初版

# SmartSense2X_EMC 数据手册 SmartSense2X_EMC V 1.00

Copyright © 2013-2014 Cypress Semiconductor Corporation. All Rights Reserved.

该数据手册包含了初步信息

| 使用资源 | PSoC® 模块 | | | | API 存储器（字节） | | 引脚（除传感器外） |
|---|---|---|---|---|---|---|---|
| | CapSense® | I²C/SPI | 定时器 | 比较器 | 闪存 | RAM | |
| CY8C21x45、 CY8C22x45 | | | | | | | |
| 使用 IDAC 的单通道 | 1 | 0 | – | 1 | 1006 | 29 | 1 |
| 使用 IDAC 的双通道 | 2 | – | – | 2 | 1390 | 30 | 2 |
| 每个附加 CapSense 按键 | – | – | – | – | 8 | 12 | 1 |
| 带有自动调试噪声的特性 | 1 | – | – | – | 2937 | 52 | 2+1 可选 |
| 带有自动调试灵敏度的特性 | 1 | – | – | – | 2954 | 52 | 2+1 可选 |
| 使用带有 5 个元件的电容式滑条时，静态代码和 RAM 均会增加。 | – | – | – | – | 613 | 90 | 5 |
| 每个附加滑条段 | – | – | – | – | 8 | 12 | 1 |
| 带有自动调试噪声的特性 | 2 | – | 2 | – | 3442 | 52 | 2+1 可选 |
| 带有自动调试灵敏度的特性 | 2 | – | – | – | 2954 | 52 | 2+1 可选 |
| 当使用滑条双工法 （5 个传感器）时，静态代码和 RAM 均会增加 | – | – | – | – | 10 | – | – |

## 特性和概述

- 是一种双通道 SmartSense™。
- 可通过手动和自动设置阈值，检测手指触摸。
- 可配置每个传感器的灵敏度。
- 背景扫描。
- 使用按键自动复位和防抖动。
- 使用按键、滑条和接近传感器。
- 通过 SmartSense2X_EMC 向导，可完成传感器和引脚分配。
- 对交流电源噪声、其他 EMI 和供电电压变化具有较强的抗噪能力。

# 简介

SmartSense2X_EMC 用户模块使用了双通道 CapSense® CSD2X 用户模块顶层的 SmartSense 算法。SmartSense2X_EMC 用户模块实现了 CapSense 电容式感应。 CapSense 是一种人机界面技术，该技术通过检测人体的电容执行。它使用一个带有导电界面 （通常是蚀刻在 PCB 表面上的焊盘）的传感器进行检测。由于 CapSense 会检测人体电容，所以它能够检测诸如塑料或玻璃外覆层等绝缘层。这些外覆层通常构成了设备的外型。这些特性使 CapSense 成为按键和电位器等机械输入器件的完美替代品。

SmartSense2X_EMC 用户模块具有背景扫描特性，因此在硬件中正在进行扫描时 CPU 仍能够执行其他操作。使用 LED 渐暗或电机驱动等 CapSense Plus 应用时，该属性非常有用。

## 调制器电容引脚

SmartSense2X_EMC 需要一个外部调制电容 Cmod，该电容从 Vss 连接到专用 PSoC 引脚 P0[5] 或 P0[7]。 SmartSense2X_EMC 双通道配置需要两个外部调制电容，必须将这些电容连接到引脚 P0[5] 和 P0[7] （调制器电容引脚的属性不可用，引脚 P0[5] 和 P0[7] 均被锁定）。

所选的引脚不得用于其他任何用途。外部调制电容的建议值为 2.2 nF。必须使用陶瓷电容。温度电容系数并不重要。赛普拉斯推荐在所有 CapSense 传感器走线上安装大小为 560 Ω 的串联电阻，以抑制 RF 的干扰。将该电阻放置在离 PSoC 器件最近的位置。

## PCB 等级

图 1 和图 2 分别显示的是单通道和双通道的 SmartSense2X_EMC 用户模块的原理图。物理传感器通常是一个导电的图案，被安装在与 PSoC 相连的 PCB 上，并且它上面带有一个绝缘覆盖层。有关 PCB 等级 CapSense 实现的详细信息，请参考设计指南 CapSense 入门。

图 1. SmartSense2X_EMC 单通道原理图

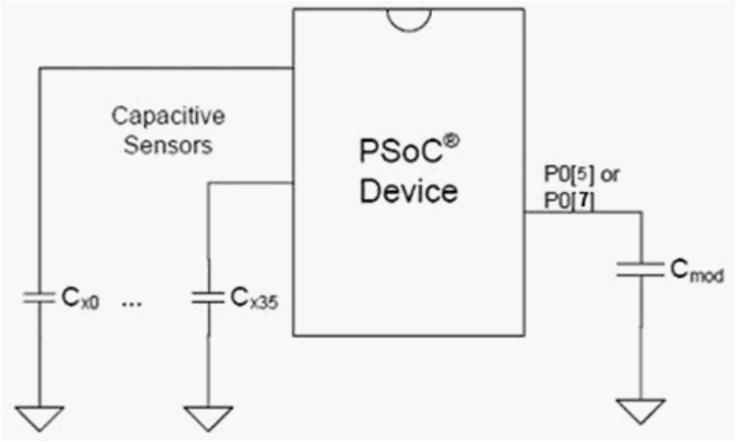
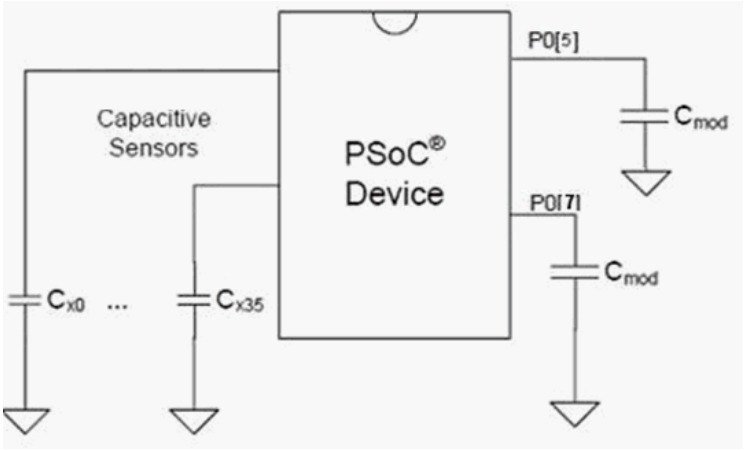


图 2. SmartSense2X 双通道原理图

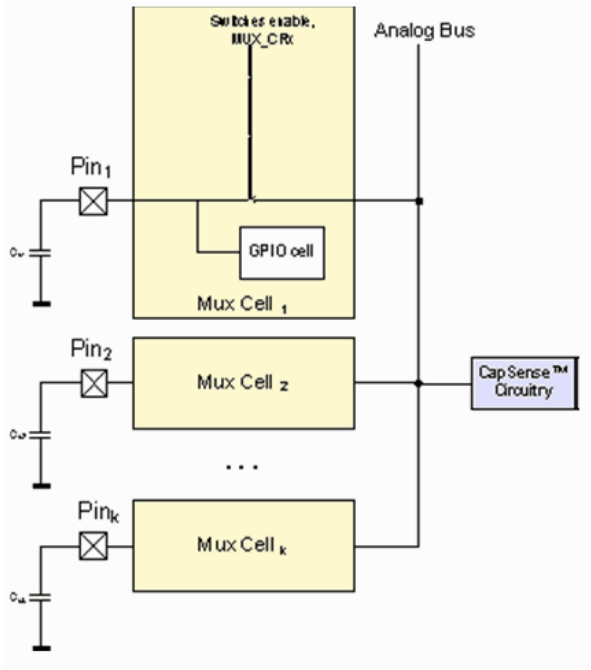



## IC 等级

SmartSense2X_EMC 用户模块通过使用 sigma-delta 调制器以及多个 CSD 参数来使能电容式感应算法（CSD 参数在运行时间或上电复位 （POR）时自动进行动态配置）。除了软件层 （它包括支持背景扫描的 SmartSense 算法以及用于提高抗噪能力的算法）外， SmartSense2X_EMC 用户模块硬件的用途和 CSD2X 硬件的用途几乎完全相同。

通过 CY8C22x45 器件的模拟复用器 （AMUX）总线，可以将电容感应模拟电路连接到任何 PSoC 引脚。SmartSense2X_EMC 用户模块可将活动传感器连接到 AMUX 总线，从而 CapSense 电路可以测量传感器电容，并将该电容值转换为数字代码。通过依次设置 MUX_CRx 寄存器中的相应位，固件可以连续扫描各个传感器。如图 3 所示。

图 3. AMUX 框图



### 软件

SmartSense2X_EMC 软件组件具备下面各特性：

- 在运行时自动调试算法会对模拟电容感应电路进行配置，以获得最佳性能。这些算法结合了物理传感器的特性、 IC 特性和用户模块的传感器灵敏度参数。
- 运行时， API 函数对电容转换电路中的原始计数值进行分析，以确定传感器状态并根据环境变化进行补偿。
- 通过背景扫描算法， CPU 在硬件中正在进行扫描时仍能够执行其他操作。
- 对于连续的附属型传感器 （例如：滑条和触摸板），会提供 API 函数，用于插入一个分辨率高于传感器物理间距的位置。
- 高级软件功能能够适应滑条双工法，因此一个 I/O 引脚能被路由到两个物理传感器。这样可以将针对特定数量的滑条元件消耗的 I/O 数量降低一半。
- 针对外部噪声的抗噪能力改进各种算法。

## 推荐阅读

赛普拉斯推荐在使用 SmartSense2X_EMC 用户模块实现 CapSense 设计前，应该先阅读下面文档。这些文件可在赛普拉斯网站 www.cypress.com 上获取。

- Capsense 入门
- CY8C20xx6A/H CapSense 设计指南
- CY8C21x34/B CapSense 设计指南
- CY8C20x34 CapSense 设计指南
- CY8CMBR2044 CapSense 设计指南

## 芯片级视图

CY8C22x45 系列器件具有内置的模拟总线。可以使电容传感器连接到任意 PSoC 引脚上。SmartSense2X_EMC 用户模块使用内部预充电开关，以在时钟信号相位 Ph1 充电给活动传感器，并在相位 Ph2 将模拟总线连接至传感器。 Sigma-delta 调制电容以及比较器的输入端始终与模拟总线相连接。通过设置 MUX_CRx 寄存器中的相应位，固件可以连续执行传感器扫描。

SmartSense2X_EMC 使用了下面各引脚：

- 传感器引脚
- 外部调制电容 （Cmod）引脚 P0[5] 或 P0[7] （仅适用于单通道配置）
- 屏蔽电极引脚

### 引脚及布线情况

CY8C22x45 器件的双通道配置具有左屏蔽电极输出和右屏蔽电极输出分别使用于左侧和右侧通道。在 SmartSense2X_EMC 向导中，屏蔽电极 （若使能）被路由到左侧通道的 P1[6] 或 P3[6] 引脚上，或右侧通道的 P1[7] 或 P3[7] 引脚上。在具有较少引脚的器件上， P3[6] 和 P3[7] 可能不可用。对于单通道配置，连接取决于使用的是右侧还是左侧 CapSense 通道。应将选定引脚的驱动模式设置为强驱动模式。

## 放置

SmartSense2X_EMC 多用户 （MUM）具有两个选择窗口：

- 在第一个选择窗口中：

  - 使用 IDAC 的单通道配置：该配置使用了一个模拟 / 比较列模块。左侧模拟复用器总线和右侧模拟复用器总线是相互连接的。不需要任何外部 Rb。 IDAC 用于给 Cmod 和 Cp 充电。

  - 使用 IDAC 的双通道配置：该配置使用两个模拟 / 比较列模块。左侧模拟复用器总线和右侧模拟复用器总线分别扫描传感器。不需要任何外部 Rb。

- 在第二个选择窗口中 （选择第一个窗口后会出现）：

  - CapSense0 模块：该配置使用了一个模拟 / 比较列模块 ACE02 和一个 Capsense0 模块。

  - CapSense1 模块：该配置使用了一个模拟 / 比较列模块 ACE03 和一个 Capsense1 模块。

图 4. SmartSense2X_EMC MUM：第一个选择窗口

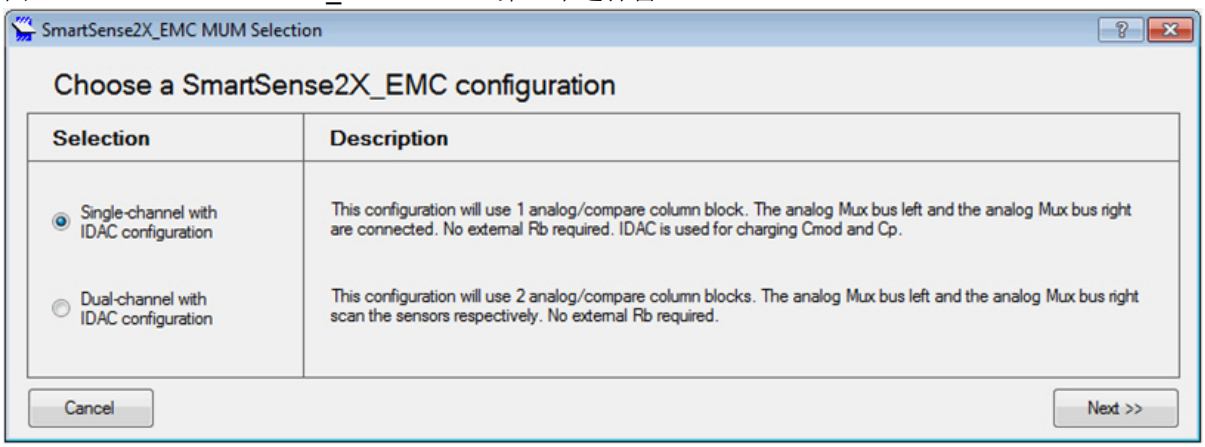
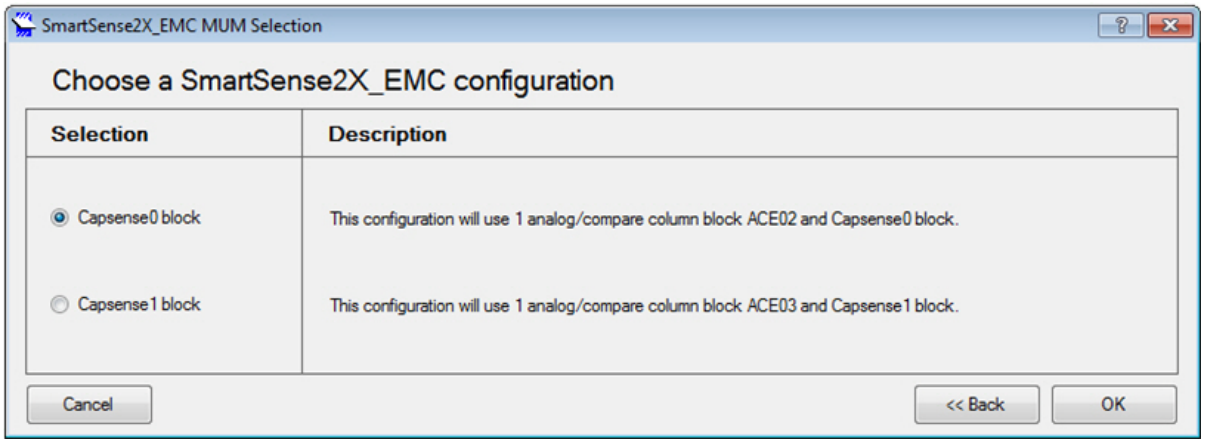



