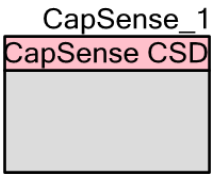
电容式感应（CapSense® CSD）

3.40



特性

- 支持用户定义的按键、滑条、触摸板和接近电容传感器的各种组合。
- 提供 SmartSense™ 自动调试或通过集成式 PC GUI 进行手动调试。
- 对交流电力线噪音、EMC 噪音和电源电压变化所具有的较强的抗扰度。
- 两个可选扫描信道（并行同步）提高了传感器的扫描速率。
- 即使存在水膜或水滴的情况下，屏蔽电极仍可为可靠的运行提供保证。
- 使用 CapSense 自定义程序指导传感器和引脚的分配

概述

电容式感应通过使用 Delta-Sigma 调制器（CapSense CSD）组件，能够有效地测量触摸感应按键、滑条、触摸板和接近检测等应用中传感器感应电容的微小变化。阅读本数据手册后，请阅读以下文档。具体情况请参见赛普拉斯半导体公司网址 www.cypress.com：

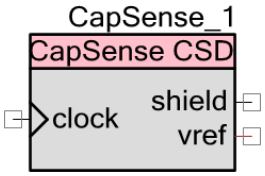
- *Capsense 入门*
- *PSoC®3 和 PSoC®5LP CapSense® 设计指南*

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。

输入/输出接口

本节介绍 CapSense CSD 组件的各种输入和输出接口。I/O 列表中的星号（*）表示，在 I/O 说明部分中所列出的特定条件下，该 I/O 可能不可见。



时钟 — 输入*

为 CapSense CSD 组件提供时钟。时钟输入仅在选中 **Enable clock input**（使能时钟输入）参数时才可见。

屏蔽 — 输出 *

此输出连接屏蔽电极信号。只有使能了 **Shield**（屏蔽）参数时，才可以使用该连接。请参考 Shield（屏蔽电极）参数。有关屏蔽电极使用的更多信息，请参考 Shield Electrode Use and Restrictions（屏蔽电极使用及限制）一节。

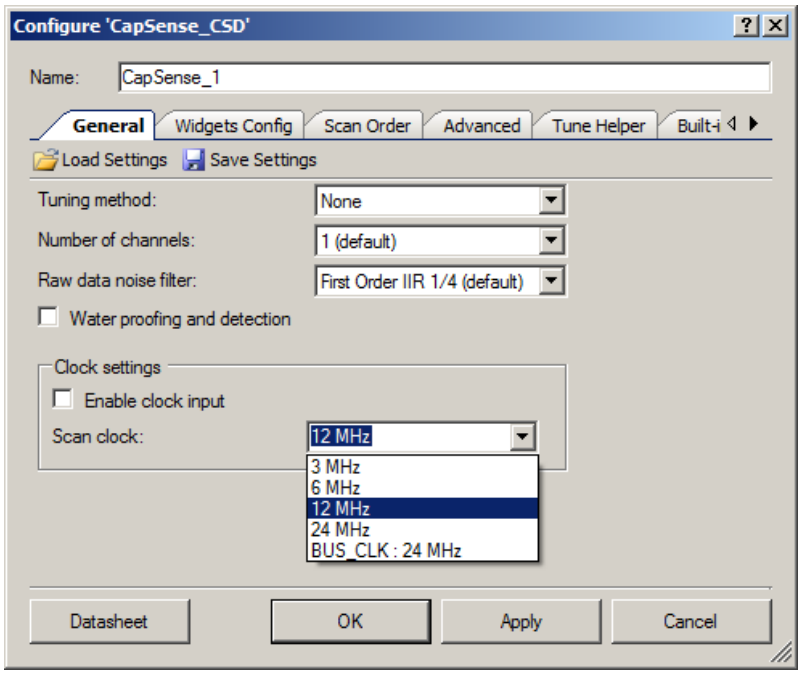
vref — 输出 *

此输出连接模拟基准电压。可用于调整屏蔽信号振幅。只有使能了 IDAC **电流源模式**中的 **Shield** 选项时，才可使用该连接。SIO 用于屏蔽信号时，Vref 输出应连接到 SIO 参考。有关 vref 使用的详细信息，请参考功能说明一节。

组件参数

将 CapSense CSD 组件拖放到您的设计区（TopDesign），然后双击打开 **Configure** 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工程。**Load Settings** 用于加载以前保存的设置。

被保存的组件配置和调试数据可以导入调谐器 GUI。

Tuning method （调试方法）

此参数指定了调试方法。 其中有三个选项：

- **“Auto (SmartSense)（自动（SmartSense））”** — 提供 CapSense CSD 组件的自动调校。

  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试参数。在该模式下，需要额外的 RAM 和 CPU 资源。

重要提示 — 在 SmartSense 模式下只能向项目原理图上放置一个 CapSense_CSD 组件。可使用 EZI2C 通信组件进行 SmartSense 调试，该组件在 **Tuner Helper**（调试器助手）选项卡中指定用于将目标设备中的数据传输到调试器 GUI。

- **Manual**（手动） — 允许用户使用调谐器 GUI 手动调校 CapSense CSD 组件。

  要启动 GUI， 请右击符号， 然后选择 **“Launch Tuner**（启动调谐器）”。有关手动调校的更多信息， 请参阅本数据手册中的“调谐器 GUI 用户指南”一节。手动调试时需要 EZI2C 通信组件，该组件在 **Tuner Helper**（调试器助手）选项卡中指定用于在目标设备和调试器 GUI 之间传输数据。

- **“None**（禁用）”（默认设置） — 禁用调校。

  所有调校参数均存储于闪存中。 只有在 CapSense 组件的所有参数调校完成后，才可用此选项。 如果用户使用该选项，“Tuner（调谐器）”将工作在只读模式。

通道数量

此参数指定了硬件扫描通道的数量。

- **1**（默认）— 最好用于 1 到 20 个传感器。 组件每次能够执行一次电容式扫描。每次连续地扫描一个传感器。由于硬件只实现了一个单通道，此选项将实现最少的硬件资源使用。
  - AMUX 总线连接在一起。

  注意：如果所有的电容传感器均位于芯片的左侧（例如，偶数号端口 GPIO：P0[X]、P2[X]、P4[X]）或右侧（奇数号端口 GPIO，例如：P1[X]、P3[X]、P5[X]），AMUX 总线不连接在一起；仅使用一半 AMUX 总线。

  注意：端口引脚 P15[0-5]与左侧和右侧的不同 AMUX 总线均有连接。P12[X]和 P15[6-7]没有与 AMUX 总线连接。选定部分请参照 TRM。
  - 此组件能够扫描 1 至（GPIO 数量-1）个电容传感器。
  - 需要一个 C_{MOD} 外部电容。
- **2 —** 最好用于多于 20 个传感器。此组件能够同时执行两个电容式扫描。同时使用左侧和右侧 AMUX 总线，每个通道一个。同时连续扫描左右两个传感器（一个右传感器和一个左传感器）。如果一个通道具有的传感器比其他通道多，在其他通道的传感器扫描结束后，传感器更多的通道将继续扫描其阵列中的其余传感器，每次一个，直到完成。与一个通道相比，两个通道需要使用双倍的资源，但同时也使传感器扫描速率翻了一倍。
  - 左侧 AMUX 总线可以扫描 1 至（偶数端口 GPIO 数量 -1）个电容传感器。
  - 右侧 AMUX 总线可以扫描 1 至（奇数端口 GPIO 数量 -1）个电容传感器。
  - 需要两个 C_{MOD} 外部电容，每个通道 1 个。
  - 并行扫描以相同的扫描速率进行。

原始数据（Raw Data）噪声滤波器

该参数用于选择原始数据滤波器类型。 只能选择一个滤波器，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，请参见本文档中功能说明一节的滤波器部分。

- **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。
- **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。
- **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。

Water proofing and detection（防水及检测）

此功能配置 CapSense CSD，使其支持防水功能（默认为禁用）。此功能设置以下参数：

- 使能屏蔽电极输出
- 添加“保护”传感器

  注意：如果防水无需 Guard widget（保护传感器），则可以在 **Advanced**（高级） 选项卡上将其禁用。

使能时钟输入

通过此参数可以选择组件使用内部时钟，还是显示用户提供的时钟连接的输入终端（默认为禁用）。

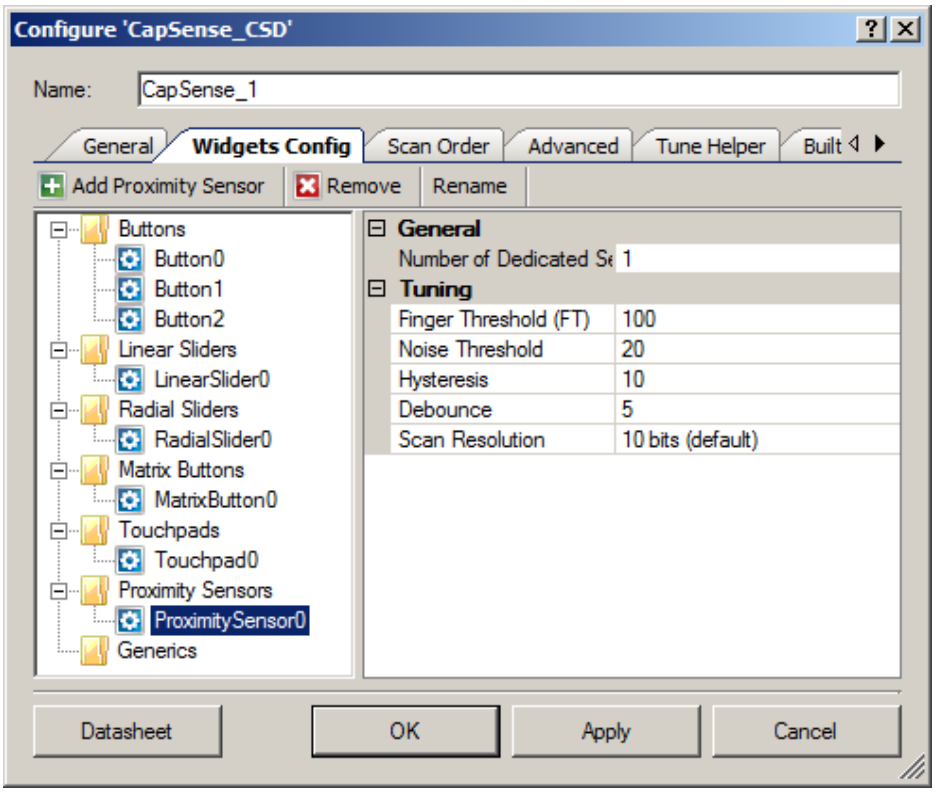
注意：如果调试方法为 **Auto (SmartSense)**，则该选项不可用。因为定制器必须知道时钟频率后才能计算内部数据。

扫描时钟

此参数指定内部 CapSense 组件时钟频率。值的范围为 3 MHz 至 24 MHz（默认为 12 MHz）。如果选择了 **Enable clock input**（使能时钟输入），则该特性不可用。

注意：Analog Switch Drive Source（模拟开关驱动源）设置为 **FF Timer**（FF 定时器）、**Digital Implementation**（数字实现）设置为 **FF Timer**（FF 定时器）时、或以上二者，不支持小于或等于 BUS_CLK 的 CapSense CSD 时钟，用户应选择 BUS_CLK。

Widget Config（Widget 配置）选项卡



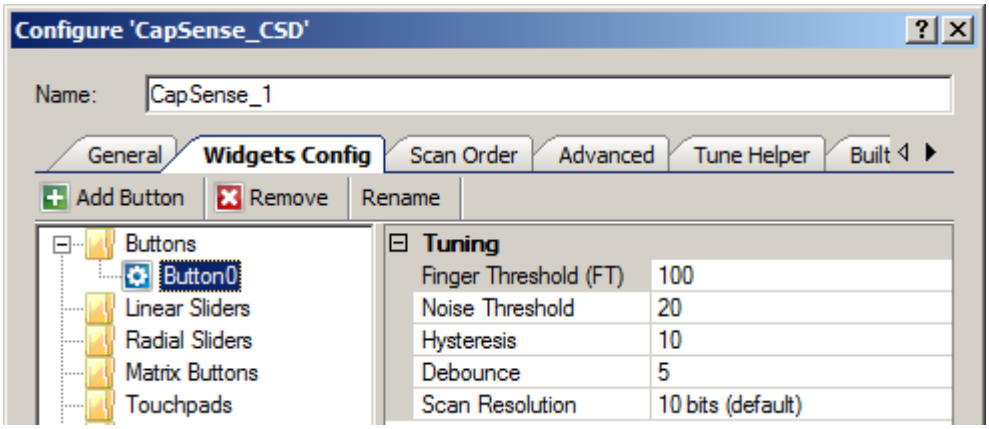
有关各种参数的定义， 请参见功能说明一节中的内容。

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。 触摸板由多个行传感器和列传感器组成。
  - **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计数（raw counts）。 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 widget** （删除 Widget）（热键 — Delete） — 从列表中删除所选的 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。

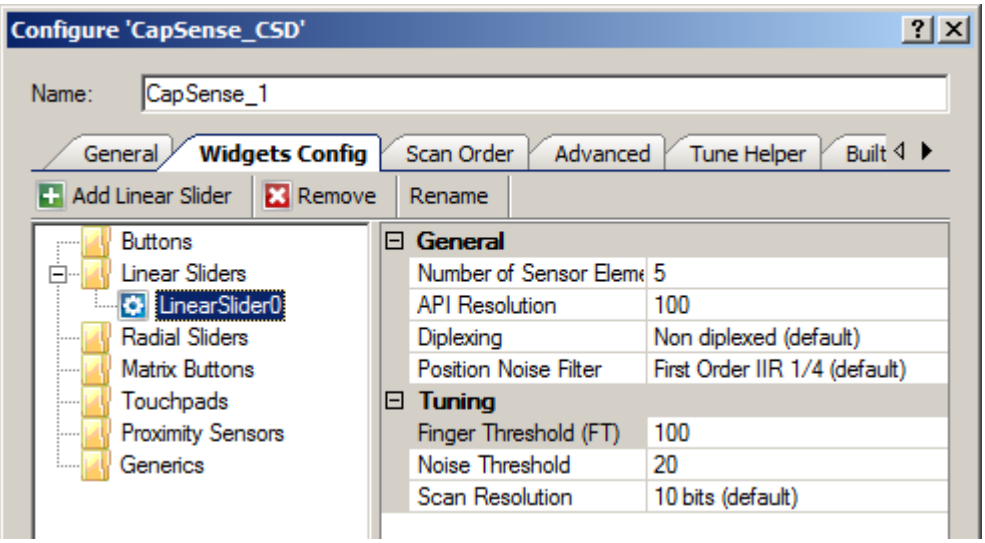
- **Noise Threshold**（噪声阈值） — 定义传感器噪声阈值。如果计数值高于该阈值，则不更新基线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致误触摸或触摸遗漏。如果噪声阈值过高，手指触摸可能会被认为是噪声，人为使基线错误的升高，从而导致手指触摸遗漏。默认值为 **20**。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。

- **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]。

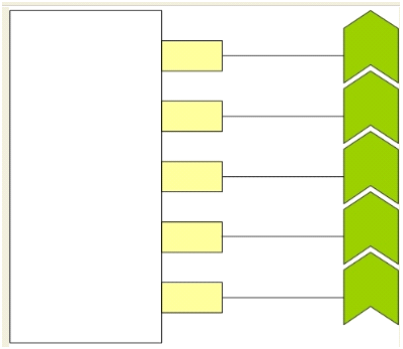
- **Scan Resolution**（扫描分辨率） — 定义扫描分辨率。该参数会影响按键 Widget 传感器的扫描时间。N 位的扫描分辨率最大原始计数为 $2^N - 1$。提高分辨率可提高触摸检测的灵敏度和信噪比（SNR），但会延长扫描时间。默认值为 **10 位**。

Linear Sliders（线性滑条）

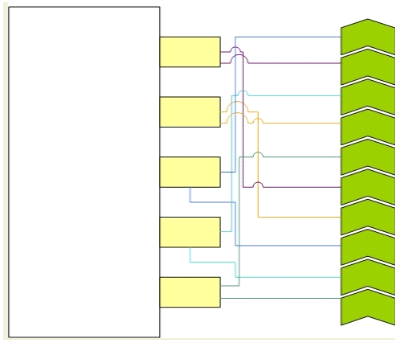


General（通用）：

- **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]。
- **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。



Non Diplexed



Diplexed

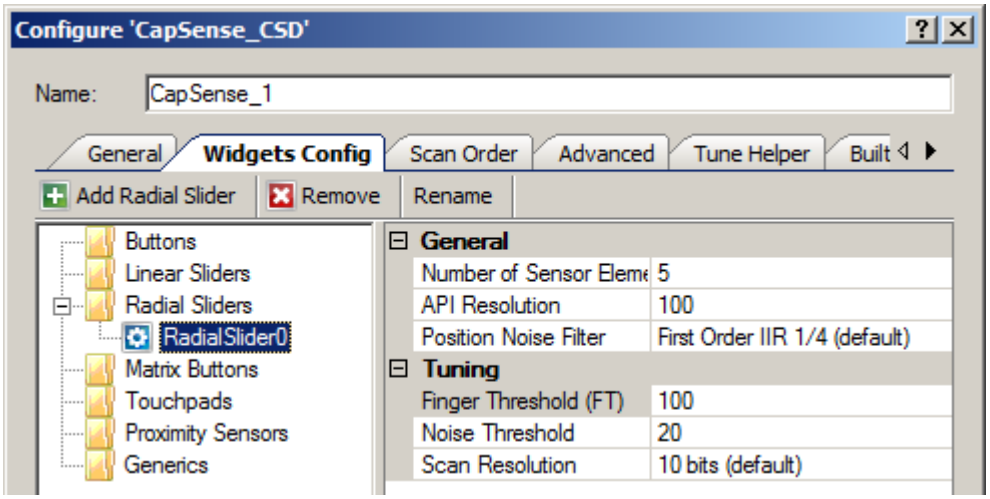
- **Position Noise Filter**（位置噪声滤波器） — 选择噪声滤波器类型以进行位置计算。对于一个选定的 Widget，仅可应用一个滤波器。有关滤波器类型的详情，请参见本文档中功能说明一节的滤波器部分。
  - **None（无）**

- **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（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**。值的有效范围为 [1…255]。
- **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，手指触摸可能会被解释为噪声，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）— 用于定义扫描分辨率。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**。

Radial Slider（辐射滑条）



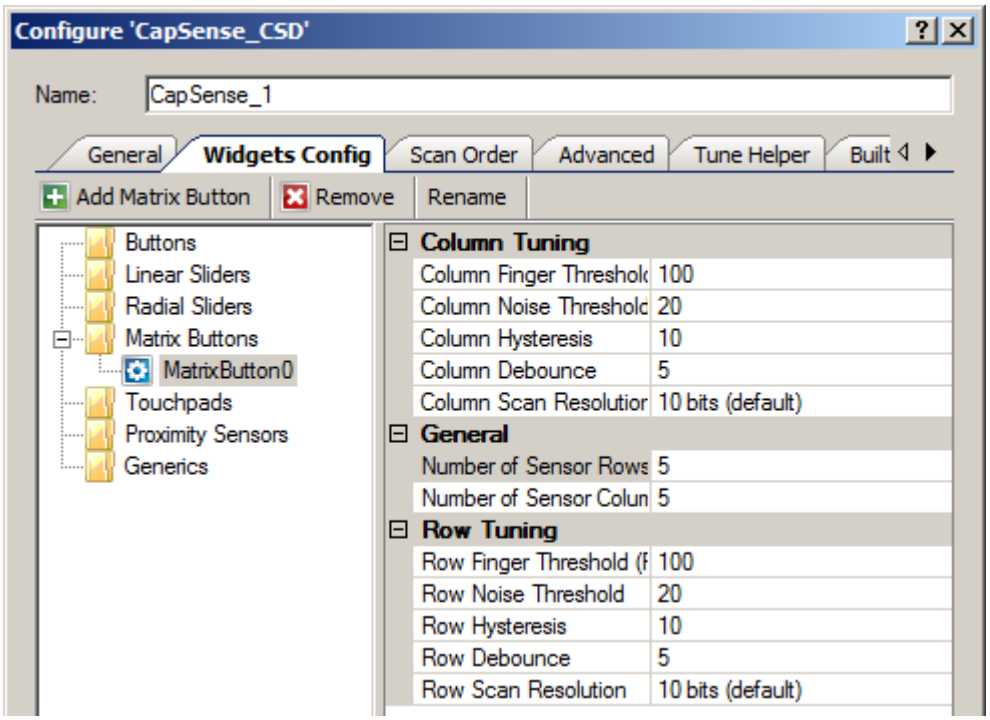
General（通用）：

- **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]。
- **Position Noise Filter**（位置噪声滤波器） — 选择噪声滤波器的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，请参见本数据手册功能说明一节中的滤波器部分。
  - **None（无）**
  - **Median（中值滤波）**
  - **Averaging（均值滤波）**
  - **First Order IIR 1/2（一阶 IIR1/2 滤波）**
  - **First Order IIR 1/4**(default)（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，手指触摸可能会被解释为噪声，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**。

Matrix Buttons（矩阵按键）



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** 值不能大于 254，16 位分辨率 widget 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]。

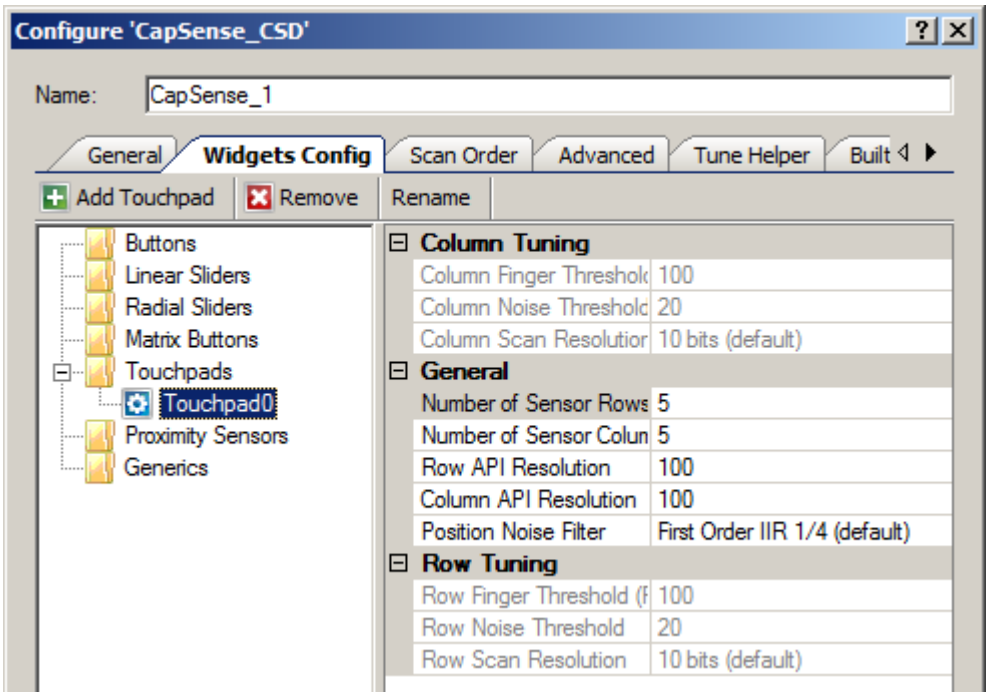
- Column and Row Hysteresis（列/行迟滞） — 添加矩阵按键列和行的传感器活动状态转换的差分迟滞。如果传感器处于非活动状态，则差值计数必须大于手指阈值与迟滞的和才被认为是有效的状态转换条件。如果传感器处于活动状态，则差值计数必须低于手指阈值与迟滞的差才被认为是有效的状态转换条件。迟滞有助于确保低振幅传感器噪声和手指少量移动不会导致按键状态的循环。默认值为 10。8 位分辨率 widget 的有效值范围为[1…255]，16 位分辨率 widget 的有效值范围为[1...65535]。8 位分辨率 widget 的 **Finger Threshold + Hysteresis**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

