初版

CapSense® Sigma-Delta 数据手册 CSDe V 2.10

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	
CY8C20xx6L							
用户模块	1	–	1	1	1344	39	1
滑条 API	–	–	–	–	584	79	0
每个传感器	–	–	–	–	5	13	1

特性与概述

- 使用 sigma-delta 数据转换方法使 PSoC® 器件的 CY8C20xx6L 系列实现 CapSense® 电容式感应功能
- 具有抗 GPIO 电流瞬变、 V_{DD} 波动、进入 / 退出睡眠引起的噪声以及 IDAC RTS 噪声的能力
- 通过可配置的系统参数，可以在各种应用中优化其性能
- 最多支持 36 个电容式传感器和 6 个滑条
- 能够检测低至 0.1 pF 的触摸，就是可以通过 15mm 厚的玻璃或 5mm 厚的塑料检测到手指触摸
- 支持将电容式传感器配置为独立按键和作为相关阵列以形成滑条。
- 通过使用复用法，滑条元件的有效数量能达到专用 I/O 引脚数的两倍
- 支持通过插值使滑条的分辨率高于物理间距。
- 利用屏蔽电极能够在存在较高的寄生电容和有水膜的情况下进行可靠的操作。
- 通过使用 CSDe 向导实现传感器和引脚分配
- 外部组件和PCB布局的要求与CSD的一样，除了48QFN封装需要将P4[2]引脚连接至PCB上的隔离板（未接地）
- 在 5 至 45 pF 的有效范围内，不用进行任何硬件调试 （手动或自动）
- 无论传感器的电容 （CP）为多大，每个传感器的积分时间是一个常数，其值为 400 us。
- CSDe 用户模块与 SmartLED 用户模块配合使用。

CSDe 用户模块使用差分电容式感应实现 CSD。

快速入门

1. 如果使用，请选择并放置所需要专用引脚的用户模块 （如 I^2C 或 LCD）。根据需要，进行分配端口和引脚。
2. 选择并放置 CSDe 用户模块。
3. 在工作区浏览器中右键单击 CSDe 用户模块，以访问 CSDe 向导 （在本用户模块数据手册中的后面章节对该向导加以介绍）。
4. 设置所需的传感器、滑条和旋转滑条的数量。
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 CSDe 向导内，指定与外部调制电容相连的引脚。
8. 如果需要，请输入用于屏蔽用户模块参数列表中传感器的引脚 （请参考下面部分的说明）。
9. 生成应用，并切换到应用编辑器。
10. 根据需要调整采样代码，以执行独立传感器、滑条传感器或触摸板。
11. 通过 PSoC Designer™ 生成的十六进制文件，对目标电路板上的 PSoC 器件进行编程。

简介

CapSense 是一种人机界面技术，通过使用由导电表面 （通常是蚀刻在 PCB 上的焊盘）构成的传感器检测人体的电容而实现。由于 CapSense 会检测人体电容，所以它能够检测诸如塑料或玻璃外覆层等绝缘层。这些外覆层通常构成器件的外壳。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。
- 为更小终端产品减少外形尺寸。
- 高级的用户接口特性，如 LED 效果和接近传感。
- 可以使用没有损耗或者寿命无限的元件来提高可靠性。
- 提高了抵制破损的能力，因为没有使用机械接口。
- 因为不需要机械输入器件的穿透或其他机械工作，因此降低了加工成本。

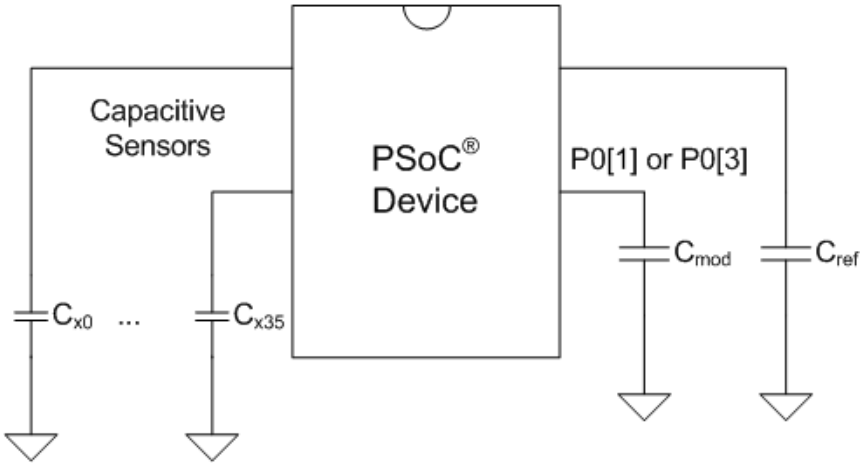
CSDe 将 sigma-delta 电容应用在数字代码的转换电路中，它的重要属性包括低 EMC 辐射和卓越的抗 EMI 性能。

CSDe 用户模块包括：PCB、 IC 和软件组件。

PCB

图 1 显示的是 CSDe 的原理图。物理传感器通常是一个导电的图案，被安装在与 PSoC 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 实现的完整信息，请参考 CY8C20xx6A/H CapSense 和 CapSense 入门的设计指南。

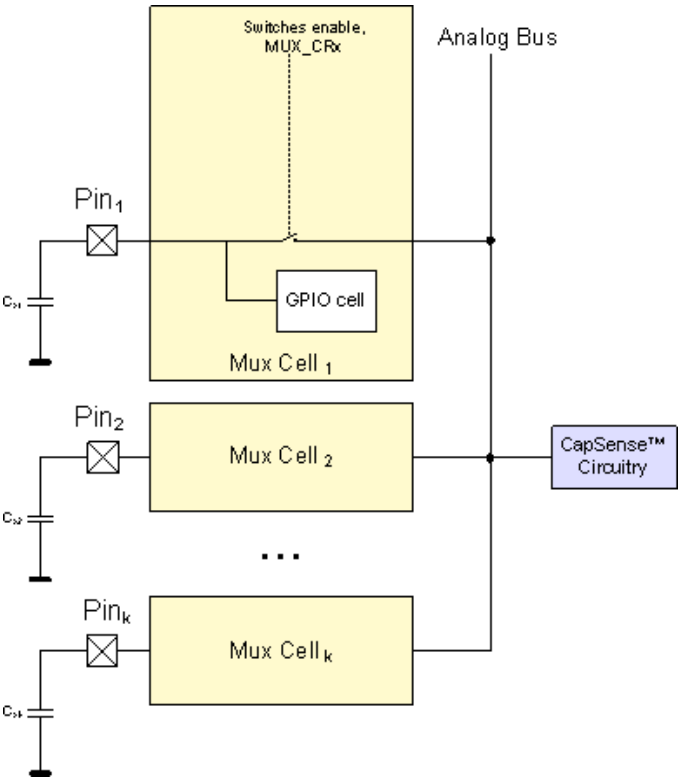
图 1. CSDe 原理图



IC

通过 CY8C20xx6L 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。 CSDe 用户模块将活动和参考传感器连接到模拟复用器总线上，因此 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。通过依次设置 MUX_CRx 寄存器中的相应位，固件可以连续地扫描各传感器，如图 2 所示。

图 2. CY8C20xx6L 模拟复用器总线 （AMUX）框图



软件

CSDe 软件组件的属性如下所示：

- 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根据环境变化进行补偿。
- 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。
- 高层的软件函数执行滑条复用方法，从而能够将单个 I/O 引脚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 I/O 数减半。

推荐阅读

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

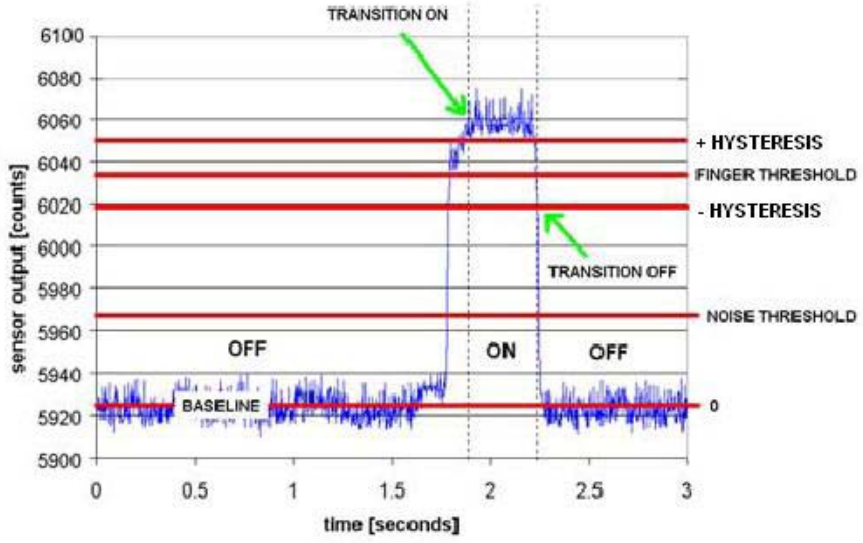
- Capsense 入门
- CY8C20xx6A/H CapSense 设计指南
- CY8C21x34/BCapSense 设计指南
- CY8C20x34CapSense 设计指南
- CY8CMBR2044 CapSense 设计指南

电容式感应的实现

按键

CapSense 按键类似于机械式按键。这些按键可用于独立控制，如打开 / 关闭开关、功能键、菜单键，等等。 API 函数对各传感器中的电容信号 （原始数值）进行监控并将这些值与可调试阈值参数进行比较。触摸某个传感器时，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，传感器就被激活。图 3 显示的是激活传感器的一个典型信号 （蓝线）。调试各阈值 （红线）以提供所需要的性能。

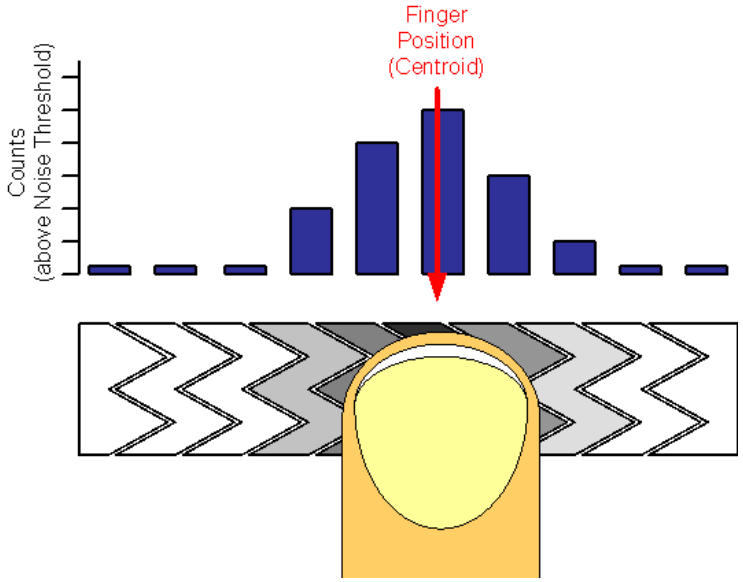
图 3. 激活传感器的电容信号



滑条

CapSense 滑条类似于机械电位器。各滑条用于对连续电平进行控制，如照明调光器、音量控件、图形均衡器、速度控件等。通过使用一组相邻的电容传感器，可以实现 CapSense 滑条。当手指启动某个滑条时，相邻的几个传感器将记录电容信号的增量，如图 4 所示。通过计算活动传感器组的中间位置，可以确定触摸的正确位置。滑条中的传感器实际最小数量为五，最大值仅受限于 PSoC 器件上可用的 I/O 引脚数量。

