iDAC 值。如果需要用其他 iDAC 设置扫描某些传感器，则使用此函数。此函数可与 CSD_ScanSensor() 函数结合使用。

C 原型：

```
void  CSDPLUS_SetIdacValue(BYTE bIdacValue, BYTE bCompensationIDACValue)
```

汇编：

```
mov   A, bIdacValue
mov   X, bCompensationIDACValue
lcall CSDPLUS_SetIdacValue
```

参数：

bIdacValue — 设置 iDAC 值。该值的有效范围是 0 到 127。

bCompensationIdacValue — 设置 IDAC 补偿值。值的有效范围是 0 到 127。

返回值：

无

其他影响：

**

CSDPLUS_SetPrescaler

说明：

此函数覆盖预分频器用户模块参数的值。如果需要用预分频器设置扫描某些传感器，则使用此函数。

C 原型：

```
void  CSDPLUS_SetPrescaler(BYTE bPrescaler)
```

汇编：

```
mov   A, bPrescaler
lcall CSDPLUS_SetPrescaler
```

参数：

bPrescaler — 设置预分频器的值。下表列出了合适的值：

名称	数值	预分频器
CSDPLUS_PRESCALER_1	0x00	1
CSDPLUS_PRESCALER_2	0x01	2
CSDPLUS_PRESCALER_4	0x02	4
CSDPLUS_PRESCALER_8	0x03	8
CSDPLUS_PRESCALER_16	0x04	16
CSDPLUS_PRESCALER_32	0x05	32
CSDPLUS_PRESCALER_64	0x06	64
CSDPLUS_PRESCALER_128	0x07	128
CSDPLUS_PRESCALER_256	0x08	256

返回值：

无

其他影响：

**

CSDPLUS_CalibrateSensors

说明：

该函数会调整 iDAC 电流以使获取的原始计数接近 wValue 值，并将结果存储在 CSDPLUS_baDAC[] 和 CSDPLUS_baCompensationDAC[] 全局阵列中。只有 “AutoCalibration” 用户属性被设置为 “Enabled”（使能），该函数才可用。

C 原型：

```
void  CSDPLUS_CalibrateSensors(WORD wValue)
```

汇编：

```
mov   A, [wValue+1]
mov   X, [wValue]
lcall CSDPLUS_CalibrateSensors
```

参数：

WORD：wValue — 目标原始数据值。

返回值：

无

其他影响：

**

CSDPLUS_ClearSensors

说明：

通过为每个传感器依次调用 CSDPLUS_wGetPortPin() 和 CSDPLUS_DisableSensor()，该函数将所有的传感器清除为非采样状态。

C 原型：

```
void  CSDPLUS_ClearSensors(void)
```

汇编：

```
lcall CSDPLUS_ClearSensors
```

参数：

无

返回值：

无

其他影响：

**

CSDPLUS_wReadSensor

说明：

该函数返回 A （LSB）和 X （MSB）中的关键原始扫描值。

C 原型：

```
WORD  CSDPLUS_wReadSensor(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDPLUS_wReadSensor
```

参数：

bSensor — 范围是 0 到 （n – 1），其中 ‘n’ 是 CSDPLUS 向导中设置的传感器数量与滑条中包括的传感器数量之和。 CSDPLUS_wGetPortPin() 使用传感器编号来确定所选的活动传感器的端口和位掩码。

返回值：

WORD：传感器的扫描值， A 中的 LSB 和 X 中的 MSB。

其他影响：

**

CSDPLUS_wGetPortPin

说明：

该函数返回给定的传感器端口号和引脚屏蔽。传递的参数对 CSDPLUS_Sensor_Table[] 中的数据编制索引并进行选择。可以将返回值传递到 CSDPLUS_EnableSensor()、 CSDPLUS_DisableSensor() 中。

C 原型：

```
WORD  CSDPLUS_wGetPortPin(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDPLUS_wGetPortPin
```

参数：

bSensor — 范围是 0 到 （n – 1），其中 ‘n’ 是 CSDPLUS 向导中设置的传感器数量与滑条中包括的传感器数量之和。 CSDPLUS_wGetPortPin() 使用传感器编号来确定所选的活动传感器的端口和位掩码。

返回值：

A => 传感器位图

X => 端口编号

其他影响：

**

CSDPLUS_EnableSensor

说明：

该函数配置所选的传感器，以便在下一个测量周期中进行测量。通过 CSDPLUS_wGetPortPin() 函数可以选择端口和传感器，其中端口编号和传感器位掩码分别被加载到 X 和 A 中。修改驱动模式以便将所选端口和引脚置于模拟高阻模式以及使能正确的模拟复用器总线输入。这还可以使能比较器功能。

