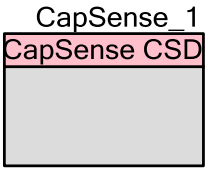
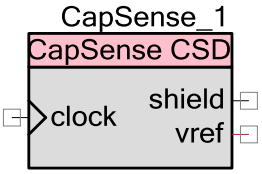
电容式感应 (CapSense®CSD)

3.10

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。
- 提供 SmartSense™ 自动调节或通过集成式 PC GUI 进行手动调节。
- 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

电容式感应使用 Delta-Sigma Modulator（Delta-Sigma 调制器） (CapSense CSD) 组件，能够为测量触摸感应按键、滑条、触摸板和接近检测等应用程序中的电容提供通用有效的方法。

阅读本数据手册后，请阅读以下文档。具体见赛普拉斯半导体公司网站 www.cypress.com：

- *CapSense 入门*
- *Waterproof Capacitive Sensing* 防水电容式感应——AN2398

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，甚至可用于淋雨或受潮的应用中。这些应用包括汽车、室外设备、ATM、公众接入系统、手机和 PDA 等便携式设备以及厨房和浴室应用。

输入/输出连接

本节介绍 CapSense CSD 组件的各种输入和输出连接。I/O 列表中的星号 (*) 表示，在 I/O 说明中列出的情况下，该 I/O 可能隐藏在该符号中。

时钟 – 输入*

为 CapSense CSD 组件提供时钟。时钟输入仅在选中**启用时钟输入**参数时可见。

屏蔽 – 输出*

此输出连接屏蔽电极信号。仅当启用屏蔽电极时，才可使用该连接。Component Parameters （参数和设置）一节介绍了有关屏蔽使用的详细信息。

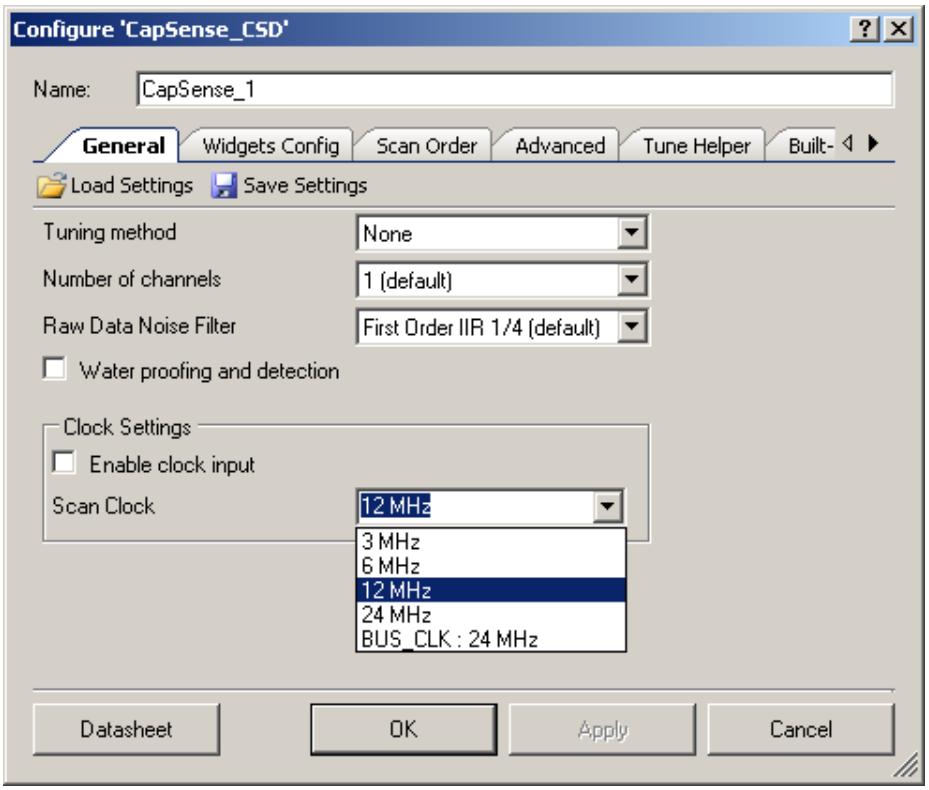
vref – 输出*

此输出连接模拟基准电压。可用于调整屏蔽信号振幅。仅当 IDAC **电流源模式**中启用了**屏蔽**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 基准。有关 vref 使用的详细信息，请参考本数据手册的功能描述一节。

Component Parameters（组件参数）

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

一般选项卡



加载设置/保存设置

保存设置用于保存为组件配置的所有设置和调节数据。这样，在新项目中就能够快速复制。**加载设置**用于加载以前保存的设置。

存储的设置也可用于将设置和调节数据导入调节器 GUI。

调节方法

此参数指定了调节方法。其中有三个选项：

- **Auto (SmartSense)**（自动(SmartSense)）– 提供 CapSense CSD 组件的自动调节。

  它是我们对所有设计建议的调节方法。运行时，固件算法不断地确定最佳的调节参数。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 调节，该组件在 **Tuner Helper**（调节器助手）选项卡中指定用于将目标设备中的数据传输到调节器 GUI。

- **Manual**（手动）– 允许用户使用 Tuner （调节器）GUI 手动调节 CapSense_CSD 组件。

  要启动 GUI，请右击符号，然后选择**启动调节器**。有关手动调节的更多信息，请参阅本数据手册中的调节器 GUI 用户指南一节。手动调节需要 EZI2C 通信组件，该组件在 **Tuner Helper**（调节器助手）选项卡中指定用于在目标设备和调节器 GUI 之间传输数据。

- **无**（默认设置）– 禁用调节。

  所有调节参数均存储于闪存中。只有在 CapSense 组件的所有参数调节完成后，才可使用此选项。如果用户使用该选项，Tuner 将工作在只读模式。

通道数量

此参数指定了实现的硬件扫描通道数量。

- **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 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 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** – 最适用于 20 个以上的传感器。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。在连续时间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。

原始数据噪声滤波器

此参数选择原始数据滤波器。仅可选择一个滤波器，且将该滤波器应用于所有传感器。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。

- **无–** 没有提供滤波器。没有导致滤波器固件或 SRAM 变量开销。
- **中值 –** 按顺序排列最近三个传感器值，并返回中值。
- **均值 –** 返回最近三个传感器值的简单均值。
- **一阶 IIR 1/2 –** 返回增加到前一滤波器值 1/2 的最近传感器值的 1/2。IIR 滤波器需有所有滤波器类型的最低固件和 SRAM 开销。
- **一阶 IIR 1/4**（一阶 IIR 1/4）（默认）**–** 返回增加到前一滤波器值四分之三的最近传感器值的四分之一。
- **Jitter**（抖动）**–** 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。
- **一阶 IIR 1/8 –** 返回增加到前一滤波器值 7/8 的最近传感器值的 1/8 。
- **一阶 IIR 1/16 –** 返回增加到前一滤波器值 15/16 的最近传感器值的 1/16。

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：

- 启用屏蔽输出终端
- 添加“保护”Widget

  注如果防水无需“Guard（保护）”widget，则可以在 **Advanced**（高级）选项卡上将其删除。

启用时钟输入

此参数选择组件是使用内部时钟，还是显示用户提供的时钟连接的输入终端（默认为禁用）。

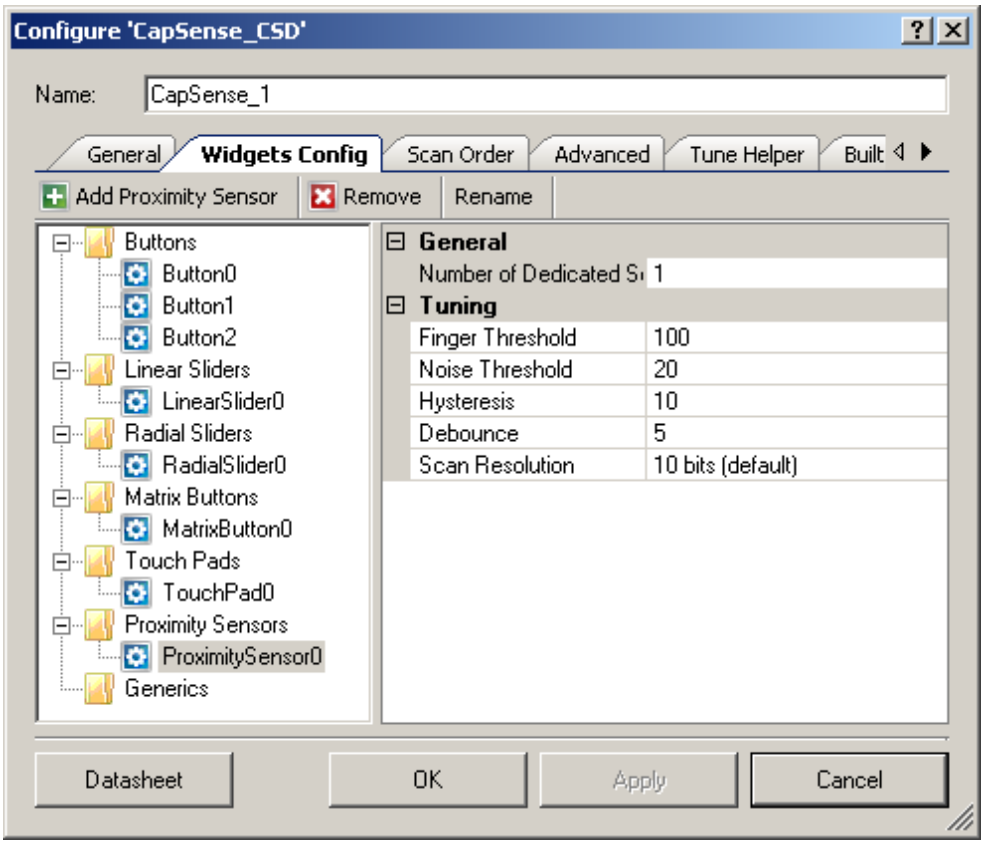
注如果调节方法为 **Auto (SmartSense)**（自动(SmartSense)），则该选项不可用，因为，自定义程序必须知道时钟频率才能计算内部数据。

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**（启用时钟输入），则该特性不可用。

注 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，不支持小于等于 BUS_CLK 的 CapSense CSD 时钟，用户应选择 BUS_CLK。

Widget 配置选项卡



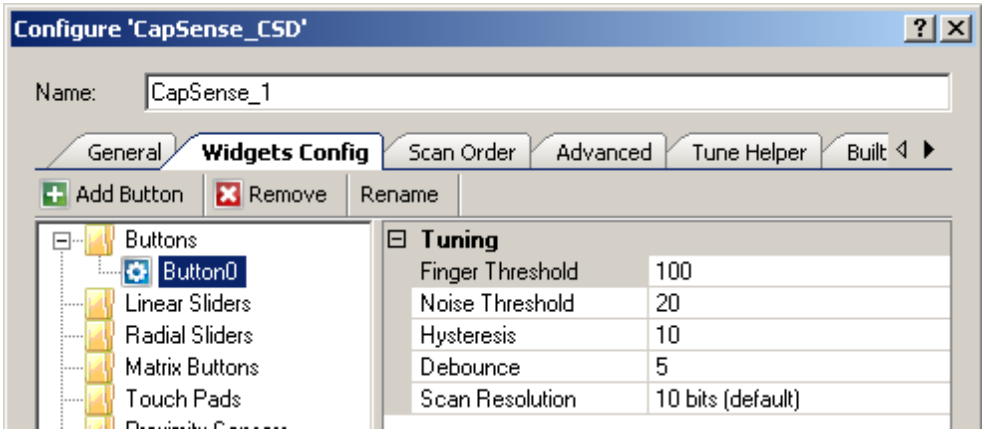
有关各种参数的定义，请参见功能描述一节。

工具栏

该工具栏包含以下命令：

- **添加 Widget**（热键 – Insert）– 向树中添加选定的 Widget 类型。Widget 类型包括：
  - **按键** – 按键检测到单个传感器上存在指压，并提供单个机械按键替代。
  - **线性滑条** – 线性滑条提供一个关于指压位置整数值，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。
  - **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。
  - **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指压。矩阵按键提供了一种扫描大量按键的有效方法。
  - **触摸板** – 触摸板返回触摸板区域内指压的 X 和 Y 坐标。触摸板包含多个行传感器和列传感器。
  - **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。
  - **通用传感器** – 通用传感器提供来自单个传感器的原始数据。这样用户可以创建唯一或高级传感器，而这是经其他类型传感器进行了处理的输出所不能实现的。
- **删除 Widget**（热键为 – Delete）– 从树中删除选定的 Widget。
- **重命名**（热键为 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。

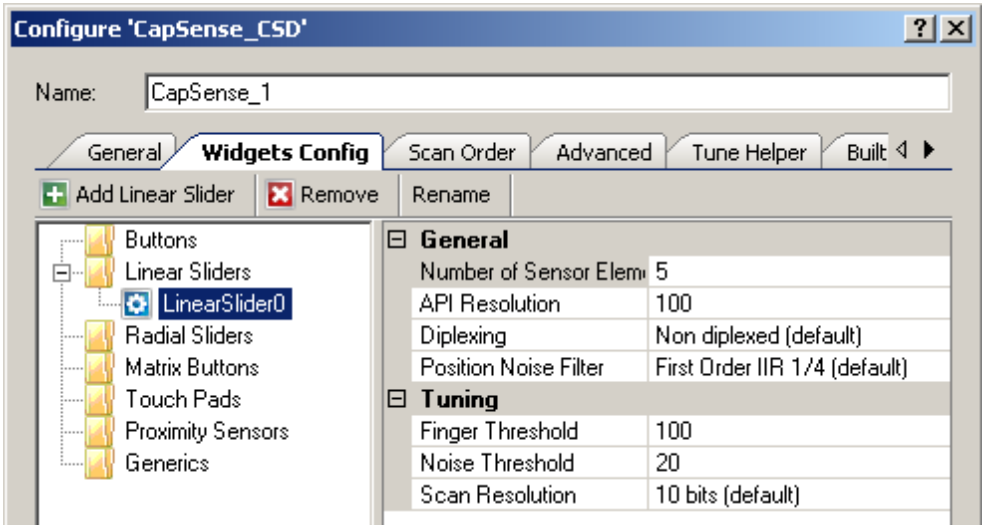
按键



调节：

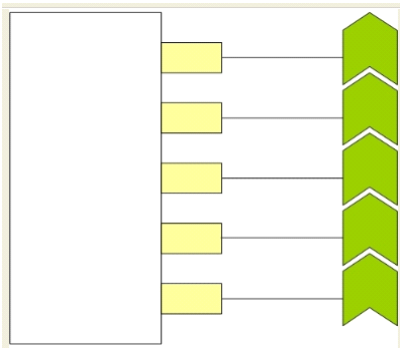
- **手指阈值** – 定义造成触摸灵敏度增大或减小的传感器活动阈值。当传感器扫描值大于此阈值时，该按键报告为被触摸。默认值为 **100**。值的有效范围为 [1…255]。**手指阈值** + **迟滞**不可超过 254。
- **噪声阈值** – 定义传感器噪声阈值。如果计数值高于此阈值，则不更新基准线。如果噪声阈值过低，传感器和热偏移可能被忽略不计。这样会导致假象触摸或被遗漏的触摸。如果噪声阈值太高，手指触摸可能会被解释为噪声，导致基准线错误地升高，导致手指触摸遗漏。默认值为 **20**。值的有效范围为 [1…255]。
- **迟滞** – 添加传感器活动状态转换的差分迟滞。如果传感器处于非激活状态，则差值计数必须大于手指阈值与迟滞的和。如果传感器处于激活状态，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抖动。默认值为 **10**。值的有效范围为 [1…255]。**手指阈值** + **迟滞**不可超过 254。
- **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，差异计数值必须保持在手指阈值加迟滞值之上。默认值为 5。去抖动确保高频率高振幅的噪声不会导致对按压按键的误检。值的有效范围为 [1…255]。
- **扫描分辨率** – 定义扫描分辨率。此参数对按键 Widget 内传感器的扫描时间会产生影响。N 位扫描分辨率的最大原始计数为 2^N-1。增加分辨率可提高触摸检测的灵敏度和信噪比 (信噪比)，但会增加扫描时间。默认值为 **10 位**。

