CapSense® 逐次逼近多频数据手册 CSAMFS V 1.20

Copyright © 2009-2010 Cypress Semiconductor Corporation. All Rights Reserved.

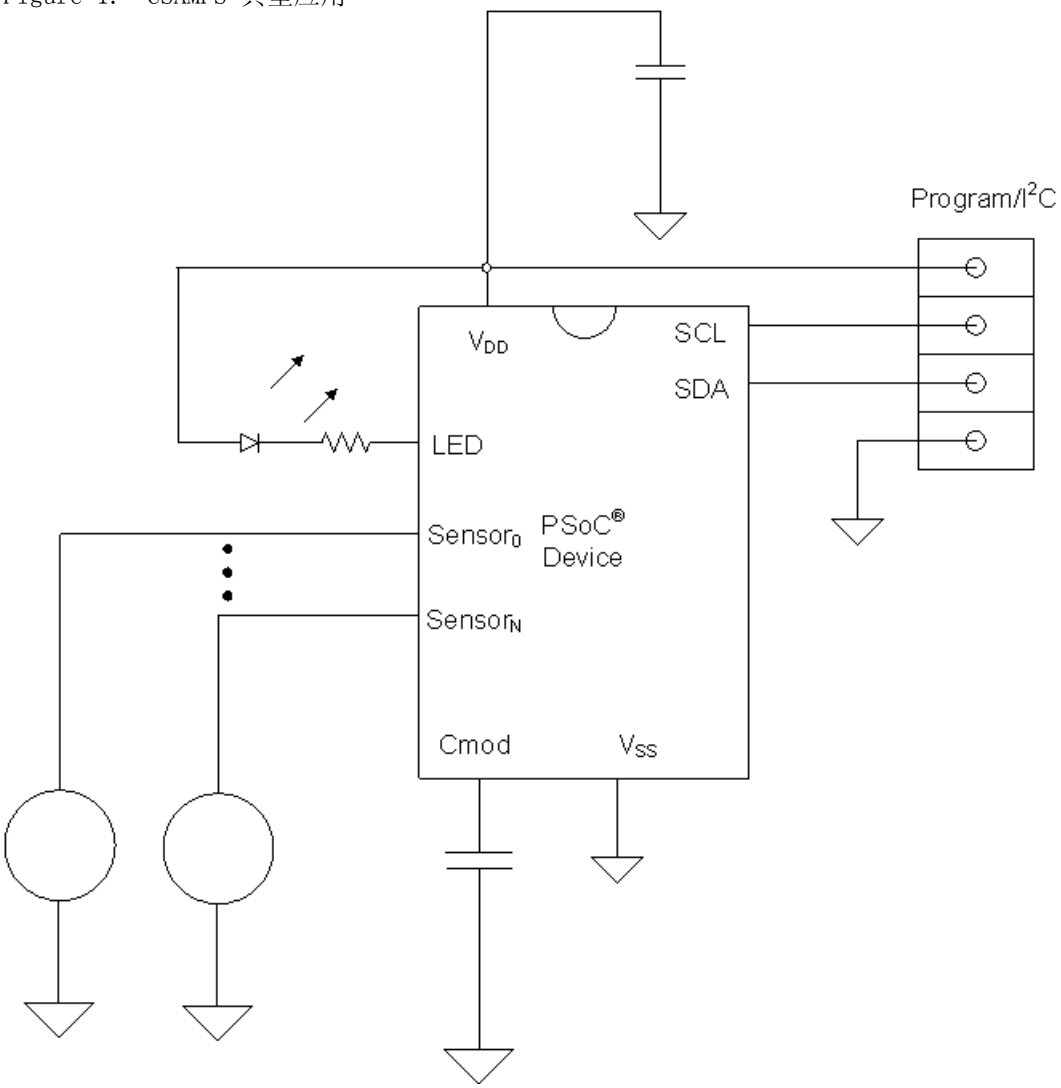
资源	PSoC® 模块				API 内存 （字节）典型值		引脚 （每个外部 I/O）
	I2C/SPI	CapSense	比较器	定时器	闪存	RAM	
CY8C20x34, CY8C20x24, CY8C20x66, CY8C20x36, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20x46, CY8C20x96, CYONSFN2053, CYONSFN2061, CYONSFN2151, CYONSFN2161, CYONSFN2162	–	1	1	–	1930	90	1

