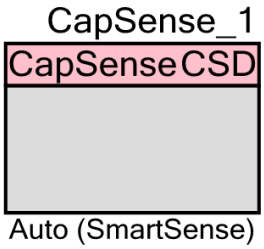
PSoC 4 电容式感应（CapSense® CSD）

2.10



特性

- 一流的信噪比性能
  - 抗介电和辐射外部噪声的出色抗造性能
  - 超低辐射
  - 支持 CapSense 按键：覆盖层厚度为 15 mm 的玻璃和 5 mm 的塑料
- SmartSense™自动调试
  - 在运行时设置并维持最佳传感器性能
  - 开发和量产过程中无需手动调试
- 高级用户界面功能：防水
  - 即使存在水滴的情况下，屏蔽电极仍可保证可靠地运行
  - 在存在水分或流水的情况下，保护传感器可防止误触模
- 支持用户定义的按键、线性滑条、辐射滑条、触摸板和接近电容传感器的组合
- 可轻松使用应用编程接口（API）快速开发原型
- 集成的基于 PC 的图形户界面（GUI）可用于手动调试模式下的调试（请参考 *PSoC® 4 CapSense® 调试指南*。）

注意：本文档选择了围绕 PSoC 4 器件进行讲解。 这里提到的 PSoC 4 指的是 PSoC 4 和 PSoC 4 BLE（蓝牙低能耗）器件。该组件还支持 PRoC BLE 器件。

概述

电容式感应通过使用 Delta-Sigma 调制器（CapSense CSD），能够灵活和有效地测量各种用户界面屏幕应用（如电容式触摸按键、滑条、触摸板、触摸屏和接近感应传感器）中传感器感应电容的微小变化，也可以检测手指触摸。

请阅读随附本数据手册的文档。具体情况请参见赛普拉斯半导体公司网址 www.cypress.com：

- *Capsense 入门*
- *PSoC 4 CapSense 设计指南*

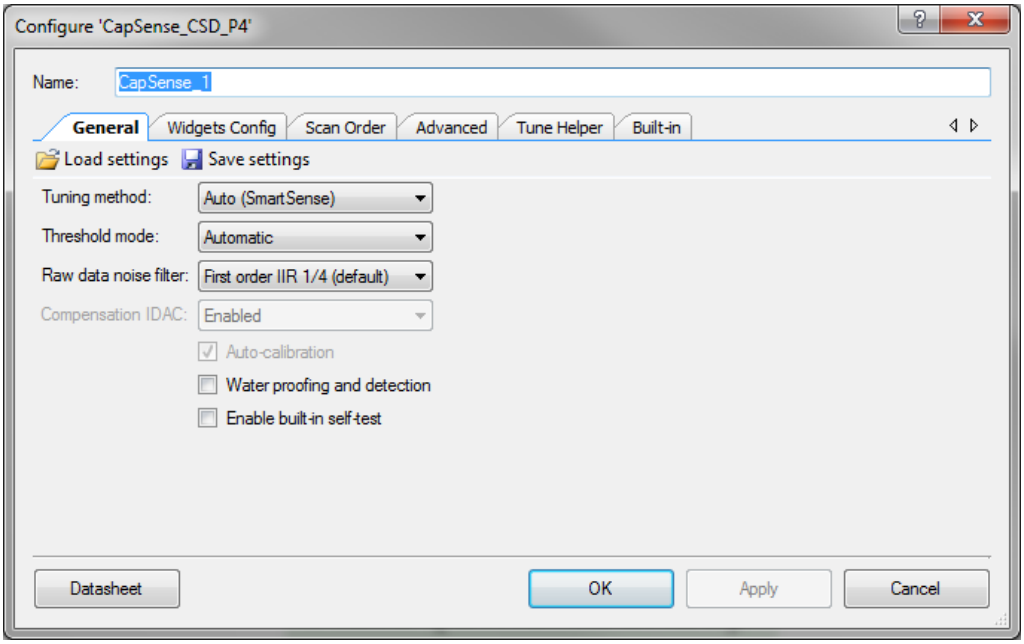
何时使用 CapSense 组件

电容式感应系统可用于多种应用中，以代替传统按键、开关和其它控件；甚至可用于淋雨或潮湿的工作环境的应用中。这些应用包括汽车、室外设备、ATM 机、公共接入系统、手机和 PDA 等便携式设备以及厨房和浴室应用。

组件参数

将 CapSense CSD 组件拖放到您的设计上，然后双击打开 Configure（配置）对话框。该对话框有若干选项卡，可引导您完成对 CapSense CSD 组件的配置过程。

General（常规）选项卡



Load Settings/Save Settings（加载设置/保存设置 ）

Save Settings 用于保存组件中所有当前配置和调试数据。这样，在新项目中就能够快速复制已保存设置到新的项目。 Load Settings **用于加载先前保存的设置。**

可使用被保存的组件设置来导入设置和调试数据。

Tuning method（调试方法）

此参数指定了调试方法。调试操作包括选择具体硬件配置的最佳参数。

其中有三个选项：

- **Auto (SmartSense)**（自动） — 在支持的寄生电容（Cp）范围内（即从 5 pF 到 55 pF）对 CapSenseCSD 组件进行自动调试。

  它是我们对所有设计推荐的调试方法。运行时，固件算法不断地确定最佳的调校参数。在该模式下，需要额外的 RAM 和 CPU 资源。如果需要特定调试（严格控制扫描时间或 Cp 高于 55 pF），请将 **Tuning method** 参数设置为“Manual with Run-Time Tuning”或“Manual”。

  注意：可使用 I^2C 通信进行 SmartSense 调试，该组件在 **Tune Helper**（调试助手）选项卡中指定用于将目标器件中的数据传输到调试器 GUI。

- **Manual with Run-Time Tuning** （运行时手动调试） — 该选项允许您在运行时间中使用调试器 GUI 手动调试 CapSense CSD 组件。通过调试器 GUI 或使用 API 修改调试参数，可以实现运行时调试操作。调试参数被存储在 RAM 中。

  要启动调试器 GUI，请右击符号，然后选择 **Launch Tuner**（启动调试器）。有关手动调试的详细信息，请参考 *PSoC® 4 CapSense® 调试指南*。 手动调试需要 I^2C 通信，该通信在 **Tune Helper**（调试助手）选项卡中指定用于在目标器件和调谐器 GUI 之间传输数据。

- **Manual**（手动） — 该选项用于禁止调试。

  设置为 **Manual** 项（禁止运行时调试）时不允许组件在运行时调试，并且所有可能调试参数被保存在闪存内。

Threshold mode（阀值模式）

当 **Tuning method** 参数被设置为“Auto (SmartSense)”时，通过该参数可以指定阈值模式。选中任何手动调试选项时，该参数均不可用。在手动调试模式下，所有参数均被手动设置。

有两个选项：

- **Automatic**（自动）（**默认**） — 在该模式下，SmartSense 算法会自动计算并设置所有传感器阈值。
- **Flexible**（灵活） — 组件实现的灵活阈值。 在该模式下，组件接受每个 Widget 的“手指阈值”，然后根据手指阈值计算出其它阈值参数：
  - 低基线复位= 30
  - 迟滞 = 手指阈值的 12.5 %
  - 噪声阈值 = 手指阈值的 50%
  - 负噪声阈值 = 手指阈值的 50%

Raw Data Noise Filter（原始数据噪声滤波器）

该参数用于选择原始数据滤波器类型。仅可选择一个滤波器，且该滤波器将应用于所有传感器。在传感器扫描期间，用户应使用滤波器来降低噪声的影响。有关滤波器类型的详情，请参见本文档中的 Filters（滤波器）部分。

- **None**（无） — 不使用滤波器。没有滤波器固件和 SRAM 的开销。
- **Median**（中值滤波） — 按顺序排列最近三个传感器值，并返回中值。
- **Averaging**（均值滤波） — 返回最近三个传感器值的简单均值。
- **First Order IIR 1/2**（一阶 IIR 1/2 滤波） — 返回前一滤波器值的 1/2 与最近传感器值的 1/2 的和。在所有滤波器类型中 IIR 滤波器占用最少的固件和 SRAM 开销。
- **First Order IIR 1/4**（一阶 IIR 1/4 滤波）（默认） — 返回前一滤波器值的 3/4 与最近传感器值的 1/4 的和。
- **First Order IIR 1/8**（一阶 IIR 1/8 滤波） — 返回前一滤波器值的 7/8 与最近传感器值的 1/8 的和。
- **First Order IIR 1/16** （一阶 IIR 1/16 滤波） — 返回前一滤波器值的 15/16 与最近传感器值的 1/16 的和。
- **Jitter**（抖动滤波） — 如果最近传感器值大于上一传感器值，那么先前的滤波器值按 1 递增，如果小于上一传感器值，则递减。

Compensation IDAC（补偿 IDAC）

该参数用于使能基线 IDAC。使能基线 IDAC 能够提高灵敏度和信噪比（SNR）。在 CapSense 操作期间，**Compensation IDAC** 一直被连接到 amuxbus 上，用于补偿传感器的寄生电容。

- Disabled (default)（禁用 — 默认选择）
- Enabled（使能）

注意：对于自动（SmartSense）调试方法，将始终使能 **Compensation IDAC** 参数。

Auto-calibration（自动校准）选框

用于使能或禁用手动调试方法选项的 IDAC 自动校准。默认为 Disabled（禁用）。

Water proofing and detection（防水及检测）

该特性用于配置 CapSense CSD，使其支持防水功能（默认为禁用）。该功能将使能屏蔽电极。其设置参数如下：

- 在 PSoC Creator 设计范围资源引脚编辑器中，使能 Shield output terminal（屏蔽输出端）

  注意：不应该在 SmartSense 调试模式下使用屏蔽电极。
- 添加保护传感器（Guard Widget）

  注意：如果保护 Widget 不需要防水性能，可以在 Advanced（高级）选项卡上禁用它。

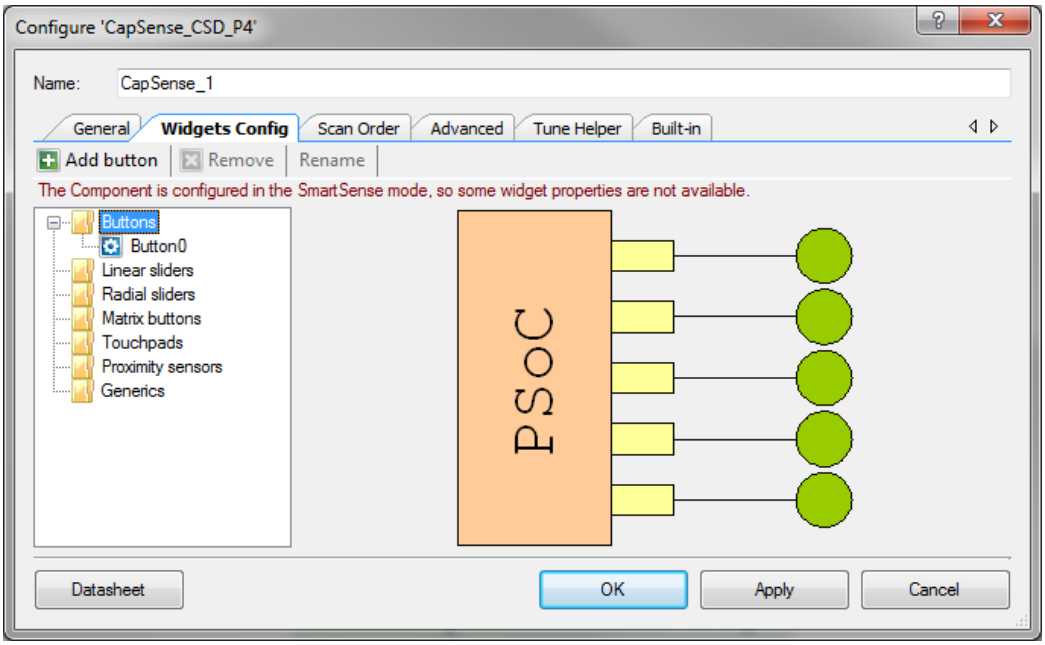
Enable BIST（使能 BIST）

该参数用于使能允许测量 Cp 和 Cmod 的 Built In Self Test（内置自测试 — BIST）API。为正常操作 SmartSense，必须满足下面条件：

- Cmod = 2.2 nF
- 传感器的 Cp 值 < 55 pF

  注意：如果 Cp > 55 pF，您可以使用手动调试方法选项，并根据较高传感器 Cp 调试各个传感器，以便使检测时钟频率满足 5RC 时间常量。

Widget Config 选项卡



有关各种参数的定义， 请参见 功能说明一节中的内容。

工具栏

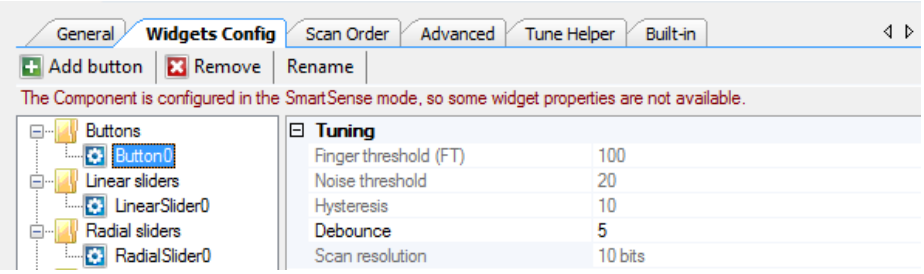
工具栏包含以下指令：

- **Add widget** （添加 Widget）（热键 — Insert）— 向列表中添加选定的 Widget 类型。Widget 类型包括：
  - **Buttons**（按键） — 按键可以检测到单个传感器上存在手指触摸，可以替换机械按键。
  - **Linear Sliders**（线性滑条） — 线性滑条提供一个关于手指触摸位置整数值，该值是通过对若干个传感器上的信号进行插值的方法得到的。
  - **Radial Sliders**（辐射滑条） — 辐射滑条类似于线性滑条，不同之处是传感器置于一个圆圈中。
  - **Matrix Buttons**（矩阵按键） — 矩阵按键检测行传感器和列传感器形成的交叉点处存在的手指触摸。矩阵按键提供了一种扫描大量按键的有效方法。
  - **Touchpads**（触摸板） — 触摸板返回触摸板区域内手指触摸的 X 和 Y 坐标。触摸板包含多个行传感器和列传感器。
  - **Proximity Sensors**（接近感应传感器） — 接近感应传感器经过优化，可在离传感器很远的距离检测是否存在手指、手掌或其它大物体。 这样不需要实际触摸。

- **Generic Sensors**（通用传感器） — 通用传感器提供了来自单个传感器的原始数据（raw data）。这样用户可以创建特殊或高级传感器，但是不能得到其它传感器类型的经过处理的输出。

- **Remove** （删除）（热键 — Delete） — 从列表中删除选定的 Widget。
- **Rename**（重命名）（热键 — F2） — 打开一个对话框，更改选定的 Widget 名称。也可以双击 Widget 以打开该对话框。

Buttons（按键）



Tuning（调试）：

- **Finger Threshold**（手指阈值） — 定义增大或减小触摸灵敏度的传感器活动阈值。当传感器扫描值大于该阈值时，该按键被报告为“有触摸”状态。默认值为 **100**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。

  8 位分辨率 widget 的 **Finger Threshold + Hysteresis** 的值不能大于 254，16 位分辨率 widget 的值不能大于 65534。

- **Noise Threshold**（噪声阈值） — 定义传感器噪声阈值。 如果计数值高于该阈值，则不更新基线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致误触摸或触摸遗漏。如果噪声阈值过高，手指触摸可能会被认为是噪声，人为使基线错误的升高，从而导致手指触摸遗漏。默认值为 **20**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。

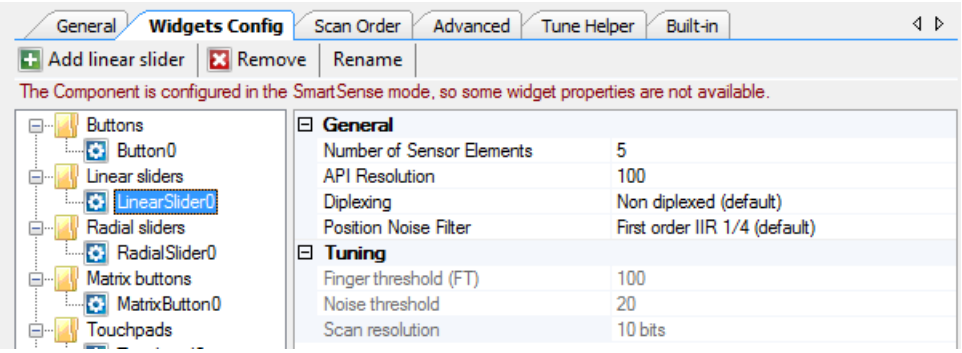
- **Hysteresis**（迟滞）— 添加传感器活动状态转换的差分迟滞。如果传感器处于非活动状态，则差值计数必须大于手指阈值与迟滞的和才被认为是有效的状态转换条件。如果传感器处于活动状态，则差值计数必须低于手指阈值与迟滞的差才被认为是有效的状态转换条件。迟滞有助于确保低振幅传感器噪声和手指少量移动不会导致按键状态的循环。默认值为 **10**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。8 位分辨率 widget 的 **Finger Threshold + Hysteresis** 值不能大于 254，16 位分辨率 widget 的值不能大于 65534。

- **Debounce**（去抖动） — 添加一个去抖动计数器来检测传感器活动状态转换。为了让传感器能够从非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。默认值为 **5**。Debounce（去抖动）确保高频率高振幅的噪声不会导致对按键的误检。取值范围为[1…255]。
- **Scan Resolution**（扫描分辨率） — 定义扫描分辨率。该参数会影响按键 Widget 传感器的扫描时间。N 位的扫描分辨率最大原始计数为 $2^N - 1$。提高分辨率可提高触摸检测的灵敏度和信噪比（SNR），但会延长扫描时间。默认值为 **10 位**。取值范围为[6…16]。

注意：这些参数（Finger Threshold 参数除外）不适用于 SmartSense 模式并由 SmartSense 算法自动设置。在手动模式下，推荐使用下面各值：

- 手指阈值 = 信号值的 80%
- 噪声阈值 = 负噪声阈值 = 手指阈值的 50%（Advanced 选项卡中）
- 迟滞 = 手指阈值的 12.5%
- 去抖动值 = 3
- 低基线复位 = 30（Advanced 选项卡中）

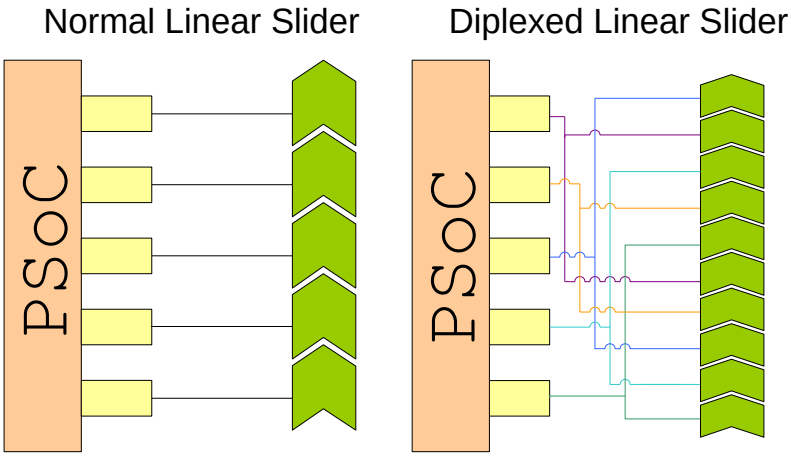
Linear Sliders（线性滑条）



General（通用）：

- **Numbers of Sensor Elements**（传感器元件数量） — 定义滑条内的元件数量。一个良好的 API 分辨率与传感器元件数之间的比例是 20:1。过多增大该比例会造成计算的手指位置处噪声增加。取值范围为[2…32]。默认值为 **5** 个元件。
- **API Resolution**（API 分辨率） — 定义滑条分辨率。 位置值将在该范围以内变化。取值范围为[1…255]。默认值为 100。

- **Diplexing – Non diplexed**（双工 – 非双工）（默认）或 **Diplexed**（双工）。双工模式允许两个滑条传感器共享一个器件引脚，从而减少了给定数量的滑条传感器所需的引脚总数。每个双工滑条的传感器元件的最少数量为 5 个。



