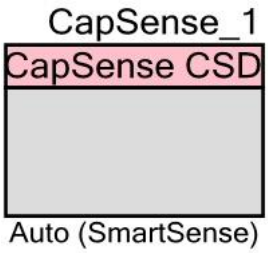
PSoC 4 电容式感应 (CapSense® CSD)

1.10



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
- 支持 SmartSense™ 自动调校或通过集成式 PC GUI 进行手动调校
- 对交流电力线噪音、EMC 噪音和电源电压变动有较强的抗干扰能力
- 支持屏蔽电极，保证传感器上存在水膜或水滴的情况下可靠工作
- 使用 CapSense 配置工具设置传感器的分配

概述

电容式感应使用 Delta-Sigma 调制器（CapSense CSD）组件，能够有效的测量触摸感应按键、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。具体请见赛普拉斯半导体公司网址 www.cypress.com：

- *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、开关和其他控件，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。

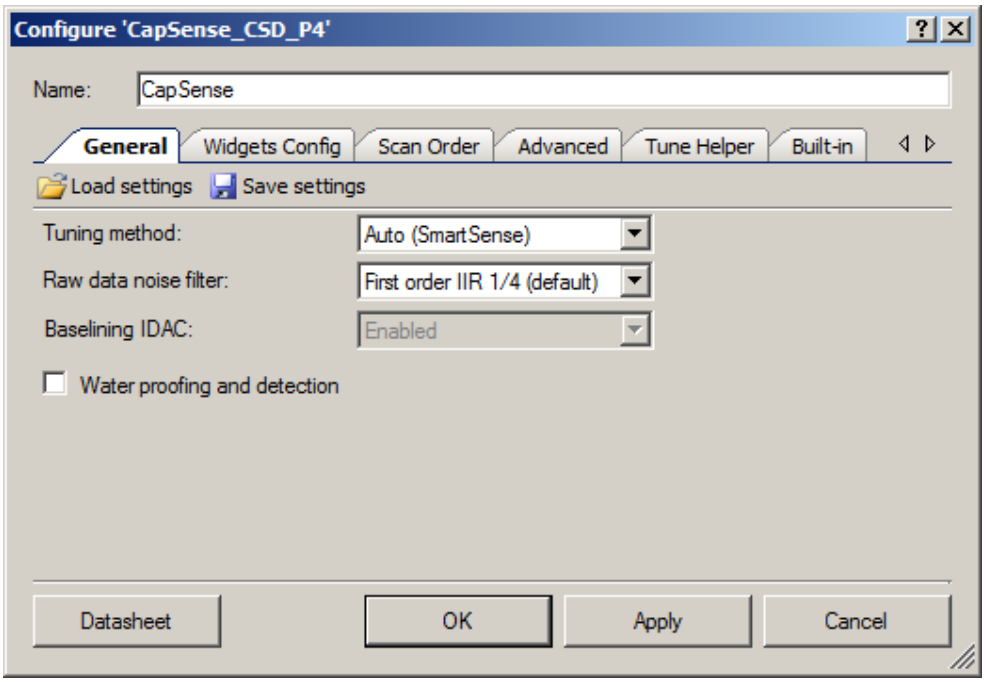
输入/输出接口

N/A

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 选项卡



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

Save Settings 用于保存组件当前的所有配置和调校数据。这样，在新项目中就能够快速复制已保存设置到新的模块。**Load Settings** 用于加载以前保存的设置。

被保存的组件配置和调校数据可以导入调谐器 GUI。

Tuning method（调校方法）

此参数指定了调校方法。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校方法。运行时，固件算法不断地确定最佳的调校参数。该模式需要额外的 RAM 和 CPU 资源。

  注意 — 可使用 I2C 接口进行 SmartSense 调校，该组件在 **Tuner Helper**（调谐器助手）选项卡中指定用于将目标设备中的数据传输到调谐器 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，请右击组件图标，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。手动调校需要 I2C 通信，该通信在 **Tuner Helper** 选项卡中指定用于在目标设备和调谐器 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。

原始数据（Raw Data）噪声滤波器

此参数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。关于滤波器类型的详情，见本文档功能描述一节的滤波器部分。

- **None** — 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。

Baselining（基准线）IDAC

该参数用于使能基准线 IDAC 。使能基准线 IDAC 能够提高灵敏度和信噪比（SNR）。基准线 IDAC 在 CapSense 操作期间一直被连接到 amuxbus 上，用于补偿传感器寄生电容。

- 禁用（默认）
- 启用

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）。启用此功能需要使能屏蔽电极。此功能设置以下参数：

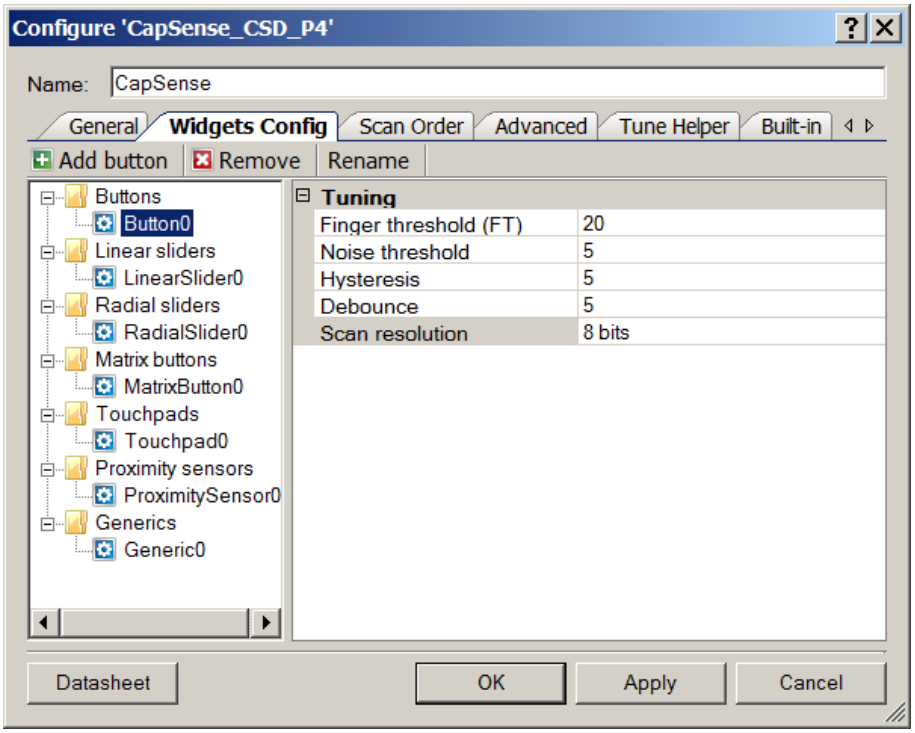
- 使能屏蔽电极输出

  注意：不建议在 SmartSense 调校模式下使用屏蔽电极。

- 添加“保护”传感器

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

Widget Config（Widget 配置）选项卡



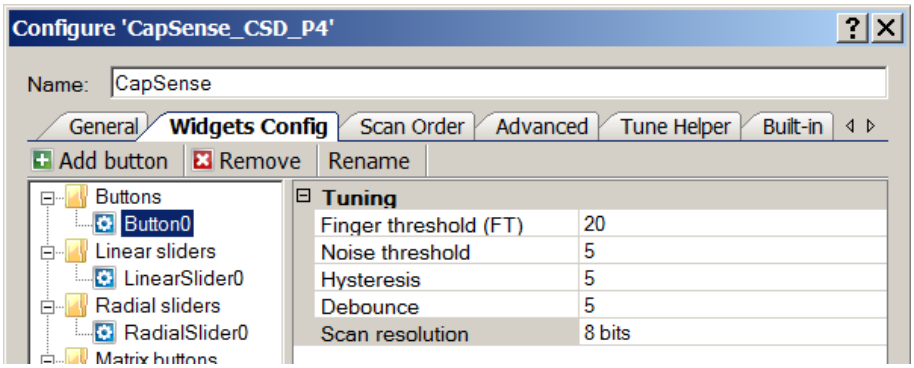
有关各种参数的定义，请参见功能描述一节。

Toolbar（工具栏）

该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) — 按键检测到单个传感器上存在手指触摸，并提供单个机械按键替代。
  - **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**（接近传感器）— 接近传感器经过优化，可在离传感器很远的距离检测是否存在手指、手掌或其他大物体。这能够避免实际接触的需要。
  - **Generic Sensors**（通用传感器）— 通用传感器提供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，而这是输出经过处理的其他类型传感器所不能实现的。
- **Remove widget**（删除 Widget）（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。

按键



Tuning（调校）：

- **Finger Threshold**（手指阈值）— 定义相关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，该按键报告为被触摸。默认值为 **100**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。8 位 widget 分辨率的 **Finger Threshold** + **Hysteresis** 的值不能大于 254，16 位 widget 分辨率的不能大于 65534 。

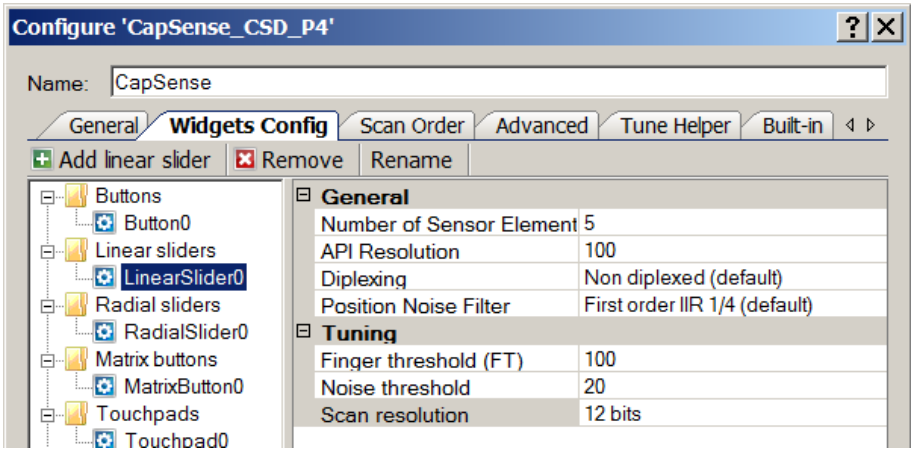
- **Noise Threshold**（噪声阈值）— 定义传感器噪声阈值。如果计数值高于此阈值，则不更新基准线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致触摸错误或触摸遗漏。如果噪声阈值太高，手指触摸可能会被解释为噪声，导致基准线错误地升高，导致手指触摸遗漏。默认值为 **20**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。

- **Hysteresis**（迟滞）— 添加传感器活动状态转换的差分 hysteresis（迟滞）。如果传感器处于非活动状态，则差值计数必须大于 finger threshold(手指阈值)加 hysteresis(迟滞)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 finger threshold(手指阈值)减 hysteresis(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 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。 **Finger Threshold** + **Hysteresis** 不可超过 254。

- **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不活动状态切换到活动状态，在规定的样本数量内，差异计数值必须保持在手指阈值加迟滞值之上。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 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。

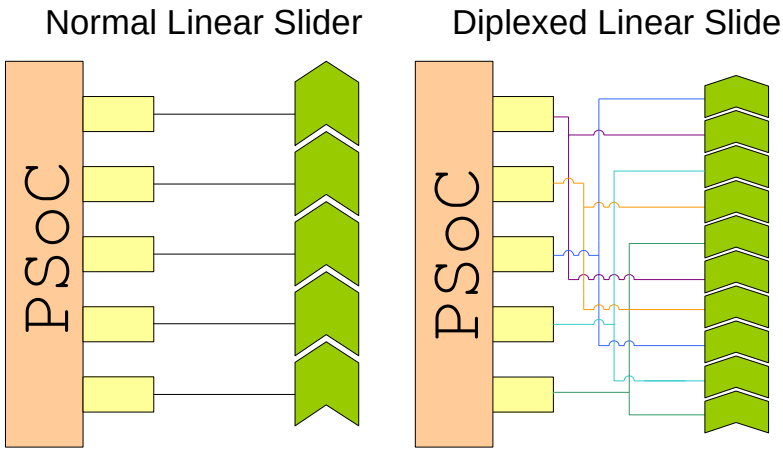
- **Scan Resolution**（扫描分辨率）— 定义扫描分辨率。此参数对按键 Widget 传感器的扫描时间会产生影响。N 位的扫描分瓣率最大原始计数为 $2^N - 1$。提高分辨率可提高触摸检测的灵敏度和信噪比（SNR），但会增加扫描时间。默认值为 **10 位**。

Linear Sliders（线性滑条）



General（常规）：

- **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。值的有效范围为[2…32]。默认值为 **5** 个元件。
- **API Resolution**（API 分辨率）— 定义滑条分辨率。位置值将在此范围以内变化。值的有效范围为[1…255]。
- **Diplexing**（双工）— **Non diplexed**（非双工）（默认）或 **Diplexed**（双工）。双工允许两个滑条传感器共享一个器件引脚，减少给定数量的滑条传感器所需的引脚总数。



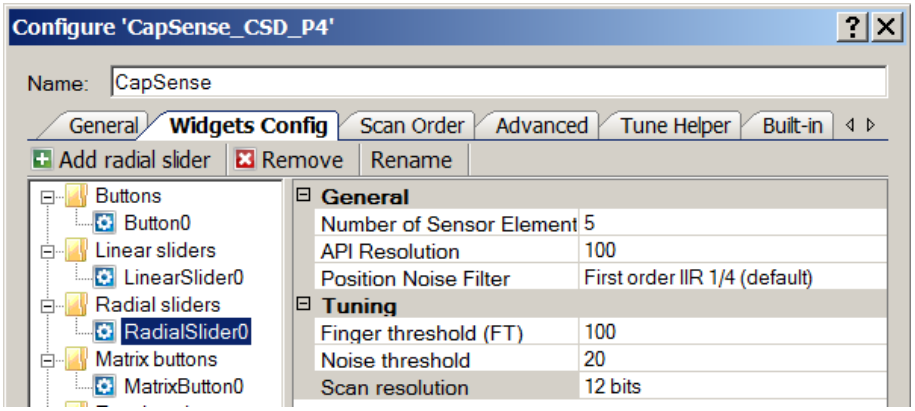
- **Position Noise Filter**（位置噪声滤波器）— 选择噪声滤波器类型以进行位置计算。对于一个选定的 Widget，仅可应用一个滤波器。有关滤波器类型的详情，见本文档功能描述一节的滤波器部分。