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**。

Touchpads（触控板）



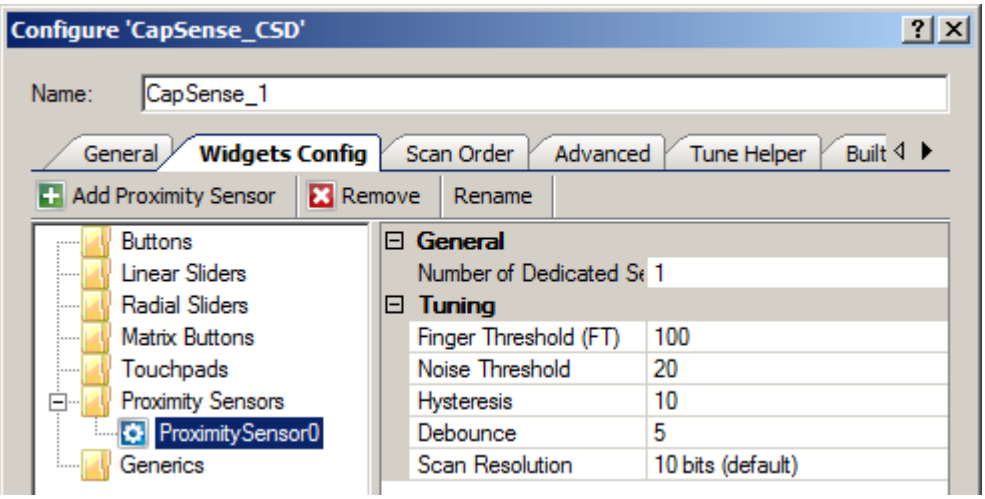
Tuning（调试）：

- **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。这样可造成中心计算错误。默认值为 **20**。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 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**。

General（通用）：

- **Numbers of Sensors Column and Row**(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**。
- **API Resolution Column and Row**（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。取值范围为[1…255]。
- **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 IIR1/2 滤波）**
  - **First Order IIR 1/4**(default)**（一阶 IIR 1/4 滤波，默认）**
  - **Jitter（抖动滤波）**

Proximity Sensors（接近传感器）



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，不符合其他类 Widget 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()*函数以使能接近 widget。

代码示例

```
CapSense_1_EnableWidget(CapSense_1_PROXIMITYSENSOR0__PROX);
```

其中，“CapSense_1”是组件实例名称，“PROXIMITYSENSOR0”是接近 widget 名称。

General（通用）：

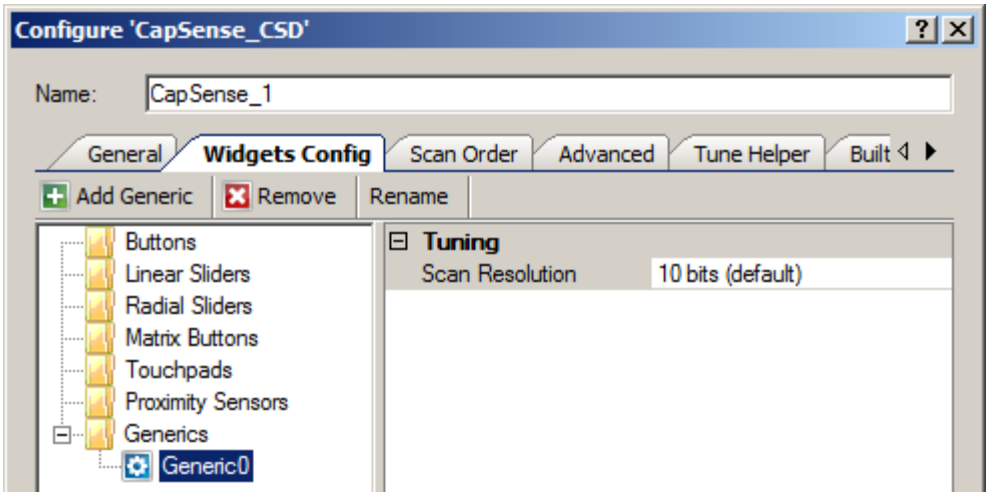
- **Number of Dedicated Sensor Elements**（专用传感器元件数量） — 选择专用接近感应传感器的数量。这些传感器单元不包括用于其它 Widget 类型的传感器。任何 Widget 传感器均可单独使用，或并行连接来实现接近感应传感器。
  - **0** — 接近感应传感器仅扫描一个或多个现有传感器来确定接近情况。没有任何新传感器被分配给该 Widget。
  - **1**（默认） — 系统中专用接近传感器的数量。

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 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，可能忽略传感器和热偏移。这样会导致误触或触摸遗漏。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]。
- **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]。
- **Debounce**（去抖动） — 添加一个去抖动计数器来检测传感器活动的状态转换。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感应的误检。取值范围为[1…255]。
- **Scan Resolution**（扫描分辨率）— 用于定义扫描分辨率。 此参数对接近 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值为 $2^N - 1$。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。我们建议接近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 **10 位**。

Generics（通用）



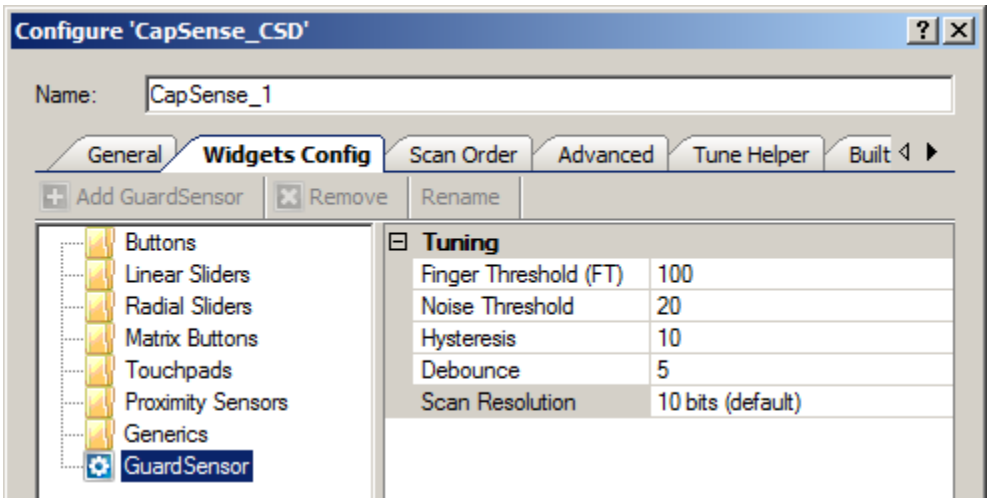
Tuning（调试）：

- **Scan Resolution**（扫描分辨率）— 用于定义扫描分辨率。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 $2^N - 1$。提高分辨率可改善触摸检测的灵敏度和信噪比，但会延长扫描时间。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。

Guard Sensor（保护传感器）

可通过使用 **Advanced**（高级）选项卡添加或删除此特殊传感器。防护传感器不会像其他传感器一样报告指压，但会报告其他 Widget 附近的无效条件来抑制其更新。更多有关此传感器的类型以及应何时适用的信息，请参考本数据手册中功能说明一节的防护传感器实现部分。



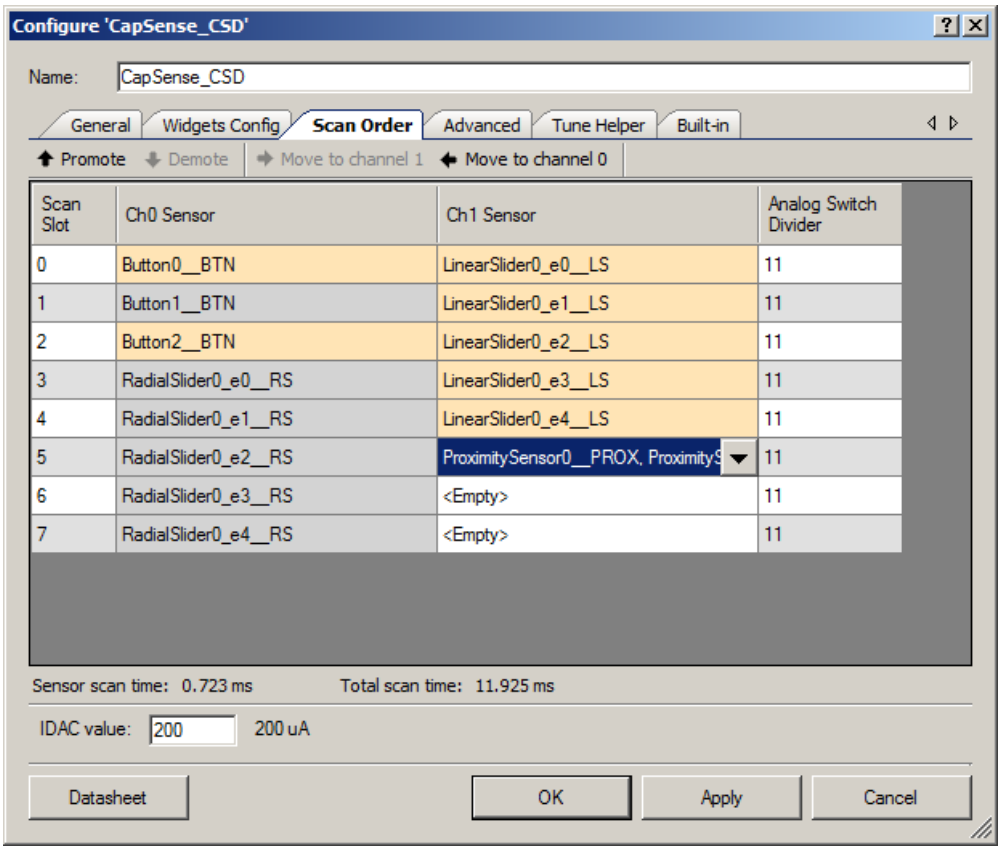
Tuning（调试）：

- **Finger Threshold**（手指阈值） — 定义造成触摸灵敏度增大或减小的传感器活动阈值。 当传感器扫描值大于此阈值时，该防护传感器报告为被触摸。 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** 值不能大于 254，16 位分辨率 widget 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，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 的有效值范围是[1…255]，16 位分辨率 widget 的有效值范围是[1..65535]。

- **迟滞** — 为传感器的活动状态转换添加了差分迟滞。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。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。传感器要想从非活动状态切换到活动状态，差值必须在指定的采样数内保持超过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防护传感器的误检。默认值为 **5**。值的有效范围为[1…255]。
- **Scan Resolution**（扫描分辨率） — 定义扫描分辨率。此参数对防护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。默认值为 **10 位**。

“Scan Order（扫描顺序）”选项卡



Toolbar（工具栏）

该工具栏包含以下指令：

- **Promote/Demote**（热键 — +/-） — 选定的 widget 在数据网格（data grid）内上下移动。如果选定了 Widget 的一个或多个元件，则将选定整个 Widget。
- **Move to Channel 1/Channel 0**（热键为 Shift + 1/0） — 将选中的 widget 移到另一个通道。此选项仅在双通道设计中处于活动状态。 如果选定了 Widget 的一个或多个元素，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，因为它通常不与其他传感器同时扫描。

更多热键

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

Analog Switch Divider(模拟开关分频器)列

指定**模拟开关分频器**的值，并确定扫描槽的预充电开关输出频率。 值的有效范围为[1…255]。默认值为 **11**。

如果 **Analog Switch Drive Source**（模拟开关驱动源）被设为 **Direct**（直接），或 **Multiple Analog Switch Divider**（多个模拟开关分频器）被禁用（位于 **Advanced**（高级）选项卡），则该列被隐藏。

IDAC 值

指定选定传感器的 IDAC 值。 只有将 **IDAC 源**选定为 **Current Source**（电流源）（在 **Advanced**（高级）选项卡下）时，此选项才有效。有效范围介于 0 至 255 之间。默认值为 **200**。

如果将 **Current Source**（电流源）设置为 **External Resistor**（外部电阻），则此参数不可用。

灵敏度

灵敏度是激活传感器所需的 Cs（传感器电容）的标称变化。值的有效范围为[1…100]，对应的灵敏度级别为：0.1、0.2、0.3 和 10 pF。默认值为 **2**。针对丝印层材料的不同厚度，通过 **Sensitivity**（灵敏度）设置传感器的整体灵敏度。越厚的材料应使用越低的灵敏度值。

只有将 **Tuning method**（调试方法）参数设置为 **Auto (SmartSense)**（自动 (SmartSense）时，此选项才可用。

传感器扫描时间

显示典型系统中选定传感器所需的近似扫描时间。

当选择 **Auto (SmartSense)**（自动 （SmartSense））作为调试方法时，显示值可能不准确，因为在调试时参数发生了变化。当 CapSense CSD 组件输入时钟频率未知时，会显示 **Unknown**（未知）。

CapSense CSD 组件的以下参数对传感器的扫描时间产生影响：

- 扫描速度
- 分辨率
- CapSense CSD 时钟

注意：这里显示的扫描时间包括扫描的时间和预计的设置和预处理时间。它不是一个确定的值，因为它取决于设计其他部分、所选编译器、及所选器件（PSoC 3 还是 PSoC 5LP）。

总扫描时间

显示扫描所有传感器所需的总扫描时间。此值为近似传感器扫描时间，因此可以与实际值稍有不同。

当选择 **Auto(SmartSense)**（自动 （SmartSense））作为调试方法时，显示值可能不准确，因为在调试时参数发生了变化。当 CapSense CSD 组件输入时钟频率未知时，会显示 **Unknown**（未知）。

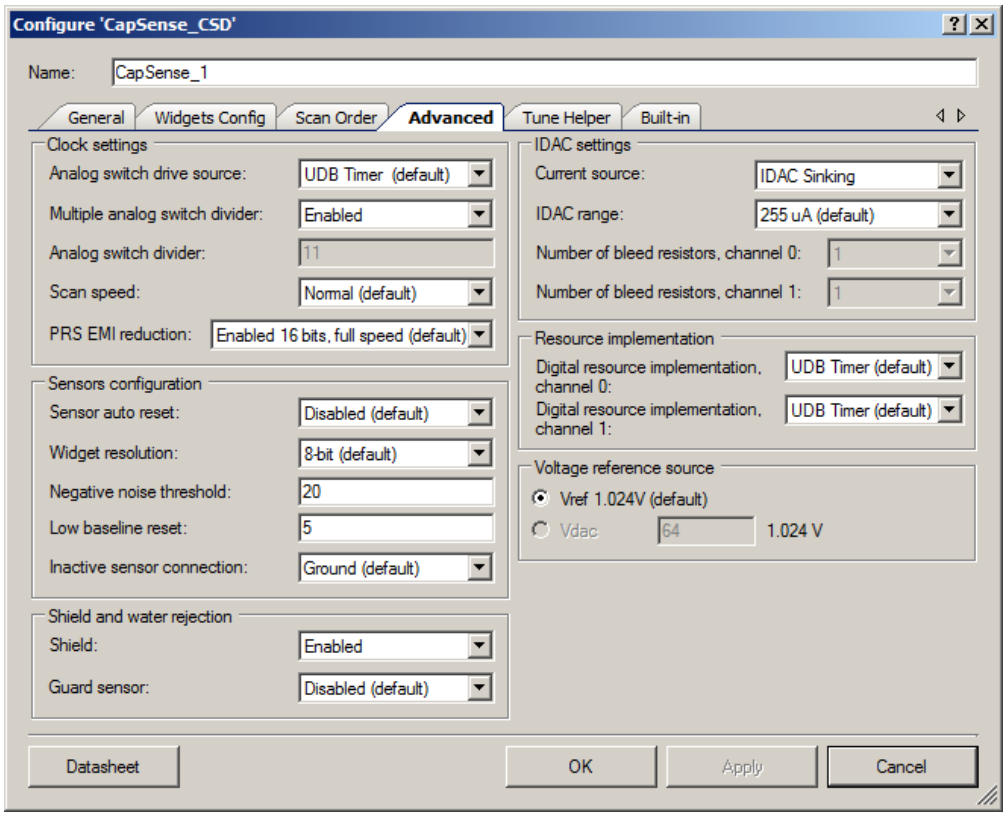
Widget 列表

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。

接近扫描传感器可使用专用接近传感器，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。例如，电路板可能有一根围绕着按键阵列的线路，且接近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。在接近扫描传感器上提供了下拉菜单，可选择进行扫描的一个或多个传感器，以检测接近。

与接近传感器一样，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。从提供的下拉菜单中选择传感器。

Advanced（高级）选项卡



Analog Switch Drive Source（模拟开关驱动源）

此参数指出模拟开关分频器的来源，以确定传感器切换到/出调制电容 C_{MOD} 的速率。实现“固定功能定时器”模块（FF 定时器）中的定时器可以使 UDB 使用的资源最小。

- **Direct**（直接） — 不使用 FF 定时器或 UDB 资源，但会将器件的最大时钟频率限定为模拟开关的频率。在大多数设计中不建议使用。
- **UDB 定时器**（默认） — 使用 UDB 资源
- **FF 定时器** — 不使用 UDB 资源

多模拟开关分频器

该参数确定模拟开关分频器的使用。如果使能，每个扫描槽均使用一个专用的模拟开关分频器值；如果不使能，则各传感器仅使用一个模拟开关分频器值。

如果 **Analog Switch Drive Source**（模拟开关驱动源）被设置为 **Direct**（直接），则此功能不可用。

模拟开关分频器

此参数指定模拟开关分频器的值，并确定预充电开关输出频率。值的有效范围为[1…255]。默认值为 **11**。

如果 **Analog Switch Drive Source**（模拟开关驱动源）被设置为 **Direct**（直接），或 **Multiple Analog Switch Divider**（多模拟开关分频器）被 **Enabled**（使能），则此功能不可用。

传感器以预充电时钟的速度被不断切换至和从调制电容 C_{MOD} 切换。**Analog Switch 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。

扫描速度

此参数指定 CapSense CSD 组件数字逻辑时钟频率，以确定传感器扫描时间。 扫描速度越低，所花时间越长，但所提供的信噪比更高，且具有更好的抗电源和温度变化能力，

- **Slow**（慢速） — 将组件输入时钟除以 16
- **Normal**（正常）（默认） — 将组件输入时钟除以 8
- **Fast**（快速） — 将组件输入时钟除以 4
- **Very Fast**（非常快） — 将组件输入时钟除以 2

表 1. 扫描时间（以 µs 为单位）与扫描速度及分辨率的对比

分辨率（以位为单位）	扫描速度			
	非常快	快速	正常速度	慢速
8	58	80	122	208
9	80	122	208	377
10	122	208	377	718
11	208	377	718	1400
12	377	718	1400	2770
13	718	1400	2770	5500
14	1400	2770	5500	10950
15	2770	5500	10950	21880
16	5500	10950	21880	43720

注意：表 1 的扫描时间基于以下设置估算而来。主设备时钟和 CPU 时钟 = 48 MHz，CapSense CSD 时钟 = 24 MHz，通道数量 = 1。此时间包括传感器设置时间、样品转换间隔和数据处理时

间。通过将提供的值进行线性换算，这些值可用来估算其他时钟频率以及附加传感器的扫描速度。

由于自定义程序对设置和预处理时间作了近似处理，此处显示的值可能与自定义程序扫描时间预估的不同。

PRS 减少电磁干扰（EMI）

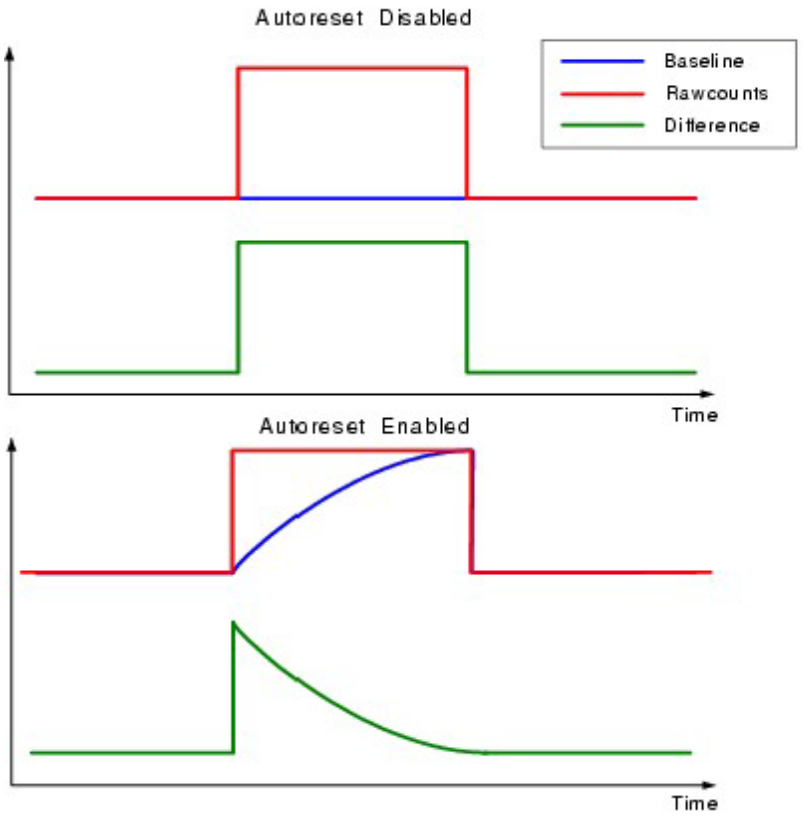
通过该参数指出是否采用 Psuedo Random Sequence（伪随机序列）（PRS）发生器来生成模拟预充电时钟。建议使用 PRS，因为它可以扩大 CapSense 模拟开关频率的频谱，从而减少电磁干扰辐射和灵敏度。PRS 时钟源通过 **Analog Switch Divider**（模拟开关分频器）设置提供。如果没有使能 PRS EMI 降低，则将使用单一频率，从而造成基频和谐波的辐射增加：