线性滑条

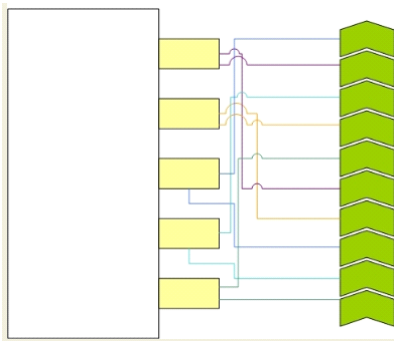


一般：

- **传感器元件数量 –** 定义滑条内的元件数量。API 分辨率与传感器元件的最佳比为 20:1。将 API 分辨率与传感器元件之比增加过多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** 个元素。
- **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]。
- **双工 – 非双工**（默认）或**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。



Non Diplexed



Diplexed

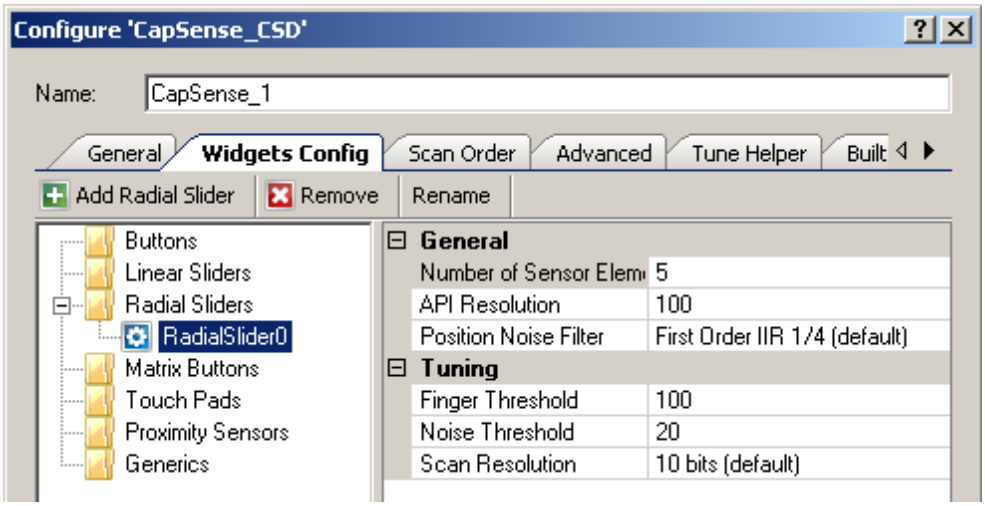
- **位置噪声滤波器 –** 选择噪声滤波器类型以进行位置计算。对于一个选定的 Widget，仅可应用一个滤波器。关于滤波器类型的详情，见本文档功能描述一节的滤波器部分。
  - ❑ 无
  - ❑ 中位数

- ❑ **平均数**
- ❑ **一阶 IIR 1/2**
- ❑ **一阶 IIR 1/4**（默认）
- ❑ **抖动**

调节：

- **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，该按键报告为被触摸。默认值为 **100**。值的有效范围为 [1…255]。
- **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**。值的有效范围为 [1…255]。
- **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 位的扫描分辨率最大原始计数为 2^N-1。增加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



一般：

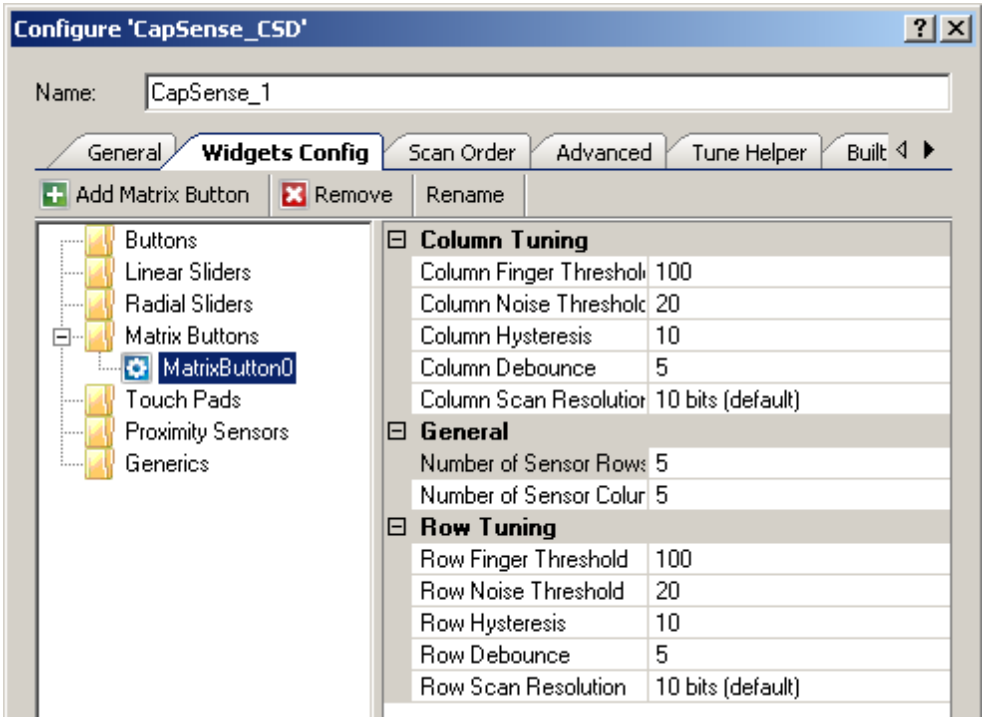
- **传感器元件数量** – 定义滑条内的元件数量。API 分辨率与传感器元件的最佳比为 20:1。将 API 分辨率与传感器元件之比增加过多会造成分辨率计算的噪声增加。值的有效范围为 [2…32]。默认值为 **5** 个元素。
- **API 分辨率** – 定义滑条分辨率。位置值将在此范围以内变化。值的有效范围为 [1…255]。

- **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。
  - **无**
  - **中位数**
  - **平均数**
  - **一阶 IIR 1/2**
  - **一阶 IIR 1/4**（默认）
  - **抖动**

调节：

- **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，该按键报告为被触摸。默认值为 **100**。
- **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**。值的有效范围为 [1…255]。
- **扫描分辨率 –** 定义扫描分辨率。此参数对辐条滑块 Widget 内所有传感器的扫描时间会产生影响。N 位的扫描分辨率最大原始计数为 2^N-1。增加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**。

矩阵按键



调节：

- **列/行手指阈值** – 定义导致触摸灵敏度增大或减小的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**。值的有效范围为 [1…255]。**手指阈值** + **迟滞**不可超过 254。

- **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**。值的有效范围为 [1…255]。

- **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**。值的有效范围为 [1…255]。**手指阈值** + **迟滞**不可超过 254。

- **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。去抖动确保高频率高振幅的噪声不会导致对按压按键的误检。值的有效范围为 [1…255]。

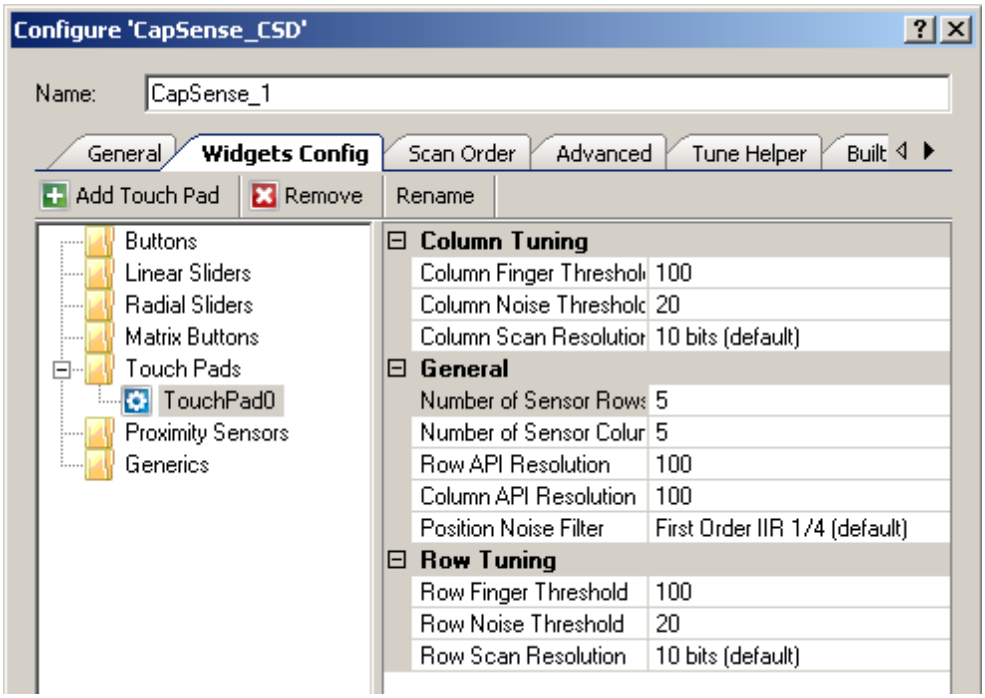
- **列/行扫描分辨率** – 定义矩阵按键列和行的扫描分辨率。此参数对矩阵按键 Widget 列/行内所有传感器的扫描时间产生影响。N 位的扫描分辨率最大原始计数为 2^N-1。增加分辨率可提高触摸

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**。

一般：

- **传感器列/行数量** – 定义形成矩阵的列和行的数量。值的有效范围为 [2…32]。列和行的元件数量默认值均为 5 个。

触摸板



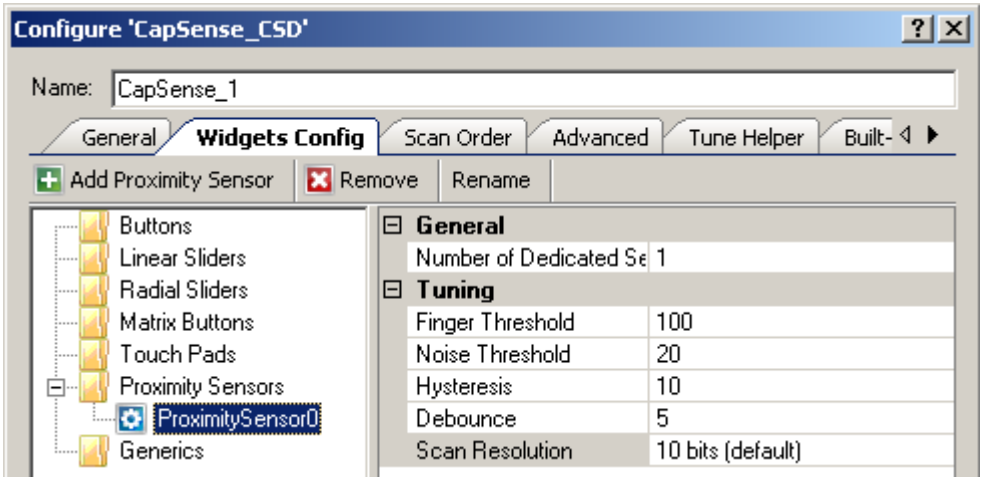
调节：

- **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**。值的有效范围为 [1…255]。
- **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**。值的有效范围为 [1…255]。
- **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 位的扫描分辨率最大原始计数为 2^N-1。增加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**。

一般：

- **传感器列和行的数量 –** 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**。
- **API 分辨率列和行 –** 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]。
- **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。
  - **无**
  - **中位数**
  - **平均数**
  - **一阶 IIR 1/2**
  - **一阶 IIR 1/4**（默认）
  - **抖动**

接近传感器



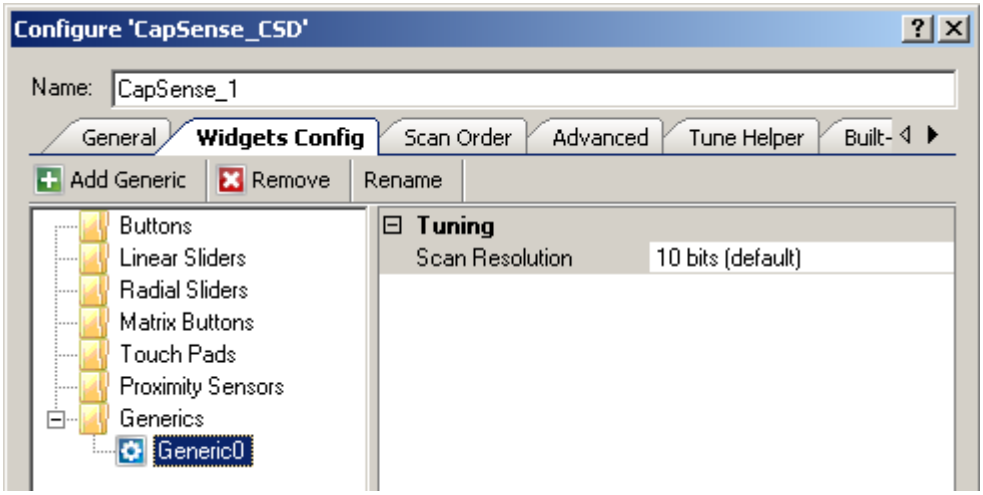
一般：

- **专用传感器元件数量 –** 选择专用接近传感器的数量。这些传感器元件不包括用于其他 Widget 的所有其他传感器。任何 Widget 传感器均可单独使用，或并行连接以建立接近传感器。
  - **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。

调节：

- **手指阈值** – 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**。值的有效范围为 [1…255]。**手指阈值** + **迟滞**不可超过 254。
- **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。值的有效范围为 [1…255]。
- **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。值的有效范围为 [1…255]。
- **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。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。值的有效范围为 [1…255]。
- **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 位的扫描分辨率最大原始计数为 2^N-1。增加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**。

普通



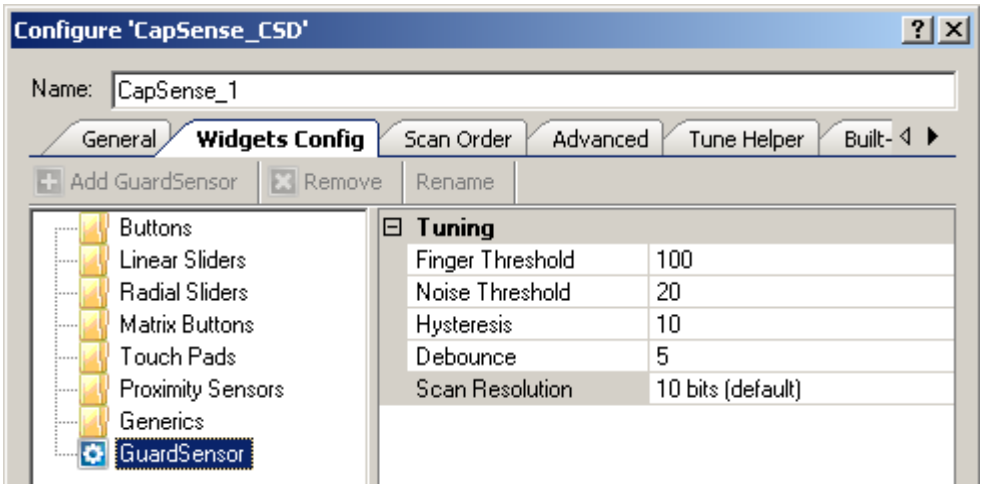
调节：

- **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。增加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**。

为每个通用 Widget 仅提供一个调节选项，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。

防护传感器

可使用**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。关于此传感器类型以及应何时适用的更多信息，请参考本数据手册功能描述一节中的防护传感器实现部分。