- **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（调校）：

- **Finger Threshold**（手指阈值）— 定义相关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，该按键报告为被触摸。默认值 **100**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。
- **Noise Threshold**（噪声阈值）— 定义滑条元素的传感器噪声阈值。如果计数值高于此阈值，则不更新基准线。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。如果噪声阈值太高，手指触摸可能会被解释为噪声，且人为增加基准线，导致质心位置计算错误。质心计算中不考虑低于此阈值的计数值。默认值为 **20**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。
- **Scan Resolution**（扫描分辨率）— 定义扫描分辨率。此参数对线性滑块 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**。

辐射滑条



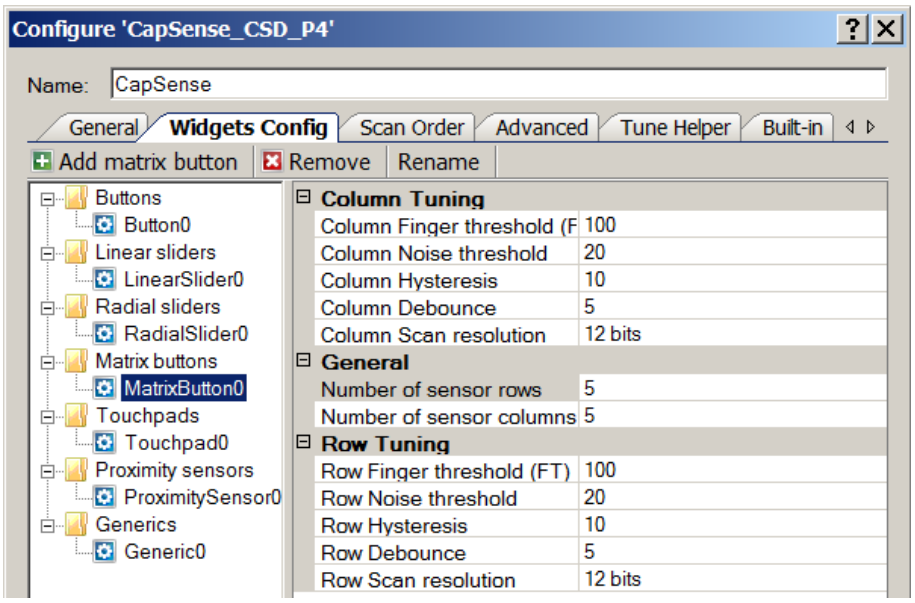
General（常规）：

- **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。值的有效范围为[2…32]。默认值为 **5** 个元件。
- **API Resolution**（API 分辨率）— 定义滑条分辨率。位置值将在此范围以内变化。值的有效范围为[1…255]。
- **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。关于滤波器类型的详情，见本数据手册功能描述一节的滤波器部分。
  - 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（调校）：

- **Finger Threshold**(手指阈值）— 定义相关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，该按键报告为被触摸。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。如果计数值高于此阈值，则不更新基准线。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。如果噪声阈值太高，手指触摸可能会被解释为噪声，且人为增加基准线，导致质心位置计算错误。质心计算中不考虑低于此阈值的计数值。默认值为 **20**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。
- **Scan Resolution**（扫描分辨率）— 定义扫描分辨率。此参数对 Radial slider widget(辐条滑块 widget) 内所有传感器的扫描时间会产生影响。N 位的扫描分瓣率最大原始计数为 $2^N - 1$。提高分辨率可提高触摸检测的灵敏度和 SNR，但会增加扫描时间。默认值为 **10 位**。

矩阵按键



Tuning（调校）：

- **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此阈值时，该按键报告为被触摸。默认值为 **100**。8 位 widget 分辨率的有效值范围是[1..65535]，16 位 widget 分辨率的有效值范围是[1..65535]。8 位 widget 分辨率的 **Finger Threshold** + **Hysteresis** 的值不能大于 254，16 位 widget 分辨率的不能大于 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。如果计数值高于此阈值，则不更新基准线。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。如果噪声阈值太高，手指触摸可能会被解释为噪声，且人为增加基准线。这样会导致触摸遗漏。默认值为 **20**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[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。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。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 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1…65535]。8 位 widget 分辨率的 **Finger Threshold** + **Hysteresis** 不能大于 254，16 位 widget 分辨率的不能大于 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到激活状态，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的和。默认值为 **5**。Debounce（去抖动）确保高频率高

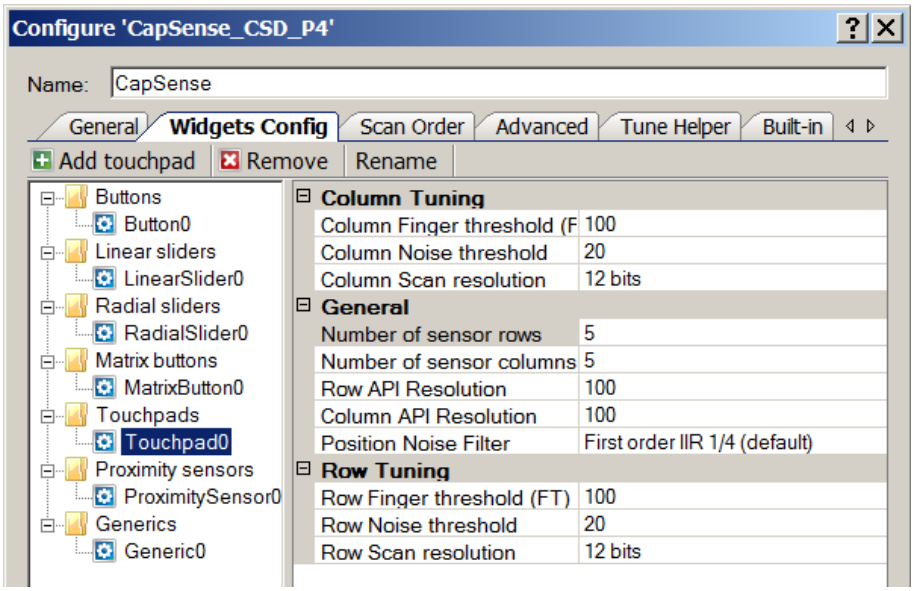
振幅的噪声不会导致对按键的误检。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。

- **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。 此参数对矩阵按键 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，但会增加扫描时间。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（常规）：

- **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。值的有效范围为[2…32]。列和行的元素数量默认值为 **5** 个。

触控板



Tuning（调校）：

- **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此阈值时，触摸板报告触摸位置。默认值为 **100**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[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。如果计数值高于此阈值，则不更新基准线。质心位置计算中不考虑低于此阈值的计数值。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致触摸错误或触摸遗漏。如果

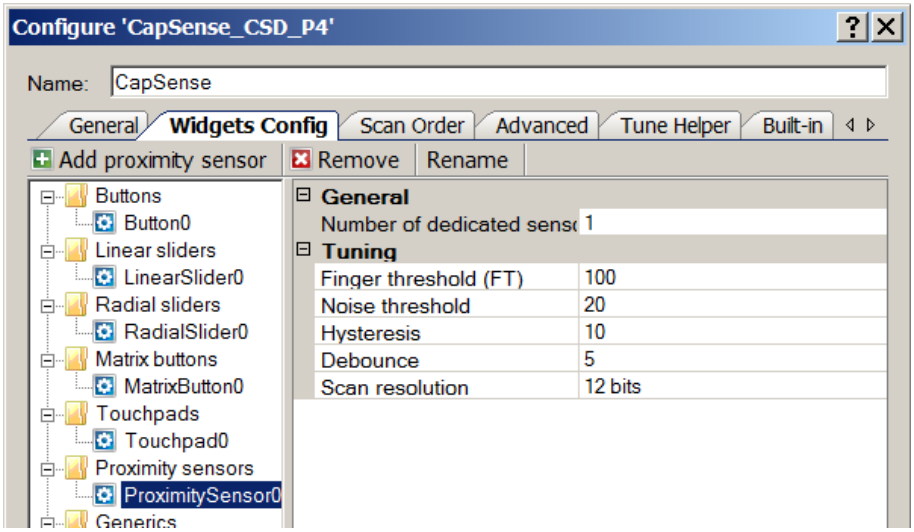
噪声阈值太高，手指触摸可能会被解释为噪声，且人为增加基准线。这样可造成质心计算错误。默认值为 **20**。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。

- **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。此参数对触摸板 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，但会增加扫描时间。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（常规）：

- **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。值的有效范围为 [2…32]。列和行的元件数量默认值均为 **5** 个。
- **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。值的有效范围为[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。关于滤波器类型的详情，见本数据手册功能描述一节的滤波器部分 。
  - 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）

Proximity Sensors（接近传感器）



注意：除接近 Widget 以外，所有 Widget 都在默认情况下启用。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 中手动启用接近 Widget。使用 *CapSense_EnableWidget ()*函数以启用接近 widget。

General（常规）：

- **Number of Dedicated Sensor Elements** — 选择专用接近传感器的数量。这些传感器单元不包括用于其他类型的 Widget 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传感器。
  - **0** — 接近传感器仅扫描一个或多个现有传感器以确定接近情况。没有针对此 Widget 分配新的传感器。
  - **1**（默认）— 系统中专用接近传感器的数量。

Tuning（调校）：

- **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活动阈值。当传感器扫描值大于此阈值时，该接近传感器报告为被触摸。默认值为 **100**。8 位 widget 分瓣率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。8 位 widget 分瓣率的 **Finger Threshold** + **Hysteresis** 的值不能大于 254，16 位 widget 分瓣率的不能大于 65534。
- **Noise Threshold**（噪声阈值）— 定义传感器噪声阈值。如果计数值高于此阈值，则不更新基准线。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。如果噪声阈值太高，手指触摸可能会被解释为噪声，且人为增加基准线。这样会导致触摸遗

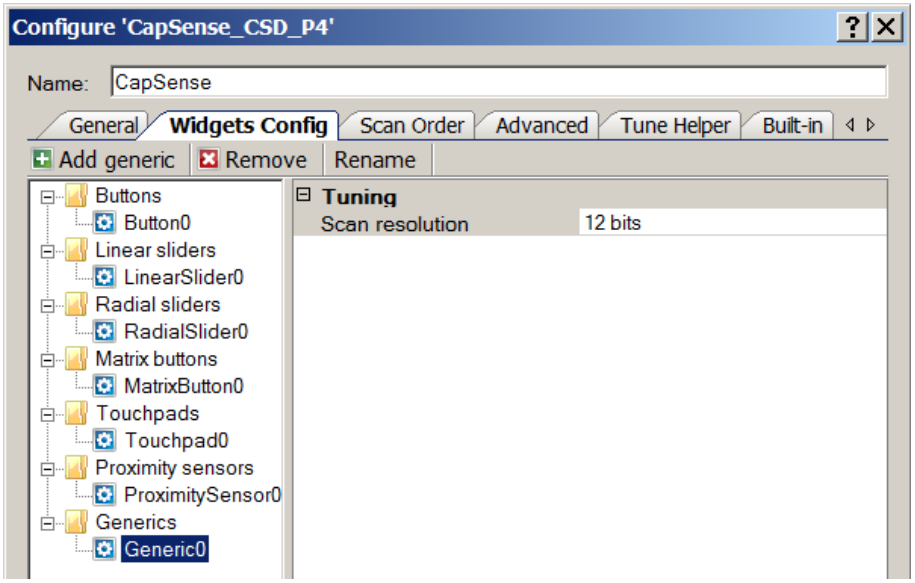
漏。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。

- **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。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。迟滞有助于确保低振幅传感器噪声和手指和身体的小幅移动不会导致接近传感器状态的循环。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[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。8 位 widget 分辨率的有效值范围是[1…255]，16 位 widget 分辨率的有效值范围是[1..65535]。

- **Scan Resolution**（扫描分辨率）— 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。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）