- **禁用**
- **使能 8 位** — 使能 8 位可以提供更好的信噪比，但由于它的重复周期比较短，因此会增加电磁干扰。
- **使能 16 位，全速**（默认） — 16 位可以提供更低的信噪比，并且可以使减少电磁干扰的效果更佳
- **使能 16 位，1/4 全速** — 与**使能 16 位，全速相比，**要求一个更快的 4 倍时钟频率，这样才能获得相同的 PRS 时钟输出

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。设计工程师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。值的有效范围为[5…255]。默认值为 20。

Low Baseline Reset（低基准线复位）

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。

Shield（屏蔽电极）

此参数指定使能或禁用屏蔽电极输出，其中屏蔽电极输出用来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 Shield Electrode Use and Restrictions（屏蔽电极使用及限制）一节。

- **禁用**（默认）
- **Enabled（使能）**

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

此参数定义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的噪声。
- **Hi-Z Analog** — 使非活动状态传感器处于高阻抗模式。
- **Shield** — 给所有未扫描传感器提供屏蔽波形. 屏蔽信号的振幅等于 V_{DDIO}。当与屏蔽电极一同使用时，可增强防水能力并降低噪声。如果 **Shield** 被**禁用**，则此功能不可用。

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。更多有关防护传感器的信息，请参考此数据手册“功能说明 ”一节中的“防护传感器实现”。

- **禁用**（默认）
- **使能**

电流源

CapSense CSD 要求高精度的电流源，以检测在传感器上的触摸。**IDAC 灌电流**和 **IDAC 源电流**要求在 PSoC 器件上使用硬件 IDAC。**External Resistor** 使用用户在 PCB 上提供的电阻，而不是 IDAC，它在限制 IDAC 的应用中很有帮助。

- **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，提供电流放电通路。建议在大多数设计中使用**“IDAC Sourcing”**，因为相比其他两种方法，它提供的信噪比最高，但可能要求额外的 VDAC 资源来对其他模式并不要求的 Vref 电平进行设置。

- **IDAC Sinking —** IDAC 从调制电容 C_{MOD} 吸收电流。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。尽管其 SNR（信噪比）通常没有 **IDAC Sourcing** 模式高，但该模式也适用于大多数设计。
- **外部电阻 —** 除 IDAC 由一个接地泄露电阻 Rb 替代外，该模式与 IDAC 灌电流配置的功能相同。泄露电阻在调制电容、C_{MOD} 和 GPIO 之间连接。GPIO 配置为“开漏驱动低电平”的驱动模式，允许 C_{MOD} 通过 Rb 放电。该模式需要的模拟资源最少，仅能在因为资源限制而必需的时候。由于此模式不需要 IDAC 或 VDAC，因此模式会使组件的电源配置最低。 如果功耗是系统关键的考虑因素，它会很有用。

IDAC 范围

此参数指定 **Current Source**（电流源）的 IDAC 范围。如果将 **Current Source** 项设置为 **External Resistor**（外部电阻），则禁用该参数。默认值是几乎所有 CapSense 设计的最佳选择。 较低和较高的电流范围通常仅用于非触控电容式传感器。

- **32 uA**
- **255 uA**（默认）
- **2.04 mA**

泄露电阻的数量，通道 0/通道 1

此参数指定泄露电阻的数量。泄露电阻的最大数量为每个通道三个。如果 **Current Source** 被设置为 **IDAC 源电流**或 **IDAC 灌电流**，则此功能不可用 。支持多个泄露电阻，以允许协助系统调试的多达三组传感器的不同电流。大多数具有相似传感器大小的设计仅要求一个泄露电阻。

数字资源实现，通道 0/通道 1

此参数指定用于实现 CapSense（包括一个定时器和一个计数器）数字部分的资源类型。对于大多数设计，不应改变此参数，因为它旨在提供最大的实现灵活性。

- **UDB 定时器（默认） —** 大多数可以灵活实现，但使用有用的 UDB 资源
- **FF 定时器 —** 通过 FF 定时器实现释放 UDB 资源，但不支持 **扫描速度**为**非常快的情况**。

电压参考源

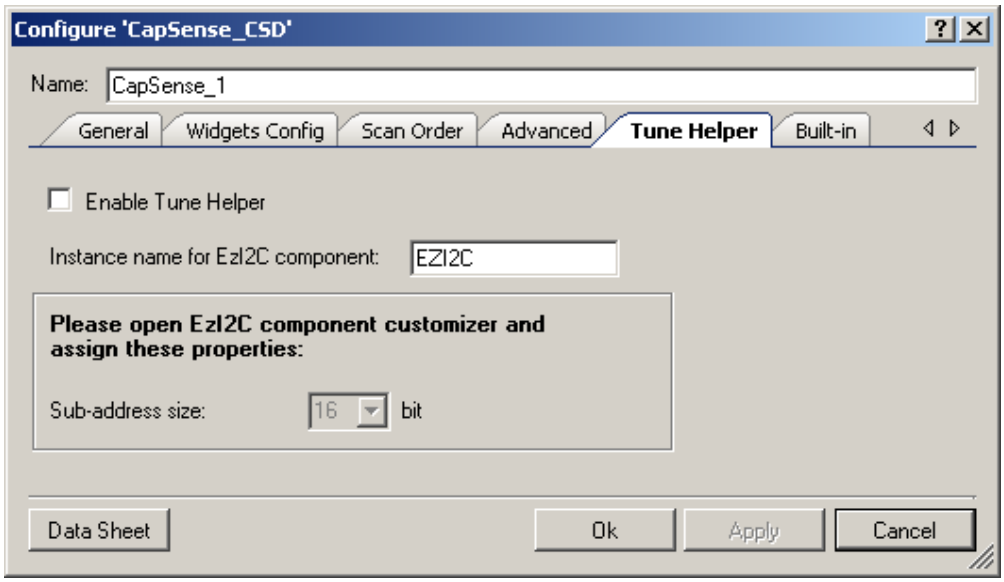
此参数指定基准源电压的类型和水平。最好有与 **IDAC 源电流** 模式一样高的基准电压，以及与 **IDAC 灌电流**或**外部电阻电流源**模式一样低的基准电压。

- **Vref 1.024V**（默认） — IDAC 灌电流模式的最佳选择

- **Vdac** — IDAC 源电流模式的最佳选择。允许使用电压 DAC 调整基准电压，以最大化可用范围。仅在 **Current Source** 被设置为 **IDAC 电流源**，并且需要 VDAC 器件资源时，基准源 VDAC 才可用。基准电压增加，灵敏度也会增加，但对屏蔽电极的影响会减少。值的有效范围为[0…255]。默认值为 64。

当选定 VDAC 时，则不使用 CapSense 缓冲器，因为其设计用于低电压。这会导致 C_{MOD} 在启动时从 VDAC 充电至 Vref。将 C_{MOD} 充电至 Vref 所需的时间量可能导致基准线初始化失败。通常，再次进行准线初始化即可解决此问题。

“Tune Helper”选项卡



Enable Tune Helper（使能调谐助手）

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。默认被禁用。

Instance name for EZI2C component（EZI2C 组件的实例名称）

此参数定义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。

注意：没有实时设计规则检查来确保实际实例名称与此处输入的实例名称相匹配。您必须确定它们是匹配的。如果名称不匹配，则会因为 API 名称错误而在项目构建过程中生成构建错误。

有关如何使用调谐器 GUI 的更多信息，请参考此数据手册中的调谐器 GUI 用户指南一节。

调谐器 GUI 用户指南

本节提供相关指令和信息，能帮助用户使用 CapSense 调谐器。

电容式感应（CapSense® CSD） 1
特性 1
概述 1
何时使用 CapSense 组件 1
输入/输出接口 2
时钟 — 输入* 2
屏蔽 — 输出 * 2
vref — 输出 * 2
组件参数 3
General（常规）选项卡 3
Load Settings/Save Settings（加载设置/保存设置 ） 3
Tuning method （调试方法） 3
通道数量 4
原始数据（Raw Data）噪声滤波器 5
Water proofing and detection（防水及检测） 5
使能时钟输入 5
扫描时钟 6
Widget Config（Widget 配置）选项卡 6
工具栏 7
Buttons（按键） 7
Linear Sliders（线性滑条） 9
Radial Slider（辐射滑条） 10
Matrix Buttons（矩阵按键） 12
Touchpads（触控板） 13
Proximity Sensors（接近传感器） 15
Generics（通用） 16
Guard Sensor（保护传感器） 17

“Scan Order（扫描顺序）”选项卡 18
Toolbar（工具栏） 19
更多热键 19
Analog Switch Divider(模拟开关分频器)列 19
IDAC 值 19
灵敏度 19
传感器扫描时间 20
总扫描时间 20
Widget 列表 20
Advanced（高级）选项卡 21
Analog Switch Drive Source（模拟开关驱动源） 21
多模拟开关分频器 21
模拟开关分频器 22
扫描速度 22
PRS 减少电磁干扰（EMI） 23
Sensor Auto Reset（传感器自动复位 ） 23
Widget Resolution （Widget 分辨率） 24
Negative Noise Threshold（负噪声阈值） 24
Low Baseline Reset（低基准线复位） 24
Shield（屏蔽电极） 25
Inactive Sensor Connection（非活动状态传感器连接） 25
Guard Sensor（保护传感器） 25
电流源 25
IDAC 范围 26
泄露电阻的数量，通道 0/通道 1 26
数字资源实现，通道 0/通道 1 26
电压参考源 26
“Tune Helper”选项卡 27
Enable Tune Helper（使能调谐助手） 27

Instance name for EZI2C component（EZI2C 组件的实例名称） 27
调谐器 GUI 用户指南 28
CapSense 调试过程 39
在 PSoC Creator 中创建设计 39
放置并配置 EZI2C 组件 39
放置并配置 CapSense 组件 40
选择“Auto (SmartSense)” 41
配置用户 CapSense 组件 42
添加代码 43
构建设计并编程 PSoC 器件 43
启动调谐器应用 43
配置通信参数 44
开始调试 45
编辑 CapSense 参数值 45
根据需要重复各步骤 45
关闭调谐器应用 45
CapSense 验证过程 46
开始验证 46
激励传感器 47
验证显示 48
验证结果 49
手动调试过程 50
调谐器 GUI 界面 53
通用界面 53
调试选项卡 54
Graphing Tab（图解选项卡） 56
Validation Tab（验证选项卡） 57
选项卡 58
验证高级属性 59

Save and Load Settings（保存/加载设置）功能 60
应用编程接口 61
通用 API 62
void CapSense_Start(void) 62
说明： 62
参数： 62
返回值： 62
其他影响 62
void CapSense_Stop(void) 62
说明： 62
参数： 62
返回值： 62
其他影响： 62
void CapSense_Sleep(void) 63
说明： 63
参数： 63
返回值： 63
其他影响： 63
void CapSense_Wakeup(void) 63
说明： 63
参数： 63
返回值： 63
其他影响： 63
void CapSense_Init(void) 63
说明： 63
参数： 63
返回值： 63
其他影响： 63
void CapSense_Enable(void) 63

说明： 63
参数： 63
返回值： 63
其他影响： 63
void CapSense_SaveConfig(void) 64
说明： 64
参数： 64
返回值： 64
其他影响： 64
void CapSense_RestoreConfig(void) 64
说明： 64
参数： 64
返回值： 64
其他影响： 64
扫描特定的 API 64
void CapSense_ScanSensor(uint8 sensor) 65
说明： 65
参数： 65
返回值： 65
其他影响： 65
void CapSense_ScanEnabledWidgets(void) 65
说明： 65
参数： 65
返回值： 65
其他影响： 65
uint8 CapSense_IsBusy (void) 65
说明： 65
参数： 65
返回值： 65

其他影响： 65
void CapSense_SetScanSlotSettings(uint8 slot) 65
说明： 65
参数： 65
返回值： 65
其他影响： 65
void CapSense_ClearSensors(void) 66
说明： 66
参数： 66
返回值： 66
其他影响： 66
void CapSense_EnableSensor(uint8 sensor) 66
说明： 66
参数： 66
返回值： 66
其他影响： 66
void CapSense_DisableSensor(uint8 sensor) 66
说明： 66
参数： 66
返回值： 66
其他影响： 66
uint16 CapSense_ReadSensorRaw(uint8 sensor) 66
说明： 66
参数： 66
返回值： 66
其他影响： 66
void CapSense_SetRBleed(uint8 rbleed) 67
说明： 67
参数： 67

返回值： 67
其他影响： 67
高级 API 67
void CapSense_InitializeSensorBaseline(uint8 sensor) 68
说明： 68
参数： 68
返回值： 68
其他影响： 68
void CapSense_InitializeEnabledBaselines(void) 68
说明： 68
参数： 68
返回值： 68
其他影响： 68
void CapSense_InitializeAllBaselines(void) 68
说明： 68
参数： 68
返回值： 68
其他影响： 68
void CapSense_UpdateSensorBaseline(uint8 sensor) 69
说明： 69
参数： 69
返回值： 69
其他影响： 69
void CapSense_UpdateEnabledBaselines(void) 69
说明： 69
参数： 69
返回值： 69
其他影响： 69
void CapSense_EnableWidget(uint8 widget) 69

说明：.....69
参数：.....69
返回值：.....69
其他影响：.....69
void CapSense_DisableWidget(uint8 widget).....70
说明：.....70
参数：.....70
返回值：.....70
其他影响：.....70
uint8 CapSense_CheckIsWidgetActive(uint8 widget).....70
说明：.....70
参数：.....70
返回值：.....70
其他影响：.....70
uint8 CapSense_CheckIsAnyWidgetActive(void).....70
说明：.....70
参数：.....70
返回值：.....70
其他影响：.....70
uint16 CapSense_GetCentroidPos(uint8 widget).....71
说明：.....71
参数：.....71
返回值：.....71
其他影响：.....71
uint16 CapSense_GetRadialCentroidPos(uint8 widget).....71
说明：.....71
参数：.....71
返回值：.....71
其他影响：.....71

uint8 CapSense_GetTouchCentroidPos(uint8 widget, uint16* pos) 72
说明： 72
参数： 72
返回值： 72
其他影响： 72
uint8 CapSense_GetMatrixButtonPos(uint8 widget, uint8* pos) 72
说明： 72
参数： 72
返回值： 72
其他影响： 72
调谐器助手 API 73
void CapSense_TunerStart(void) 73
说明： 73
参数： 73
返回值： 73
其他影响： 73
void CapSense_TunerComm(void) 73
说明： 73
参数： 73
返回值： 73
其他影响： 73
引脚 API 74
void CapSense_SetAllSensorsDriveMode(uint8 mode) 74
说明： 74
参数： 74
返回值： 74
其他影响： 74
void CapSense_SetAllCmodsDriveMode(uint8 mode) 74
说明： 74

参数：.....74
返回值：.....74
其他影响：.....74
void CapSense_SetAllRbsDriveMode(uint8 mode).....75
说明：.....75
参数：.....75
返回值：.....75
其他影响：.....75
数据结构.....75
CapSense_sensorRaw [ ].....75
CapSense_sensorEnableMask[ ].....76
CapSense_portTable[ ]和 CapSense_maskTable[ ].....76
CapSense_sensorBaselineLow[ ].....76
CapSense_sensorBaseline[ ].....76
CapSense_sensorSignal[ ].....77
CapSense_sensorOnMask[ ].....77
常量.....77
传感器常量.....78
Widget 常量.....79
MISRA 合规性.....79
示例固件源代码.....80
引脚分配.....80
侧面.....80
传感器引脚 — CapSense_cPort — 引脚分配.....81
CapSense_cCmod_Port — 引脚分配.....81
CapSense_cRb_Ports —引脚分配.....82
中断服务子程序.....82
双通道模式 ISR 优先级设置.....82
功能说明.....83
定义.....83

传感器 83
扫描时间 83
CapSense Widget 83
手指阈值 83
噪声阈值 83
去抖动 83
迟滞 83
API 分辨率 — 插值和缩放比例 84
双工 85
滤波器 87
中值滤波器 87
均值滤波器 87
一阶 IIR 滤波器 87
抖动滤波器 88
水对 CapSense 系统的影响 88
防水和检测 88
屏蔽电极 88
Shield Electrode Use and Restrictions（屏蔽电极使用及限制） 89
当前模式 IDAC 源 89
当前模式 IDAC 灌电流能力和外部电阻 91
防护传感器实现 91
框图和配置 92
IDAC 源电流 92
IDAC 灌电流 93
IDAC 禁用，使用外部 Rb 93
资源 94
数字资源： 94
模拟资源： 95
API 存储器占用情况 95

直流和交流电的电气特性 96
电源电压.....96
噪音.....96
功耗.....97
图 — 原始计数与电源对比 97
组件勘误表.....100
组件更改 100

CapSense 调谐器在手动调校模式下，根据特定的系统环境帮助调节 CapSense 组件。它还能够在组件处于 SmartSense 模式下显示调试值（只读）和性能。当组件处于非调试模式时不支持调试，因为所有参数均存储在闪存中，并且为实现最低 SRAM 使用率，这些参数处于只读状态。

CapSense 调试过程

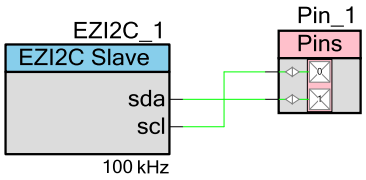
使用和调试 CapSense 组件的典型过程如下：

在 PSoC Creator 中创建设计

若有需要，可以参考 PSoC Creator 帮助。

放置并配置 EZI2C 组件

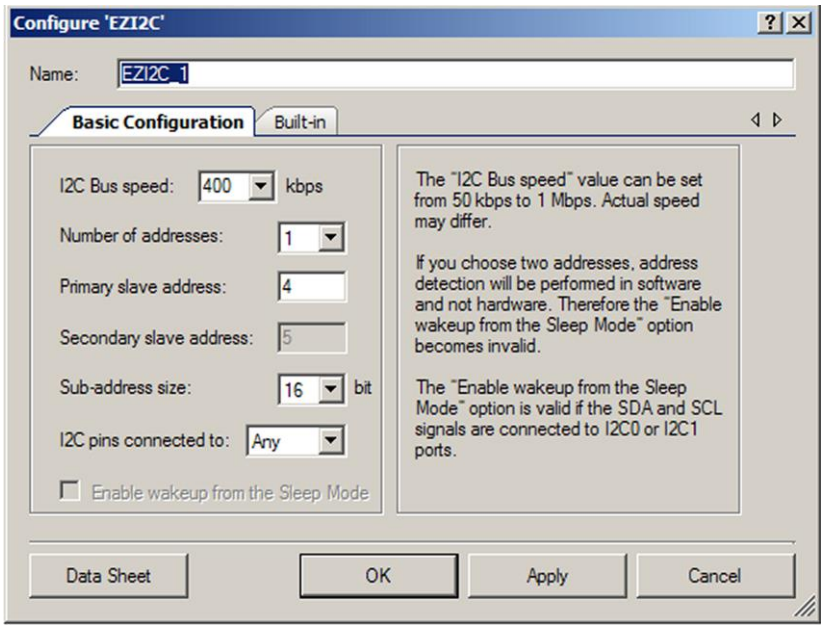
将 EZI2C 组件从组件目录拖动到您的设计中。



双击可打开 **Configure**（配置）对话框。

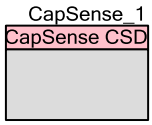
更改如下参数，然后单击 **OK**（确定）关闭对话框。

- 辅助地址大小必须为 16 位。
- 实例名称必须匹配用于 **CapSense CSD 配置**对话框在**调试助手**选项卡下的名称，这样生成的 API 才能发挥功能。



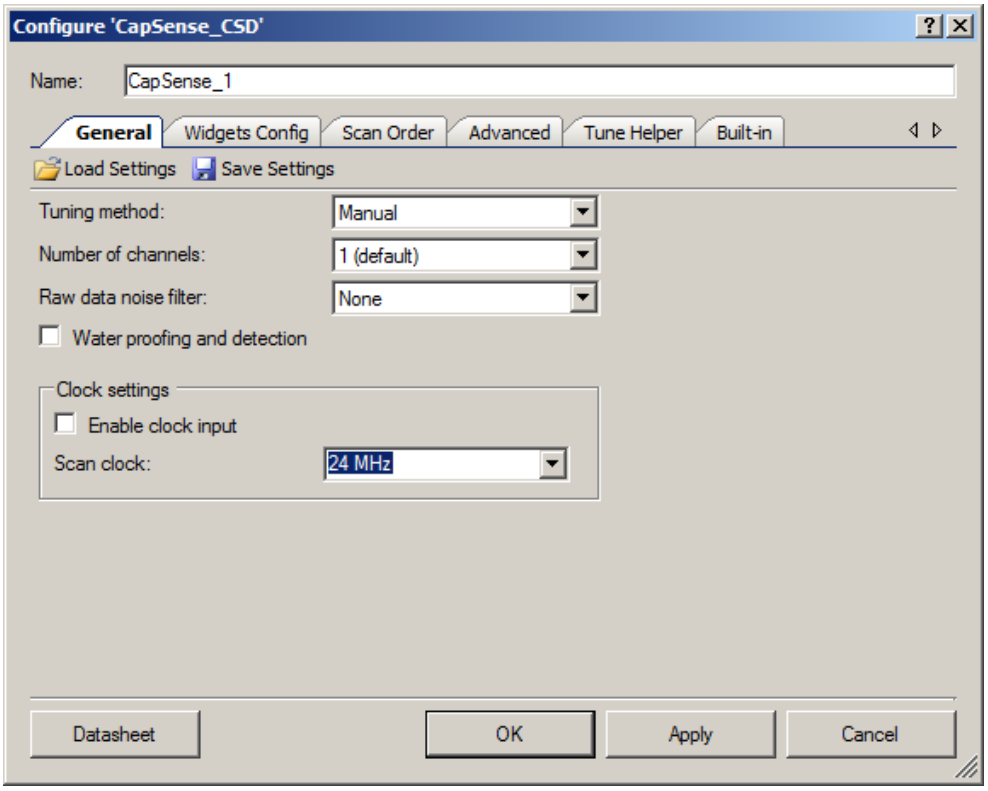
放置并配置 CapSense 组件

1. 将 CapSense_CSD 组件从组件目录拖动到您的设计中。



双击它以打开 **Configure** 对话框。

根据您应用的需要更改 CapSense CSD 参数。将 **Tuning method**（调试方法）选择为 **Manual** 或 **Auto (SmartSense)**。单击 **OK** 关闭对话框并保存所选的参数。



选择“Auto (SmartSense)”

自动（SmartSense）允许用户根据特定的系统具体情况自动调试 CapSense CSD 组件。在运行时，通过固件计算 CapSense CSD 参数。该模式中使用了额外的 RAM 和 CPU 时间。自动（SmartSense）消除了手动调试 CapSense CSD 组件参数容易产生错误和重复过程的问题，确保系统正确工作。选择自动(SmartSense)将调试以下 CSD 参数：