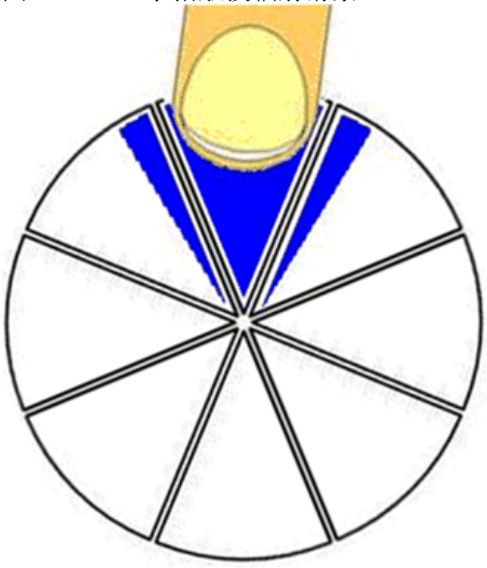
图 4. 滑条上手指的插值质心位置



辐射滑条

CSDe 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。如图 5 所示。在两种情况下，如果发生触摸，可通过质心算法计算各个传感器的信号 （这些传感器与具有最大信号的传感器相邻），以确定触摸的正确位置。辐射滑条未采用复原法。 CSDe 用户模块为辐射滑条提供两个特殊的 API 函数。第一个函数 CSDe_wGetRadiaPos() 返回中心位置，第二个函数 CSDe_wGetRadialInc() 则返回以分辨率单位表示的手指移位。当手指以顺时针方向移动时， 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都受限制，其限制为（传感器所用的引脚数量 – 1）x 2^8 – 1，对于复用型滑条该值为（2 x 传感器所用的引脚数量 – 1）x 2^8 – 1。

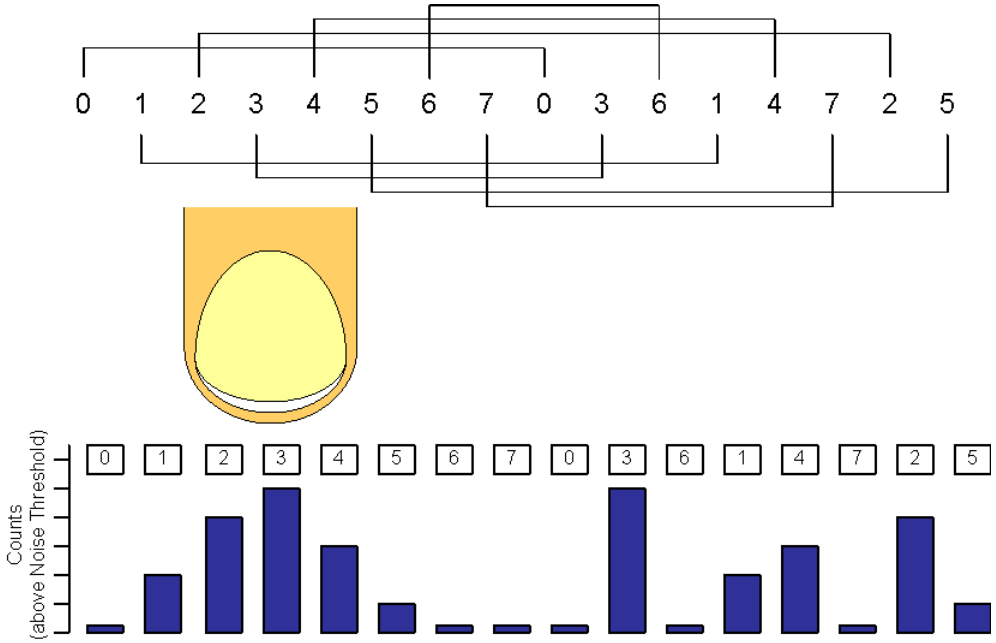
图 5. 手指触摸辐射滑条



复用

使用复用法时，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。物理位置的前半部分 （或为数字中较低部分）会根据 CSDe 向导分配的端口引脚被映射。物理传感器位置的后（或上）半部分将通过图 6 中所示的模式被自动映射。

图 6. 由 CSDe 进行的复用滑条阵列索引编制



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导致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）上各引脚的映射情况符合服用算法中 ‘ 按 3 编制索引 ’ 序列的要求。复用滑条中传感器对的电容应该合理匹配 （不超过 10 pF）。当选择复用时，复用传感器索引表由 CSDe 向导自动生成。表 1 显示的是双工成 28 个 PSoC I/O 引脚的 56 个滑条段的复用序列。

表 1. 不同滑条段数量的双工序列

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

滑条段总数	段序列
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}）

CSDe 需要一个外部调制电容 C_{mod}，从 Vss 连接到一个专用 PSoC 引脚：P0[1] 或 P0[3]。可以在 CSDe 向导中的 “Global Settings （全局设置）> Modulator Capacitor Pin” （调制电容引脚）下面进行 C_{mod} 引脚分配。所选引脚不得用于其他任何用途。外部调制电容的建议值为 2.2 nF。应该使用陶瓷电容。温度电容系数并不重要。赛普拉斯推荐在所有 CapSense 传感器走线上安装大小为 560 Ω 的串联电阻，以抑制 RF 的干扰。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）。其目的在于当覆盖层上有水时将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的背面或其外侧，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。这样，将使传感器电容信号相应增加，可达到能够错误激活传感器的值。驱动屏蔽电极抵消寄生电容耦合，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

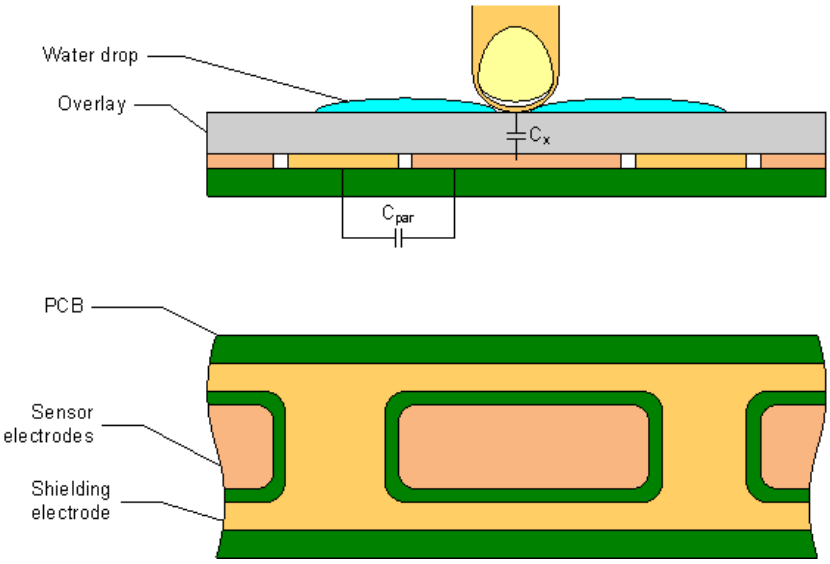


图 7 显示的是按键的驱动屏蔽电极。作为另一种替代方法，屏蔽电极可以安装在背面的 PCB 层上，其中包括按键下面的平面。在这种情况下，推荐使用填充模式，填充率约为 30 到 40%。不需要额外添加地层。

屏蔽电极可以连接到专用 PSoC 引脚 （P0[7] 或 P1[2]）。选定引脚的驱动模式应当设置为强驱。可在 PSoC 器件和屏蔽电极之间连接 560 Ω 斜率限制电阻，以减少散发的电磁干扰。

电源要求

表 2. CSDe 供电电压要求

参数	最小值	典型值	最大值	单位	测试条件和注释
V_{DD}	最小绝对 VDD 规格为 1.8[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。