- **Position Noise Filter**（位置噪声滤波器） — 选择噪声滤波器类型以进行位置计算。对于一个选定的 Widget，只能应用一个滤波器。有关滤波器类型的详情，请参见本文档中的 Filters 滤波器部分。
  - **None**（无）
  - **Median**（中值滤波）
  - **Averaging**（均值滤波）
  - **First Order IIR 1/2**（一阶 IIR 1/2 滤波）
  - **First Order IIR 1/4**（一阶 IIR 1/4 滤波，默认）
  - **Jitter**（抖动滤波）

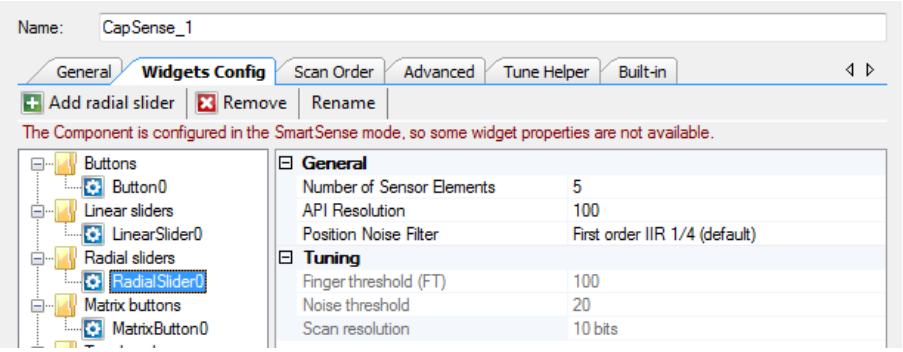
Tuning（调试）：

- **Finger Threshold**（手指阈值） — 定义增大或减小触摸灵敏度的传感器活动阈值。 当传感器扫描值大于该阈值时，该按键被报告为“有触摸”状态。默认值为 **100**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。
- **Noise Threshold**（噪声阈值） — 定义滑条元素的传感器噪声阈值。如果计数值高于该阈值，则不更新基线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致误触摸或触摸遗漏。如果噪声阈值过高，手指触摸可能会被认为是噪声，且认为增加基线，导致中心位置计算错误。质心计算中不包括低于该阈值的计数值。默认值为 **20**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。

- **Scan Resolution**（扫描分辨率） — 定义扫描分辨率。该参数对线性滑条 Widget 内所有传感器的扫描时间会产生影响。N 位的扫描分辨率的最大原始计数为 $2^N - 1$。 提高分辨率可提高触摸检测的灵敏度和信噪比，但会延长描时间。默认值为 **10 位**。取值范围为[6…16]。

注意： Noise Threshold 和 **Scan Resolution** 参数不适用于 SmartSense 模式并由 SmartSense 算法自动设置。

辐射滑条



General（通用）：

- **Numbers of Sensor Elements**（传感器元件数量） — 定义滑条内的元件数量。一个良好的 API 分辨率与传感器元件之间的比例是 20:1。过多增大该比例会造成计算分辨率时噪声增加。取值范围为[2…32]。默认值为 **5** 个元件。
- **API Resolution**（API 分辨率） — 定义滑条的分辨率。位置值将在该范围以内变化。取值范围为[1…255]。默认值为 100。
- **Position Noise Filter**（位置噪声滤波器） — 选择噪声滤波器类型以进行位置计算。对于一个选定的 Widget，仅可应用一个滤波器。有关滤波器类型的详情，请参见本数据手册中功能说明一节的滤波器部分。
  - None（无）
  - Median（中值滤波）
  - Averaging（均值滤波）
  - First Order IIR 1/2（一阶 IIR 1/2 滤波）
  - First Order IIR 1/4 (default)（一阶 IIR1/4 滤波，默认）
  - Jitter （抖动滤波）

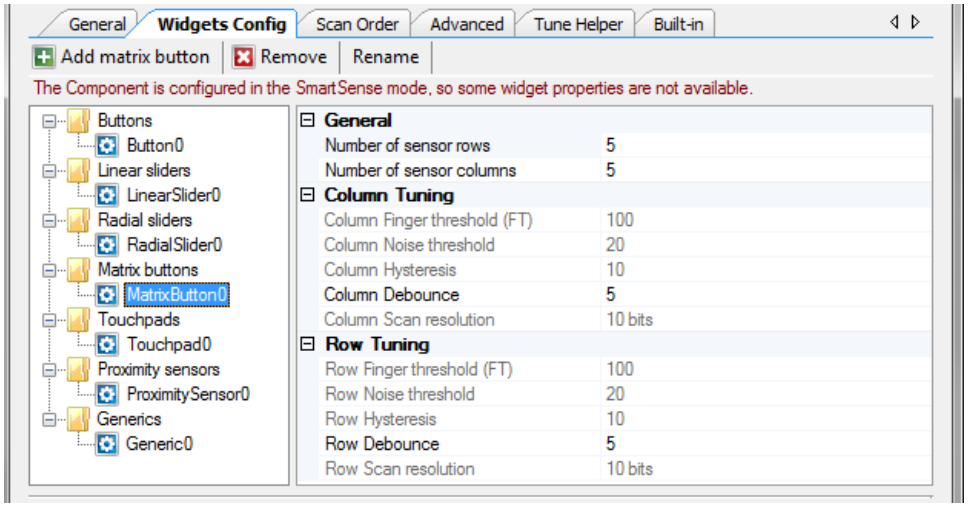
Tuning（调试）：

- **Finger Threshold**（手指阈值） — 定义增加或减少触摸灵敏度的传感器活动阈值。 当传感器扫描值大于该阈值时，该按键被报告为“有触摸”状态。默认值为 **100**。
- **Noise Threshold**（噪声阈值） — 定义滑条元素的传感器噪声阈值。如果计数值高于该阈值，则不更新基线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致误触摸或触摸遗漏。如果噪声阈值过高，手指触摸可能会被认为是噪声，且认为增加基线，导致中心位置计算错误。质心计算中不包括低于该阈值的计数值。默认值为 **20**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。
- **Scan Resolution**（扫描分辨率） — 定义扫描分辨率。该参数对辐条滑块 widget 内所有传感器的扫描时间会产生影响。N 位的扫描分辨率的最大原始计数为 $2^N - 1$。 提高分辨率可提高触摸检测的灵敏度和信噪比，但会延长描时间。默认值为 **10 位**。取值范围为[6…16]。

注意： Noise Threshold 和 **Scan Resolution** 参数不适用于 SmartSense 模式并由 SmartSense 算法自动设置。

注意：建议不将 Position Noise Averaging（位置噪声均值滤波）和 IIR 滤波器用于辐射滑条，因为这些滤波器使用上一次数据更新当前数据。当手指从最后滑条段式移到第一个滑条段式时，这样的更新操作可能导致错误的位置计算。

矩阵按键



General（通用）：

- **Number of sensor columns and rows**（列/行传感器数量） — 定义形成矩阵的列和行的数量。取值范围为[2…32]。列和行的元件数量默认值为 **5**。

Tuning（调试）：

- **Column and Row Finger Threshold**（列/行手指阈值） — 定义相关触摸灵敏度增大或减小的 matrix button（矩阵按键）列和行的传感器活动阈值。当传感器扫描值大于该阈值时，该按键被报告为“有触摸”状态。默认值为 **100**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。8 位分辨率 widget 的 **Finger Threshold + Hysteresis** 的值不能大于 254，16 位分辨率 widget 的值不能大于 65534。

- **Column and Row Noise Threshold**（列/行噪声阈值）— 定义矩阵按键列和行的传感器噪声阈值。如果计数值高于该阈值，则不更新基线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致误触摸或触摸遗漏。如果噪声阈值过高，手指触摸可能会被认为是噪声，且认为增加基线。这样会导致触摸遗漏。默认值为 **20**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。

- **Column and Row Hysteresis** （列/行迟滞） — 添加矩阵按键列和行的传感器活动状态转换的差分迟滞。如果传感器处于非活动状态，则差值计数必须大于手指阈值与迟滞的和才被认为是有效的状态转换条件。如果传感器处于活动状态，则差值计数必须低于手指阈值与迟滞的差才被认为是有效的状态转换条件。迟滞有助于确保低振幅传感器噪声和手指少量移动不会导致按键状态的循环。默认值为 **10**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1…65535]。8 位分辨率 widget 的 **Finger Threshold + Hysteresis** 的值不能大于 254，16 位分辨率 widget 的值不能大于 65534。

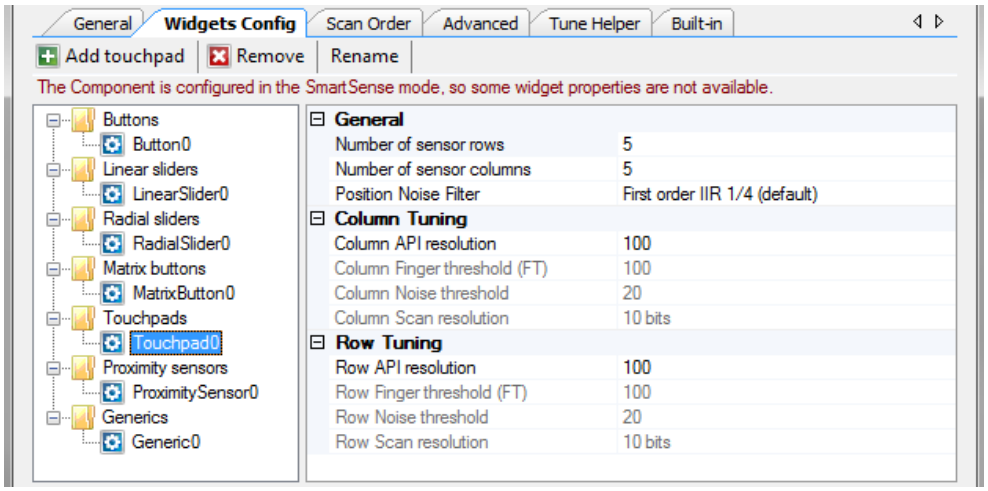
- **Column and Row Debounce** （列/行去抖动） — 添加矩阵按键列/行的传感器活动状态切换检测的去抖动计数器。为了让传感器能够从未激活状态切换为激活状态，在规定的样本数量内，差异计数值必须大于手指阈值与迟滞之和。默认值为 **5**。Debounce（去抖动）确保高频率高振幅的噪声不会导致对按键的误检。取值范围为[1…255]。

- **Column and Row Scan Resolution**（列/行扫描分辨率） — 定义 matrix button（矩阵按键）列和行的扫描分辨率。该参数对矩阵按键 widget 列/行内所有传感器的扫描时间产生影响。N 位的扫描分瓣率的最大原始计数为 $2^N - 1$。提高分瓣率可提高触摸检测的灵敏度和信噪比，但会延长描时间。列和行扫描分辨率应相同，以获得相同的灵敏度水平。默认值为 **10 位**。取值范围为[6…16]。

注意：Noise Threshold、Hysteresis、Debounce 和 Scan Resolution 参数不适用于 SmartSense 模式并由 SmartSense 算法自动设置。在手动模式下，推荐使用下面各值：

- 手指阈值 = 信号值的 80%
- 噪声阈值 = 负噪声阈值 = 手指阈值的 50%（Advanced 选项卡中）
- 迟滞 = 手指阈值的 12.5%
- 去抖动值 = 3
- 低基线复位 = 30（Advanced 选项卡中）

Touchpad（触摸板）



General（通用）：

- **Numbers of sensor columns and rows**（列/行传感器数量）— 定义组成触摸板的列和行的数量。取值范围为[2…32]。列和行的元件数量默认值均为 **5**。
- **Position Noise Filter**（位置噪声滤波器） — 将噪声滤波器添加到位置计算中。对于一个选定的 Widget，仅可应用一个滤波器。有关滤波器类型的详情，请参见本数据手册中功能说明一节的滤波器部分。
  - None（无）
  - Median（中值滤波）
  - Averaging（均值滤波）
  - First Order IIR 1/2（一阶 IIR 1/2 滤波）

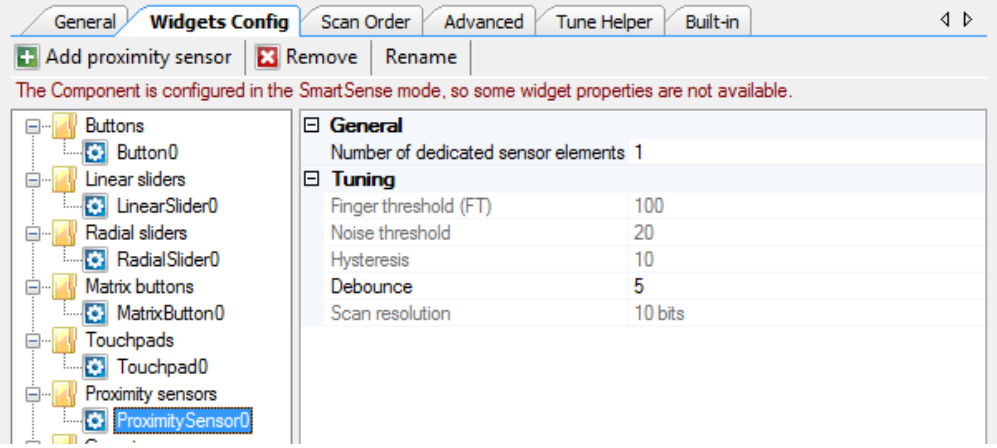
- First Order IIR 1/4 (default)（一阶 IIR1/4 滤波，默认）
- Jitter （抖动滤波）

Tuning（调试）：

- **Column and Row API Resolution**（列/行 API 分辨率） — 定义触摸板列和行的分辨率。在该范围内将报告手指的位置值。默认值为 **100**。取值范围为[1…255]。
- **Column and Row Finger Threshold**（列/行手指阈值） — 定义导致触摸灵敏度增大或减小的触摸板列和行的传感器活动阈值。当传感器扫描值大于该阈值时，触摸板将报告触摸位置。默认值为 **100**。8 位分辨率 widget 的有效值范围是[1…255]，16 位分辨率 widget 的有效值范围是[1...65535]。
- **Column and Row Noise Threshold**（列/行噪声阈值） — 定义触摸板列和行的传感器噪声阈值。如果计数值高于该阈值，则不更新基线。质心位置计算不包括低于该阈值的计数值。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致误触摸或触摸遗漏。如果噪声阈值过高，手指触摸可能会被认为是噪声，且认为增加基线。这样可造成质心计算错误。默认值为 **20**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1…65535]。
- **Column and Row Scan Resolution**（列/行扫描分辨率） — 定义触摸板列和行的扫描分辨率。该参数对触摸板 widget 列/行内所有传感器的扫描时间产生影响。N 位的扫描分辨率的最大原始计数为 $2^N - 1$。 提高分辨率可提高触摸检测的灵敏度和信噪比，但会延长扫描时间。列和行扫描分辨率应相等，以获得相同的灵敏度水平。默认值为 **10 位**。取值范围为[6…16]。

注意： Noise Threshold 和 **Scan Resolution** 参数不适用于 SmartSense 模式并由 SmartSense 算法自动设置。

Proximity Sensors（接近感应传感器）



注意：除了接近 Widget 外，所有 Widget 默认情况下为使能状态。由于接近 Widget 的扫描时间较长，不符合其它 Widget 类所需的快速响应，因此必须在 API 中手动使能接近 Widget。使用 CapSense_EnableWidget()函数使能接近感应 widget。欲了解有关接近感应传感器的更多信息，请参见如何使用接近感应传感器的内容。

General（通用）：

- **Number of Dedicated Sensor Elements**（专用传感器元件数量） — 选择专用接近感应传感器的数量。这些传感器单元不包括用于其它 Widget 类型的传感器。任何 Widget 传感器均可单独使用，或并行连接来实现接近感应传感器。
  - **0** — 接近感应传感器仅扫描一个或多个现有传感器来确定接近情况。没有任何新传感器被分配给该 Widget。
  - **1**（默认） — 系统中专用接近感应传感器的数量。所有专用传感器将组成一个复合接近感应传感器，并由通用参数扫描。

Tuning（调试）：

- **Finger Threshold**（手指阀值） — 定义导致触摸接近灵敏度增大或减小的传感器的活动阈值。当传感器扫描值大于此阈值时，该接近感应传感器报告为有触摸。默认值为 **100**。8 位分辨率 widget 的有效值范围是[1…255]，16 位分辨率 widget 的有效值范围是[1...65535]。8 位分辨率 widget 的 **Finger Threshold + Hysteresis** 的值不能大于 254，16 位分辨率 widget 的值不能大于 65534。
- **Noise Threshold**（噪声阀值） — 定义传感器噪声阀值。 如果计数值高于该阈值，则不更新基线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致触摸错误或触摸遗漏。如果噪声阈值过高，手指触摸可能会被认为是噪声，且认为增加基线。这样会导致触摸遗漏。8 位分辨率 widget 的有效值范围是[1…255]，16 位分辨率 widget 的有效值范围是[1...65535]。默认值为 20。

- **Hysteresis**（迟滞） — 添加传感器活动状态转换的差分迟滞。如果传感器处于非活动状态，则差值计数必须大于手指阈值与迟滞的和才被认为是有效的状态转换条件。如果传感器处于活动状态，则差值计数必须低于手指阈值与迟滞的差才被认为是有效的状态转换条件。迟滞有助于确保低振幅传感器噪声以及手指和身体的小幅移动不会导致接近感应传感器状态的循环。8 位分辨率 widget 的有效值范围是[1…255]，16 位分辨率 widget 的有效值范围是[1...65535]。默认值为 10。

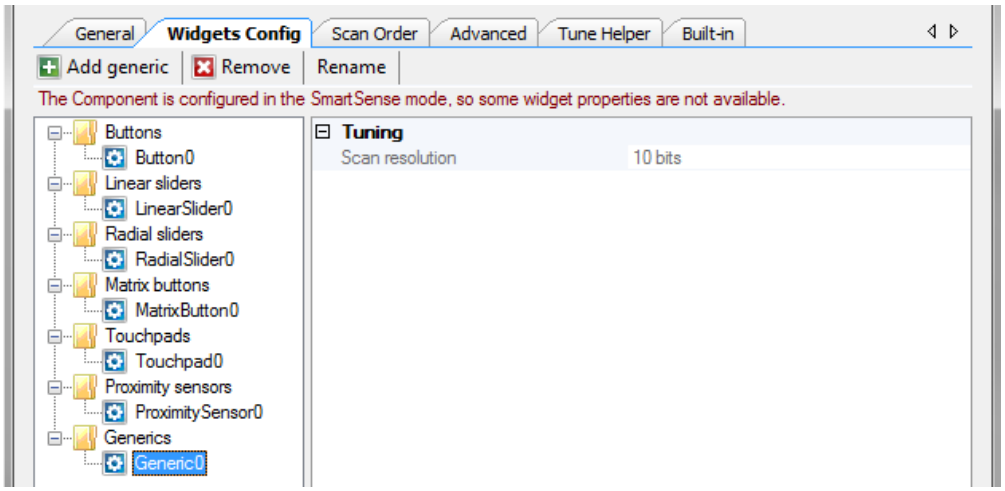
- **Debounce**（去抖动） — 添加一个去抖动计数器来检测传感器活动状态转换。为了让传感器能够从未激活状态切换为激活状态，在规定的样本数量内，差异计数值必须大于手指阈值与迟滞之和。去抖动确保高频率高振幅的噪声不会导致对接近事件的误检。取值范围为[1…255]。默认值为 5。

- **Scan Resolution**（扫描分辨率） — 定义扫描分辨率。该参数对接近 Widget 的扫描时间产生影响。N 位的扫描分辨率的最大原始计数为 $2^N - 1$。提高分辨率可提高触摸检测的灵敏度和信噪比，但会增加扫描时间。我们建议接近感应检测使用的分辨率应比典型按键所使用的分辨率更高，以增大检测范围。默认值为 **16 位**。取值范围为[6…16]。

注意：Noise Threshold、Hysteresis、Debounce 和 **Scan Resolution** 参数不适用于 SmartSense 模式并由 SmartSense 算法自动设置。在手动模式下，推荐使用下面各值：

- 手指阈值 = 信号值的 80%
- 噪声阈值 = 负噪声阈值 = 手指阈值的 50%（Advanced 选项卡中）
- 迟滞 = 手指阈值的 12.5%
- 去抖动值 = 3
- 低基线复位 = 30（Advanced 选项卡中）