图 5. SmartSense2X_EMC MUM：第二个选择窗口



CY8C22x45 SmartSense2X_EMC MUM 具有两种配置。下表列出的是这些配置的缩写。

| 配置 | 缩略词 |
|---|---|
| Capsense0 上使用 IDAC 的单通道配置 | SmartSense2X_EMC_SGLI22 |
| Capsense1 上使用 IDAC 的单通道配置 | SmartSense2X_EMC_SGRI22 |
| 使用 IDAC 的双通道 | SmartSense2X_EMC_DBLI22 |

## 模块资源

当实例化用户模块时，会自动放置用户模块的子模块，其他放置仅适用于单通道的配置。SmartSense2X_EMC 用户模块使用 CapSense 块和一个比较器块。

必须在启动 SmartSense2X_EMC 向导之前放置需要特定引脚资源 （包括 LCD 和 I2CHW）的用户模块，以建立 SmartSense2X_EMC 用户模块的引脚连接。在放置电容传感器的连接时，请勿使用 P1[0] 和 P1[1]。这些引脚用于对器件进行编程，而且可能存在过大的走线电容值，这样会影响传感器的灵敏度和噪声。下面各图分别显示的是单通道和双通道的模块资源。

图 6. 模块资源 — SmartSense2X_EMC 单通道

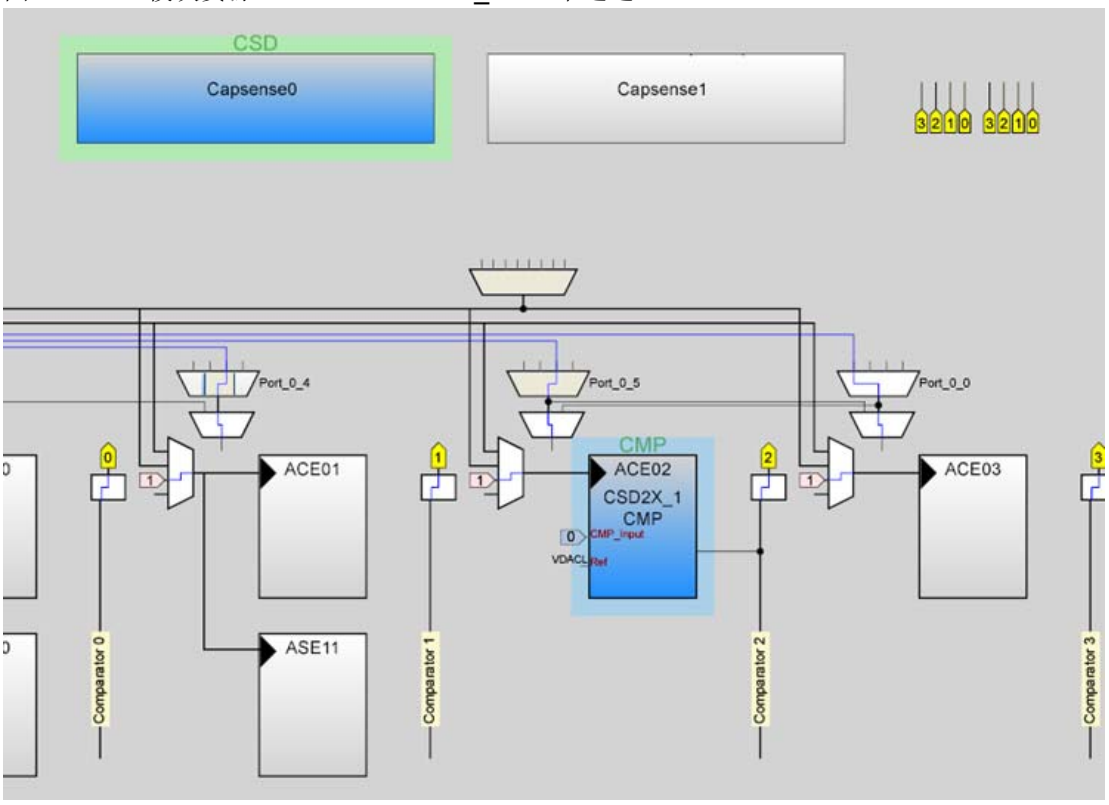
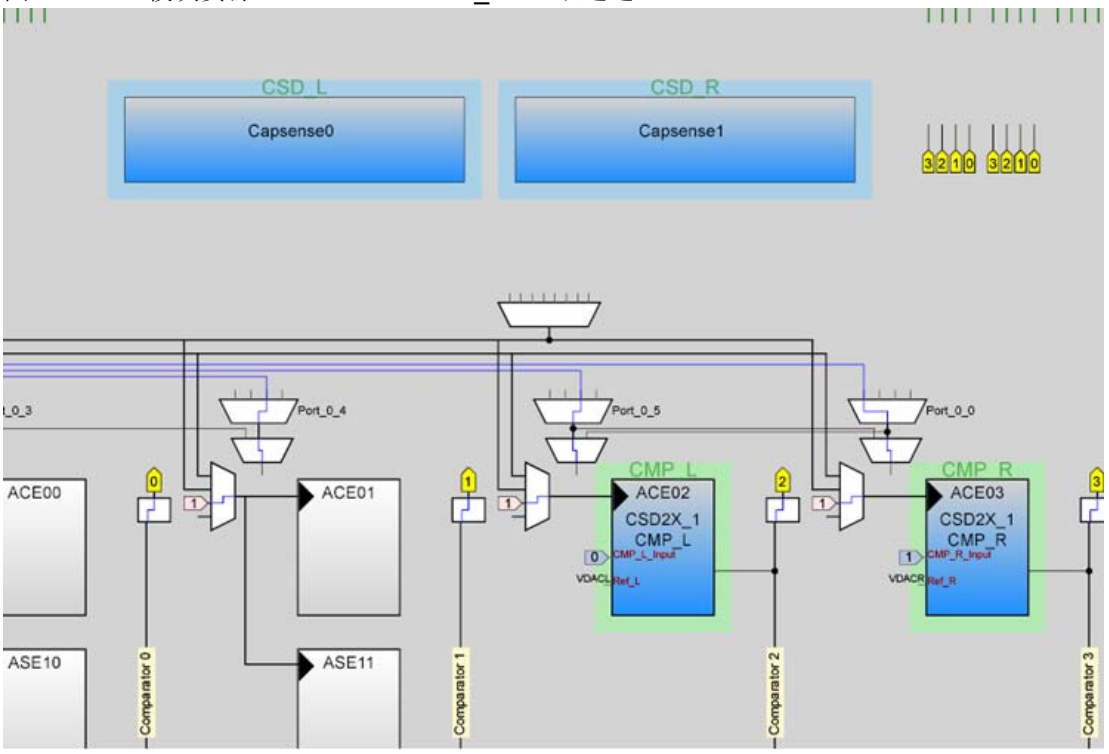



图 7. 模块资源 — SmartSense2X_EMC 双通道



## SmartSense2X_EMC 向导

下图显示的是 SmartSense2X_EMC 用户模块向导的 GUI。

图 8. SmartSense2X_EMC 单通道配置向导

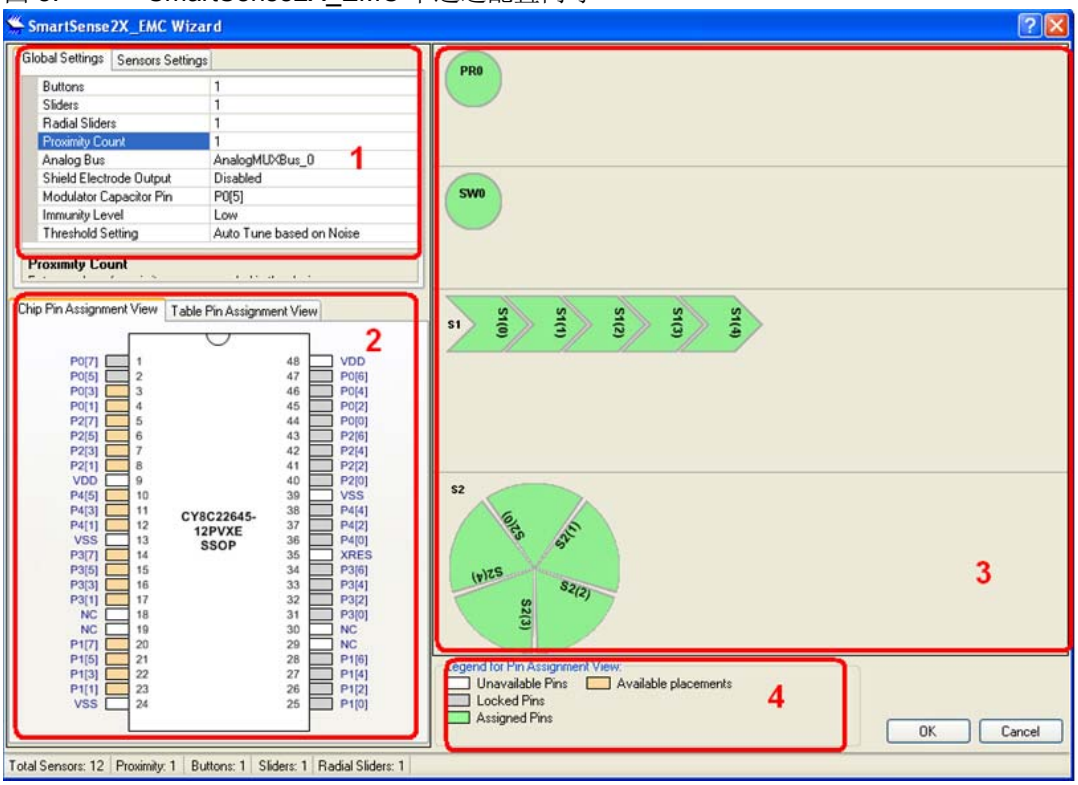
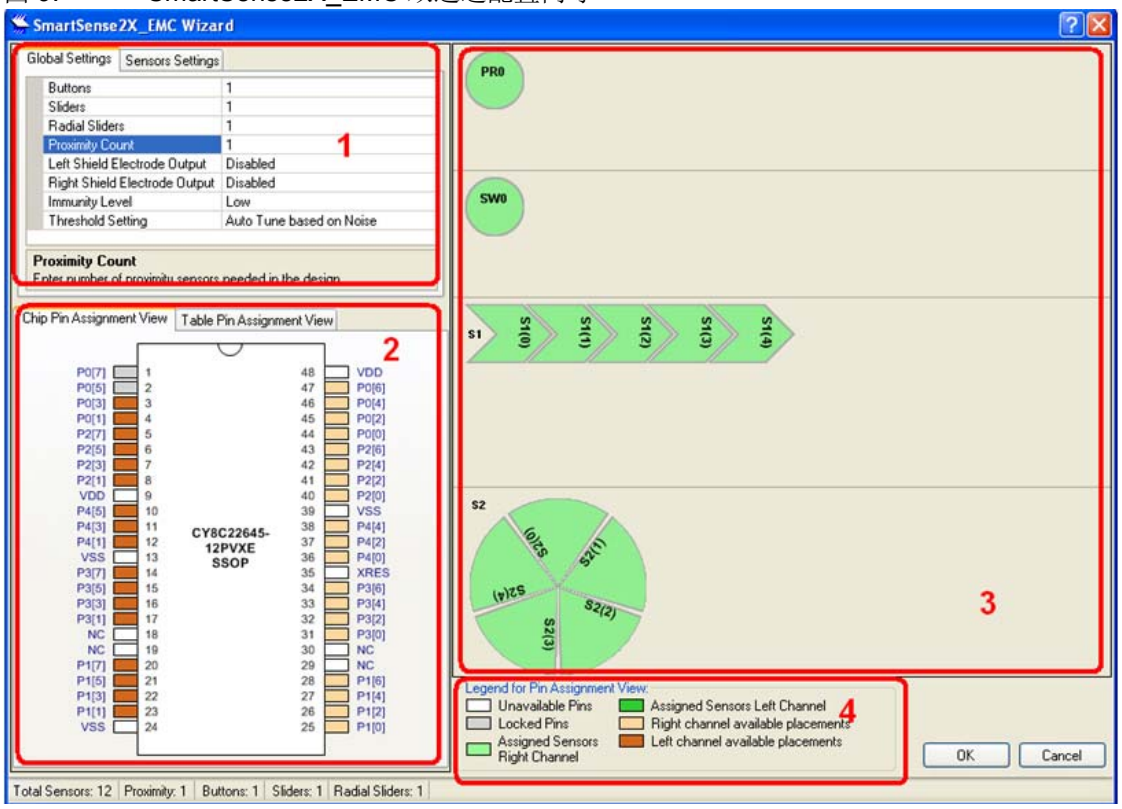