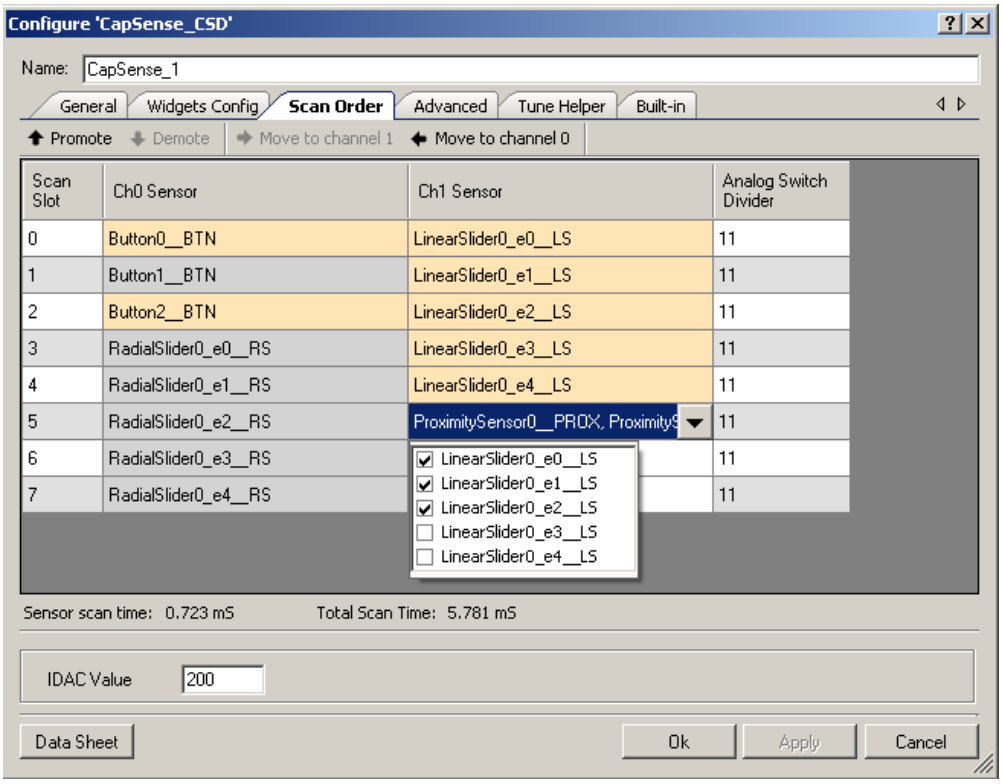


调节：

- **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**。值的有效范围为 [1…255]。**手指阈值** + **迟滞**不可超过 254。
- **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。默认值为 **20**。有效范围为 [1…255]。
- **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少量移动不会导致按键状态的循环。默认值为 **10**。值的有效范围为 [1…255]。**手指阈值** + **迟滞**不可超过 254。

- **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]。
- **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 位的扫描分辨率最大原始计数为 2^N-1。增加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**。

“扫描顺序”选项卡



工具栏

该工具栏包含以下命令：

- **升级/降级** (热键 – +/-) – 使选定 widget 在数据网格 (data grid) 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。
- **移动到通道 1/通道 0** (热键为 – Shift + 1/0) – 移动选中 widget 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 (仅适用于通用和接近 widgets)。

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（位于选项卡），则该列是隐藏的。

IDAC 值

指定选定传感器的 IDAC 值。此选项仅当 **IDAC 源**选定为**电流源**（在 **Advanced**（高级）选项卡下）时才处于活动状态。有效范围介于 0 至 255 之间。 默认值为 **200**。

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**. **Sensitivity**（灵敏度）设置传感器的整体灵敏度，对应于丝印层材料的不同厚度。越厚的材料应使用越低的灵敏度值。

仅当 **Tuning method**(调节方法)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)）作为调节方法时，显示值可能不准确，因为在调节时参数发生变化。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 5）。

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)）作为调节方法时，显示值可能不准确，因为在调节时参数发生变化。当 CapSense CSD 组件输入时钟频率未知时，会显示 **Unknown**（未知）。

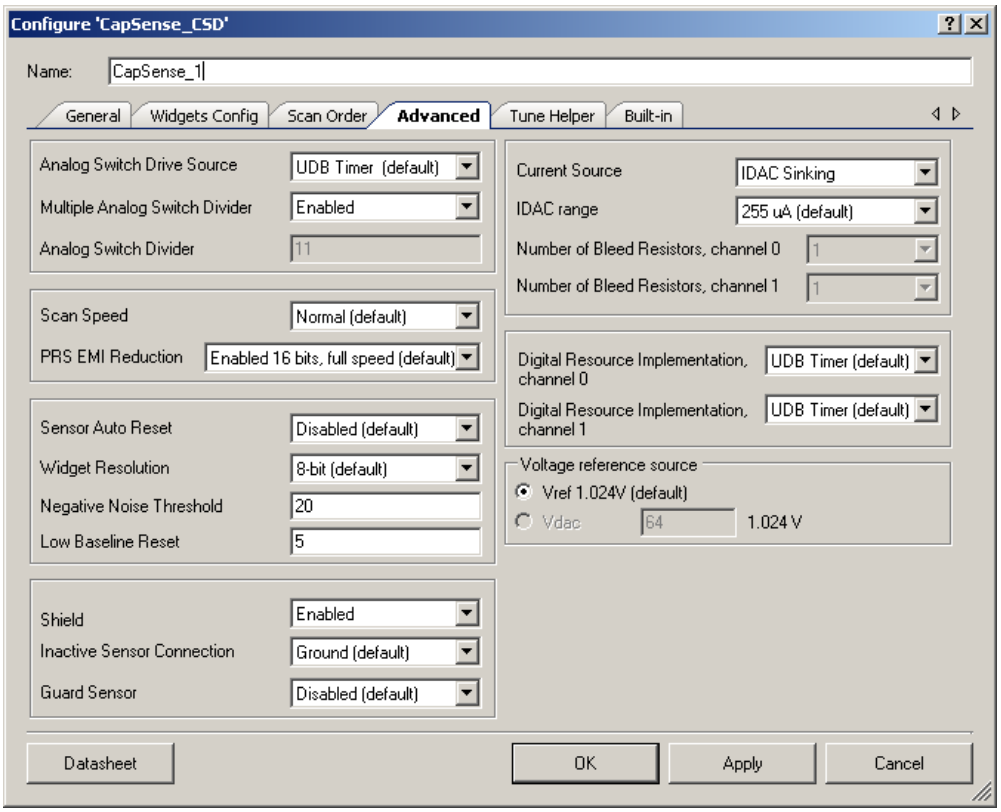
Widget 列表

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 Tab（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，以确定传感器切换至调制电容 C_{MOD} 和从调制电容 C_{MOD} 切换时的速率。实现“固定功能定时器”模块（FF 定时器）中的定时器会导致 UDB 使用的资源最小。

- **Direct**（直接）– 不使用 FF 定时器或 UDB 资源，但将器件最大时钟频率限制为与模拟开关频率相同。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。如果启用，每个扫描槽均使用一个专用的 Analog Switch Divider（模拟开关分频器）值；如果不启用，则各传感器仅使用一个 Analog Switch Divider（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**（启用），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调制电容 C_{MOD} 和从调制电容 C_{MOD} 切换。**Analog Switch Divider**（模拟开关分频器）对 CapSense CSD 时钟进行分割，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	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

注表 1 的扫描时间基于以下设置估算而来。主控时钟和 CPU 时钟 = 48 MHz，CapSense CSD 时钟 = 24 MHz，通道数量 = 1。扫描时间按一次传感器扫描的时间间隔来测量。此时间包括传感器

设置时间、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 EMI 降低

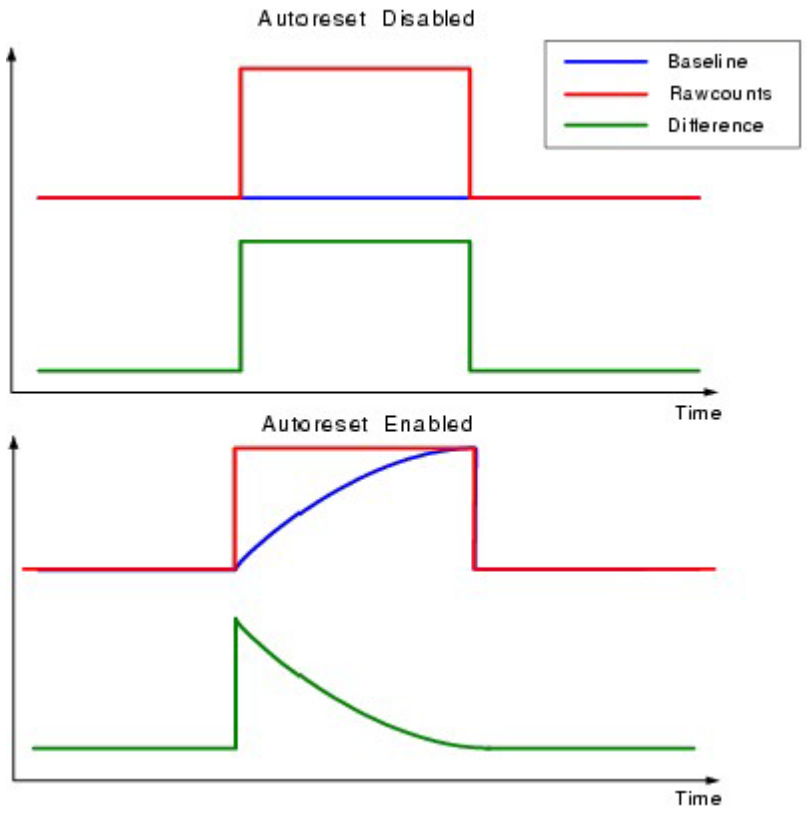
此参数指出是否将采用 Psuedo Random Sequence （伪随机序列） (PRS) 发生器来生成模拟预充电时钟。建议使用 PRS，因为它可以扩大 CapSense 模拟开关频率的频谱，从而减少电磁干扰辐射和灵敏度。PRS 时钟源通过 **Analog Switch Divider**（模拟开关分频器）设置提供。如果没有启用 PRS EMI 降低，则将使用单一频率，从而造成基频和谐波的辐射增加。

- **禁用**
- **启用 8 位** – 8 位提供更好的信噪比，但其较短的重复周期会增加电磁干扰。
- **启用 16 位，全速**（默认）– 16 位提供更低的信噪比，而且减少电磁干扰的效果更佳
- **启用 16 位，1/4 全速** – 与“**启用 16 位，全速**”相比，要求 4 倍的更高时钟频率，才能获得相同的 PRS 时钟输出

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**”。

- **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 10s），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。
- **禁用**（默认）– 系统状况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，按键可以继续永远报告其被按压状态。设计人员可能需要提供一种针对应用的方法，确定按键是否卡住或无反应。



Widget 分辨率

此参数指定 Widget 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，应进行调节，将额定值移到近似中间范围 (~128)。需要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）

负噪声阈值

本参数指出复位到原始计数级别的基线，其原始计数和基线级别之间的负的差值。它用于 **Sensor Auto Reset**（传感器自动复位）参数为 **Enabled**（启用）时。

低基准线复位

该参数定义了复位基准线到原始计数必须经过的采样次数，这种采样中原始计数必须小于基准线值。

屏蔽

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 Shield Electrode Use and Restrictions（屏蔽电极使用及限制）一节。

- **禁用** (默认)
- **启用**

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

- **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 模拟** – 使非活动状态传感器处于高阻抗模式。
- **屏蔽** – 给所有未扫描传感器提供屏蔽波形。屏蔽信号的振幅等于 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。

防护传感器

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**防水并检测**（在**常规**选项卡下），则启用此功能。有关防护传感器的更多信息，请参考此数据手册功能描述一节中的防护传感器实现 。

- **禁用** (默认)
- **启用**

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 电流吸收**或 **IDAC 源**要求在 PSoC 器件上使用硬件 IDAC。**外部电阻**使用用户在 PCB 上提供的电阻，而不是 IDAC，这在限制 IDAC 的应用中很有帮助。

- **IDAC 电流源**（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，提供电流吸收。建议在大多数设计中使用 **IDAC 源**，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 电流吸收** – IDAC 从调制电容 C_{MOD} 接收电流。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。尽管其信噪比通常没有 **IDAC 源**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 电流吸收配置功能相同。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**电流源**的 IDAC 范围。如果**电流源**设置为**外部电阻**，则此参数被禁用。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**电流源**被设置为 **IDAC 源**或“**IDAC 吸收**”，则此功能不可用 。支持多个泄露电阻，以允许协助系统调节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）**–** 大多数灵活实现，但使用有价值的 UDB 资源
- **FF 定时器 –** FF 定时器实现释放 UDB 资源，但不支持**扫描速度 =非常快**的情况。

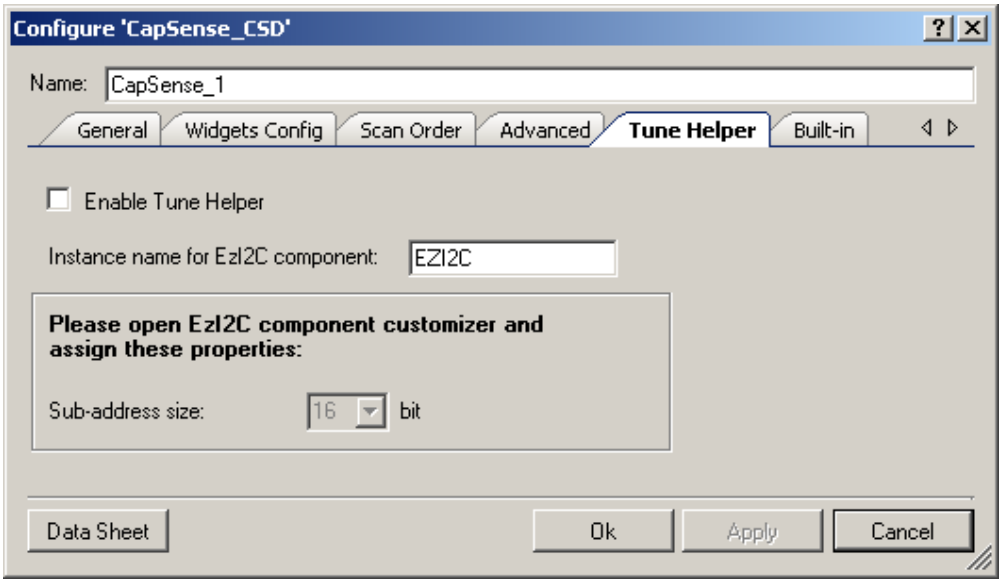
电压基准源

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

- **Vref 1.024V**（默认）**–** IDAC 电流吸收模式的最佳选择
- **Vdac –** IDAC 源模式的最佳选择。允许使用电压 DAC 调整基准电压，以最大化可用范围。仅在**电流源**被设置为 **IDAC 电流源**时且需要 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。

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。

调节助手选项卡



启用调节助手

此参数添加多种功能，使用户可轻松与调节器 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，当调节完成或调节方法改变时，无需删除这些功能。默认为被禁用。

EZI2C 组件的实例名称

此参数定义您的设计中将用于与调节器 GUI 进行通信的 EZI2C 组件的实例名称。

注不存在实时设计规则检查，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。

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

资源

下表显示 CapSense CSD 模拟和引脚资源。

资源	模拟资源			引脚（每个外部 I/O）
	VIDAC	比较器	CapSense 缓冲器	
通道：1 电流模式：外部电阻	0	1	1	2 + 屏蔽 + 传感器数量
通道：2 电流模式：外部电阻	0	2	2	4 + 屏蔽 + 传感器数量
通道：1 电流模式：IDAC 电流吸收	1	1	1	1 + 屏蔽 + 传感器数量
通道：2 电流模式：IDAC 电流吸收	2	2	2	2 + 屏蔽 + 传感器数量
通道：1 电流模式：IDAC 电流源 Vref：VDAC	2	1	1	1 + 屏蔽 + 传感器数量
通道：2 电流模式：IDAC 电流源 Vref：VDAC	4	2	2	2 + 屏蔽 + 传感器数量

下表显示 CapSense CSD 数字资源 （仅扫描和休眠 API 被包含在闪存和 RAM 的使用中）

说明	数字资源						API 存储器（字节）	
	数据路径	宏单元	状态寄存器	控制寄存器	计数器 7	中断	闪存	RAM
通道：1 电流模式：外部电阻	4	19	0	1	1	1	1270	11
通道：2 电流模式：外部电阻	6	31	0	1	1	2	2262	17
通道：1 电流模式：IDAC 电流吸收	4	19	0	1	1	1	1345	10
通道：2 电流模式：IDAC 电流吸收	6	31	0	1	1	2	2446	15
通道：1 电流模式：IDAC 源 Vref：VDAC	4	18	0	1	1	1	1452	11
通道：2 电流模式：IDAC 源 Vref：VDAC	6	30	0	1	1	2	2656	17

下表显示 CapSense CSD 高级 API 资源。

项目描述	API 存储器（字节）	
	闪存	RAM
Widget 类型：按键 计数：4	1197	22
Widget 类型：非双工线性滑条 大小：5 个传感器	1866	25
Widget 类型：双工线性滑条 大小：5 个传感器	2304	25
Widget 类型：矩阵按键 大小：5x5 个传感器	1526	55
Widget 类型：辐射滑条 大小：5 个传感器	1704	25

项目描述	API 存储器（字节）	
	闪存	RAM
Widget 类型：触控板 大小：5x5 个传感器	2289	48

调节器 GUI 用户指南

本节提供相关指令和信息，能帮助用户使用 CapSense 调节器。

CapSense 调节器在手动调节模式下，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 模式下显示调节值（只读）和性能。当组件处于不调节模式时不支持调节，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 调节过程