Generics（通用）

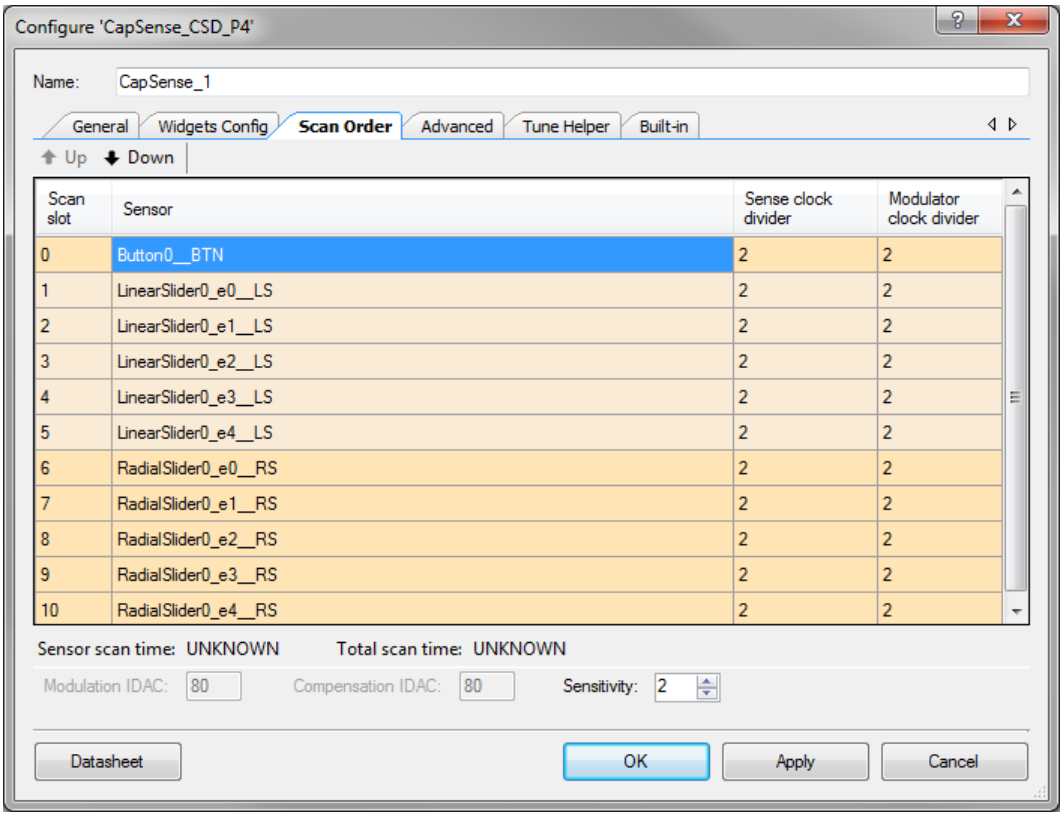


Tuning（调试）：

- **Scan Resolution**（扫描分辨率） — 定义扫描分辨率。该参数会影响通用 Widget 的扫描时间。N 位的扫描分辨率最大原始计数为 $2^N - 1$。提高分辨率可提高触摸检测的灵敏度和 SNR（信噪比），但会增加扫描时间。默认值为 **10 位**。

为每个 Generic Widget 仅提供一个调试选项，因为所有高级处理均留给客户，以支持不适于任何预定义 Widget 的 CapSense 传感器和算法。

Scan Order（扫描顺序）选项卡



注意：扫描顺序对性能不产生任何影响；默认扫描顺序符合大多数应用。

工具栏

工具栏包含以下指令：

- **Up/Down**（上/下）（热键 — 加/减） — 使选定的 Widget 在数据网格中上下移动。如果选定了 Widget 的一个或多个元件，则将选定整个 Widget。

注意：如果扫描顺序变化，则应重新分配引脚。

注意：默认情况下接近感应传感器不参与扫描过程。 其扫描必须在运行时手动启动，因为它通常与其它传感器不同时扫描。

其它热键：

- **Ctrl + A —** 选择所有传感器。
- **Delete —** 从复合传感器中删除所有传感器（仅适用于通用和接近 widget）。

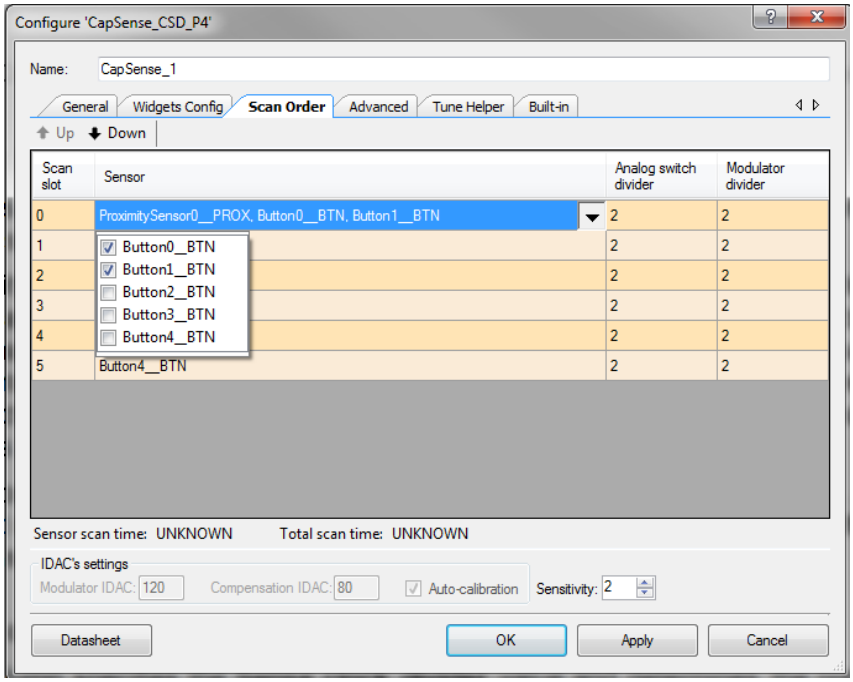
Widget List （Widget 列表）

Widget 以灰色和橙色交替行的形式在表中列出。与 Widget 相关的所有传感器使用相同的颜色，以高亮显示不同的 Widget 元件。

Complex sensors（复合传感器）

接近扫描传感器可使用专用接近感应传感器，或它们可以从专用传感器和/或其他传感器的组合中检测接近性。这样的复合传感器组成一个传感器扫描插槽，其中所有专用传感器在扫描过程中都有相同的参数。

例如，电路板可能有一根围绕着按键阵列的线路，且接近感应传感器可能由该线路和阵列中的所有按键组成。所有这些传感器均同时扫描，以检测接近感应。在接近扫描传感器上提供了下拉菜单列表，可选择进行扫描一个或多个传感器，以检测接近感应。通过勾选下拉菜单列表中每个传感器的相应选框，可以将这些传感器分配给复合接近感应传感器。



与接近感应传感器一样，通用传感器也可由多个传感器组成。通用传感器可从专用传感器、任何其它现有传感器或多个传感器中获取数据。从提供的下拉菜单列表中选择传感器。

Sense clock divider（感应时钟分频器）

该列指定 **Sense clock divider** 值，并确定扫描槽的预充电开关输出频率。PSoC 4100/PSoC 4200 器件的取值范围为[2…255]，PSoC 4000 器件的取值范围为[1…255]。默认值为 2。

如果（在 **Advanced** 选项卡上的）**Individual frequency setting**（单一频率设置）被禁用，则该列会隐藏。

感应时钟分频器是合理调试 Capsense 设计的最重要的硬件参数。它取决于选定的 HFCLK（IMO）和将被扫描的传感器的 Cp 值。下表显示的是基于这些参数的感应时钟分频器的推荐设置：

Cp（pF）	PSoC 4000			PSoC 4100/PSoC 4200 [1]		
	12 MHz	6 MHz	3 MHz	48 MHz	24 MHz	12 MHz
<15	1	1 [2]	1 [2]	2	2 [2]	2 [2]
16-34	2	1	1 [2]	4	2	2 [2]
35-60	4	2	1	8	4	2

Modulator clock divider（调制器时钟分频）

该列指定 **Modulator clock divider** 值，并确定扫描槽的调制器输入频率。PSoC 4100/PSoC 4200 器件的取值范围为[2…255]，PSoC 4000 器件的取值范围为[1…255]。默认值为 2。

如果（在 **Advanced** 选项卡上的）**Individual frequency setting**（单一频率设置）被禁用，则该列会隐藏。

有关时钟配置的详情，请参见功能说明一节的 CapSense 时钟部分。

传感器扫描时间和总扫描时间标签

传感器扫描时间标签显示了选定传感器的硬件扫描时间：

$(2^{resolution}-1)$ / 调制器时钟

总扫描时间是所有传感器扫描时间的总和。

注意：这些标签显示不包含处理时间的扫描时间。

在自动（Smartsense）调试模式下，不会显示扫描时间。它取决于分辨率，在自动（Smartsense）模式中自动设置该分辨率。传感器扫描时间章节提供了自动（Smartsense）调试模式中的传感器扫描时间和分辨率值。

---

[1] 在 PSoC 4100/PSoC 4200 器件中，感应时钟还取决于调制器时钟分频器的值，因为这些分频器值是相关联的。调制器时钟分频器的数据值为 2。更多详细的信息，请参考 CapSense Clocking（CapSense 时钟）部分。

[2] 不推荐将感应时钟值和 Cp 值结合使用，因为开关频率会过低，不能提供良好的性能。对于该 Cp 值，建议增加 HFCLK 频率。

Modulation IDAC（调制 IDAC）

该字段指定调制 IDAC 值。4x 范围的有效范围介于 0 至 255（对于 PSoC 4100/ PSoC 4200 器件为 0 至 250）之间，8x 范围的有效范围为 0 至 125。默认值为 **80**。有关 IDAC 配置的详情，请参见本数据手册中功能说明一节的 CapSense 模拟系统部分。

Compensation IDAC（补偿 IDAC）

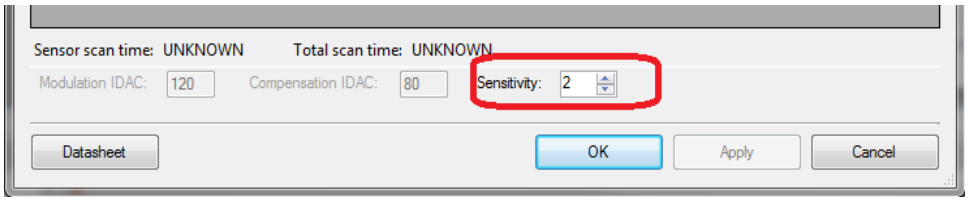
该字段指定步长 IDAC 值。有效范围为 0 至 127。默认值为 **80**。

注意：感应时钟分频、调制器时钟分频、补偿 IDAC 和调制 IDAC 等参数均不适用于 SmartSense 模式。有关 SmartSense 和手动模式中的其它调试信息，请参见 *PSoC® 4 CapSense® 调试指南*。

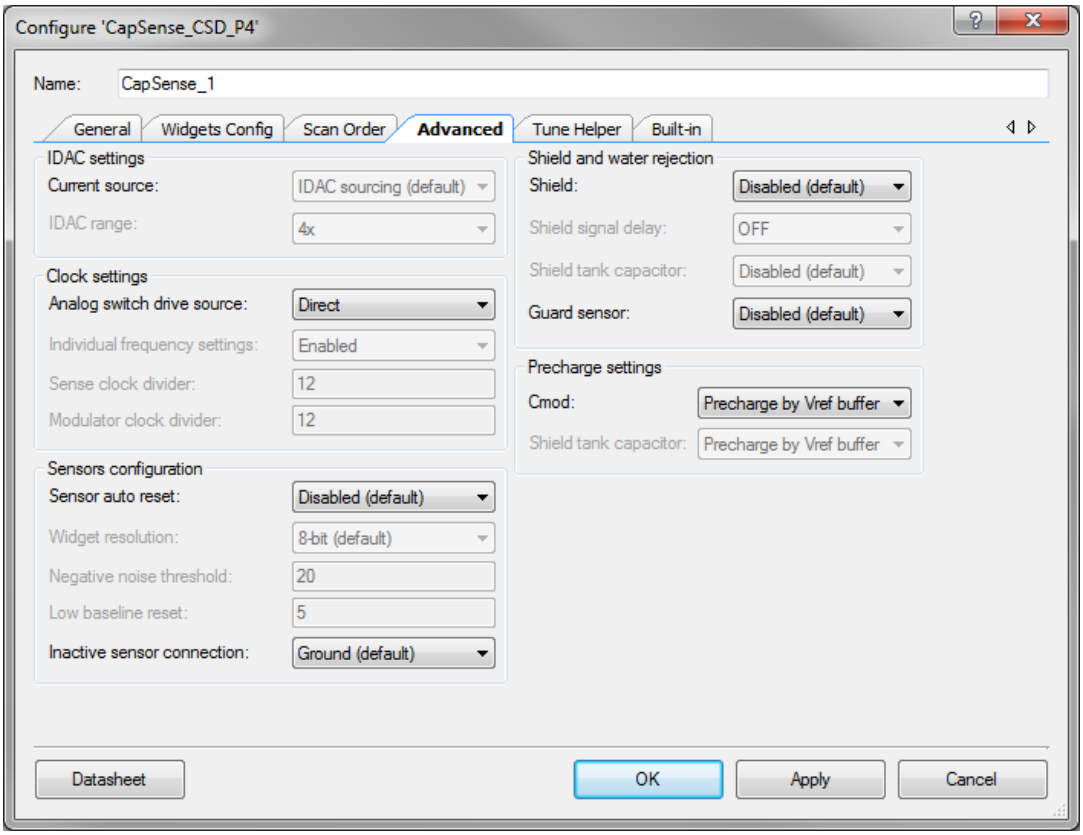
Sensitivity（灵敏度）

SmartSense 模式中的 **Sensitivity** 参数表示激活传感器时所需的 Cs（传感器电容）的额定变化。取值范围为[1…10]，对应的灵敏度级别为：0.1、0.2、0.3 以及 1 pF。默认值为 2。推荐的范围为 0.1 - 0.4 pF。针对覆盖层材料的不同厚度，通过灵敏度参数设置传感器的整体灵敏度。材料越厚，灵敏度值越低。

Sensitivity 参数仅适用于自动（Smartsense）调试模式：



Advanced（高级）选项卡



Current Source（电流源）

CapSense CSD 组件要求高精度的电流源，以检测传感器上的触摸。**IDAC Sinking** 和 **IDAC Sourcing** 需要使用 PSoC 器件上的 IDAC。

- **IDAC Sourcing**（默认） — IDAC 将电流提供给调制电容 C_{MOD}。模拟开关经配置在调制电容 C_{MOD} 和 GND 之间交替，提供灌电流。由于 **IDAC Sourcing** 能提供最大的信噪比，因此推荐将它使用于大部分设计。
- **IDAC Sinking** — IDAC 从调制电容 C_{MOD} 灌入电流。模拟开关经配置在 V_{DD} 和调制电容 C_{MOD} 之间交替，提供电流源。尽管信噪比通常没有 **IDAC Sourcing** 模式高，但该模式适用于大多数设计。

IDAC range（IDAC 范围）

该参数指定 **Current Source**（电流源）的 IDAC 范围。较低和较高的电流范围通常仅用于非触控电容式传感器。

- 4x（默认值）

- 8x

Analog Switch Drive Source（模拟开关驱动源）

该参数指定 **Sense Clock Divider**（感应时钟分频器）的来源，以确定传感器切换至/从调制电容 C_{MOD} 切换出的速率。

- Direct (default) （直接 — 默认值）
- PRS-8b
- PRS-12b
- PRS-Auto

注意：请参考 *PSoC® 4 CapSense® 调试指南*中“CapSense 调试过程”部分，以确定何时能直接使用时钟或 PRS。

Individual Frequency Settings （单独频率设置）

该参数定义了 **Sense Clock Divider**（感应时钟分频）的使用。如果使能该参数，每个扫描槽将使用特定的感应时钟分频值（在 **Scan Order** 选项卡下设置）。否则，传感器只使用该参数下面设置的 **Sense Clock Divider** 值和 **Modulator Clock Divide** 值。如果各个传感器的寄生电容不同，推荐使用单独频率设置。

Sense Clock Divider（感应时钟分频）

该参数指定 **Sense Clock Divider** 值，并确定预充电开关输出频率。PSoC 4100/PSoC 4200 器件的取值范围为[2…255]，PSoC 4000 器件的取值范围为[1…255]。默认值为 **12**。

如果**使能 Individual Frequency Settings** 项，该特性不可用。

传感器以预充电时钟的速度不断切换至或从调制电容 C_{MOD} 切换出。**Sense Clock Divider** 对 CapSense CSD 时钟进行分频，产生预充电时钟。当分频器值减少时，传感器以更快的速度切换，原始计数增加，反之亦然。

有关时钟配置的详情，请参见本数据手册中 CapSense 时钟部分。

Modulator Clock Divider（调制器时钟分频）

该参数用于指定 **Modulator Clock Divider** 值，并确定调制器输入频率。PSoC 4100/PSoC 4200 器件的取值范围为[2…255]，PSoC 4000 器件的取值范围为[1…255]。默认值为 **12**。

分频值越低，扫描时间越短，反之亦然。

如果**使能 Individual Frequency Settings** 项，该特性不可用。

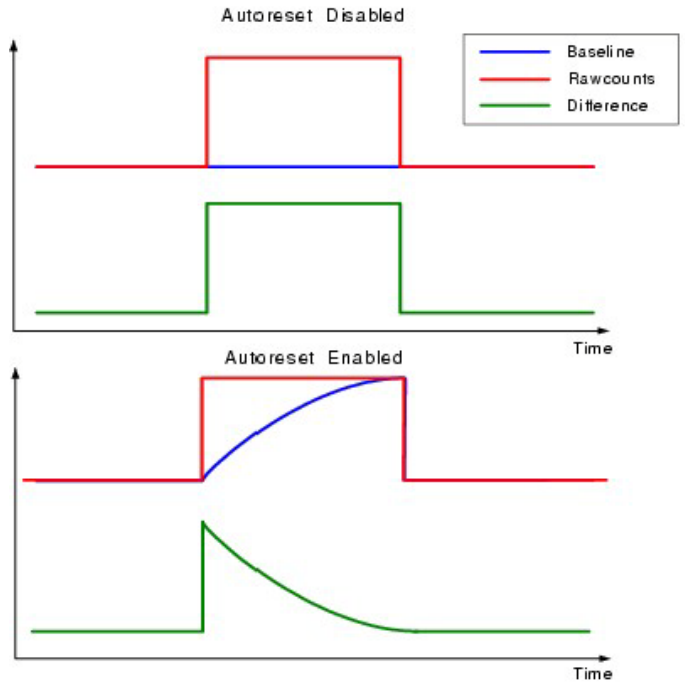
注意：在 PSoC 4100/PSoC 4200 器件中，**Modulator Clock Divider** 值要为 **Sense Clock Divider** 值的倍数，因为这些分频值是相关联的。更多有关信息，请参考 CapSense Clocking（CapSense 时钟）一节。

Sense Clock Divider 和 **Modulator Clock Divider** 在 SmartSense 模式中不可用。有关 SmartSense 和手动模式中的其它调试信息，请参见 *PSoC® 4 CapSense® 调试指南*。

Sensor Auto Reset（传感器自动复位）

该参数使能自动复位，从而使基线随时更新，无论差值计数高于或低于噪声阈值。如果禁用自动复位，则只有差值计数在正/负噪声阈值（噪声阈值为镜面值）以内时，基线才会更新。 如果在无传感器触摸而原始信号突然上升时，传感器始终打开，要使能该参数，否则应将其保留为 **Disabled**（禁用）。

- **Enabled**（使能） — 自动复位确保基线随时更新，从而避免按键按压遗漏和按键卡住，但限制了报告按键被按压的最大持续时间。此设置限制传感器的最大持续时间（典型值为 5 到 10 秒），但是当无任何传感器触摸而原始信号（raw count）突然上升时，可以阻止传感器始终打开。原始信号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。
- **Disabled**（禁用）（默认） — 系统的异常会通过持续超出噪声阈值的方式使基线停止更新。从而造成按键按压遗漏或按键卡住。其优势是，按键可以持续报告其被按压状态。设计人员可能需要提供一种确定按键是否卡住或无反应的应用方法。