特性与概述

- 扫描 1 到 28 个电容传感器
- 扫描由 2 到 28 个元件构成的电容性滑条
- 采用复用法让滑条的物理分辨率翻倍
- 滑条插值分辨率可达 65535 分之 1
- 采用多个滑条传感器构成触摸板
- 传感器灵敏度、检测阈值和采样率可调
- 使用 CSAMFS 向导分配传感器 / 引脚
- 集成了能够处理温度变化的基准线更新算法
- 环境和物理传感器变动补偿

CapSense® 逐次逼近多频扫描 (CSAMFS) 用户模块使用开关式电容电路、一个模拟复用器、数字计数功能和用于补偿环境和物理传感器变动的高层软件例程实现了一个电容性触摸传感器阵列。 传感器阵列可以包含独立传感器、滑动条传感器和利用一对正交滑动传感器实现的触摸板的各种组合。 高层软件例程负责处理滑条复用。 滑条复用使得一个引脚能够测量两个不同物理位置处的两个电气传感器。 复用可增强滑条的分辨率，而无需另外增加一个 IO。

Figure 1. CSAMFS 典型应用



快速入门

1. 如果用到，请选择和布置需要专用引脚 （如 I^2C 或 LCD）的用户模块，并分配端口和引脚。
2. 选择和布置 CSAMFS 用户模块。
3. 右键单击 CSAMFS 用户模块以访问 CSAMFS 向导。
4. 设置按钮传感器个数、滑条配置以及引脚分配和关联项。
5. 设置引脚和全局用户模块参数。
6. 生成应用并切换到应用编辑器。
7. 改编样本代码以实现按钮、滑条或触摸板。

[+] Feedback

功能说明

电容传感器由物理、电气和软件组件组成。

CSAMFS 用户模块所测量的每个传感器都是一个电容，其一端接地，另一端连接到一个 PSoC 引脚。 导体的存在会增加接地传感器的电容。 这种电容变化可控制传感器的激活。

应用信息

本小节将逐步说明如何使用 CSAMFS 用户模块设计一个电容感应系统。 我们假定 PSoC 设备使用的电源为 Vdd = 2.7V 到 5.0V，且 C_{mod} 是一个 X7R 型电容。

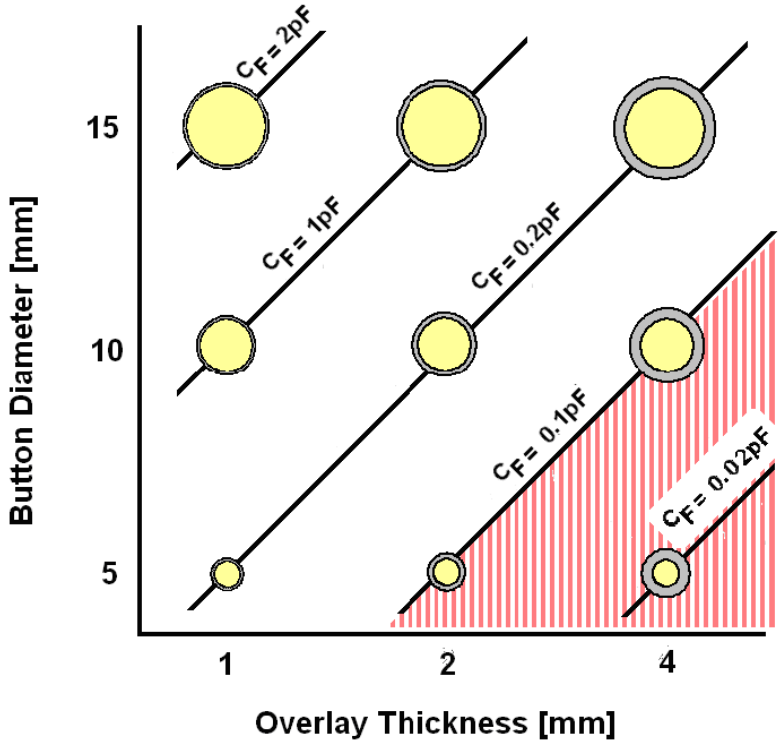
1. **按钮大小：** 使用图 2 中的图示选择针对特定外覆层厚度可产生正确水平的手指电容的按钮大小。 最小手指电容 C_F 为 0.1 pF，但建议采用 0.2 pF 以获得额外的设计余量。 按钮与接地填充物之间的间隙等于外覆层厚度。