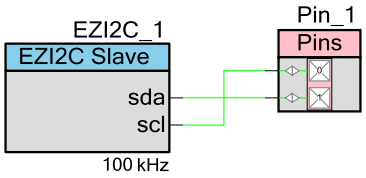
使用和调节 CapSense 组件的典型过程如下：

在 PSoC Creator 中创建设计

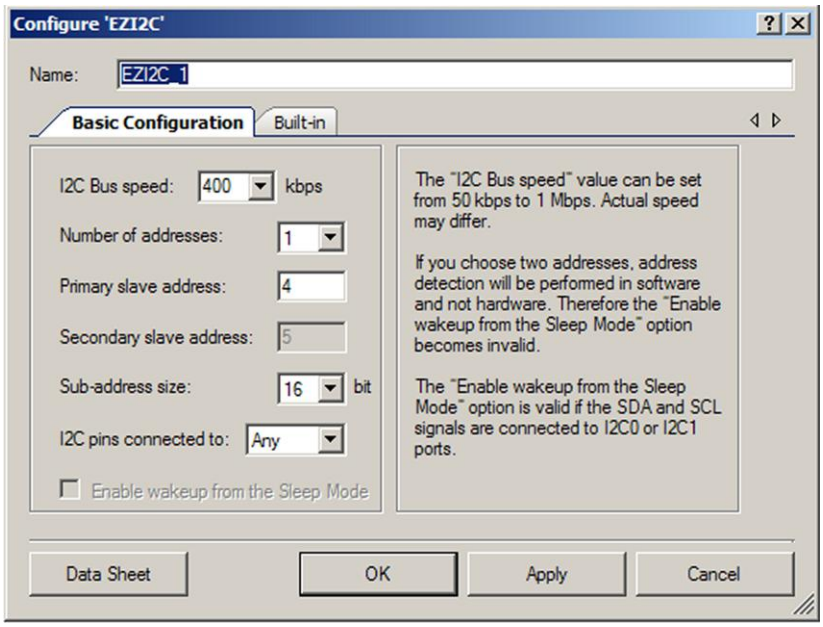
根据需要参考 PSoC Creator 帮助。

放置并配置 EZI2C 组件

1. 将 EZI2C 组件从组件目录拖动到您的设计中。

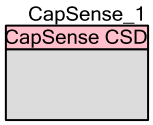


2. 双击可打开**配置**对话框。
3. 更改如下参数，然后单击 **OK**（确定）关闭对话框。
   - ❑ 辅助地址大小必须为 16 位。
   - ❑ 实例名称必须匹配用于 **CapSense CSD 配置**对话框在**调节助手**选项卡下的名称，这样生成的 API 才能发挥功能。

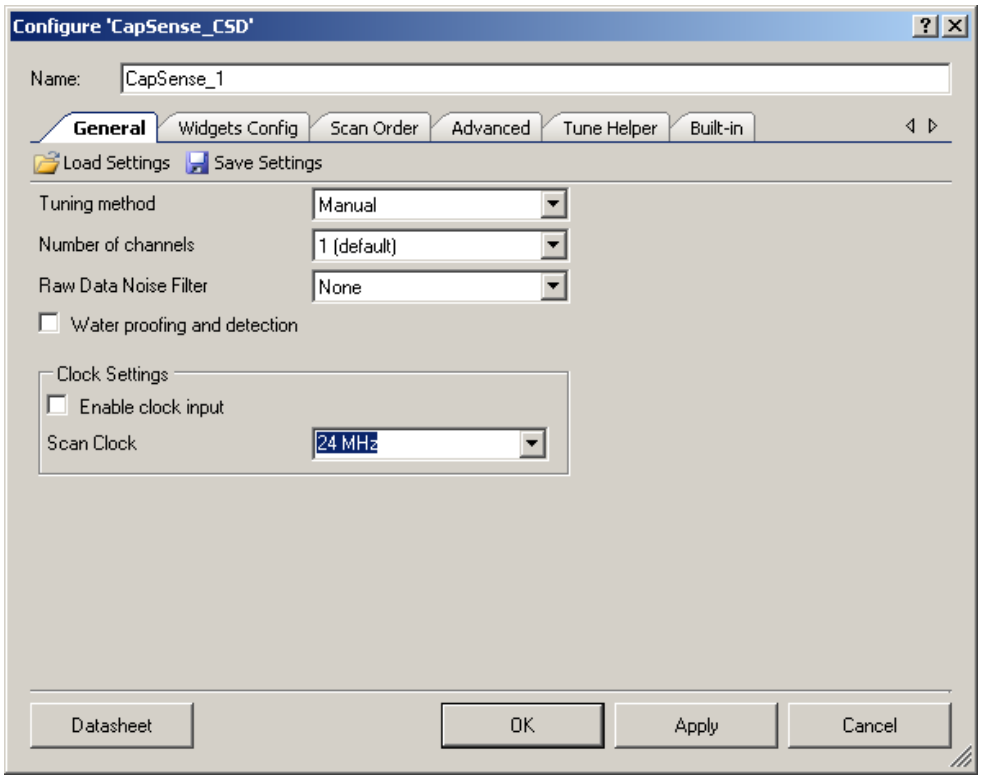


放置并配置 CapSense 组件

1. 将 CapSense_CSD 组件从组件目录拖动到您的设计中。



2. 双击可打开**配置**对话框。
3. 根据您应用的需要更改 CapSense CSD 参数。将**调节方法**选择为**手动**或**自动 (SmartSense)**。单击**确定**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节 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)消除了手动调节 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)调节以下 CSD 参数：

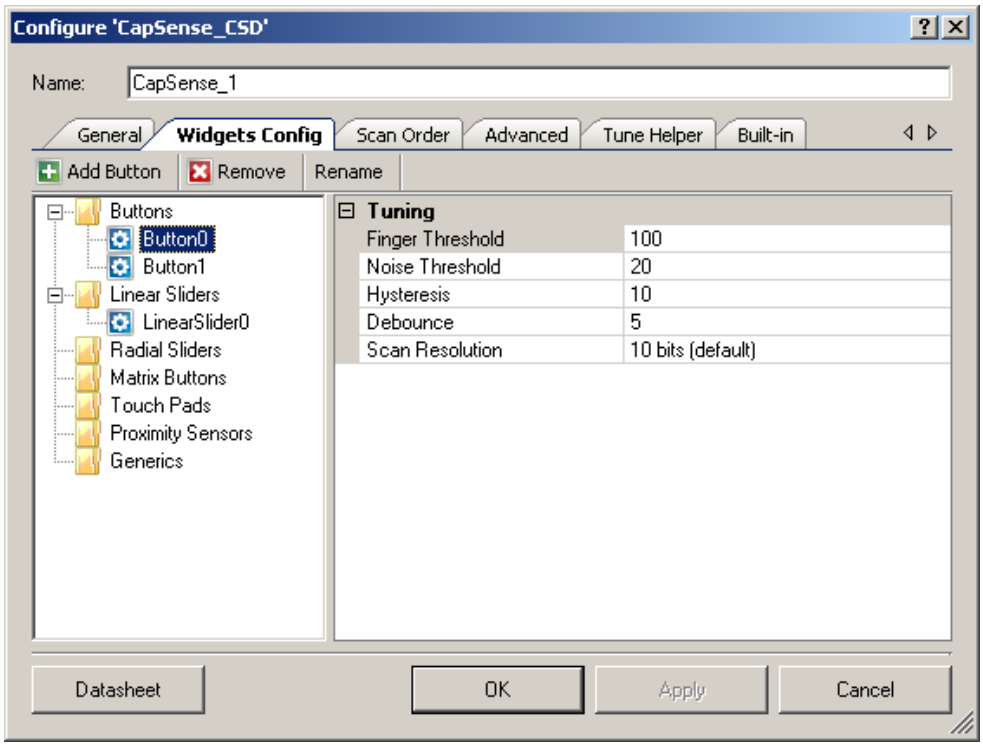
参数	计算
手指阈值	在传感器扫描期间连续计算。
噪声阈值	在传感器扫描期间连续计算。
IDAC 值	在 CapSense CSD 启动时计算一次。
模拟开关分频器	在 CapSense CSD 启动时计算一次。
扫描分辨率	在 CapSense CSD 启动时计算一次。

以下是自动 (SmartSense)调节方法的硬件参数限制：

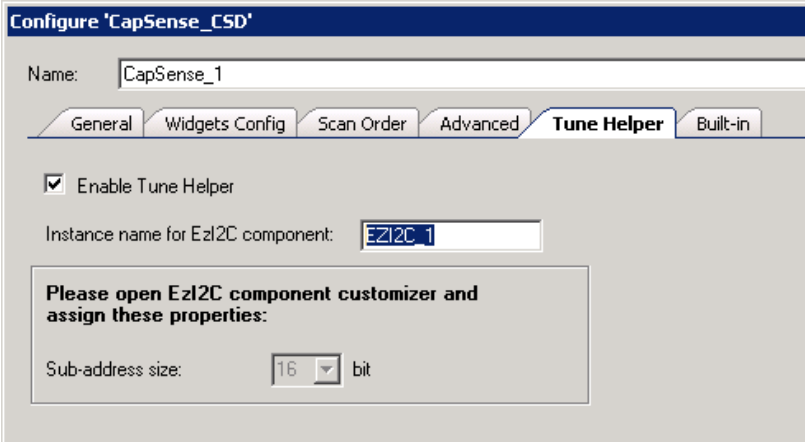
参数	所需的设置
扫描时钟	时钟必须为内部时钟（**常规**选项卡下的**启用时钟输入**被清除）。
电流源	IDAC 电流源。
PRS EMI 降低	启用 16 位。
扫描速度	正常
Vref	1.024 V

配置用户 CapSense 组件

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



2. 在 **Tune Helper**（调节助手）选项卡中：必须输入 EzI2C 组件实例名称，且必须选中 **Enable Tune Helper**（启用调节助手）复选框。



添加代码

- 将调节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：

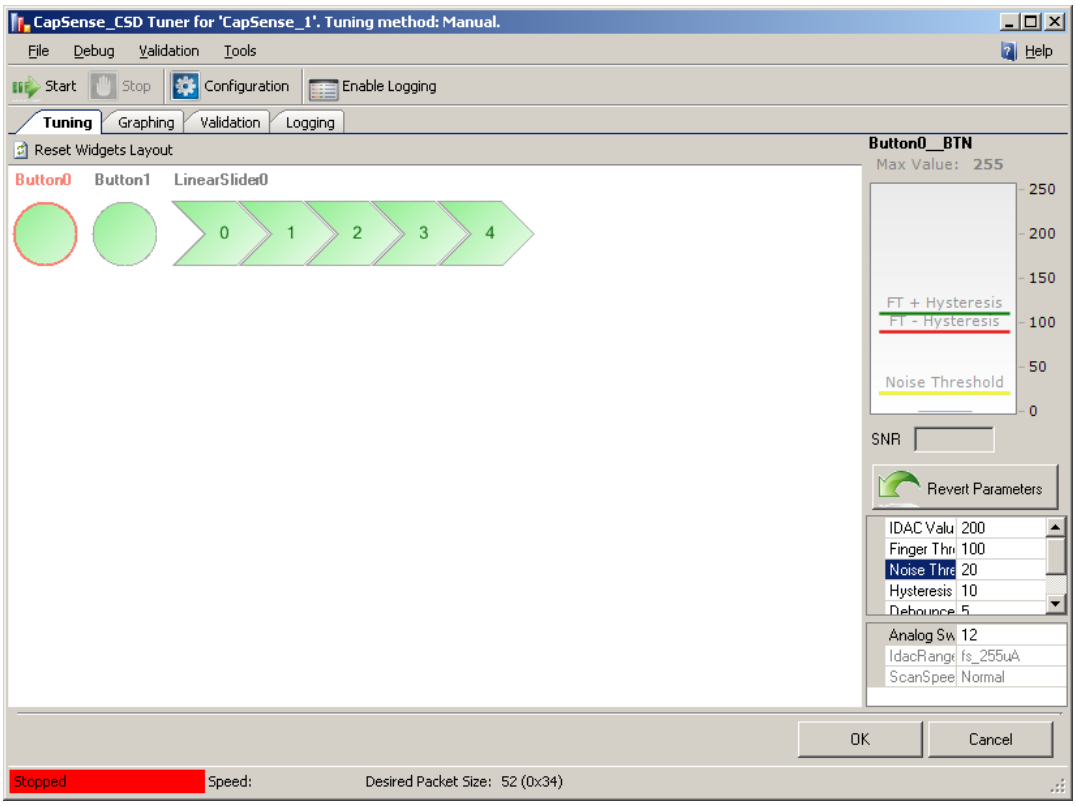
```
void  main()
{
    CYGlobalIntEnable;
    CapSense_1_TunerStart();
    while(1)
     {
        CapSense_1_TunerComm();
     }
}
```

构建设计并对 PSoC 器件进行编程

根据需要参考 PSoC Creator 帮助。

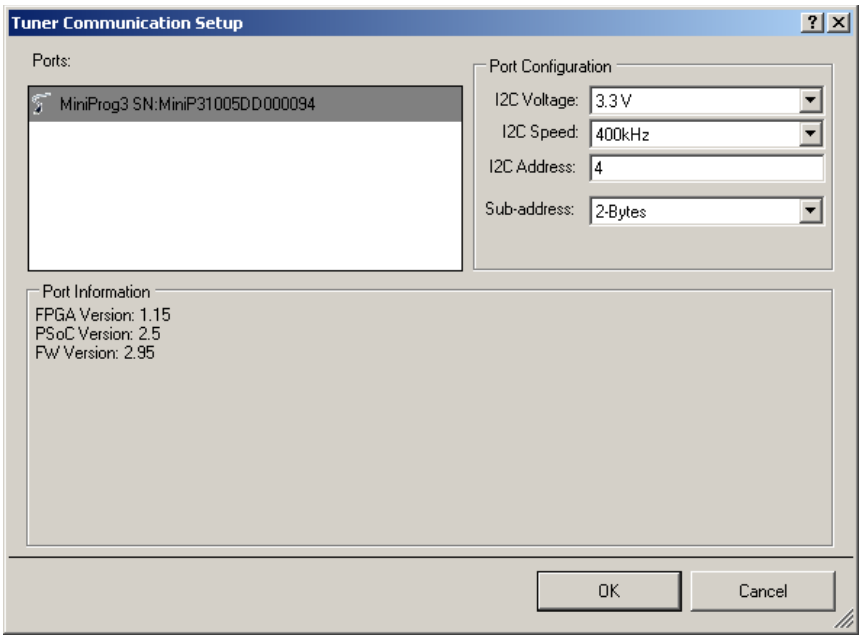
启动调节器应用

- 右击 CapSense CSD 组件图标，然后从上下文菜单中选择 **Launch Tuner**（启动调节器）。调节器应用将打开。



Configure Communication Parameters（配置通信参数）

1. 单击 **配置**以打开“**调节器通信**”对话框。



2. 设置通信参数，然后按**确认**。

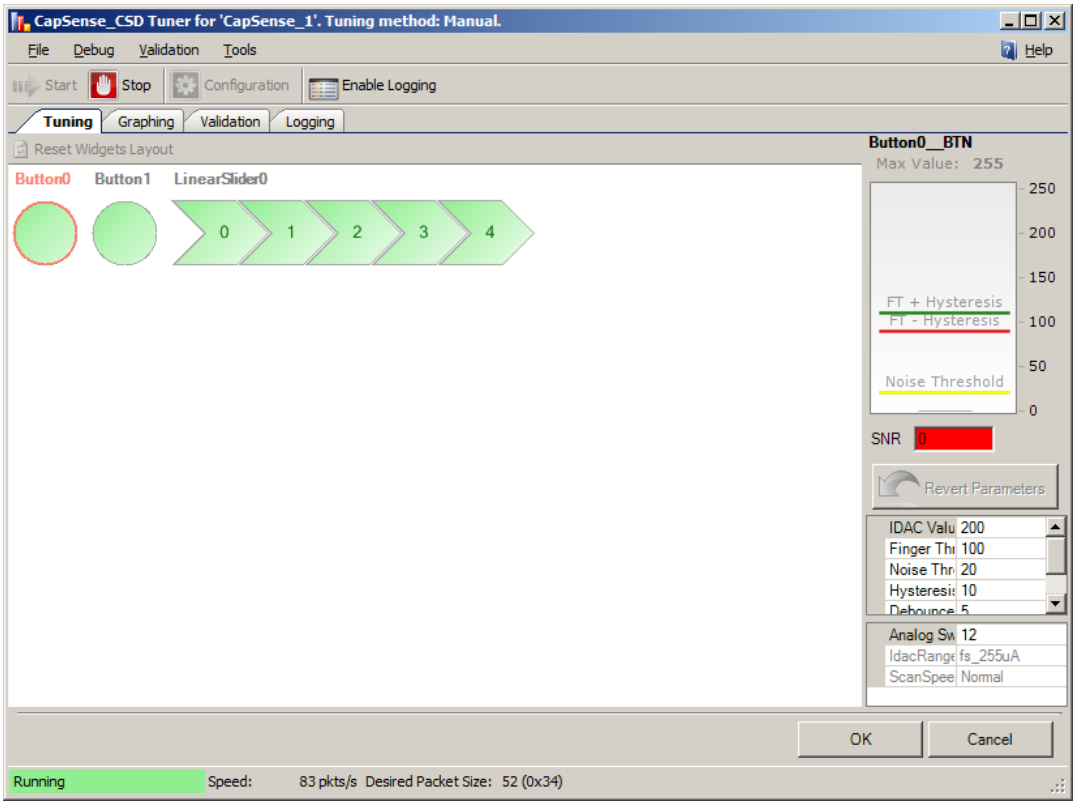
重要说明：属性必须与 EZ I2C 组件中的属性完全一致：**I2C 总线速度**，**I2C Address**，**Sub-address**（辅助地址）= 2 字节。