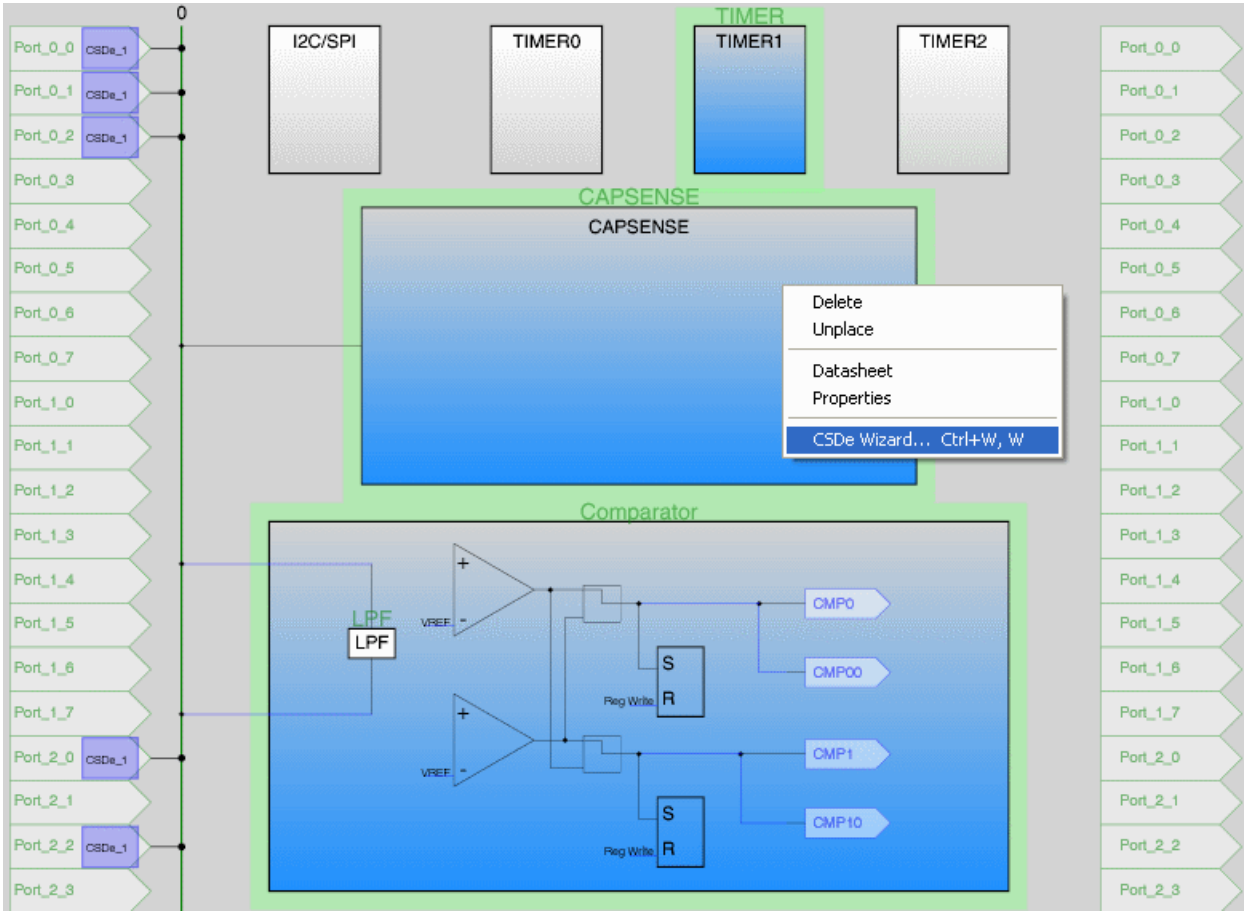
a. 1.8 V。 V_{DD} 下降到 1.8 V 以下会引入过多噪声。

放置

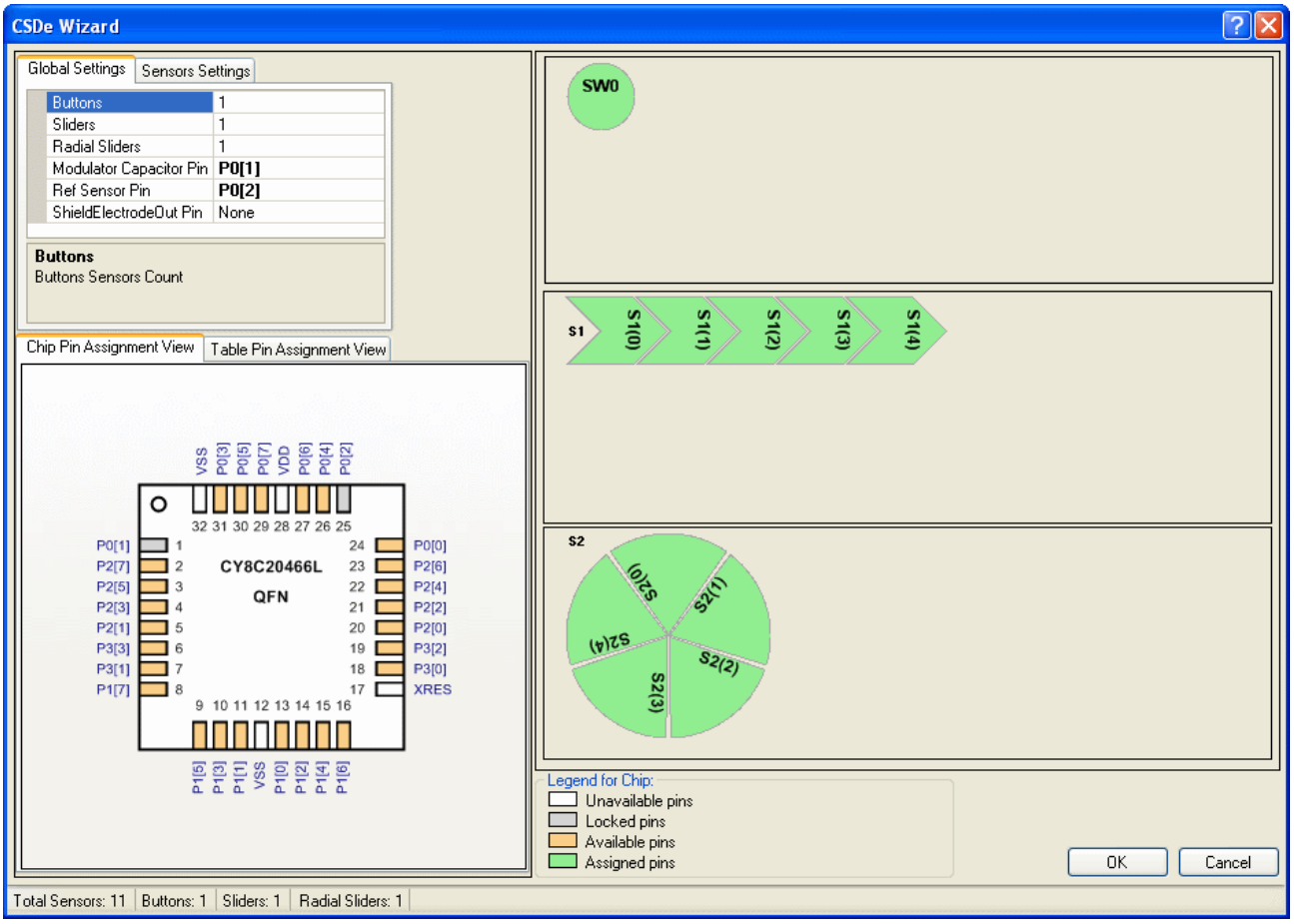
当使用用户模块时，定时器、 CapSense 和比较器模块将被分配给 CSDe。备用放置不可用。在启动 CSDe 向导之前，必须放置需要专用引脚资源的用户模块，包括 LCD 和 I2CHW。这样，传感器被映射到 CSDe 向导中的 I/O 引脚时，会保留专用引脚并且不能将其意外分配给传感器。在放置电容传感器的连接时，请勿使用 P1[0] 和 P1[1]。这些引脚用来对部件进行编程，而且有可能存在过大的布线电容值，从而会影响传感器的灵敏度。

CSDe 向导

1. 要访问 CSDe 向导，请在芯片编辑器中右键单击任意用户模块，并选择 “CSDe 向导 ”。



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



向导引脚图标

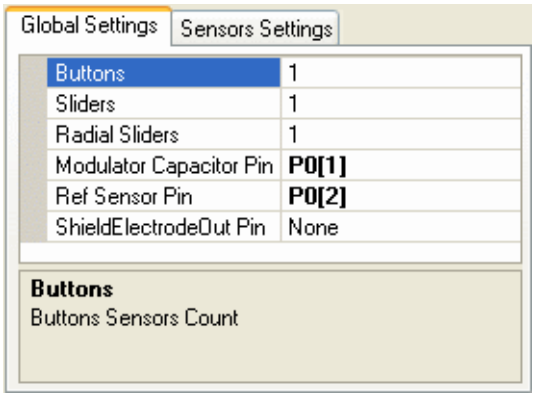
- 白色 — 该引脚不能作为 CapSense 输入使用。
- 灰色 — 引脚被锁定。此情况有两种可能的原因。一种可能的原因是另一个用户模块 （如 LCD 或 I2C）已占用了该引脚。第二种可能性是引脚已更改为非默认名称。要恢复引脚的默认名称，请在 “ 引脚布局 ” 视图中的 Select （选择）菜单展开引脚，并选择 Default （默认）。现在即可在向导中分配引脚。
- 橙色 — 可以分配引脚。
- 绿色 — 引脚已分配为 CapSense 输入。

3. 选择全局设置标签以键入独立按键、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。

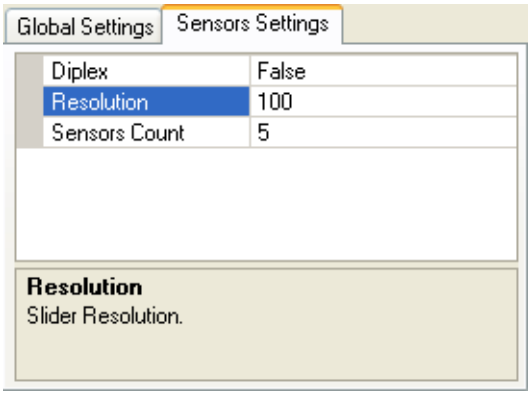
4. 选择调制器电容引脚 （C_{mod}）。您可以选择 P0[1] 或 P0[3]。