   板设计与外覆层的组合是否满足手指电容的最低要求？

   如果回答为 “ 否 ”，请使用薄一些的外覆层，或增加传感器的大小。

   如果回答为 “ 是 ”，请转到第 2 步。

Figure 2. 按钮直径与外覆层厚度



示例： 塑料外覆层为 2 mm 厚。 设计目标为 C_F = 0.2 pF。 图 2 显示按钮直径需要为 10 mm，并且传感器板与接地填充物之间的间隙为 2 mm。

[+] Feedback

2. **CP 位于 5 到 50 pF 范围内：** 检查 CSAMFS 用户模块要监控的 C_P 的范围。 使用经验法则估算每个 PSoC 引脚上的电容负载，即在具有 8mil 走线的 63mil 厚的 2-layer 板上，走线电容大约为 2 pF/ 英寸。

   是否每个传感器都位于 5 pF 到 50 pF 的 C_P 范围内？

   如果回答为 “ 否 ”，请将传感器板更靠近 PSoC 设备一些，或考虑使用另一种 CapSense 方法，如 CSD。 任何长于 24″ 的走线都将超过 50pF 限值。

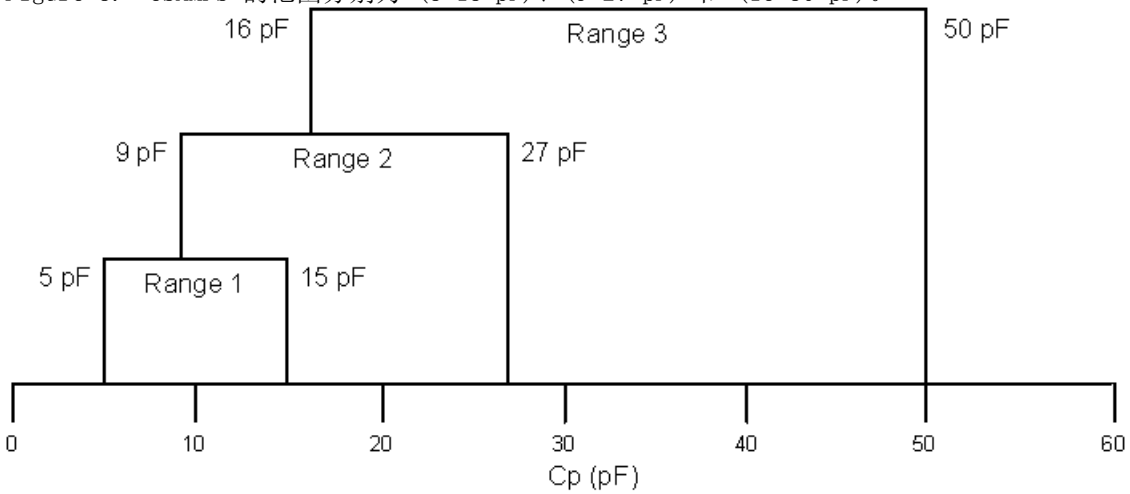
   如果回答为 “ 是 ”，请转到第 3 步。

   示例： 系统有 12 个触摸传感器。 所有传感器的传感器板到 PSoC 引脚的距离都小于 4″。 这些传感器的 PCB 空板的 C_P 范围为 9 pF 到 18 pF。 考虑到 PSoC 设备的封装效应（加 3 pF），PSoC 引脚上的总负载在 12 pF 到 21 pF 之间，可轻松满足 5-50 pF 的 C_P 限值要求。