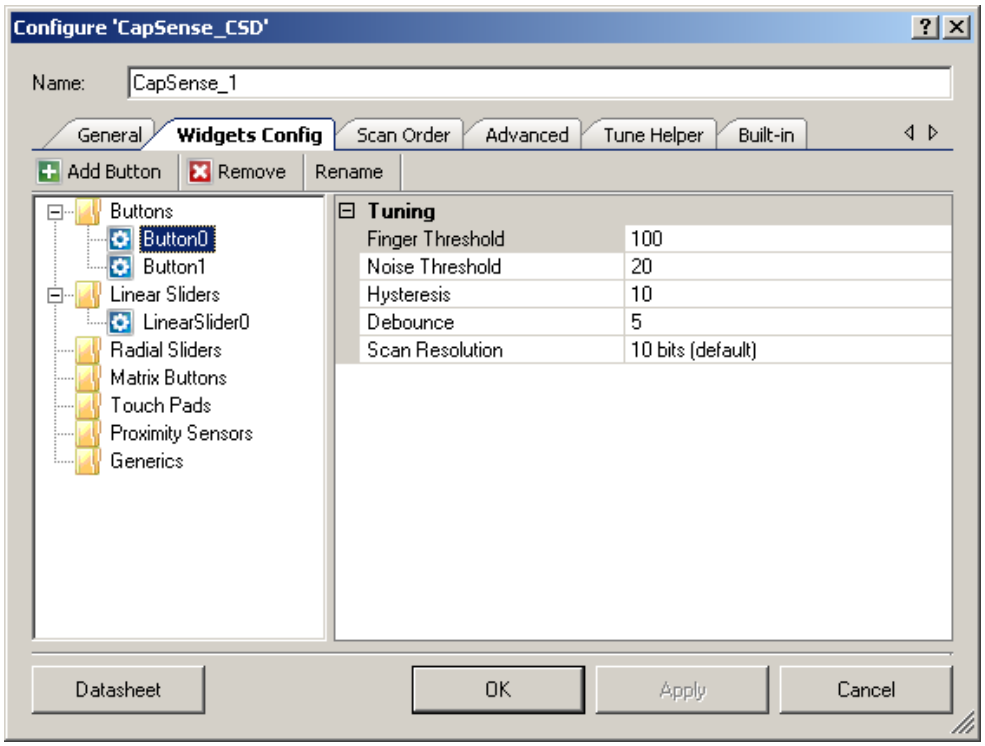
参数	计算
手指阈值	在传感器扫描期间连续计算。
噪声阈值	在传感器扫描期间连续计算。
IDAC值	在CapSense CSD启动时计算一次。
模拟开关分频器	在CapSense CSD启动时计算一次。
扫描分辨率	在CapSense CSD启动时计算一次。

以下是自动（SmartSense）调试方法的硬件参数限制：

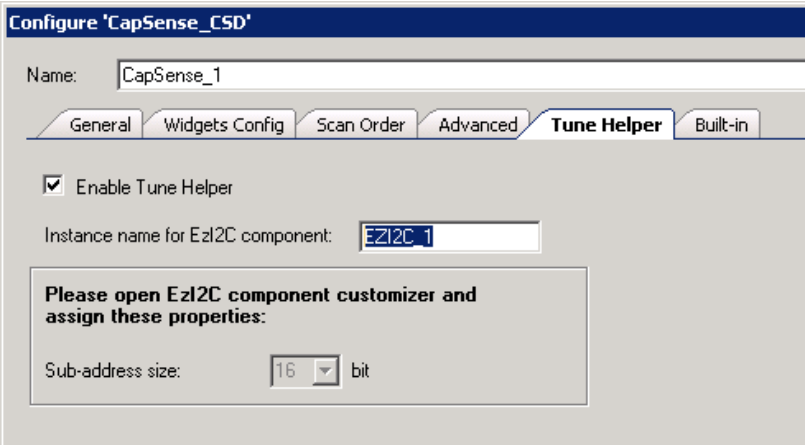
参数	所需的设置
扫描时钟	时钟必须为内部时钟（**常规**选项卡下的使能**时钟输入**被清除）。
电流源	IDAC源/IDAC灌电流。
PRS减少电磁干扰（EMI）	使能16位。
扫描速度	正常

配置用户 CapSense 组件

1. 在 **Widgets Config** 选项卡上添加并配置“widgets”。



在 **Tune Helper**（调试助手）选项卡中，必须输入 EzI2C 组件实例名称，且必须选中 **Enable Tune Helper**（使能调试助手）复选框。



添加代码

- 将调试器初始化和通信代码添加到项目 *main.c* 文件中。*main.c* 文件示例：

```
 void  main()
{
    CYGlobalIntEnable;
    CapSense_1_TunerStart();

    /*All widgets are enabled by default except proximity widgets. Proximity
    widgets must be manually enabled as their long scan time is incompatible
    with the fast response required of other widget types.
    */

    while(1)
    {
        CapSense_1_TunerComm();
    }
}
```

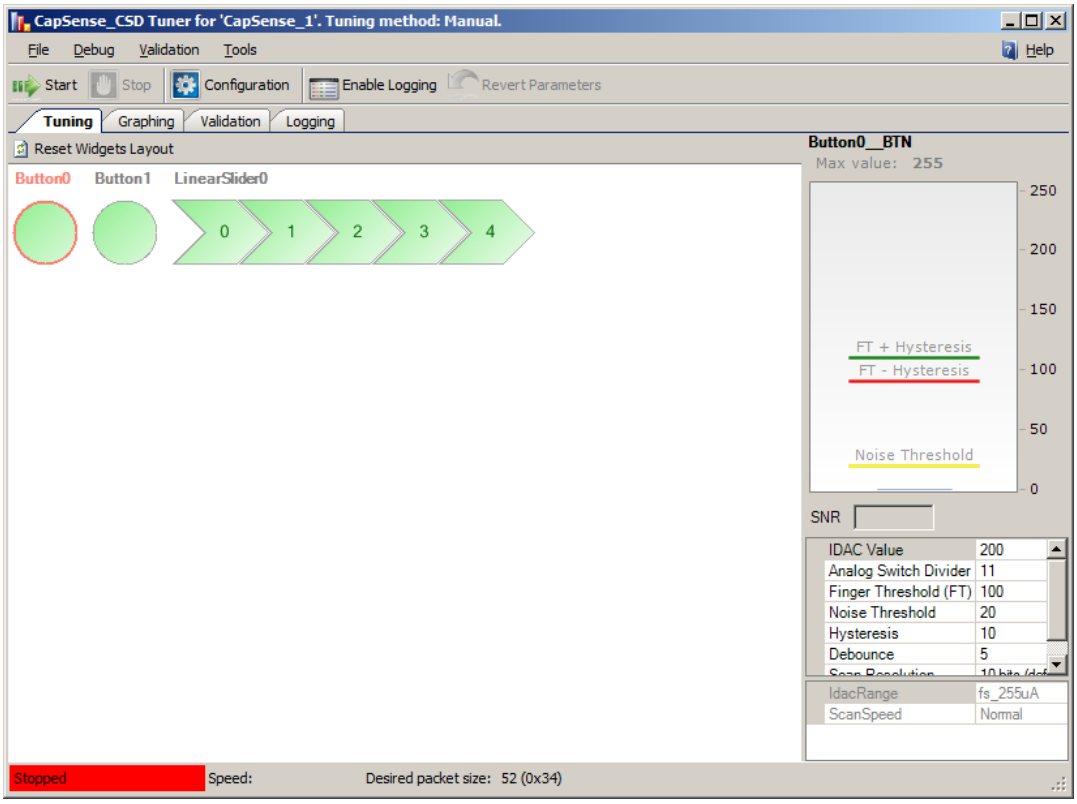
构建设计并编程 PSoC 器件

若需要，可以参考 PSoC Creator 帮助。

启动调谐器应用

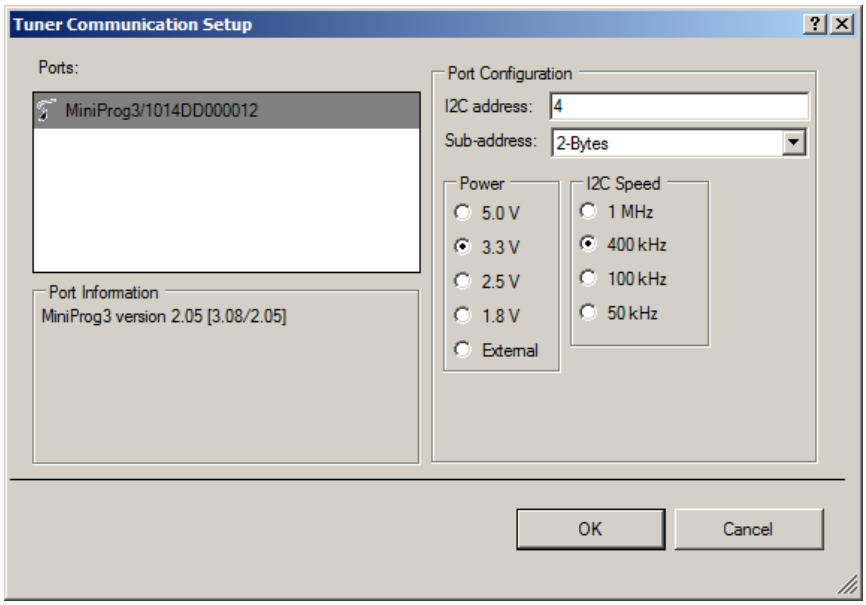
- 右击 CapSense CSD 组件图标，然后从上下文菜单中选择 **Launch Tuner** 项。

  调谐器应用将打开。



配置通信参数

1. 点击 **Configuration**，以打开 **Tuner Communication** 对话框。



2. 设置通信参数，然后按 **OK**。

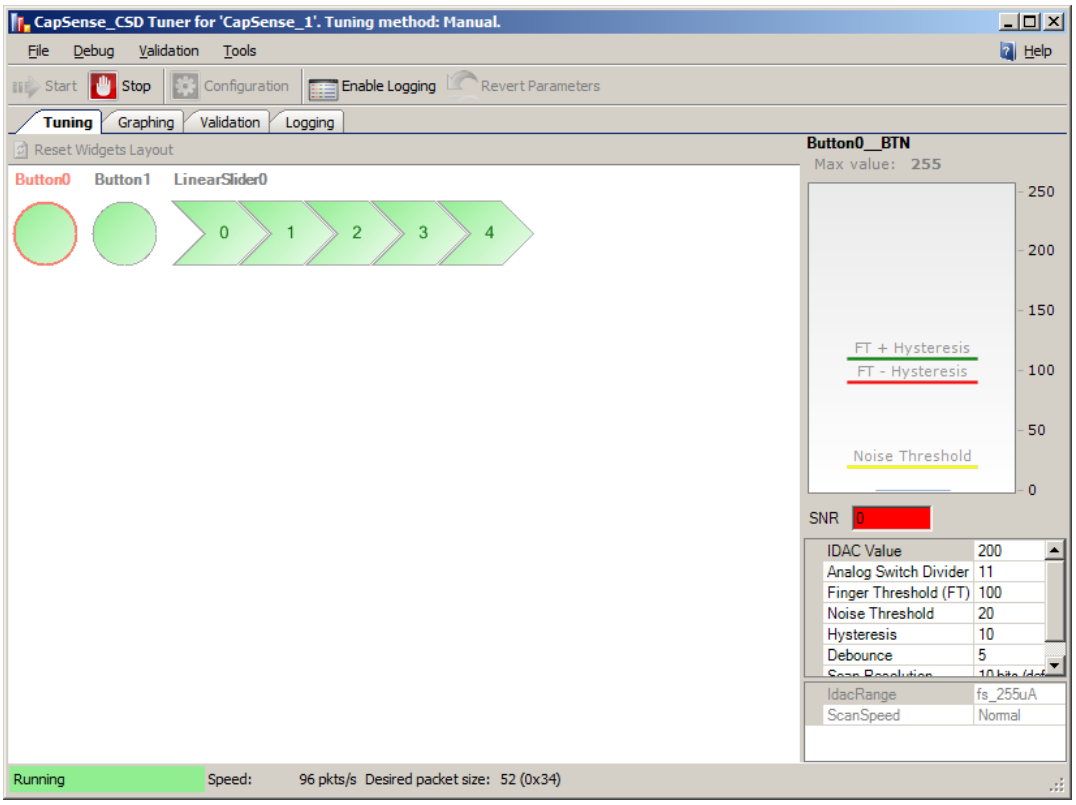
重要提示：属性必须与 EZ I2C 组件中的属性完全一致：**I2C 总线快速**、**I2C 地址**和**辅助地址大小** = 2 字节。

开始调试

- 点击调试 GUI 上的 **Start**。 所有 CapSense 元件开始显示它们的值。

编辑 CapSense 参数值

- 编辑其中一个元件的参数值，按**[Enter]**键或转入另一个选项后，参数值会自动应用。GUI 继续显示扫描数据，但现在根据更新参数的应用发生了改变。请参考本数据手册后面的调谐器 GUI 界面一节。



根据需要重复各步骤

- 根据需要重复上述各步骤，直到调试完成并 CapSense 组件提供可靠的触控传感器结果为止。

关闭调谐器应用

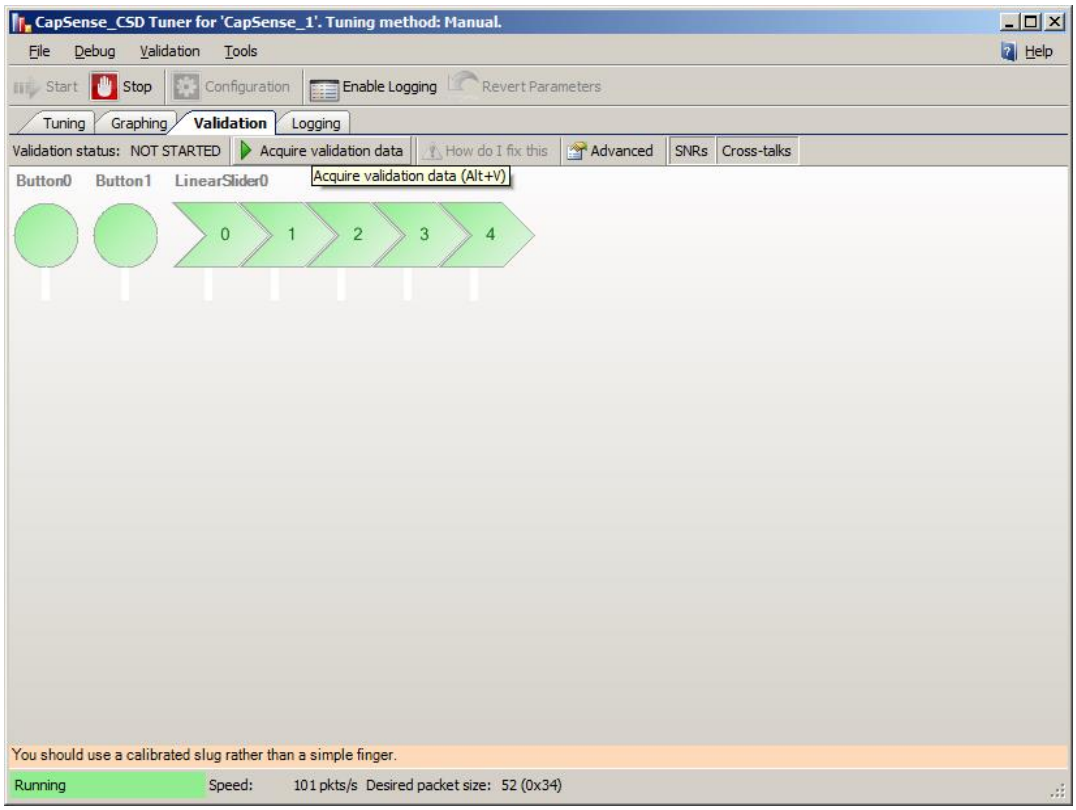
- 单击 **OK**，参数被写回 CapSense_CSD 实例。调谐器应用对话关闭。

CapSense 验证过程

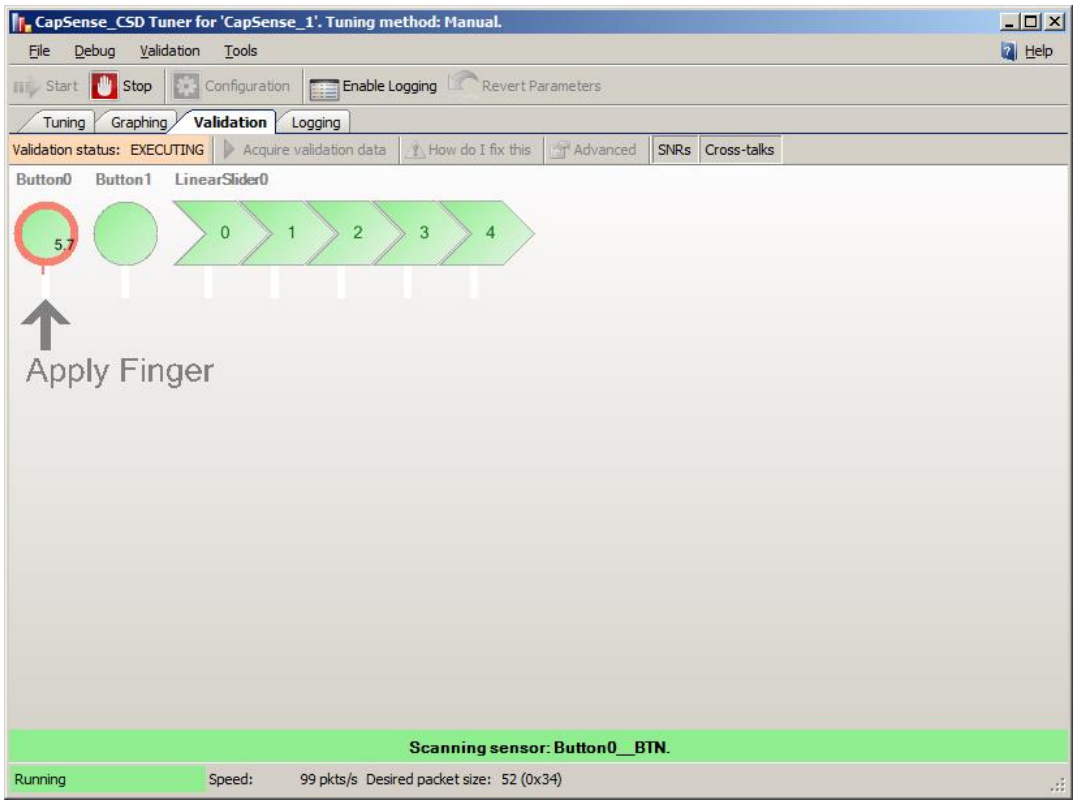
验证机制确定电路板是否已经过充分调试。下面介绍的是使用调试器验证特性来验证 CapSense 设计的典型过程。

开始验证

1. CapSense 调试过程开始扫描前，必须准备好调谐器和硬件。请参见前面章节，准备系统进行扫描的有关内容。
2. 在 **Validation**（验证）选项卡上，点击 **Acquire Validation Data**（获取验证数据）项。开始出现所有 CapSense 元件对应的值。



激励传感器

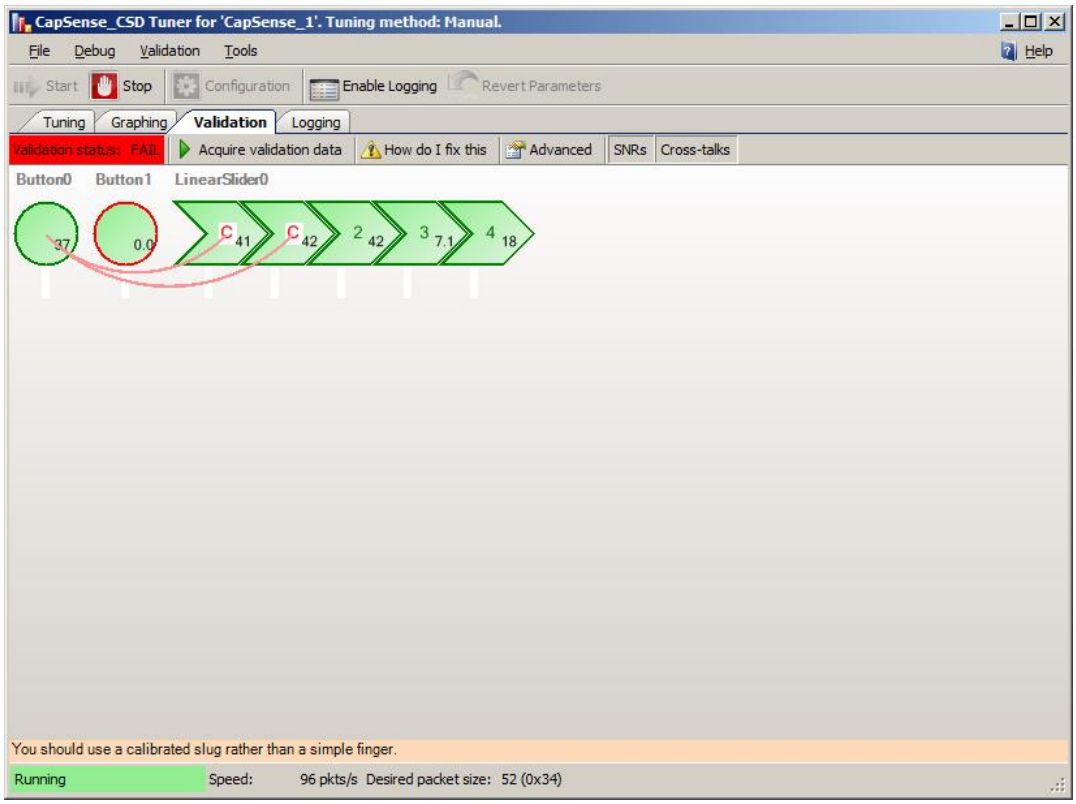


会有信息提示您在各传感器上用一个手指触摸。每次提示您按压 CapSense 元件时，布局图上会有闪烁的红色箭头指向目标，并有文字提示 **PRESS HERE**（按此处）。Tuner 下方出现的文字将指导用户完成验证过程。

- 要开始当前传感器的扫描，可按键盘上任意键。

建议使用经过校准的仿真手指，而不是通过手指按压来激励传感器。

验证显示



信噪比警告出现如下：

- **闪烁红色**高亮区围绕所有信噪比小于 **Sufficient Value**(足够值)的 CapSense 传感器。
- **闪烁黄色**高亮区围绕信噪比介于 **Sufficient**（足够值）和 **Optimal Values**（最优值）之间的任何 CapSense 传感器。
- 闪烁绿色**高亮区围绕的 CapSense Sensor 信噪比**高于 Optimal Values（最优值）。

串扰效应警告出现如下：

- **Individual Crosstalk Check**（单个串扰检查）。在验证过程中，软件监视除用户被告知需激励的元件以外的所有元件。如果某一元件显示不同的计数，且超出 **Crosstalk Threshold Percentage**（串扰阈值百分比）（未直接激励时），则将产生串扰警告。这通过一条闪烁的线来显示，这条线介于显示出非预期计数的元件和受到激励的元件之间。
- **Worst Case Crosstalk Check**（最差情形串扰检查） 进行每个单独串扰检查时，软件将记录着各不同计数测量值。在这个过程完成时，对最差情形串扰检查进行了估算。