5. 请选择参考传感器引脚 （C_{ref}）。你可以从下拉列表中选择一个。

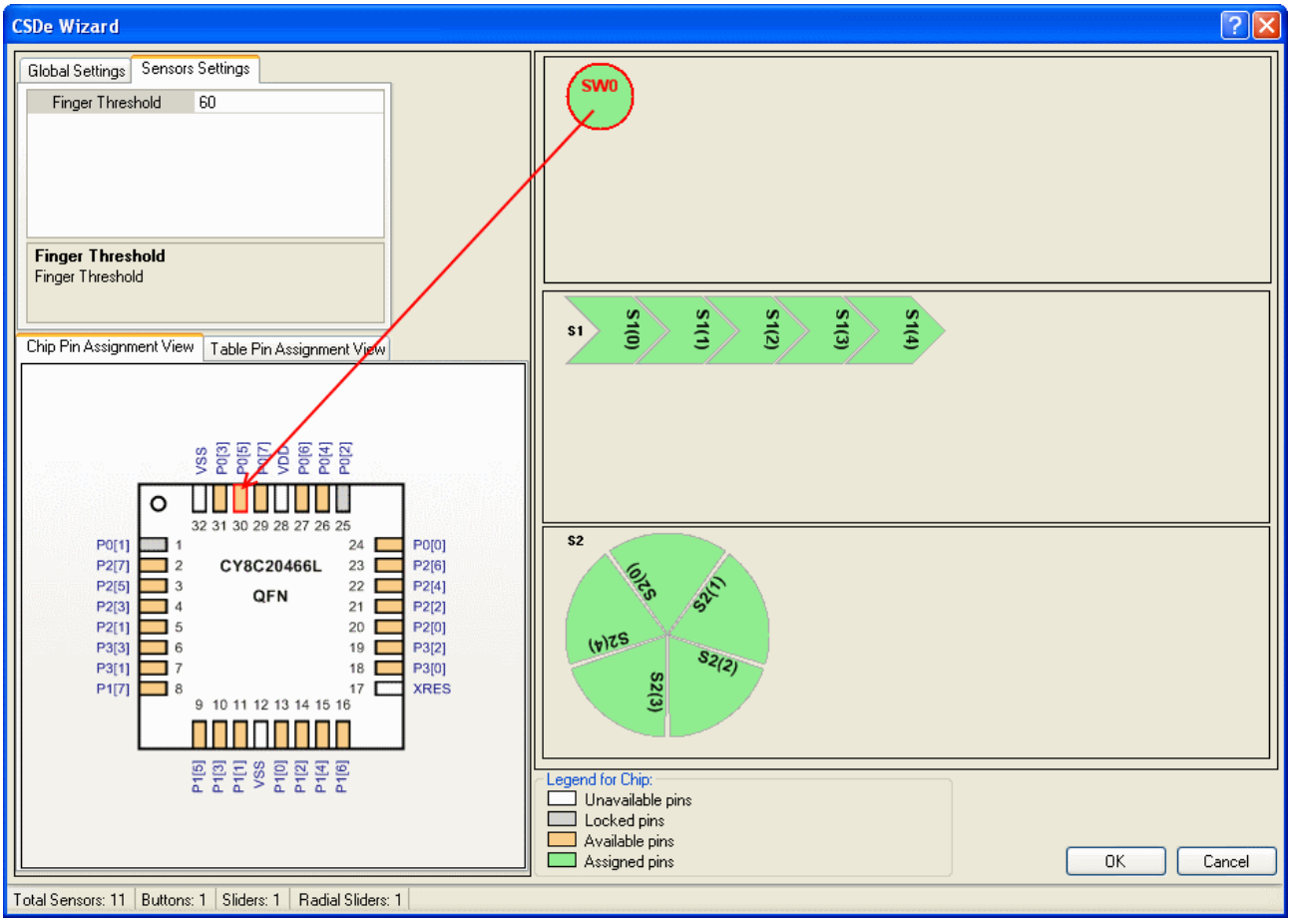
6. 如果需要，选择屏蔽电极输出引脚。您可以选择 P0[7] 或 P1[2]。



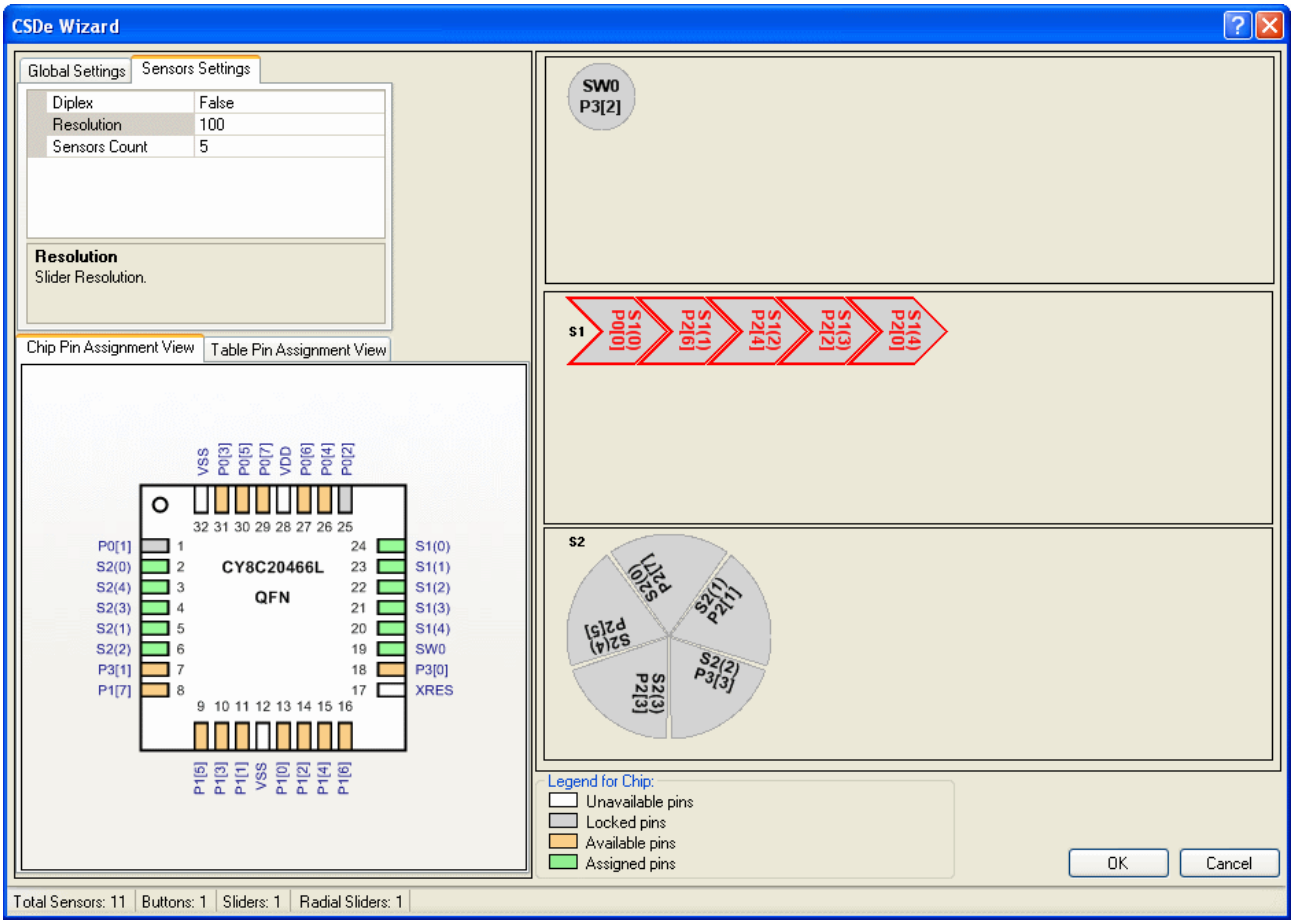
7. 选择 Sensors Settings （传感器设置）选项卡以设置按键、滑条和辐射滑条。要更改设置，请单击以激活其中一个滑条。键入每个滑条中传感器元件的数量。滑条传感器中的传感器实际最小数量为五，最大值受限于引脚数量。输入数据后，按 [Enter] 键更新显示屏。



8. 键入输出分辨率。最小值为 5。 CSDe 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。软件报告的滑条触摸结果在零和分辨率 –1 之间。
9. 如果需要，选择 “ 复用 ”。这样会把为传感器选定的引脚数量映射为板上传感器位置数量的两倍数值。仅显示了复用传感器的前半部分；后半部分按前面 “ 复用 ” 一节所述的内容自动映射。有关引脚连接的复用表，请参见 “ 复用 ” 一节。



10. 在引脚分配视图中将传感器拖到引脚上，这样就可以将传感器分配给引脚。您可以选择在 “ 芯片引脚分配视图 ” 或 “ 表格引脚分配视图 ” 中将传感器拖放到引脚上。选择 I/O 引脚后，引脚变为绿色，不再可用。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。避免选择已经指定给其他用户模块的引脚。
11. 对其他传感器重复以上操作。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 ” 窗口并选择刷新即可更新引脚连接。



要想更改引脚分配，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。该引脚分配取消，然后可以将其重新分配。

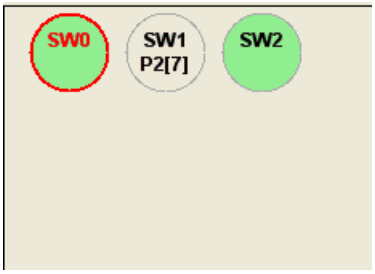
向导完成后，单击 “Generate Application” （生成应用）。根据您所输入的传感器数量、引脚分配、双工和分辨率，会生成一组表格。

传感器表示章节

本章节通过图形方式表示在一个项目中表示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或表格引脚分配视图内。所分配的传感器以灰色显示。本章节包括下面三个部分：

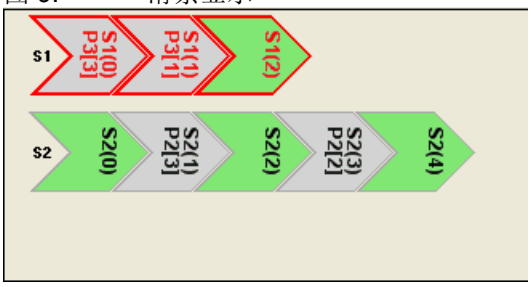
- 按键表示 — 按键传感器在向导中如下图所示。每个按键传感器元件都有自己的标题。所有按钮为芯片和表格引脚分配视图支持拖放式操作功能。如果已经分配好按键，则按键标题的下方将显示被分配的端口引脚编号。如果选择了按键传感器，将按键 widget 被设置为红色的帧。

图 8. 按键表示



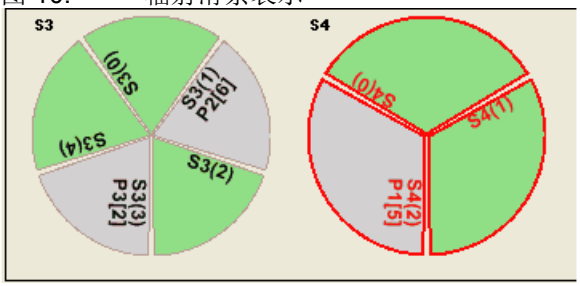
- 滑条表示 — 滑条被显示为一系列的传感器段，如下面的截图所示。滑条的每个传感器都显示了特定名称。滑条中的所有传感器均可在芯片和表格引脚分配视图中实现拖放操作。如果滑条段已被分配，每个段会显示被分配的端口引脚编号。如果已选择一个滑条，将滑条中的所有传感器以红色的帧设置。

图 9. 滑条显示



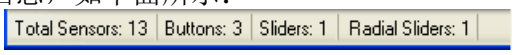
- 辐射滑条显示 — 辐射滑条被显示为饼状段，如下图所示。在辐射滑条中的每个段显示了传感器名称。辐射滑条中的所有传感器均可在芯片和表格引脚分配视图中实现拖放操作。如果辐射滑条段已被分配，每个段会显示分配的端口引脚编号。如果一个辐射滑条已被选择，辐射滑条中的所有传感器以红色的帧设置。

图 10. 辐射滑条表示



状态栏

状态栏显示了与设计有关的通用信息，如下面所示：



- 传感器总计 — 显示了设计中所使用的传感器总数量。
- 按键 — 显示了设计中所使用的按键总数。
- 滑条 — 显示了设计中所使用的线性滑条总数。
- 辐射滑条 — 显示在设计中所使用的辐射滑条总数。

向导按键

CSDe 用户模块向导提供了具有预定义功能的按键。

1. “OK” — 使用该按键可检查向导的参数是否正确，并检查是否已经分配了所有传感器。如果各参数正确，那么向导将保存各参数并关闭；否则，它将显示相应的警告信息，并且不保存参数，而会保持打开状态。
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” — 该按键调用帮助页面以提供有关如何使用 CSDe 用户模块向导的参考信息。它简要地描述 CSDe 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。