图 9. SmartSense2X_EMC 双通道配置向导



## 向导参数说明

在 Global Settings （全局设置）选项卡和 Sensor Settings （传感器设置）选项卡中介绍了向导参数。

## Global Settings 选项卡

Global Settings 选项卡包括以下各参数：Buttons （按键）、 Sliders （滑条）、 Radial Sliders （辐射滑条）、 Proximity Count （接近感应计数）、 Modulator Capacitor Pin （调制器电容引脚）、 Immunity Level （抗噪级别）、 Threshold Setting （阈值设置）和 Analog Bus （模拟总线）。

图 10. 单通道配置的全局设置

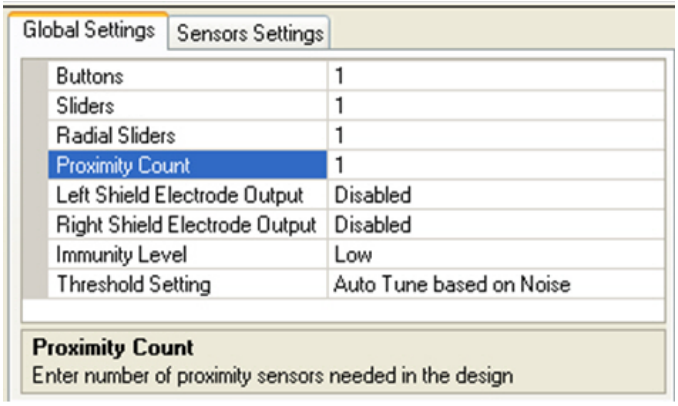
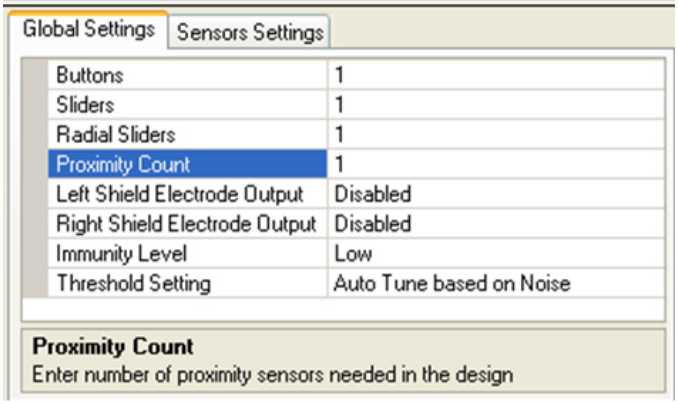



图 11. 双通道配置的全局设置

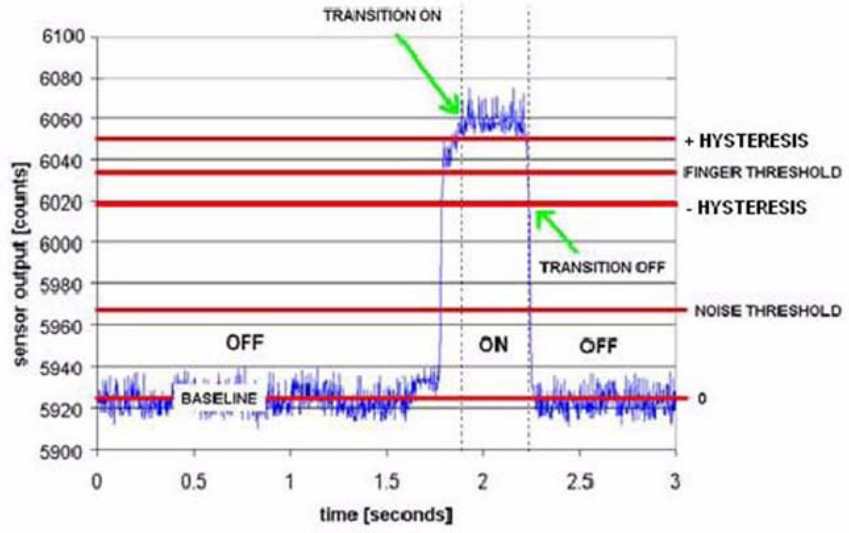



### Buttons （按键）

Buttons 参数定义了单传感器的数量。电容传感器数量的最大值为 37。它受可用的器件端口引脚数的限制。 CapSense 按键类似于机械式按键。这些按键可用于精细的控制，如打开 / 关闭开关、功能键、菜单键，等等。 API 函数监控着来自每个传感器的电容信号，并将这些值和阈值级别进行对比（该阈值是通过 SmartSense2X_EMC 自动调试算法计算得到的）。触摸传感器时，它的电容信号将增加；如果该增量满足 CSD 决策逻辑决定的值，那么将激活该传感器。图 12 显示的是传感器活动

时的一个典型信号 （蓝线）。 SmartSense2X_EMC 根据用户输入的自动设置阈值 （红线），以提供系统所需的操作。此外，可以访问手动模式，从而能够设置每个传感器的手指阈值。

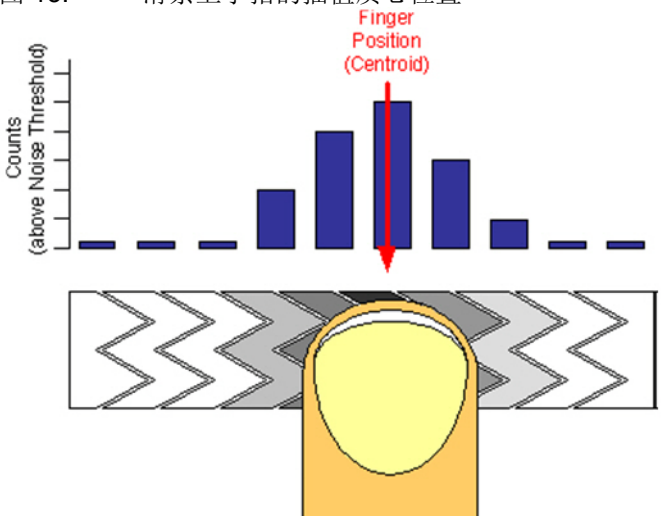
图 12. 传感器活动时的电容信号




### Sliders （滑条）

Sliders 参数定义了线性滑条的数量。滑条的最大数量是通过传感器的最大数量定义的。另外，它受可用的器件端口引脚数的限制。通过使用一组相邻的电容传感器，可以实现 CapSense 滑条。当手指启动滑条时，一些相邻的传感器会寄存电容信号的增加量 （请参考下图）。通过计算活动传感器组的中心位置，可以确定触摸的实际位置。滑条中的传感器最小数量被限制为两个，但实际上最小数量为五个 （默认值），最大数量则受限于 PSoC 器件上可用的 I/O 引脚数量。

图 13. 滑条上手指的插值质心位置

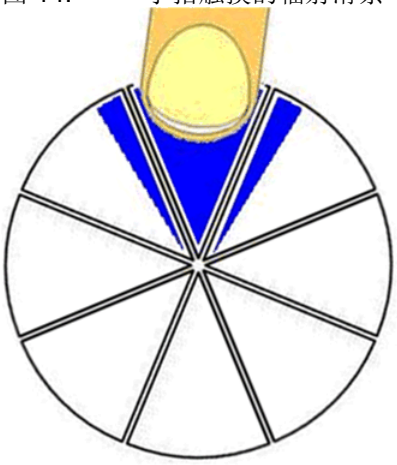



### Radial Sliders （辐射滑条）

Radial Sliders 参数定义了辐射滑条的数量。滑条的最大数量是通过传感器的最大数量定义的。它也受可用的器件端口引脚数的限制。滑条的最大数量等于所选模块上可用的最大引脚数量除以 2 （其中，传感器的最小数量为 2，并且能够将该值设置给一个滑条的情况）。 SmartSense2X_EMC 支持两种滑条类型：线性和辐射。线性滑条有起点和终点，而辐射滑条却没有。在任何一种情况下，如果发生了触摸，中心算法计算各个传感器的信号 （这些传感器与具有最大信号的传感器相邻），以内插

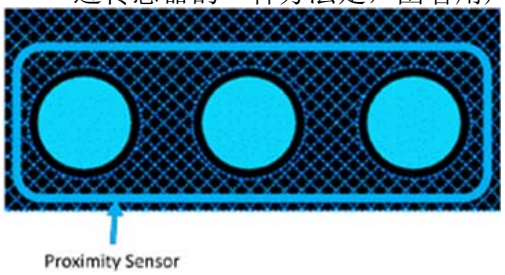
触摸的正确位置。辐射滑条未采用双工法。分辨率受限于 （传感器使用的引脚数量 – 1）x $2^8$ – 1；对于双工滑条，此值为 （2 x 传感器使用的引脚数量 – 1）x $2^8$ – 1。

图 14. 手指触摸的辐射滑条



**Proximity Count （接近感应计数）**

接近传感器在手或其他导体触碰触摸面之前能检测到它的存在。例如，假设有一只手要在黑暗中操控汽车音频系统。当您的手靠近触摸屏时，接近传感器会使音频系统按键通过背光 LED 发光。实现接近传感器的一种方法是，围着用户接口铺上一条长走线。




**Shield Electrode Output （屏蔽电极输出）**

该参数启用或禁用支持可选屏蔽电极的算法。双通道配置分别具有左屏蔽电极输出和右屏蔽电极输出等属性，分别使用于左侧和右侧通道。在 SmartSense2X_EMC 向导中，屏蔽电极 （若使能）被路由到左侧通道的 P1[6] 或 P3[6] 引脚上，或右侧通道的 P1[7] 或 P3[7] 引脚上。在具有较少引脚的器件上， P3[6] 和 P3[7] 可能不可用。对于单通道配置，连接取决于使用的是右侧还是左侧 CapSense 通道。应将选定引脚的驱动模式设置为强驱动模式。

表 1. 单通道配置的设置

| 类型 | 枚举类型 |
|---|---|
| 范围 | 禁用， GOO[6] 到 P1[6]、 GOO[6] 到 P3[6] |
| 默认值 | 禁用 |

表 2. 双通道配置的设置

| 类型 | 枚举类型 |
|---|---|
| 左屏蔽电极输出 | |
| 范围 | 禁用， GOO[6] 到 P1[6]、 GOO[6] 到 P3[6] |
| 默认值 | 禁用 |
| 右屏蔽电极输出 | |
| 范围 | 禁用， GOO[7] 到 P1[7]、 GOO[7] 到 P3[7] |
| 默认值 | 禁用 |

**Modulator Capacitor Pin （调制器电容引脚）**

SmartSense2X_EMC 需要一个外部调制电容 Cmod，该电容从 Vss 连接到专用 PSoC 引脚 P0[5] 或 P0[7]。 SmartSense2X_EMC 双通道配置需要两个外部调制电容，必须将这些电容连接到引脚 P0[5] 和 P0[7] （调制器电容引脚的属性不可用，引脚 P0[5] 和 P0[7] 均被锁定）。

所选的引脚不得用于其他任何用途。外部调制电容的建议值为 2.2 nF。必须使用陶瓷电容。温度电容系数并不重要。赛普拉斯推荐在所有 CapSense 传感器走线上安装大小为 560 Ω 的串联电阻，以抑制 RF 的干扰。将该电阻放置在离 PSoC 器件最近的位置。

**Threshold Setting （阈值设置）**

选择自动阈值设置或手动阈值设置。阈值设置包括下面各选项：

- Manual （手动调试）
  - 将所有阈值的范围手动设置为 1 到 255。在这种情况下，应在 Sensor Settings 选项卡中调整每个传感器的 Finger Threshold 参数。Hysteresis、NoiseThreshold、NegativeNoiseThreshold、BaselineUpdateThreshold 和 LowBaselineReset 的值是固定的，并且通过使用 Finger Threshold 参数可计算得到。

- Auto Tune based on Sensitivity （基于灵敏度的自动调试）
  - 在该选项中，用户模块将手指阈值设置为 SmartSense 初始化过程中所计算的灵敏度。计算灵敏度选择的方法与 CY8C20xx6 SmartSense 相同。

- Auto Tune based on Noise （基于噪声自动调试）
  - 整个自动调试操作可确保最低的信噪比为 5:1 — 用户模块在运行时间内通过成熟的 SmartSense 算法更新手指阈值。只有 Immunity Level 参数被选为 ‘Low’ （低）时，才能选择基于噪声的自动调试选项。默认设置为 ‘Auto Tune based on Noise’ （基于噪声的自动调试）。

**Analog Bus （模拟总线）**

该参数用于确定所使用的一个或两个模拟复用器的用途。如果使用了 AnalogMUXbus_0，则只会扫描连接至奇数引脚 （除引脚 P0[7] 外）的传感器。默认设置为使用 AnalogMUXbus_0 模拟总线。如果使用了 AnalogMUXbus_0，则两个模拟复用器总线未被连接。这样，其他用户模块可以使用 AnalogMUXbus_1 而不会发生冲突。

表 3. 模拟总线参数

| 类型 | API |
|---|---|
| 范围 | AnalogMUXbus_0<br>两者 |
| 默认值 | AnalogMUXbus_0 |

1. 仅适用于单通道配置。

2. 通过该参数，可以使用 SmartSense2X_EMC 用户模块以及其他要求模拟复用器总线的用户模块。

如果同时使用 SmartSense2X_EMC 用户模块和占用同一个模拟复用器总线的用户模块 （如 AMuxN），它们之间可能会发生冲突。如果需要同步操作，那么用户模块要使用其它模拟复用器总线的引脚。