  对各传感器，出现一个合计，它是等于 **Worst Case Crosstalk Sensor Count**（最差情形串扰传感器计数）的串扰效应的数量。最大的串扰值是合计中第一个元件，第二大是第二个，以此类推。例如，如果得到以下串扰计数（1、5、3、2、4、1、1、0），而 **Worst Case**

Crosstalk Sensor Count 是 2，则 **Worst Case Crosstalk** 计算结果将为（5 + 4 = 9）。如果该值超过 **Worst Case Crosstalk Threshold**，则会在所显示传感器中间用**闪烁的字符 “C”** 对其进行标记。

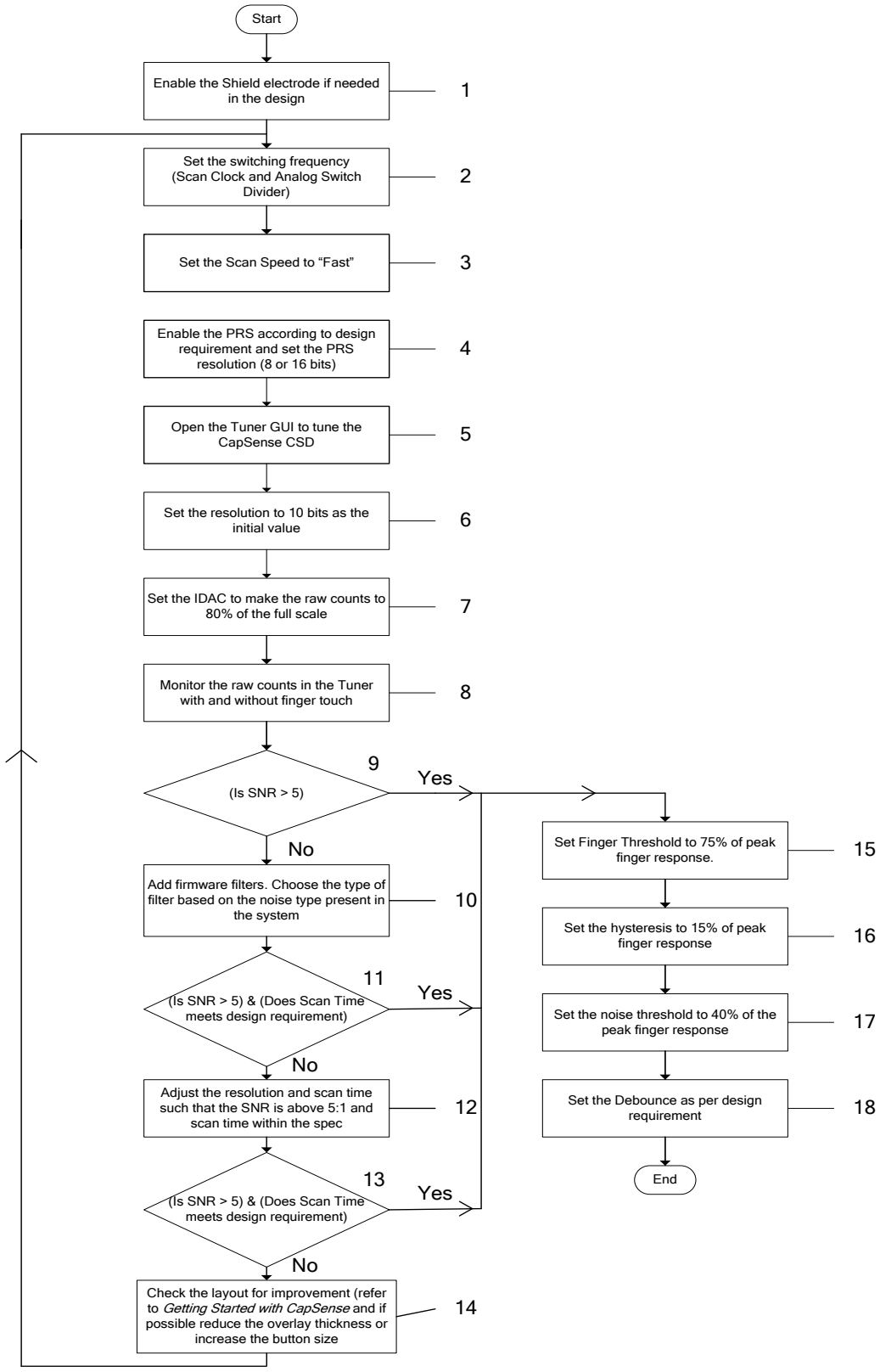
验证结果

如果验证过程发现了故障，则将显示 **Validation Report**（验证报告）。该报告包括以下信息：

- 低于 **Optimal Value**（最优值）的任何信噪比值
- 低于 **Sufficient Value**（充分值）的任何信噪比值
- 出现最差情形串扰故障的任何信号，并出现这种故障的串扰个数

可以通过点击 **Validation** （验证）选项卡上的 **How do I fix this** 按键，打开 Validation Report（验证报告）。

手动调试过程


Start
Enable the Shield electrode if needed in the design
1
Set the switching frequency (Scan Clock and Analog Switch Divider)
2
Set the Scan Speed to “Fast”
3
Enable the PRS according to design requirement and set the PRS resolution (8 or 16 bits)
4
Open the Tuner GUI to tune the CapSense CSD
5
Set the resolution to 10 bits as the initial value
6
Set the IDAC to make the raw counts to 80% of the full scale
7
Monitor the raw counts in the Tuner with and without finger touch
8
9
(Is SNR > 5)
Yes
No
Add firmware filters. Choose the type of filter based on the noise type present in the system
10
11
(Is SNR > 5) & (Does Scan Time meets design requirement)
Yes
No
Adjust the resolution and scan time such that the SNR is above 5:1 and scan time within the spec
12
13
(Is SNR > 5) & (Does Scan Time meets design requirement)
Yes
No
Check the layout for improvement (refer to Getting Started with CapSense and if possible reduce the overlay thickness or increase the button size
14
Set Finger Threshold to 75% of peak finger response.
15
Set the hysteresis to 15% of peak finger response
16
Set the noise threshold to 40% of the peak finger response
17
Set the Debounce as per design requirement
18
End

1. 屏蔽的使能或禁用依设计要求而定。在传感器覆盖层可能变湿（见 AN2398）的应用中，屏蔽电极可用于屏蔽 EMI，或降低过高的 Cp 值。如果当前模式被设为 **IDAC Sourcing** 、且基准电压为 **Vref 1.024V**，则屏蔽信号必须连接到 SIO 引脚。其他情况屏蔽可连接到任何引脚。更多信息，请参见本文档后面的屏蔽电极一节。
2. 各个电容式传感器的开关频率设置应能使传感器满充满放。**Scan Clock**（扫描时钟）和 **Analog Switch Divider**（模拟开关分频器）确定在哪一频率对传感器电容器进行开关。**Scan Clock**（扫描时钟）是 CapSense 组件的首要时钟源。**Analog Switch Divider**（预分频器）分割扫描时钟，产生开关时钟，用来对各传感器进行充电和放电。如果传感器充放电不完全，可通过增加 **Analog Switch Divider**（模拟开关分频器）来降低开关频率。

   要测试传感器是否满充满放，可通过探头探查各个传感器引脚。注意：通过示波器探头观察传感器电容的电压时，探头的电容会增加到传感器寄生电容中。 在 10x 模式下使用探头可减小探头电容。如果可能，可使用 FET 输入探头。 确保电容器完全充放电；如果不能，可在组件配置中增大 **Analog Switch Divider** 的值。在调谐器 GUI 中不能更改该参数，因此必须在组件配置中进行设置。因此，当该值在组件 **Configure**（配置）对话框中改变时，项目将重建，并针对器件进行编程。
3. **Scan Speed**（扫描速度）初始值被设置为 **Fast**（快速）。如果未达到信噪比（信噪比）或扫描时间要求，可以在后期对它进行更改。
4. 默认使能 PRS，以减少外部 EMI 对 CapSense 的影响，并减少传感器扫描辐射。在易受 EMI 作用影响的设计中启动 PRS。PRS 的分辨率可设为 8 或 16 位，具体依扫描时间而定。如果扫描时间长，使用 PRS 16；如果扫描时间短，则使用 PRS 8。
5. 打开“CapSense 调谐器 GUI”。
6. 将分辨率设置为 10。

   增加分辨率和扫描速度可以提高灵敏度，但会增加扫描时间。因此，在扫描时间和灵敏度之间要进行一定的折中。

   在调试过程中，分辨率的最佳初始值应为 10；虽然 8 和 9 也可用作初始值，但设计中的外覆层厚度小于 1 毫米。
7. 在 GUI 中更改 IDAC 值，直至原始计数达到满标度值的 80%。分辨率全量程的值为 2 的分辨率次方。注意降低 IDAC 值可增加原始计数(raw count)，反之亦然。如果任何 IDAC 值都不能达到满标度值的 80%，则应在组件配置中更改 IDAC 的范围。IDAC 范围不能在“Tuner GUI”中进行更改；而应在组件 **Configure**（配置）对话框中更改。当 **Configure**（配置）对话框中的值变化以后，应再次建立项目，并针对器件对其编程。
8. 监测手指触摸和离开两种情况下的原始计数（raw count）。注意：峰—峰噪声和峰值手指触摸反应。按如下公式计算信噪比：

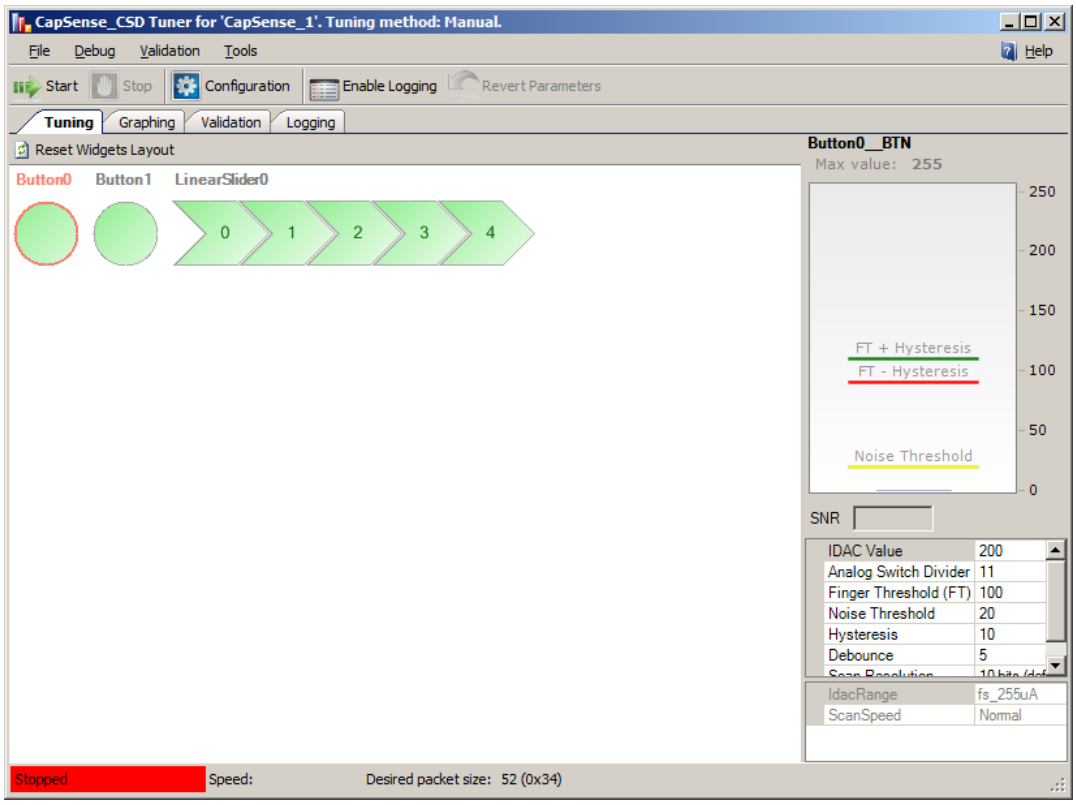
$$\text{SNR} = \frac{\text{Peak Finger Response}}{\text{Peak to Peak Noise when finger is not present}}$$

9. 为获得良好的 CapSense 设计，信噪比应大于 5。检查总扫描时间是否符合设计要求。如果未达到信噪比要求，请增加固件滤波器。请参考滤波器一节，然后选择与系统噪声相适应的滤波器类型。对于大部分设计，从一阶 IIR 1/4 滤波器开始，因为它要求的 SRAM 最小，并能提供快速反应。
10. 按第 8 步骤检查信噪比。同时，检查是否满足了设计的扫描时间要求。点击 GUI 上的 **OK** 按键，会将 GUI 中经过调试的值更新到组件中。组件根据参数设置计算出了扫描时间的近似值。扫描时间将显示在“scan order”（扫描顺序）选项卡中。如果设计中分辨率高且扫描速度低的传感器数量多，则所有传感器的总扫描时间会使传感器扫描间隔变得很长。

    如果**信噪比低于 5**，会增加分辨率、扫描速度，或两者。这样，扫描时间将延长。因此，分辨率和扫描时间两者均应进行调试，以便达到高于 5 的信噪比，并使扫描时间低于设计规定。再次检查信噪比和扫描。如果不能达到 5:1 的信噪比并将扫描时间限制在设计规定范围内，请从 PCB 布局或覆盖层设计中寻找可改进之处。请参考 *CapSense 入门*，了解 PCB 设计指导。可以通过减小覆盖层的厚度、或增加按键直径来增加灵敏度。
11. 达到 5:1 的信噪比后，需要设置以下固件参数。
    - 手指阈值是用于确定传感器是否处于活动状态的阈值参数。将该值设为手指峰值响应的 75%。
    - 将迟滞设为手指峰值响应的 15%。
    - 将噪声阈值设为手指响应的 40%。
    - 去抖动确保像静电放电事件这样的高频、高振幅噪声不至于引起按键激活。去抖值应是很小的数，比如 1 或 2，因为能够触发假按键触发的尖峰信号或高频噪声不会有两个扫描长度那么宽。在快速扫描设计中，去抖值应设置为更高的值，比如 5。

调谐器 GUI 界面

通用界面



顶部面板按键如下：

- **Start**（或主菜单项 **Debug > Start**）— 开始从芯片上读取和显示数据。如果配置，还将开始图解和日志。
- **Stop**（或主菜单项 **Debug > Stop**）— 停止从芯片上读取和显示数据。
- **Configuration**(或主菜单项 **Debug > Configuration**）— 打开 **Communication Configuration**（通信配置）对话框。
- **Enable Logging**（或主菜单项 **Debug > Start**）— 使能将从器件接收到的数据日志到日志文件。

主菜单：

- **File > Settings > Load Settings from File** — 从 XML 调试文件导入设置并将所有数据加载到调谐器。
- **File > Help** — 打开帮助文件。

其他项重复上面板和底面板按键的功能。

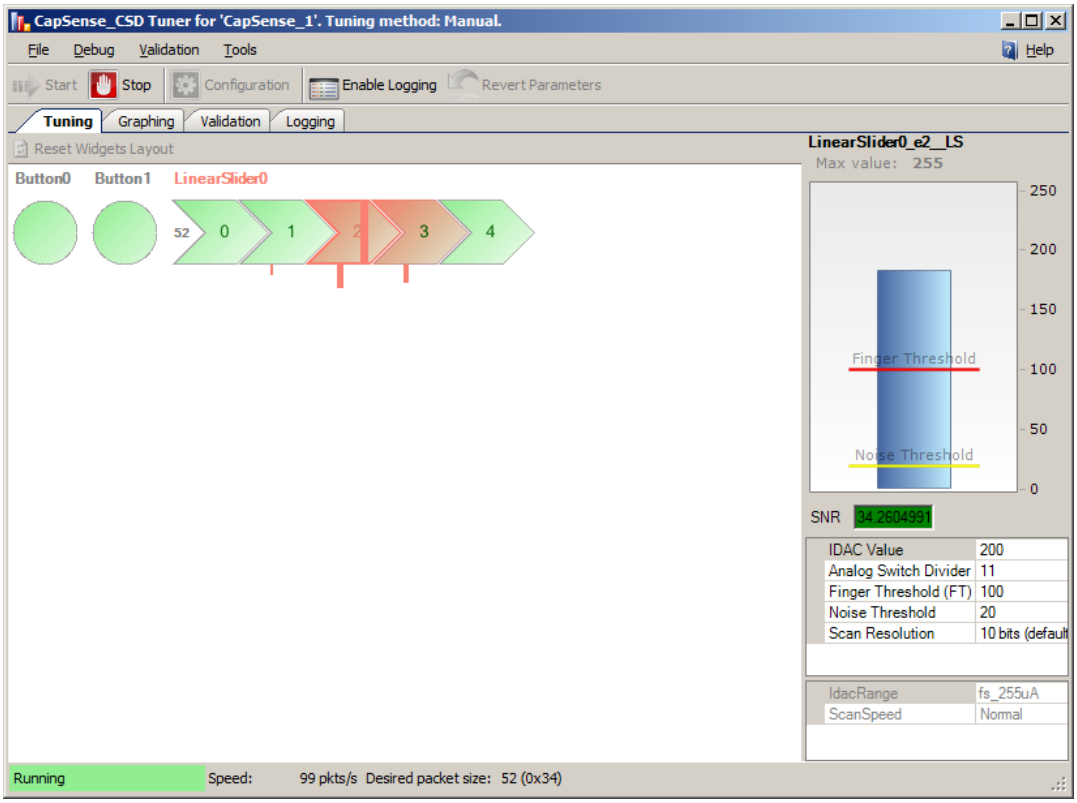
选项卡：

- **Tuning** — 显示在工作区配置的所有组件 Widget。这样，您就可以按照 Widget 出现在物理 PCB 或外壳的类似方式对 Widget 进行排列。该选项卡用来调试 Widget 参数以及可视化 Widget 数据和状态。
- **Graphing** — 在图表上详细显示单个 Widget 数据。
- **Logging** — 提供日志数据功能和调试功能。

底面板按键：

- **OK**（或主菜单项 **File > Apply Changes and Close**）— 将参数的当前值应用到 CapSense 组件实例，并退出 GUI。
- **Cancel**（或主菜单项 **File > Exit**）— 退出 GUI 而不将参数值应用到组件实例。

调试选项卡

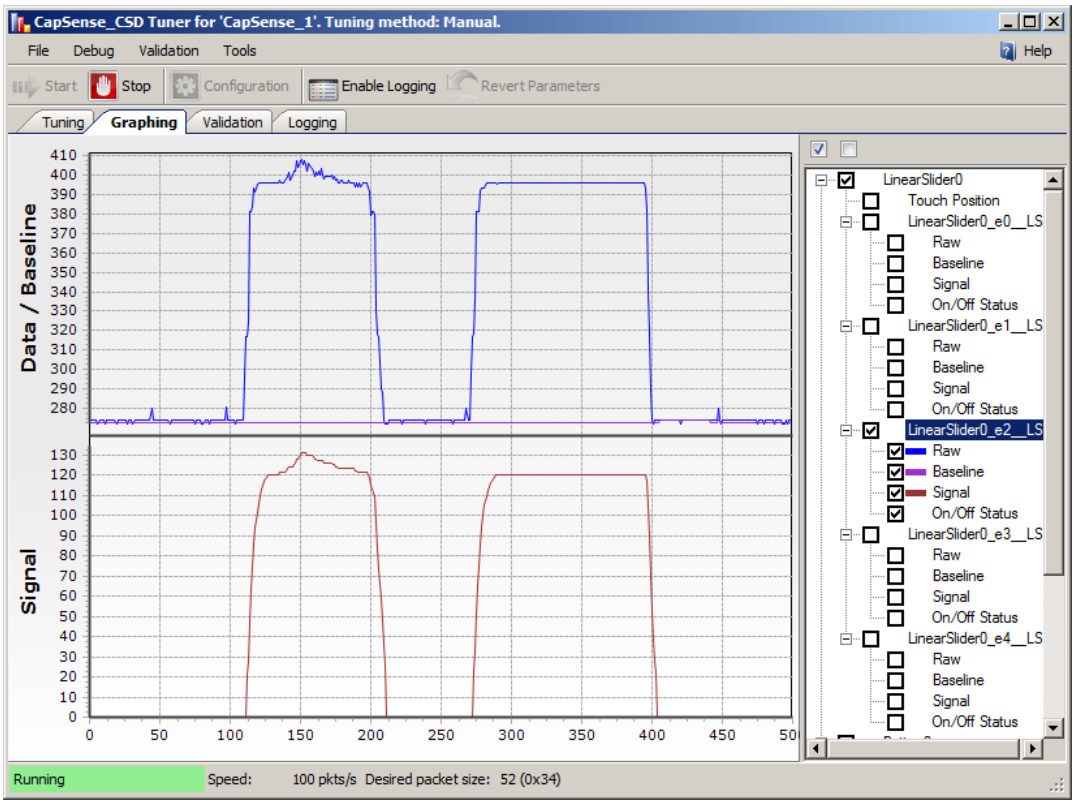


- **Widgets schematic**（Widgets 原理图）— 包含所有配置的 Widget 的图形表示。如果 Widget 由多个传感器组成，可以选定单个传感器进行详细分析。每一“widget”均可在原理图内移动。

- **Reset Widget Layout**（重置 widget 布局）按键 — 将各 widget 移动到原理图内默认的位置。
- **Bar graph**（条形图） — 显示选定传感器的信号值。
  - 通过双击 Max Value（最大值）标签，可调整到详细视图条形图的最大比例。有效范围介于 1 和 255 之间，默认值为 **255**。
  - 当前手指打开阈值显示为穿过条形图的一条**绿线**。
  - 当前手指关闭阈值显示为穿过条形图的一条**红线**。
  - 当前噪声阈值显示为穿过条形图的一条**黄线**。
- **SNR**(信噪比) — 对选定传感器的信噪比进行实时计算。信噪比值低于 5 表示不佳，显示为红色；5 至 10 表示临界，显示为黄色；大于 10 表示良好，显示为绿色。信噪比值根据之前接收到的数据进行计算。
- **Revert Parameters**（还原参数）按键 — 将参数重置为其初始值，并将这些值发送到芯片。初始值为启动 GUI 时所显示的值。
- **Sensor properties**（传感器属性）— 根据 Widget 类型，显示选定传感器的属性。其位于右侧面板上。
- **General CapSense properties**（CapSense 的常规特性，只读）— 显示 CapSense CSD 组件的全局属性，该属性不可在运行时进行更改。下列内容仅供参考。其位于右侧面板底部。
- **Widget 控件上下文菜单**（此功能仅适用于 GUI 内 Widget 控件的布局）：
  - **Send To Back**（置于背面）— 将 Widget 控件置于视图背面。
  - **Bring To Front**（置于正面）— 将 Widget 控件置于视图正面。
  - **Rotate Clockwise 90**（顺时针旋转 90 度）— 将 Widget 控件顺时针旋转 90 度。（仅适用于线性滑条）
  - **Rotate Counter Clockwise 90**（逆时针旋转 90 度）— 将 Widget 控件逆时针旋转 90 度。（仅适用于线性滑条）
  - **Flip Sensors**（翻转传感器）— 颠倒传感器的顺序。（仅适用于线性和辐射滑条）
  - **Flip Columns Sensors**（翻转列中的传感器）— 颠倒列中传感器的顺序。（仅适用于触控板和矩阵按键）
  - **Flip Row Sensors**（翻转行中的传感器）— 颠倒行中传感器的顺序。（仅适用于触控板和矩阵按键）