3. **CP 范围：** 手指灵敏度和转换时间按图 3 所示的 C_P 范围之一指定。 这些范围分别为 5-15 pF、9-27 pF 和 16-50 pF。

   找出最适合在第 1 步中得到的 C_P 值的范围，并转到第 4 步。

Figure 3. CSAMFS 的范围分别为 (5-15 pF)、(9-27 pF) 和 (16-50 pF)。



   示例： PSoC 引脚上的总负载在 12 pF 到 21 pF 之间。 最适合这些值的 C_P 范围为范围 2，即 9 pF 到 27 pF。

[+] Feedback

4. **差值和阈值：** 估算差值，并设置手指和噪音阈值参数。 差值是由手指触摸传感器引起的计数增加。 差值可使用手指灵敏度 S_{FINGER} 和手指电容 C_F 按公式 1 计算得出。 手指阈值和噪音阈值可使用公式 2 和公式 3 计算得出 （请参见 AN2403）。

Equation 1

$$DifferenceCounts = S_{Finger} \times C_F$$

Equation 2

$$FingerThreshold = 0.75 \times DifferenceCounts$$

Equation 3

$$NoiseThreshold = 0.5 \times DifferenceCounts$$

计算这三个值并转到第 5 步。

示例： 系统具有下列参数。

C_P = 10 到 22 pF，C_F = 0.2 pF

IDACSetting = 7

SettlingTime = 245

C_{mod} = 2700 pF，X7R (+/-20%)

CSAMFS 时钟 = 6 MHz

IMO = 12 MHz

CPU = 6 MHz

解出差值、手指阈值和噪音阈值。

差值 = 500 计数 /pF * 0.2 pF = 100 计数

手指阈值 = .75 * 100 = 75 计数

噪音阈值 = .5 * 100 = 50 计数

5. **扫描时间：** 使用转换时间 t_C 按公式 4 估算扫描所有传感器的扫描时间 t_{SCAN}。 每个传感器的转换时间为 900 µs。

Equation 4

$$t_{SCAN} = t_C \times (numberofsensors)$$

计算此值并转到第 6 步。

示例： 系统在第 4 步定义了 12 个传感器。

tSCAN = 900 ms/ 传感器 * 12 个传感器 = 10.8 ms

[+] Feedback

6. **平均供电计数：** 使用活动电流 I_{active} 和睡眠电流 I_{sleep} 以及报告率按公式 5 估算平均供电电流 I_{DDCS}。

Equation 5

$$I_{DDCS} = [t_{SCAN} \times I_{active} + (ReportRate - t_{SCAN}) \times I_{sleep}] / (ReportRate)$$

示例： 从第 5 步继续，添加下列参数。

ReportRate = 100 ms

I_active = 1.13 mA

I_sleep = 2.6 mA

解出平均供电电流 IDDSC。

I_{DDCS} = [10.8 ms * 1.13 mA + 89.2 ms * 2.6 mA] / 100 ms = 124 mA

7. **串联电阻：** 确定抗射频干扰所需的串联电阻。
示例： PCB 走线为铜质，长度超过 25 mm。 在这些条件下，建议为所有 CapSense 输入使用 560Ω 的串联电阻。

直流和交流电气特性

除非另有说明，否则 Vdd = 2.7V 到 5.0V，C_{mod} = X7R 型电容，容差 ±20%。

Table 1. CSAMFS 用户模块的电气规范

符号	说明	条件	最小值	典型值	最大值	单位
C_P	寄生电容[a]	请参见有关 C_P 范围的注释。[b]	5		50	pF
C_F	手指电容[c]		0.1			pF
N	输出计数器分辨率		16		16	位
S_{FINGER}	手指灵敏度[d]	C_P = 5 到 15 pF IDAC 设置 = 5 建立时间 = 120 C_{mod} = 1200 pF CSAMFS 时钟 = 6 MHz IMO = 12 MHz CPU = 6 MHz	500			计数 /pF
		C_P = 9 到 27 pF IDAC 设置 = 7 建立时间 = 245 C_{mod} = 2700 pF CSAMFS 时钟 = 6 MHz， IMO = 12 MHz CPU = 6 MHz	500			计数 /pF