C 原型：

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

汇编：

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

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

CSDPLUS_DisableSensor

说明：

该函数禁用由 CSDPLUS_wGetPortPin() 函数所选的传感器。驱动模式被更改为 “ 强 （001）”，可以将传感器有效接地。端口引脚与 “ 模拟复用器总线 ”（AnalogMuxBus）的连接被关闭。函数参数由 CSDPLUS_wGetPortPin() 函数返回。

C 原型：

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

汇编：

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

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

CSDPLUS_GetSnsParasiticCapacitance

说明：

该 API 返回传感器的寄生电容，单位为 pF。

C 原型：

```
BYTE CSDPLUS_GetSnsParasiticCapacitance(BYTE bSensor)
```

参数：

bSensor A => 传感器编号。

返回值：

A => 传感器的寄生电容，单位为 pF。

其他影响：

**

固件源代码示例

示例 1. 此代码用于启动用户模块，并连续扫描传感器。可以使用通信部分将数值传输给 PC 绘图工具。

```
//-------------------------------------------------------------------------
// Sample C code for the CSDPLUS User 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;
    CSDPLUS_Start();
CSDPLUS_InitializeBaselines(); //scan all sensors first time, init baseline
    CSDPLUS_SetDefaultFingerThresholds();
//
// Loop Forever
//
while (1)
    {
CSDPLUS_ScanAllSensors(); //scan all sensors in array (buttons and sliders)
       CSDPLUS_UpdateAllBaselines(); //Update all baseline levels;

//detect if any sensor is pressed
        if(CSDPLUS_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 CSDPLUS_waSnsResult[x] <- Raw Counts
// OUTPUT CSDPLUS_waSnsDiff[x] <- Difference
// OUTPUT CSDPLUS_waSnsBaseline[x] <- Baseline
// OUTPUT CSDPLUS_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 CSDPLUS User Module
//--------------------------------------------------------------------------

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

void main(void)
{
    M8C_EnableGInt;

CSDPLUS_Start(); // Start CSDPLUS UM
    CSDPLUS_SetDefaultFingerThresholds(); // Set default thresholds for button
    // Initialize baseline for sensor number "3"
    CSDPLUS_InitializeSensorBaseline(3);

    while (1)
    {
        // Scan continuously sensor number "3" which is connected
CSDPLUS_ScanSensor(3);
        CSDPLUS_UpdateSensorBaseline(3); //Update Baseline for sensor 3
```

```
        if(CSDPLUS_bIsSensorActive(3)) // check if sensor 3 is touched
        {
            // Add user code here to proceed the buttons pressing
        }
    }
}
```

配置寄存器

CSDPLUS 用户模块使用定时器 1、CapSense、比较器和 Capsence 屏蔽信号 PSoC 模块。CSDPLUS 用户模块使用定时器 1、 CapSense 和比较器 PSoC 模块。通过一组寄存器对每个模块进行个性化和参数化设置。该章节对这些设置进行了简要说明。这些寄存器的符号名称在用户模块实例的 C 语言和汇编语言接口文件 （“.h” 和 “.inc” 文件）中定义。