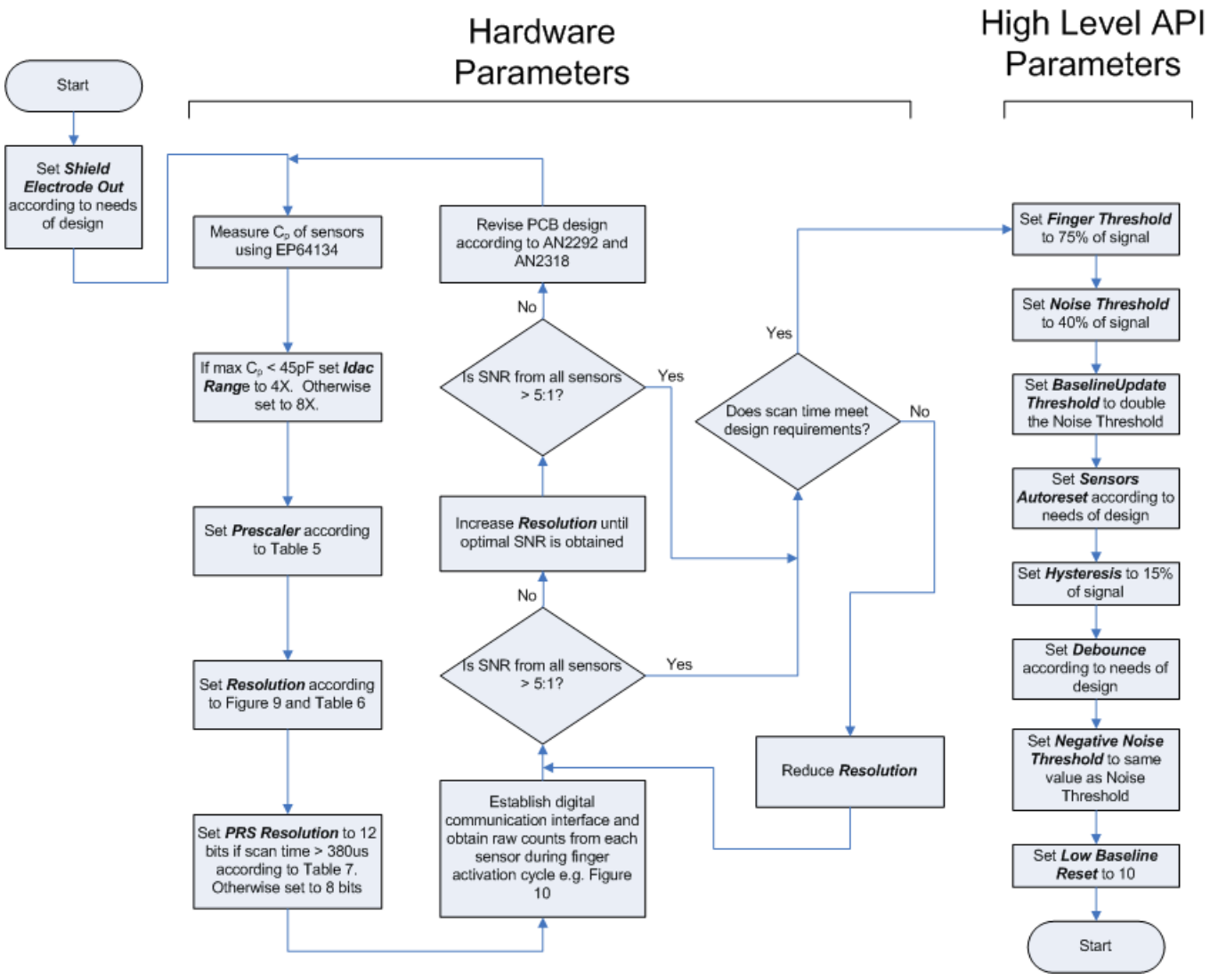
用户模块参数 — 调试指南

在 CSDe 向导中完成配置和 I/O 引脚分配之后，必须设置模块参数。请注意：更改任何用户模块参数生效时，都必须重新生成项目。

图 8 显示的是 CSDe UM 参数调试过程的流程图。 CSDe UM 参数可分为两大类：硬件参数和高级 API 参数。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，每个传感器的灵敏度之间的互补关系由硬件参数设定和若干高级 API 参数设置决定。设计师必须记住

当任何硬件参数改变时，确保对相应的高级 API 参数作出调整。调试 CSDe 用户模块参数应始终从硬件参数开始。

图 11. CSDe 用户模块参数的调试流程图



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。本节描述了这些参数，并指导如何根据系统特性或其他参数调试它们。

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。在设计中，传感器的寄生电容的总值 （C_p），和 / 或传感器的灵敏度在一个较大范围内浮动，那么全局硬件参数设置可能会不适合所有传感器。在此情况下，使用 SetPrescaler(i) 和 SetScanResolution(i) API 函数来配置每个传感器的硬件参数，其中 (i) 是在调用 ScanSensor(i)API 函数之前的传感器索引。示例代码部分包含一个这样的例子。

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 CSDe 设计时，此为最重要的硬件 UM 参数。预分频器取决于所选预充源、 IMO 和传感器的 C_p 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	注解 1	注解 1
7-11	2	1	注解 1
12-15	2	1	注解 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。

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 6 显示了基于 C_p 和手指电容值 C_f 的分辨率推荐设置。C_f 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。默认设置为 12。

图 12. 基于外覆层厚度和圆形传感器直径的手指电容值 （C_f）

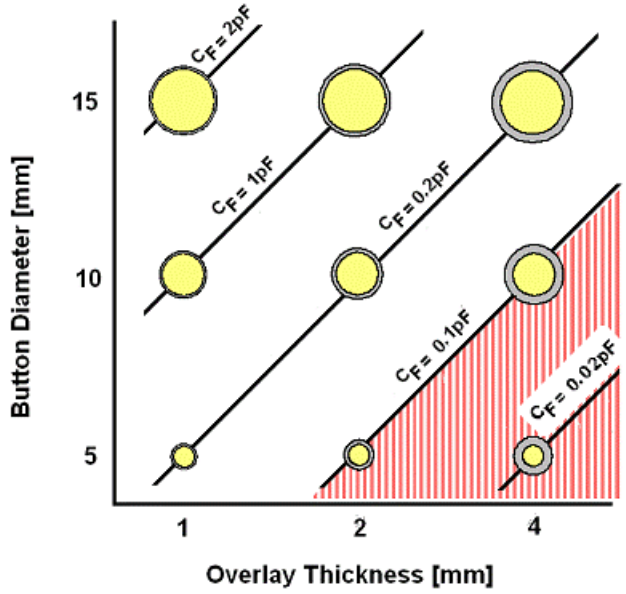


表 4. 基于手指电容值和 Cp 的分辨率设置

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

PRS 分辨率

此参数更改 PRS 序列长度。可能值为 8 和 12 比特。对应的序列长度为 511 和 2047 个输入时钟周期。如果需要极短的扫描时间，则必须使用 8 位 PRS 来避免过大噪声。扫描时间由分辨率 （不应该与 PRS 分辨率相互混淆）参数确定。如果扫描时间 ≤ 380 ms，则将 PRS 分辨率设置为 8 位；如果扫描时间 > 380 ms，则将 PRS 分辨率设置为 12 位。默认设置为 8 位。

滤波器系数

该参数设置了 IIR 滤波器系数。可能值分别为 2、 4、 8、 16、 32 和 64。默认设置为 2。

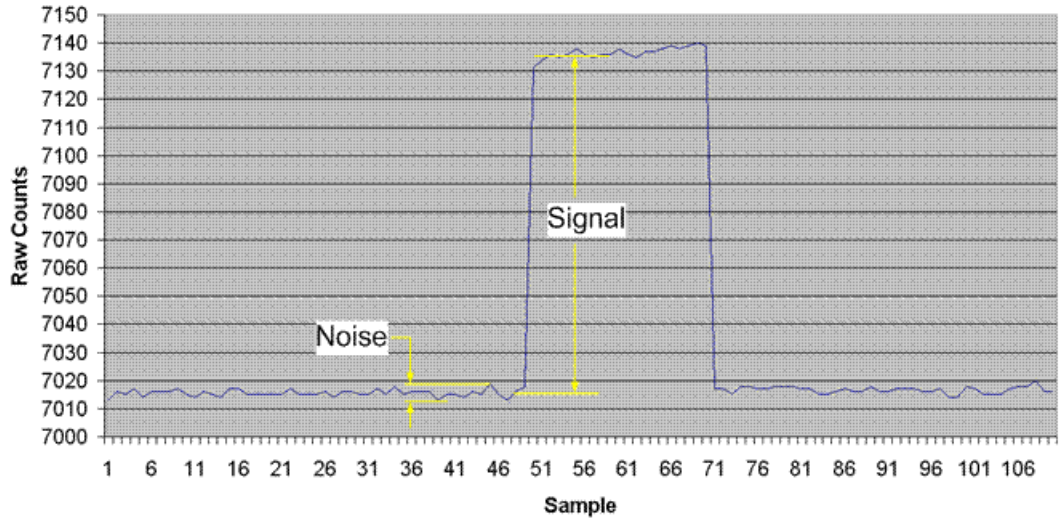
高层 API 参数

高层 API 参数决定高层固件算法的行为，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。为确定适当的参数值，您必须在系统中建立数字通信接口，以监控每个传感器手指激活事件期间的原始信号、基准线和差值信号。数据分别存储于如下五个数组：CSDe_waSnsBaseline[]、CSDe_waRawCount[]、CSDe_wRefSnsResult[]、 CSDe_waSnsResult[] 和 CSDe_waSnsDiff[]。CSDe_waRawCount — 存储了不包括参考传感器值的实际原始信号。 CSDe_waSnsResult — 储存了传感器的实际原始信号和参考传感器原始信号间的差值。

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。噪声与信号的强度取决于 EMI 环境、 PCB 布局、覆盖层厚度及其他系统特性。因此，用于设置这些参数的基础数据必须取自最终汇编状态并且与服务具有相同的 EMI 环境的系统。

图 10 显示的是传感器在一个手指激活周期中获取的典型原始信号值，即某个传感器被激活然后再取消激活状态。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，遵循的高层参数描述包括基于噪声与信号值设置参数的方法的信息。根据 CY8C20xx6A/H CapSense 设计指南，为了 CapSence 系统操作的健壮性，信噪比 （SNR）应至少为 5：1。若信噪比低于 5：1，则需调整硬件参数，或根据 CapSense 入门的设计指南更改 PCB 布局，至少将信噪比提高至 5：1。

图 13. 传感器在一个手指激活周期中的典型原始计数值



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，以用于更新原始计数基准线。取值范围为 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。噪声阈值是应用于所有传感器的全局参数。噪声阈值的良好起点是原始数值信号的 40% （请参考图 10）。默认设置为 10。

BaselineUpdate （基准线更新）阈值

CSDe 使用 “ 水桶 ” 方法来更新 CSDe_UpdateSensorBaseline() API 函数中的基准线计数。在下列情况下，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 AND 值。
2. 原始信号值和基准线之间的差值低于噪声阈值。

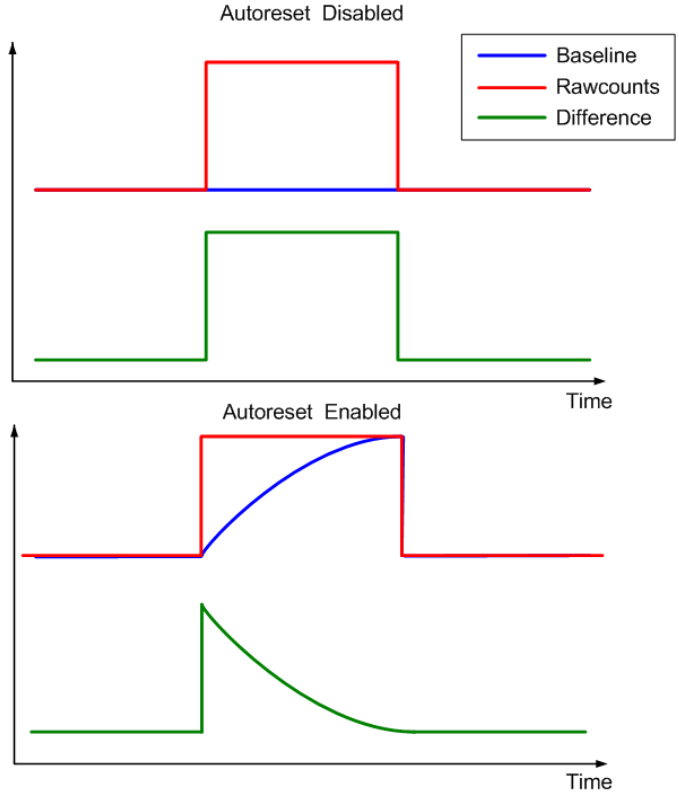
   基准线更新阈值参数设置 “ 水桶 ” 必须达到的值，以增加基准线。该参数的良好起点是噪声阈值的两倍。取值范围为 0 到 255 ；默认值为 100。