开始调节

- 单击调节 GUI 上的**开始**。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节器 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，且 CapSense 组件提供可靠的触控传感器结果。

关闭调节器应用

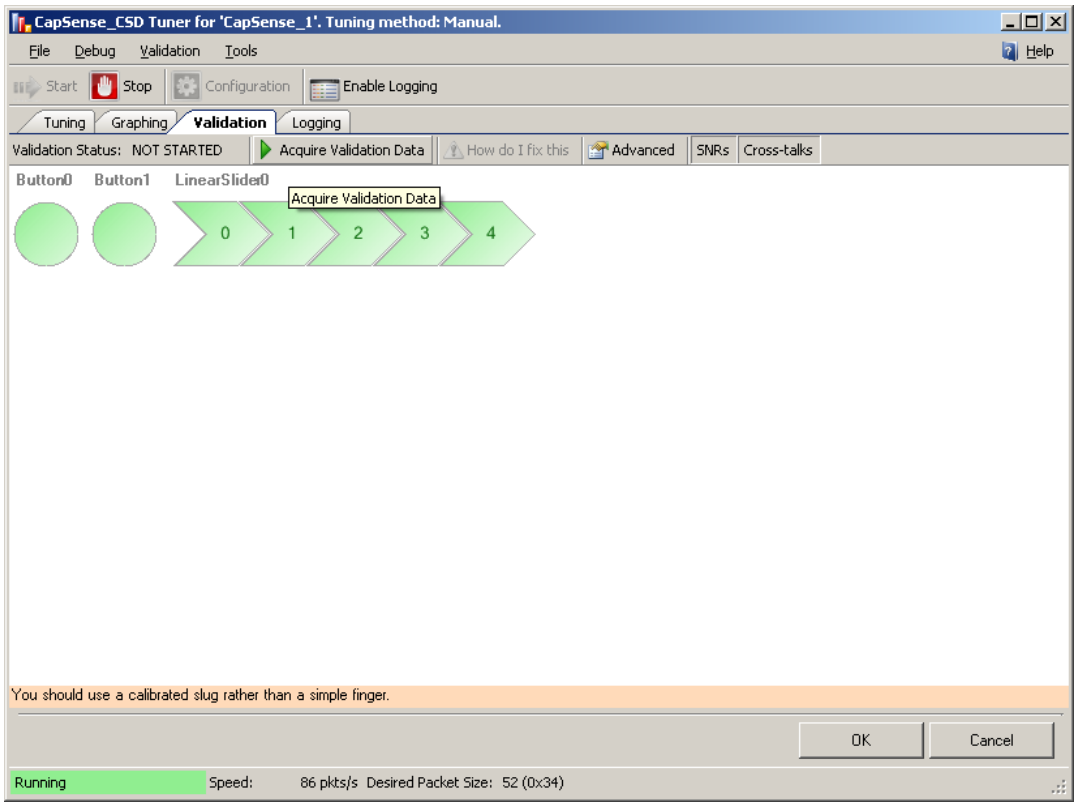
- 单击 **确定**，参数被写回 CapSense_CSD 实例。Tuner（调节器）应用对话关闭。

CapSense 验证过程

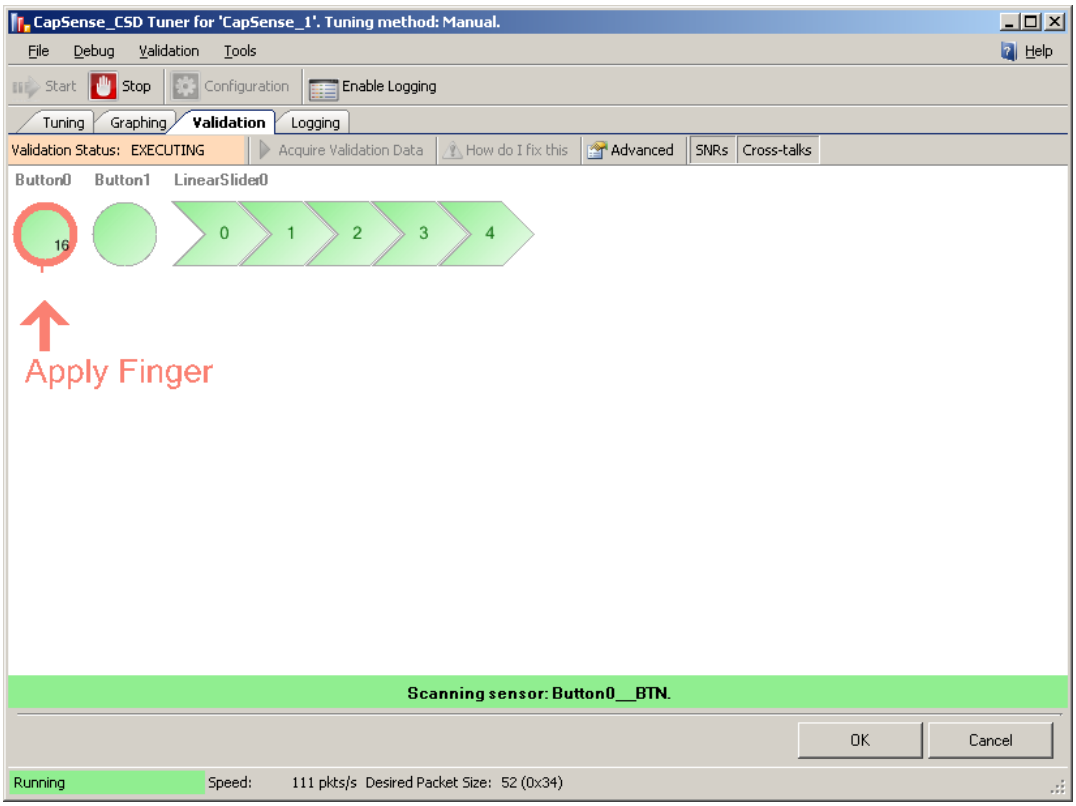
验证机制确定电路板是否已经过充分调节。下面是使用（调节器验证）特性来验证 CapSense 设计的典型过程。

开始验证

1. 开始扫描前，必须准备好 Tuner 和硬件。准备系统进行扫描的有关内容，参见前面 CapSense 调节过程一节。
2. 在 **Validation**（验证）选项卡，单击 **Acquire Validation Data**（获取验证数据）。开始出现所有 CapSense 元件对应的值。



激励传感器

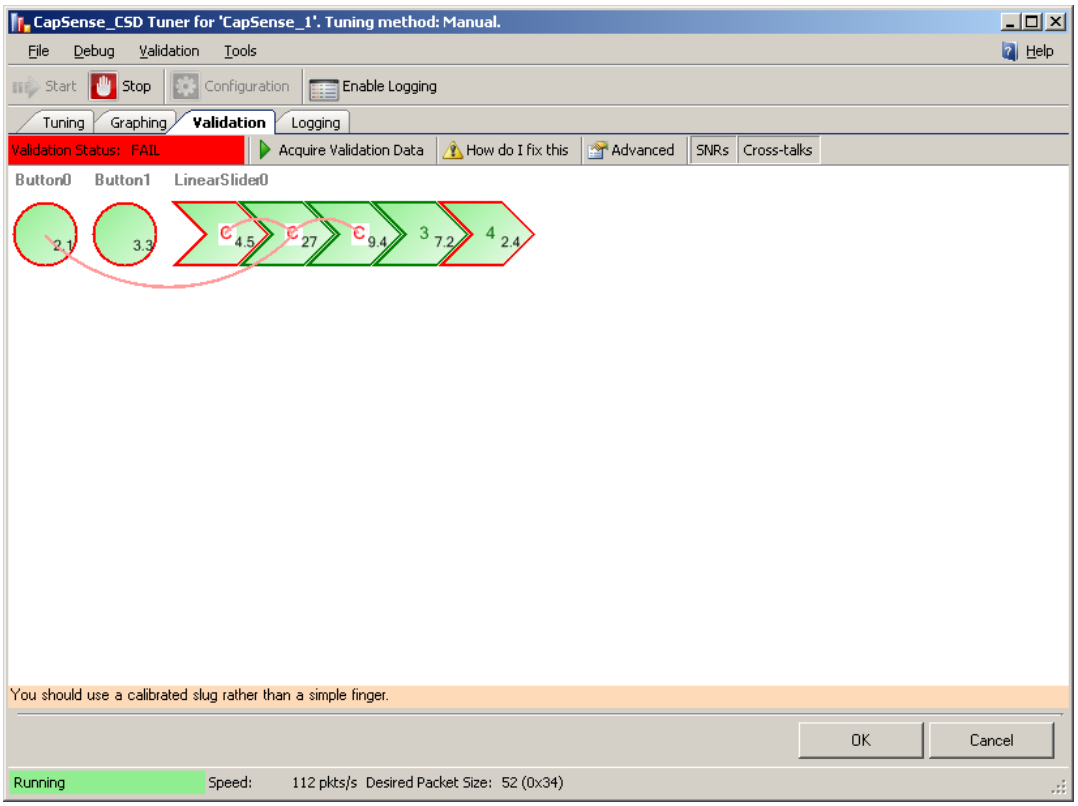


会有屏幕提示你在各传感器上用手指触摸。每次提示您按压 CapSense 元件时，会有闪烁的红色箭头出现在布局图上指向目标，并有文字提示 **PRESS HERE**（按这里）。Tuner 下方出现的文字将指导用户完成验证过程。

- 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小棒，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 **Sufficient Value**（足够值）的任何 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 信噪比 高于 **Optimal Value**（最优值）的任何 CapSense 传感器。

串扰效应警告出现如下：

- **Individual Crosstalk Check**（个体串扰检查）。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，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（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。

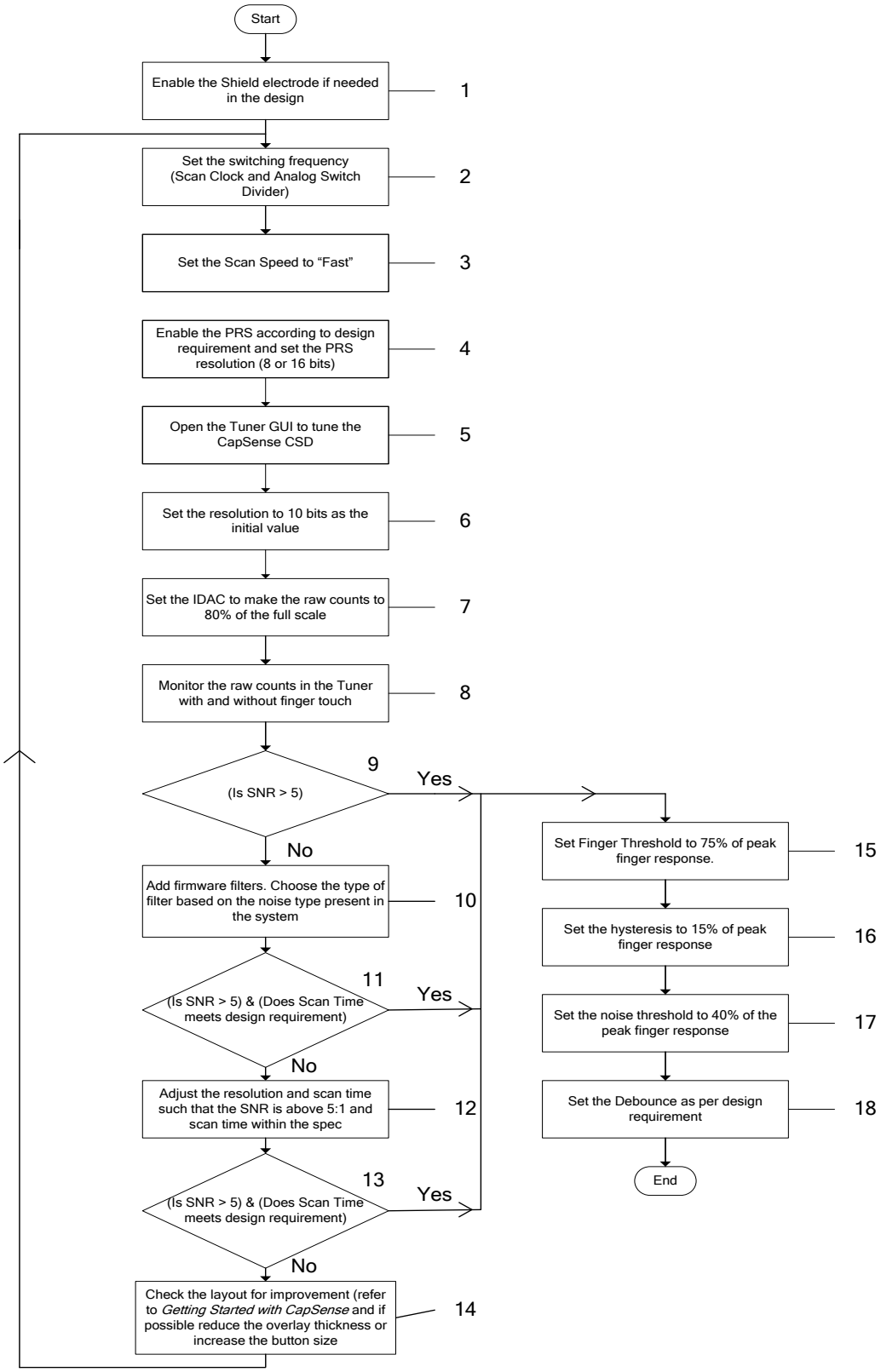
验证结果

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 **Optimal Value**（最优值）的任何 信噪比 值
- 低于 **Sufficient Value**（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**（怎样修复）按钮，也可以打开 Validation Report（验证报告）。

手动调节过程


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. Shield（屏蔽）根据设计要求进行启用或禁用。在传感器丝印层可能变湿（参见 AN2398）的应用中，屏蔽有用，可用来屏蔽 EMI，或降低过高的 Cp。如果当前模式被设为 **IDAC Sourcing**（IDAC 源）、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，参见本文档后面的屏蔽电极一节。
2. 各电容式传感器的开关频率的设置，应使传感器能被完全充放电。**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**（模拟开关分频器）（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。

   要检查传感器充放电是否完全，可检测传感器各引脚。注意，在用示波器探头观测传感器电容器的电压时，探头的电容也被增加到了传感器寄生电容。使用探头的 10x 档，可降低探头的电容。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**（模拟开关分频器）的值。在 Tuner （调节器）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. The **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。在易于产生 EMI 效应的设计中，会启用它。根据扫描时间，PRS 的分辨率被设为 8 或 16 位。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 Tuner 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。

   调节过程 10 的分辨率是不错的初始值，然而如果设计的丝印层薄于 1mm，也可以使用第一点的 8 和 9 的分辨率，作为初始值。
7. 在 GUI 中更改 IDAC 值，直至原始计数达到满标度值的 80%。满标度值为 2^Resolution。注意降低 IDAC 值可增加原始计数，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. 监控有手指出现和无手指出现时的原始计数。注意峰峰噪声和峰值指压反应。计算 信噪比 如下，

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

9. 对于好的 CapSense 设计，信噪比 应高于 5。检查总扫描时间是否适合于设计。如果未达到 信噪比 要求，请增加固件滤波器。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。