**Immunity Level （抗噪级别）**

该参数用于定义抗噪级别。可选的级别包括：Low （低）、 Medium （中）和 High （高）。抗噪级别越高，可提高在高噪声环境中的性能，但会增加所使用的存储器空间，从而减少受支持的传感器的最大数量。如果抗噪级别被设为 ‘ 低 ’ ，则滑条被禁用。

## Sensor Settings （传感器设置）选项卡

Sensor Settings 选项卡包括以下各参数：Diplex （双工）、 Finger Threshold （手指阈值）、 Resolution （分辨率）、 Sensitivity Level （灵敏度）和 Sensors Count （传感器数量）。

图 15. 按键的传感器设置：阈值被设置为基于噪声的自动调试

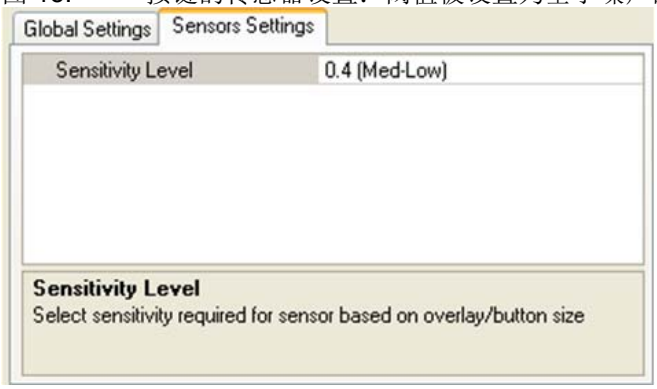



图 16. 按键的传感器设置：阈值设置为基于灵敏度的自动调试

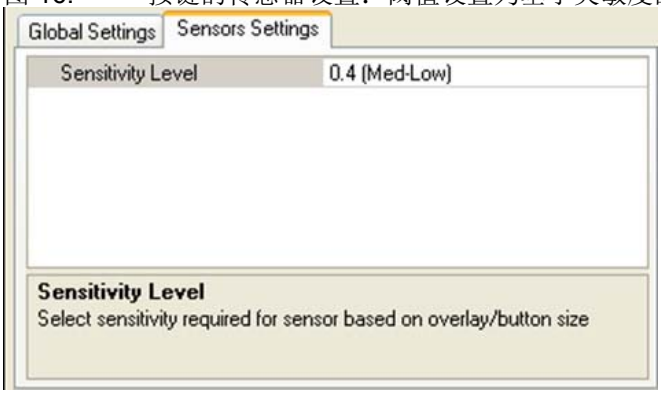



图 17. 按键的传感器设置：阈值设置为手动调试

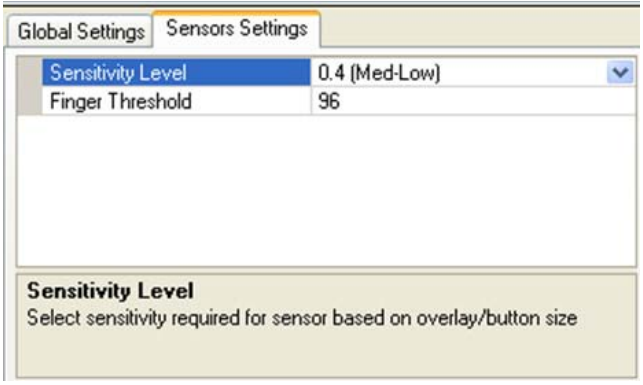


图 18. 接近传感器的传感器设置：阈值设置为基于噪声的自动调试




图 19. 接近传感器的传感器设置：阈值设置为基于灵敏度的自动调试




图 20. 接近传感器的传感器设置：阈值设置为手动调试

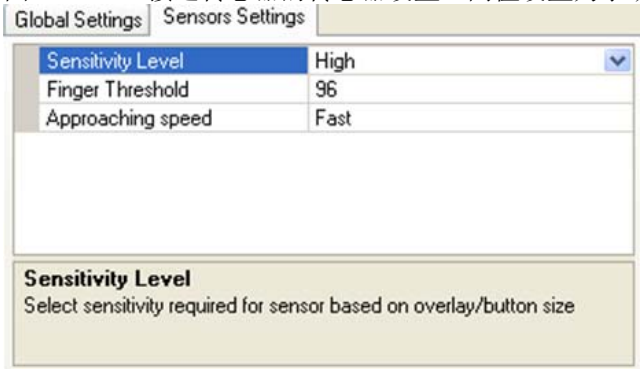



图 21. 线性滑条的传感器设置：阈值设置为基于噪声的自动调试

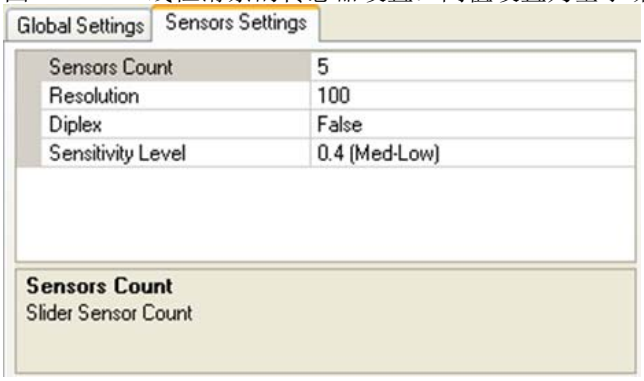


图 22. 线性滑条的传感器设置：阈值设置为基于灵敏度的自动调试

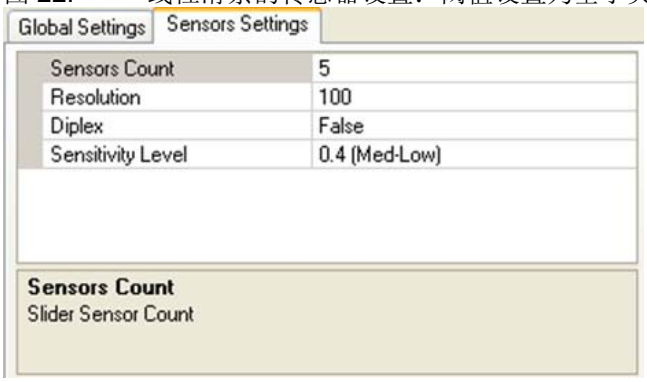



图 23. 线性滑条的传感器设置：阈值设置为手动调试

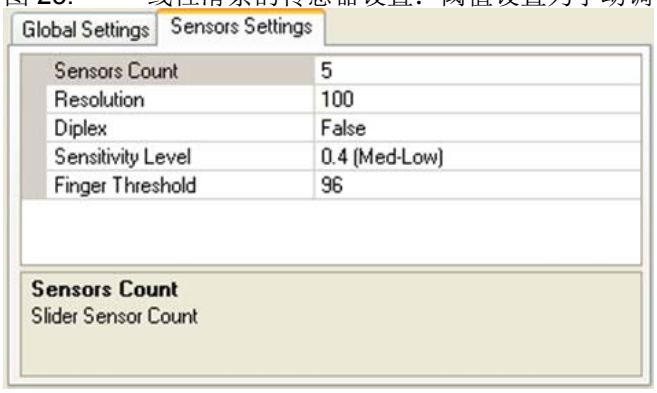



图 24. 辐射滑条的传感器设置：阈值设置为基于噪声的自动调试

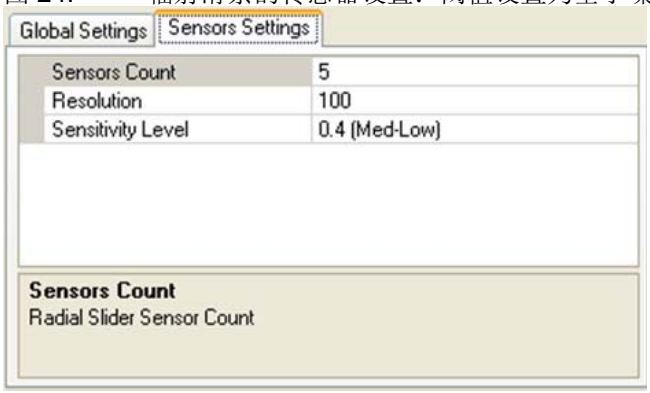



图 25. 辐射滑条的传感器设置：阈值设置为基于灵敏度的自动调试

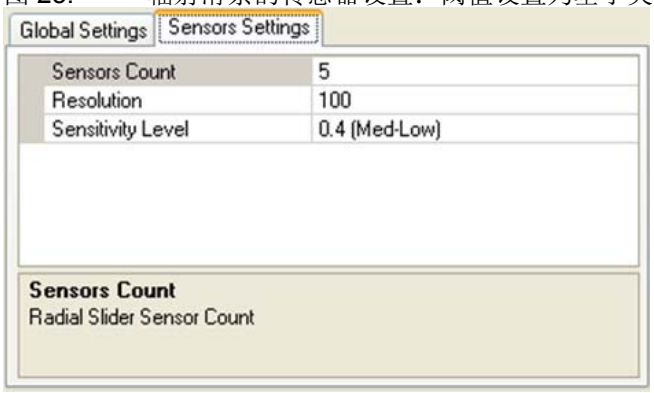
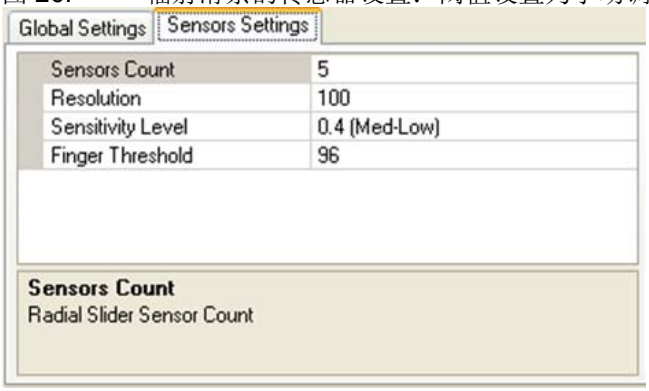


图 26. 辐射滑条的传感器设置：阈值设置为手动调试




**Finger Threshold （手指阈值）**

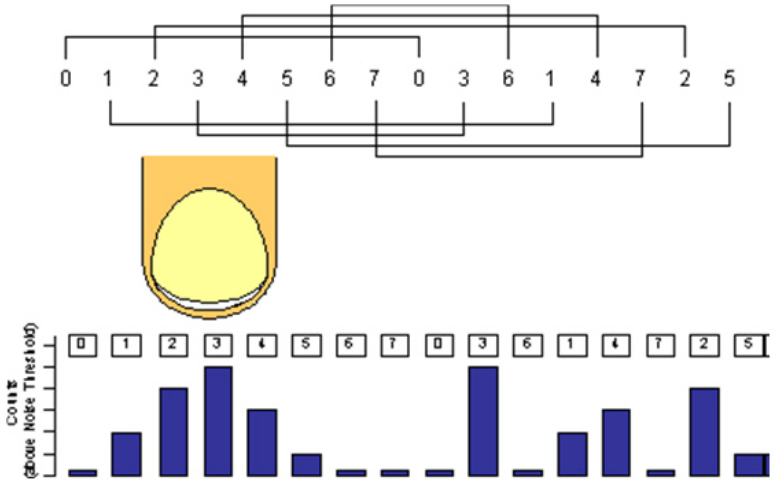
该阈值用于确定每个按键传感器的状态。如果任何一个传感器处于活动状态， bIsAnySensorActive() 函数将返回 1。如果所有传感器均被关闭，则 bIsAnySensorActive() 函数将返回 0。取值范围为 1 到 255。默认值为 80。只有 Threshold Setting 全局参数被选择为 ‘manual’ （手动）模式时，手指阈值才可见。

当 “Threshold Setting” 被设置为 “Auto Tune based on Sensitivity” （基于灵敏度的自动调试）或 “Auto Tune based on Noise” （基于噪声的自动调试）时，手指阈值的属性被禁用。

**Diplex （双工）**

该选项可启用或禁用滑条双工特性。该参数仅适用于线性滑条。使用双工时，作为滑条元素的每个 PSoC 引脚都会映射到滑条传感器阵列中的两个物理位置上。根据 SmartSense2X_EMC 向导分配的端口引脚，物理位置的前半部分 （或为数字中较低的部分）会被映射。物理传感器位置的后 （或上）半部分将通过下图中所示的模型自动映射。

图 27. 由 SmartSense2X_EMC 进行的双工滑条阵列索引编制




越接近滑条下半部分的强信号，将导致上半部分产生相同程度的伪信号。然而，在上半部分中，这些结果被分散和断连。质心算法搜索相邻最强的一组信号，以确定解析的滑条位置。用于映射上半部分传感器的模型可确保：一半部分中的有效信号模型不会导致另一半部分的有效信号模型，如图 27 所示。

必须确保传感器到印制电路板 （PCB）上各引脚的映射情况符合双工算法的 ‘ 按 3 编制索引 ’ 序列。双工滑条中传感器对的电容必须合理匹配 （不能超过 10 pF）。当您选择双工法时， Smart-

Sense2X_EMC 向导会自动生成双工传感器索引表。下表显示的是双工成 28 个 PSoC I/O 引脚的 56 个滑条段的双工序列。

表 4. 不同滑条段数量的双工序列

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

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

**Resolution （分辨率）**

通过该选项，可将传感器分辨率的范围设置为 5 到 （传感器所用的引脚数量 – 1） × $2^8$ – 1 ；对于双工滑条，该值范围为 5 到 （2 × 传感器所用的引脚数量 – 1) × $2^8$ – 1。该参数仅适用于线性滑条和辐射滑条。

**Sensors Count （传感器数量）**

Sensors Count 是指滑条或辐射滑条的物理传感器总数量。该参数仅适用于线性滑条和辐射滑条。默认值为 5，最小值为 2。最大值由器件中可用传感器的最大数量决定。