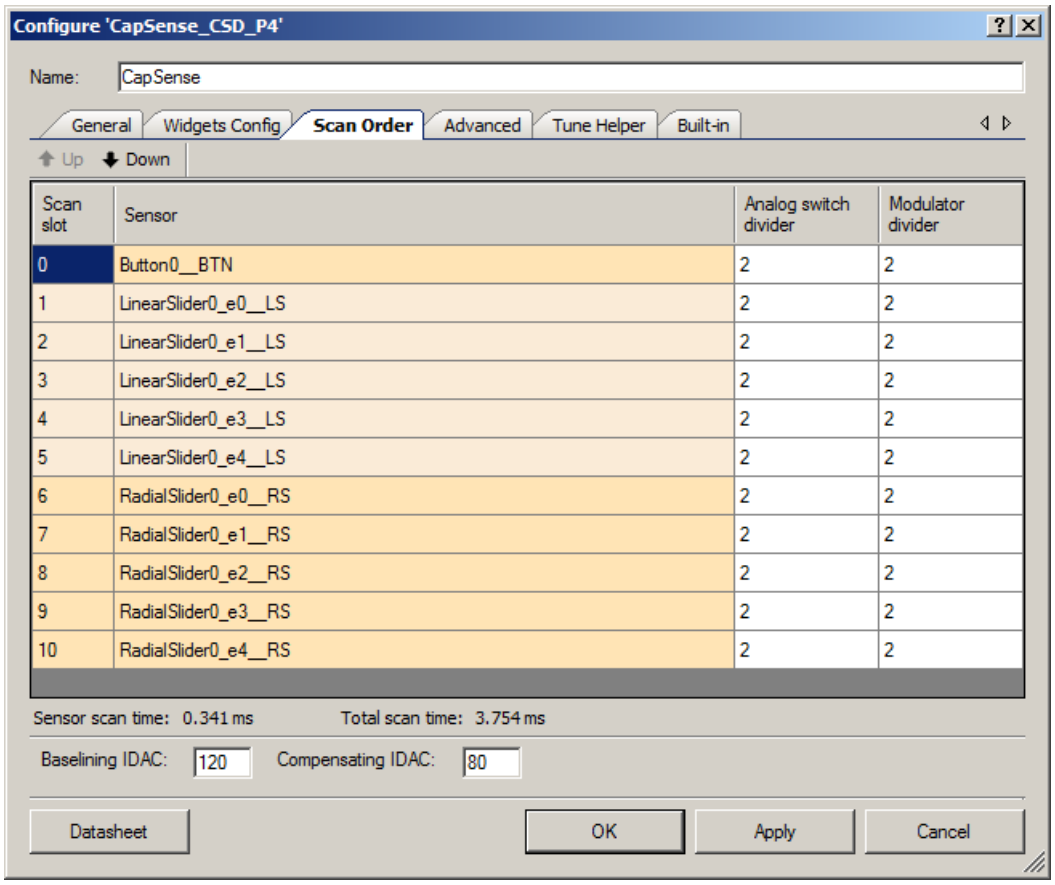


Tuning（调校）：

- **Scan Resolution**（扫描分辨率）— 定义扫描分辨率。此参数对通用 Widget 的扫描时间产生影响。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



Toolbar（工具栏）

该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：接近传感器默认从扫描过程中排除。其扫描必须在运行时手动启动，因为其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 generic and proximity widgets)。

Analog Switch Divider(模拟开关分频器)列

指定 **Analog Switch Divider (**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[2…255]。默认值为 12。

如果 **Individual frequency setting**（单一频率设置）（位于 **Advanced** 选项卡）被禁用，则该列是隐藏的。

Modulator Divider（调制器分频器）列

指定 **Modulator Divider** 值，并确定扫描槽的调制器输入频率。值的有效范围为[2…255]。默认值为 12。

如果 **Individual frequency setting**（单一频率设置）（位于 **Advanced** 选项卡）被禁用，则该列是隐藏的。

基准线 IDAC 值

指定基准线 IDAC 值。4x 范围的有效范围介于 0 至 250 之间，8x 范围的有效范围介于 0 至 125 之间。默认值为 **200**。

补偿 IDAC 值

指定补偿 IDAC 值。有效范围介于 0 至 127 之间。默认值为 **80**。

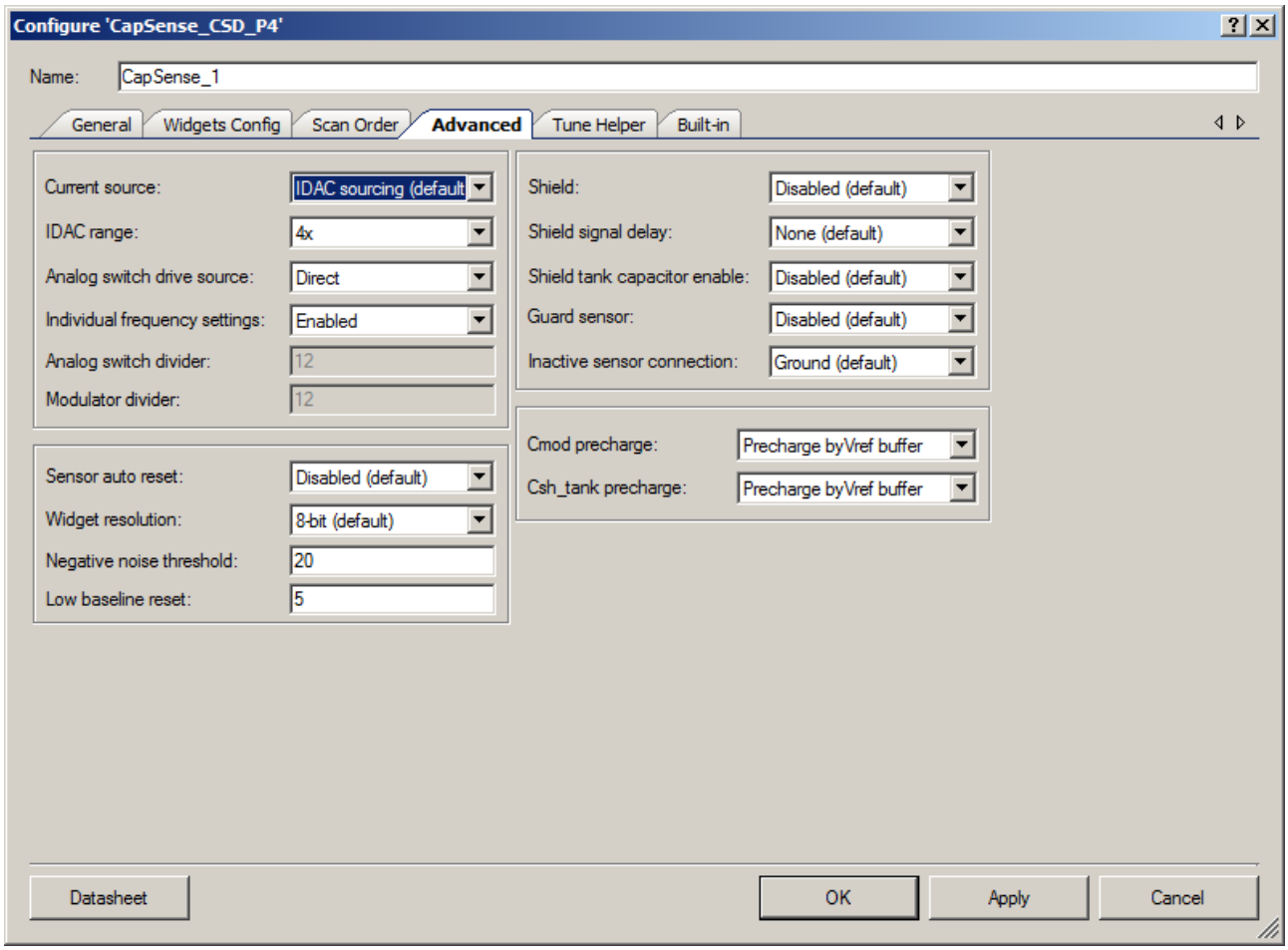
Widget 列表

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，以凸显不同的 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，电路板可能有一根布线，整个将按键阵列围绕起来，且接近传感器可能由此布线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，可选择一个或多个传感器进行扫描，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、任何其他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



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 Sinking** 和 **IDAC Sourcing** 需要使用 PSoC 器件上的 IDAC。

- **IDAC Sourcing**（默认）— IDAC 将电流提供到调制电容 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。

IDAC range（IDAC 范围）

此参数指定**电流源**的 IDAC 范围如果将 **Current Source** 项设置为 **External Resistor**，则禁用此参数。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。

- 4x（默认值）
- 8x

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。

- 直接（默认值）
- PRS-8b
- PRS-12b
- PRS-Auto

Individual Frequency Settings（单一频率设置）

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

Analog Switch Divider (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[2…255]。默认值为 **12**。

如果 **Individual Frequency Settings**（单一频率设置）**启用**，则该功能不可用。

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。**模拟开关分频器**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。

Modulator Divider（调制器分频器）

此参数用于指定调制器分频器的值，并确定调制器输入频率。值的有效范围为[2…255]。默认值为 **12**。

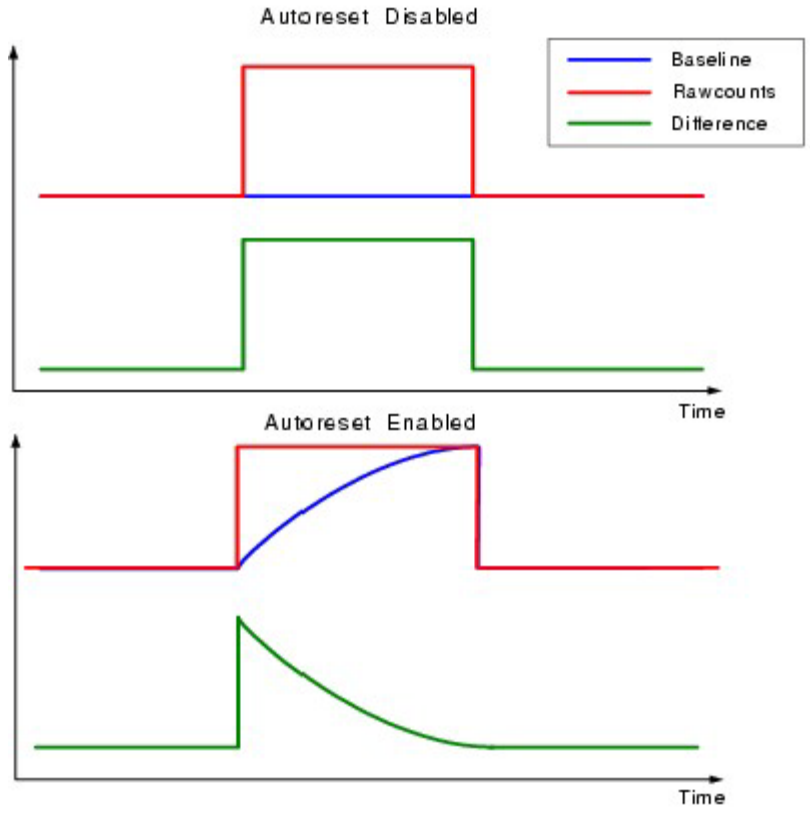
如果 **Individual Frequency Settings**（单一频率设置）**启用**，则该功能不可用。

注意：“Modulator Divider”（调制器分频器）应是“Analog Switch Divider”（模拟开关分频器）的整数倍，因为这些分频器是相连的。有关详细信息，请参见“CapSense 时钟”一节。

Sensor Auto Reset（传感器自动复位 ）

此参数启用自动复位，从而使基准线随时更新，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，基准线才会更新。除非是遇到在无任何物体未触碰传感器而原始计数突然上升时传感器始终打开的问题，否则应将此参数保留为**禁用**。

- **Enabled**（启用）— 自动复位确保基准线随时更新，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），但是当无任何物体触碰传感器而原始计数突然上升时，可以阻止传感器始终打开。原始计数突然上升可能是由电源电压剧烈波动、高能射频噪声源或温度快速变化所导致。
- **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准线停止更新从而造成按键按压遗漏或按键卡住。其优势是，按键可以永远继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，应当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，但条件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）

本参数指出复位到原始计数（raw count）级别的基线，其原始计数（raw count）和基线级别之间的负的差值。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准线到原始计数（raw count）必须经过的采样次数，这种采样中原始计数（raw count）必须小于基准线值。值的有效范围为[1…255]。默认值为 5。

Shield（屏蔽电极）

此参数指定启用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，请参考屏蔽电极一节。

- 禁用（默认）
- 启用

Shield signal delay（屏蔽信号延迟）

此参数用于指定 CSD Shield 相对于 csd_sense 的延迟 HFCLK 周期数。

- 无（默认）
- 1 个周期
- 2 个周期

Csh_tank（屏蔽槽电容）**启用**

此参数用来指定是否启用将片外 Csh_tank 电容与屏蔽电容并联的引脚。此电容用来增加屏蔽电容。同样，将 Cmod 预充电或 Csh_tank 预充电设置为“通过 IO 缓冲进行预充电”时，也要启用 Ctank 电容。

- 禁用（默认）
- 启用

Guard Sensor（保护传感器）

此参数可启用防护传感器，帮助检测需要防水的应用中的水滴。如果选中 **Water Proofing and detection**（防水并检测）（在 **General** 选项卡下），则启用此功能。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）
- 启用

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**（接地）(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扫描传感器上的信号振幅。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**屏蔽**被**禁用**，则此功能不可用。

Cmod precharge（Cmod 预充电）

此参数用来指定用于 Cmod 电容的预充电源。

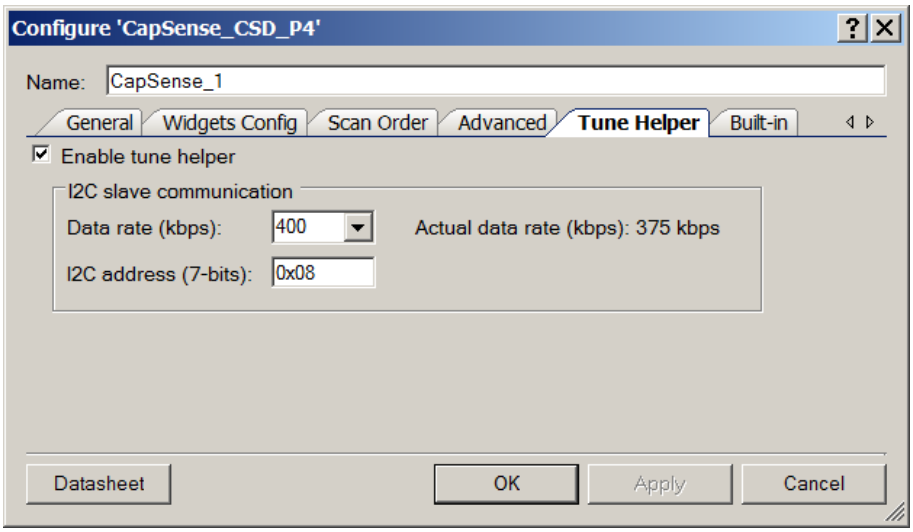
- 通过 Vref 缓冲进行预充电（默认）
- 通过 IO 缓冲进行预充电

Csh_tank precharge（Csh_tank 预充电）

此参数用来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（启用调谐助手）

此参数添加多种功能，使用户可轻松与调谐器 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。

Data rate（数据速率）

此参数用于设置 I^2C 数据速率值（可达 400 kbps）；实际速度可能会基于可用的时钟速度和分频器范围而有所不同。标准数据速率为 50、100 和 400（默认）kbps。如果 **Enable Tune Helper** 被禁用，则此参数将被忽略。