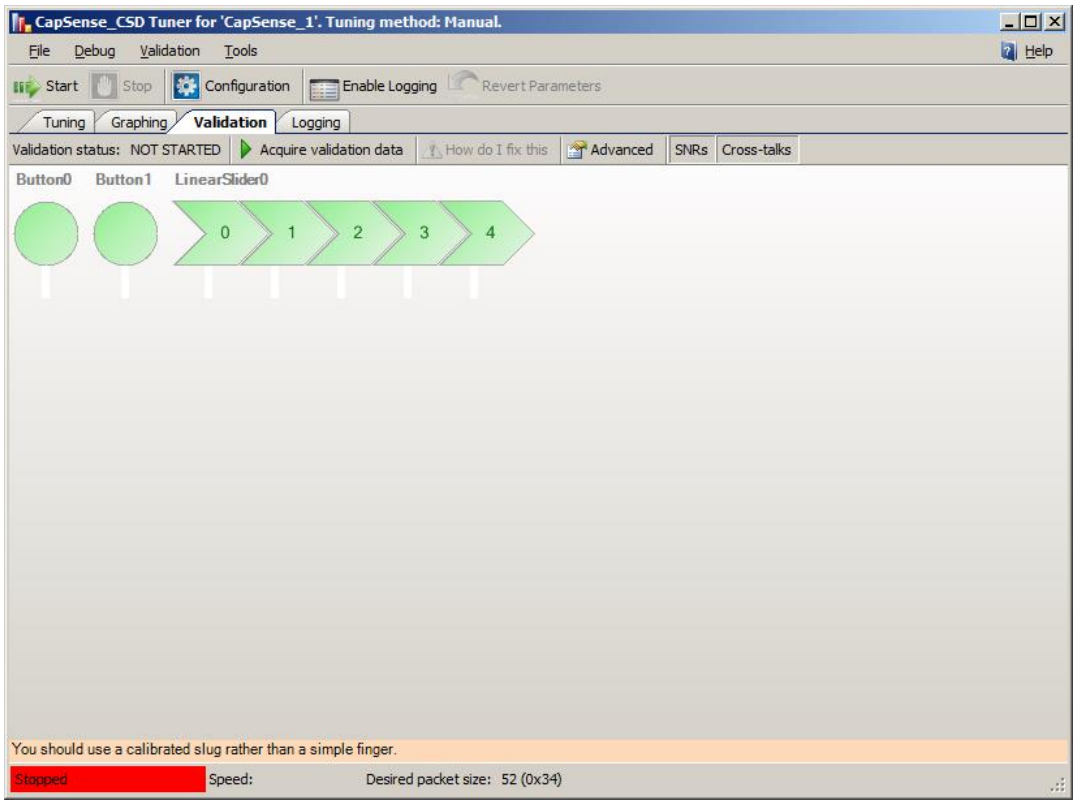
- **Exchange Columns and Rows**（交换行列）— 将列中的传感器置于行中，行中的传感器置于列中。（仅适用于触控板和矩阵按键）

Graphing Tab（图解选项卡）



- **Chart area**（图表区域）— 显示从树视图中所选项目的图表。如果右键单击菜单项 **Export to .jpg**，则能生成图表区的屏幕截图，保存为“.jpg”格式文件。
- **Tree view**(树视图) — 提供各种组合的 Widget 和传感器数据，这些数据在日志功能使能时可显示在图表中，并日志在文件中。开/关状态数据的值只能得到记录，不能显示在图表中。

Validation Tab（验证选项卡）



Validation选项卡仅用于诊断目的。该选项卡有Widget布局视图，但不能对布局进行编辑。这个布局部分只能用于显示。

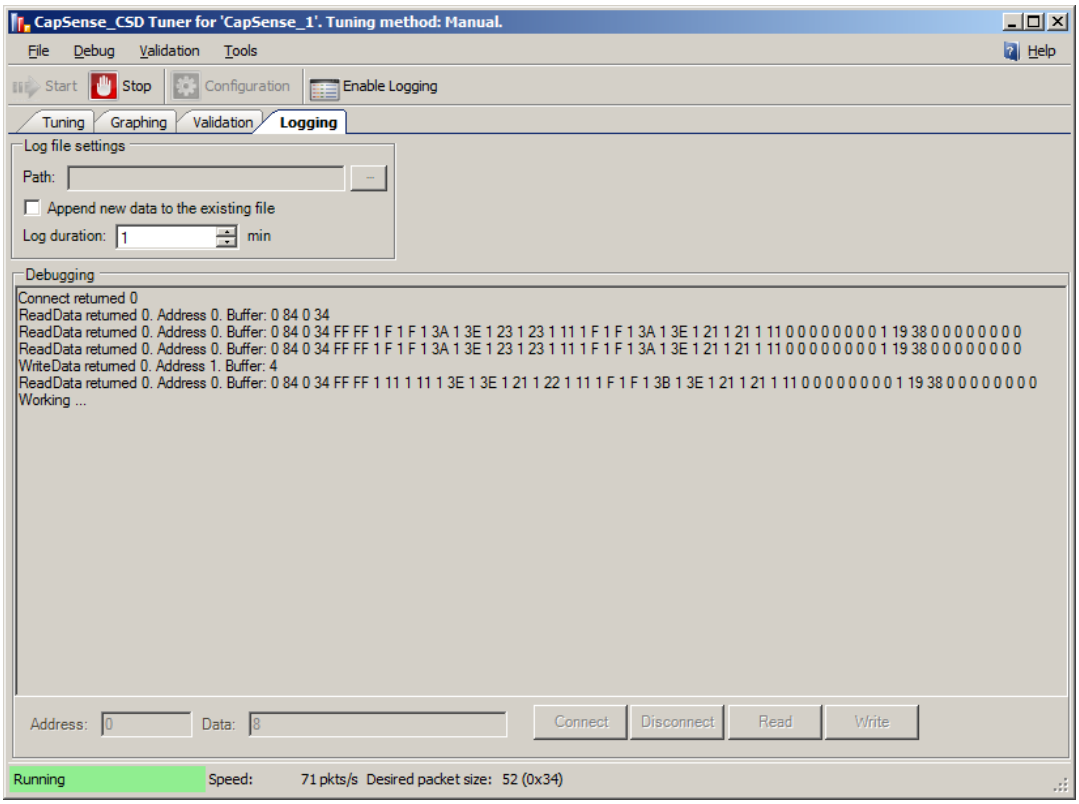
- **Widgets schematic**（Widgets 原理图）— 包含所有配置的 Widget 的图形表示。

顶部面板控件：

- **Validation Status** 标签 — 显示验证状态。它有以下信息：
  - **VALIDATION NOT STARTED** — 自上次设计修改以来，未进行过验证过程。
  - **PASS** — 完整验证过程已完成，且无故障。
  - **FAIL** — 验证过程发现了故障；将显示验证报告。
- **Acquire Validation Data** 按键（或主菜单项 **Validation > Acquire Validation Data**）— 开始验证过程。该过程引导用户完成一系列操作，在其中会提示您按顺序按压各传感器。
- **How do I fix this** 按键 — 打开一个报告，上面列出未通过验证传感器的建议修复方案。只有先前完成了验证过程并发现了设计错误的情况下，该按键才可用。
- **Advanced** 按键（或主菜单项 **Validation > Validation Advanced properties**）— 打开验证属性的验证窗口（更多信息，请参考验证高级属性一节）。

- **SNRs**（信噪比）按键 — 在 widget 原理图上打开或关闭信噪比（更多信息，请参见{验证显示）。
- **Crosstalks**（串扰）按键 — 在“widget”原理图上打开或关闭串扰显示（更多信息，请参见验证显示）。

选项卡



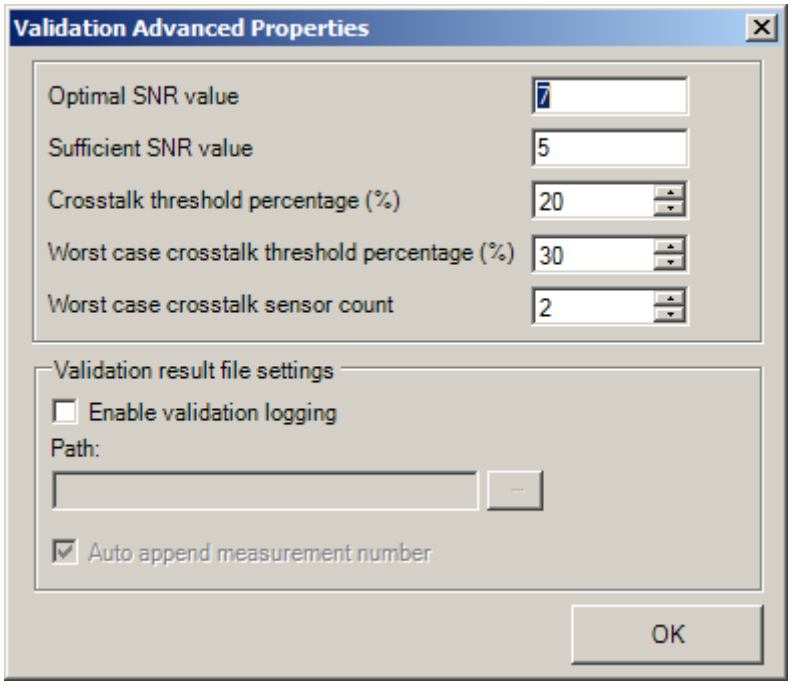
- 通过选中 **Graphing** 选项卡树视图中的复选框，可选定将要记录的数据。
- **Path** — 定义记录文件的路径（文件扩展名为“.csv”）。
- **Append new data to existing file**（将新数据附加到现有文件）复选框 — 选中该选项后，新数据即会附加到现有文件中。如果未选中，旧文件将从文件中被擦除，代之以新数据。
- **Log duration**（记录持续时间）— 定义记录持续时间，单位为分钟。有效范围介于 1 和 480 之间，默认值为 **255**。

Debugging group（调试组）

该功能的存在仅为调试目的。它能帮助用户研究调试器通信错误。

- Debugging log（调试日志）窗口 — 显示调谐器执行的通信指令。所有通信日志错误均被记录于此。如果成功启动了调谐器，则仅记录最开始的几条通信指令。
- **Connect**（连接）— 连接到 PSoC 器件。
- **Disconnect**（断开连接）— 断开与 PSoC 器件的连接。
- **Address** （地址）— 指出 PSoC 器件的地址。
- **Read**（读取）— 从 PSoC 器件读取数据。地址字段定义缓冲区中的地址。数据字段定义待读取的字节数。
- **Write**（写入）— 将数据写入 PSoC 器件内。地址字段定义缓冲区中的地址。数据字段定义待写入的数据。

验证高级属性



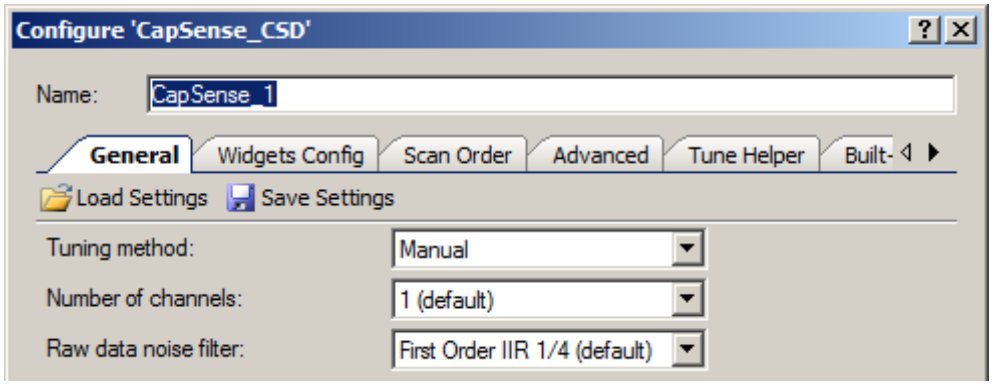
- **Optimal SNR Value**（信噪比最佳值）— 定义最佳的信噪比值。有效范围介于 0 和 100 之间，默认值为 **7**。
- **Sufficient SNR Value**（信噪比充分值）— 定义充分的信噪比值。有效范围介于 0 和 100 之间，默认值为 **5**。
- **Crosstalk Threshold Percentage (%)**（串扰阈值百分比） — 定义每一个传感器串扰阈值占手指阈值的百分比。有效范围介于 0 和 100 之间，默认值为 **20**。

- **Worst Case Crosstalk Threshold Percentage (%)**（最差情形串扰阈值百分比）— 定义最差情形串扰阈值占最差情形串扰的百分比。有效范围介于 0 和 100 之间，默认值为 **30**。
- **Worst Case Crosstalk Sensor Count**（最差情形串扰传感器计数）— 定义用于计算最差情形串扰传感器的数量；有效范围从 0 到 100；默认值为 **2**。
- **Enable Validation Logging**（使能验证日志）— 使能对验证数据的日志。
- **Path**（路径）— 定义验证数据日志文件的路径（文件扩展名为“.csv”）。
- **Auto Append Measurement Number**（自动附加测量编号）选框 — 如果勾选，在每次启动验证过程后，日志文件名将会递增（如“validation001.csv”），数据将被保存在一个新文件中。

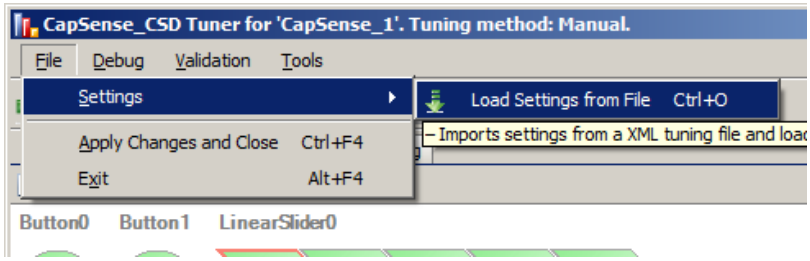
Save and Load Settings（保存/加载设置）功能

调谐器 GUI 也可以作为独立应用程序打开。在这种情况下，用户必须使用 CapSense CSD 组件调谐器 GUI 的“保存和加载设置”功能。

1. 点击定制器中的 **Save Settings** 按键。



2. 在 **Save File** 对话框中，指定文件的名称及其保存位置。
3. 打开调谐器窗口，然后依次选择 **File > Settings > Load Settings from File**。



4. 在 **File Open** 对话框中，通过组件设置指向之前保存的文件。设置会自动载入调谐器中。

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。

组件可用于支持以下编译器的开发系统中：

- Keil 8051 编译器
- ARM GCC 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，需要创建到静态库的路径。该库位于 PSoC Creator 安装目录中：
“PSoC Creator\psoc\content\CyComponentLibrary\CyComponentLibrary.cylib\CortexM3\IAR”

默认情况下，PSoC Creator 将实例名称“CapSense_1”分配给指定设计中组件的第一个实例。您可以将其重命名为遵循标识符语法规则的任何唯一值。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”。

注意：为了删除编译器的警告，要将 CYREENTRANT 关键词添加到所有 CapSense CSD API 中。包括“.cyre”文件中的下面各函数。并非所有 API 都是真正可重入的函数。组件 API 源文件中的注释指出了适用的函数。

另外，还将组建所提供的所有函数（即 CapSense_CSD 的一部分）添加到“.cyre”文件中。这些函数包括：

```
CapSense_CompCH0_SetSpeed()
CapSense_CompCH1_SetSpeed()

CapSense_IdacCH0_SetValue()
CapSense_IdacCH0_SetRange()
CapSense_IdacCH0_DacTrim()
CapSense_IdacCH1_SetValue()
CapSense_IdacCH1_SetRange()
CapSense_IdacCH1_DacTrim()

CapSense_AMuxCH0_Set()
CapSense_AMuxCH0_Unset()

CapSense_AMuxCH1_Set()
CapSense_AMuxCH1_Unset()
```

下表对各个函数进行了概述。以下各节将更详细地介绍每个函数。

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中所用子组件的活动模式电源模板位。
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（活跃）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，还原CapSense配置和非自保持寄存器值。
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（活跃）模式电源模板位。
CapSense_SaveConfig()	保存CapSense非自保持寄存器的配置。将所有传感器复位到非活动状态。
CapSense_RestoreConfig()	还原CapSense配置和非自保持寄存器值。

void CapSense_Start(void)

说明： 这是开始执行组件操作的首选方法。CapSense_Start()依次调用CapSense_Init()和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

其他影响： 应在完成所有扫描操作后调用该函数。

void CapSense_Sleep(void)

说明： 这是为器件的低功耗模式准备组件的首选方法。禁用CapSense中所用子组件的Active（活动）模式电源模板位。调用CapSense_SaveConfig()函数以保存CapSense非自保持寄存器的定制器配置，并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： 应该在完成扫描操作后调用该函数。

此函数不能将CapSense组件所使用的引脚置于最低功耗状态。要更改引脚的驱动模式，请使用引脚API部分中所述的函数。

void CapSense_Wakeup(void)

说明： 还原CapSense配置和非自保持寄存器值。通过为CapSense中所用的子组件设置活动模式电源模板位，可恢复组件的使能状态。

参数： 无

返回值： 无

其他影响： 该函数不能将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： 使能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非自保持寄存器的配置。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： 应在完成扫描操作后调用该函数。

此函数不能将CapSense组件所使用的引脚置于最低功耗状态。要更改引脚的驱动模式，请使用引脚API一节所述的函数。

void CapSense_RestoreConfig(void)

说明： 还原CapSense配置和非自保持寄存器值。

参数： 无

返回值： 无

其他影响： 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()	设定扫描设置，然后开始扫描每条通道上的单个传感器或一组传感器。
CapSense_ScanEnabledWidgets()	首选的扫描方法。扫描所有已使能的Widget。
CapSense_IsBusy()	返回传感器扫描状态。
CapSense_SetScanSlotSettings()	设定所选扫描插槽的扫描设置（一个传感器或一对传感器）。
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()	禁用所选的传感器，以便在下一个扫描周期内不对其进行扫描。
CapSense_ReadSensorRaw()	返回CapSense_SensorRaw[ ]陈列中传感器的原始数据。
CapSense_SetRBleed()	设定引脚，以用于泄露电阻(Rb)连接（若采用多个泄露电阻）。

void CapSense_ScanSensor(uint8 sensor)

说明： 设定扫描设置，然后开始扫描每条通道上的一个传感器或一对传感器。若配置了两条通道，则可以同时扫描两个传感器。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有唯一编号。该编号由CapSense定制器依次分配。

参数： uint8 sensor：传感器编号

返回值： 无

其他影响： 无

void CapSense_ScanEnabledWidgets(void)

说明： 这是扫描所有已使能Widget的首选方法。开始对已使能Widget内的一个传感器或一对传感器进行扫描。ISR对传感器进行持续扫描，直到所有已使能Widget均扫描完毕。使用ISR可确保该函数无阻塞。

除接近Widget以外，所有Widget都在默认情况下使能。 由于接近感应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： 如果未使能任何Widget，则此次函数调用将无效。

uint8 CapSense_IsBusy (void)

说明： 返回传感器扫描状态。

参数： 无

返回值： uint8：返回扫描状态。‘1’ — 正在扫描，‘0’ — 扫描完成。

其他影响： 无

void CapSense_SetScanSlotSettings(uint8 slot)

说明： 对定制器中提供的扫描设置进行设定，或对所选扫描插槽（用于双通道设计的一个传感器或一对传感器）的向导进行设定。这些扫描设置提供了每个传感器的IDAC值（用于IDAC配置）以及分辨率。在一个Widget内，所有传感器的分辨率均相同。

参数： uint8 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(uint8 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： uint8 sensor：传感器编号

返回值： 无

其他影响： 无

void CapSense_DisableSensor(uint8 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： uint8 sensor：传感器编号

返回值： 无

其他影响： 无

uint16 CapSense_ReadSensorRaw(uint8 sensor)

说明： 返回全局CapSense_SensorRaw[ ]陈列中传感器的原始数据。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。原始数据可用于执行CapSense所提供框架以外的计算。

参数： uint8 sensor：传感器编号

返回值： uint16：当前的原始数据值

其他影响： 无

void CapSense_SetRBleed(uint8 rbleed)

说明： 设定引脚，以用于泄漏电阻（Rb）连接。可在运行时调用此函数，以便从定制器中定义的Rb引脚设置中选出当前Rb引脚设置。此函数将会覆盖组件参数设置。只有将**Current Source**（电流源）设置为**External Resistor**（外部电阻）时，才能使用此函数。

若某些传感器需要用不同的泄露电阻值进行扫描，则此函数有效。例如，常规按键可以用较小的泄露电阻值进行扫描。接近检测器可以用较大的泄露电阻以较低的频率进行扫描，以便最大限度延长接近检测距离。此函数可结合CapSense_ScanSensor()函数使用。

参数： uint8 rbleed：CapSense定制器中定义的泄露电阻的序列号。

返回值： 无

其他影响： 泄露电阻的数量被限制在3个以内。此函数不会检查超出限制的数量。

高级 API

这些 API 函数用于处理传感器 Widget 的原始数据。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()	通过扫描所选传感器，加载含初始值的CapSense_sensorBaseline[sensor]阵列元素。
CapSense_InitializeEnabledBaselines()	仅通过扫描使能的传感器，才可加载含初始值的CapSense_sensorBaseline[]阵列。 该函数仅在双通道设计中可用。
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[]陈列并调用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[]”陈列相比较，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_signal[]陈列，以确定线性滑条上是否存在手指触摸，并返回位置。

函数	说明
CapSense_GetRadialCentroidPos()	检查CapSense_signal[]陈列，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。

void CapSense_InitializeSensorBaseline(uint8 sensor)

说明： 通过扫描选定的一个传感器（单通道设计）或一对传感器（双通道设计）来加载含初始值的CapSense_SensorBaseline[sensor]陈列元素。将原始计数（raw count）值复制到每个传感器的基准线陈列中。初始化原始数据滤波器（如果已使能）。

参数： uint8 sensor：传感器编号

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[ ]陈列。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（raw counts）过滤器（如果已使能）。

该函数仅在双通道设计中可用。

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(uint8 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]陈列元素。该函数通过将之前的基准线从当前原始计数（raw count）值中扣除并将其存储在CapSense_signal[sensor]中，来计算差值计数。

如果使能自动复位选项，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使能，则可应用于这些数值。

参数： uint8 sensor：传感器编号

返回值： 无

其他影响： 无

void CapSense_UpdateEnabledBaselines(void)

说明： 检查CapSense_sensorEnableMask[]陈列并调用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(uint8 widget)