定时器 1 模块寄存器

- 组 0
  - 定时器 1 配置寄存器：PT1_CFG

    可编程定时器配置寄存器 （PT1_CFG）对 PSoC 的可编程定时器进行配置。
  - 定时器 1 数据寄存器 0：PT1_DATA0

    定时器 1 数据寄存器 0 （PT1_DATA0）保持可编程定时器的计数值的低 8 位。
  - 定时器 1 数据寄存器 1：PT1_DATA1

    定时器 1 数据寄存器 1 （PT1_DATA1）为器件保持可编程定时器的计数值的 8 位。

CapSense 模块寄存器

- 组 0
  - CapSense 控制寄存器 0：CS_CR0

    CapSense 控制寄存器 0 （CS_CR0）控制了 CapSense 计数器的操作。
  - CapSense 控制寄存器 1：CS_CR1

    CapSense 控制寄存器 1 （CS_CR1）包含了附加的 CapSense 系统控制选项。
  - CapSense 控制寄存器 2：CS_CR2

    CapSense 控制寄存器 2 （CS_CR2）包含了附加的 CapSense 系统控制选项。
  - CapSense 控制寄存器 3：CS_CR3

    控制寄存器 3 （CS_CR3）包含控制位，主要用于低通滤波器和参考缓冲区。
  - CapSense 计数器低字节寄存器：CS_CNTL

    CapSense 计数器低字节寄存器 （CS_CNTL）包含了低字节计数器的当前计数。

- CapSense 计数器高字节寄存器：CS_CNTH

  CapSense 计数器高字节寄存器 （CS_CNTH）包含了高字节计数器的当前计数。
- CapSense 状态寄存器：CS_STAT

  CapSense 状态寄存器 （CS_STAT）控制 CapSense 计数器选项。
- CapSense 斜率控制寄存器：CS_SLEW

  CapSense 斜率控制寄存器 （CS_SLEW）为驰张振荡器启用并控制快式斜率模式。

比较器模块寄存器

■ 组 0

- 比较器控制寄存器 0：CMP_CR0

  比较器控制寄存器 0 （CMP_CR0）对比较器的输入范围进行启用并配置。
- 比较器控制寄存器 1：CMP_CR1

  比较器控制寄存器 1 （CMP_CR0）配置比较器输出选项。
- 比较器复用寄存器：CMP_MUX

  比较器复用器寄存器 （CMP_MUX）包含了比较器 0 和比较器 1 的输入选择控制位。
- 比较器 LUT 控制寄存器：CMP_LUT

  比较器 LUT 控制寄存器 （CMP_LUT）选择逻辑函数。

附加的寄存器受 CSDPLUS 用户模块的影响。

■ 组 0

- 模拟复用配置寄存器：AMUX_CFG

  使用该寄存器来配置积分电容引脚连接到模拟全局总线。
- 伪随机序列和预分频器控制寄存器：PRS_CR

  该寄存器控制预分频器和伪随机序列发生器的输出。
- DAC 电流数据寄存器：IDAC_D

  该寄存器对决定输出 IDAC 电流的 8 位乘法因素进行指定。

■ 组 1

- 输出至端口 0 的寄存器：OUT_P0

  该寄存器允许特定的内部信号输出到端口 0 引脚。
- 输出覆盖至端口 1 寄存器 （Output Override to Port 1 Register） : OUT_P1