**Sensitivity Level （灵敏度级别）**

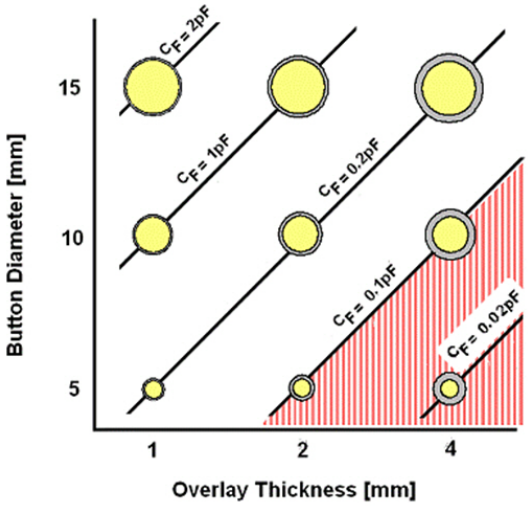
Sensitivity Level 参数用于设置激活按键传感器时所需的电容信号变化 （单位为 pF）。可用设置值为 0.1 pF （高）、 0.2 pF （偏高）、 0.3 pF （中）、 0.4 pF （偏低）和 0.5 pF （低）。默认设置为 0.5 pF。 0.1 pF 对应最高灵敏度， 0.5 pF 对应最低灵敏度。

接近传感器的灵敏度级别只有低和高，默认情况下被设置为 ‘ 高 ’ 。

下图显示的是按键大小、覆盖层厚度 （丙烯酸塑料）和传感器响应速度间的关系，并且该图可作为设置传感器灵敏度的指南使用。Sensor Sensitivity 参数总是被设置为下图所指示的传感器响应速度或

更低。这样可确保能够稳定地运行。请注意，必须避免设置为红色阴影的区域的值，因为传感器响应值低于 SmartSense2X_EMC 可检测到的最小值 0.1。

图 28. 按键大小、覆盖层厚度和传感器响应 （单位为 pF）之间的关系




**Approaching Speed （接近速度）**

该参数决定了接近传感器的基线被更新的速率。它的可用值包括：慢速、中速和快速，默认情况下被设置为快速。将 Threshold Setting 参数设置为 Manual （手动）时，该参数才可用。

## 引脚分配视图

在 “ 引脚分配视图 ” 中，通过将开关或传感器拖放到器件的引脚上，能够将开关或传感器分配给各个引脚。您可以选择在 “ 芯片引脚分配视图 ” 或 “ 表引脚分配视图 ” 中将开关或传感器拖放到引脚上。分配端口引脚后，它会变为绿色，且不再可用。通过将端口引脚拖回未提交的表格，可改变传感器的分配情况。请务必选择已经指定给其他用户模块的引脚。

要想更改引脚的分配，请将光标放在已分配的引脚上，单击该引脚，然后将其拖放至开关框外面。此引脚现在处于未分配状态，您可以重新分配它。

右键单击 “ 芯片引脚分配视图 ” 和 “ 表引脚分配视图 ” 时，会出现 “Clear All Pins” （清除所有引脚）选项。通过该选项，可以撤销所有芯片引脚的分配内容。

## 传感器元素区域

传感器元素区域包括 Buttons View （按键视图）、 Sliders View （滑条视图）和 Radial Sliders View （辐射滑条视图）。

### *按键视图*

该 SmartSense2X_EMC 向导视图显示了 CapSense 按键传感器。在引脚分配视图中通过将传感器拖放到引脚上，可以将这些传感器分配给各个引脚。

### *滑条视图*

SmartSense2X_EMC 向导的该视图显示的是线性滑条。在引脚分配视图中，将滑条传感器拖放到引脚上可实现将每个滑条传感器分配给某个引脚。

*辐射滑条视图*

SmartSense2X_EMC 向导的该视图显示的是辐射滑条。在引脚分配视图中，将滑条传感器拖放到引脚上即可将每个滑条传感器分配给一个引脚。

### 引脚图标区域

图标对引脚的颜色进行了介绍。

白色 — 引脚不能作为 CapSense 输入使用。

灰色 — 引脚被锁定。这种情况有两种可能的原因。第一种可能是另一个用户模块 （如 LCD 或 $I^2C$）已占用了该引脚。另一种可能是引脚名称已更改，不再是默认值。要想恢复使用引脚的默认名称，请在引脚分布视图中展开该引脚，然后从 **Select** 菜单选择 **Default**。现在即可在向导中分配引脚。

橙色 — 引脚可用于分配。

绿色 — 引脚已作为 CapSense 输入进行分配。

### 状态栏

状态栏显示了下面信息：传感器总数、按键数量、滑条数量以及辐射滑条数量。

### “OK” 按键

OK 按键用于存储 SmartSense2X_EMC 配置。根据您所键入的传感器数量、引脚分配、双工和分辨率，会生成一组表格。这些表位于 SmartSense2X_EMC.asm 和 SmartSense2X_EMCHL.asm 中。

*传感器表*

传感器表位于 *SmartSense2X_EMC.asm* 中，每个传感器条目均包含两个字节。第一个字节是端口号，第二个字节是位的掩码 （不是位编号）。下面的示例表格包含六个传感器：

```
SmartSense2X_EMC_Sensor_Table:
_ SmartSense2X_EMC_Sensor_Table
dw 0x0140 // Port 1 Bit 6
dw 0x0301 // Port 3 Bit 0
dw 0x0304 // Port 3 Bit 2
dw 0x0308 // Port 3 Bit 3
dw 0x0302 // Port 3 Bit 1
dw 0x0108 // Port 1 Bit 3
```

*分组表*

分组表定义了每个按键传感器组或滑条组。每个滑条对应一个条目，独立按键传感器再对应一个条目。第一个条目通常是独立的按键传感器。每个条目为六个字节。第一个字节表示传感器表中组开始的索引。第二字节代表该组中传感器的数量。第三个字节表示是否对滑条采用了双工法 （4：已采用， 0：未采用）。第四、五、六个字节是固定点乘数，将其与滑条质心相乘可以得出 SmartSense2X_EMC 向导中指定的分辨率。

```
SmartSense2X_EMC_Group_Table:
_ SmartSense2X_EMC_Group_Table:
; Group Table:
; Origin Count Diplex? DivBtwSw(wholeMSB, wholeLSB, fractByte)
db 0x0, 0x3,    0x00,    0x00, 0x00, 0x00 ; Buttons
db 0x3, 0x8,    0x4,      0x0,   0x0,   0x44 ; Slider 1
```

### *双工表*

双工表扫描顺序数据用于已启用双工法的滑条组。否则，将创建标签而不放置任何数据。该表由两个部分组成：每个滑条的传感器映射，以及每个单独滑条对其表格的引用。下面展示了一个八传感器滑条的典型示例：

```
DiplexTable_0:
; This group is not a diplexed slider
DiplexTable_1:
 db 0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5// 8 switch slider

 SmartSense2X_EMC_Diplex_Table:
_SmartSense2X_EMC_Diplex_Table:
 db >DiplexTable_0, <DiplexTable_0
 db >DiplexTable_1, <DiplexTable_1
```

### *顺序表*

顺序表中的每个传感器占用一个字节。该字节是与通道无关的原始计数结果阵列中的左传感器顺序。该表由向导生成，反映了传感器顺序。

```
SmartSense2X_EMC_Order_Table_Left:
_SmartSense2X_EMC_Order_Table_Left:
DB 0x01 // Position 1
SmartSense2X_EMC_Order_Table_Right:
_SmartSense2X_EMC_Order_Table_Right:
DB 0x00,0x02 // Position 0 and 2
```

### *手指阈值表*

手指阈值表定义了每个传感器的标准手指阈值。下面是一个 1 按键和 5 传感器滑条的典型示例。

```
SmartSense2X_EMC_Finger_Threshold_Table:
_ SmartSense2X_EMC_Finger_Threshold_Table:
db 255, 80, 80, 80, 80, 80
```

### *灵敏度级别表*

灵敏度级别列表定义了每个传感器的灵敏度。下面是一个 1 按键和 5 传感器滑条的典型示例。

```
_Sensitivity_Level_Table:
SmartSense2X_EMC_Sensitivity_Level_Table:
_ SmartSense2X_EMC_Sensitivity_Level_Table:
db, 4, 4, 4, 4, 4, 4
```

表 5. 按键和滑条的灵敏度级别表值

| 参数值 | 表值 |
|---|---|
| 0.1 （高） | 1 |
| 0.2 （偏高） | 2 |
| 0.3 （中） | 3 |
| 0.4 pF （偏低） | 4 |
| 0.5 pF （低） | 5 |

表 6. 接近传感器的灵敏度级别表格值

| 参数值 | 表值 |
|---|---|
| 高 | 1 |
| 低 | 5 |

*接近速度表*

接近速度表格定义了每个接近传感器的接近速度。为了简化 ASM 代码，该表也包含了按键和滑条的可选默认值 （等于 2）。下面是典型的示例：

```
SmartSense2X_EMC_Approaching_Speeds:
_ SmartSense2X_EMC_Approaching_Speeds:
db 2, 2, 1, 0, 2
```

表 7. 按键和接近传感器的接近速度表值

| 参数值 | 表值 | 注意 |
|---|---|---|
| 慢速 | 1 | |
| 中等速度 | 0 | |
| 快速 | 2 | 按键和滑条也有该值。 |

## 参数和资源

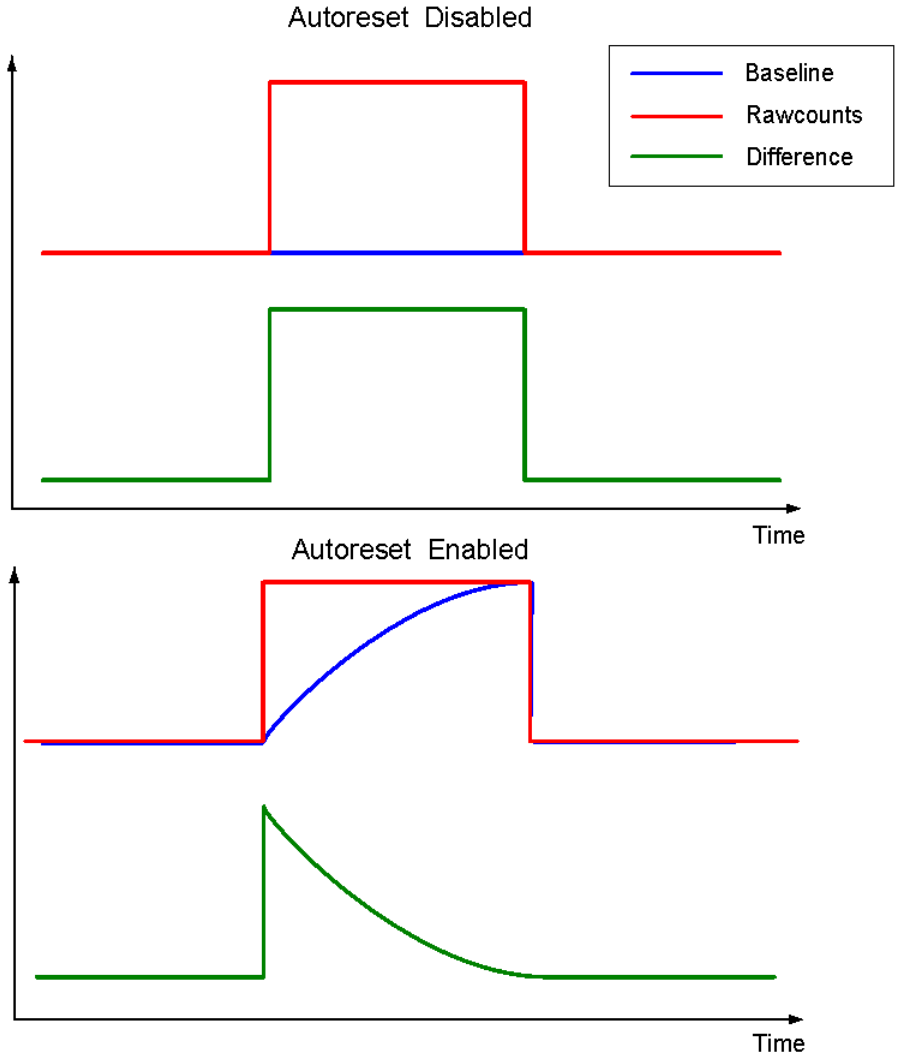
在 SmartSense 向导中完成配置和 I/O 引脚分配后，必须设置模块参数。请注意：更改任何用户模块参数生效时，都必须重新生成项目。

**Sensors Autoreset （传感器自动复位）**

该参数确定基线是随时更新，还是仅当信号差值低于噪声阈值时才更新。如果将该参数默认设置为 **Disabled** （禁用），则仅当原始计数与基线之间的差值低于噪声阈值时，才会更新基线。图 27 说明了该参数对基线更新的影响。当 Sensors Autoreset 设置为 **Enabled** （使能）时，无论噪声阈值如何，均始终更新基线。这样会限制传感器的最大激活时间 （通常为 5 - 10 s），但对防止传感器因非触摸引起的原始计数突然上升而被停滞带来了好处。原始计数突然上升可能是由电源电压剧烈波动、高能射频噪声源或温度快速变化所导致。

当 Sensors Autoreset 设置为 Disabled （禁用），则仅当原始计数与基线之间的差值低于噪声阈值时，基线才进行更新。应该将该参数设置为 Disabled 的默认状态。请参考该参数的其他说明附录部分。

图 29. 传感器自动复位参数对基线更新的影响




| 类型 | API |
|---|---|
| 范围 | 禁用、使能 |
| 默认值 | 禁用 |

**Debounce （防抖动）**

该参数为传感器活动的瞬变增加了去抖动计数器。为了让传感器能够从未激活状态切换为激活状态，在该参数指定的样本数量内，差异计数值必须大于手指阈值与迟滞之和。防抖动计数器由 bIsSensorActive 或 bIsAnySensorActive API 函数递增。