Widget Resolution（Widget 分辨率）

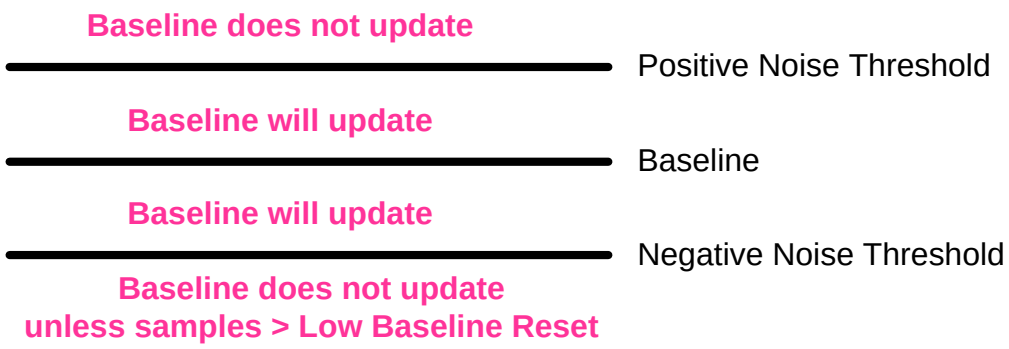
该参数定义了 widget 所报告的信号分辨率。8 位（1 字节）是默认选项，适用于绝大多数应用场合。如果 Widget 值超出 8 位范围，则系统过于敏感，应进行调试，将额定值移到近似中间范围（~128）。要求高精度的滑条和触控板 Widget 可受益于 16 位分辨率。通过避免 8 位分辨率可能出现的舍入误差，16 位分辨率可增加线性度，但其副作用是每个传感器会多使用两个字节的 SRAM。

- 8 位（1 字节）— 默认
- 16 位（2 字节）

Negative Noise Threshold（负噪声阈值）

该参数指定原始计数和基线级别之间的负差值，用于将基线复位到原始计数级别。如果原始计数低于该级别，基线将不复位，除非达到 **Low Baseline Reset**（低基线复位）参数的限制。在这种情况下，将复位基线。下图显示的是噪声阈值和基线复位之间的关系。负噪声阈值的良好起点是使用噪声阈值的起点。

取值范围为[5…255]。默认值为 20。



Low Baseline Reset（低基线复位）

该参数定义了复位基线到原始计数（raw count）级别必须经过的采样次数，这种采样中原始计数（raw count）必须小于基线值。取值范围为[1…255]。默认值为 5。

Inactive Sensor Connection（非活动状态传感器连接）

该参数定义了没有经过主动扫描的所有传感器的默认传感器连接。

- **Ground**（默认）— 应当用于绝大多数应用场合，因为这样可以减少主动扫描的传感器噪声。
- **Hi-Z Analog**（高阻抗模拟）— 使非活动状态传感器处于高阻抗模式。
- **Shield**（屏蔽） — 给所有未扫描传感器提供屏蔽波形. 屏蔽信号的振幅等于扫描传感器上的信号振幅。当与屏蔽电极一起使用时，可增强防水能力并降低噪声。如果 **Shield**（屏蔽）被**禁用**，该功能不可用。

  注意：当将“Shield”设置为“使能”时，非活动状态传感器连接会改为屏蔽。

Shield（屏蔽）

该参数指定使能或禁用屏蔽电极输出，屏蔽电极输出用于消除水滴或水膜的影响。有关屏蔽电极使用的更多信息，请参考屏蔽电极一节。

- 禁用（默认选择）
- 使能

Shield signal delay（屏蔽信号延迟）

该参数用于指定 CapSense 屏蔽相对于传感器引脚上的信号的延迟 HFCLK 周期数。

- 无（默认）
- 一个周期
- 两个周期

注意：为了能正确地屏蔽，屏蔽信号需要与传感器上的信号处于相同的相位。

Shield tank capacitor enable（屏蔽槽电容使能）

该参数用来指定是否使能将片外 Ctank 电容与屏蔽电容并联的引脚。该电容用于增加屏蔽电容。当屏蔽 Cp 值确实很高时，通过屏蔽槽电容可以降低屏蔽和传感器时钟之间的相差。同样，将 Cmod 预充电或 Csh_tank 预充电设置为“Precharge by IO buffer”（通过 IO 缓冲进行预充电）时，也要使能 Ctank 电容。

- 禁用（默认选择）
- 使能

Guard Sensor（保护传感器）

该参数可使能保护传感器，有助于检测需要防水的应用中的水滴。如果选中（在 **General** 选项卡下的）**Water Proofing and detection**（防水并检测），则使能该功能。有关保护传感器的更多信息，请参见本数据手册的功能描述一节。

- 禁用（默认选择）
- 使能

Cmod precharge（Cmod 预充电）

该参数指定用于 Cmod 电容的预充电源。

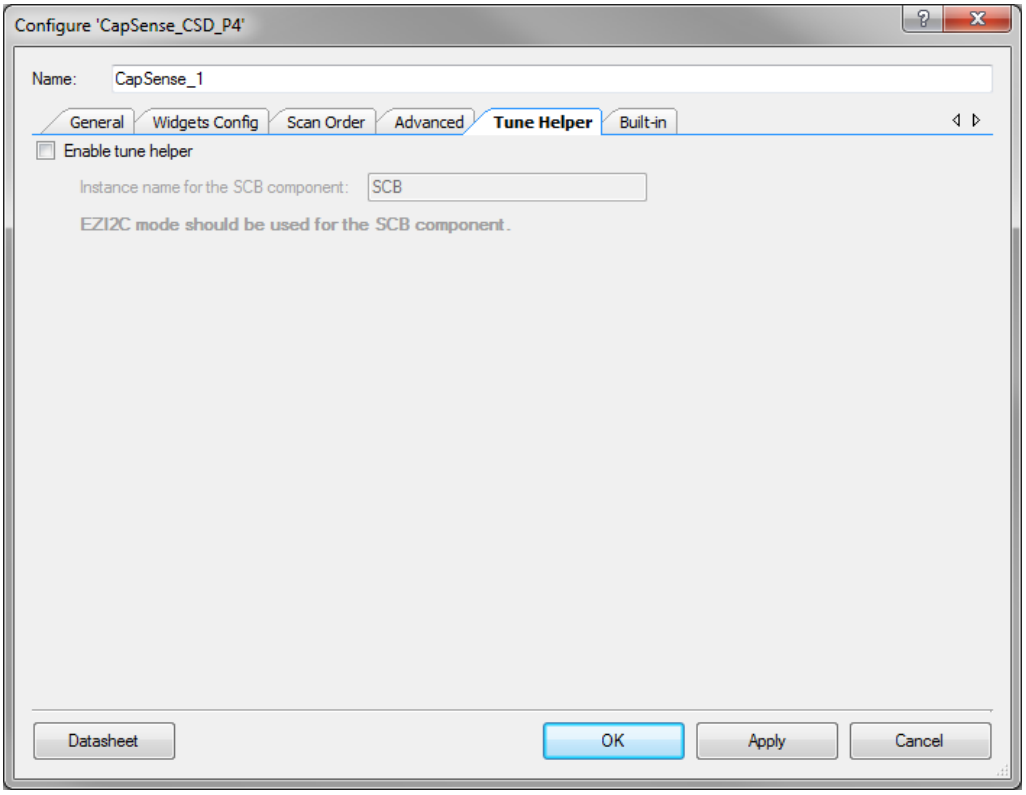
- 通过 Vref 缓冲进行预充电（默认）
- 通过 IO 缓冲进行预充电

Csh_tank precharge（Csh_tank 预充电）

该参数指定用于驱动屏蔽电极的 Vref 源。

- 通过 Vref 缓冲进行预充电（默认）
- 通过 IO 缓冲进行预充电

Tune Helper（调谐助手）选项卡



Enable Tune Helper（使能调谐器助手）

该参数添加了多种功能，使用户可轻松与调试器 GUI 进行通信。如果要使用 Tuner GUI，请勾选此特性。如果未选中此选项，仍然提供通信功能，但不执行任何操作。因此，当调试完成或调试方法改变时，无需删除这些功能。默认禁用此选项。

EzI2C component instance name （EZI2C 组件的实例名称）

该参数定义了您的设计中将用于与调试器 GUI 进行通信的 EZI2C 组件的实例名称。

更多有关调试器 GUI 的使用方法，请参考 *PSoC® 4 CapSense® 调试指南*。

应用编程接口

通过应用编程接口（API），您可以使用软件进行配置组件。下表对各个函数进行了概述。以下各节将更加详细地介绍每个函数。

组件可用于支持以下编译器的集成开发环境（IDE）中：

- ARM GCC 编译器
- ARM MDK 编译器
- ARM RealView 编译器
- IAR C/C++编译器

注意：使用 IAR 嵌入式工作台时，需要指出导航到静态库的路径。该库位于以下 PSoC Creator 安装目录：

PSoC Creator\psoc\content\CyComponentLibrary\CyComponentLibrary.cylib\CortexM0\IAR

默认情况下，PSoC Creator 将实例名称“CapSense_1”分配给所提供设计中组件的第一个组件实例。您可以将其重新命名为符合标识符语法规则的任意唯一值。实例名称会成为每个全局函数名称、变量和符号常量的前缀。出于可读性考虑，下表中使用的实例名称为“CapSense”。

通用 API

以下是将组件置于工作或停止状态的通用 CapSense API 函数：

函数	说明
CapSense_Start()	启动组件的首选方法。初始化寄存器，并使能CapSense中所用子组件的活跃模式电源模板位。在Smartsense调试模式中，API根据所计算的寄生电容调整感应时钟分频、IDAC和分辨率参数。
CapSense_Stop()	禁用组件中断，并调用CapSense_ClearSensors()，以便将所有传感器复位到非活动状态。
CapSense_Sleep()	为进入低功耗模式的器件准备组件。禁用CapSense中所用子组件的Active（活动）模式电源模板位，保存非自保持寄存器并将所有传感器复位到非活动状态。
CapSense_Wakeup()	当器件从低功耗模式的睡眠模式中唤醒后，将恢复CapSense配置和非保留寄存器值。
CapSense_Init()	初始化随定制器提供的CapSense默认配置
CapSense_Enable()	使能CapSense中所用子组件的Active（活动）模式电源模板位。
CapSense_SaveConfig()	保存CapSense配置。
CapSense_RestoreConfig()	恢复CapSense配置。

void CapSense_Start(void)

说明： 这是开始执行组件操作的首选方法。CapSense_Start()调用CapSense_Init()函数，然后调用CapSense_Enable()函数。初始化寄存器，并启动CapSense组件的CSD方法。将所有传感器复位到非活动状态。使能传感器扫描中断。选定SmartSense调试模式时，调试过程应用于所有传感器。在Smartsense调试模式中，API根据所计算的寄生电容调整感应时钟分频、IDAC和分辨率参数。在调用其他任何API前，必须先调用CapSense_Start()子程序。

参数： 无

返回值： 无

其他影响 如果选择了自动（Smartsense）调试或者自动校准方法，全局中断必须在执行CapSense_Start()前被使能。

void CapSense_Stop(void)

说明： 停止传感器扫描、禁用组件中断并将所有传感器复位到非活动状态。禁用CapSense中所用子组件的活动模式电源模板位。

参数： 无

返回值： 无

副作用： 应在完成所有扫描操作后调用该函数。

void CapSense_Sleep(void)

说明： 这是为器件低功耗模式准备组件的首选方法。禁用CapSense中所用子组件的活动模式电源模板位。调用CapSense_SaveConfig()函数以保存CapSense的客户配置，并将所有传感器复位到非活动状态。

参数： 无

返回值： 无

副作用： 应该在完成扫描操作后调用该函数。

该函数不能将CapSense组件所使用的引脚置于最低功耗状态。

void CapSense_Wakeup(void)

说明： 恢复CapSense配置。通过为CapSense中所用的子组件设置活动模式电源模板位，可恢复组件的启用状态。

参数： 无

返回值： 无

副作用： 该函数不会将CapSense组件所使用的引脚还原到之前的状态。

void CapSense_Init(void)

说明： 初始化定义组件操作的定制器所提供的CapSense默认配置。将所有传感器复位到非活动状态。

参数： 无

返回值： 无

其他影响： 无

void CapSense_Enable(void)

说明： 启用CapSense中所用子组件的活动模式电源模板位。

参数： 无

返回值： 无

其他影响： 无

void CapSense_SaveConfig(void)

说明： 保存CapSense配置。将所有传感器复位到非活动状态。

参数： 无

返回值： 无

副作用： 应在完成扫描操作后调用该函数。
该函数不能将CapSense组件所使用的引脚置于最低功耗状态。

void CapSense_RestoreConfig(void)

说明： 恢复CapSense配置。

参数： 无

返回值： 无

副作用： 应在完成扫描操作后调用该函数。
该函数不能将CapSense组件所使用的引脚还原到它们之前的状态。

扫描特定的 API

以下 API 函数用于执行 CapSense 传感器扫描。

函数	说明
CapSense_ScanSensor()	设置扫描设置，然后开始扫描单个传感器或一组传感器。
CapSense_ScanWidget()	设置扫描设置，然后开始扫描widget。
CapSense_ScanEnabledWidgets()	首选扫描方法。扫描所有已启用的Widget。
CapSense_IsBusy()	返回传感器扫描状态。
CapSense_SetScanSlotSettings()	设置所选扫描插槽（传感器）的扫描设置。
CapSense_ClearSensors()	将所有传感器重置到非采样状态。
CapSense_EnableSensor()	配置所选的传感器，以便在下一个扫描周期中对其进行扫描。
CapSense_DisableSensor()	禁用所选传感器，以便在下一个扫描周期不对其进行扫描。
CapSense_ReadSensorRaw()	从CapSense_SensorResult[ ]阵列返回传感器原始数据。
CapSense_ReadCurrentScanningSensor()	执行传感器扫描过程中返回扫描的传感器编号。

void CapSense_ScanSensor(uint32 sensor)

说明： 设置扫描设置，然后开始扫描传感器。扫描完成后，ISR将已测量的传感器原始数据复制到全局原始感应器阵列中。使用ISR可确保该函数无阻塞。每个传感器在传感器阵列中只有唯一一个编号。该编号由CapSense定制器依次分配。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： 无

其他影响： 无

void CapSense_ScanWidget (uint32 widget)

说明： 设置扫描设置，然后开始扫描widget。

参数： uint32 widget：Widget编号。每个Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__"widget type"      "Widget number"
```

示例：

```
#define CapSense_TOUCHPAD0__TP                     5
```

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： 无

其他影响： 无

void CapSense_ScanEnabledWidgets(void)

说明： 这是扫描所有已启用Widget的首选方法。开始对已启用Widget内的传感器进行扫描。ISR对传感器进行持续扫描，直到所有已启用Widget均扫描完毕。使用ISR可确保该函数无阻塞。

除接近感应Widget以外，所有Widget都在默认情况下使能。由于接近Widget的扫描时间较长，不符合其他Widget所需的快速响应，因此必须手动使能接近感应Widget。

参数： 无

返回值： 无

副作用： 如果未启用任何Widget，则此次函数调用将无效。

uint32 CapSense_IsBusy (void)

说明： 返回传感器扫描状态。

参数： 无

返回值： uint32：返回扫描状态。‘1’ — 正在扫描，‘0’ — 扫描完成。

其他影响 无

void CapSense_SetScanSlotSettings(uint32 slot)

说明： 对定制器中提供的扫描设置或对所选扫描插槽（传感器）的向导进行设置。这些扫描设置提供了每个传感器的IDAC值以及分辨率。在一个Widget内，所有传感器的分辨率均相同。

参数： uint32 slot：扫描插槽编号

返回值： 无

其他影响： 无

void CapSense_ClearSensors(void)

说明： 通过依次断开与Analog MUX Bus（模拟复用器总线）连接的所有传感器并将其连接至非活动状态，使所有传感器都复位到非采样状态。

参数： 无

返回值： 无

其他影响： 无

void CapSense_EnableSensor(uint32 sensor)

说明： 配置选定的传感器，以在下一个测量周期中对其进行扫描。将对应引脚设置为Analog HI-Z（模拟高阻模式），并连接至Analog Mux Bus（模拟复用器总线）。这也会影响比较器输出。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： 无

其他影响： 无

void CapSense_DisableSensor(uint32 sensor)

说明： 禁用选定的传感器。断开与模拟复用器总线连接的对应引脚，并将其置于非活动状态。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： 无

其他影响： 无

uint16 CapSense_ReadSensorRaw(uint32 sensor)

说明： 从全局CapSense_SensorResult[ ]阵列返回传感器的原始数据。每个扫描传感器在传感器阵列中有唯一编号。该编号由CapSense定制器依次分配。原始数据（raw data）可用于执行CapSense所提供框架以外的计算。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint16：当前的原始数据值

副作用： 无

uint32 CapSense_ReadCurrentScanningSensor(void)

说明： 该API返回正在扫描的传感器ID。如果未扫描任何传感器，API将返回0xFFFFFFFF。

参数： 无

返回值： uint32：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

其他影响: 无

高级 API

这些 API 函数用于处理传感器 Widget 的原始数据（raw data）。取回已扫描的传感器的原始数据，并转化为按键的开关状态、滑条的位置或触控板的 X 轴和 Y 轴。

函数	说明
CapSense_InitializeSensorBaseline()	通过扫描选定的传感器，加载含初始值的CapSense_SensorBaseline[sensor]阵列元素。
CapSense_InitializeEnabledBaselines()	仅扫描使能的传感器，可加载含初始值的CapSense_sensorBaseline[]阵列。 该函数仅在双通道设计中可用。

函数	说明
CapSense_InitializeAllBaselines()	通过扫描所有的传感器，加载含初始值的CapSense_sensorBaseline[]阵列。
CapSense_UpdateSensorBaseline()	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线。更新后的基线使用“k = 256”的低通滤波器。
CapSense_UpdateEnabledBaselines()	检查CapSense_sensorEnableMask[]阵列并调用CapSense_UpdateSensorBaseline()函数，以更新已使能传感器的基线。
CapSense_EnableWidget()	为扫描过程使能Widget中的所有传感器元件。
CapSense_DisableWidget()	禁用Widget中所有正处于扫描过程的传感器元件。
CapSense_CheckIsWidgetActive()	将选定的Widget与CapSense_Signal[]阵列相比较，以确定其是否存在手指触摸。
CapSense_CheckIsAnyWidgetActive()	使用CapSense_CheckIsWidgetActive()函数检查CapSense CSD组件中是否存在处于活动状态的Widget。
CapSense_GetCentroidPos()	检查CapSense_sensorSignal[]阵列，以确定线性滑条上是否存在手指触摸，并返回位置。
CapSense_GetRadialCentroidPos()	检查CapSense_sensorSignal[]阵列，以确定辐射滑条上是否存在手指触摸，并返回位置。
CapSense_GetTouchCentroidPos()	若有手指存在，则该函数可通过计算触控板内的质心算出该手指在X轴和Y轴上的位置。
CapSense_GetMatrixButtonPos()	如有手指触摸，该函数计算矩阵按键上手指的行位置和列位置。
CapSense_CheckIsSensorActive()	如果传感器处于活动状态，则返回“true”（真）。
CapSense_GetBaselineData()	读取传感器基线。
CapSense_GetDiffCountData()	返回差值计数数据。
CapSense_GetNormalizedDiffCountData()	返回标准化的差值计数数据。
CapSense_GetNoiseThreshold()	返回噪声阈值。
CapSense_GetNegativeNoiseThreshold()	返回负噪声阈值。
CapSense_GetNoiseEnvelope()	返回测量的噪声包络线值。
CapSense_GetFingerThreshold()	返回手指阈值。
CapSense_GetFingerHysteresis()	返回迟滞值。
CapSense_WriteSensorRaw()	写入原始计数值。
CapSense_SetBaselineData()	写入基线值。
CapSense_SetSensitivity()	设置灵敏度值。

函数	说明
CapSense_GetSensitivityCoefficient()	返回K系数。
Capsense_SetDebounce()	设置去抖动值。
Capsense_GetDebounce()	返回去抖动值。
CapSense_SetFingerHysteresis()	设置传感器的迟滞值。
CapSense_SetNoiseThreshold()	设置噪声阈值。
CapSense_SetNegativeNoiseThreshold()	设置负噪声阈值。
CapSense_SetLowBaselineReset()	设置低基线复位阈值。
CapSense_GetLowBaselineReset()	返回低基线复位阈值。
CapSense_SetFingerThreshold()	返回手指阈值。
CapSense_SetDiffCountData()	设置差值计数数据。
CapSense_GetWidgetNumber()	返回传感器的Widget编号。
CapSense_UpdateThresholds()	更新阈值。
CapSense_UpdateBaselineNoThreshold()	更新传感器基线而不更新阈值。
CapSense_SetIDACRange()	设置IDAC范围。
CapSense_GetIDACRange()	返回IDAC范围。
CapSense_SetModulationIDAC()	设置调制IDAC的值。
CapSense_GetModulationIDAC()	返回调制IDAC的值。
CapSense_SetCompensationIDAC()	设置补偿IDAC的值。
CapSense_GetCompensationIDAC()	返回补偿IDAC的值。
CapSense_SetSenseClkDivider()	设置感应时钟分频器的值。
CapSense_GetSenseClkDivider()	返回感应时钟分频器的值。
CapSense_SetModulatorClkDivider()	设置调制采样时钟分频器的值。
CapSense_GetModulatorClkDivider()	返回调制采样时钟分频器的值。
CapSense_SetScanResolution()	设置传感器扫描分辨率的值。
CapSense_GetScanResolution()	返回传感器扫描分辨率的值。
CapSense_SetDriveModeAllPins()	设置端口引脚的驱动模式。
CapSense_RestoreDriveModeAllPins()	将所有CapSense端口引脚的驱动模式恢复到原始状态。
CapSense_SetUnscannedSensorState()	设置未扫描传感器的状态。
CapSense_UpdateWidgetBaseline()	为widget的使能传感器更新基线。

函数	说明
CapSense_EnableRawDataFilters()	为传感器信号使能原始数据滤波器。
CapSense_DisableRawDataFilters()	为传感器信号禁用原始数据滤波器。

void CapSense_InitializeSensorBaseline(uint32 sensor)

说明： 通过扫描选定的传感器，加载含初始值的CapSense_SensorBaseline[sensor]阵列元素。将原始计数值复制到每个传感器的基线阵列中。初始化原始数据滤波器（如果已使能）。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： 无

其他影响： 无

void CapSense_InitializeEnabledBaselines(void)

说明： 扫描所有已使能的Widget。对于扫描过程中使能的所有传感器，原始计数值被复制到CapSense_sensorBaseline[ ]阵列。对扫描过程禁用的传感器，将CapSense_sensorBaseline[ ]初始化为零。初始化原始数据滤波器（如果已使能）。

参数： 无

返回值： 无

其他影响： 无

void CapSense_InitializeAllBaselines(void)

说明： 使用CapSense_InitializeSensorBaseline() 函数扫描所有的传感器，以加载含初始值的CapSense_SensorBaseline[ ]阵列。将原始计数值复制到所有传感器的基线阵列中。初始化原始数据滤波器（如果已使能）。

参数： 无

返回值： 无

其他影响： 无

void CapSense_UpdateSensorBaseline(uint32 sensor)

说明： 传感器的基线是针对每个传感器独立计算得出的历史计数值。使用k = 256的低通滤波器更新CapSense_SensorBaseline[sensor]阵列元素。该函数通过将当前原始计数值减去之前的基线值计算得出差值计数，并将该结果存储在CapSense_sensorSignal[sensor]中。

如果启用自动复位选项，则基线将被更新，而不受噪声阈值的影响。

如果自动复位选项被禁用，则在信号值大于噪声阈值的情况下，基线将会停止更新，而在信号值小于负的噪声阈值的情况下，该基线将会复位。

若原始数据滤波器在计算基线之前启用，则可应用于这些数值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： 无

其他影响： 无

void CapSense_UpdateEnabledBaselines(void)

说明： 检查CapSense_sensorEnableMask[]阵列并调用CapSense_UpdateSensorBaseline()函数，以更新所有已使能传感器的基线。

参数： 无

返回值： 无

其他影响： 无

void CapSense_EnableWidget(uint32 widget)

说明： 使能选定的“Widget”传感器，以作为扫描过程中的一部分。

参数： uint32 widget：Widget编号。每个Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__"widget type"          5
```