10. 检查 信噪比，如步骤 8 所示。同时，检查是否满足了设计的扫描时间要求。点击 GUI 上的 **OK**（确定）按钮，会将 GUI 中经过调节的值更新到组件中。根据参数设置计算出了扫描时间的近似值。扫描时间显示于扫描命令选项卡。如果设计中分辨率高且扫描速度低的传感器有很多，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。

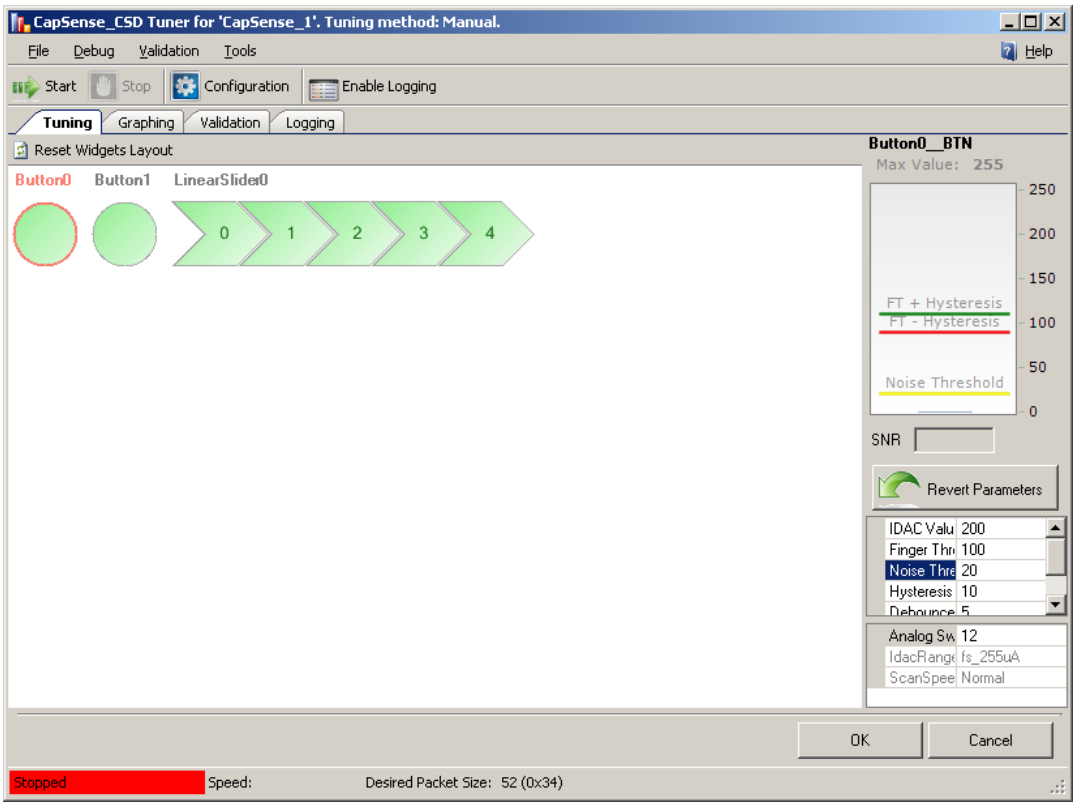
    如果 信噪比 低于 5，增加分辨率、扫描速度，或者两者。这样扫描时间会变长。因此，分辨率和扫描时间两者均应进行调节，以便达到高于 5 的信噪比、并使扫描时间低于设计规定。再次检查 信噪比 和扫描。如果不能达到 5:1 的 信噪比 并将扫描时间限制在设计规定范围内，请从 PCB 布局或丝印层设计中寻找可改进之处。参考 *Getting Started with CapSense*（CapSense 入门指南），了解 PCB 设计指导。您也可以减小丝印层的厚度、或增加按键直径，都可以增加敏感度。

11. 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。将该参数设置为手指反应峰值的 75%。
    - ❑ 将迟滞设置为手指反应峰值的 15%。
    - ❑ 将噪声阈值设置为手指反应的 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点的值，比如 5 。

调节器 GUI 界面

一般界面



顶部面板按钮如下：

- **开始**（或主菜单项**调试 > 开始**）– 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。
- **停止**（或主菜单项**调试 > 停止**）– 停止从芯片上读取和显示数据。
- **配置**（或主菜单项**调试 > 配置**）– 打开**通信配置** 对话框。
- **启用日志**（或主菜单项**调试 > 开始**）– 启用将从器件接收到的数据日志到日志文件。

主菜单：

- **文件 > 设置 > 从文件加载设置** – 从 XML 调节文件导入设置并将所有数据加载到调节器。
- **文件 > 帮助** – 打开帮助文件。

  其他项重复上面板和底面板按键的功能。

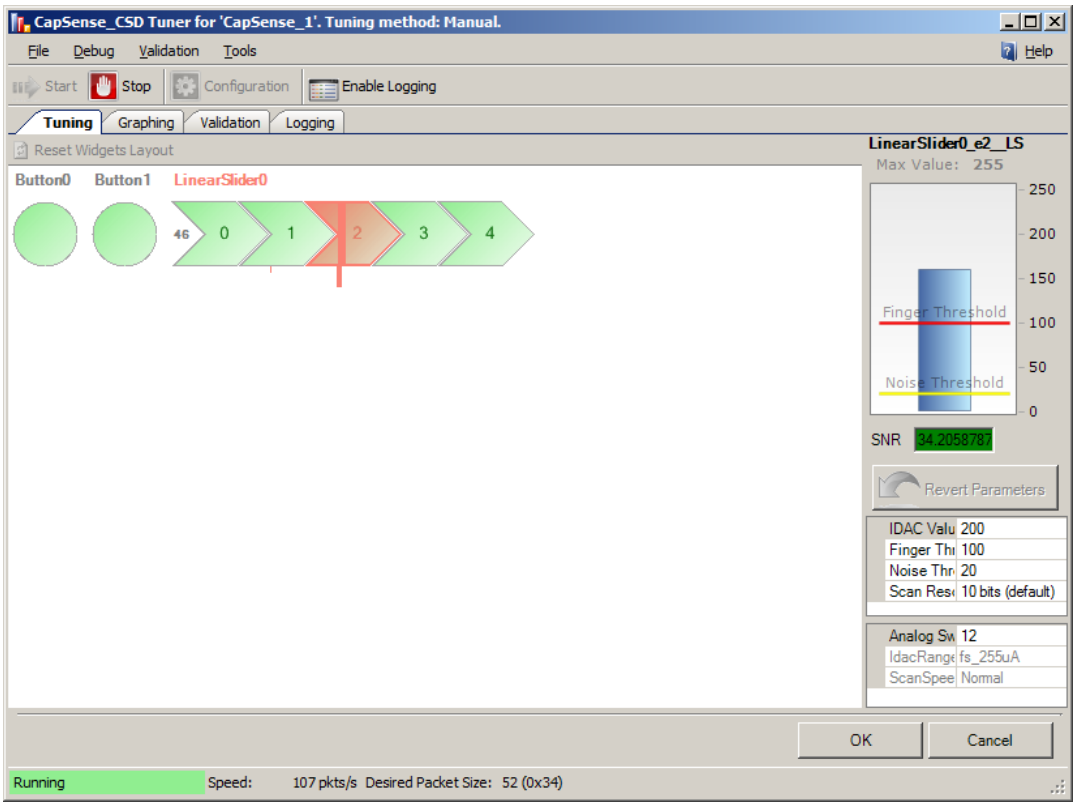
选项卡：

- **调节选项卡 –** 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。
- **图解选项卡 –** 在图表上详细显示单个 Widget 数据。
- **日志选项卡 –** 提供日志数据功能和调试功能。

底面板按键：

- **确定**（或主菜单项**文件 > 应用更改并关闭**）– 将参数的当前值应用到 CapSense 组件实例，并退出 GUI。
- **取消**（或主菜单项**文件 > 退出**）– 退出 GUI 而不将参数值应用到组件实例。

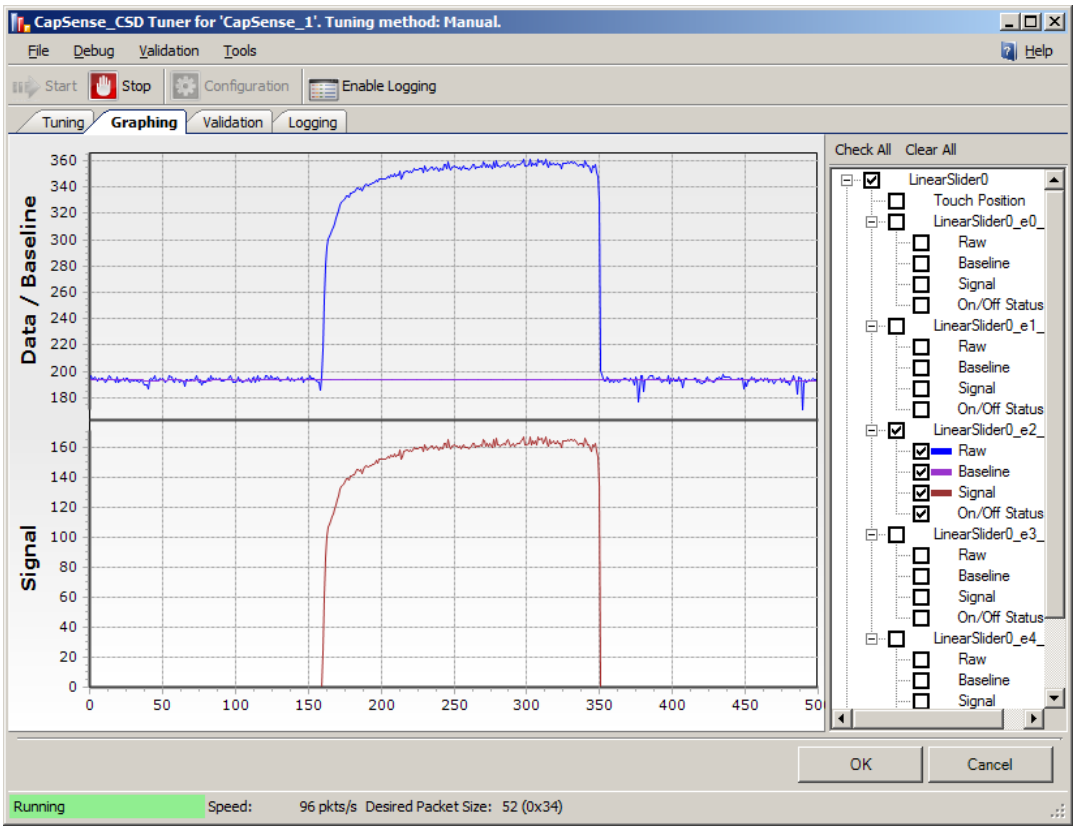
调节选项卡



- **Widget 原理图 –** 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 Widget Layout** （重置 widget 布局）按钮 – 将各 widget 移动到原理图内的默认位置。

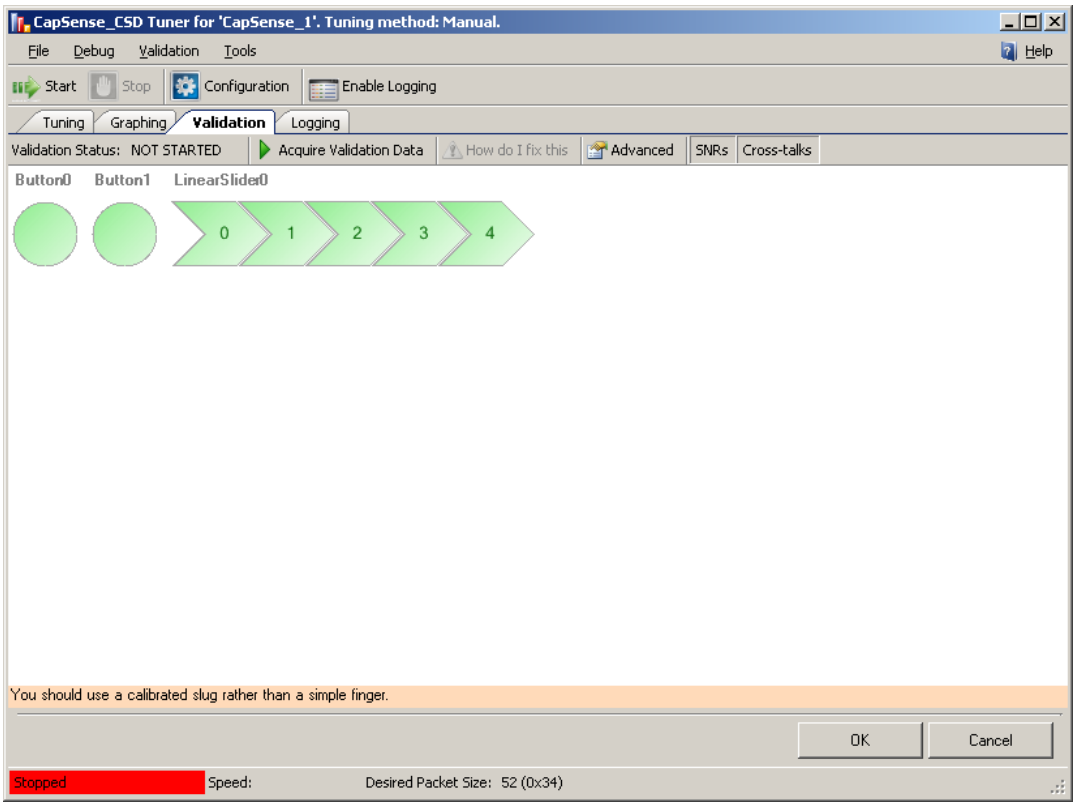
- **条形图 –** 显示选定传感器的信号值。
  - 通过双击 Max Value（最大值）标签，可调整到详细视图条形图的最大比例。有效范围介于 1 和 255 之间，默认值为 **255**。
  - 当前手指打开阈值显示为穿过条形图的一条**绿线**。
  - 当前手指关闭阈值显示为穿过条形图的一条**红线**。
  - 当前噪声阈值显示为穿过条形图的一条**黄线**。
- **信噪比 –** 对选定传感器的信噪比进行实时计算。信噪比值低于 5 表示不佳，显示为红色；5 至 10 表示临界，显示为黄色；大于 10 表示良好，显示为绿色。信噪比值根据之前接收到的数据进行计算。
- **还原参数**按键 **–** 将参数重置为其初始值，并将这些值发送至芯片。初始值为启动 GUI 时所显示的值。
- **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。
- **CapSense 的一般特性**（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 Tab（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。On/Off Status（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（验证）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。

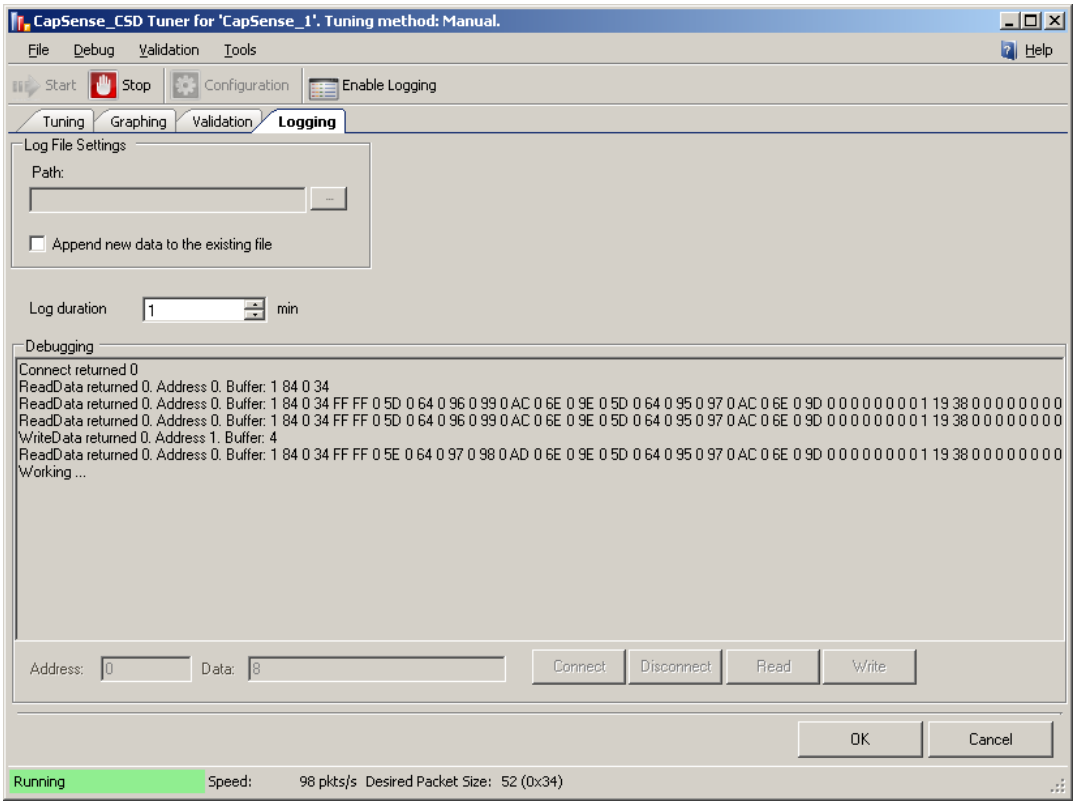
- **Widget 原理图 –** 包含所有配置的 Widget 的图形表示。