说明： 使能所选的Widget传感器，以作为扫描过程中的一部分。

参数： uint8 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。

返回值： 无

其他影响： 无

void CapSense_DisableWidget(uint8 widget)

说明： 禁用所有扫描过程中选定的Widget传感器。

参数： uint8 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。

返回值： 无

其他影响： 无

uint8 CapSense_CheckIsWidgetActive(uint8 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。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。此函数还更新传感器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： uint8 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。

返回值： uint8：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，则计为1，如果Widget内所有的传感器均处于非活动状态，则计为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。

uint8 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()，以便在调用此函数后，CapSense_sensorOnMask[ ]阵列为最新。

参数： 无

返回值： uint8：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(uint8 widget)

说明： 检查CapSense_signal[ ]陈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。位置滤波器（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： uint8 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。

返回值： uint16：线性滑条的位置数值

其他影响： 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(uint8 widget)

说明： 检查CapSense_signal[ ]陈列在辐射滑条内是否存在指压。 将手指位置计算为CapSense定制器中指定的API分辨率。位置滤波器（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： uint8 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。

返回值： uint16：辐射滑条的位置数值。

其他影响： 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。

uint8 CapSense_GetTouchCentroidPos(uint8 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触摸，则返回“1”。位置滤波器（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): pointer to an array of two uint16, where touch
postion will be stored:
    pos[0] - X position;
    pos[1] - Y position.
```

返回值： uint8：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：

uint8 CapSense_GetMatrixButtonPos(uint8 widget, uint8* pos)

说明： 如果触摸了矩阵按键，该函数会计算手指的行位置和列位置。如果触摸了矩阵按键，则返回一个‘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): pointer to an array of two uint8, where touch
postion will be stored:
    pos[0] - column position;
    pos[1] - row position.
```

返回值： uint8：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：

调谐器助手 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EZI2C组件，初始化基准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组件和EZI2C组件。同时初始化基准线并用当前已使能的传感器启动传感器扫描循环。

除接近Widget以外，所有Widget都在默认情况下使能。由于接近感应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 组件传输到调谐器 GUI。从调谐器 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。调谐器 GUI 参数不会被传输回 CapSense CSD 组件。

此函数正在执行阻塞操作，并等待调谐器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： 无

引脚 API

这些 API 函数用于更改 CapSense 组件所用引脚的驱动模式。这些 API 大多用于将 CapSense CSD 组件引脚置于强驱动模式，以便在器件处于低功耗模式时将漏电流降到最低。

功能	说明
CapSense_SetAllSensorsDriveMode()	为CapSense组件内电容传感器所使用的所有引脚设置驱动模式。
CapSense_SetAllCmodsDriveMode()	为CapSense组件内C_{MOD}电容所使用的所有引脚设置驱动模式。
CapSense_SetAllRbsDriveMode()	为CapSense组件内泄露电阻（Rb）所使用的所有引脚设置驱动模式。仅在**Current Source**（电流源）被设置为**External Resistor**（外部电阻）时才可用。

void CapSense_SetAllSensorsDriveMode(uint8 mode)

说明： 为CapSense组件内电容传感器所使用的所有引脚设置驱动模式。

参数： uint8 mode：想要的驱动模式 有关驱动模式的信息，请参考引脚组件数据手册。

返回值： 无

其他影响： 无

void CapSense_SetAllCmodsDriveMode(uint8 mode)

说明： 为CapSense组件内C_{MOD}电容所使用的所有引脚设置驱动模式。

参数： uint8 mode：想要的驱动模式 有关驱动模式的信息，请参考引脚组件数据手册。

返回值： 无

其他影响： 无

void CapSense_SetAllRbsDriveMode(uint8 mode)

说明： 为CapSense组件内泄露电阻（Rb）所使用的所有引脚设置驱动模式。仅在**Current Source**（电流源）被设置为**External Resistor**（外部电阻）时才可用。

参数： uint8 mode：想要的驱动模式 有关驱动模式的信息，请参考引脚组件数据手册。

返回值： 无

其他影响： 无

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，可以使用绘图工具来显示阵列的内容。全局陈列为：

- CapSense_sensorRaw [ ]
- CapSense_sensorEnableMask [ ]
- CapSense_portTable[ ]和 CapSense_maskTable[ ]
- CapSense_sensorBaseline [ ]
- CapSense_sensorBaselineLow[ ]
- CapSense_sensorSignal [ ]
- CapSense_sensorOnMask[ ]

CapSense_sensorRaw [ ]

此陈列包含每个传感器的原始数据。 陈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陈列 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[ ]数据由以下函数使用：

- CapSense_ScanEnabledWidgets()

CapSense_portTable[ ]和 CapSense_maskTable[ ]

这些陈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[ ]

此陈列存储低通滤波器中所使用每个传感器的基准线数据分数字节，用于基准线更新。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()

CapSense_sensorBaseline[ ]

此陈列存储每个传感器的基准线数据。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[ ]

此陈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准线计算而来的传感器信号数据。阵列大小等于传感器总数。**Widget Resolution** 参数将此陈列的分辨率定义为 **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[ ]

这是一个保持传感器开/关状态的字节陈列。

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），依次类推。此字节陈列存储的元件数足以包含所有的传感器。如果传感器开启（活动状态），则位值为 1；如果传感器关闭（非活动状态），则位值为 0。CapSense_sensorOnMask[ ]数据通过以下函数进行更新：

- CapSense_CheckIsWidgetActive()
- CapSense_CheckIsAnyWidgetActive()

常量

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

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

对于双通道设计，属于某个通道的传感器数量进行规定如下：

- CapSense_TOTAL_SENSOR_COUNT__CH0 — 规定属于通道 0 的传感器总数。

- CapSense_TOTAL_SENSOR_COUNT__CH1 — 规定属于通道 1 的传感器总数。
- CapSense_CSD_TOTAL_SCANSLOT_COUNT — 规定通道 0 或通道 1 中最大的传感器数量。

传感器常量

每个传感器均提供一个常量。可在以下函数中将这些常量作为参数使用：

- CapSense_EnableSensor()
- CapSense_DisableSensor()

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*

例如：

```
#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 名称必须在 CapSense CSD 组件中唯一。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	触控板
MB	矩阵按键
PROX	接近感应传感器

别名	说明
GEN	通用传感器
GRD	保护传感器

Widget 常量

为每个 Widget 提供一个常量。可在以下函数中将这些常量作为参数使用：

- CapSense_CheckIsWidgetActive()
- CapSense_EnableWidget()和 CapSense_DisableWidget()
- CapSense_GetCentroidPos()
- CapSense_GetRadialCentroidPos()
- CapSense_GetTouchCentroidPos()

常量由以下内容组成：

实例名称 + *Widget 名称* + *Widget 类型*

例如：