示例：

```
#define CapSense_MY_VOLUME1__LS                        5
#define CapSense_MY_UP__BNT                            6
```

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： 无

其他影响： 无

void CapSense_DisableWidget(uint32 widget)

说明： 禁用所有扫描过程中选定的Widget传感器。

参数： uint32 widget：Widget编号。每个Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__"widget type"          5
```

示例：

```
#define CapSense_MY_VOLUME1__RS                        5
#define CapSense_MY_UP__MB                             6
```

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： 无

其他影响： 无

uint32 CapSense_CheckIsWidgetActive(uint32 widget)

说明： 将选定的传感器CapSense_Signal[ ]阵列值与其手指阈值进行比较。考虑了迟滞和去抖动。若传感器处于活动状态，则迟滞量会降低阈值。若传感器处于非活动状态，则迟滞量会提高阈值。若满足活动阈值，则去抖动计数器会增加一，直至传感器达到有效切换，此时该API将“Widget”设置为活动状态。该函数还会更新传感器CapSense_SensorOnMask[ ]阵列中的位。

触控板和矩阵按键Widget需要在行和列中都拥有处于活动状态的传感器，以返回Widget活动状态。

参数： uint32 widget：Widget编号。每个Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__"widget type"          5
```

示例：

```
#define CapSense_MY_VOLUME1__LS                        5
```

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint32：Widget传感器状态。如果Widget内有一个或多个传感器处于活动状态，则为1，如果Widget内所有的传感器均处于非活动状态，则为0。

其他影响 该函数还更新从属于Widget的所有传感器的CapSense_sensorOnMask[ ]值。如果转换至活动状态，则每次调用时还会修改去抖动计数器。

uint32 CapSense_CheckIsAnyWidgetActive(void)

说明： 将CapSense_Signal[ ]阵列中的所有传感器与其手指阈值进行比较。针对每个Widget调用Capsense_CheckIsWidgetActive()，以便在调用该函数后，CapSense_sensorOnMask[ ]阵列为最新。

参数： 无

返回值： uint32：如有Widget处于活动状态则计为1，如果没有Widget处于活动状态则计为0。

其他影响 和CapSense_CheckIsWidgetActive()函数具有同样的其他影响，但并非针对所有传感器。

uint16 CapSense_GetCentroidPos(uint32 widget)

说明： 检查CapSense_Signal[ ]阵列在线性滑条内是否存在手指触摸。将手指位置计算为CapSense定制器中指定的API分辨率。位置过滤器（如果已使能）将应用于该结果中。仅在CapSense定制器定义了线性滑条Widget时，才能使用该函数。

参数： uint32 widget：Widget编号。每个线性滑条Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__LS            5
```

示例：

```
#define CapSense_MY_VOLUME1__LS               5
```

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint16：线性滑条的位置数值

副作用： 如果滑条Widget内有任何传感器处于活动状态，则函数将返回从零至CapSense定制器中设置的API分辨率的值。如果没有任何传感器处于活动状态，则该函数返回 0xFFFF。如果在执行质心/双工算法时出现错误，则该函数返回0xFFFF。

没有为该函数提供Widget参数检查。不正确的Widget值可导致意外的位置计算结果。

注意：如果滑条段的噪声计数大于噪声阈值，则此子例程可能生成假的手指触摸结果。设置噪声阈值时应小心（显著大于噪声级别），以便噪声不会产生假的手指触摸。

uint16 CapSense_GetRadialCentroidPos(uint32 widget)

说明： 检查CapSense_Signal[ ]阵列在辐射滑条内是否存在手指触摸。将手指位置计算为CapSense定制器中指定的API分辨率。位置过滤器（如果已使能）将应用于该结果中。仅在CapSense定制器定义了辐射滑条Widget时，才能使用该函数。

参数： uint32 widget：Widget编号。每个辐射滑条Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__RS          5
```

示例：

```
#define CapSense_MY_VOLUME2__RS             5
```

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint16：辐射滑条的位置数值。

副作用： 如果滑条Widget内有任何传感器处于活动状态，则函数将返回从零至CapSense定制器中设置的API分辨率的值。如果没有任何传感器处于活动状态，则该函数返回0xFFFF。

没有为该函数提供Widget类型参数检查。不正确的Widget值可导致意外的位置计算结果。

注意：如果滑条段的噪声计数大于噪声阈值，则此子例程可能生成假的手指触摸结果。设置噪声阈值时应小心（显著大于噪声级别），以便噪声不会产生假的手指触摸。

uint32 CapSense_GetTouchCentroidPos(uint32 widget, uint16* pos)

说明： 如果触控板上有手指触摸，则该函数可通过计算触控板传感器内的质心来计算手指在X轴和Y轴上的位置。手指在 X 轴和 Y 轴上的位置根据 CapSense 定制器中设置的 API 分辨率进行计算。如果触控板上存在手指，则返回“1”。位置过滤器（如果已使能）将应用于该结果中。仅在CapSense定制器定义了触控板时，才能使用该函数。

参数： uint8 widget：Widget编号。每个触控板Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__TP          5
```

示例：

```
#define CapSense_MY_TOUCH1__TP              5
```

所有Widget名称均为大写。

(uint16* pos)：指向两个“uint16”阵列的指针，触摸位置将存于该阵列：

pos[0] — X轴位置；

pos[1] — Y轴位置。

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint32：如果触控板上存在手指触摸，返回值为1。不存在手指触摸时，则返回值为0。

其他影响： 无

uint32 CapSense_GetMatrixButtonPos(uint32 widget, uint8* pos)

说明： 如果矩阵按键上有一个手指，该函数计算手指的行位置和列位置，并且会返回一个‘1’值。只有当CapSense定制器定义了矩阵按键时，该函数才可用。

参数： uint8 widget：Widget编号。每个矩阵按键Widget都有以下格式的定义：