Sensors Autoreset （传感器自动复位）

此参数确定基准线是否随时更新，还是仅当信号差值低于噪声阈值时更新。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。当传感器自动复位为 “Enabled” （使能）时，无论噪声阈值如何，都会更新基准线。此设置可限制传感器的最大激活时间 （通常为 5 - 10s）。但是它会防止传感器因非触摸引起原始计数突然上升而被停滞。原始计数突然上升可能是由电源电压剧烈波动、高能射频噪声源或温度快速变化所导致。

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其默认的 “Disabled” （禁用）状态。请参考该参数的其他说明附录。

图 14. 传感器自动复位参数在基准线上的影响

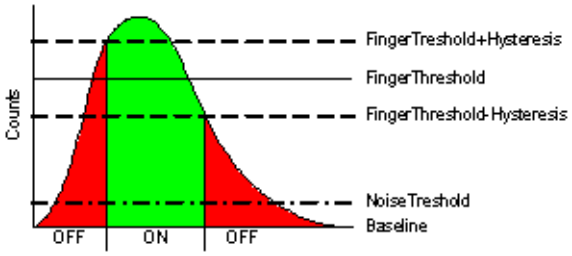


迟滞

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信号的 15%（请参考图 10）；默认设置为 10。

图 15. 手指阈值和迟滞在按键传感器状态上的关系



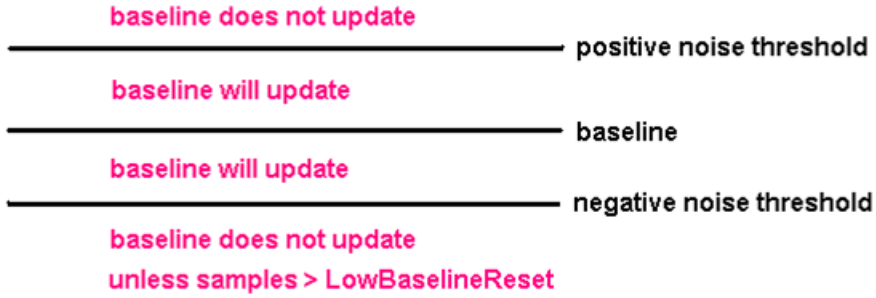
防抖动

该参数为传感器激活的瞬变增加了去抖动计数器。为了让传感器能够从未激活状态切换为激活状态，在该参数规定的样本数量内，差异计数值必须大于手指阈值与迟滞之和。去抖动计数器按 CSDe_bIsSensorActive 或 CSDe_bIsAnySensorActive API 函数递增。取值范围为 1 到 255。如果设置为‘1’，则将没有去抖动但会提供最快的响应。默认设置为 3。

负噪声阈值

当原始信号低于基线时，将使用该参数。它建立了一个相对于当前基准线的级别，如果超过这个级别，基准线将复位到原始信号。如果原始信号低于该级别，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的起点。默认设置为 10。

图 16. 噪声阈值和基准线复位之间的关系。



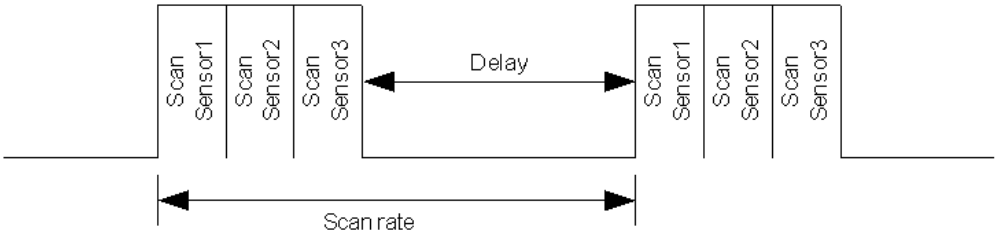
低基准线复位

该参数与负噪声阈值一起使用，用以定义复位基准线到原始信号间必须执行的采样次数，这种采样中的原始信号必须小于基准线值。完成由低基准线复位参数所指定的采样数量后，如果原始信号仍低于基准线值与负噪声阈值之差，则基准线将下降为原始信号值。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。建议将起始点设置为 10；默认设置为 50。

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 17. 典型传感器扫描



为了确保基准线正常运行，推荐保持设计中的扫描速率至少为 15 ms。这意味着具有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可消耗 15 ms。为实现低功耗目的，良好的设计会使 CapSense 控制器进入睡眠模式，而不是使其进入固件延迟子程序。

应用编程接口 （API）

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

只能将此用户模式的一个实例放置在项目中，且它也应用于可加载的配置。每次放置用户模块时，都会为其分配一个实例名称。默认情况下， PSoC Designer 向给定项目中此用户模块的第一个实例分配 CSDe_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 CSDe。

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，则调用函数需要保留在调用前的 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，并且从 PSoC Designer 的 1.0 版本起已强制使用。 C 编译器自动遵循该要求。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 CSDe，启动其采样，并停止 CSDe。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 CSDe 前缀。未能使用正确的名称是常见的语法错误原因。

API 函数使用不同的全局阵列。不得手动更改这些阵列。但是，可以出于调试目的检查这些值。例如，可以使用绘图工具显示阵列的内容。有多种全局阵列：

- CSDe_waRawCount[]
- CSDe_waSnsResult[]
- CSDe_waSnsBaseline[]
- CSDe_waSnsDiff[]
- CSDe_baSnsOnMask[]
- CSDe_baBtnFThreshold[]

CSDe_waRawCount[] — 这是一个整数阵列， CSDe_ScanSensor() 函数使用该阵列储存每个实际传感器扫描的原始信号。阵列大小与传感器数量相等。

CSDe_waSnsResult[] 这是一个整数阵列，它包含每个传感器的原始信号。通过将该值减去参考传感器扫描的原始信号，可以得到最终的扫描结果。阵列大小与传感器数量相等。通过下列函数更新 **CSDe_waSnsResult[]** 数据：

- **CSDe_ScanSensor()**
- **CSDe_ScanAllSensors()**

CSDe_waSnsBaseline[] — 这是一个整数阵列，它包含了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 CSDe_waSnsBaseline[] 阵列：

- CSDe_UpdateAllBaselines();
- CSDe_UpdateSensorBaseline();
- CSDe_InitializeBaselines().

CSDe_waSnsDiff[] — 这是一个整数阵列，它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。

CSDe_baSnsOnMask[] — 这是一个字节阵列，它包含传感器的开 / 关状态 （针对按键或滑条传感器）。CSDe_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）。CSDe_baSnsOnMask[1] 包含传感器 8 到 15 的掩码位 （如果需要），等等。此字节阵列包含的元素数足以包含所有放置的传感器。按键开启时位值为 1，关闭时位值为 0。 CSDe_baSnsOnMask[] 数据由 CSDe_blsSensorActive(BYTE bSensor) 函数或 CSDe_bIsAnySensorActive() 子程序更新。

CSDe_baBtnFThreshold[]— 这是一个字节阵列，用于存储每个传感器的阈值。阵列大小与传感器数量相等。

CSDe_baDAC[]— 这是一个字节阵列，用于存储每个实际传感器的 AutoCalibrated IDAC 值。CSDe_CalibrateSensor() 函数可设置每个传感器的值，并且被 CSDe_ScanSensor() 函数使用。阵列大小与传感器数量相等。

该用户模块 API 将使用以下变量组：

- **CSDe_wRefSnsResult —** 该变量组保存了参考传感器的扫描结果；
- **CSDe_fCSDeFlags —** 该字节标志显示的是 CSDe 状态：

名称	数值	说明
CSDe_CALIBRATION_FLAG	0x01	被用于从实际传感器和 CSDe_ScanSensor() 中的 IIR 滤波器应用的减去，进行抑制参考传感器原始计数的减法。在校准过程中，必要为每个实际传感器确定一个适当的 IDAC 设置。
CSDe_INITIALIZATION_FLAG	0x02	被用于从实际传感器和 CSDe_ScanSensor() 中的 IIR 滤波器应用的减去，进行抑制参考传感器原始计数的减法。在初始化过程中，必要为每个实际传感器指定一个适当的 IDAC 设置。

- **CSDe_bFilterCoef1 —** 它保存了 IIR 滤波器乘数的字节变量。该变量通过 CSDe_Start() 函数被初始化，并通过 CSDe_ScanSensor() 函数被访问。根据 “ 滤波器系数 ” 参数的选择情况，初始化该变量。使用下面的公式计算第一系数：
  Coef1 = $\log_2$ （“ 滤波器系数 ”），其中 “ 滤波器系数 ” 是在 CSDe 用户模块参数窗口中所选的值。

- **CSDe_bFilterCoef2 —** 保存 IIR 滤波器分频值的字节变量。该变量在 CSDe_Start() 函数被始化并在 CSDe_ScanSensor() 函数进行访问。根据 “ 滤波器系数 ” 参数的选择，进行初始化该变量。使用以下公式计算第一系数：
  Coef2 = （“ 滤波器系数 ” - 1），其中 “ 滤波器系数 ” 是 CSDe 用户模块参数窗口中所选的值。
- **CSDe_bRefDAC —** 保存了参考传感器的 AutoCalibrated IDAC 值的字节变量。通过 CSDe_CalibrateRefSensor() 函数设置该值，另外 CSDe_ScanRefSensor() 函数使用该值进行扫描参考传感器。

CSDe_Start

说明：

该函数启动 CSDe 用户模块，初始化全局变量，配置 C_{mod} 并将它连接到 AMUX 总线，配置和连接驱动屏蔽 （如果选定），配置 CapSense 模块和相关硬件。

C 原型：

```
void  CSDe_Start(void)
```

汇编：

```
lcall CSDe_Start
```

参数：

无

返回值：

无

其他影响：

**

CSDe_Stop

说明：

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

C 原型：

```
void  CSDe_Stop(void)
```

汇编：

```
lcall CSDe_Stop
```

参数：

无

返回值：

无

其他影响：

**

CSDe_Resume

说明：

调用 CSDe_Stop() 之后，该函数恢复用户模块操作。

C 原型：

```
void  CSDe_Resume(void)
```

汇编：

```
lcall CSDe_Resume
```

参数：

无

返回值：

无

其他影响：

**

CSDe_ScanRefSensor

说明：

该函数会扫描参考传感器。扫描结果被保存在 RAM 中的 CSDe_wRefSnsResult 内。

C 原型：

```
void  CSDe_ScanRefSensor(void)
```

汇编：

```
lcall CSDe_ScanRefSensor
```

参数：

无

返回值：

无

其他影响

**

CSDe_ScanSensor

说明：

该函数将扫描由 bSensor 指定的实际传感器。扫描实际传感器之前，该函数先要调用 CSDe_Scan-RefSensor() 函数以扫描参考传感器。然后才会扫描实际的传感器，并将扫描的实际传感器结果减去扫描的参考传感器结果。用于自动校准参考和实际传感器的常量确保参考传感器的原始信号始终小于

实际传感器的原始信号，这样可以防止发生移出状态。使用 IIR 滤波器过滤该差值，并将结果存储在 RAM 的 CSDe_waSnsResult() 中。 IIR 滤波器的系数由 CSDe_FILTER_COEF1 和 CSDe_FILTER_COEF2 常量确定。