实际数据速率

实际数据速率显示了将用当前设置运行的通信组件上的数据速率。所选数据速率可能和实际数据速率不同。影响实际数据速率计算的因子为：HFCLK 时钟和内部组件时钟的精度。

I2C 地址（7 位）

这是从器件将识别的 I^2C 地址。您可以选择介于 0 至 127 之间的从器件地址，默认值为 **8**。此地址为右对齐的 7 位从器件地址，它不包括读/写位。

您可以输入十进制或十六进制的值；对于十六进制数值，请在地址之前键入“0x”。如果 **Enable Tune Helper** 被禁用，则此参数将被忽略。

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。

The CapSense 调谐器在手动调试模式下，根据特定的系统环境帮助调试 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 调校过程

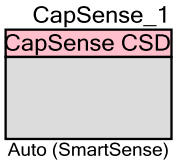
使用和调试 CapSense 组件的典型过程如下：

在 PSoC Creator 中创建设计

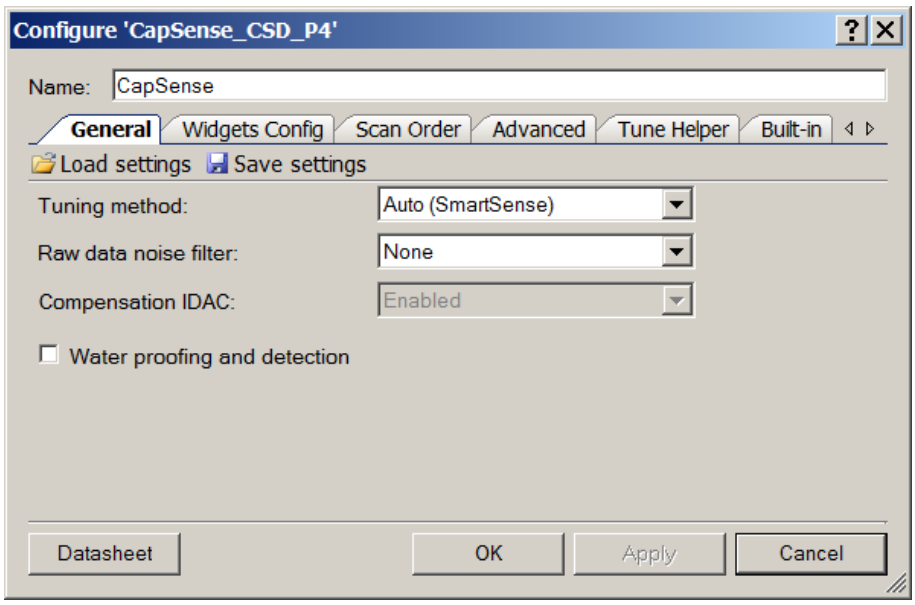
若有需要，可以参考 PSoC Creator 帮助。

放置并配置 CapSense 组件

1. 将 **CapSense_CSD** 组件从组件目录拖动到您的设计中。



2. 双击可打开 **Configure** 对话框。
3. 根据您应用的需要更改 CapSense CSD 参数。将 **Tuning method** 选择为 **Manual** 或 **Auto**（**SmartSense**）。单击 **OK** 关闭对话框并保存选定的参数。



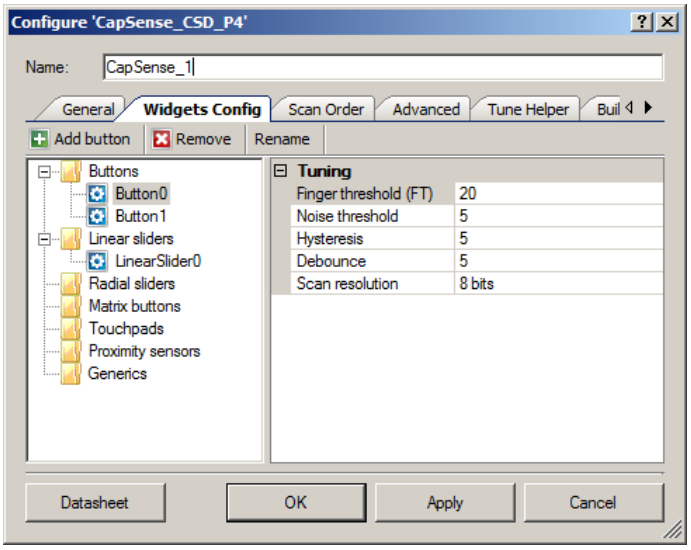
选择“自动(SmartSense)”

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。CapSense CSD 参数在运行时通过固件计算。此模式中使用了额外的 RAM 以及 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以及重复过程的问题，确保系统正确工作。选择“自动(SmartSense)”调校以下 CSD 参数：

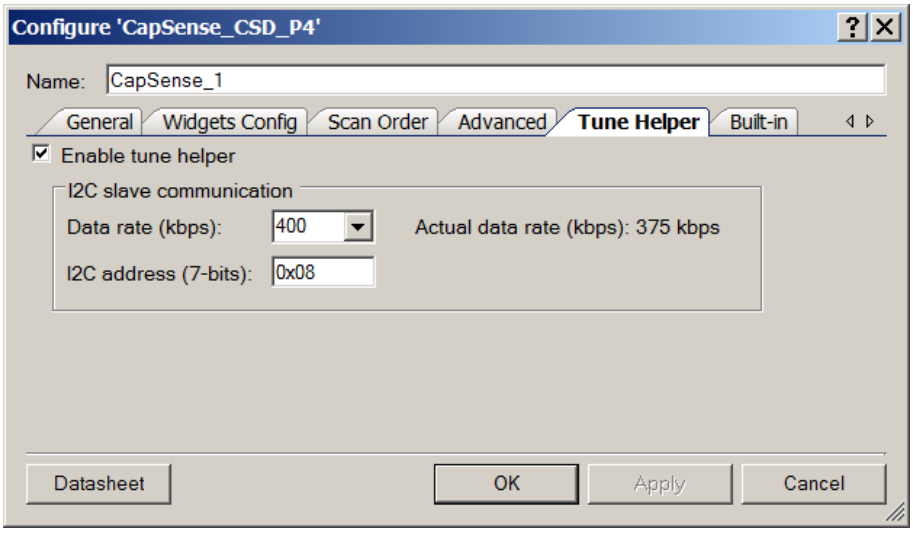
参数	计算
手指阈值	在传感器扫描期间连续计算。
噪声阈值	在传感器扫描期间连续计算。
Baselining(基准线)IDAC	在CapSense CSD启动时计算一次。
补偿 IDAC	在CapSense CSD启动时计算一次。
模拟开关分频器	在CapSense CSD启动时计算一次。
调制器分频器	在CapSense CSD启动时计算一次。
负噪声阈值	在CapSense CSD启动时计算一次。
低基线复位	在CapSense CSD启动时计算一次。