  该寄存器允许特定的内部信号输出到端口 1 引脚。
- 模拟复用器端口位使能寄存器：MUX_CR0

  该寄存器用于控制模拟复用器总线和相应引脚间的连接。

– AnalogMuxDAC0 寄存器：IDAC_0_REG

使用该寄存器进行设置 IDAC0 代码。

– AnalogMuxDAC1 寄存器：IDAC_1_REG

该寄存器对决定输出 IDAC 电流的 8 位乘法因素进行指定。

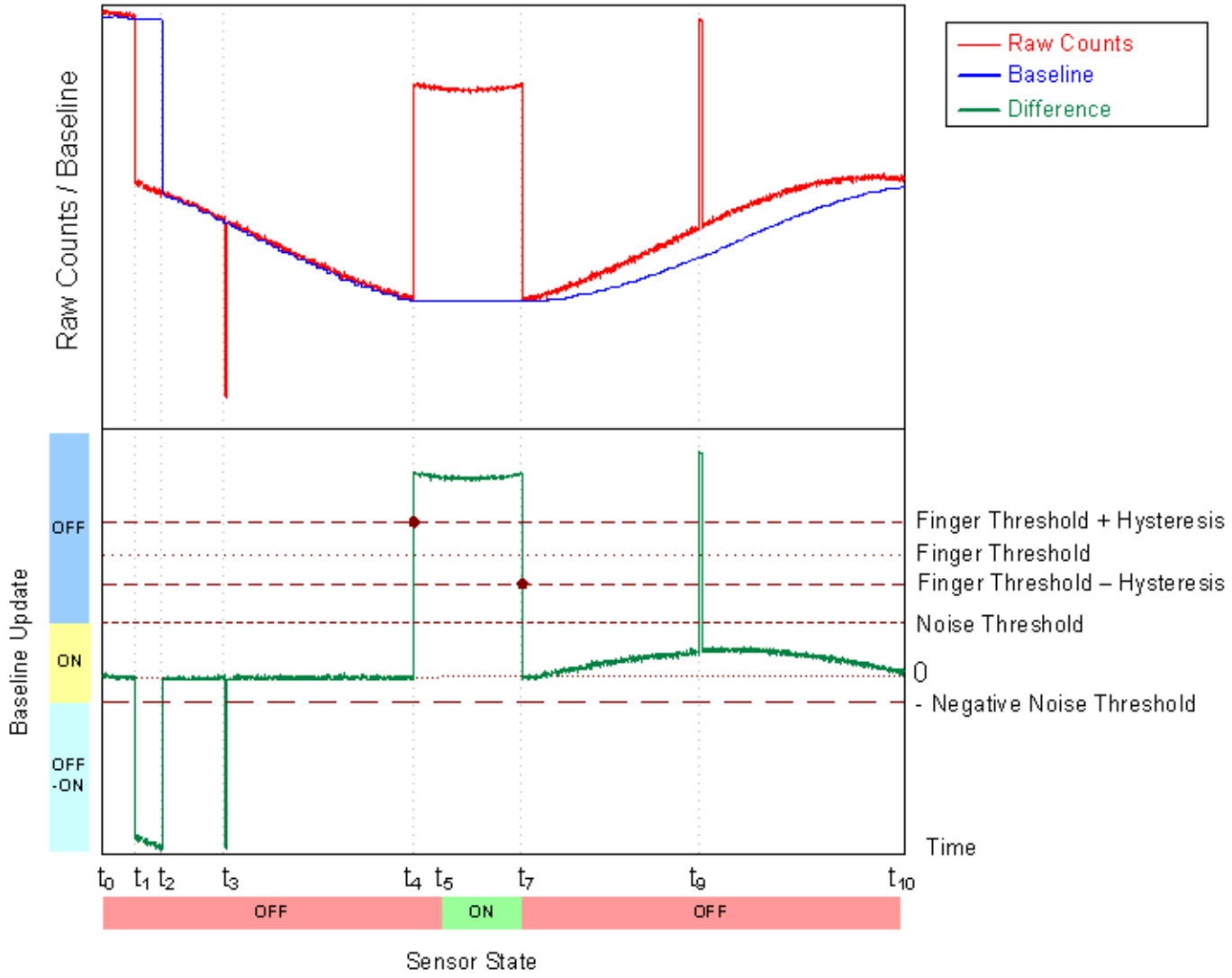
附录

以下部分介绍用户模块数据手册中通常没有包含的信息。赛普拉斯工程师编写了详细信息用于帮助您设计 CapSense 应用程序。该信息中的某些内容将来会被转移到应用笔记中。

CSDPLUS 参数的交互

图 14 和图 15 说明了基准线更新和决策逻辑操作，使您更好地了解如何设置用户参数以获得最佳性能。图 14 说明了当传感器自动复位参数设置为 Disabled （禁用）时的系统操作。图 15 显示的是 Enabled （使能）的传感器自动复位参数。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。

图 21. “ 传感器自动复位 ” 设为 “ 禁用 ” 时原始计数、基准线值、差值信号变化示例



在 t_0 处，由于湿度或温度变化，接近于基准线级别的原始计数开始缓慢下降。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 ” （NegativeNoiseThreshold）参数 （绝对值），因此通过跟踪原始计数的最小值来更新基准线，保留原始计数信号的较小值。

在 t_1 处，原始记录快速下降，负差超过 NegativeNoiseThreshold。如果手指位于传感器上时为器件加电，经过一段时间手指被移开，则会发生这种情况。此时，基准线更新机制冻结，内部超时计数器被激活。当差值信号低于 “ 低基准线复位 ” （LowBaselineReset）样品的 NegativeNoiseThreshold 时，基准线复位。这是在 t_2 处发生的。

第二大负差信号尖峰脉冲发生在 t_3 处；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 “ 低基准线复位 ” （LowBaselineReset）参数，因此保留基准线，对尖峰脉冲进行滤波。这可以阻止假基准线复位和导致假触摸的检测。

传感器是在 t4 处被触摸的。当差值信号超过 “ 手指阈值 + 迟滞 ” （FingerThreshold + Hysteresis）值时，内部防抖动计数器会激活。如果信号超过此值的量大于去抖动样本数量，则传感器状态设置为开启。这是在 t5 处发生的。当差值信号在 t7 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。由于采样单元中的尖峰脉冲持续时间不超过去抖动值， t9 处的瞬时正值尖峰脉冲由去抖动计数器筛选。

原始计数在 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 ”） 时，使用水桶算法来更新基准线。差值信号与漂移速率间比例变化。可以使用基准线更新阈值参数来控制基准线更新速度。参数值越低，基准线更新速度越快。