C 原型：

```
void  CSDe_ScanSensor(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDe_ScanSensor
```

参数：

A => 传感器编号

返回值：

无

其他影响

**

CSDe_ScanAllSensors

说明：

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

C 原型：

```
void  CSDe_ScanAllSensors(void)
```

汇编：

```
lcall CSDe_ScanAllSensors
```

参数：

无

返回值：

无

其他影响

**

CSDe_ClearSensors

说明：

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

C 原型：

```
void  CSDe_ClearSensors(void)
```

汇编：

```
lcall CSDe_ClearSensors
```

参数：

无

返回值：

无

其他影响：

**

CSDe_wReadSensor

说明：

该函数返回 A 和 X 中的关键原始信号的扫描值 （分别为 LSB 和 MSB）。

C 原型：

```
WORD  CSDe_wReadSensor(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDe_wReadSensor
```

参数：

A => 传感器编号

返回值：

传感器的扫描值、 A 中的 LSB 和 X 中的 MSB。

其他影响：

**

CSDe_wGetPortPin

说明：

该函数返回给定的传感器的端口号和引脚屏蔽。传递的参数对 CSDe_Sensor_Table[] 中的数据编制索引并进行选择。返回值可以传递给 CSDe_EnableSensor() 和 CSDe_DisableSensor()。

C 原型：

```
WORD  CSDe_wGetPortPin(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDe_wGetPortPin
```

参数：

bSensor — 有效范围为 0 到 （n – 1），其中 ‘n’ 是 CSDe 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。 CSDe_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：

**

CSDe_EnableSensor

说明：

该函数配置所选的传感器，以便在下个测量周期中进行测量。通过使用 CSDe_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，以便使所选的端口和引脚进入模拟高阻态模式并使能正确的模拟复用器总线输入。这样同样也可以使能比较器的功能。

C 原型：

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

汇编：

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

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

CSDe_DisableSensor

说明：

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

C 原型：

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

汇编：

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

参数：

A => 传感器位图

X => 端口编号

返回值：

无

其他影响：

**

CSDe_SetScanResolution

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。当某些传感器需要以不同的扫描速度和分辨率进行扫描时 （例如：常用按键和接近感应的检测器），此函数非常有效。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。

C 原型：

```
void  CSDe_SetScanResolution(BYTE bResolution)
```

汇编：

```
mov   A, bResolution
lcall CSDe_SetScanResolution
```

参数：

bResolution — 设置分辨率值。

返回值：

无

其他影响：

**

CSDe_SetPrescaler

说明：

该函数覆盖用户模块参数的预分频器值。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  CSDe_SetPrescaler(BYTE bPrescaler)
```

汇编：

```
mov   A, bPrescaler
lcall CSDe_SetPrescaler
```

参数：

bPrescaler — 设置预分频器值。下表列出了合适的值：

名称	数值	预分频器
CSDe_PRESCALER_1	0x00	1
CSDe_PRESCALER_2	0x01	2
CSDe_PRESCALER_4	0x02	4
CSDe_PRESCALER_8	0x03	8
CSDe_PRESCALER_16	0x04	16
CSDe_PRESCALER_32	0x05	32
CSDe_PRESCALER_64	0x06	64
CSDe_PRESCALER_128	0x07	128
CSDe_PRESCALER_256	0x08	256

返回值：

无

其他影响：

**

CSDe_CalibrateRefSensor

说明：

该函数执行了一个二进制搜索，以自动将参考传感器校准为由 CSDe_REFCALIBRATION_TARGET_MSB 和 CSDe_REFCALIBRATION_TARGET_LSB 常量指定的级别。参考传感器的 IDAC_D 设置被保存在 RAM 的 CSDe_bRefDAC 中。

C 原型：

```
void  CSDe_CalibrateRefSensor(void)
```

汇编：

```
lcall CSDe_CalibrateRefSensor
```

参数：

无

返回值：

无

其他影响：

**

CSDe_CalibrateSensor

说明：

该函数执行了一个二进制搜索，以自动校准实际传感器到由 CSDe_SNSCALIBRATION_TARGET_MSB 和 CSDe_SNSCALIBRATION_TARGET_LSB 常量指定的级别。 bSensor 的 IDAC_D 设置被保存在 RAM 的 CSDe_baDAC 中。

C 原型：

```
void  CSDe_CalibrateSensor(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDe_CalibrateSensors
```

参数：

A => 传感器编号

返回值：

无

其他影响：

**

CSDe_UpdateSensorBaseline

说明：

针对每个传感器独立计算得出的历史计数值被称为这个传感器的基准线。此基准线使用 “ 水桶方法 ” 进行更新。

“ 水桶方法 ” 使用下列算法：

1. 每次调用 CSDe_UpdateSensorBaseline() 时，通过从原始信号值中减去以前的基线来计算差值信号。此差值存储在 CSDe_waSnsDiff[] 阵列中，用户可以使用此值。

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

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 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  CSDe_UpdateSensorBaseline(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDe_UpdateSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

其他影响：

**

CSDe_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。将计算迟滞。根据传感器当前是否开启，对手指阈值增加或消减迟滞值。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。该函数还可更新 CSDe_baSnsOnMask[] 阵列中传感器的位。

C 原型：

```
BYTE  CSDe_bIsSensorActive(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDe_bIsSensorActive
```

参数：

A => 传感器编号

返回值：

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

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

其他影响：

**

CSDe_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 CSDe_bIsSensorActive()，以便在调用此函数后更新 CSDe_baSnsOnMask[] 阵列。

C 原型：

```
BYTE  CSDe_bIsAnySensorActive(void)
```

汇编：

```
lcall CSDe_bIsAnySensorActive
```

参数：

无

返回值：

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

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

其他影响：

**

CSDe_SetDefaultFingerThresholds

说明：

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

C 原型：

```
void  CSDe_SetDefaultFingerThresholds(void)
```

汇编：

```
lcall CSDe_SetDefaultFingerThresholds
```

参数：

无

返回值：

无

其他影响：

**

CSDe_InitializeSensorBaseline

说明：

该函数将扫描所选的传感器，将初始值加载到 CSDe_waSnsBaseline[bSensor] 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。此函数可用于复位单个传感器的基准线。

C 原型：

```
void  CSDe_InitializeSensorBaseline(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall CSDe_InitializeSensorBaseline
```

参数：

A => 传感器编号

返回值：

无

其他影响：

**

CSDe_InitializeBaselines

说明：

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

C 原型：
```
void  CSDe_InitializeBaselines(void)
```

汇编：
```
lcall CSDe_InitializeBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSDe_UpdateAllBaselines

说明：

该函数将使用 CSDe_bUpdateSensorBaseline() 函数更新所有传感器的基线。

C 原型：
```
void  CSDe_UpdateAllBaselines(void)
```

汇编：
```
lcall CSDe_UpdateAllBaselines
```

参数：

无

返回值：

无

其他影响：

**

CSDe_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。

C 原型：
```
void  CSDe_SetSliderIdac(void)
```

汇编：
```
lcall CSDe_SetSliderIdac
```

参数：

无

返回值：

无

其他影响：

**

CSDe_wGetCentroidPos

说明：

该函数根据差值阵列计算坐标。如果存在，则偏移和长度都存储为临时变量，并根据 CSDe 向导中指定的分辨率计算坐标位置。仅在滑条是由 CSD 向导定义时，此函数才可用。

C 原型：

```
WORD  CSDe_wGetCentroidPos(BYTE bSnsGroup)
```

汇编：

```
mov   A, bSnsGroup
lcall CSDe_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 用于按键。滑条包含在组 1 和更高的组中。

返回值：

滑条的位置数值、 A 中的 LSB 和 X 中的 MSB。

其他影响：

此子程序通过减去噪声阈值来修改差值计数。每次扫描后只能调用此子程序一次，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，此函数会将数值从零恢复为 CSDe 向导中设置的分辨率值。如果没有任何传感器处于活动状态，此函数将返回 –1 （FFFFh）。如果在执行坐标计算 / 双工算法时出现了错误，该函数将返回 –1 （FFFFh）。若需要，可以使用 CSDe_bIsSensorActive() 子程序确定触摸了哪些滑条段。

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

CSDe_wGetRadialPos

说明：

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

C 原型：

```
WORD CSDe_wGetRadialPos(BYTE bSnsGroup)
```

汇编：

```
mov   A, bSnsGroup
lcall CSDe_wGetRadialPos
```

参数：

bSnsGroup A => 组编号

此参数是正在使用的辐射滑条的编号。该编号可以通过辐射滑条表示法的左侧上的 CSDe UM 向导获取 （例如：对于 s2，辐射滑条编号为 2）。

返回值：

辐射状滑条的位置值、 A 中的 LSB 和 X 中的 MSB。

其他影响：

此子程序在每次扫描后只能调用一次，以避免得到负的差值和基准线更新。如果应用监控差值信号，在差值计数数据传输后调用该子程序。

如果某个滑条传感器处于活动状态，此函数会将数值从零恢复为 CSDe 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大于噪声阈值，则此子子程序可能生成假的质心结果。设置噪声阈值时请务必小心 （应使之比噪声级别足够高），以便噪声不会产生假的质心。

CSDe_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。该函数与 CSDe_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  CSDe_wGetRadialInc(BYTE bSnsGroup)
```

汇编：

```
mov   A, bSnsGroup
lcall CSDe_wGetRadialInc
```

参数：

bSnsGroup A => 组编号

该参数是使用的辐射滑条的编号。可以通过 CSDe UM 向导从辐射滑条表示法的左侧获取其编号（例如：s2 表示辐射滑条编号为 2）。

返回值：

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

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

其他影响：

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

示例固件源代码

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

```
//-------------------------------------------------------------------------
// Sample C code for the CSDe 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;
  CSDe_Start();
CSDe_InitializeBaselines() ; //scan all sensors first time, init baseline
  CSDe_SetDefaultFingerThresholds() ;
//
// Loop Forever
//
while (1) {
CSDe_ScanAllSensors(); //scan all sensors in array (buttons and sliders)
    CSDe_UpdateAllBaselines(); //Update all baseline levels;

//detect if any sensor is pressed
 if(CSDe_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 CSDe_waSnsResult[x] <- Raw Counts
// OUTPUT CSDe_waSnsDiff[x] <- Difference
// OUTPUT CSDe_waSnsBaseline[x] <- Baseline
// OUTPUT CSDe_baSnsOnMask[x] <- Sensor On/Off
}
}
```

示例 2. 下面的代码演示了两个传感器配置为用户模块向导时的传感器用途示例。

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

CSDe_Start(); // Start CSDe UM
     CSDe_SetDefaultFingerThresholds(); // Set default thresholds for butt
```

```
// Initialize baseline for sensor number "3"
     CSDe_InitializeSensorBaseline(3);