配置用户 CapSense 组件

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



2. 在 **Tune Helper** 选项卡上：必须选中 **Enable Tune Helper** 复选框。



添加代码

- 将调谐器初始化及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)
{
```

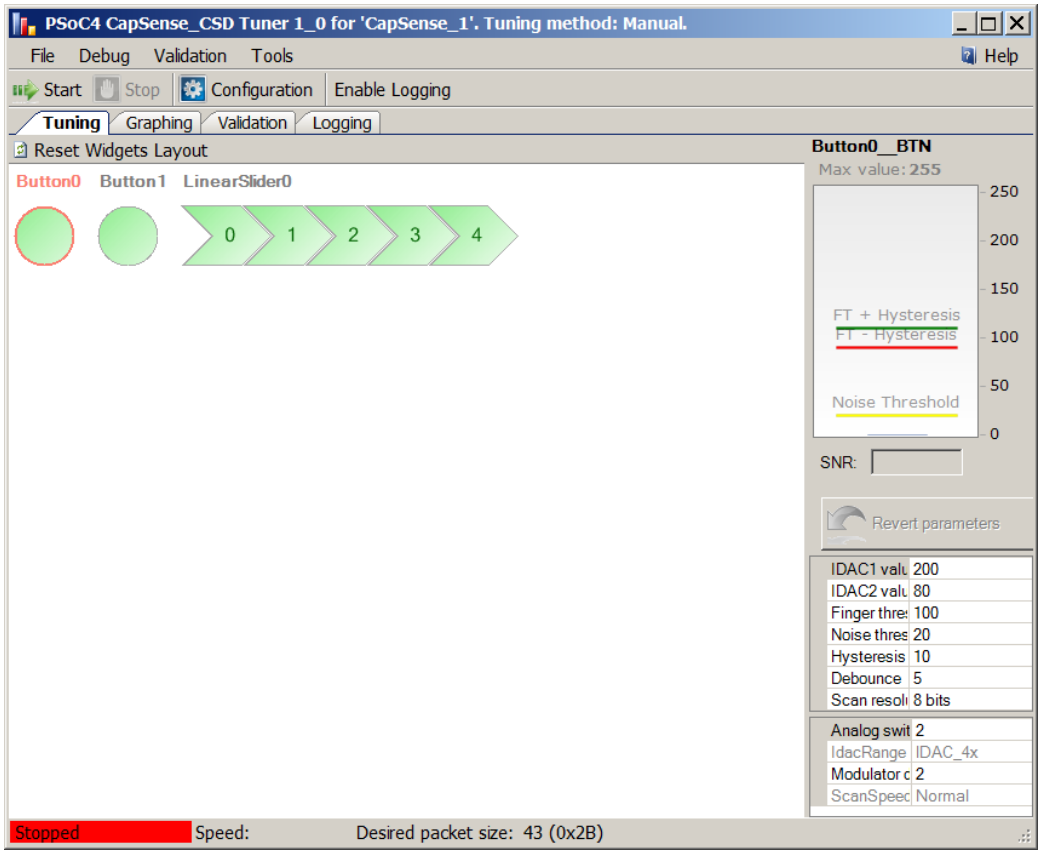
```
    CapSense_1_TunerComm();
    }
    }
```

构建设计并对 PSoC 器件进行编程

根据需要参考 PSoC Creator 帮助。

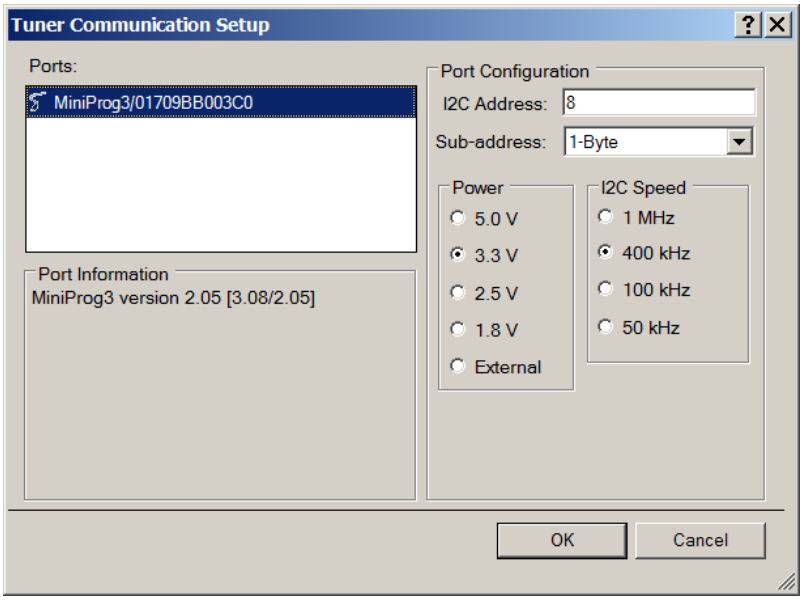
启动调谐器应用

- 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**。

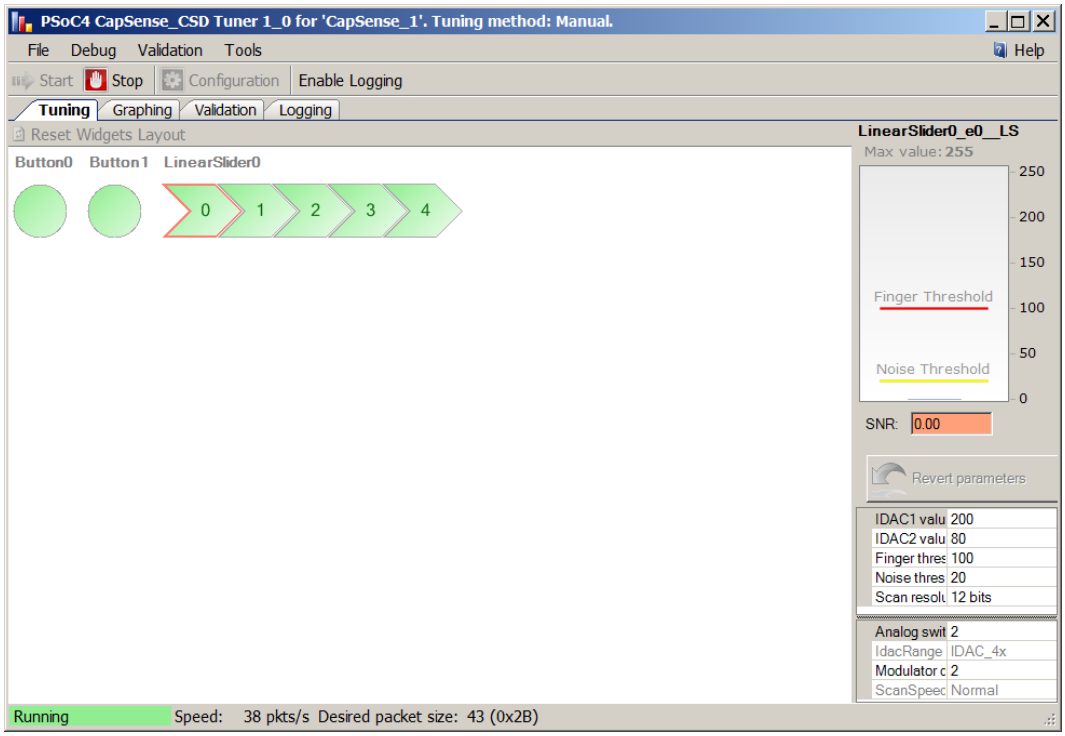
重要说明：属性必须与 **Tune Helper** 选项卡中的完全一致：**I2C Speed**、**I2C Address**。必须将 **Sub-address** 设置为 **1-Byte**。

开始调试

- 点击调试 GUI 上的 **Start**。所有 CapSense 元件开始显示其值。

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，其发生了改变。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

- 根据需要重复步骤，直到调节完成为止，且 CapSense 组件提供可靠的触控传感器结果。

关闭调谐器应用

- 依次选择 **File->Apply Changes and Close**，参数回写到 CapSense_CSD 实例。调谐器应用对话关闭。

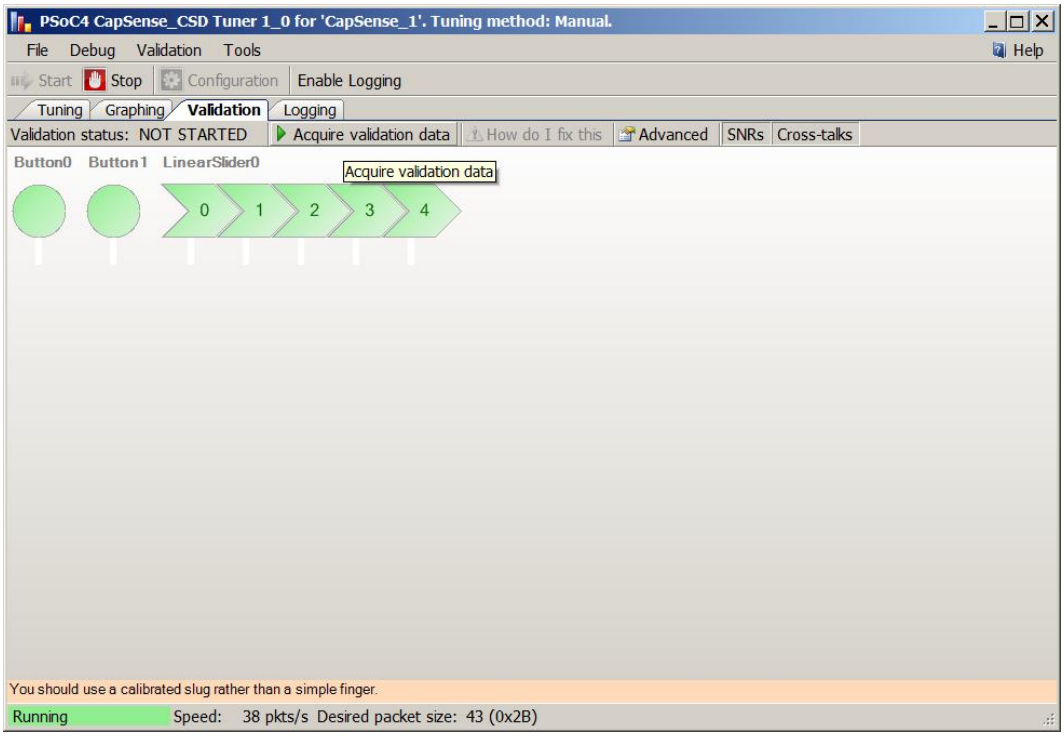
CapSense 验证过程

验证机制确定电路板是否已经过充分调试。下面是使用（调谐器验证）特性来验证 CapSense 设计的典型过程。

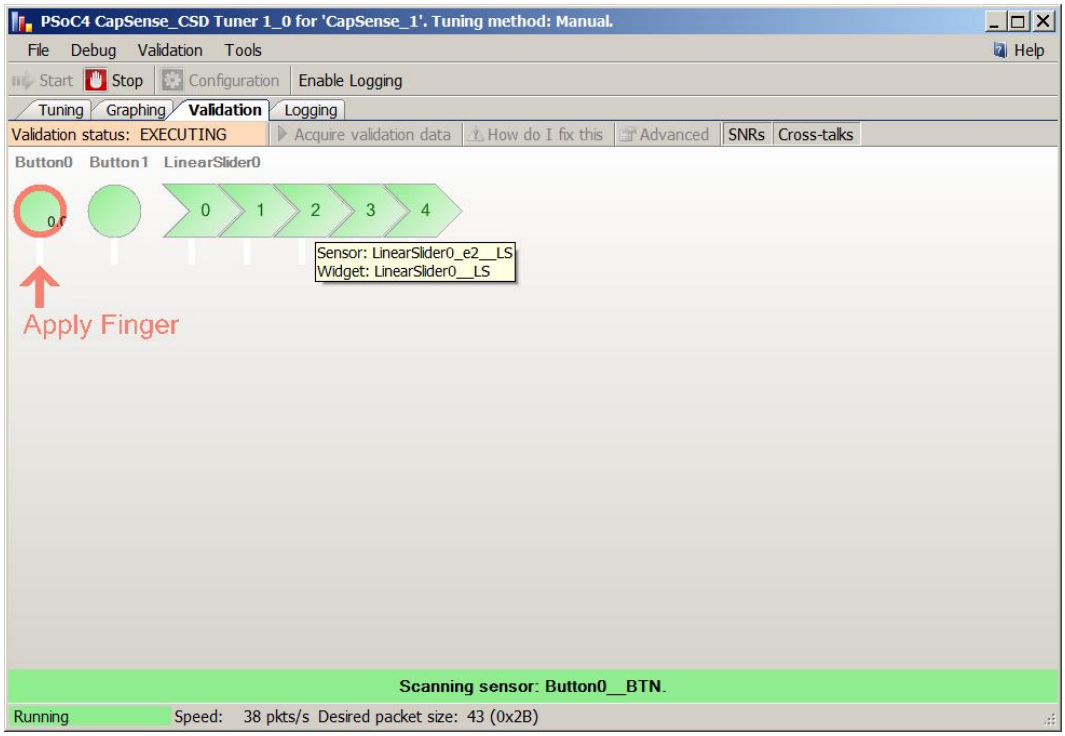
开始验证

1. 开始扫描前，必须准备好“Tuner”和硬件。请参见前面章节，CapSense 调校过程，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

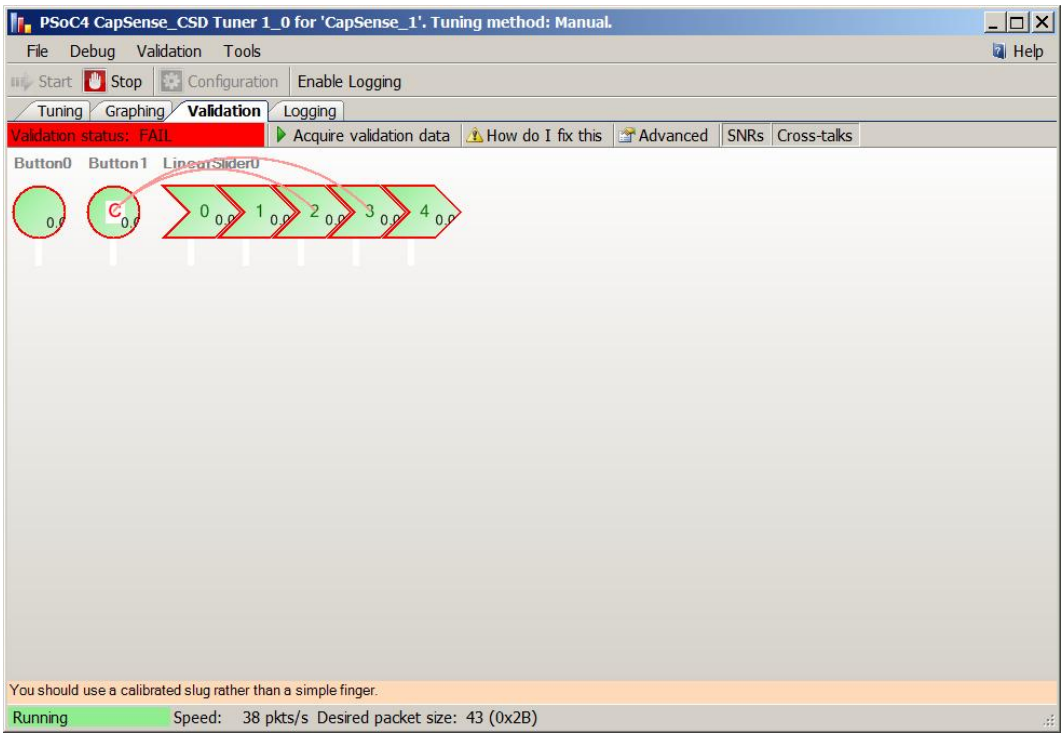


会有屏幕提示你在各传感器上用手指触摸。每次提示您按压 CapSense 元件时，会有闪烁的红色箭头出现在布局图上指向目标，并有文字提示 **PRESS HERE**（按这里）。Tuner 下方出现的文字将指导用户完成验证过程。

- 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**高亮区围绕信噪比小于**足够值**的任何 CapSense 传感器。
- **闪烁黄色**高亮区围绕信噪比介于**足够值**和**最优值**之间的任何 CapSense 传感器。
- **闪烁绿色**高亮区围绕信噪比高于**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）随着各单独串扰检查的进行，软件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，该数值等于**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**（最差情形串扰传感器计数）是 2，则 **Worst Case Crosstalk**（最差情形串扰）计算结果将为(5 + 4 = 9)。

  如果该值超过 **Worst Case Crosstalk Threshold**（最差情形串扰阈值），则会在所显示传感器中间用**闪烁的字符 “C”**对其进行标记。

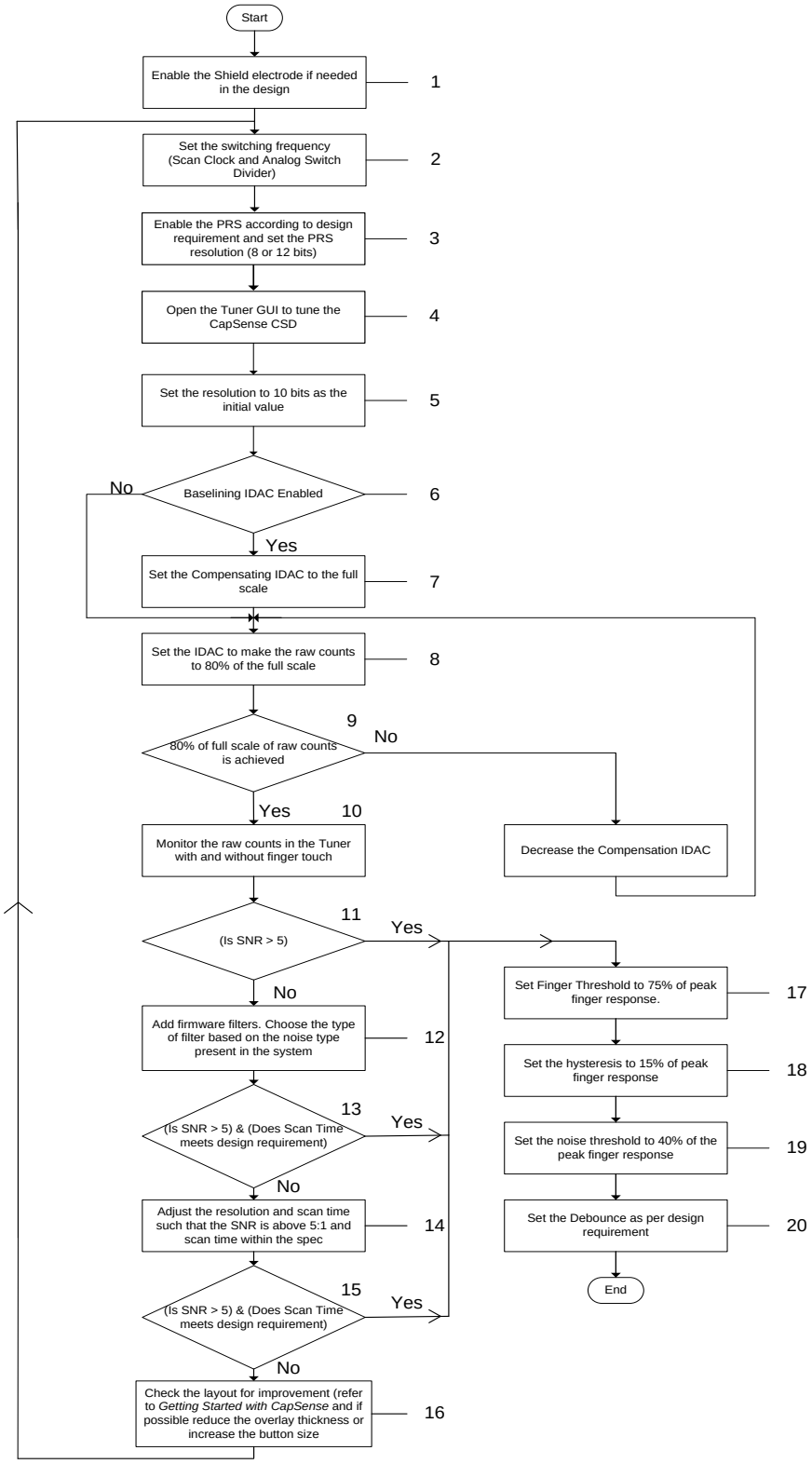
验证结果

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**最优值**的任何信噪比值。
- 低于**足够值**的任何信噪比值。
- 出现最差情形串扰故障的任何信号，以及出现该种故障的串扰个数

点击 **Validation**（验证）选项卡上的 **How do I fix this** 按钮，也可以打开 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
Enable the PRS according to design requirement and set the PRS resolution (8 or 12 bits)
3
Open the Tuner GUI to tune the CapSense CSD
4
Set the resolution to 10 bits as the initial value
5
No
Baselining IDAC Enabled
6
Yes
Set the Compensating IDAC to the full scale
7
Set the IDAC to make the raw counts to 80% of the full scale
8
9
80% of full scale of raw counts is achieved
No
Yes
10
Monitor the raw counts in the Tuner with and without finger touch
Decrease the Compensation IDAC
11
(Is SNR > 5)
Yes
No
Add firmware filters. Choose the type of filter based on the noise type present in the system
12
13
(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
14
15
(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
16
Set Finger Threshold to 75% of peak finger response.
17
Set the hysteresis to 15% of peak finger response
18
Set the noise threshold to 40% of the peak finger response
19
Set the Debounce as per design requirement
20
End

1. 屏蔽的使能或禁用依设计要求而定。在传感器丝印层可能变湿（见 AN2398）的应用中，屏蔽电极，可用来屏蔽“EMI”，或降低过高的“Cp”。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（保护电极）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** 是 CapSense 组件的首要时钟源。**Analog Switch Divider** 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** 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：通过示波器探头观察传感器电容的电压时，应在传感器寄生电容中增加探头电容。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** 对话框中改变时，项目将重建，并针对器件进行编程。
3. 默认启用 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 12 位，具体依扫描时间而定。如果扫描时间较长，使用 PRS 12；如果扫描时间较短，则使用 PRS 8。
4. 打开“CapSense 调谐器 GUI”。
5. 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，需要在扫描时间和灵敏度之间进行权衡。

   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 mm。
6. 在需要时启用基准线 IDAC。
7. 启用基准线 IDAC，再将补偿 IDAC 设置为满标度。
8. 在 GUI 中更改基准线 IDAC 值，直至原始计数(raw count)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** 对话框中更改。当 **Configure** 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。
9. 如果任何基准线 IDAC 值都不能达到满标度值的 80%，则应降低补偿 IDAC 值并重复第 8 步。
10. 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：

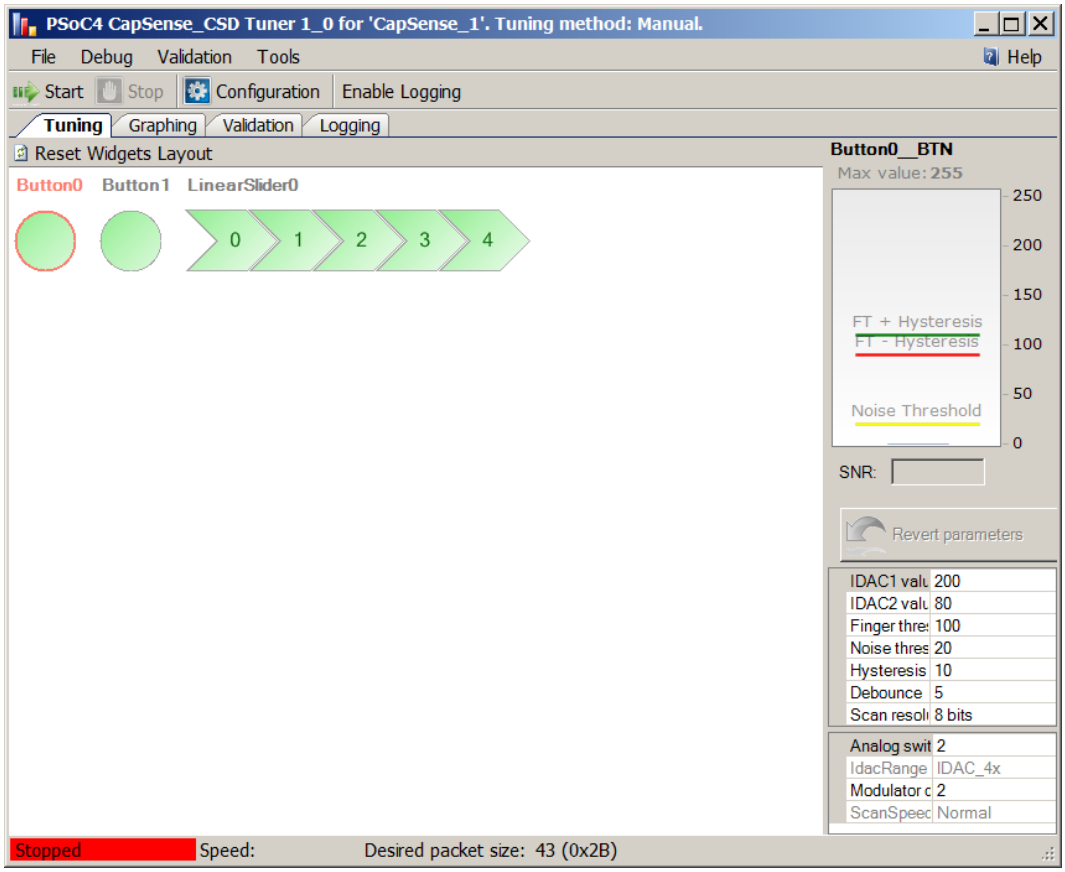
$$SNR = \frac{\text{Peak Finger Response}}{\text{Peak to Peak Noise when finger is not present}}$$

11. 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，然后选择与系统存在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。
12. 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** 按钮，会将 GUI 中经过调试的值更新到组件中。扫描时间近似值将由组件根据参数设置进行设置。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**，增加分辨率、扫描速度，或者两者。这样，扫描时间将相应延长。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 布局或丝印层设计中寻找可改进之处。请参考 *Capsense 入门指南*，了解 PCB 设计指导。您也可以减小丝印层的厚度、或增加按键直径，都可以增加敏感度。
13. 达到 5:1 的信噪比后，设置以下固件参数。
    - Finger Threshold（手指阈值）是一个参数，固件用它作为确定传感器是否处于活动状态的阈值。将此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%。
    - 平稳性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点的值，比如 5 。

调谐器 GUI 界面

常规界面



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** 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。

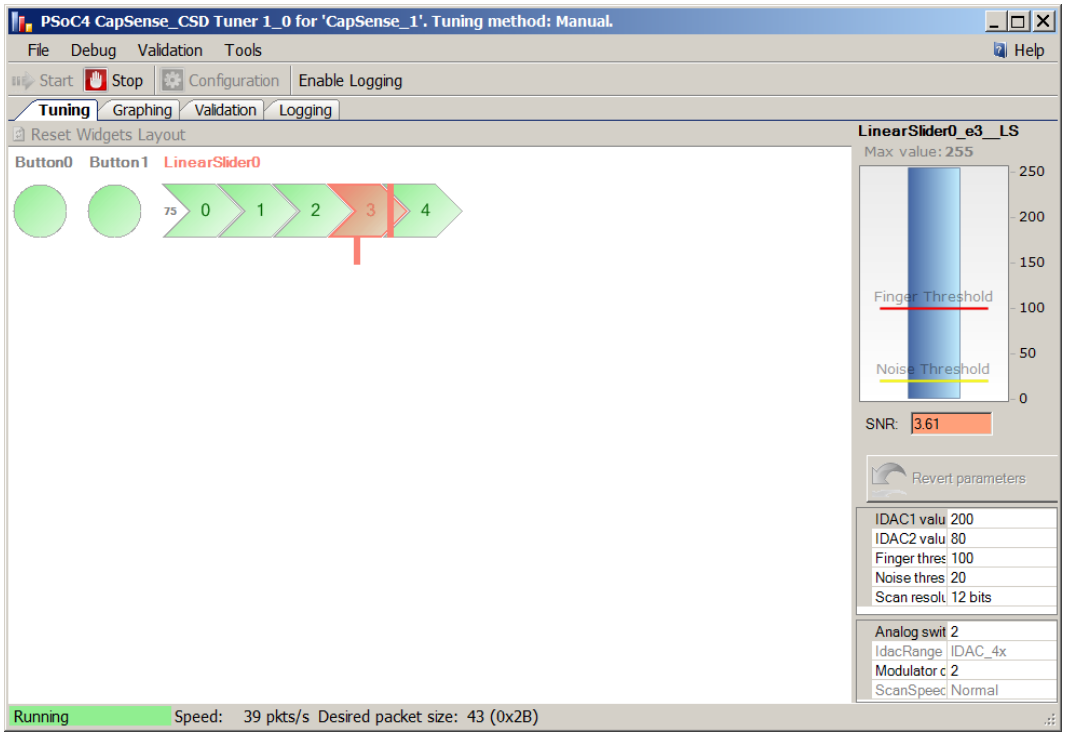
选项卡：

- **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。此选项卡用来调节 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。

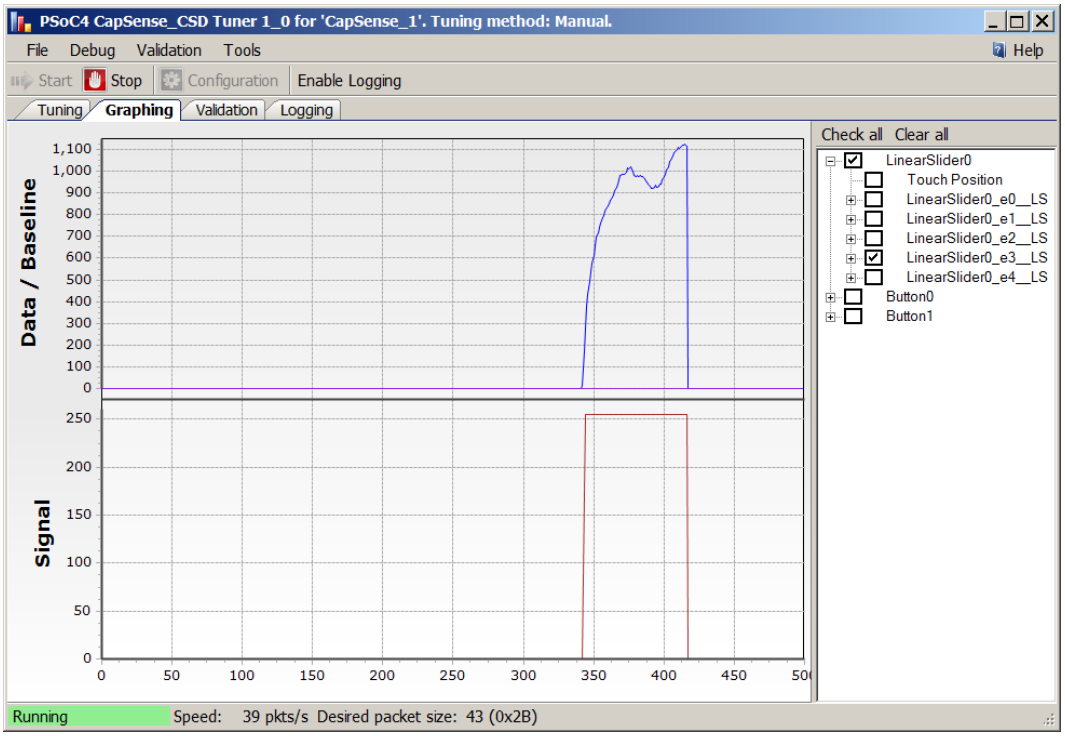
Tuning（调校）选项卡



- **Widgets schematic** — 包含所有配置的“Widget”的图形表示。如果“Widget”由一个以上传感器组成，则会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 Widgets Layout**（重置 Widget 布局）按键 — 将各“Widget”移动到原理图内的默认位置。
- **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。8 位 Widget 分辨率的有效范围介于 1 和 255 之间，默认值为 **255**。16 位 Widget 分辨率的有效范围介于 1 和 32767 之间，默认值为 **32767**。