取值范围为 1 到 255。如果该参数设置为 ‘1’，则没有去抖动，但会提供最快的响应。默认设置为 3。

| 类型 | API |
|---|---|
| 范围 | 1-255 |
| 默认值 | 3 |

**Background Scanning （背景扫描）**

该参数用于启用和禁用背景扫描。默认设置为 Disable （禁用）。如果该参数被设置为 Enable （使能），则会实现背景扫描。

| 类型 | API |
|---|---|
| 范围 | 使能 / 禁用 |
| 默认值 | 禁用 |

**FMEA_Shorts_Test**

该参数用于使能 / 禁用 FMEA GND、 VDD 和传感器短接测试。

| 类型 | 枚举类型 |
|---|---|
| 范围 | 使能 / 禁用 |
| 默认值 | 禁用 |

**FMEA_Cmod_Test**

该特性可使能对 Cmod 和 Rb 计算的测试。

| 类型 | 枚举类型 |
|---|---|
| 范围 | 使能 / 禁用 |
| 默认值 | 禁用 |

## 应用编程接口 （API）

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

每次放置用户模块时，都会为其分配一个实例名称。默认情况下， PSoC Designer 向给定项目中该用户模块的第一个实例分配 SmartSense_1。可将该值更改为符合标识符语法规则的任意唯一值。所分配的实例名称作为每个全局函数名称、变量和常量符号的前缀。为简便起见，在以下说明中将实例名称缩写为 SmartSense。

**注意 ****：在这种情况下，同所有用户模块的 API 一样， A 和 X 寄存器的值可以通过调用 API 函数来更改。如果在调用后需要 A 和 X 的值，则调用函数要保留在调用前的 A 和 X 的值。选择这种 “ 寄存器易失 ” 策略是为了提高效率，并且从 PSoC Designer 的 1.0 版本起已强制使用。 C 编译器自动遵循该要求。汇编

语言编程人员也必须保证他们的代码遵守该策略。虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

对于大型储存器模型器件，调用程序需要保留 CUR_PP、 IDX_PP、 MVR_PP 以及 MVW_PP 等寄存器中的所有值。尽管一些寄存器现在未被修改，但是无法保证在将来的版本中也会如此。

提供了进入点以初始化 SmartSense，启动其采样，并停止 SmartSense。在所有情况下，模块的实例名称会替换下列进入点中显示的 SmartSense 前缀。未能使用正确的名称是常见的语法错误原因。

API 函数使用不同的全局阵列。请勿手动更改这些阵列。但是，您可以出于调试目的检查这些值。例如，可以使用绘图工具显示阵列的内容。下面是一些全局阵列：

- SmartSense2X_EMC_waSnsBaseline[]
- SmartSense2X_EMC_waSnsResult[]
- SmartSense2X_EMC_waSnsDiff[]
- SmartSense2X_EMC_baSnsOnMask[]
- SmartSense2X_EMC_baDAC[]
- SmartSense2X_EMC_baCompensationDAC[]
- SmartSense2X_EMC_baDACCodeBaseline[]
- SmartSense2X_EMC_bScanComplete
- SmartSense2X_EMC_baSnSSignal[]

**注意：** <Instance_Name> 是本文档中所有 API 和变量说明的 SmartSense2X_EMC。

**SmartSense2X_EMC_waSnsBaseline[]**：这是一个包含每个传感器的基线数据的整数阵列。阵列大小等于传感器数量。通过下列函数更新 SmartSense_EMC_waSnsBaseline[] 阵列：

- SmartSense2X_EMC_UpdateAllBaselines()
- SmartSense2X_EMC_UpdateSensorBaseline()
- SmartSense2X_EMC_InitializeBaselines()

**SmartSense2X_EMC_waSnsResult[]**：这是一个包含每个传感器的原始信号的整数阵列。阵列大小等于传感器数量。通过下列函数更新 SmartSense2X_EMC_waSnsResult[] 的数据：

- SmartSense2X_EMC_ScanSensor()
- SmartSense2X_EMC_ScanAllSensors().

**SmartSense_EMC_waSnsDiff[]**：这是一个整数阵列，其中包含每个传感器中原始数据与基准数据之间的差值。阵列大小等于传感器的数量。通过下面各函数更新 SmartSense2X_EMC_waSnsDiff [] 数据：

- SmartSense2X_EMC_UpdateAllBaselines()
- SmartSense2X_EMC_UpdateSensorBaseline()

**SmartSense_EMC_baSnsOnMask[]**：这是一个字节阵列，用于保持传感器的开 / 关状态 （针对按键或滑条传感器）。 SmartSense_EMC_baSnsOnMask[0] 包含传感器 0 到 7 的掩码位 （传感器 0 的是位 0，传感器 1 的是位 1）。 SmartSense_EMC_baSnsOnMask[1] 包含传感器 8 到 15 的掩码位 （如果需要），以此类推。此字节阵列包含的元素数足以包含所有被放置的传感器。按键开启时位值为 1，关闭时位值为 0。通过下列函数更新 SmartSense2X_EMC_baSnsOnMask[] 数据：

- SmartSense2X_EMC_bIsSensorActive()
- SmartSense2X_EMC_bIsAnySensorActive()

**SmartSense2X_EMC_baDAC []**：它是一个包含了所有传感器的 IDAC 寄存器值的字节阵列。扫描前，该阵列中的值被复制为 SmartSense2X_EMC_bIdacValue 变量 。

**Smartsense2X_EMC_baCompensationDAC []**：这是一个包含每个传感器的补偿 IDAC 寄存器值的字节阵列。扫描前，将该阵列中的值复制到 SmartSense2X_EMC_bCompensationIdacValue 变量。

**SmartSense2X_EMC_baDACCodeBaseline []**：对于可在运行时更改的每个传感器，该表包含了已校准的电流值。这在复杂项目中很有帮助。

对于双通道配置：

```
SmartSense2X_EMC_baDACCodeBaselineL:
_SmartSense2X_EMC_baDACCodeBaselineL:
BLK SmartSense2X_EMC_TotalLeftSensorCount
SmartSense2X_EMC_baDACCodeBaselineR:
_SmartSense2X_EMC_baDACCodeBaselineR:
BLK SmartSense2X_EMC_TotalRightSensorCount
```

对于单通道配置：

```
SmartSense2X_EMC_baDACCodeBaselineL:
_SmartSense2X_EMC_baDACCodeBaselineL:
BLK SmartSense2X_EMC_TotalSensorCount
SmartSense2X_EMC_baDACCodeBaselineR:
_SmartSense2X_EMC_baDACCodeBaselineR:
BLK SmartSense2X_EMC_TotalSensorCount
```

**SmartSense2X_EMC_bScanComplete []**：仅在使能背景扫描性能时，该变量才有效。一旦传感器扫描结束，都要设置该变量。

**SmartSense2X_EMC_baSnSSignal[]**：这是一个包含每个传感器的标准信号的整数阵列。

## SmartSense2X_EMC_Start

**说明：**

用于初始化寄存器并启动用户模块。在调用任何其他用户模块函数之前，必须先调用此函数。

**C 原型：**

```
void  SmartSense2X_EMC_Start()
```

**汇编：**

```
lcall  SmartSense2X_EMC_Start
```

**参数：**

无

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_Stop

**说明：**

将 SmartSense2X_EMC 模块恢复到其闲置的默认配置，出于其他目的释放 AMUX 总线，禁用内部中断，并调用 SmartSense2X_EMC_ClearSensors() 来使所有传感器重置为它们的未激活状态。

**C 原型：**

```
void  SmartSense2X_EMC_Stop()
```

**汇编：**

```
lcall  SmartSense2X_EMC_Stop
```

**参数：**

无

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_Resume

**说明：**

调用 SmartSense2X_EMC_Stop 之后，恢复用户模块操作。

**C 原型：**

```
void SmartSense2X_EMC_Resume()
```

**汇编：**

```
lcall SmartSense2X_EMC_Resume
```

**参数：**

无

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_ScanSensor(BYTE bSensor)

## SmartSense2X_EMC_ScanSensor(BYTE bLeftSensor, BYTE bRightSensor)

**说明：**

扫描选定的传感器。每个传感器在传感器阵列中只有唯一一个编号。对于单通道配置，该编号由 CSD2X 向导按顺序分配。例如， Sw0 为传感器 0， Sw1 为传感器 1，依此类推。

对于双通道配置，传感器编号为 0 到最大通道传感器编号之间的一个值。例如，如果左通道有两个传感器，则它们的值分别为 0 和 1。如果右通道也有两个传感器，则它们的值也分别为 0 和 1。如果 0xFF 值作为传感器编号传入该函数，则不扫描该通道的任何传感器。

**C 原型：**

在单通道配置中：

```
void SmartSense2X_EMC_ScanSensor(BYTE bSensor);
```

在双通道配置中：

```
void  SmartSense2X_EMC_ScanSensor(BYTE bSensorLeft, BYTE bSensorRight);
```

**汇编：**

在单通道配置中：

```
mov A,  bSensor
lcall SmartSense2X_EMC_ScanSensor
```

在双通道配置中：

```
mov   A,  bSensorLeft
mov   X,  bSensorRight
lcall SmartSense2X_EMC_ScanSensor
```

**参数：**

在单通道配置中：

A => 传感器编号

在双通道配置中：

A => 左通道传感器编号

X => 右通道传感器编号

**返回值：**

无

**其他影响**

**

## SmartSense2X_EMC_ScanAllSensors

**说明：**

该函数通过调用每个传感器的 SmartSense2X_EMC_ScanSensor() 扫描所有已配置的传感器。

**C 原型：**

```
void SmartSense2X_EMC_ScanAllSensors()
```

**汇编：**

```
lcall  SmartSense2X_EMC_ScanAllSensors
```

**参数：**

无

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_ClearSensors

**说明：**

通过为每个传感器依次调用 SmartSense2X_EMC_wGetPortPin() 和 SmartSense2X_EMC_DisableSensor()，将所有的传感器清除为非采样状态。

**C 原型：**

```
void SmartSense2X_EMC_ClearSensors()
```

**汇编：**

```
lcall  SmartSense2X_EMC_ClearSensors
```

**参数：**

无

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_wReadSensor(BYTE bSensor)

**说明：**

返回 A 和 X 中的关键原始扫描值 （分别为 LSB 和 MSB）。

**C 原型：**

```
WORD  SmartSense2X_EMC_wReadSensor(BYTE bSensor);
```

**汇编：**

```
mov  A,  bSensor
lcall SmartSense2X_EMC_wReadSensor
```

**参数：**

A => 传感器编号

**返回值：**

传感器的扫描值， A 中的 LSB 和 X 中的 MSB。

**其他影响：**

**

## SmartSense2X_EMC_wGetPortPin(BYTE bSensor)

**说明：**

返回指定传感器的端口号和引脚掩码。传递的参数对 SmartSense2X_EMC_Sensor_Table[] 中的数据编制索引并进行选择。可将返回值传递给 SmartSense2X_EMC_EnableSensor() 和 SmartSense2X_EMC_DisableSensor()。该函数仅适用于单通道配置。

**C 原型：**

```
WORD  SmartSense2X_EMC_wGetPortPin(BYTE bSensor);
```

**汇编：**

```
mov   A,  bSensor
lcall  SmartSense2X_EMC_wGetPortPin
```

**参数：**

bSensor 是传感器编号，其范围为 0 到 (n – 1)，其中 ‘n’ 是 SmartSense2X_EMC 向导中设置的传感器数量与滑条中包含的传感器数量之和。 SmartSense2X_EMC_wGetPortPin() 使用传感器编号来确定所选的活动传感器的端口和位掩码。

**返回值：**

A => 传感器位图

X => 端口编号

**其他影响：**

**

## SmartSense2X_EMC_wGetPortPinLeft — 双通道

**说明**

返回指定传感器的端口号和引脚掩码。传递的参数会索引并选择 SmartSense2X_EMC_Sensor_Table_Left[] 中的数据。并且能够将返回值传递给 SmartSense2X_EMC_EnableSensor() 和 SmartSense2X_EMC_DisableSensor()。该函数仅适用于双通道配置。

**C 原型：**

```
WORD SmartSense2X_EMC_wGetPortPinLeft(BYTE bSensor)
```

**汇编：**

```
mov A, bSensor
lcall SmartSense2X_EMC_wGetPortPinLeft
```

**参数：**

bSensor — 传感器编号；其范围为 0 到 (n – 1)，其中 ‘n’ 是 SmartSense2X_EMC 向导中为左通道设置的传感器总量（包括连接至左通道的滑条段数量）。 SmartSense2X_EMC_wGetPortPinLeft() 使用传感器编号进行确定所选的活动传感器端口和位俺妈。

**返回值：**

A => 传感器位图； X => 端口编号

**其他影响：**

**

## SmartSense2X_EMC_wGetPortPinRight — 双通道

**说明**

返回连接至右通道的特定传感器的端口编号和引脚掩码。传递的参数会索引并选择 SmartSense2X_EMC_Sensor_Table_Right[] 中的数据。可将返回值传递给 SmartSense2X_EMC_EnableSensor() 和 SmartSense2X_EMC_DisableSensor()。该函数仅适用于双通道配置。

**C 原型：**

```
WORD SmartSense2X_EMC_wGetPortPinRight(BYTE bSensor)
```

**汇编：**

```
mov A, bSensor
lcall SmartSense2X_EMC_wGetPortPinRight
```

**参数：**

bSensor — 传感器编号；其范围为 0 到 (n – 1)，其中 ‘n’ 是 SmartSense2X_EMC 向导为右通道设置的传感器总数量（包括连接至右通道的滑条段数量）。 SmartSense2X_EMC_wGetPortPinRight() 使用传感器编号来确定所选的活动传感器的端口和位俺妈。

**返回值：**

A => 传感器位图； X => 端口编号

**其他影响：**

**

## SmartSense2X_EMC_EnableSensor(BYTE bMask, BYTE bPort)

**说明：**

返回指定传感器的端口号和引脚掩码。传递的参数对 SmartSense2X_EMC_Sensor_Table[] 中的数据编制索引并进行选择。可将返回值传递给 SmartSense2X_EMC_EnableSensor() 和 SmartSense2X_EMC_DisableSensor()。该函数仅适用于单通道配置。