图 22. 传感器自动复位参数设置为 “ 使能 ” 时的原始计数、基准线、差值信号的变化示例

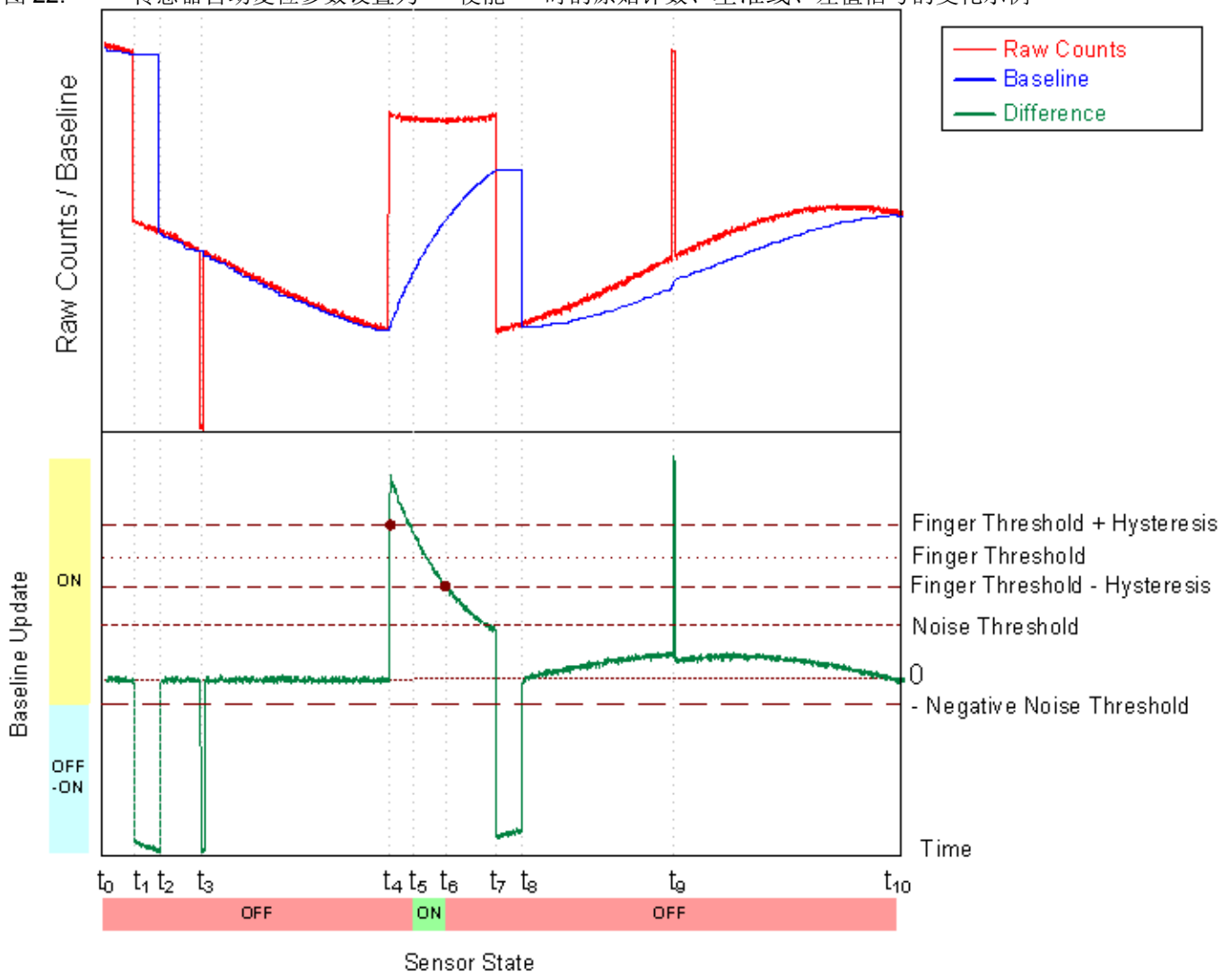


图 15 中的系统操作与上一实例中的操作类似，但有以下区别：

- 在 t_6 处触摸传感器时，由于活动基准线更新算法，触摸持续时间会减少。
- 手指移开后，基准线会在 LowBaselineReset （低基准线复位）采样 （t_8）后复位，这会短时阻止触摸检测。这可作为附加的去抖动机制。

版本历史记录

版本	创作者	说明
1.00	DHA	初始版本。
1.10	DHA	1. 在用户模块数据手册中更新了 RAM 和 ROM 的用量。 2. 将以下的用户模块变量使用到应用代码内：CSD_bNoiseThreshold、CSD_bNegativeNoiseThreshold、 CSD_bBaselineUpdateThreshold、 CSD_bHysteresis、CSD_bDebounce 以及 CSD_bLowBaselineReset。
1.20	MYKZ	1. 纠正了有关保存滑条信息的问题。 2. 更新了基准线算法，用于检查负差值计数。 3. 将 GetSnsParasiticCapacitance 添加到用户模块 API。 4. 更新了用于复位 PRS 的 ScanSensor() 函数。 5. 纠正了 wGetCentroidPos 函数，以便当不触摸传感器时，仍然返回有效值。 6. 修正了 GetSnsParasiticCapacitance 函数中存储器分页的问题。

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

Copyright © 2012-2014 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。