  - 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。

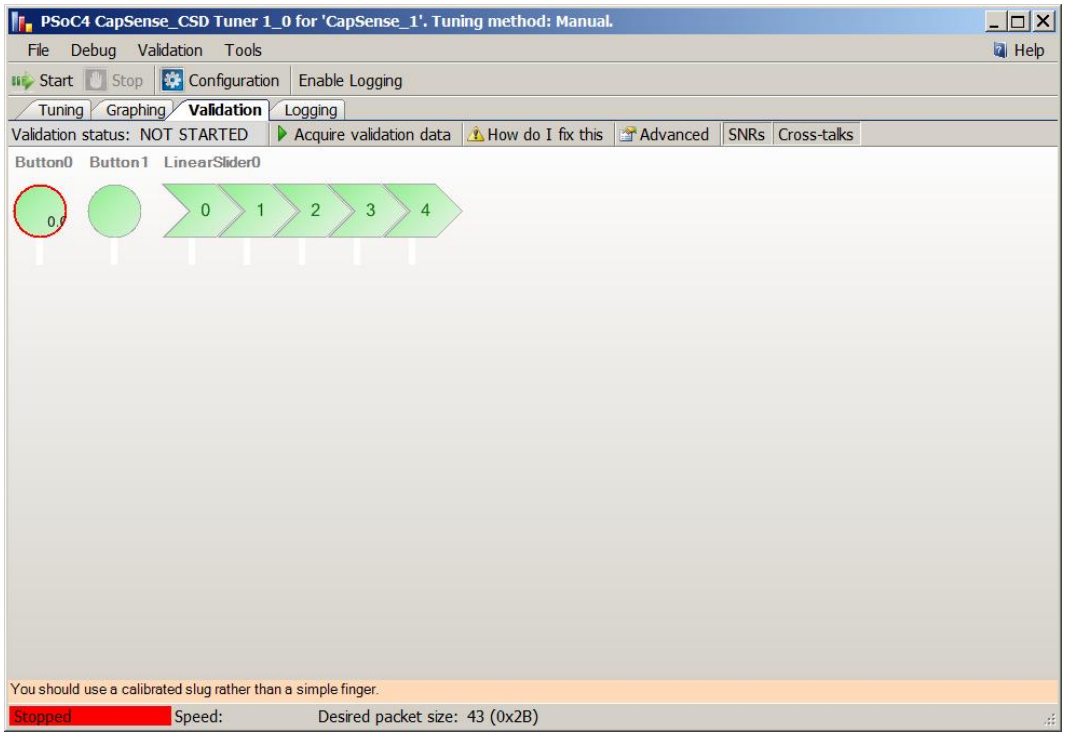
- **Sensor properties**（传感器属性）— 根据“Widget”类型，显示选定传感器的属性。其位于右侧面板上。
- **General CapSense properties**（CapSense 的常规特性，只读）— 显示 CapSense CSD 组件的全局属性，该属性不可在运行时进行更改。下列内容仅供参考。其位于右侧面板底部
- **Widget 控件上下文菜单**（此功能仅适用于 GUI 内“Widget”控件的布局）：
  - **置于背面** — 将“Widget”控件置于视图背面。
  - **置于正面** — 将“Widget”控件置于视图正面。
  - **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）
  - **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）
  - **翻转传感器** — 颠倒传感器的顺序。（仅适用于线性和辐射滑条）
  - **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）
  - **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）
  - **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 (绘图) 选项卡



- **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。
- **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启用时可显示在图表中，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（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。

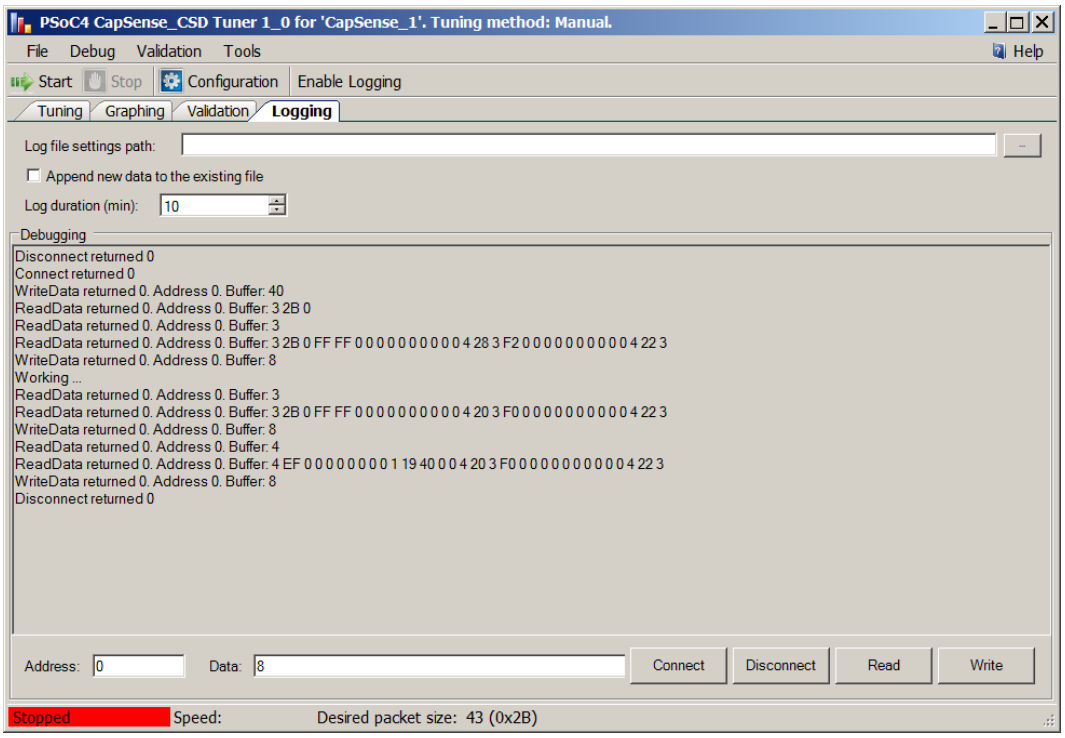
- **Widgets schematic** — 包含所有配置的“Widget”的图形表示。

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 Validation Advanced Properties（**验证高级属性**））。
- **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，请参阅验证显示）。

Logging（记录）选项卡



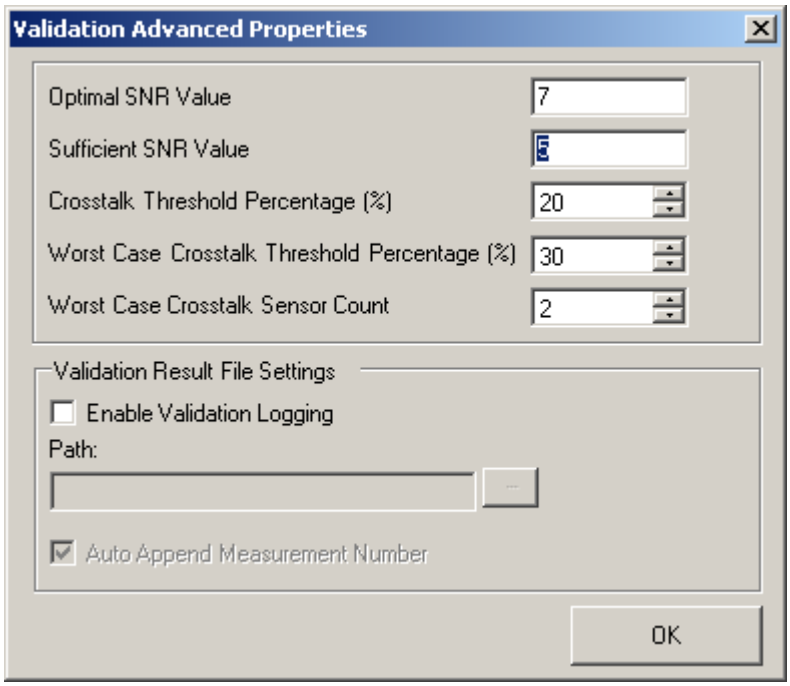
- 通过选中 **Graphing** 选项卡树视图中的复选框，可选定将要记录的数据。
- **Path** — 定义记录文件的路径（文件扩展名为“.csv”）。
- **Append new data to existing file** 复选框 — 选中该选项后，新数据即会附加到现有文件中。如果未选中，旧文件将从文件中被擦除，代之以新数据。
- **Log duration** — 定义记录持续时间，单位为分钟。有效范围介于 1 和 480 之间，默认值为 **255**。

Debugging（调试）组

该功能的存在仅为调试目的。它能帮助用户研究调谐器通信错误。

- 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。

Validation Advanced Properties（验证高级属性）



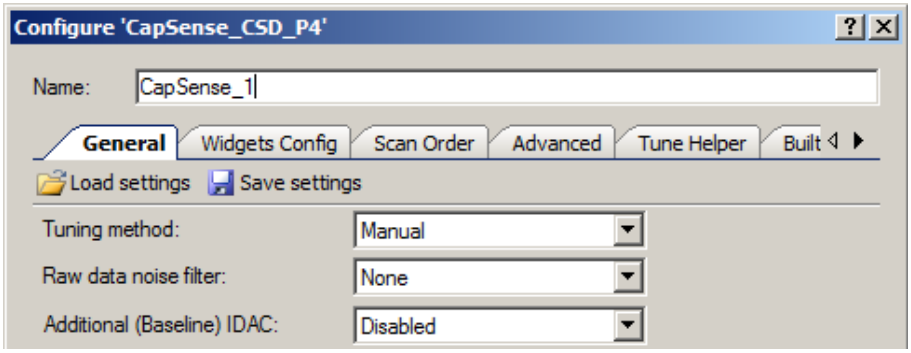
- **Optimal SNR Value**（信噪比最佳值）— 定义最佳的信噪比值。有效范围介于 0 和 100 之间，默认值为 **7**。
- **Sufficient SNR Value**（信噪比充分值）— 定义充分的信噪比值。有效范围介于 0 和 100 之间，默认值为 **5**。

- **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**（启用验证日志）— 启用对验证数据的日志。
- **Path** — 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）复选框 — 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。

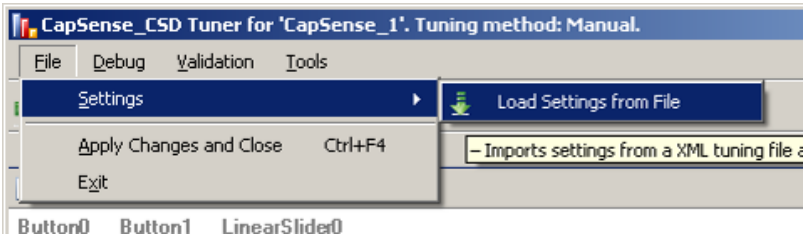
Save and Load Settings（保存/加载设置功能）

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。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。

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、变量和常量符号的前缀。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，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()	启用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，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调节模式时，调节过程应用于所有传感器。在调用其他任何API子程序之前，必须先调用CapSense_Start()子程序。

参数： 无

返回值： 无

副作用： 无

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Active（活动）模式电源模板位。

参数： 无

返回值： 无

副作用： 应在完成所有扫描后调用此函数。

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Active（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： 应该在完成扫描后调用此函数。

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

部分中所述的函数。

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

副作用： 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

副作用： 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： 应在完成扫描后调用此函数。

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

一节所述的函数。

void CapSense_RestoreConfig(void)

说明： 恢复CapSense配置。

参数： 无

返回值： 无

副作用： 应在完成扫描后调用此函数。

此函数不能将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_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()	配置选定的传感器，以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[ ]数组返回传感器原始计数。

void CapSense_ScanSensor（uint8 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计数（raw counts）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

副作用： 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Widget以外，所有Widget 都在默认情况下启用。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，因此必须手动启用接近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。

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。

副作用： 无

void CapSense_SetScanSlotSettings（uint8 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： uint8 slot：扫描插槽编号

返回值： 无

副作用： 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

副作用： 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（模拟高阻抗模式），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

副作用： 无

void CapSense_DisableSensor（uint8 sensor）

说明： 禁用选定的传感器。断开与Analog Mux Bus（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

副作用： 无

uint16 CapSense_ReadSensorRaw（uint8 sensor）

说明： 从全局CapSense_SensorResult[ ]数组返回传感器的原始计数。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。原始计数（raw counts）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：当前的原始计数值

副作用： 无

高级 API

这些 API 函数用于处理传感器 Widget 的原始计数（raw counts）。原始计数（raw counts）可从已扫描的传感器中进行取回，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]数组元素。
CapSense_InitializeEnabledBaselines()	仅扫描使能的传感器，可加载含初始值的CapSense_SensorBaseline[]数组。 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[]数组。
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[]数组并调用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[]”数组相比较，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[]数组，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[]数组，以确定辐射滑条上是否存在手指触摸，并返回位置。

函数	说明
CapSense_GetTouchCentroidPos()	若有手指存在，则此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，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： 通过扫描所选传感器，加载含初始值的CapSense_SensorBaseline[sensor]数组元素。将原始计数（raw count）值复制到每个传感器的基准线数组中。初始化原始计数(raw counts）过滤器（如果已启用）。

参数： uint8 sensor：传感器编号

返回值： 无

副作用： 无

void CapSense_InitializeEnabledBaselines(void)