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

配置寄存器

该 CSDe 用户模块使用了定时器 1、 CapSense 和比较器 PSoC 模块。通过一组寄存器对每个块进行个性化和参数化设置。本节对用户模块使用的一组寄存器进行了简要描述。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相关配置。
  - 定时器 1 数据寄存器 0：PT1_DATA0 可编程定时器数据寄存器 0 （PT1_DATA0）包含了可编程定时器的计数值低 8 位。
  - 定时器 1 数据寄存器 1：PT1_DATA1 可编程定时器的数据寄存器 1 （PT1_DATA1）包含器件的可编程定时器计数值的 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）控制着 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 CapSense 控制寄存器 3 （CS_CR3）包含的控制位主要使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当前的计数值。
- CapSense 计数器高字节寄存器：CS_CNTH CapSense 计数器高字节寄存器 （CS_CNTH）包含高字节计数器的当前计数值。
- CapSense 状态寄存器：CS_STAT CapSense 状态寄存器 （CS_STAT）控制着 CapSense 计数器的各选项。
- CapSense 斜率控制寄存器：CS_SLEW CapSense 斜率控制寄存器 （CS_SLEW）使能和控制着驰张振荡器的快速转换模式。

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）使能和配置了比较器的输入范围。
  - 比较器控制寄存器 1：CMP_CR1 比较器控制寄存器 1 （CMP_CR1）配置比较器的输出选项。
  - 比较器复用寄存器：CMP_MUX 比较器复用寄存器 （CMP_MUX）包含控各个制位，用于比较器 0 和 1 的输入选项。
  - 比较器 LUT 控制寄存器：CMP_LUT 比较器 LUT 控制寄存器 （CMP_LUT）可选择逻辑函数。

受 CSDe 用户模块的影响的附加寄存器

- 组 0
  - 模拟复用器配置寄存器：AMUX_CFG 使用该寄存器来配置积分电容引脚与模拟全局总线间的连接。
  - 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。
  - 当前 DAC 数据寄存器：IDAC_D 该寄存器指定了用于确定输出 IDAC 电流的 8 位乘法因子。
- 组 1
  - 输出至端口 0 寄存器：OUT_P0 该寄存器允许特定的内部信号输出到端口 0 引脚上。

– 输出至端口 1 寄存器：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情况。

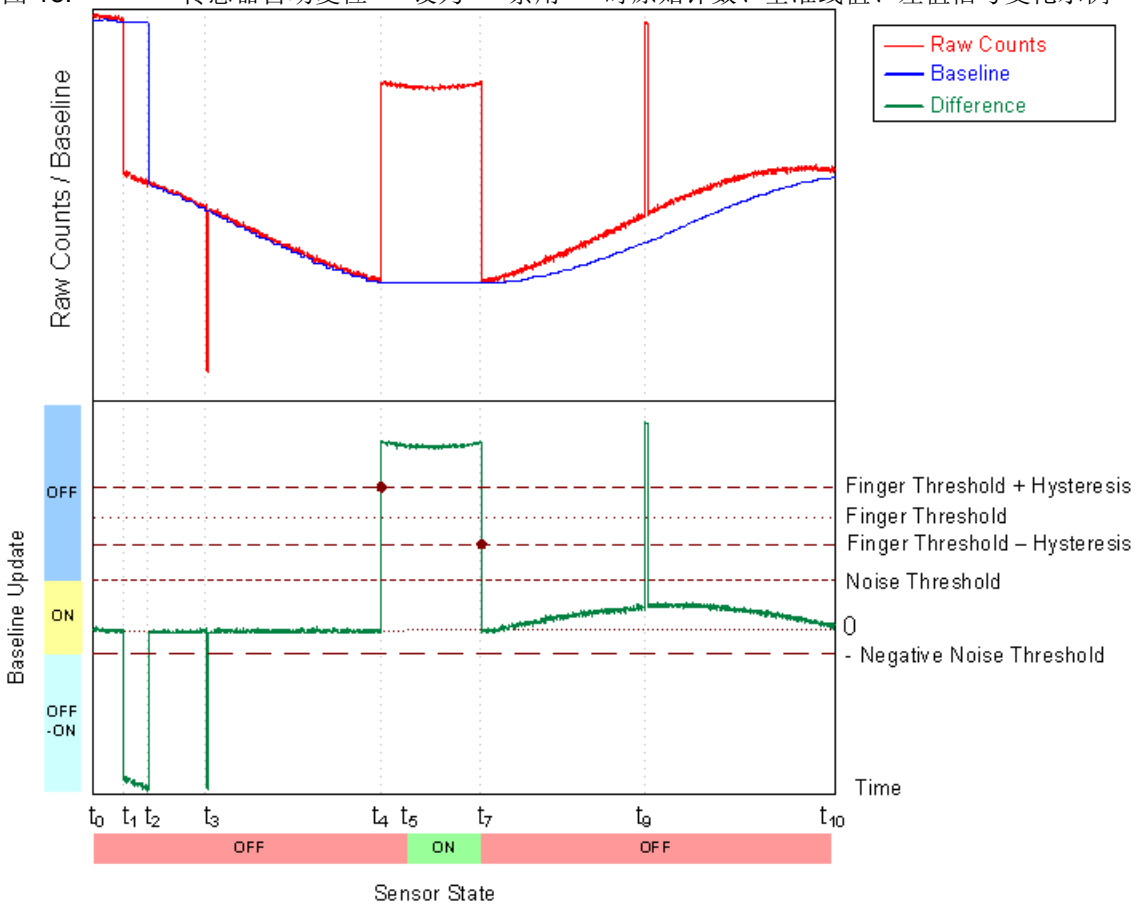
附录

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

CSDe 参数的交互

图 15 和图 16 说明了基准线更新和决策逻辑操作，对于更好地了解如何设置用户参数以获得最佳性能很有帮助。图 15 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 16 显示的是传感器自动复位参数被设置为 **Enabled** （使能）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）。数据是在一些人工测试中收集的，这些测试展现了原始计数慢速和快速变化时的系统操作。慢速变化可能是温度或湿度变化所致，快速变化可能是由传感器触摸、ESD 事件或强射频场的影响触发的。

图 18. “ 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（负噪声阈值）。如果手指位于传感器上时器件加电，过一段时间后手指移开，则会发生这种情况。此时，基准线更新机制冻结，内部超时计数器会激活。当差值信号低于 “ 低基准线复位 ”（LowBaselineReset）样品的 “ 负噪声阈值 ”（NegativeNoiseThreshold）时，基准线复位。这是在 t_2 处发生的。

第二种大的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检测。

传感器是在 t_4 处被触摸的。当差值信号超过 “ 手指阈值 + 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防抖动样品，则传感器状态设置为开启。这是在 t_5 处发生的。当差值信号在 t_7 处下降到 “ 手指阈值 – 迟滞 ”（FingerThreshold – Hysteresis）水平之下时，传感器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，t_9 处的瞬时正值尖峰脉冲由去抖动计数器筛选。

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

图 19. 传感器自动复位参数设置为 “ 使能 ” 时的原始信号、基线、差值信号的变化示例

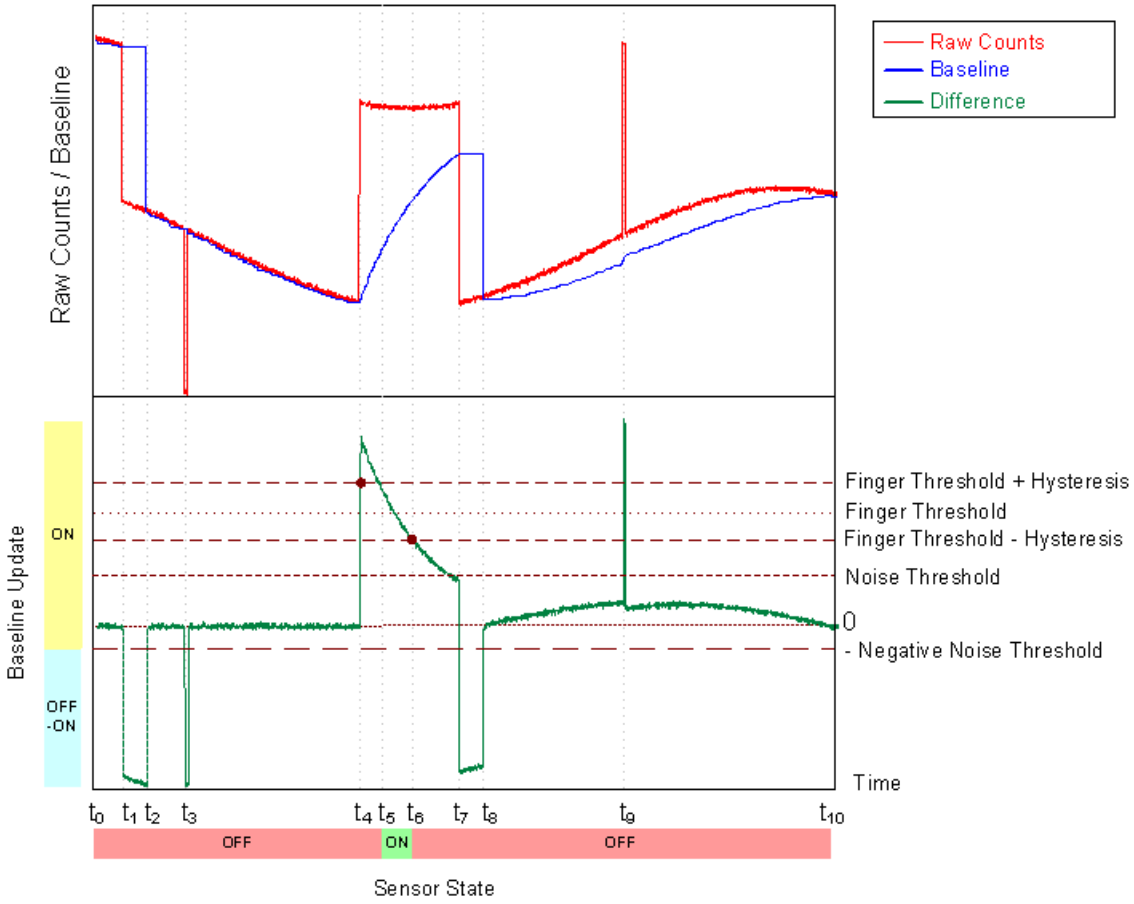


图 16 中的系统操作与上一实例中的操作类似，但有以下区别：

- 在 t_6 处触摸传感器时，触摸持续时间因活动基线更新算法而会减少。
- 手指移开后，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. 纠正了 UM 向导中的分辨率值计算，以便在更改双工后寻址错误。 将 CSD_MODE 位的设置从 ScanSensor API 转移到 Start API。
2.00	DHA	1. 删除 inc 文件中冗余的常数。 2. 更新了 “ 滤波器系数 ” 属性可用值列表。 3. 设置了传感器的 PRS 预充电源和参考传感器的预分频器。 4. 设置了提供给 RefSensor 的预分频器频率。 5. 为避免传感器数量最大时发生编译错误， CSDe_waRawCount 变量被转移到新创建的 RAM3 区内。 6. 更新了 “ 滤波器系数 ” 部分并在数据手册中解释了动态重配置的限制。 7. 更改了 Refsensor 的校准算法。
2.10	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，以便检查负差值计数。 3. 将用户模块转换到传统的状态。

注意 PSoC Designer 5.1 在所有用户模块基本介绍中都引入了 “ 版本历史 ”。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 Cypress Semiconductor Corporation. All Rights Reserved. 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。