[+] Feedback

符号	说明	条件	最小值	典型值	最大值	单位
		C_P=16 到 50 pF IDAC 设置 = 5 建立时间 = 160 C_{mod} = 5600 pF CSAMFS 时钟 = 3 MHz IMO=12 MHz CPU=3 MHz	500			计数 /pF
t_C	转换时间，单个传感器。[d]	C_P = 5 到 15 pF IDAC 设置 = 5 建立时间 = 120 C_{mod} = 1200 pF CSAMFS 时钟 = 6MHz IMO = 12 MHz CPU = 6 MHz			700	ms/ 传感器
		C_P = 9 到 27 pF IDAC 设置 = 7 建立时间 = 245 C_{mod} = 2700 pF CSAMFS 时钟 = 6 MHz IMO = 12 MHz CPU = 6 MHz			900	ms/ 传感器
		C_P = 16 到 50 pF IDAC 设置 = 5 建立时间 = 160 C_{mod} = 5600 pF CSAMFS 时钟 = 3 MHz IMO = 12 MHz CPU = 3 MHz			2500	ms/ 传感器
I_{DDCS}	平均供电电流	Vdd = 3.3V C_P=5 到 15 pF 4 按钮扫描 100 ms 报告频率		35	50	mA
R_S	抗射频干扰的串联电阻[e] [5]	长度超过 25 mm 的高导电性走线 （铜或银墨）	300		560	欧姆

a. C_P 包括 2-3 pF 的封装相关电容。

b. 手指灵敏度和转换时间按三个 C_P 范围之一指定：(5-15 pF)、(9-27 pF)、(16-50 pF)

c. 手指电容是由手指触摸传感器引起的 C_P 的增加。

d. 选择 IDAC 设置以使手指触摸产生至少 50 个计数的信号。

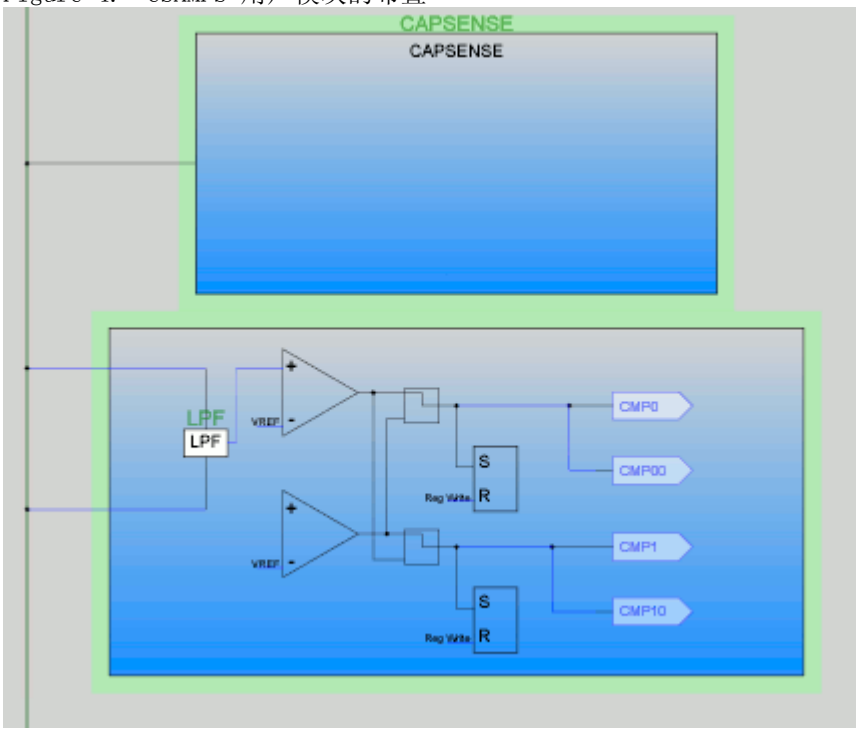
e. 较薄的低导电性 ITO 薄膜无需 R_S。 电阻放在 PSoC 引脚的 10 mm 范围内。