```
#define CapSense_"widget_name"__MB            5
```

示例：

```
#define CapSense_MY_TOUCH1__MB                5
```

所有Widget名称均为大写。

(uint8* pos)：指向两个“uint8”阵列的指针，触摸位置将存于该阵列：

pos[0] — 列位置；

pos[1] — 行位置。

所有Widget名称均为大写。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint8：如果触控板上存在手指，则为1，不存在手指，则为0。

其他影响： 无

uint32 CapSense_CheckIsSensorActive(uint32 sensor)

说明： 将选定的传感器CapSense_sensorSignal[ ]阵列值与其手指阈值进行比较。需要考虑到迟滞和去抖动。根据传感器当前是否开启，对手指阈值增加或消减迟滞值。若传感器处于活动状态，则迟滞量会降低阈值。若传感器处于非活动状态，则迟滞量会提高阈值。为传感器活动的瞬变增加了去抖动计数器。该函数还更新传感器CapSense_sensorOnMask[ ]阵列中的位。

参数： uint32 – sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32：如果扫描传感器处于活动状态，返回值为1；如果扫描传感器处于非活动状态，则返回值为0。

其他影响 该函数还更新传感器CapSense_sensorOnMask[ ]阵列中的位。

uint16 CapSense_GetBaselineData(uint32 sensor)

说明： 这是读取组件内的传感器基线的函数。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint16：该API返回参数表示的传感器基线值。

其他影响： 无

uint16 CapSense_GetDiffCountData(uint32 sensor)

说明： 该API返回差值计数数据。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint16：该API返回参数表示的差值计数值。

其他影响： 无

uint16 CapSense_GetNormalizedDiffCountData(uint32 sensor)

说明： 该API返回标准化的差值计数数据。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint16：该API返回参数表示的标准化差值计数值。

其他影响： 无

uint8 CapSense_GetNoiseThreshold(uint32 widget)

说明： 该API返回噪声阈值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint8：该API返回参数表示的widget的噪声阈值。

其他影响： 无

uint8 CapSense_GetNegativeNoiseThreshold(uint32 widget)

说明： 该API返回负噪声阈值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint8：该API返回参数表示的Widget的负噪声阈值。

其他影响： 无

uint16 CapSense_GetNoiseEnvelope(uint32 sensor)

说明： 该API返回测量的噪声包络线值。该API的最小值为1，并且它从不会返回0（如噪声）。仅在SmartSense（自动调试）被使能时，该API才可用。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint16：该API返回参数表示的传感器的噪声包络线值。

其他影响： 无

uint8/uint16 CapSense_GetFingerThreshold(uint32 widget)

说明： 该API返回手指阈值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint8/uint16：该API返回参数表示的Widget手指阈值。

其他影响： 无

uint8 CapSense_GetFingerHysteresis(uint32 widget)

说明： 该API返回迟滞值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint8：该API返回参数表示的widget迟滞。

其他影响： 无

void CapSense_WriteSensorRaw(uint32 sensor, uint16 data)

说明： 该API包括两个参数：传感器编号和原始计数值。该API将参数通过的原始计数值写入到传感器原始计数陈列中。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint16 data：传感器原始计数。

返回值： 无

其他影响： 无

void CapSense_SetBaselineData(uint32 sensor, uint16 data)

说明： 该API包括两个参数：传感器编号和基线值。

该API将参数通过的数据值写入到传感器基线阵列中。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint16 data：传感器基线。

返回值： 无

其他影响： 无

void CapSense_SetSensitivity(uint32 sensor, uint32 data)

说明： 该API设置传感器的灵敏度值。灵敏度值使用于在CapSense_Start API中执行的自动调试算法。

调用CapSense_Start API前，该API由应用层调用。执行CapSense_Start API后调用该API将不产生任何影响。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint32 data：传感器的灵敏度。可能的值为：

1 – 灵敏度为0.1 pF

2 – 灵敏度为0.2 pF

3 – 灵敏度为0.3 pF

4 – 灵敏度为0.4 pF

对于所有其他的值，将灵敏度值设置为0.4 pF。

返回值： 无

其他影响： 无

uint32 CapSense_GetSensitivityCoefficient(uint32 sensor)

说明： 该API为相应的传感器返回K系数。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32：相应传感器的K值。

其他影响 无

void Capsense_SetDebounce(uint32 widget, uint8 value)

说明： 该API设置去抖动值。该API对widget内的所有传感器产生影响。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

uint8 value：去抖动值

返回值： 无

其他影响： 无

uint8 Capsense_GetDebounce(uint32 widget)

说明： 该API返回去抖动值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint8:返回去抖动值。

其他影响 无

void CapSense_SetFingerHysteresis(uint32 widget, uint8 value)

说明： 该API设置迟滞值传感器。该API对widget内的所有传感器产生影响。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

uint8 value：迟滞值。

返回值： 无

其他影响： 无

void CapSense_SetNoiseThreshold(uint32 widget, uint8 value)

说明： 该API设置widget内所有传感器的噪声阈值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

uint8 value：噪声阈值。

返回值： 无

其他影响： 无

void CapSense_SetNegativeNoiseThreshold(uint32 widget, uint8 value)

说明： 该API设置widget的负噪声阈值。该API对widget内的所有传感器产生影响。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

uint8 value：负噪声阈值。

返回值： 无

其他影响： 无

void CapSense_SetLowBaselineReset(uint32 sensor, uint8 value)

说明： 该API设置传感器的低基线复位阈值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

uint8 value：低基线复位阈值。

返回值： 无

其他影响： 无

uint8 CapSense_GetLowBaselineReset(uint32 sensor)

说明： 该API返回传感器的低基线复位阈值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint8：返回低基线复位阈值。

其他影响 无

void CapSense_SetFingerThreshold(uint32 widget, uint8/16 value)

说明： 该API设置Widget的手指阈值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

uint8/16 value：Widget的手指阈值。

返回值： 无

其他影响： 无

void CapSense_SetDiffCountData(uint32 sensor, uint16/uint8 value)

说明： 该API设置每个传感器的差值计数数据。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint16/uint8 value：差值计数数据。

返回值： 无

其他影响： 无

uint32 CapSense_GetWidgetNumber(uint32 sensor)

说明： 该API返回传感器的widget编号。

参数： uint32 sensor：传感器编号。传感器编号的范围是从0到N。N值可以是0到传感器总数减1。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32：返回传感器的widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

其他影响： 无

void CapSense_UpdateThresholds(uint32 sensor)

说明： 当SmartSense被使能时，该API将计算给定传感器的阈值参数，并对存储每个传感器的阈值参数的相应阵列/变量进行参数更新。有两个可能的方法能够计算上面所述的阈值。

当自动阈值被使能，该API将根据传感器的测量噪声包络线计算阈值参数。在此模式下，API将计算给定传感器的手指阈值和其他阈值。

当自动阈值被禁用时，该API将不计算手指阈值。此时，手指阈值应由应用固件设置。该API可以根据调用者设置的手指阈值计算所有其他阈值。在此模式下，调用该API前，API会预期调用方设置相应的手指阈值。

该API适用于所有传感器类型。

参数： uint32 sensor：传感器编号。传感器编号的范围是从0到N。N值可以是0到传感器总数减1。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： 无

其他影响： 无

void CapSense_UpdateBaselineNoThreshold(uint32 sensor)

说明： 该API更新了给定的传感器的基线。该API不计算或修改给定传感器相关的阈值参数。

参数： uint32 sensor：传感器编号。传感器编号的范围是从0到N。N值可以是0到传感器总数减1。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： 无

其他影响： 传感器基线变量将被更新。

void CapSense_SetIDACRange(uint32 iDacRange)

说明： 将IDAC范围设置为4x（1.2 uA/位）或8x（2.4uA/位）模式。IDAC范围对于所有传感器和调制以及补偿IDAC很常用。

参数： uint32 iDacRange：表示IDAC范围的值。

0 — IDAC范围被设置为4x（1.2 uA/位）

1或>1 — IDAC范围被设置为8x（2.4 uA/位）

返回值： 无

其他影响： 无

uint32 CapSense_GetIDACRange(void)

说明： 返回表示IDAC范围的值。组件采用该范围来扫描传感器。IDAC对于所有传感器通用。

参数： 无

返回值： uint32 iDacRange：表示IDAC范围的值。

0 — IDAC范围被设置为4x（1.2 uA/位）

1或>1 — IDAC范围被设置为8x（2.4 uA/位）

其他影响 无

void CapSense_SetModulationIDAC(uint32 sensor, uint32 modIdacValue)

说明： 设置传感器的调制IDAC值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint32 modIdacValue：表示调制IDAC数据寄存器值。

返回值： 无

其他影响： 无

uint32 CapSense_GetModulationIDAC(uint32 sensor)

说明： 返回传感器的调制IDAC值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32返回调制IDAC数据寄存器值。

其他影响： 无

void CapSense_SetCompensationIDAC(uint32 sensor, uint32 compIdacValue)

说明： 设置传感器的调制IDAC值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint32 compIdacValue：表示补偿IDAC数据寄存器值。

返回值： 无

其他影响： 无

uint32 CapSense_GetCompensationIDAC(uint32 sensor)

说明： 返回传感器的补偿IDAC值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32：返回补偿IDAC数据寄存器值。

其他影响： 无

void CapSense_SetSenseClkDivider(uint32 sensor, uint32 senseClk)

说明： 设置传感器的感应时钟分频器值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint32 senseClk：表示感应时钟值。

返回值： 无

其他影响： 无

uint32 CapSense_GetSenseClkDivider(uint32 sensor)

说明： 返回传感器的感应时钟分频器值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32：返回传感器的感应时钟分频器值。

其他影响： 无

void CapSense_SetModulatorClkDivider(uint32 sensor, uint32 modulatorClk)

说明： 设置传感器的调制采样时钟分频器值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint32 – modulatorClk：表示调制采样时钟的值。

返回值： 无

其他影响： 无

uint32 CapSense_GetModulatorClkDivider(uint32 sensor)

说明： 返回传感器的调制采样时钟分频器的值。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32：返回传感器的调制采样时钟分频器的值。

其他影响 无

void CapSense_SetScanResolution(uint32 widget, uint32 resolution)

说明： 设置widget的传感器扫描分辨率的值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

uint32 resolution：表示分辨率值。*CapSense.h*文件中的下面定义将被使用：

- CapSense_RESOLUTION_6_BITS
- CapSense_RESOLUTION_7_BITS
- CapSense_RESOLUTION_8_BITS
- CapSense_RESOLUTION_9_BITS
- CapSense_RESOLUTION_10_BITS
- CapSense_RESOLUTION_11_BITS
- CapSense_RESOLUTION_12_BITS
- CapSense_RESOLUTION_13_BITS
- CapSense_RESOLUTION_14_BITS
- CapSense_RESOLUTION_15_BITS
- CapSense_RESOLUTION_16_BITS

返回值： 无

其他影响： 无

uint32 CapSense_GetScanResolution(uint32 widget)

说明： 返回widget的分辨率值。

参数： uint32 widget：Widget编号。*Capsense_CSHL.h*文件包含widget编号的定义。更多有关信息，请参考Widget常量一节。

返回值： uint32：返回widget的分辨率。返回值对应于*CapSense.h*文件中的下面定义：

- CapSense_RESOLUTION_6_BITS
- CapSense_RESOLUTION_7_BITS
- CapSense_RESOLUTION_8_BITS
- CapSense_RESOLUTION_9_BITS
- CapSense_RESOLUTION_10_BITS
- CapSense_RESOLUTION_11_BITS
- CapSense_RESOLUTION_12_BITS
- CapSense_RESOLUTION_13_BITS
- CapSense_RESOLUTION_14_BITS
- CapSense_RESOLUTION_15_BITS
- CapSense_RESOLUTION_16_BITS

其他影响： 无

void CapSense_SetDriveModeAllPins(uint32 driveMode)

说明： 该API将CapSense组件（传感器、保护、屏蔽、屏蔽槽和Cmod）使用的端口引脚的驱动模式设置为参数表示的驱动模式。

参数： uint32 driveMode：是表示驱动模式的参数。

值：

CY_SYS_PINS_DM_ALG_HIZ — 高阻抗模拟

CY_SYS_PINS_DM_DIG_HIZ — 高阻抗数字

CY_SYS_PINS_DM_RES_UP — 电阻上拉

CY_SYS_PINS_DM_RES_DWN — 电阻下拉

CY_SYS_PINS_DM_OD_LO — 开漏，驱动低电平

CY_SYS_PINS_DM_OD_HI — 开漏，驱动高电平

CY_SYS_PINS_DM_STRONG — 强驱动

CY_SYS_PINS_DM_RES_UPDWN — 电阻上/下拉

返回值： 无

其他影响 仅在停止CapSense组件后，才应调用该API。

void CapSense_RestoreDriveModeAllPins(void)

说明： 该API将所有CapSense端口引脚的驱动模式恢复到原始状态。该API是CapSense_SetDriveModeAllPins API的补充部分。

参数： 无

返回值： 无

其他影响： 无

void CapSense_SetUnscannedSensorState(uint32 sensor, uint32 sensorState)

说明： 该API设置未扫描传感器的状态。可以将状态设置为接地、高阻或屏蔽电极状态。如果屏蔽被使能，可以将未扫描的传感器连接至屏蔽电极。如果屏蔽被禁用，以及该API由表示屏蔽状态的参数调用，那么，未扫描的传感器将被连接至地。

参数： uint32 sensor：该参数表示传感器ID。Capsense.h文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

uint32 sensorState：该参数表示未扫描传感器的状态。

返回值： 无

其他影响 仅在停止CapSense组件后，才应调用该API。

void CapSense_UpdateWidgetBaseline (uint32 widget)

说明： 传感器的基线是针对Widget中每个传感器独立计算得出的历史计数值。它使用k = 256的低通滤波器更新CapSense_SensorBaseline[sensor]阵列元素。对于属于Widget的传感器编号，该函数通过将当前原始计数值减去之前的基线值计算得出差值计数，并将该结果存储在CapSense_sensorSignal[sensor]中。

如果启用自动复位选项，则基线将被更新，而不受噪声阈值的影响。

如果自动复位选项被禁用，则在信号值大于噪声阈值的情况下，基线将会停止更新，而在信号值小于负的噪声阈值的情况下，该基线将会复位。

参数： uint32 widget：widget编号

返回值： 无

其他影响： 更新了CapSense_sensorBaseline[ ]阵列。

void CapSense_EnableRawDataFilters(void)

说明： 该API为传感器信号使能原始数据滤波器。

参数： 无

返回值： 无

其他影响： 无

void CapSense_DisableRawDataFilters(void)

说明： 该API为传感器信号禁用原始数据滤波器。

参数： 无

返回值： 无

其他影响： 无

调试器助手 API

这些 API 函数与调试器 GUI 一起使用 。

函数	说明
CapSense_TunerStart()	初始化CapSense CSD和内部通信组件，初始化基线并启动传感器扫描循环。
CapSense_TunerComm()	执行调试器GUI之间的通信。

void CapSense_TunerStart(void)

说明： 初始化CapSense CSD和内部通信组件。

除接近感应Widget以外，所有Widget都在默认情况下使能。由于接近Widget的扫描时间较长，不符合其他Widget所需的快速响应，因此必须手动使能接近感应Widget。

参数： 无

返回值： 无

其他影响： 如果选择了自动（Smartsense）调试或者自动校准方法，全局中断（CyGlobalIntEnable;）要在执行CapSense_TunerStart()前被使能。

void CapSense_TunerComm(void)

说明： 执行与调试器GUI之间的通信功能。

- 手动模式：将传感器扫描和Widget处理结果从CapSense CSD组件传输到调试器GUI。从调试器GUI读取新参数并将其应用于CapSense CSD组件。
- 自动（SmartSense）：执行与调试器GUI之间的通信功能。将传感器扫描和Widget处理结果传输到调试器GUI。自动调试参数也传输到调试器GUI。调试器GUI参数不会被传输回CapSense CSD组件。

该函数正在执行阻塞操作，并等待调试器GUI修改CapSense CSD组件缓冲区以允许新数据。

参数： 无

返回值： 无

其他影响: 直到成功连接到调试器GUI为止，该API不允许执行代码并返回值。

内置自测试 API

这些 API 函数用于检查准确的硬件设置（如 Cmod、寄生电容、屏蔽电极和外部屏蔽槽电容）。

函数	说明
CapSense_GetSensorCp()	返回传感器的寄生电容值。
CapSense_MeasureCmod()	测量CMOD外部电容的值（单位为pF）。
CapSense_MeasureCShield()	测量屏蔽电极的电容值。
CapSense_MeasureCShieldTank()	测量外部屏蔽槽电容器的电容值。

uint32 CapSense_GetSensorCp(uint32 sensor)

说明： 该API返回传感器的Cp（寄生电容）（单位为pF）。

参数： uint32 sensor：传感器编号。*Capsense.h*文件包含传感器编号的定义。更多有关信息，请参考传感器常量一节。

返回值： uint32：该API返回参数表示的传感器的寄生电容（Cp）值。传感器Cp值的单位为pF。

其他影响 无

uint32 CapSense_MeasureCmod(void)

说明： 该API测量CMOD外部电容的值（单位为pF）。

参数： 无

返回值： uint32：返回测量的CMOD（单位为pF）。

其他影响: 在调用该API前应该停止组件。

uint32 CapSense_MeasureCShield(void)

说明： 该API实行测量屏蔽电极的电容值方法。调用该API时，它将返回屏蔽电极的电容值（单位为pF）。

参数： 无

返回值： uint32：返回屏蔽电极的测量电容，单位为pF。

其他影响: 无。

uint32 CapSense_MeasureCShieldTank(void)

说明： 该API实现测量外部屏蔽槽电容值的方法。调用该API时，它将返回屏蔽槽电容值（单位为pF）。

参数： 无

返回值： uint32：返回屏蔽槽电容器的测量电容（单位为pF）。

其他影响: 在调用该API前应该停止组件。

数据结构

API 函数使用几种全局阵列来处理传感器和 Widget 数据。不得手动更改这些阵列。可以出于调试和调试目的对这些值进行查看。例如，可以使用绘图工具显示阵列的内容。全局阵列为：

阵列	说明
CapSense_sensorRaw[]	此阵列包含每个传感器的原始数据。阵列大小等于传感器总数（CapSense_TOTAL_SENSOR_COUNT）。CapSense_sensorRaw [ ]数据通过以下函数更新： • CapSense_ScanSensor() • CapSense_ScanEnabledWidgets() • CapSense_InitializeSensorBaseline() • CapSense_InitializeAllBaselines() • CapSense_UpdateEnabledBaselines()
CapSense_sensorEnableMask[]	这是一个保持传感器扫描状态的字节阵列CapSense_sensorEnableMask[0]，包含传感器0到7的掩码位（传感器0为0位，传感器1为1位）。CapSense_sensorEnableMask[1]包含传感器8到15的掩码位（如果需要），依次类推。此字节阵列存储的元件数足以包含所有的传感器。位值指定是否通过CapSense_ScanEnabledWidgets()函数调用对传感器进行扫描：1 — 传感器被扫描，0 — 传感器未被扫描。CapSense_sensorEnableMask[ ]数据通过以下函数进行更改： • CapSense_EnabledWidget() • CapSense_DisableWidget() • CapSense_sensorEnableMask[ ]数据为以下函数使用： • CapSense_ScanEnabledWidgets()
CapSense_portTable[] 和 CapSense_maskTable[]	这些阵列包含每个传感器的端口和引脚掩码，以指定传感器连接的引脚。 • 端口 — 定义引脚所属的端口号。 • 掩码 — 定义端口内的引脚号。
CapSense_sensorBaselineLow[]	此阵列存储低通滤波器中所使用每个传感器的基线数据的小数字节，用于基线更新。阵列大小等于传感器总数。CapSense_sensorBaselineLow[ ]阵列通过以下函数更新： • CapSense_InitializeSensorBaseline() • CapSense_InitializeAllBaselines() • CapSense_UpdateSensorBaseline() • CapSense_UpdateEnabledBaselines()

阵列	说明
CapSense_sensorBaseline[]	此阵列存储每个传感器的基线数据。阵列大小等于传感器总数。CapSense_sensorBaseline[ ]阵列通过以下函数更新： • CapSense_InitializeSensorBaseline() • CapSense_InitializeAllBaselines() • CapSense_UpdateSensorBaseline() • CapSense_UpdateEnabledBaselines()。
CapSense_sensorSignal[]	此阵列存储通过从每个传感器的当前原始数据中减去以前的基线计算得到的传感器信号数据。阵列大小等于传感器总数。Widget分辨率参数将此阵列的分辨率定义为1字节或2字节。CapSense_sensorSignal[ ]阵列通过以下函数更新： • CapSense_InitializeSensorBaseline() • CapSense_InitializeAllBaselines() • CapSense_UpdateSensorBaseline() • CapSense_UpdateEnabledBaselines()。
CapSense_sensorOnMask[]	这是uint8阵列，可保持（按键、矩阵按键或滑条）传感器为‘on’（活动）或‘off’（非活动）状态。CapSense_sensorOnMask[0]包含传感器0到7的掩码位（传感器0为0位，传感器1为1位）。CapSense_sensorOnMask[1]包含传感器8到15的掩码位（如果需要），以此类推。uint8阵列包含的元素足以包含所有放置的传感器。如果传感器开启，则位（bit）的值为1；如果传感器关闭，则位（bit）的值为0。
CapSense_ModulatorIDAC[]	此阵列包含了每个传感器的8位IDAC值。阵列大小等于传感器总数。
CapSense_CompensationIDAC[]	此阵列包含了每个传感器的7位IDAC值。阵列大小等于传感器总数。
CapSense_senseClkDividerVal[]	此阵列包含每个传感器的感应时钟分频器的值。仅在定制器中使能单一频率设置时，才会生成此阵列。
CapSense_sampleClkDividerVal[]	此阵列包含每个传感器的调制时钟分频器的值。仅在定制器中使能单一频率设置时，才会生成此阵列。
CapSense_rawFilterData1[]	此阵列用于存储任一使能原始数据滤波器的之前样本。 CapSense_rawFilterData1[ ]阵列通过下列函数更新： • CapSense_UpdateSensorBaseline()
CapSense_rawFilterData2[]	此阵列用于存储使能的原始数据滤波器的之前样本。它仅用于中值或均值滤波器 （这些滤波器也使用CapSense_rawFilterData1阵列存储之前的样本）。 CapSense_rawFilterData2[ ]阵列通过下列函数更新： • CapSense_UpdateSensorBaseline()

阵列	说明
CapSense_lowBaselineResetCnt[]	将该函数的元素作为计数器使用，可以决定应不应该对每个扫描传感器进行基线复位。如果差值信号是负向，并高于CapSense_NEGATIVE_NOISE_THRESHOLD，那么计数器将递增。当计数器达到CapSense_LOW_BASELINE_RESET值，则对传感器的基线进行重新初始化，同时，也将计数器设为零。The CapSense_lowBaselineResetCnt[ ]数据通过以下函数进行更新： • CapSense_UpdateSensorBaseline()
CapSense_fingerThreshold[]	此阵列包含确定是否有触摸的每个传感器上的信号电平。
CapSense_noiseThreshold[]	此阵列包含每个传感器的确定电容扫描中的噪声等级的信号等级。低于阈值的噪声用于更新传感器的基线。相反，高于阈值的噪声不用于更新基线。
CapSense_hysteresis[]	此阵列包含每个Widget的迟滞值。 CapSense_debounceCounter[]阵列通过下列函数更新： • CapSense_CalculateThresholds()
CapSense_debounce[]	此阵列包含每个Widget中去抖动特性的去抖动值。为包含此参数的各Widget设置该去抖动值。这些Widget包括：按键、矩阵按键、接近感应传感器和保护传感器。所有其他widget没有去抖动参数，并使用此阵列中数值为0的最后元素（0表示“无去抖动”）。CapSense_debounce[]阵列用于初始化CapSense_debounceCounter[]阵列。
CapSense_debounceCounter[]	此阵列包含传感器的当前去抖动计数器。如果传感器处于活动状态（传感器信号值超过了手指阈值与迟滞值之和），计数器将会递减。当它达到1时，将设置传感器ON掩码（CapSense_sensorOnMask），同时，计数器的值复位为CapSense_debounce[]阵列的默认值。如果传感器处于非活动状态（触摸释放），并传感器信号的值低于手指阈值与迟滞值之差，计数器的行为也会相同。通过CapSense_CheckIsSensorActive()函数来实施此功能。 CapSense_debounceCounter[]阵列通过下列函数更新： • CapSense_BaseInit() • CapSense_CheckIsSensorActive()

常量

以下常量已进行定义。一些常量有条件地进行定义，且只有在当前配置需要时才会显示。

- CapSense_TOTAL_SENSOR_COUNT — 定义 CapSense CSD 组件内传感器的总数。

传感器常量

为每个传感器提供一个常量。将传感器作为参数使用的所有函数都能够使用这些常量。例如，下面 API 将传感器作为参数使用：

ScanSensor()、ReadSensorRaw()、CheckIsSensorActive()、InitializeSensorBaseline()、UpdateSensorBaseline()、GetBaselineData()、GetDiffCountData()、GetNormalizedDiffCountData()、GetNoiseEnvelope()、WriteSensorRaw()、SetBaselineData()、SetSensitivity()、GetSensitivityCoefficient()、SetLowBaselineReset()、GetLowBaselineReset()。

常量名称由以下内容组成：

实例名称 + *“_SENSOR”* + *Widget 名称* + *元件* + *“元件号”* + *“__”* + *Widget 类型*

这些常量包含在生成的代码（Capsense.h）中。其名称必须为大写。

例如：

```
/* Define Sensors */
#define CapSense_SENSOR_TP1_ROW0__TP    0
#define CapSense_SENSOR_TP1_ROW1__TP    1
#define CapSense_SENSOR_TP1_COL0__TP    2
#define CapSense_SENSOR_TP1_COL0__TP    3
#define CapSense_SENSOR_LS0_E0__LS      5
#define CapSense_SENSOR_LS0_E1__LS      6
#define CapSense_SENSOR_PROX1__PROX     7
```

- Widget 名称 — 用户定义的“Widget”名称（必须为有效“C”式标识符）。Widget 名称在 CapSense CSD 组件中必须是唯一的。所有“Widget”名称均为大写。
- 元件号 — 只有包含多个元件的 Widget（如辐射滑条）才有元件号。对于触控板及矩阵按键，元件号由“Col”或“Row”及其编号组成（如：Col0、Col1、Row0、Row1）。对于线性滑条及辐射滑条，元件号由字母“e” 及其编号组成（如：e0、e1、e2、e3）。
- “Widget”类型 — 存在多种“Widget”类型：

别名	说明
BTN	按键
LS	线性滑条
RS	辐射滑条
TP	触摸板和触控板
MB	矩阵按键
PROX	接近传感器
GEN	通用传感器

别名	说明
GRD	保护传感器

“Widget”常量

为每个 Widget 提供一个常量。将 Widget 作为参数使用的所有函数都能够使用这些常量。例如，下面 API 将 Widget 作为参数使用：

CapSense_CheckIsWidgetActive()、CapSense_EnableWidget()、CapSense_DisableWidget()、CapSense_GetCentroidPos()、CapSense_GetRadialCentroidPos()、CapSense_GetTouchCentroidPos()、ScanWidget()、GetMatrixButtonPos()、GetNoiseThreshold()、GetNegativeNoiseThreshold()、GetFingerThreshold()、GetFingerHysteresis()、SetDebounce()、GetDebounce()、SetFingerHysteresis()、SetNoiseThreshold()、SetNegativeNoiseThreshold()、SetFingerThreshold()。

常量由以下内容组成：

实例名称 + *Widget 名称* + *Widget 类型*

这些常量包含在生成的代码（Capsense_CSHL.h）中。其名称必须为大写。

例如：