顶部面板控件：

- **Validation Status** （验证状态）标签 **–** 显示验证状态。它有以下信息：
  - **VALIDATION NOT STARTED**（验证未开始）**–** 自上次设计修改以来，未进行过验证过程。
  - **PASS**（通过）**–** 完整验证过程已完成，且无故障。
  - **FAIL**（未通过）**–** 验证过程发现了故障；将显示验证报告。
- **Acquire Validation Data**（获取验证数据）按钮（或主菜单项 **Validation**（验证）**> Acquire Validation Data**（获取验证数据））**–** 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**（怎样修复）按钮 **–** 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，该按钮才可用。

- **Advanced button**（高级按钮）（或主菜单项 **Validation**（验证）**> Validation Advanced properties**（验证高级属性））——打开验证属性的属性窗口（更多信息，参见 Validation Advanced Properties（验证高级属性））。
- **SNRs**（信噪比）按钮 – 在 widget 原理图上打开或关闭信噪比显示（更多信息，参见验证显示）。
- **Crosstalks**（串扰）按钮 – 在 widget 原理图上打开或关闭串扰显示（更多信息，参见验证显示）。

“选项卡”



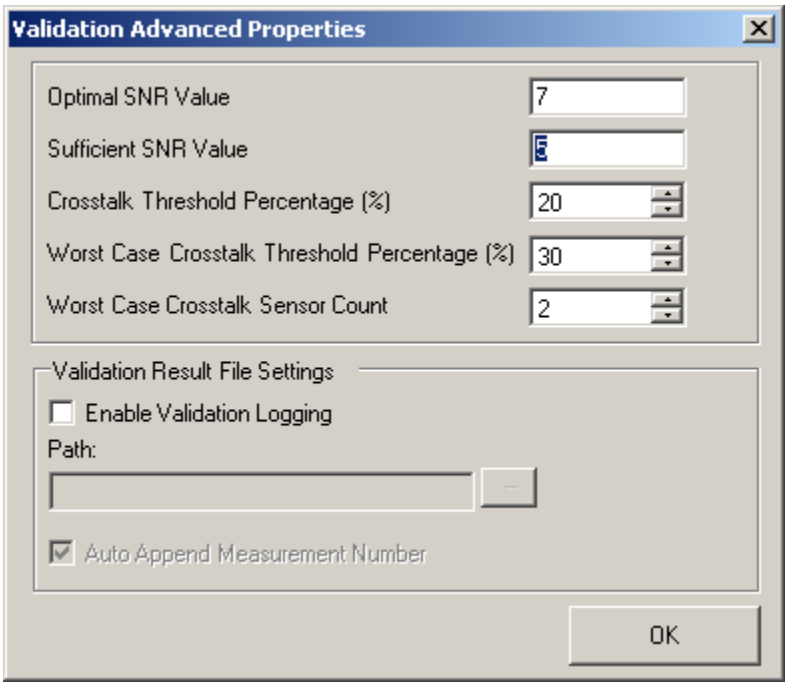
- 通过选中 **Graphing**（绘图）选项卡树视图中的复选框，可选定将要记录的数据。
- **Path**（路径）– 定义文件的路径（文件扩展名为 .csv）。
- **Append new data to existing file**（将新数据附加到现有文件）复选框 – 选中该选项后，新数据即会附加到现有文件中。如果未选中，旧文件将从文件中被擦除，代之以新数据。
- **Log duration**（持续时间）– 定义持续时间，以分钟为单位。有效范围介于 1 和 480 之间，默认值为 **255**。

Debugging （调试）组

该功能的存在仅为调试目的。它能帮助用户研究 Tuner 通信错误。

- 调试日志窗口 – 显示 Tuner 执行的通信命令。所有通信日志错误均被记录于此。如果成功启动了 Tuner，则仅记录最开始的几条通信命令。
- **Connect**（连接）– 连接至 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 器件中读取数据。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。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）



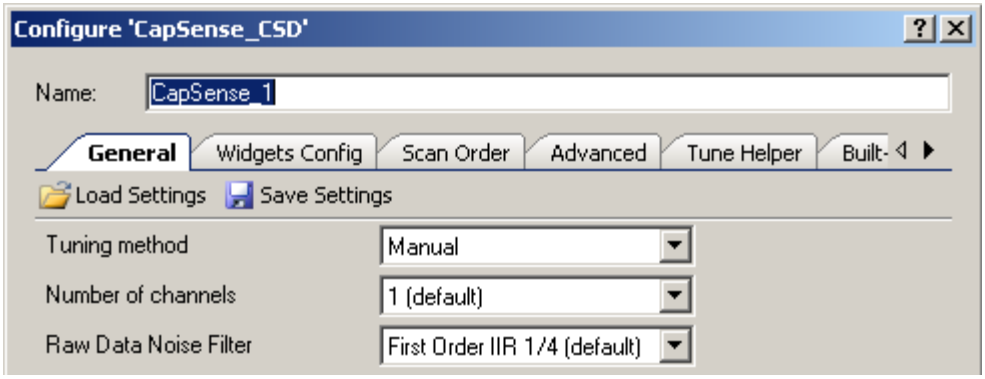
- **Optimal SNR Value**（信噪比最佳值）– 定义最佳的信噪比值。有效范围介于 1 和 100 之间，默认值为 **7**。
- **Sufficient SNR Value**（信噪比充分值）– 定义充分的信噪比值。有效范围介于 1 和 100 之间，默认值为 **5**。

- **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 1 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 1 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的传感器的数量；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**（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）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。

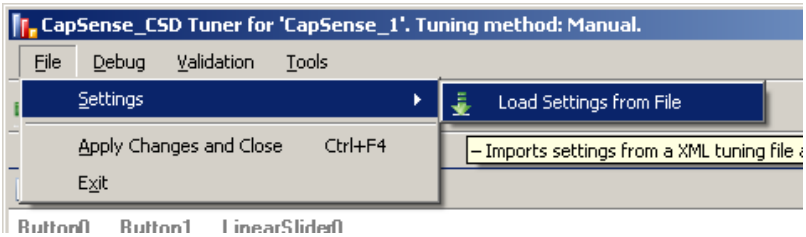
Save and Load Settings（保存/加载设置功能）

Tuner 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 组件调节器 GUI 的 Save and Load Settings（保存和加载设置）功能。

1. 单击定制器中的 **Save Settings**（保存设置）按键。



2. 在 **Save File**（保存文件对话框）窗口中，指定文件的名称及其保存位置。
3. 打开调节器窗口，并在 **文件 中单击 文件 > 设置 > 加载设置**。



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

应用程序编程接口

应用程序编程接口 (API) 例程允许您使用软件配置组件。下表对各个函数进行了概述。以下各节将详细介绍每个函数。

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，并启用 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 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() 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。启用传感器扫描中断。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 例程之前，必须先调用 CapSense_Start() 例程。

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 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

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

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 Active（活动）模式电源模板位，可还原组件的启用状态。

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 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

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

引脚 API 一节描述的函数。

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

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()	为下一个扫描周期中的扫描配置选定的传感器。
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

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

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： 为下一个测量周期中的扫描配置选定的传感器。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: 当前原始数据值

副作用： 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有序数。

返回值： 无

副作用： 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。原始数据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()	仅扫描启用的传感器，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称为这个传感器的基准线。更新后的基准线使用 k = 256 的低通滤波器。
CapSense_UpdateEnabledBaselines	检查 CapSense_SensorEnableMask[] 数组并调用 CapSense_UpdateSensorBaseline 函数，以更新已启用传感器的基准线。
CapSense_EnableWidget()	为扫描过程启用 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，以确定其是否存在指压。
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指压，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指压，并返回位置。
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] 数组元素。将原始计数值复制到每个传感器的基准线数组中。初始化原始数据过滤器（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[ ]。初始化原始数据过滤器（如果已启用）。

该函数仅在双通道设计中可用。

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（如果已启用）。

参数： 无

返回值： 无

副作用： 无

void CapSense_UpdateSensorBaseline(uint8 sensor)

说明： 传感器的基准线为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值中扣除并将其存储在 CapSense_SensorSignal[sensor] 中，来计算差值计数。

如果启用自动复位选项，则基准线将独立于噪声阈值进行更新。

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，基准线将会停止更新，而在信号值小于负的噪声阈值的情况下，该基准线将会复位。

若原始数据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，以更新所有已启用传感器的基准线。

参数： 无

返回值： 无

副作用： 无

void CapSense_EnableWidget(uint8 widget)

说明： 启用选定的 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 设定为活动状态。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（如果已启用）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（如果已启用）将应用于该结果中。只有当 CapSense 定制器定义了辐射滑条 Widget 时，此函数才可用。

参数： uint8 widget: Widget 编号。对每个辐射滑条 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（如果已启用）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。对于每个触控板 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：如果触摸板上存在手指，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。对于每个 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：如果触摸板上存在手指，则为 1，不存在手指，则为 0。

副作用：

调节器助手 API

这些 API 函数与调节器 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()	执行调节器 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启用的传感器启动传感器扫描循环。

参数： 无

返回值： 无

副作用： 无

void CapSense_TunerComm(void)

说明： 执行与调节器 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 组件传输至调节器 GUI。从调节器 GUI 读取新参数并将其应用于 CapSense CSD 组件。
- 自动 (SmartSense)：执行与调节器 GUI 之间的通信功能。将传感器扫描和 Widget 处理结果传输至调节器 GUI。自动调节参数也传输至调节器 GUI。调节器 GUI 参数不会被传输回 CapSense CSD 组件。

调节器 GUI 修改 CapSense CSD 组件缓冲器时，该函数正在执行阻止操作并等待，以允许新数据。

参数： 无

返回值： 无

副作用： 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。

函数	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) 模式：想要的驱动模式。有关驱动模式的信息，请参见引脚组件数据手册。

返回值： 无

副作用： 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) 模式：想要的驱动模式。有关驱动模式的信息，请参见引脚组件数据手册。

返回值： 无

副作用： 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) 模式：想要的驱动模式。有关驱动模式的信息，请参见引脚组件数据手册。

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。可以出于调试和调节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[ ] and 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——扫描传感器，0——不扫描传感器。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[ ] and 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准线数据分数字节，用于基准线更新。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准线数据。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计数中减去以前的基准线计算而来的传感器信号计数。数组大小等于传感器总数。**Widget 分辨率**参数将此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。

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

每个传感器均提供有一个常量。以下函数中可用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）。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 – 仅具有多个元件的 Widget（如辐射滑条）有元件号。对于触控板及矩阵按键，元件号由字“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 – 存在若干种 Widget 类型：