**C 原型：**

```
void  SmartSense2X_EMC_EnableSensor(BYTE bMask, BYTE bPort);
```

**汇编：**

```
mov   X,  bPort
mov   A,  bMask
lcall  SmartSense2X_EMC_EnableSensor
```

**参数：**

A => 传感器位图

X => 端口编号

**返回值：**

无。

**其他影响：**

**

## SmartSense2X_EMC_DisableSensor(BYTE bMask, BYTE bPort)

**说明：**

禁用 SmartSense2X_EMC_wGetPortPin() 函数选定的传感器。驱动模式更改为 “Strong” （强驱动）（即 001）。这样可以将传感器有效接地。端口引脚与 AnalogMuxBus （模拟复用器总线）的连接被断开。 SmartSense2X_EMC_wGetPortPin() 函数返回功能的参数。

**C 原型：**

```
void  SmartSense2X_EMC_DisableSensor(BYTE bMask, BYTE bPort);
```

**汇编：**

```
mov   X,  bPort
mov   A,  bMask
lcall  SmartSense2X_EMC_EnableSensor
```

**参数：**

A => 传感器位图

X => 端口编号

**返回值：**

无。

**其他影响：**

**

## SmartSense2X_EMC_UpdateSensorBaseline(BYTE bSensorNum)

**说明：**

针对每个传感器独立计算得出的历史计数值被称为这个传感器的基线。

此基线使用 “ 水桶算法 ” 进行更新。

“ 水桶算法 ” 使用下列算法：

- 每次调用 SmartSense2X_EMC_UpdateSensorBaseline() 时，通过将原始计数值减去以前的基线值来计算差值计数。该差值存储在 SmartSense2X_EMC_waSnsDiff[] 阵列中。
- 如果禁用了 Sensors Autoreset，则每次调用 SmartSense2X_EMC_UpdateSensorBaseline() 时，将对差值和噪声阈值进行比较。如果差值低于噪声阈值，将被累加到虚拟的桶中。如果差值高于噪声阈值，则不更新该桶。如果使能了 Sensors Autoreset，则无论噪声阈值参数如何，差值都会累加到虚拟桶中。
- 虚拟桶中的累计差值计数达到 BaselineUpdateThreshold 后，基线会按 1 递增，且桶形裕量将复位为 0。
- 如果差值计数低于噪声阈值，则保留在 waSnsDiff[] 阵列中的值将复位为 0。因此，此阵列不包含数值大于 0 但低于噪声阈值的元素。

**C 原型：**

```
void  SmartSense2X_EMC_UpdateSensorBaseline(BYTE bSensorNum);
```

**汇编：**

```
mov   A,  bSensorNum
lcall   SmartSense2X_EMC_UpdateSensorBaseline
```

**参数：**

A => 传感器编号

**返回值：**

无。

**其他影响：**

**

## SmartSense2X_EMC_UpdateAllBaselines

**说明：**

使用 SmartSense2X_EMC_bUpdateSensorBaseline() 函数更新所有传感器的基线。

**C 原型：**

```
void  SmartSense2X_EMC_UpdateAllBaselines();
```

**汇编：**

```
lcall  SmartSense2X_EMC_UpdateAllBaselines
```

**参数：**

无

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_bIsSensorActive(BYTE bSensorNum)

**说明：**

检查给定传感器与手指阈值之间的差值计数阵列。迟滞值也被计算在内。根据传感器当前是否开启，对手指阈值增加或减去迟滞值。如果传感器处于活动状态，则降低该阈值。如果传感器处于非活动状态，则增大该阈值。该函数还可更新 SmartSense2X_EMC_baSnsOnMask[] 阵列中传感器的位。

**C 原型：**

```
BYTE SmartSense2X_EMC_bIsSensorActive(BYTE bSensorNum);
```

**汇编：**

```
mov   A,  bSensorNum
lcall SmartSense2X_EMC_bIsSensorActive
```

**参数：**

bSensorNum A => 传感器编号

**返回值：**

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

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

**其他影响：**

**

## SmartSense2X_EMC_bIsAnySensorActive

**说明：**

检查所有传感器与手指阈值之间的差值计数阵列。针对每个传感器调用 SmartSense2X_EMC_bIsSensorActive()，以便在调用此函数后更新 SmartSense2X_EMC_baSns-OnMask[] 阵列。

**C 原型：**

```
BYTE  SmartSense2X_EMC_bIsAnySensorActive();
```

**汇编：**

```
lcall  SmartSense2X_EMC_bIsAnySensorActive
```

**参数：**

无

**返回值：**

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

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

**其他影响：**

**

## SmartSense2X_EMC_InitializeSensorBaseline(BYTE bSensorNum)

**说明：**

通过扫描选定的传感器，将初始值加载到 SmartSense2X_EMC_waSnsBaseline[bSensor] 阵列元素中。原始计数值将被复制到所选传感器的基线阵列元素中。该函数可用于复位单个传感器的基线值。

**C 原型：**

```
void SmartSense2X_EMC_InitializeSensorBaseline(BYTE bSensorNum);
```

**汇编：**

```
mov A, bSensor
lcall SmartSense2X_EMC_InitializeSensorBaseline
```

**参数：**

A => 传感器编号

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_InitializeBaselines

**说明：**

通过扫描每个传感器，将其初始值加载到 SmartSense2X_EMC_waSnsBaseline[] 阵列中。原始计数值将被复制到每个传感器的基线阵列中。

**C 原型：**

```
void  SmartSense2X_EMC_InitializeBaselines();
```

**汇编：**

```
lcall  SmartSense2X_EMC_InitializeBaselines
```

**参数：**

无。

**返回值：**

无

**其他影响：**

**

## SmartSense2X_EMC_wGetCentroidPos(BYTE bSnsGroup)

**说明：**

检查质心位置的线性滑条。如果存在质心，则将偏移和长度都存储在临时变量中，并根据 SmartSense2X_EMC 向导中指定的分辨率计算质心位置。仅在滑条是由 SmartSense2X_EMC 向导定义时，此函数才可用。

**C 原型：**

```
WORD  SmartSense2X_EMC_wGetCentroidPos(BYTE bSnsGroup);
```

**汇编：**

```
mov    A,  bSnsGroup
lcall    SmartSense2X_EMC_wGetCentroidPos
```

**参数：**

bSnsGroup A => 组编号

该参数表示滑条的组编号。组 0 始终是独立按键传感器。则组 1 和更高的组是滑条传感器组。

**返回值：**

滑条的位置， LSB 位于 A 中、 MSB 位于 X 中。

**其他影响：**

该子程序通过减去噪声阈值更改差值计数的大小，并且仅在每次扫描结束后才会调用该子程序，以避免得到负差值。如果应用监控差值信号，在差值计数数据传输后调用该子程序。

如果有任何滑条传感器处于活动状态，则该函数返回从零到向导中设置的分辨率值之间的值。如果没有任何传感器被激活，则函数返回 -1 （FFFFh）。如果在执行质心 / 双工算法时出现了错误，该函数将返回 -1 （FFFFh）。您可以使用 SmartSense2X_EMC_bIsSensorActive() 函数来确定触摸了哪些滑条段。

## SmartSense2X_EMC_wGetRadialPos(BYTE bSnsGroup)

**说明：**

检查质心位置的辐射滑条。如果存在质心位置，根据 SmartSense2X_EMC 向导中指定的分辨率计算质心位置。

**C 原型：**

```
WORD  SmartSense2X_EMC_wGetRadialPos(BYTE bSnsGroup);
```

**汇编：**

```
mov   A,  bSnsGroup
lcall  SmartSense2X_EMC_wGetRadialPos
```

**参数：**

bSnsGroup A => 组编号

该参数表示辐射滑条的组编号。该编号可以通过辐射滑条表示法左侧上的 SmartSense2X_EMC 向导获取 （例如：‘s2’ 表示辐射滑条的组编号为 2）。

**返回值：**

辐射滑条的位置； LSB 位于 A 中和 MSB 位于 X 中。

**其他影响：**

该子程序通过减去噪声阈值更改了差值计数，仅在每次扫描结束后才调用该子程序，以避免负差值及基线值更新。如果应用监控差值信号，在差值计数数据传输后调用该子程序。

如果有任何滑条传感器处于活动状态，则该函数返回从零到向导中设置的分辨率值之间的值。如果没有任何传感器被激活，则函数返回 -1 （FFFFh）。如果在执行质心算法时出现了错误，该函数将返回 -1 （FFFFh）。您可以使用 SmartSense2X_EMC_bIsSensorActive() 函数来确定触摸了哪些滑条段。

## SmartSense2X_EMC_wGetRadialInc(BYTE bSnsGroup)

**说明：**

返回实际手指移位情况，即手指的当前位置与先前位置之间的差值。该函数与 SmartSense2X_EMC_wGetRadialPos() 同时使用。

**C 原型：**

```
WORD  SmartSense2X_EMC_wGetRadialInc(BYTE bSnsGroup);
```

**汇编：**

```
mov   A,  bSnsGroup
lcall   SmartSense2X_EMC_wGetRadialInc
```

**参数：**

bSnsGroup A => 组编号

该参数表示辐射滑条的组编号。该编号可以通过辐射滑条表示法左侧上的 SmartSense2X_EMC 向导获取 （例如：‘s2’ 表示辐射滑条的组编号为 2）。

**返回值：**

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

手指移位值是手指的当前位置与先前位置之间的差值。

**其他影响：**

仅在调用 SmartSense2X_EMC_wGetRadialPos() 后才能调用该函数，因为后面要使用存储在全局变量中的内部数据。

## BOOL SmartSense2X_EMC_bIsScanComplete

**说明：**

检查 bScanComplete 变量中的扫描完成标志，然后返回 TRUE 或 FASLE 值。该 API 在调用后还可复位该标志。将 “BackgroundScanning” 属性设置为 “Enable” （使能）时，该 API 可用。

**C 原型：**

```
BOOL SmartSense2X_EMC_bIsScanComplete()
```

**汇编**

```
lcall  SmartSense2X_EMC_bIsScanComplete
```

**参数：**

无

**返回值：**

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

A => 1：表示扫描已完成， 0：表示扫描未完成

**其他影响：**

**

## SmartSense_EMC_UpdateSensorSignal(BYTE bSensorNumber)

**说明：**

使用标准差值更新 SmartSense_EMC_baSnSSignal[] 阵列。

**参数：**

A => 传感器编号。

**返回值：**

A => 标准差值。

**其他影响：**

**

## SmartSense2X_EMC_bFMEA_CheckGndShort(void)

**说明**

该 API 用于检查是否有某个传感器短路接地。 SmartSense2X_EMC_bFMEA_CheckVddShort() 函数用于更新 SmartSense2X_EMC_baSensorShortGnd[] 阵列，该函数仅在 FMEA 特性被使能时才有效。 SmartSense2X_EMC_baSensorShortGnd[0] 包含传感器 0 到 7 的掩码位 （传感器 0 的是位 0，传感器 1 的是位 1）。 SmartSense2X_EMC_baSensorShortGnd[1] 包含传感器 8 到 15 的掩码位（如果需要），依次类推。此字节阵列包含的元素数足以包含所有被放置的传感器。

其他的 SmartSense2X_EMC_baSnsShortChk[] 阵列的结构与 SmartSense2X_EMC_baSensor-ShortGnd[] 的结构相同。通过下面任何 API，都可以更新 SmartSense2X_EMC_baSnsShortChk[]：bFMEA_CheckSensorShort、 bFMEA_CheckVddShort、 bFMEA_CheckGndShort、 bFMEA_C-mod_Check （或 bFMEA_Left_Cmod_Check、 bFMEA_Right_Cmod_Check）。下面各表列出了单通道和双通道配置的 SmartSense2X_EMC_baShieldShort 值。

| 常量 （单通道配置） | 数值 | 注释 |
|---|---|---|
| SmartSense2X_EMC_SHIELD_NO_SHORT | 0x00 | 屏蔽电极不短接地面 |
| SmartSense2X_EMC_SHIELD_GND_SHORT | 0x02 | 屏蔽电极短接地面 |

| 常量 （单通道配置） | 数值 | 注释 |
|---|---|---|
| SmartSense2X_EMC_LEFT_SHIELD_NO_SHORT | 0x00 | 左屏蔽电极不短路接地 |
| SmartSense2X_EMC_LEFT_SHIELD_GND_SHORT | 0x02 | 左屏蔽电极被短路接地 |
| SmartSense2X_EMC_RIGHT_SHIELD_NO_SHORT | 0x00 | 右屏蔽电极不短路接地 |
| SmartSense2X_EMC_RIGHT_SHIELD_GND_SHORT | 0x20 | 右屏蔽电极被短路接地 |