```
/* Widgets constants definition */
#define CapSense_UP__BTN        0
#define CapSense_DOWN__BTN      1
#define CapSense_VOLUME__SL     2
#define CapSense_TOUCHPAD__TP   3
```

示例固件源代码

在 Find Example Project 对话框中，PSoC Creator 提供了大量的示例项目，包括原理图和示例代码（**File > Example Project...**）。要获取组件特定的示例，请打开器件目录中的对话框或原理图中的器件实例。要查看通用示例，请打开“Start Page”或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多信息，请参考《PSoC Creator 帮助》中主题为“查找示例项目”一节的内容。

MISRA 合规性

本节介绍了 MISRA-C:2004 合规性和本器件的偏差情况。定义了下面两种类型的偏差：

- 项目偏差 — 适用于所有 PSoC Creator 组件的偏差
- 特定偏差 — 仅适用于该组件的偏差

本节提供了有关组件特定偏差的信息。《*系统参考指南*》的“MISRA 合规性”章节中介绍了项目偏差以及有关 MISRA 合规性验证环境的信息。

CapSense CSD 组件具有如下的特定偏差：

MISRA-C:2004规则	规则类别（必须（R）/建议（A））	规则说明	违规的辩解
8.8	R	外部对象或函数只能在一个文件中声明。	某些阵列的生成取决于组件的配置情况。在使用这些阵列的“.c source”文件内对它们进行声明，并不是在“.h include”文件内进行的。
11.4	A	不同指向对象类型的指针间不能进行转换。	在组件调试器助手中，指向组件结构的指针被转换为8位的数据指针，然后将它们传入到一个用于传输的I2C API内。由于I2C组件只传输字节流，因此必须进行该转换。
17.4	R	阵列索引是唯一允许的指针运算形式。	该组件具有多个使用指针参数的函数。这些参数用于传输数据阵列。可以使用阵列索引来访问它们。
19.7	A	函数应该优先于类似函数的宏使用。	类函数宏用于提高性能。

API 的存储器使用情况

根据不同的编译器、设备、所用 API 的数量以及组件配置，组件存储器的使用有着显著的不同。下表列出了在指定组件配置中可用的所有 API 的存储器使用情况。

使用 Release 模式中配置的关联编译器进行了测量，测量中采用了大小的优化设置。有关特定的设计，可分析编译器生成的映射文件以确定存储器使用情况。

配置	PSoC 4000		PSoC 4100/PSoC 4200	
	闪存字节	SRAM字节	闪存字节	SRAM字节
Widget：5个按键 调试方法：手动 补偿IDAC：使能 自动校准：使能 原始数据滤波器：一阶IIR¼滤波 BIST：使能 预充电模式：PRS_Auto 单一频率设置：使能 Widget分辨率：8位	4020	106	3944	106
Widget：5段线性滑条 调试方法：手动 补偿IDAC：使能 自动校准：使能 原始数据滤波器：一阶IIR¼滤波 位置噪声滤波器：一阶IIR¼滤波 BIST：使能 预充电模式：PRS_Auto 单一频率设置：使能 Widget分辨率：8位	4058	107	3982	107

配置	PSoC 4000		PSoC 4100/PSoC 4200	
	闪存字节	SRAM字节	闪存字节	SRAM字节
Widget：5个按键，5段线性滑条 调试方法：手动 补偿IDAC：使能 自动校准：使能 原始数据滤波器：一阶IIR¼滤波 位置噪声滤波器：抖动滤波 BIST：使能 预充电模式：PRS_Auto 单一频率设置：使能 Widget分辨率：8位	4518	2000	4442	2000
Widget：4x4矩阵按键 调试方法：手动 补偿IDAC：使能 自动校准：使能 原始数据滤波器：一阶IIR¼滤波 BIST：使能 预充电模式：PRS_Auto 单一频率设置：使能 Widget分辨率：8位	4028	163	3952	163
Widget：8x8触控板 调试方法：手动 补偿IDAC：使能 自动校准：使能 原始数据滤波器：一阶IIR¼滤波 位置噪声滤波器：一阶IIR¼滤波 BIST：使能 预充电模式：PRS_Auto 单一频率设置：使能 Widget分辨率：8位	4562	298	4490	298

引脚分配

CapSense 定制器为每个 CapSense 传感器和支持信号生成一个引脚别名。这些别名用于将传感器和信号分配给器件上的物理引脚。将 CapSense CSD 组件传感器和信号分配给“设计范围资源”文件视图中“引脚编辑器”选项卡上的引脚。

传感器引脚

提供别名，使传感器名称与 CapSense 定制器中的 Widget 类型和 Widget 名称相关联。

传感器的别名为：

“Widget”名称 + *元件号* + *“__”* + *“Widget”类型*

Cmod 引脚

外部调制器电容（C_{MOD}）的一侧应连接至物理引脚，另一侧连接至 GND。在 PSoC 4100/PSoC 4200 器件中，C_{MOD} 可以连接至 P4[2]引脚上。在 PSoC 4000 器件中，C_{MOD} 可以连接至 P0[4]引脚上。

C_{MOD} 的建议值为 2.2 nF。

屏蔽电极引脚

可以将屏蔽别名分配给任何可用的引脚。

Cshield_tank 引脚

在 PSoC 4100/PSoC 4200 器件中，可以将 Cshield_tank 连接至 P4[3]引脚上。

功能说明

定义

传感器

传感器是基板上的一个导电元件，它的电容随着触摸提高；该导电元件连接至 PSoC 的一个引脚。

传感器的示例包括：PCB（与 PSoC 相连）上的铜盘、柔性 PCB（与 PSoC 相连）上的铜或银、PET（与 PSoC 相连）上的银墨、玻璃（与 PSoC 相连）上的 ITO。

CapSense Widget

CapSense widget 是一个传感器或拥有相同属性的一组传感器，用于构成功能。

CapSense Widget 的示例包括按键 widget 或接近感应 widget，这些 widget 通常只有一个传感器用于检测有触摸或无触摸状态。由一组传感器（属性相同）构成的 widget 的示例包括线性滑条、辐射滑条、触摸板和矩阵按键 widget。

扫描时间

扫描时间是指 CapSense 组件扫描一个电容传感器的一段时间。

在**手动模式**下，传感器扫描时间取决于分辨率和调制器时钟：

扫描时间（ms） = (2^N-1)*ModDiv / clockInKHz，

其中：

- N — 分频率
- ModDiv — 调制器时钟分频
- clockInKHz — HFCLK 时钟（KHz）

注意：由于定制器对设置和预处理时间作了近似处理，因此处显示的值可能与定制器预估的扫描时间不同。

在**自动（Smartsense）调试模式**下，传感器扫描时间取决于寄生电容（Cp）和灵敏度。

下表显示的是 HFCLK = 24 MHz 时的扫描时间（µs）以及灵敏度和寄生电容。

寄生电容，pF	灵敏度			
	1	2	3	4
10	410	237	237	153
15	750	410	237	237
20	750	410	410	237
25	2800	1440	750	750
30	2800	1440	750	750
35	2800	1440	750	750
40	2800	1440	1440	750
45	2800	1440	1440	750
50	5600	2800	1440	1440

下表显示的是 HFCLK = 24 MHz 时分辨率以及灵敏度和寄生电容。

寄生电容，pF	灵敏度			
	1	2	3	4
10	12	11	11	10
15	13	12	11	11
20	13	12	12	11
25	14	13	12	12
30	14	13	12	12
35	14	13	12	12
40	14	13	13	12
45	14	13	13	12
50	15	14	13	13

注意：扫描时间基于以下设置估算而来：CPU 时钟 = 24 MHz，通道数量 = 1。扫描时间按一次传感器扫描的时间间隔来测量。此时间包括传感器设置时间、采样转换间隔和数据处理时间。通过将所提供的数值进行线性换算，这些值可于估算其他时钟频率以及附加传感器的扫描速度。

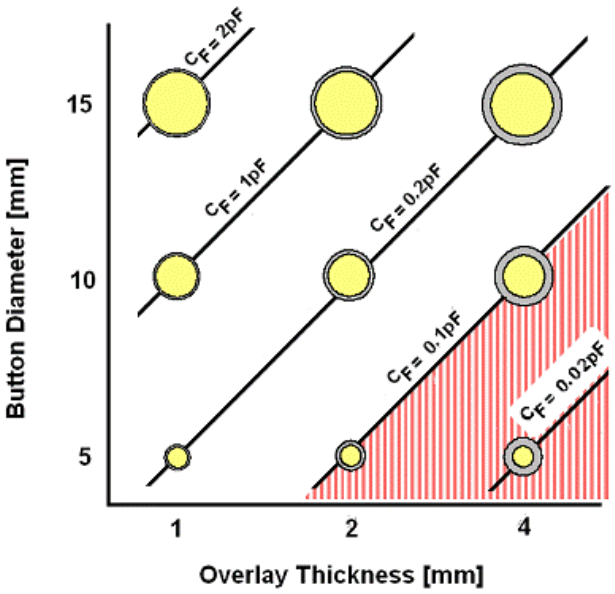
Scan Resolution（扫描分辨率）

该参数定义了扫描为 2^N-1（N — 扫描分辨率）时的最大原始计数值（全刻度范围）。提高分辨率可提高传感器的灵敏度，信噪比和抗噪性能，但是扫描时间延长。

下表显示的是基于 Cp 和手指电容值 Cf 的推荐扫描分辨率设置。Cf 是指传感器上有手指触摸时电容值的变化。Cf 值取决于覆盖层厚度、传感器大小及传感器与其他大型导体的接近程度。

Cp（pF）	Cf = 0.1 pF	Cf = 0.2 pF	Cf = 0.4 pF	Cf = 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

下图显示了 Cf 值，可将其作为覆盖层厚度和圆形传感器直径的函数使用。



Sensor Scan Slot（传感器扫描插槽）

扫描时间插槽是指 CapSense 模块扫描一个或多个结合电容传感器的时间段。在给定的扫描插槽中可以组合多个传感器，以使能联动接近感应等特性。这意味着接近感应传感器可以是一个复合传感器，通过选择其他传感器可以在 **Scan Order** 选项卡中对该传感器进行配置。这些传感器属于复合接近感应传感器的部分。当扫描该复合传感器时，这些传感器将有共用的参数。

为了减少术语混乱，传感器扫描插槽仅表示传感器被扫描的时间周期而不是该传感器本身。

复合传感器部分介绍了如何配置复合传感器。

Raw Count（原始计数）

CapSense 组件测量传感器的电容并提供以数字形式表示的结果，该结果被称为原始计数。原始计数的值随着传感器电容的增加而增加。

Baseline（基线）

传感器的原始计数值随着温度和湿度等环境因素的变化而逐渐变化。这种渐变通过基线值补偿。基线使用软件算法跟踪原始计数中的渐变。低通滤波器对原始计数突变的敏感度较低。基线值为计算差值计数提供了参考值。

Difference Count（差值计数）

差值计数指传感器的原始计数与基线之间的差值。当传感器无手指触摸时，差值计数为零。触摸传感器时，原始计数将增加，从而产生差值计数值。

Sensor State（传感器状态）

如果按键状态为 ON（有触摸），传感器的状态将为 1；如果按键状态为 OFF（无触摸），则传感器状态将为 0。ON 状态又称为活动状态，OFF 状态又称为非活动状态。

Finger Threshold（手指阈值）

该值用于确定传感器上是否有手指触摸。CapSense 组件使用手指阈值参数来判断传感器是否处于活动/非活动状态。 如果传感器的差值计数值超过手指阈值，则传感器被判定为活动状态。

注意：该定义假设迟滞水平为 0，去抖动值为 1。

Hysteresis（迟滞）

迟滞参数与手指阈值一起使用，可确定传感器的状态. 一旦差值计数超过手指阈值与迟滞之和，触摸状态将为 ON。触摸状态一直保持为 ON 状态，直到差值计数下降到小于手指阈值 - 迟滞为止。

这样，当差值计数几乎等于手指阈值时，可以避免存在触摸/无触摸状态机因受噪声影响而报告 ON 和 OFF 状态的情况。

Debounce（去抖动）

“去抖动”参数为传感器活动状态的切换添加了一个计数器。为了让传感器能够从非活动状态切换到活动状态，在特定的样本数量内（去抖动），必须保持差值计数值大于手指阈值与迟滞值之和。

Noise Threshold（噪声阈值）

对于单个传感器，“噪声阈值”参数设置了原始计数的上限值，以更新基线值。对于滑条传感器，它设置了差值计数的下限值，以计算质心。

Negative Noise Threshold（负噪声阈值）

“负噪声阈值”参数可作为负差值计数阈值使用。对于由低基线复位参数指定的采样数，如果原始计数小于基线值与负噪声阈值之差，则将基线值复位为当前的原始计数值。

Low Baseline Reset（低基线复位）

“低基线复位”参数与“负噪声阈值”参数一同使用。它计算复位基线时所需要的异常的低采样数量。它用来修正启动时手指在传感器上触摸的情况。

Sensors Autoreset（传感器自动复位）

此参数用于确定是否随时更新基线，或者仅确定差值计数低于噪声阈值的时间。

使能 Sensors Autoreset（传感器自动复位）时，会不断地更新基线。此设置限制了传感器在长期被连续触摸时可报告 ON 状态的最大持续时间（典型值为 5 至 10 秒 ），但可在无任何物体触摸传感器而原始计数突然上升的情况下，阻止传感器始终报告为 ON 状态。系统中的电气损坏或不可接受如金属物体突然落下在屏幕上等操作都会导致原始计数突然上升的现象。

如果将“传感器自动复位”参数设置为禁用，那么仅当差值计数低于噪声阈值时，才会更新基线。只要传感器上发生了触摸，传感器即可报告为 ON 状态。

Parasitic Capacitance（寄生电容，C_p）

寄生电容是指传感器上残留的电容量。这是传感器上未发生触摸时测量到的电容值

传感器的寄生电容受下面各个因素的影响：PCB 布局、PCB 材料的介电常量、PCB 厚度、覆盖层材料以及它的厚度。环境条件（如温度）也会对 PCB 材料的电气常量产生影响，这样会间接影响传感器上的寄生电容。

Finger Capacitance（手指电容，C_f）

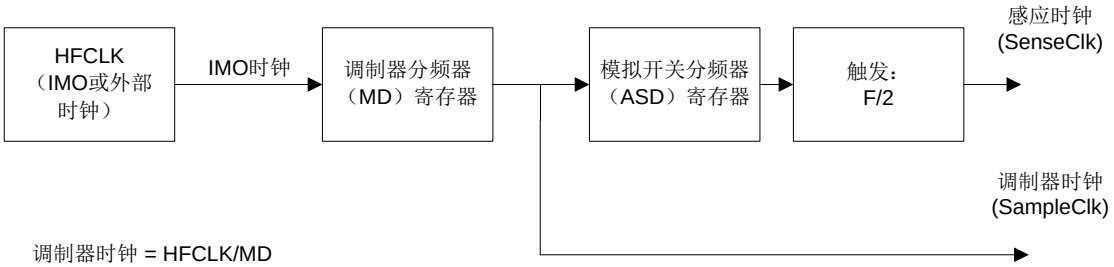
手指电容是指归因于手指触摸在传感器上的电容。

CapSense Clocking（CapSense 时钟）

PSoC 4100/PSoC 4200

PSoC 4100/PSoC 4200 器件的时钟被链接起来。下图显示的是 PSoC 4100/PSoC 4200 的 CapSense 时钟树。

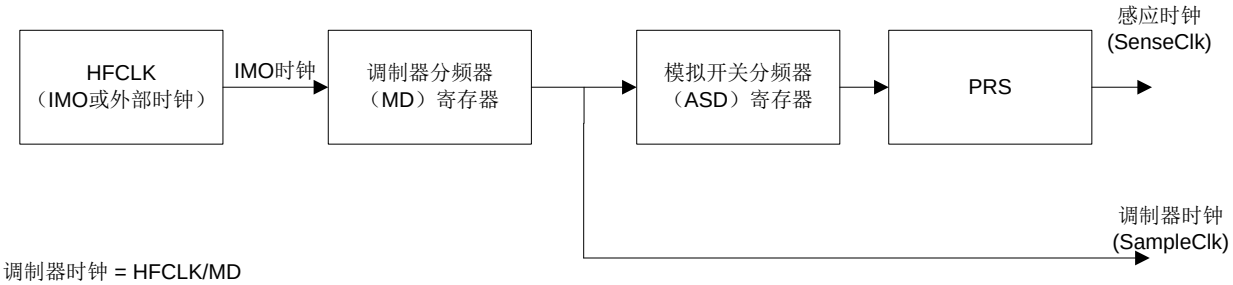
PSoC 4100/PSoC 4200中直接时钟模式的时钟：



调制器时钟 = HFCLK/MD

感应时钟 = HFCLK/MD/ASD/2

PSoC 4100/PSoC 4200中PRS时钟模式的时钟：



调制器时钟 = HFCLK/MD

PRS最大感应时钟 = HFCLK/MD/ASD/2

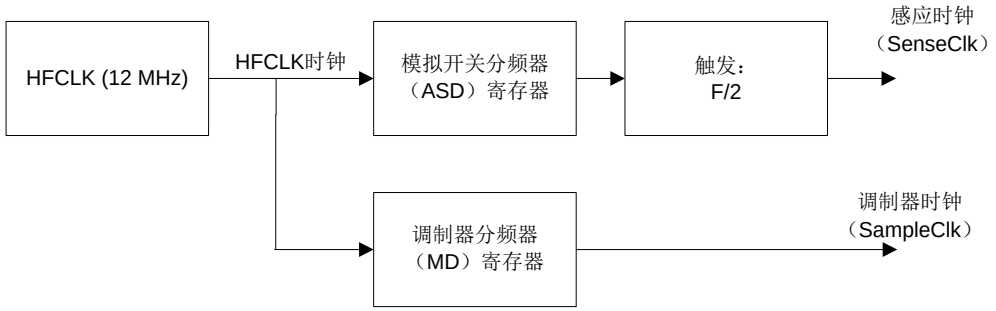
PRS平均感应时钟 = HFCLK/MD/ASD/4

将 HFCLK 时钟除以调制器时钟分频器，就形成了调制时钟。将调制时钟除以感应时钟分频器，就形成了感应时钟。例如，如果您将感应时钟分频器的值设为 8，且调制器时钟分频器的值被设为 4，那么将调制器时钟分频寄存器配置为除以 4，而且将感应时钟分频寄存器配置为除以 2。

PSoC 4000

PSoC 4000 器件的时钟被链接起来。下图显示的是 PSoC 4000 的 CapSense 时钟树。

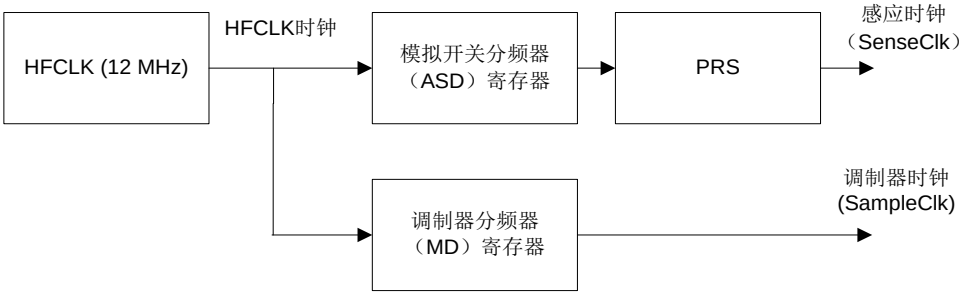
PSoC 4000中直接时钟模式的时钟：



调制器时钟 = HFCLK/MD

感应时钟 = HFCLK/ASD/2

PSoC 4000中PRS时钟模式的时钟：



调制器时钟 = HFCLK/MD

PRS 最大感应时钟 = HFCLK/ASD/2

PRS 平均感应时钟 = HFCLK/ASD/4

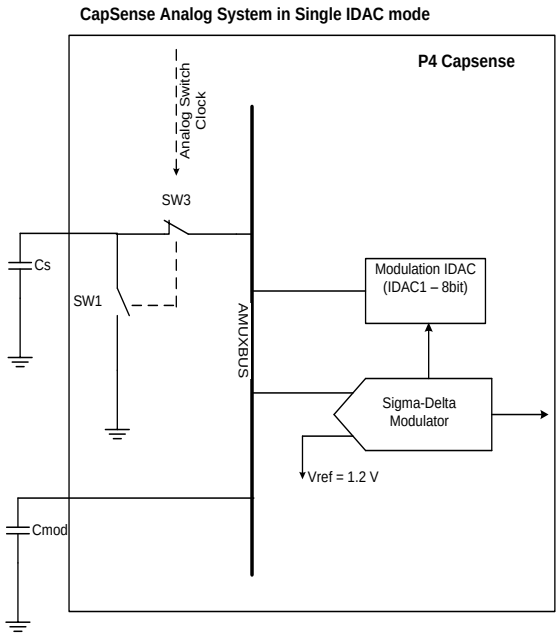
CapSense 模拟系统

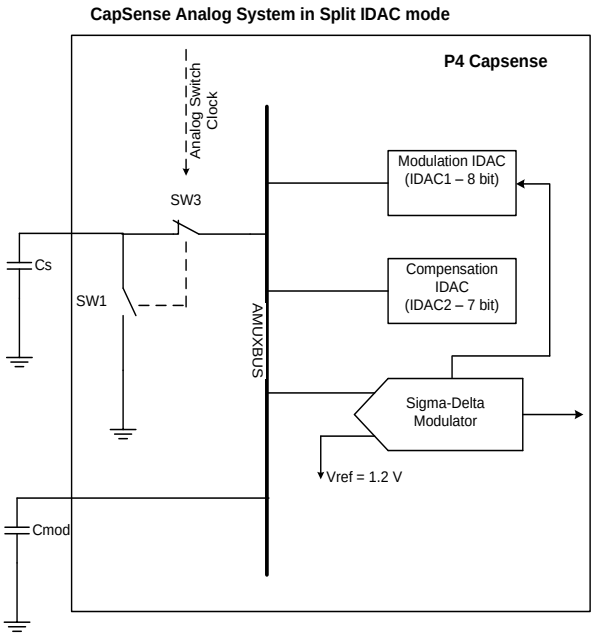
CapSense 模拟系统包括 Sigma Delta 调制器、模拟复用器总线、调制 IDAC（IDAC1 – 8 位，即主 IDAC）以及补偿 IDAC（IDAC2 – 7 位，即第二 IDAC）。

在单 IDAC 模式下，补偿 IDAC 在“定制器”通用选项卡上被禁用，且组件仅使用主 IDAC（IDAC1 – 8 位）。在这种情况下，主 IDAC 被配置为由调制器输出控制的变量。

在基线 IDAC 模式下，补偿 IDAC 在“定制器”通用选项卡上被禁用，并且组件使用两个 IDAC（8 位主 IDAC 和 7 位第二 IDAC）。

在这种情况下，由于主 IDAC（8 位）被配置为变量 IDAC，所以它被称为调制 IDAC。同样，由于第二 IDAC（7 位）被配置为固定 IDAC，所以将其称为‘补偿’。





API 分辨率 — 插值和缩放比例

在滑条传感器和触控板中，通常需要分辨手指（或其他电容器物体）更精确的位置，而非单个传感器本身的分辨率。手指在滑条传感器或触控板上的触摸面积往往大于任何一个传感器。

为了采用一个质心计算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。要求提供一定数量的相邻传感器信号，且高于噪声阈值。如果发现最强信号，将使用此信号和大于噪声阈值的相邻连续信号计算中心位置。使用少至两个、多至八个传感器计算中心位置。

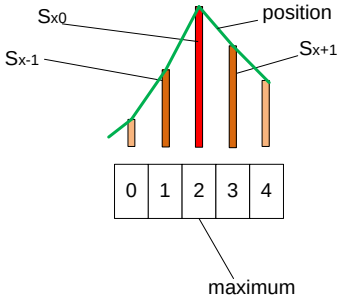
PSoC4中的CapSense_GetCentroid（CapSense_CalcCentroid）函数（对于线性滑条）

$$position = \left(\frac{S_{x+1} - S_{x-1}}{S_{x-1} + S_{x0} + S_{x+1}} + maximum \right) * (Resolution / (n-1))$$

分辨率：定制器中所设置的API分辨率，
n：定制器中的传感器要素数量。