说明： 扫描所有已启用的Widget。对于扫描过程中启用的所有传感器，原始计数(raw count)值被复制到CapSense_SensorBaseline[ ]数组。对扫描过程禁用的传感器，用零值初始化CapSense_SensorBaseline[ ]。初始化原始计数(raw counts)过滤器（如果已启用）。

参数： 无

返回值： 无

副作用： 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[ ]数组。将原始计数(raw count)值复制到所有传感器的基准线数组中。初始化原始数据过滤器（如果已使能）。

参数： 无

返回值： 无

副作用： 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]数组元素。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_SensorSignal[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。

若原始计数(raw counts）过滤器在计算基准线之前启用，则可应用于这些数值。

参数： uint8 sensor：传感器编号

返回值： 无

副作用： 无

void CapSense_UpdateEnabledBaselines（void）

说明： 检查CapSense_SensorEnableMask[]数组并调用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

副作用： 无

void CapSense_EnableWidget（uint8 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： 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

副作用： 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

副作用： 无

uint8 CapSense_CheckIsWidgetActive（uint8 widget）

说明： 将选定的传感器CapSense_Signal[ ]数组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[ ]数组中的位。

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。

副作用： 此函数还更新从属于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[ ]数组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[ ]数组为最新。

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。

副作用： 和CapSense_CheckIsWidgetActive()函数具有同样的副作用，但并非针对所有传感器。

uint16 CapSense_GetCentroidPos（uint8 widget）

说明： 检查CapSense_Signal[ ]数组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。只有当CapSense定制器定义了线性滑条Widget 时，才能使用此函数。

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

副作用： 如果滑条“Widget”内有任何传感器处于活动状态，则函数将数值从零返回至CapSense定制器中设置的API分辨率值。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。

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

注：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指压结果。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指压。

uint16 CapSense_GetRadialCentroidPos（uint8 widget）

说明： 检查CapSense_Signal[ ]数组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。位置过滤器（如果已使能）将应用于该结果中。只有当CapSense定制器定义了辐射滑条 Widget 时，才能使用此函数。

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。

副作用： 如果滑条Widget内有任何传感器处于活动状态，则函数将数值从零返回至CapSense定制器中设置的API分辨率值。如果没有任何传感器处于活动状态，则该函数返回0xFFFF。

没有为此函数提供“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：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指压结果。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指压。

uint8 CapSense_GetTouchCentroidPos（uint8 widget， uint16* pos）

说明： 如果触控板上有手指，则此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。只有当CapSense定制器定义了触控板时，才能使用此函数。

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”数组的指针，触摸位置将存于该数组：
pos[0] — X轴位置；
pos[1] — Y轴位置。

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。

副作用：

uint8 CapSense_GetMatrixButtonPos（uint8 widget， uint8* pos）

说明： 如有手指，该函数计算手指的行位置和列位置。如果矩阵按键上有一个手指，则返回一个‘1’。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”数组的指针，触摸位置将存于该数组：
pos[0] — 列位置；
pos[1] — 行位置。

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。

副作用：

调谐器助手 API

这些 API 函数与调谐器 GUI 一起使用 。

函数	说明
CapSense_TunerStart()	初始化CapSense CSD和内部通信器件，初始化基准线并启动传感器扫描循环。
CapSense_TunerComm()	执行调谐器GUI之间的通信。

void CapSense_TunerStart（void）

说明： 使CapSense CSD和内部通信器件初始化。

除接近“Widget”以外，所有“Widget”都默认使能。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”。

参数： 无

返回值： 无

副作用： 无

void CapSense_TunerComm（void）

说明： 执行与调谐器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）：执行与调谐器 GUI 之间的通信功能。将传感器扫描和“Widget”处理结果传输到调谐器 GUI。自动调试参数也传输到调谐器 GUI。调谐器 GUI 参数不会被传输回 CapSense CSD 组件。

此函数正在执行阻止操作，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

副作用： 无

数据结构

API 函数使用几种全局数组来处理传感器和 Widget 数据。不得手动更改这些数组。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数组的内容。全局数组为：

- CapSense_SensorRaw［ ］
- CapSense_SensorEnableMask［ ］
- CapSense_portTable［ ］和 CapSense_maskTable［ ］
- CapSense_SensorBaseline［ ］
- CapSense_SensorBaselineLow［ ］
- CapSense_SensorSignal［ ］
- CapSense_SensorOnMask［ ］

CapSense_SensorRaw［ ］

此数组包含每个传感器的原始数据。数组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数组 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。此字节数组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 — 传感器不被扫描。CapSense_SensorEnableMask[ ]数据通过以下函数进行更改：

- CapSense_EnabledWidget()
- CapSense_DisableWidget()

CapSense_SensorEnableMask[ ]数据为以下函数所用：

- CapSense_ScanEnabledWidgets()

CapSense_portTable[ ]和 CapSense_maskTable[ ]

这些数组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[ ]

此数组存储低通滤波器中所使用每个传感器的基线数据分数字节，用于基线更新。数组大小等于传感器总数。CapSense_SensorBaselineLow[ ]数组通过以下函数更新：

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorBaseline[ ]

此数组存储每个传感器的基线数据。数组大小等于传感器总数。CapSense_SensorBaseline[ ]数组通过以下函数更新：

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorSignal[ ]

此数组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而来的传感器信号数据。数组大小等于传感器总数。**Widget Resolution** 将词数组的分辨率定义为 **1 字节**或 **2 字节**。CapSense_SensorSignal[ ]数组通过以下函数更新：

- 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数组。

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。此字节数组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，且只有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。

传感器常量

为每个传感器提供一个常量。可在以下函数中将这些常量用作参数：

- 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 +"#element number"+ "__" + 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）。“CapSense CSD”组件中只能使用唯一一个“Widget”名称。所有“Widget”名称均为大写。
- 元件号 — 仅具有多个元件的“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	接近传感器
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用作参数：

- 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
```