别名	说明
BTN	按键
LS	线性滑条
RS	辐射滑条
TP	触摸板
MB	矩阵按键
PROX	接近传感器
GEN	通用传感器
GRD	防护传感器

Widget 常量

每个 Widget 均提供有一个常量。以下函数中可用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
```

固件源代码示例

PSoC Creator 在“查找示例项目”对话框中提供了大量包括原理图和代码示例的示例项目。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开开始页或**文件**菜单中的对话框。根据需要，使用对话框中的**滤波器选项**可缩小可选项目的列表。

有关更多信息，请参考 PSoC Creator Help（PSoC Creator）帮助中的“Find Example Project”（查找示例项目）主题。

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。

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，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*

注在双通道设计中，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 × Vref

对于 IDAC 吸收模式：

C_{MOD} = 2.2 nF × (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 1 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。

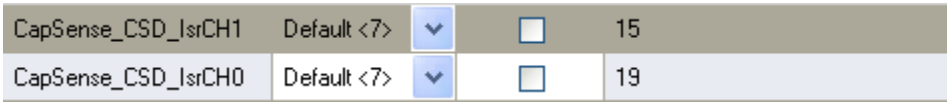
典型泄露电阻值为 500Ω 到 10 kΩ，具体取决于传感器电容。

中断服务例程

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，以便启用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。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”(On) 或“活动”(Active) 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”(Off) 或“非活动”(Inactive) 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 – 内插和测量

在滑条传感器和触摸板中，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摸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。

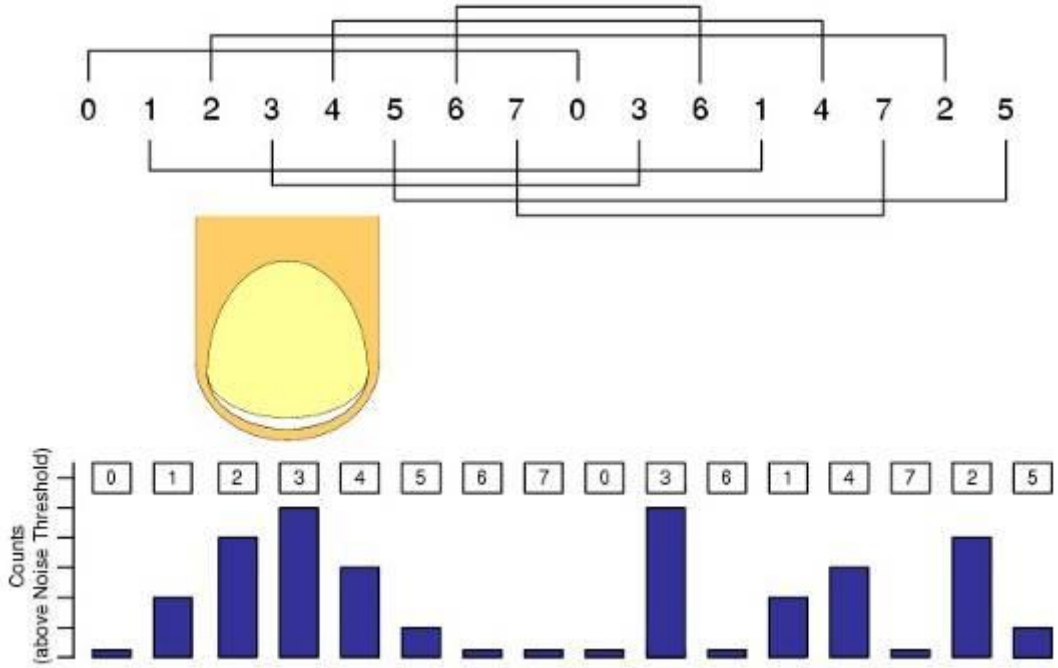
$$N_{Cent} = \frac{n_{i-1}(i-1) + n_i i + n_{i+1}(i+1)}{n_{i-1} + n_1 + n_{i+1}}$$

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 2 不同滑条段计数的双工序列

滑条段总计数	段序列
10	0,1,2,3,4,0,3,1,4,2
12	0,1,2,3,4,5,0,3,1,4,2,5
14	0,1,2,3,4,5,6,0,3,6,1,4,2,5
16	0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5
18	0,1,2,3,4,5,6,7,8,0,3,6,1,4,7,2,5,8
20	0,1,2,3,4,5,6,7,8,9,0,3,6,9,1,4,7,2,5,8
22	0,1,2,3,4,5,6,7,8,9,10,0,3,6,9,1,4,7,10,2,5,8
24	0,1,2,3,4,5,6,7,8,9,10,11,0,3,6,9,1,4,7,10,2,5,8,11
26	0,1,2,3,4,5,6,7,8,9,10,11,12,0,3,6,9,12,1,4,7,10,2,5,8,11
28	0,1,2,3,4,5,6,7,8,9,10,11,12,13,0,3,6,9,12,1,4,7,10,13,2,5,8,11
30	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0,3,6,9,12,1,4,7,10,13,2,5,8,11,14
32	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,3,6,9,12,15,1,4,7,10,13,2,5,8,11,14
34	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14

滑条段总计数	段序列
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
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：

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

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

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

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

时序抖动滤波器

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 峰到峰噪声的数据、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，这可导致器件运转出错。

防水和检测

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

- 启用屏蔽电极，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

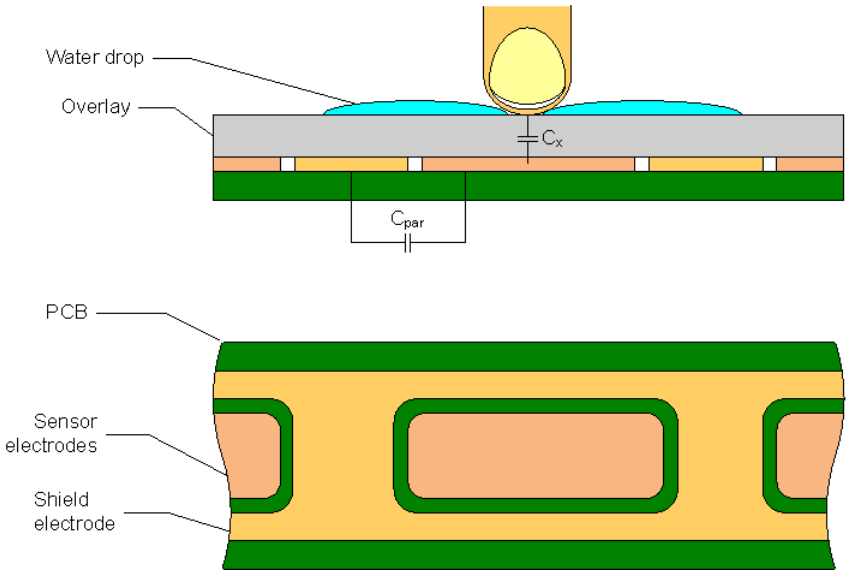


图 2 提供了一种按键屏蔽电极的可能布局组态。屏蔽电极尤其适用于透明的 ITO 触摸板器件，在这种器件中，它不但可阻止 LCD 驱动电极的噪声，同时可减少杂散电容。

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，其中包括按键下面的平板。对于这种情况，建议使用填充模式，填充率约为 30 ～ 40%。这时无需额外的接地层。

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。

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。此模式中不应使用到屏蔽的传感器连接，因为其提供了等于 Vddio 的输出。Vref = 1.024 伏设置具有布线限制，无法布线至引脚。

当前模式 IDAC 电流吸收能力和外部电阻

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。

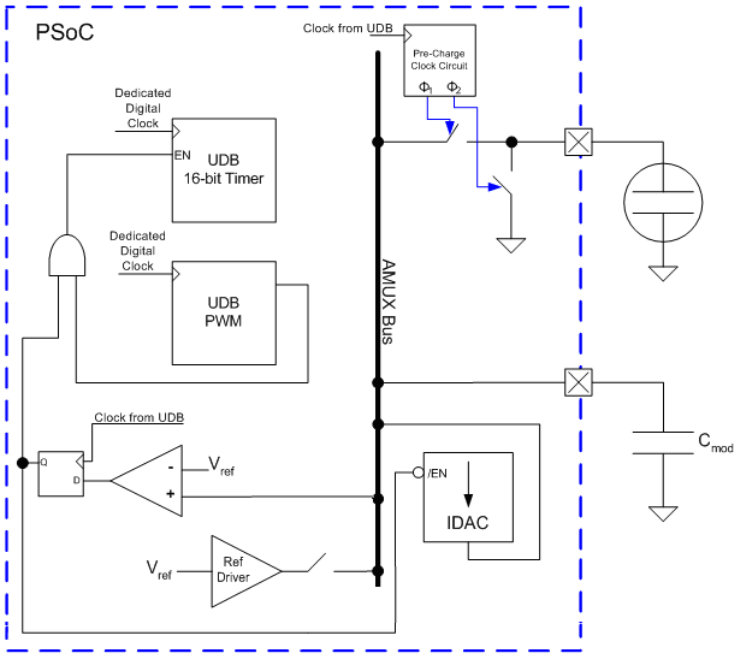
防护传感器在没有任何专门选项的情况下进行扫描。当防护传感器扫描时，屏蔽电极不会禁用。采用双通道设计的防护传感器始终最后一个进行独立扫描。

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。

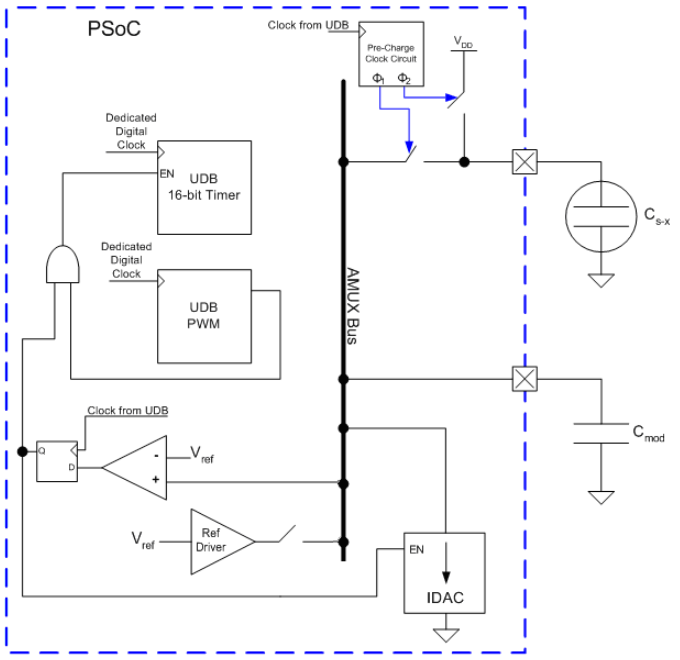
IDAC 电流输出

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。



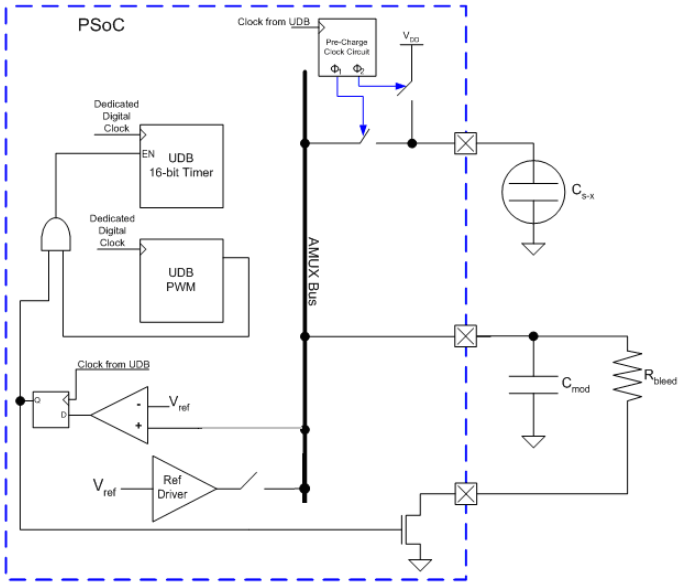
IDAC 电流吸收

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 配置为从传感器电流吸收。



IDAC 禁用，使用外部 Rb

使用外部泄露电阻 Rb 与 IDAC 电流吸收配置作用相同，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。



直流和交流电气特性

5.0-V/3.3-V 直流和交流电气特性

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

噪声

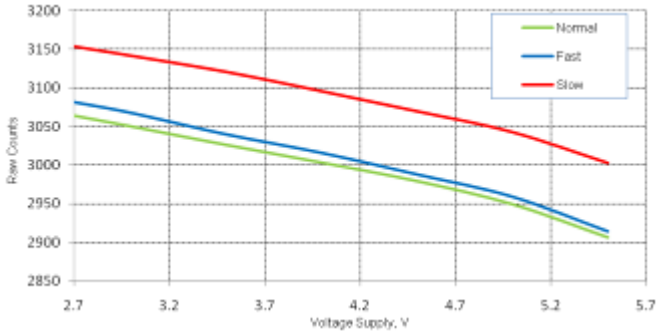
参数	测试条件和注释	最小值	典型值	最大值	单位
噪声计数，峰到峰（噪声计数/（基准线计数）	分辨率 = 16（噪声计数/（基准线计数）	-	0.2	-	%
	分辨率 = 14	-	0.3	-	%
	分辨率 = 10	-	1.0	-	%

功耗

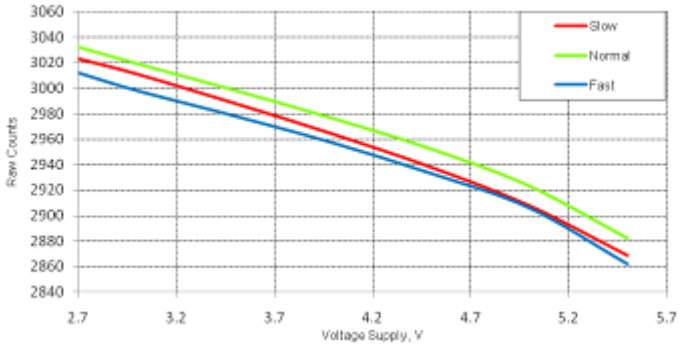
参数	测试条件和注释	最小值	典型值	最大值	单位
有功电流	V_{DD} = 3.3 伏，CPU 时钟 = 24 MHz，CapSense 扫描时钟 = 24 MHz，扫描期间的平均电流，8 个传感器	-	8	-	mA
睡眠/唤醒电流有 100ms 的报告速率	V_{DD} = 3.3 伏，CPU 时钟 = 24 MHz，CapSense 扫描时钟 = 24 MHz，扫描速度 = 超快，分辨率 = 9， 8 个传感器	-	78.9	-	µA
	V_{DD} = 3.3 伏，CPU 时钟 = 24 MHz，CapSense 扫描时钟 = 24 MHz，扫描速度 = 快速，分辨率 = 12， 8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

图 – 原始计数与电源对比

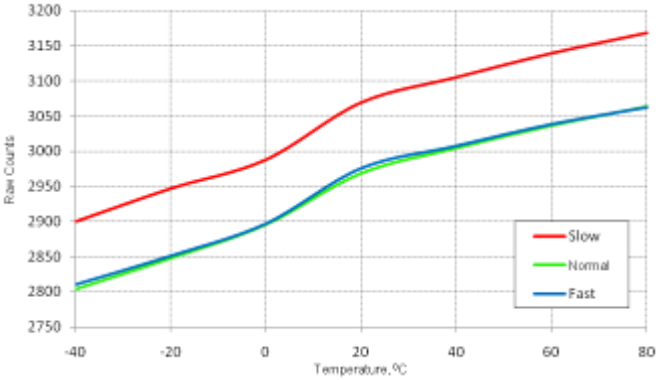
在不同扫描速度下，PRS 16 全速，原始计数与电源电压的关系



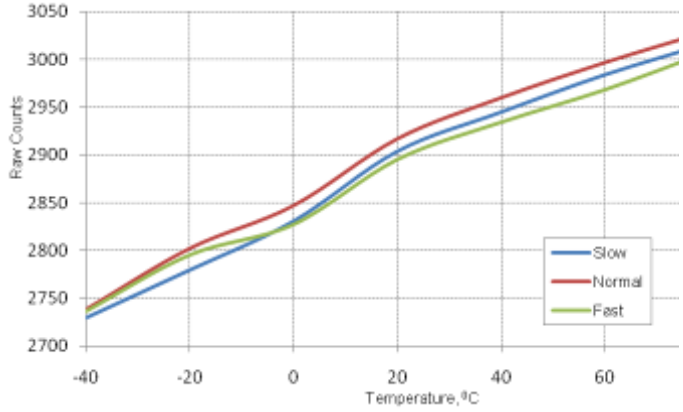
在不同扫描速度下，PRS 8，原始计数与电源电压的关系



在不同扫描速度下，PRS 16 全速，原始计数与温度的关系



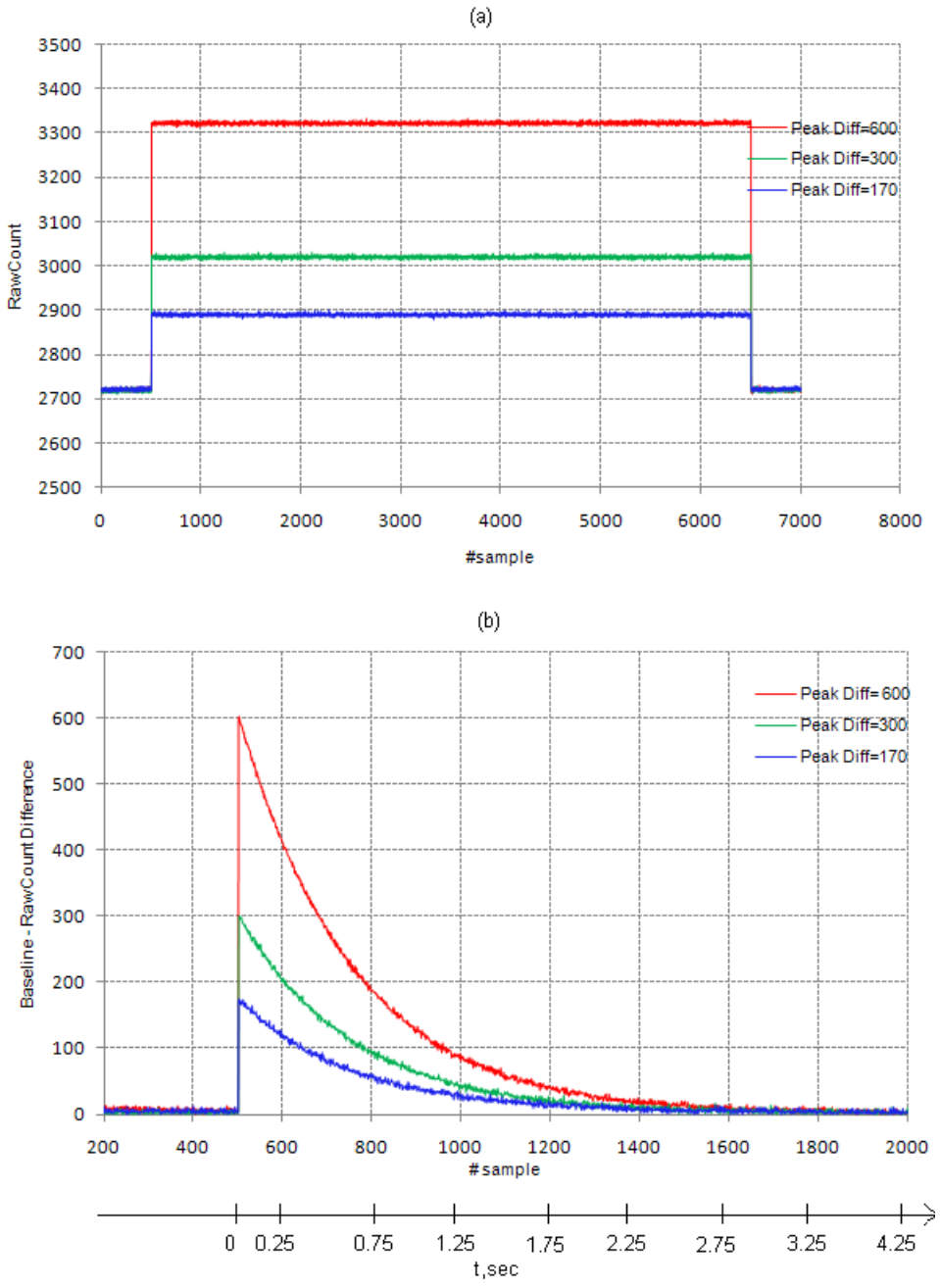
不同速率下，PRS 8，原始计数与温度的关系



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



组件更改

版本	更改说明	更改/影响原因
3.10	在实例名前加上若干变量前缀，使其在每一组件实例中是唯一的。	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 各引脚就不会被启用，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。
	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节。
	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。	允许用户看到和更改传感器的模拟开关分频器。
	允许用户使用不同 IDAC 模式 （源与吸收）和 Vref 值 （1.024V 与 VDAC），从而扩展了自动调节程序的功能。对定制器和固件部件增加了更多计算。	在 IDAC 吸收模式允许自动调节，并允许使用 VDAC 进行基准电压的产生。
	更改了启用 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。
	更改了启用自动重置时的基线更新算法	以前，当自动重置功能被启用时，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。就在它“下楼梯”的同时，它也“下楼梯”。这和 PSoC 1 CapSense 功能不同。
	更改 CapSense_GetTouchCentroidPos() API 函数，避免使用全局数组来存储位置计算。	该函数的上一版本使用了全局数组，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，用户需要计算数组索引。在新版中，触摸位置存储的数组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数组存储，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，2012。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品的内嵌电路之外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非
与赛普拉斯签订明确的书面协议，否则赛普拉斯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中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

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。除上述指定的用途之外，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。赛普拉斯保留在不做出通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用的赛普拉斯软件许可协议限制。