```
#define CapSense_UP__BTN         0
#define CapSense_DOWN__BTN       1
#define CapSense_VOLUME__SL      2
#define CapSense_TOUCHPAD__TP    3
```

MISRA 合规性

本节介绍了 MISRA-C:2004 合规性以及本组件的偏差情况。 有两种偏差类型，如下定义：

- 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

本节介绍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 组件具有如下特定偏差：

MISRA-C:2004规则	规则类别（必须（R）/建议（A））	规则说明	目标器件	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。	PSoC 3/ PSoC 5LP	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	建议	不同对象指针类型间不可显式转换。	PSoC 3/ PSoC 5LP	在组件调谐器助手中，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	陈列索引是唯一允许的指针运算形式。	PSoC 3/ PSoC 5LP	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	建议	使用时，函数应优先于类函数宏。	PSoC 3/ PSoC 5LP	类函数宏用于提高性能。

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，包括原理图和代码。要获取组件特定的示例，请右击组件并打开“Find Example Project”对话框来查看组件实例。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。

更多有关信息，请参考《PSoC Creator 帮助》部分中主题为“查找实例项目”的内容。

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至“设计范围资源”文件视图“引脚编辑器”选项卡中的引脚。

侧面

PSoC 器件内的模拟布线矩阵分为两半 — 左侧和右侧。偶数端口号引脚位于器件左侧，奇数端口号引脚位于右侧。

对于串行传感应用，可将传感器引脚分配至器件的任何一侧。如果此应用使用了少量传感器，将所有传感器信号分配至器件的一侧会使模拟资源的布线更加有效，并为其他组件释放模拟资源。

在并行传感应用中，CapSense 组件能够在两组独立硬件上执行两个同步扫描。两条并联电路中的每条电路均有一个单独的 C_{MOD} 和 Rb（如适用）及其自己的一组传感器引脚。一组占据器件右侧，另一组占据左侧。 信号名称别名指示信号关联的一侧。

传感器引脚 — CapSense_cPort — 引脚分配

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*

注意：在双通道设计中，属于一个通道的 Widget 元件仅能连接至与通道 C_{MOD} 相同的芯片侧。**引脚编辑器不会使用设计规则检查**验证引脚分配是否正确。 引脚放置错误将在构建过程中标记出来。

注意：运算放大器输出 P0[0]、P0[1]、P3[6]和 P3[7]具有比其他引脚更高的寄生电容。这导致了 CapSense 应用中来自 P0[0]、P0[1]、P3[6] 和 P3[7]手指响应减少，因此如有可能，应避免此情况。如果必须使用，若电容差不会转换为滑条和触控板位置错误，则应将其用于单个按键。

CapSense_cCmod_Port — 引脚分配

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。双通道设计需要两个 C_{MOD} 电容，一个用于器件左侧，一个用于器件右侧。可将 C_{MOD} 连接至**任何引脚**，但对于大多数高效模拟布线来说，以下引脚允许直接连接：

- 左侧：P2[0]、P2[4]、P6[0]、P6[4]、P15[4]
- 右侧：P1[0]、P1[4]、P5[0]、P5[4]

C_{MOD} 电容的别名为：

别名	说明
CmodCH0	通道0的C_{MOD}
CmodCH1	通道1的C_{MOD}。仅在双通道设计中可用。

C_{MOD} 的理想值取决于传感器扫描电压的电压摆幅。电压摆幅越高，C_{MOD} 值也越高。传感器电压摆幅取决于 IDAC 模式和基准电压设置（Vref）。C_{MOD} 建议值使用以下公式：

对于 IDAC 源模式：

$$C_{MOD} = 2.2\ nF \times Vref$$

对于 IDAC 灌电流模式：

$$C_{MOD} = 2.2\ nF \times (V_{DD} - Vref)$$

使用陶瓷电容器。电容器的温度系数并不重要。

当**Current Source**（电流源）被设置为**External Resistor**（外部电阻）时，应在确定最佳C_{MOD}值之前选择外部Rb反馈电阻值。

CapSense_cRb_Ports —引脚分配

当 **Current Source**（电流源）被设置为 **External Resistor**（外部电阻）时，需要一个外部泄露电阻（Rb）。外部泄露电阻(Rb)应连接至一个物理引脚以及调制器电容（C_{MOD}）的不接地连接。

每一通道支持多达三个泄露电阻。可为泄漏电阻分配三个引脚：cRb0、cRb1 和 cRb2。

外部泄露电阻的别名为：

别名	说明
Rb0CH0、Rb1CH0、Rb2CH0	通道0的外部电阻
Rb0CH1、Rb1CH1、Rb2CH1	通道0的外部电阻。仅在双通道设计中可用。

电阻值取决于总传感器电容。应通过以下方法选择电阻值：

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

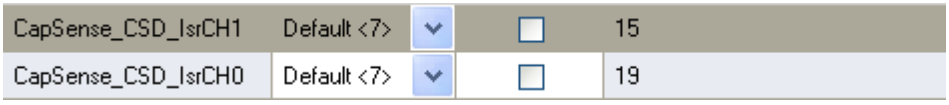
典型泄露电阻值为 500 Ω 到 10 kΩ，具体取决于传感器电容。

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。提供存根例程，您可以根据需要添加自己的代码。首次构建项目时，在 *CapSense_INT.c* 文件中生成存根例程。中断次数取决于随通道数而定的 CapSense 模式选择（每个通道一次）。您的代码必须添加在所提供的注释标签之间，在版本间得到保留。

双通道模式 ISR 优先级设置

CapSense CSD 组件的 ISR 例程不可重新进入。这会对双通道设计 ISR 优先级设置造成限制。为防止通道 ISR 例程重新进入，两个通道的 ISR 优先级必须相同。



功能说明

定义

传感器

通过一个引脚连接至 PSoC 的 CapSense 元件。传感器是基材上的一个导电元件。传感器的示例包括：FR4 上的铜、Flex 上的铜、PET 上的银墨、玻璃上的 ITO。

扫描时间

扫描时间是指 CapSense 模块扫描一个或多个电容传感器的一段时间。在给定的扫描传感器中可以组合多个传感器，以便使能诸如接近传感等模式。

CapSense Widget

CapSense Widget 由一个或多个扫描传感器构成，用于提供较高级别的功能。CapSense Widget 的一些示例包括按键、滑条、辐射滑条、触摸板、矩阵按键和接近传感器。

手指阈值

该值用于确定传感器上是否有手指触摸。

噪声阈值

用于确定电容扫描中的噪声等级。基准线算法过滤噪声，以便跟踪传感器基准线值中的电压和温度变化。

去抖动

为传感器活动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。这对于滤掉高振幅和频率噪声是有必要的。

迟滞

设置与手指阈值一起使用的迟滞值。如果需要迟滞，传感器不会被视为处于“开启”或“活动”状态，直至计数值超过手指阈值与迟滞值之和。传感器不会被视为处于“关闭”或“非活动”状态，直到所测量的计数值降至手指阈值与迟滞值差值以下。

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获取插值后的位置，首先应扫描阵列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触摸质心。可以使用最少两个、最多八个传感器来计算质心。

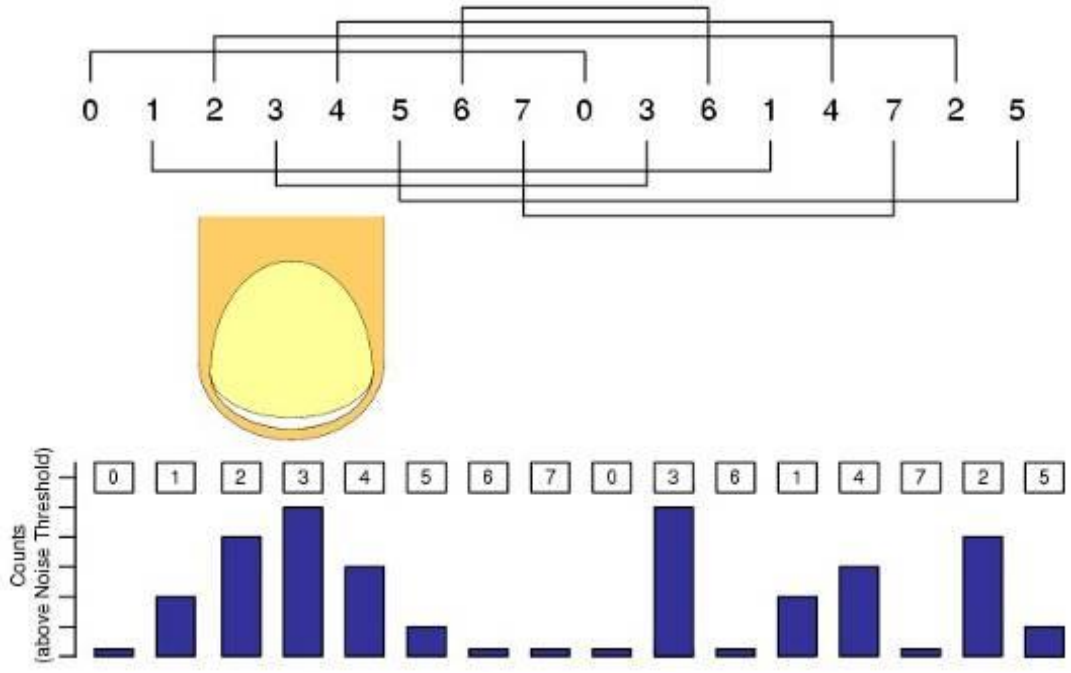
$$N_{Cent} = \frac{n_{i-1}(i-1) + n_i i + n_{i+1}(i+1)}{n_{i-1} + n_1 + n_{i+1}}$$

计算得出的值通常是分数。为了能够采用某一特定分辨率来报告质心（例如对于 12 个传感器使用 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。物理位置的前半部分（较低数值部分）按顺序映射到基部分配的传感器上，端口引脚由设计工程师使用 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“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布线。当您选择双工时，CapSense 定制器将自动生成双工传感器索引表，列在下方以供参考。

表 2. 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

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

滑条段总数	段序列
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

滤波器

CapSense 组件中提供了几种滤波器：中值、平均值、一阶 IIR 和抖动等滤波器。这些滤波器既可与传感器原始数据结合使用，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

中值滤波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，通常建议不要使能该滤波器。使能该滤波器后，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，通常建议不要使能该滤波器。使能该滤波器后，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 滤波器是原始滤波器和传感器滤波器建议使能的滤波器，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。使能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使能 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。使能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，这可导致传感器检测出错。

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。
- 添加防护传感器。防护传感器应围绕所有传感器，这样，如果覆盖任何实际的传感 Widget，则防护传感器的位置可确保水会将其覆盖。当防护传感器触发时，应通过编程方式阻止 Widget 状态的 CapSense 输出。

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

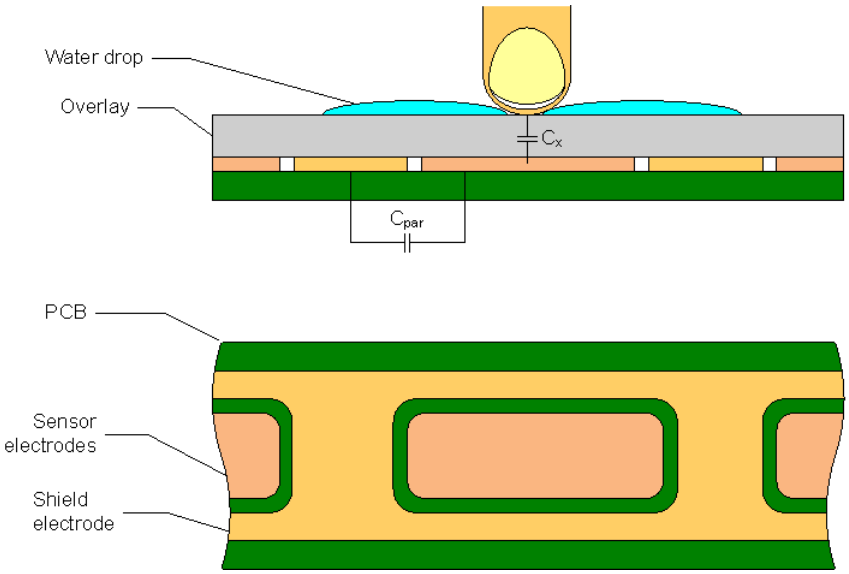


表 2 提供了一种按键屏蔽电极的可能布局组态。屏蔽电极尤其适用于透明的 ITO 触摸板器件，在这种器件中，它不但可阻止 LCD 驱动电极的噪声，同时可减少杂散电容。

在此示例中，有屏蔽电极板覆盖按键。作为另一替代方法，屏蔽电极可以安装在相对的 PCB 层上，其中包括按键下面的平板。对于这种情况，建议使用填充模式，填充率约为 30 至 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。

可将屏蔽电极连接到任何引脚。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。

Shield Electrode Use and Restrictions（屏蔽电极使用及限制）

CapSense CSD 组件为屏蔽电极的使用提供了下列模式。

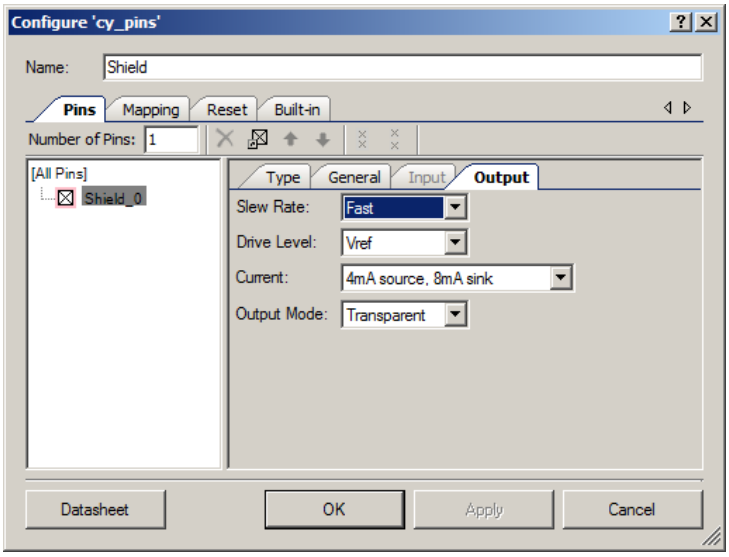
当前模式 IDAC 源

该模式有一些限制，因为传感器在 GND 和 Vref = 1.024 伏之间交替。屏蔽电极信号在 GND 和 Vddio（通常等于电源）之间交替。这种差异非常巨大，屏蔽信号可以完全抵消来自传感器的信号。可能的解决方案如下：

- 使用高 Vref 将差异降到最小值。 作为参考电压的 VDAC 可用于此目的。

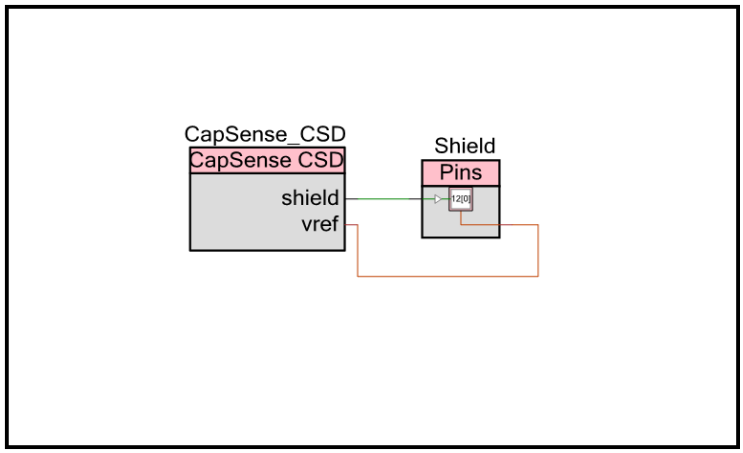
- 使用 SIO 引脚作为屏蔽以提供等于 Vref 的输出。CapSense CSD 输出 Vref 终端可用于将 Vref 布线至 SIO 引脚。此为首选方法。此模式中不能将传感器直接连接到屏蔽终端，因为其提供了等于 Vddio 的输出。Vref = 1.024 伏设置具有路由限制，无法路由至引脚。要想配置 SIO 模式中的引脚，请执行以下步骤操作：
  - 向项目原理图上放置新的数字输出引脚。
  - 导航到 **Output**（输出）选项卡。
  - 在 **Drive Level**（驱动电平）选项中选择“Vref”模式。

图 3. Vref 驱动电平的引脚配置



点击 **OK**，然后将在引脚组件上出现一个 Vref 终端。必须将该终端连接到 CapSense CSD 组件所提供的 Vref 输出。

图 4. SIO 端口和 CSD 模块之间的连接



当前模式 IDAC 灌电流能力和外部电阻

这些模式没有屏蔽和非活动传感器模式使用的限制，因为传感器在 Vddio 和 Vref = 1.024 伏之间交替。屏蔽电极信号在 GND 和 Vddio（通常等于电源）之间交替。 这种差异在该情况下不足以导致问题。

防护传感器实现

防护传感器通常用于防水应用场合来检测表面的水。

Advanced（高级）选项卡选项用于添加防护传感器。此传感器必须具有专门的布局，通常位于传感区表面周围。当防护传感器的表面有水时，Widget 将变为活动状态。Widget 活动检测固件 CapSense_1_IsWidgetActive()可用于定义防护传感器的状态。

当防护传感器触发时，CapSense Widget 的检测应在用户代码中以编程方式中断一定的时间。如果防护传感器触发，则表明有水，其他传感器将无法可靠地感应。

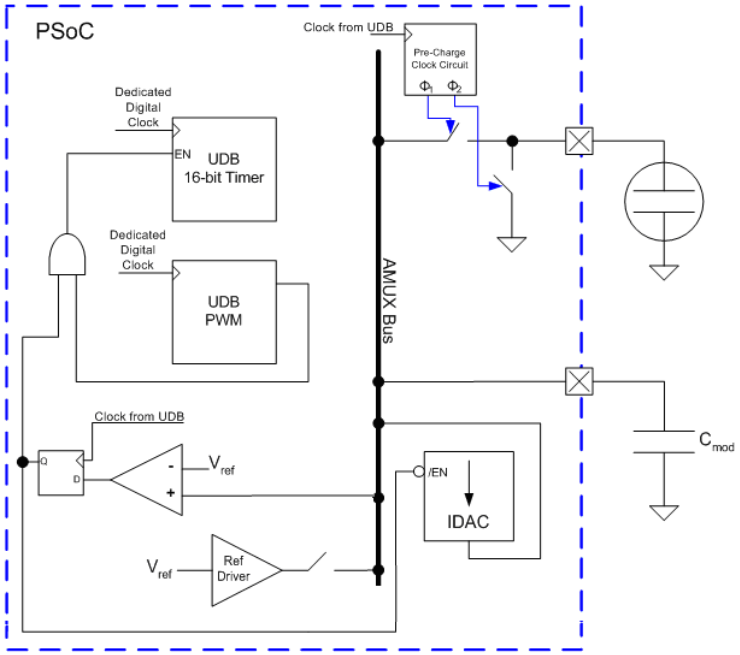
考虑到防护传感器的大小，其信号会与其他传感器的信号有所不同。这说明其表面存在的水可能要多于标准传感器表面的水。因此，在有水珠出现的情况下接收到的信号可能远远强于由手指触摸引起的信号。这将允许设置触发阈值和滤波器，以便防护传感器上的手指触摸不产生任何作用。防护传感器在没有任何专门选项的情况下进行扫描。当防护传感器扫描时，屏蔽电极不会禁用。采用双通道设计的防护传感器始终最后一个进行独立扫描。

框图和配置

使用 Sigma-Delta (CSD)调制器的电容式感应利用开关电容模拟技术和数字 Delta-Sigma 调制器将感应的开关电容电流转换为数字代码。这让按键、滑条、接近检测器、触摸板和使用导电传感器阵列的触摸屏得以实现。高级软件例程让使用双工的滑条分辨率有所提高并使物理和环境传感器变动能够得到补偿。基于基本 CSD 方法的有三种模拟硬件配置。下节有详细描述。

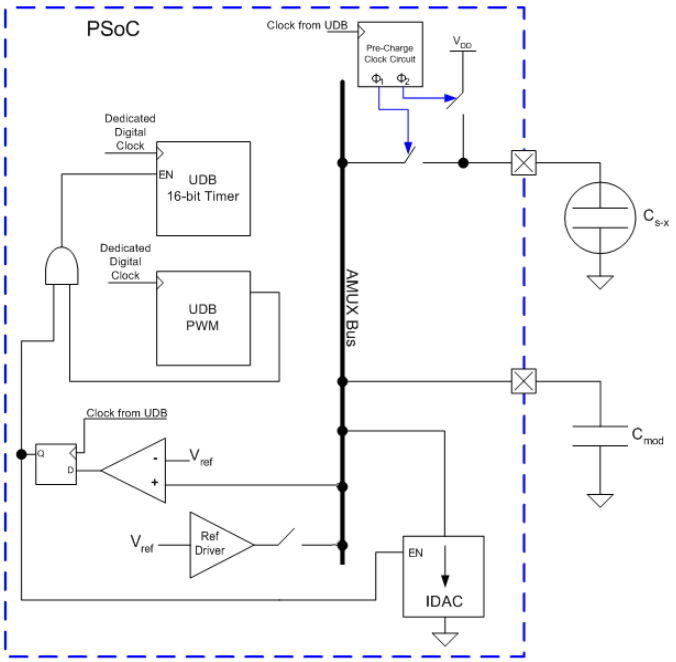
IDAC 源电流

传感器开关级配置为在 GND 和 AMUX 总线之间交替，AMUX 总线与调制电容器相连。在此配置中，IDAC 配置为将电流输送给传感器。



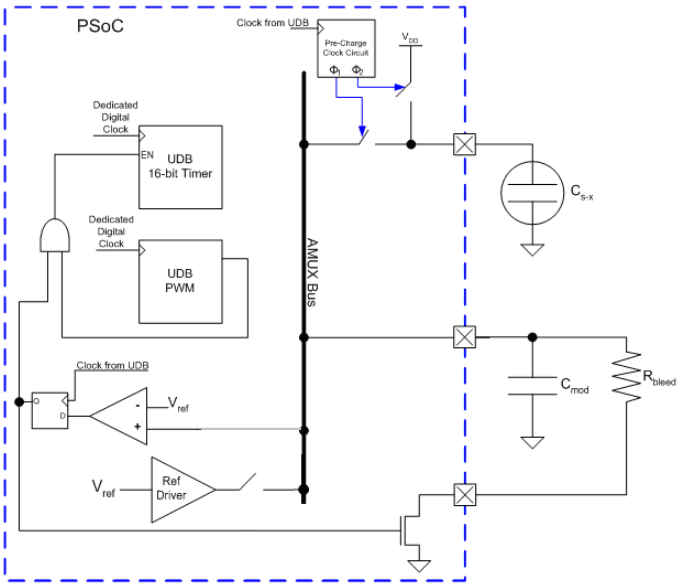
IDAC 灌电流

传感器开关级配置为在 V_{DD} 和 AMUX 总线之间交替，AMUX 总线与调制电容器相连。在此配置中，IDAC 被配置为从传感器的灌电流。



IDAC 禁用，使用外部 Rb

使用外部泄露电阻 Rb 与 IDAC 灌电流配置作用相同，区别在于 IDAC 被接地电阻 Rb 所代替。该泄露电阻以物理方式连接在 C_{MOD} 和 GPIO 之间。GPIO 在“开漏驱低”（Open-Drain Drives Low）的驱动模式下配置。此模式允许 C_{MOD} 通过 Rb 放电。



资源

CapSense CSD 使用数字和模拟资源。 可使用以下通用配置设置估计数字和模拟资源：

- 配置了 7 个传感器，以进行扫描（两个按键及一个包含了五个元素非双工的滑条）
- 模拟开关分频器：UDB 定时器
- PRS 减少电磁干扰（EMI）：已使能的 16 位，全速
- 屏蔽：禁用

数字资源：

配置	资源类型					
	Datapath 单元	宏单元	状态单元	控制单元	DMA通道	中断
通道编号：1 电流源：IDAC灌电流	5	18	1	2	–	1
通道编号：1 电流源：IDAC源电流 Vref – 1.024 V	5	17	1	2	–	1
通道编号：1 电流源：Rb	5	18	1	2	–	1
通道编号：2 电流源：IDAC灌电流	7	29	1	2	–	2
通道编号：2 电流源：IDAC源电流 Vref – 1.024 V	7	27	1	2	–	2
通道编号：2 电流源：IDAC源电流 Vref – VDAC	7	27	1	2	–	2
通道编号：2 电流源：Rb	7	29	1	2	–	2

模拟资源：

配置	资源类型		
	VIDAC	电压比较器	CapSense缓冲器
通道编号：1 电流源：IDAC灌电流	1	1	1
通道编号：1 电流源：IDAC源电流 Vref – 1.024 V	1	1	1
通道编号：1 电流源：Rb	–	1	1
通道编号：2 电流源：IDAC灌电流	2	2	2
通道编号：2 电流源：IDAC源电流 Vref – 1.024 V	2	2	2
通道编号：2 电流源：IDAC源电流 Vref – VDAC	4	2	2
通道编号：2 电流源：Rb	–	2	2

API 存储器占用情况

根据编译器、器件、所使用的 API 数量以及组件的配置情况的不同，组件所用的存储器使用情况也不一样。下表提供了在某一器件配置中的所有 API 使用的存储器使用情况。

通过使用“Release”模式中的相应编译器，可以进行测量操作。在该模式下，存储器的大小得到优化。对于特定的设计，分析编译器生成的映射文件后可以确定存储器的大小。

注意：使用以下配置估计组件存储器的大小：

Widgets：

- 按键：2
- 滑条：一个线性、非双工以及包含五个传感器的

配置	PSoC 3（Keil_PK51）		PSoC 5LP（GCC）	
	闪存字节	SRAM（字节）	闪存（字节）	SRAM（字节）
通道编号：1 电流源：IDAC灌电流	3668	65	2258	75
通道编号：1 电流源：IDAC源电流 Vref – 1.024 V	3658	65	2242	75
通道编号：1 电流源：Rb	3455	65	2030	79
通道编号：2 电流源：IDAC灌电流	5130	69	3474	75
通道编号：2 电流源：IDAC源电流 Vref – 1.024 V	5110	69	3324	75
通道编号：2 电流源：IDAC源电流 Vref – VDAC	5350	71	3722	75
通道编号：2 电流源：Rb	4657	69	3030	79

直流和交流电的电气特性

电源电压

参数	测试条件和注释	最小值	典型值	最大值	*单位*
值	--	2.7	5.0	5.5	V

噪音

参数	测试条件和注释	最小值	典型值	最大值	单位
噪声计数，峰到峰（噪声计数/（基准线计数）	分辨率 = 16（噪声计数/（基准线计数）	–	0.2	–	%
	分辨率 = 14	–	0.3	–	%

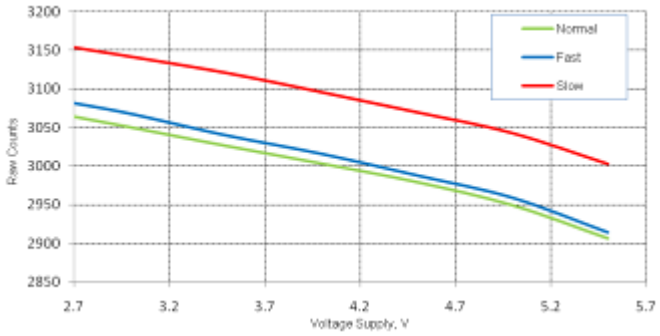
	分辨率 = 10	–	1.0	–	%

功耗

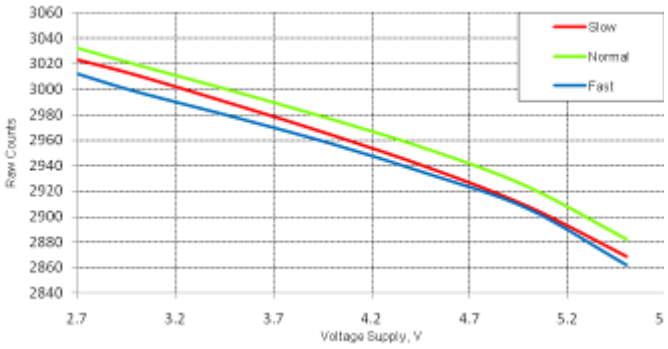
参数	测试条件和注释	最小值	典型值	最大值	单位
有功电流	V_{DD} = 3.3伏，CPU时钟 = 24 MHz，CapSense扫描时钟 = 24 MHz，扫描期间的平均电流，8个传感器	–	8	–	mA
睡眠/唤醒电流有100 ms的报告速率	V_{DD} = 3.3伏，CPU时钟 = 24 MHz，CapSense扫描时钟 = 24 MHz，扫描速度 = 超快，分辨率 = 9，8个传感器	–	78.9	–	µA
	V_{DD} = 3.3伏，CPU时钟 = 24 MHz，CapSense 扫描时钟 = 24 MHz，扫描速度 = 快速，分辨率 = 12，8个传感器	–	484	–	µA
睡眠/唤醒电流的报告速率为1s	V_{DD} = 3.3伏，CPU时钟 = 24 MHz，CapSense 扫描时钟 = 24 MHz，扫描速度 = 快速，分辨率 = 12，1个传感器	–	8.2	–	µA

图 — 原始计数与电源对比

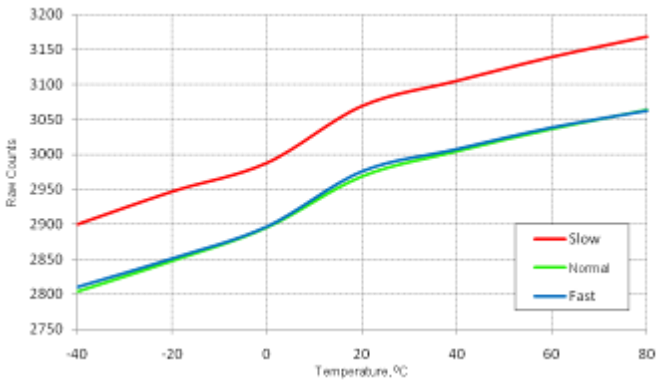
在不同扫描速度下，PRS 16全速，原始计数与电源电压的关系



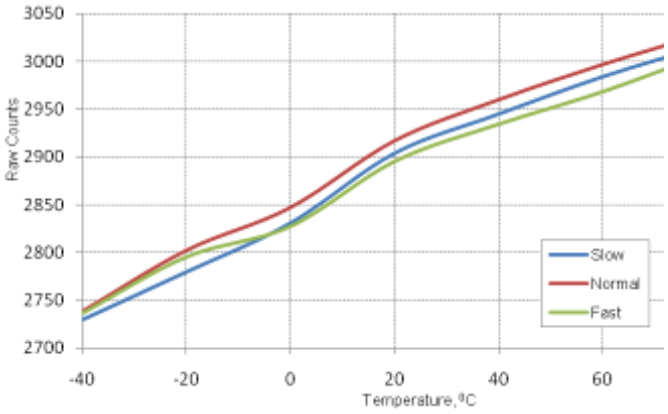
在不同扫描速度下，PRS 8，原始计数与电源电压的关系



在不同扫描速度下，PRS 16全速，原始计数与温度的关系



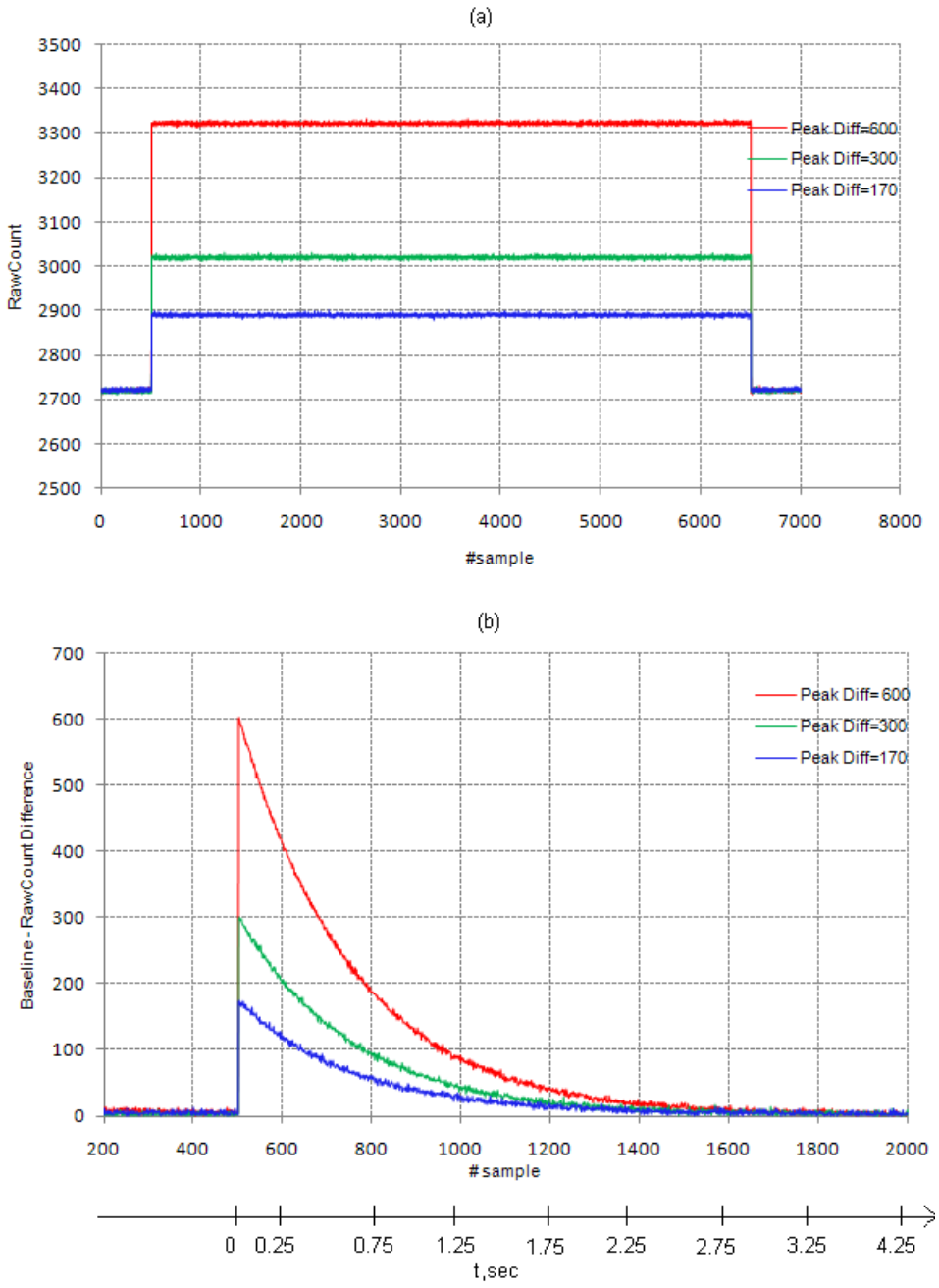
不同速率下，PRS 8，原始计数与温度的关系



不同原始计数步长变化值的基准线随时间的变动

(a) 不同步长值的原始计数步长变化

(b) 原始计数之间的差异



组件勘误表

本节列出了组件的已知问题。

赛普拉斯ID	组件版本	问题	解决方案
191257	v3.40	在没有修正PSoC Creator 3.0 SP1中的版本编号时进行更改这组件。更多信息，请参见基础知识文章KBA94159（网页地址：www.cypress.com/go/kba94159）。	解决方案是不必要的。设计不受任何影响。

组件更改

版本	更改说明	更改原因/影响
3.40.b	编辑数据手册并将其添加到组件勘误章节。	文档的组件被更改，但设计不受任何影响。
3.40.a	对数据手册进行了少量编辑。	对_Start()和_TunerStart()的API添加注释。
3.40	更新了Configure（配置）对话框的GUI界面。	更加易于使用的界面。
	更新了调谐器GUI。	更加易于使用的界面。
	更新了“MISRA合规性”章节。	经过验证后的MISRA合规性。
3.30	添加了MISRA合规性章节。	尚未根据MISRA验证此组件。
3.20	添加了PSoC 5LP支持	
	通过向“.cyre”文件添加函数名称，增加了将每个函数声明为PSoC 3的重新进入的可能性。	并非所有API都是真正的可重入函数。 组件API源文件中的注释指出了不可真正重入的函数。需要此更改为采用安全方式使用（通过标志或关键节防止并发调用）并且不是可重入函数消除编译器警告。
3.10	在某些变量前面加上了实例的名称，这样使它们在每一组件实例中都是唯一的。	当有设计中包含超过一个CapSense CSD组件时，防止变量的冲突。这种冲突可造成错误行为。该修复针对PSoC 3 ES2和PSoC 5。
	更新CapSense_EnableSensor()和CapSense_DisableSensor()函数，以便正确处理Port 15上的引脚。	允许Port 15引脚和CapSense一起使用。没有该修复，Port 15各引脚就不会被使能，且这些Port 15引脚的函数可导致内存损坏。
3.0	添加了CapSense_GetMatrixButtonPos() API函数，用于矩阵按键触摸位置计算。	以前，矩阵按键触摸位置是由用户进行计算的。新的函数简化了这个问题 。
	矩阵按键触摸位置被传输到Tuner，并在相应的GUI widget中进行显示。	因为向矩阵按键中增加了新函数用于位置计算，所以该位置被传输至Tuner。以前，触摸位置是由Tuner从API独立进行计算的。

版本	更改说明	更改原因/影响
	增加了Multiple Analog switch dividers（多模拟开关分频器）选项，提供了为每一扫描槽设置单独模拟开关分频器的功能。 组件能配合单个模拟开关分频器（与以前组件版本相同），也能配合多个模拟开关分频器。	针对不同传感器使用不同的模拟开关分频器，可实现对单个扫描槽的灵活调试。
	模拟开关分频器被传输到Tuner，然后由GUI进行显示。此外，模拟开关分频器还可以从Tuner中进行更改。	允许用户查看和更改传感器的模拟开关分频器。
	允许用户使用不同的IDAC模式 （源电流与灌电流）和Vref值（1.024 V 与 VDAC），从而扩展了自动调试程序的功能。对定制器和固件部分增加了更多的计算结果。	在IDAC灌电流模式中允许自动调试，并允许使用VDAC来产生参考电压。
	更改了使能Tuner时的数据准备和传输程序。	以前，传输数据准备和传输是并行处理的，这在执行字节交换时曾经造成了问题。
	更改了使能自动重置时的基线更新算法	以前，当自动重置功能被使能时，在原始数据跳至基线下很多计数时，基线不能立即调整回到原始数据。 这是可能发生的，比如当用户将手指放在节点上经过很长一段时间，在ON状态使基线缓慢达到原始数据时。在它“逐阶下降”的同时，也会“逐阶上升”。这和PSoC 1 CapSense功能不同。
	更改CapSense_GetTouchCentroidPos() API函数，避免使用全局阵列来存储位置计算。	该函数的上一版本使用了全局阵列，用于触摸板触摸位置存储。它需要额外的存储器，且方便了用户，因为当使用多个触摸板时，用户需要计算阵列索引。在新版中，触摸位置存储的阵列指针是作为本地参数传输给函数的。它允许使用堆栈来进行阵列存储，并避免了使用多个触摸板时进行索引计算的需要。

赛普拉斯半导体公司，2013-2016 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可权）（1）在赛普拉斯特软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。