样例固件源代码

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。要获取器件示例，请打开器件目录中的对话框或原理图中的器件实例。要查看通用样例，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到器件上的物理引脚。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。

传感器引脚 — CapSense_cPort — 引脚分配

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

传感器别名为：

“Widget”名称 + 元件号 +“__”+ “Widget”类型

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。可将 C_{MOD} 连接到 P4[2]引脚。

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例程中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。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。

功能描述

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。

噪声阈值

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

去抖动

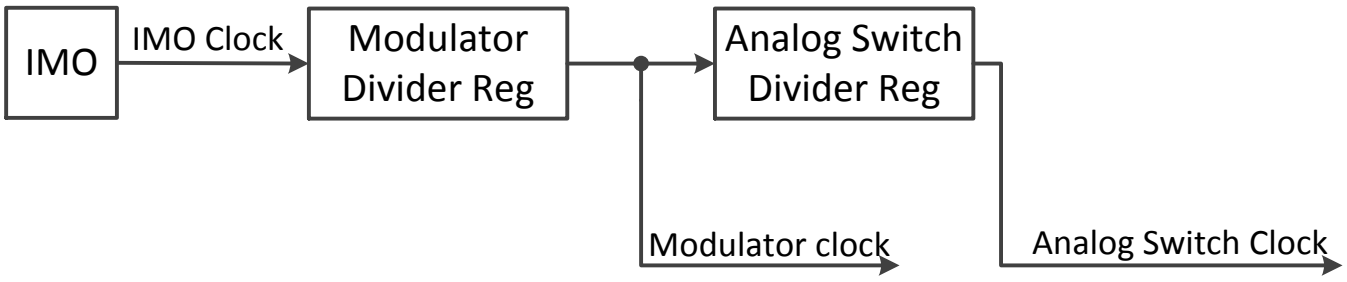
为传感器活动的瞬变增加了去抖动计数器。为了让传感器能够从非活动状态切换到活动状态，在规定的样本数量内，差异计数值必须保持在手指阈值加迟滞值之上。这对于滤去高振幅和频率噪声是有必要的。

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。

CapSense 时钟

CapSense 时钟树如下图所示。



将 IMO 时钟除以调制器分频器，就形成了调制时钟。

将调制时钟除以模拟开关分频器，就形成了模拟开关时钟。

例如，如果您将模拟开关分频器的值设为 8，且调制器分频器的值设为 4，则将调制器分频器寄存器配置为除以 4，而且将模拟开关分频器寄存器配置为除以 2。

API 分辨率—内插和测量

在滑条传感器和触控板中，通常需要更精确地分辨手指（或其他电容器物体）的位置，而非单个传感器本身的分辨率。手指接触滑条传感器或触控板的面积通常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，首先对数组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触摸中心。使用少至两个、多至八个传感器计算质心。

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

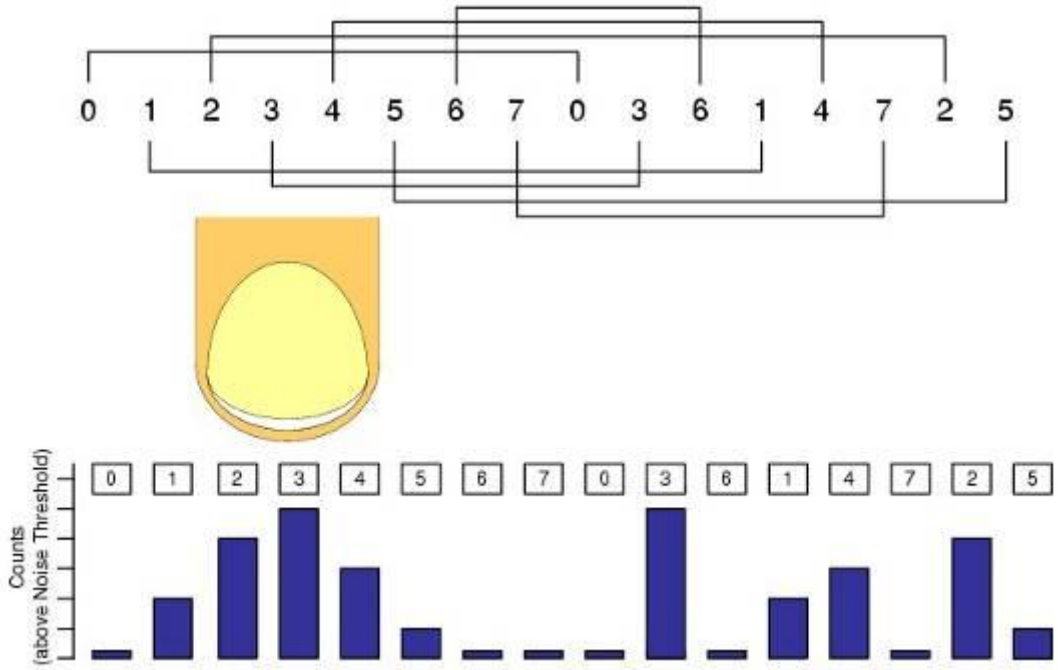
计算出的值通常是分数。为了能够采用某一特定分辨率来报告质心（例如对于 12 个传感器使用 0 到 100 的范围），要将质心值乘以标量。另一种更有效的方法是将内插和按比例计算的方法统一

到单一计算中，按所需的量级直接报告结果。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滑条传感器阵列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，并在包括文件中列出。一旦创建好次序，某一部分中相邻的传感器动作则不会导致另一部分中相邻的传感器动作。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 1. 不同滑条段计数的双工序列

滑条段总计数	段序列
10	0、1、2、3、4、0、3、1、4、2
12	0、1、2、3、4、5、0、3、1、4、2、5
14	0、1、2、3、4、5、6、0、3、6、1、4、2、5

滑条段总计数	段序列
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
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

滑条段总计数	段序列
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

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计数(raw counts)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。此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。启用此滤波器对于每个传感器（原始）和 Widget（位置）会消耗 4 个字节的 RAM。它默认为禁用。

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。启用此滤波器对于每个传感器（原始）和 Widget（位置）会消耗 4 个字节的 RAM。它默认为禁用。

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。启用此滤波器对于每个传感器（原始）和 Widget（位置）会消耗 2 个字节的 RAM。原始和位置滤波器在默认情况下启用 IIR1/4。

一阶 IIR 滤波器：

$$\mathrm{IIR}\,1/2 = 1/2\,\mathrm{previous} + 1/2\,\mathrm{current}$$

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

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

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

抖动滤波器

此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。启用此滤波器对于每个传感器（原始）和 Widget（位置）会消耗两个字节的 RAM。它默认为禁用。

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。

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。如果器件丝印层表面有水膜，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

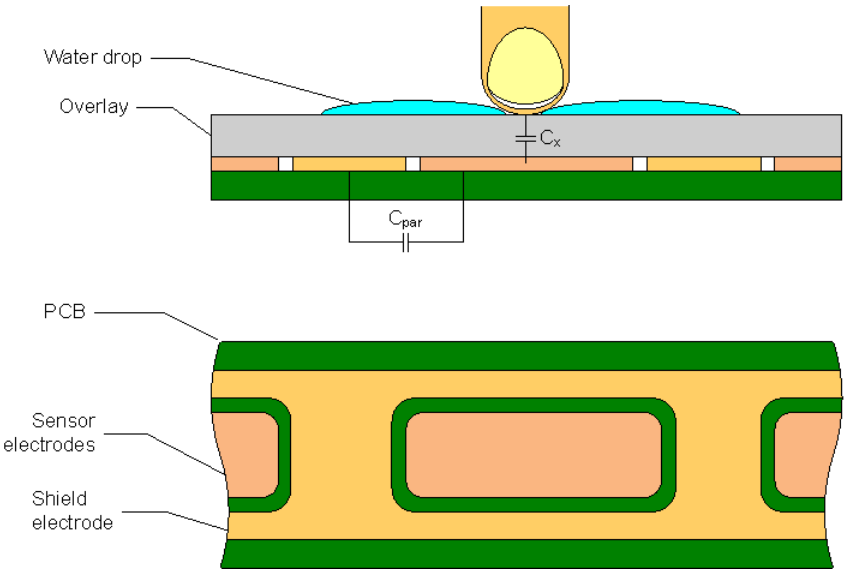


图 2 提供了一种按键屏蔽电极的可能布局组态。屏蔽电极尤其适用于透明的 ITO 触摸板器件，在这种器件中，它不但可阻止 LCD 驱动电极的噪声，同时可减少杂散电容。

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，其中包括按钮下面的平板。对于这种情况，建议使用填充模式，填充率约为 30 ～ 40%。在此种情况下，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。

资源

CapSense CSD 使用以下数字和模拟资源。

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
基准线IDAC禁用	1	0
基准线IDAC启用	1	1
SmartSense	1	1

API 存储器占用

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。下表列出了在指定器件配置中可用的所有 API 的存储器使用情况。

使用释放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最佳设置。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 4（GCC）	
	闪存（字节）	SRAM（字节）
调试方法：自动 （SmartSense） 电流源：IDAC 源 原始数据滤波器：求平均值 基线 IDAC：使能 防水：禁用	2042	812
调试方法：无 电流源：IDAC 灌电流 原始数据滤波器：无 基线 IDAC：使能 防水：禁用	1904	278
调试方法：无 电流源：IDAC 源 原始数据滤波器：无 基线 IDAC：使能 防水：禁用	1904	278

配置	PSoC 4（GCC）	
	闪存（字节）	SRAM（字节）
调试方法：无 电流源：IDAC 源 原始数据滤波器：无 基线 IDAC：使能 防水：使能	1956	287
调试方法：无 电流源：IDAC源 原始数据滤波器：无 基准线 IDAC：禁用 防水：使能	1752	287

直流和交流电气特性

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

直流电规范

参数	说明	最小值	典型值	最大值	单位	条件
VCSD	工作电压范围	1.71	–	5.5	V	

交流电规范

参数	说明	最小值	典型值	最大值	单位	条件
SNR	手指噪声计数比率	5	–	–	比率	电容范围在 9 pF 到 32 pF 之间，灵敏度为 0.1pF

组件更改

版本	更改内容	更改/影响原因
1.10	扫描时间得到优化。	

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，2013。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以外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不会以明示或暗示的方式授予任何专利许可或其他权利。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于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，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对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。赛普拉斯保留在不另行通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于合理的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器件。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。

产品使用可能受适用的赛普拉斯软件许可协议限制。