**C 原型：**

```
BYTE SmartSense2X_EMC_bFMEA_CheckGndShort(void);
```

**汇编**

```
lcall  SmartSense2X_EMC_bFMEA_CheckGndShort
```

**参数**

无

**返回值**

如果某个传感器短路接地，返回值为 1 ；

否则，返回值为 0。

**其他影响：**

**

## SmartSense2X_EMC_bFMEA_CheckVddShort(void)

**说明**

该 API 用于检查是否有某个传感器短接 Vdd。

SmartSense2X_EMC_bFMEA_CheckVddShort() 函数用于更新 SmartSense2X_EMC_baSensorShortVdd[] 阵列，该函数仅在 FMEA 特性使能时才有效。 SmartSense2X_EMC_baSensorShortVdd[0] 包含传感器 0 到 7 的掩码位 （传感器 0 的是位 0，传感器 1 的是位 1）。 SmartSense2X_EMC_baSensorShortVdd[1] 包含传感器 8 到 15 的掩码位 （如果需要），依次类推该字节阵列包含的元素数足以包含所有被放置的传感器。其他的 SmartSense2X_EMC_baSnsShortChk[] 阵列的结构与 SmartSense2X_EMC_baSensorShortGnd[] 的结构相同。通过下面任意 API，都可以更新 SmartSense2X_EMC_baSnsShortChk[]：bFMEA_CheckSensorShort、bFMEA_CheckVddShort、 bFMEA_CheckGndShort 和 bFMEA_Cmod_Check （或

bFMEA_Left_Cmod_Check、 bFMEA_Right_Cmod_Check）。下面各表列出了单通道和双通道配置的 SmartSense2X_EMC_baShieldShort 值。

| 常量 （单通道配置） | 数值 | 注释 |
|---|---|---|
| SmartSense2X_EMC_SHIELD_NO_SHORT | 0x00 | 屏蔽电极不短接 VDD |
| SmartSense2X_EMC_SHIELD_VDD_SHORT | 0x01 | 屏蔽电极短接 VDD |

| 常量 （单通道配置） | 数值 | 注释 |
|---|---|---|
| SmartSense2X_EMC_LEFT_SHIELD_NO_SHORT | 0x00 | 左屏蔽电极不短接 VDD |
| SmartSense2X_EMC_LEFT_SHIELD_VDD_SHORT | 0x01 | 左屏蔽电极短接 VDD |
| SmartSense2X_EMC_RIGHT_SHIELD_NO_SHORT | 0x00 | 右屏蔽电极不短接 VDD |
| SmartSense2X_EMC_RIGHT_SHIELD_VDD_SHORT | 0x10 | 右屏蔽电极短接 VDD |

**C 原型：**

```
BYTE SmartSense2X_EMC_bFMEA_CheckVddShort(void)
```

**汇编**

```
lcall  SmartSense2X_EMC_bFMEA_CheckVddShort
```

**参数**

无

**返回值**

如果某个传感器短接 VDD，那么返回值为 1 ；否则，返回值为 0。

## SmartSense2X_EMC_bFMEA_CheckSensorShort(void)

**说明**

该 API 用于检查是否存在两个传感器被短接。其他的 SmartSense2X_EMC_baSnsShortChk[] 阵列结构与 SmartSense2X_EMC_baSensorShortGnd[] 的结构相同。通过下面任意 API，都可以更新 SmartSense2X_EMC_baSnsShortChk[]：bFMEA_CheckSensorShort、 bFMEA_CheckVddShort、bFMEA_CheckGndShort 和 bFMEA_Cmod_Check （或 bFMEA_Left_Cmod_Check、bFMEA_Right_Cmod_Check）。 SmartSense2X_EMC_baShieldShort 的可选变量包含屏蔽电极的状态。下面各表列出了单通道和双通道配置的 SmartSense2X_EMC_baShieldShort 值。

| 常量 （单通道配置） | 数值 | 注释 |
|---|---|---|
| SmartSense2X_EMC_SHIELD_NO_SHORT | 0x00 | 屏蔽电极不短接至任何传感器 |
| SmartSense2X_EMC_SHIELD_SENS_SHORT | 0x04 | 屏蔽电极短接至一个传感器 |

| 常量 （单通道配置） | 数值 | 注释 |
|---|---|---|
| SmartSense2X_EMC_LEFT_SHIELD_NO_SHORT | 0x00 | 左屏蔽电极不短接至任何传感器 |
| SmartSense2X_EMC_LEFT_SHIELD_SENS_SHORT | 0x04 | 左屏蔽电极短接至一个传感器 |
| SmartSense2X_EMC_RIGHT_SHIELD_NO_SHORT | 0x00 | 右屏蔽电极不短接至任何传感器 |
| SmartSense2X_EMC_RIGHT_SHIELD_SENS_SHORT | 0x40 | 右屏蔽电极短接至一个传感器 |
| SmartSense2X_EMC_SHIELD_SHIELD_SHORT | 0x80 | 右屏蔽电极短接至左屏蔽电极 |

**C 原型：**

```
BYTE SmartSense2X_EMC_bFMEA_CheckSensorShort(void)
```

**汇编**

```
lcall  SmartSense2X_EMC_bFMEA_CheckSensorShort
```

**参数**

无

**返回值**

如果某个传感器短接至另一个，返回值为 1 ；否则，返回值为 0。

### SmartSense2X_EMC_bFMEA_Cmod_Check(void) — 单通道

### SmartSense2X_EMC_bFMEA_Left_Cmod_Check(void) — 双通道

### SmartSense2X_EMC_bFMEA_Right_Cmod_Check(void) — 双通道

**说明**

仅在选择 IDAC 方法并使能 FMEA 特性时，该 API 才可用。该 API 可检查用于短接测试的 Cmod 值，另外还会检查该值是否处于有效范围内。它的有效范围被定义为所推荐的最小和最大的 Cmod 值，其误差为 20%。 Cmod 的最小值为 3.7 nF，最大值为 56.4 nF。 SmartSense2X_EMC_bFMEA_Cmod_Check() 函数更新了 SmartSense2X_EMC_wCmod_Val （双通道配置中分别为 SmartSense2X_EMC_wCmod_L_Val 和 SmartSense2X_EMC_wCmod_R_Val） WORD 变量。该变量包含的 Cmod 值是按 100 比例增大的，其单位为 nF。仅在 SmartSense2X_EMC_bFMEA_Cmod_Check() API 返回 bRetVal.4 时，该值才有效。其他的 SmartSense2X_EMC_baSnsShortChk[] 阵列

结构与 SmartSense2X_EMC_baSensorShortGnd[] 的结构相同。通过下面任意 API，都可以更新 SmartSense2X_EMC_baSnsShortChk[]：bFMEA_CheckSensorShort、 bFMEA_CheckVddShort、bFMEA_CheckGndShort 和 bFMEA_Cmod_Check （或 bFMEA_Left_Cmod_Check、bFMEA_Right_Cmod_Check）。

**C 原型：**

单通道：

```
BYTE SmartSense2X_EMC_bFMEA_Cmod_Check(void);
```

双通道：

```
BYTE SmartSense2X_EMC_bFMEA_Left_Cmod_Check(void);
BYTE SmartSense2X_EMC_bFMEA_Right_Cmod_Check(void);
```

**汇编**

```
lcall  SmartSense2X_EMC_bFMEA_Cmod_Check
```

**参数**

无

**返回值**

bRetVal.0：Cmod 与 GND 短接

bRetVal.1：Cmod 与 VDD 短接

bRetVal.4：Cmod 的误差不能超过 +20%

bRetVal.8：Cmod 低于有效范围或 Cmod 被断开连接。

bRetVal.16：Cmod 高于有效范围

单通道：

| 常量 | 数值 | 说明 |
|---|---|---|
| SMARTSENSE2X_EMC_CMOD_SHORTED_TO_GND | 0 | Cmod 与 GND 短接 |
| SMARTSENSE2X_EMC_CMOD_SHORTED_TO_VDD | 1 | Cmod 与 Vdd 短接 |
| SMARTSENSE2X_EMC_CMOD_WITHIN_20 | 4 | Cmod 的误差不能超过 ±20% |
| SMARTSENSE2X_EMC_CMOD_IS_LOW | 8 | Cmod 超出范围 |
| SMARTSENSE2X_EMC_CMOD_IS_HIGH | 16 | Cmod 超出范围 |

双通道：

| 常量 | 数值 | 说明 |
|---|---|---|
| SMARTSENSE2X_EMC_LEFT_CMOD_SHORTED_TO_GND | 0 | 左通道的 Cmod 与 GND 短接 |
| SMARTSENSE2X_EMC_LEFT_CMOD_SHORTED_TO_VDD | 1 | 左通道的 Cmod 与 Vdd 短接 |
| SMARTSENSE2X_EMC_LEFT_CMOD_WITHIN_20 | 4 | 左通道 Cmod 的误差不能超过 ±20% |
| SMARTSENSE2X_EMC_LEFT_CMOD_IS_LOW | 8 | 左通道的 Cmod 超出范围 |
| SMARTSENSE2X_EMC_LEFT_CMOD_IS_HIGH | 16 | 左通道的 Cmod 超出范围 |
| SMARTSENSE2X_EMC_RIGHT_CMOD_SHORTED_TO_GND | 0 | 右通道的 Cmod 与 GND 短接 |
| SMARTSENSE2X_EMC_RIGHT_CMOD_SHORTED_TO_VDD | 1 | 右通道的 Cmod 与 Vdd 短接 |
| SMARTSENSE2X_EMC_RIGHT_CMOD_WITHIN_20 | 4 | 右通道 Cmod 的误差不能超过 ±20% |
| SMARTSENSE2X_EMC_RIGHT_CMOD_IS_LOW | 8 | 右通道的 Cmod 超出范围 |
| SMARTSENSE2X_EMC_RIGHT_CMOD_IS_HIGH | 16 | 右通道的 Cmod 超出范围 |

## 直流和交流电气特性

表 8. 直流和交流电气特性

| 参数 | 说明 | 条件 | 最小值 | 典型值 | 最大值 | 单位 |
|---|---|---|---|---|---|---|
| Cp （最大值） | 受支持的寄生电容最大值 | 温度范围：–40 ~ +80 °C | 45 | – | – | pF |
| Cp （最小值） | 受支持的寄生电容最小值 | 温度范围：–40 ~ +80 °C | – | – | 5 | pF |

## 固件源代码示例

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

```
//-------------------------------------------------------------------------
// Sample C code for the SmartSense2X_EMC module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
```

```
    M8C_EnableGInt;
    SmartSense2X_EMC_Start();
    SmartSense2X_EMC_InitializeBaselines() ; //scan all sensors first time, init
    //baseline
    // Loop Forever

    while (1)
    {
        SmartSense2X_EMC_ScanAllSensors(); //scan all sensors in array (buttons and
        //sliders)
        SmartSense2X_EMC_UpdateAllBaselines(); //Update all baseline levels;
        //detect if any sensor is pressed
        if(SmartSense2X_EMC_bIsAnySensorActive())
        {
        // Add user code here to proceed with sensor touching
        }
        // now we are ready to send all status variables to chart program
        // communication here
        // OUTPUT SmartSense2X_EMC_waSnsResult[x] <- Raw Counts
        // OUTPUT SmartSense2X_EMC_waSnsDiff[x] <- Difference
        // OUTPUT SmartSense2X_EMC_waSnsBaseline[x] <- Baseline
        // OUTPUT SmartSense2X_EMC_baSnsOnMask[x] <- Sensor On/Off
    }
}
```

**示例 2.** 该代码用于启动用户模块并连续扫描传感器；但和示例 1 不同，通过传感器的环路是在用户代码中完成的。可以使用通信部分将值传递给 PC 绘图工具。

```
//-------------------------------------------------------------------------
// Sample C code for the SmartSense2X_EMC module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

void main(void)
{
    BYTE bIndex;
    M8C_EnableGInt;
    SmartSense2X_EMC_Start();
    SmartSense2X_EMC_InitializeBaselines() ; //Scan all sensors first time, init
    //baseline
    while (1) //Loop forever
    {
        for(bIndex=0; bIndex < SmartSense2X_EMC_TotalSensorCount; bIndex++)
        //Loop through all sensors
        {
            SmartSense2X_EMC_ScanSensor(bIndex);           // Scan sensors
            SmartSense2X_EMC_UpdateSensorBaseline(bIndex); // Run baseline filter
            if(SmartSense2X_EMC_bIsSensorActive(bIndex))
            {
            // Add user code here to process the sensor touching
            }
```

```
        }
        // detect if any sensor is pressed
        // now we are ready to send all status variables to chart program
        // communication here
        //
        // OUTPUT SmartSense2X_EMC_waSnsResult[x] <- Raw Counts
        // OUTPUT SmartSense2X_EMC_waSnsDiff[x] <- Difference
        // OUTPUT SmartSense2X_EMC_waSnsBaseline[x] <- Baseline
        // OUTPUT SmartSense2X_EMC_baSnsOnMask[x] <- Sensor On/Off
    }
}
```

**示例 3.** 启用背景扫描特性时，此代码用于启动用户模块，并连续扫描传感器。可以使用通信部分将值传递给 PC 绘图工具。

```
//-------------------------------------------------------------------------
// Sample C code for the SmartSense2X_EMC module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules
void main(void)
{
    M8C_EnableGInt;
    SmartSense2X_EMC_Start();
    SmartSense2X_EMC_InitializeBaselines() ; //scan all sensors first time, init
    //baseline
    //
    // Loop Forever
    //
    while (1)
    {
        SmartSense2X_EMC_ScanAllSensors(); //scan all sensors in array (buttons and
        //sliders)
        while(!SmartSense2X_EMC_bIsScanComplete()); // wait until scanned

            SmartSense2X_EMC_UpdateAllBaselines(); //Update all baseline levels;
            // now we are ready to send all status variables to chart program
            // communication here
            //
            // OUTPUT SmartSense2X_EMC_waSnsResult[x] <- Raw Counts
            // OUTPUT SmartSense2X_EMC_waSnsDiff[x] <- Difference
            // OUTPUT SmartSense2X_EMC_waSnsBaseline[x] <- Baseline
            // OUTPUT SmartSense2X_EMC_baSnsOnMask[x] <- Sensor On/Off

        //detect if any sensor is pressed
        if(SmartSense2X_EMC_bIsAnySensorActive())
        {
            // Add user code here to proceed the sensor touching
        }
        // Do other tasks
    }
}
```

# 版本历史记录

| 版本 | 创作者 | 说明 |
|---|---|---|
| 1.00 | HPHA | 初始版本。 |

**注意：** PSoC Designer 版本 5.1 在所有用户模块数据手册中都介绍了 “ 版本历史记录 ”。本数据手册介绍了当前和先前用户模块版本之间的区别的概述。



Copyright © 2013-2014 赛普拉斯半导体公司。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不会根据专利权或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC Designer™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标， PSoC® 是赛普拉斯半导体公司的注册商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码 （软件和 / 或固件）均归赛普拉斯半导体公司 （赛普拉斯）所有，并受全球专利法规 （美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和 / 或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定用途外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对该材料提供任何类型的明示或暗示保证，包括 （但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不另行通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于合理预计可能发生运转异常和故障，并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而导致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用于赛普拉斯软件许可协议的限制。