Maximum： 质心内最大值要素的指数。

S_i：靠近最大位置的差值计数（已经消减手指阈值）



示例1：
5要素线性质心的分辨率为100。噪声阀值 = 2。
CapSense_sensorSignal= [0, 0, 100, 200, 100]。最大值 = 3；
这样，位置值： ((98-98)/(98+108+98) + 3)*100/(5-1) = 75.

示例2：
5要素线性质心的分辨率为100。噪声阀值 = 20。
CapSense_sensorSignal= [0, 10, 100, 210, 180]。最大值 = 3；
这样，位置值 = ((160-80)/(80+190+160) + 3)*100/(5-1) = 79.65 = 80 取整数

辐射滑条的第一注意：

$$位置值= \left(\frac{S_{x+1} - S_{x-1}}{S_{x-1} + S_{x0} + S_{x+1}} + maximum \right) * (Resolution / n)$$

如果位置值 < 0，那么

$$位置值 = \left(\frac{S_{x+1} - S_{x-1}}{S_{x-1} + S_{x0} + S_{x+1}} + maximum + n \right) * (Resolution / n)$$

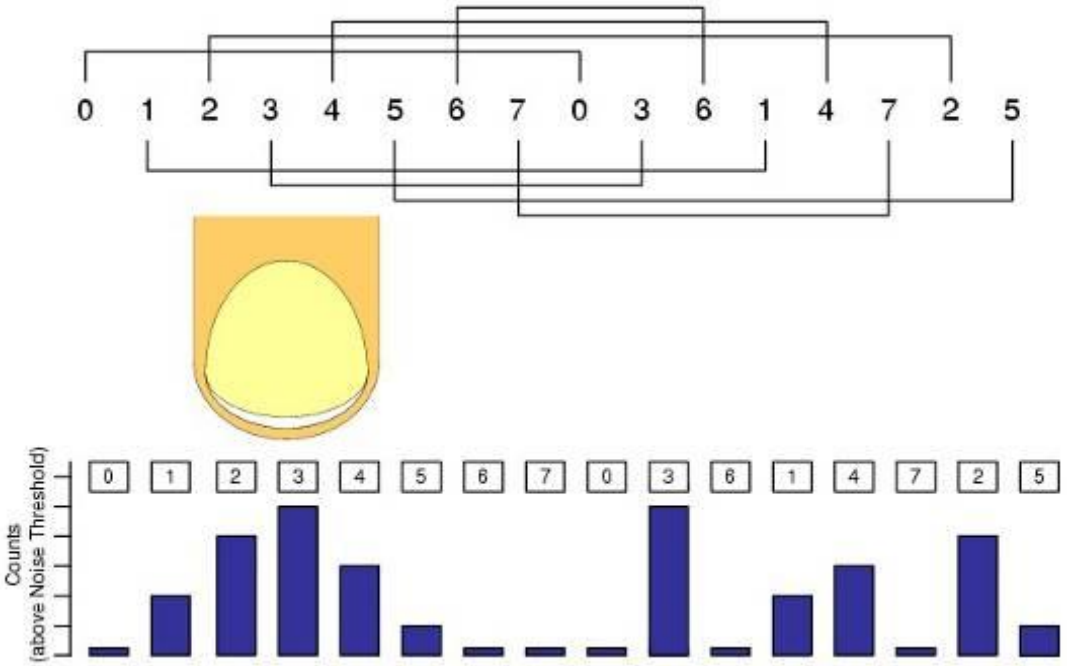
辐射滑条的第二注意：
对于辐射滑条，算法使用第一个和最后滑条段。
比如，如果CapSense_sensorSignal= [30, 0, 0, 40, 180]，则根据X0、x3和x4元素来计算辐射滑条中位置的值。但是，在线性滑条中，仅根据x3和x4元素来计算位置值。

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 个传感器使用 0 到 100 的范围），需要用质心值乘以标量。另一种更有效的方法是将内插法和按比例计算的方法统一到一个计算中，并按所需的比例因子直接报告结果。这一过程可以在高级 API 内进行处理。滑条传感器计数和分辨率都在 CapSense CSD 定制器中进行设置。

双工

在双工滑条中，滑条中的每个 PSoC 传感器连接都会映射到一组滑条传感器中的两个物理位置上。物理位置的第一半（较低数值部分）按顺序映射到基部分配的传感器上，端口引脚由设计人员使用 CapSense 定制器分配。物理传感器位置的另一半（较高数值部分）由定制器中的算法自动映射，并在包括（include）文件中列出。一旦创建好次序，一半相邻的传感器动作则不会使另一半相邻传感器的动作。小心地确定此次序，将其映射到印刷电路板上。

图 1. 双工



应当使滑条中的传感器电容均衡。根据传感器或 PCB 布局，某些传感器对可能需要更长的走线。当您选择双工法时，CapSense 定制器将自动生成双工传感器编号表，下方列出这些内容，以供参考。

表 1. 不同滑条段计数的双工序列

滑条段总数	段序列
10	0、1、2、3、4、0、3、1、4、2
12	0、1、2、3、4、5、0、3、1、4、2、5
14	0、1、2、3、4、5、6、0、3、6、1、4、2、5
16	0、1、2、3、4、5、6、7、0、3、6、1、4、7、2、5
18	0、1、2、3、4、5、6、7、8、0、3、6、1、4、7、2、5、8
20	0、1、2、3、4、5、6、7、8、9、0、3、6、9、1、4、7、2、5、8
22	0、1、2、3、4、5、6、7、8、9、10、0、3、6、9、1、4、7、10、2、5、8
24	0、1、2、3、4、5、6、7、8、9、10、11、0、3、6、9、1、4、7、10、2、5、8、11
26	0、1、2、3、4、5、6、7、8、9、10、11、12、0、3、6、9、12、1、4、7、10、2、5、8、11
28	0、1、2、3、4、5、6、7、8、9、10、11、12、13、0、3、6、9、12、1、4、7、10、13、2、5、8、11
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

滑条段总数	段序列
50	0、1、2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17、18、19、20、21、22、23、24、0、3、6、9、12、15、18、21、24、1、4、7、10、13、16、19、22、2、5、8、11、14、17、20、23
52	0、1、2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17、18、19、20、21、22、23、24、25、0、3、6、9、12、15、18、21、24、1、4、7、10、13、16、19、22、25、2、5、8、11、14、17、20、23
54	0、1、2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17、18、19、20、21、22、23、24、25、26、0、3、6、9、12、15、18、21、24、1、4、7、10、13、16、19、22、25、2、5、8、11、14、17、20、23、26
56	0、1、2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17、18、19、20、21、22、23、24、25、26、27、0、3、6、9、12、15、18、21、24、27、1、4、7、10、13、16、19、22、25、2、5、8、11、14、17、20、23、26

中断服务子程序

CapSense 组件使用每次传感器扫描结束后触发的中断。您可以在提供的存根子程序中根据需要添加自己的代码。首次构建项目时，将在 *CapSense_INT.c* 文件中生成存根子程序。您的代码必须添加在所提供的注释标签之间，在版本中得到保留。

滤波器

CapSense 组件中提供了几种滤波器：中值、平均值、一阶 IIR 和抖动。这些滤波器既可与传感器原始数据（raw data）结合使用，以降低传感器噪声，也可与滑条和触摸板的位置数据结合使用，以降低位置噪声。

中值滤波器

中值过滤器查看最近三次采样并报告中值。中值是通过整理这三次采样并取中间值来计算的。该滤波器用于消除短时噪声尖峰并生成一次采样的延迟。由于这种延迟以及 RAM 消耗，通常建议不要启用该滤波器。启用该滤波器对于每个传感器（原始）和 Widget（位置）会消耗 RAM 的 4 个字节。默认情况下，它被禁用。

均值滤波器

平均值滤波器查看位置的最近三次采样并报告简单的平均值。它用于消除短时噪声尖峰并生成一次采样的延迟。由于这种延迟以及 RAM 消耗，通常建议不要启用该滤波器。启用该滤波器对于每个传感器（原始）和 Widget（位置）会消耗 RAM 的 4 个字节。默认情况下，它被禁用。

一阶 IIR 滤波器

一阶 IIR 滤波器是原始和传感器滤波器建议启用的滤波器，因为它所需的 SRAM 最小并且响应迅速。IIR 滤波器标度最近的传感器或位置数据，并将其添加到已标度版本的前一个滤波器输出。启用该滤波器对于每个传感器（原始）和 Widget（位置）会消耗 RAM 的 2 个字节。原始和位置滤波器在默认情况下启用 IIR1/4。

一阶 IIR 滤波器：

$$IIR\ 1/2 = 1/2\,previous + 1/2\,current$$

$$IIR\ 1/4 = 3/4\,previous + 1/4\,current$$

$$IIR\ 1/8 = 7/8\,previous + 1/8\,current$$

$$IIR\ 1/16 = 15/16\,previous + 1/16\,current$$

抖动滤波器

该滤波器可消除在两个值之间切换（抖动）的原始传感器或位置数据中的噪声。如果最近的传感器值大于上一个传感器值，那么前一个滤波器值将增加 1，反之则会递减。当应用于包含四个或更少 LSB 峰-峰噪声的数据、或者慢速响应可接受时，这最有效。后者对于某些位置传感器有用。启用该滤波器对于每个传感器（原始）和 Widget（位置）会消耗 RAM 的 2 个字节。默认情况下，它被禁用。

水对 CapSense 系统的影响

水珠和手指对 CapSense 的影响相似。然而，水珠对传感区整个表面的影响不同于手指的影响。

水对 CapSense 表面的影响有几种不同形式：

- 器件表面形成的细水流。
- 单独的水珠。
- 当冲洗或浸泡器件时，水流会覆盖器件的全部或大部分表面。

水含有的盐或矿物使其具有导电性。而且，浓度越高，水的导电性就越强。肥皂水、海水和矿物质水等液体对 CapSense 有不利影响。这些液体模拟手指触摸器件表面的效果，这可导致器件运转出错。

防水和检测

此特性可配置 CapSense CSD 组件，以抑制水对 CapSense 系统的影响。此功能设置以下参数：

- 使能屏蔽电极，此电极将用于在硬件层面上补偿水珠对传感器的影响。

屏蔽电极

某些应用场合要求即使存在水膜或水珠，也能可靠地运行。白色家电、汽车、各种工业领域和其他领域应用，都需要使用不会因为水、冰和湿度的变化（会导致凝结）而提供假触发信号的电容式传感器。在这种情况下，可以使用单独的屏蔽电极。此电极位于感应电极之后或其周围。如果器件覆盖层表面有水膜，则屏蔽和感应电极之间的耦合会加剧。屏蔽电极有助于降低寄生电容的影响，为处理传感电容的变化提供了更具动态性的数值范围。

在某些应用场合，选择屏蔽电极信号及其相对于感测电极的位置，使由于潮湿所导致两个电极之间耦合的增大，引起感测电极电容测量值负向触摸变化，这样做很有用。这样可以抑制由于潮湿造成的误触，从而简化高级软件 API 的工作。CapSense CSD 组件支持屏蔽电极的单独输出，从而简化 PCB 布线。

图 2. 可能的屏蔽电极 PCB 布局

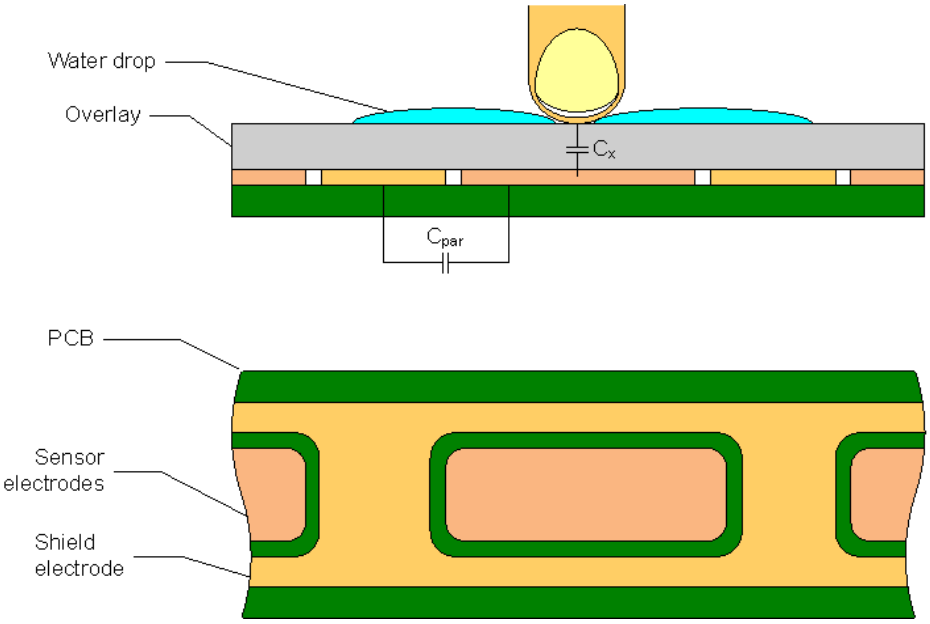


图 2 显示的是一种按键屏蔽电极的可能布局组态。屏蔽电极尤其适用于透明的 ITO 触摸板器件，在这种器件中，它不但可阻止 LCD 驱动电极的噪声，同时可减少杂散电容。

在该示例中，屏蔽电极板覆盖了按键。作为另一种替代方法，屏蔽电极可以安装在相对的 PCB 层上，其中包括按键下面的平板。对于这种情况，建议使用填充模式，填充率约为 30 ~ 40%。这时，无需附加的接地层。

如果屏蔽电极与感应电极之间出现水珠，“寄生电容”（C_{PAR}）将增加，调制器电流下降。

屏蔽电极可以连接到任何引脚。将驱动模式设置为慢速强驱动可以降低接地噪声和辐射。另外，可以在 PSoC 器件与屏蔽电极之间连接上升限制电阻。

如何使用接近感应传感器

通过接近传感器可以检测传感器周围的三维空间内是否有手指。然而，类似于 CapSense 按键，接近传感器的实际输出是一个 ON/OFF 状态。通过使用 CapSense_CheckIsSensorActive()或 CapSense_CheckIsWidgetActive() API，可以检测接近感应传感器的 ON/OFF 状态。

根据传感器的结构，在几厘米到几十厘米的距离内，接近感应传感器可以检测到一个手指。为了增加检测到的距离，接近感应传感器环绕直径也要增加。实际上，已配置好的接近感应传感器具有 16 位的扫描分辨率，它需要的扫描时间比通用传感器的扫描时间会更长。由于扫描时间长，在默认情况下，将不会扫描接近感应 Widget。使用 CapSense_EnableWidget()函数使能接近感应 widget。

CapSense_GetDiffCountData() API 可用于读取接近感应传感器上的信号级别。定制器为 *Capsense_CSHL.h* 和 *Capsense.h* 文件中的接近感应 Widget/传感器编号提供了#difine。更多有关信息，请参考 Widget 常量和传感器常量各节。

实现接近感应传感器的另一种方法是将多个传感器放在一起。具体情况是，通过固件将多个传感器板组合成一个较大的传感器。该方法的缺点是会产生较大的寄生电容。更多有关信息，请参考本文档中复合传感器一节的内容。

使用资源

数字资源

配置	资源类型	
	CSD固定模块	中断
所有配置	1	1

模拟资源

配置	资源类型	
	8位CapSense IDAC	7位CapSense IDAC
补偿IDAC被禁用	1	0
补偿IDAC被使能	1	1
SmartSense	1	1

直流和交流的电气特性

除非另有说明，否则这些规范的适用条件是：–40 °C ≤ T_A ≤ 85 °C，T_J ≤ 100 °C 且电压范围为 1.71 V 到 5.5 V。

直流规范

参数	说明	最小值	典型值	最大值	单位	条件
V_{CSD}	工作电压范围	1.71	–	5.5	V	

交流规范

参数	说明	最小值	典型值	最大值	单位	条件
信噪比	手指计数与噪声比率	5	–	–	比例	1) 电容值范围 = 9 ~ 35 pF，灵敏度 = 0.1 pF。 2) 电容值范围 = 9 ~ 45 pF，灵敏度 = 0.2 pF。
IDAC1	8位分辨率的差分非线性（DNL）	-1	–	1	LSB	
IDAC1	8位分辨率的积分非线性（INL）	-3	–	3	LSB	
IDAC2	7位分辨率的差分非线性（DNL）	-1	–	1	LSB	
IDAC2	7位分辨率的积分非线性（INL）	-3	–	3	LSB	
IDAC1_CRT1	高范围的Idac1（8位）输出电流	–	612	–	µA	
IDAC1_CRT2	低范围的Idac1（8位）输出电流	–	306	–	µA	
IDAC2_CRT1	高范围的Idac2（7位）输出电流	–	305	–	µA	
IDAC2_CRT2	低范围的Idac2（7位）输出电流	–	153	–	µA	

组件更改

版本	更新内容	更改原因/影响
2.10.a	编辑了数据手册。	将默认值添加到一些参数并阐明了器件支持。 说明了CapSense_TunerComm() API为阻塞调用。 将新的参数添加到交流规范 添加了CapSense_EnableRawDataFilters 和 CapSense_DisableRawDataFilters API。
2.10	屏蔽被禁用时，会将定制器中的“Shield tank capacitor”（屏蔽槽电容）字段设置为“Disabled”（禁用）。 屏蔽电极被禁用时，会将“Shield signal delay”（屏蔽信号延迟）被设为“None (default)”（无（默认）），并在定制器中以灰色显示。 屏蔽电极被禁用时，会将“Shield signal delay”（屏蔽信号延迟）被设为“None (default)”（无（默认））。 屏蔽槽电容的预充电设置在定制器中以灰色显示。 已向本数据手册添加了有关使用接近感应特性的其他说明内容。 本应用手册中提供了扫描时间值和分辨率。 纠正了CSD只被配置为通用时的构建错误。 更新了调试器，以显示当Auto Calibration选项被使能时在手动调试模式下的实际IDAC值。 Scan Order选项卡上的灵敏度参数在定制器中以灰色显示，以便能够手动调试。	新器件与特性。

版本	更新内容	更改原因/影响
2.0	添加了PSoC 4000器件的支持。 更新了调试和扫描算法。 在对话框中更改了调试模式和IDAC的名称： • 已将“Baselining IDAC”重命名为“Modulation IDAC”，并且它始终是8位； • 已将“Compensating IDAC”重命名为“Compensation IDAC”，并且它始终为7为； • 已将“None Tuning Method”重命名为“Manual”； • 已将“Manual Tuning”方法重命名为“Manual with run-time tuning”方法； • 已将CapSense_idac1Settings阵列改名为CapSense_modulationIDAC阵列； • 已将CapSense_idac2Settings阵列改名为CapSense_compensationIDAC阵列； 为参数设置/读取添加了新API。 添加了BIST支持。 添加了手动模式的自动校准支持。	新器件。 更好的性能。 增加可用性。
1.11	更改了多个全局阵列的名称及说明，并添加了一些非描述性全局阵列。	
	已添加了MISRA合规性章节。	未证明该组件符合MISRA-C:2004编码准则。
1.10	扫描时间已得到优化。	
1.0.a	已更新PSoC 4 CapSense设计指南的链接，并对数据手册进行多处编辑。	
1.0	初始版本。	

©赛普拉斯半导体公司，2014-2015。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。也不根据专利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，否则赛普拉斯不保证产品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®和 CapSense®是注册商标；SmartSense™、PSoC Creator™和 Programmable System-on-Chip™是赛普拉斯半导体公司的商标。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。除上述指定的用途外，未经赛普拉斯明确的书面许可，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：赛普拉斯不针对此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。赛普拉斯保留在不做出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中，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。

产品使用可能受适用于赛普拉斯软件许可协议的限制。