[+] Feedback

布置

用户模块的各个部分是在安装 UM 时自动放置的，没有可替换的放置位置。 需要特定引脚资源（包括 LCD 和 EZI2C）的用户模块必须在为 CSAMFS 用户模块建立端口引脚连接之前先设置好。 打开向导时，这些选择会反映在向导中。

Figure 4. CSAMFS 用户模块的布置



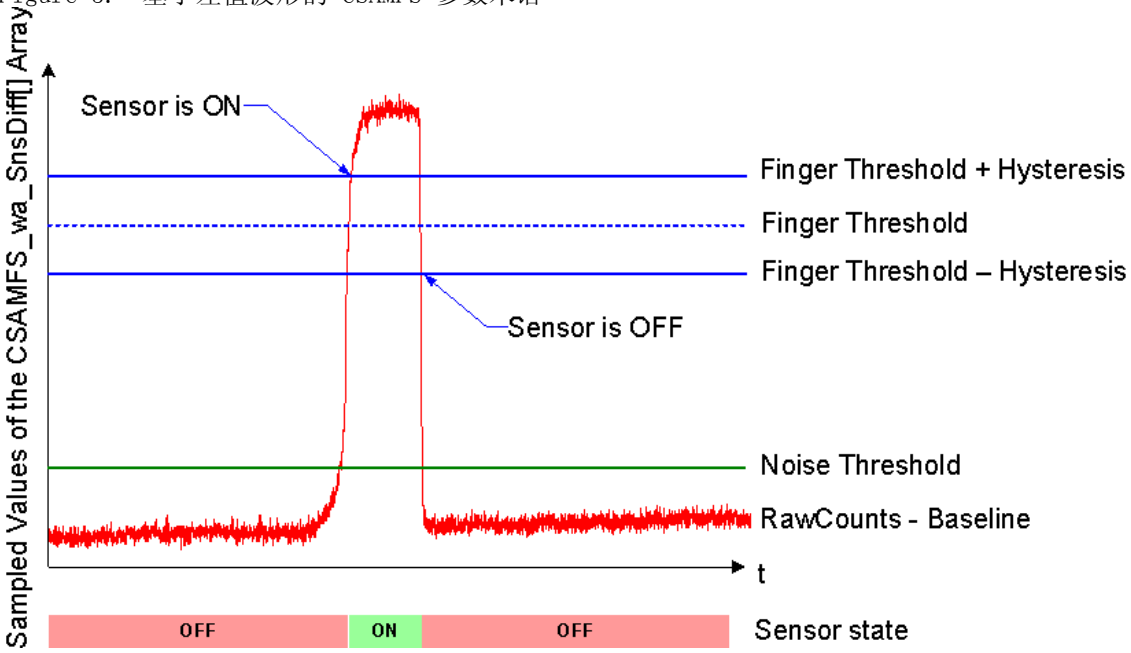
布置电容感应连接时请勿使用 P1[0] 和 P1[1]。 这些引脚用来给芯片编程，可能带有会降低传感器性能的过量布线电容。

[+] Feedback

参数和资源

图 5 揭示了 CSAMFS 参数中使用的一些术语。

Figure 5. 基于差值波形的 CSAMFS 参数术语



手指阈值

此阈值使用差值波形来决定每个按钮传感器的状态。 如果 wa_SnsDiff[] 数组中存储的差值达到或超过手指阈值，传感器将激活。

可能值的范围为 3 到 255。

噪音阈值

对于单个传感器，此参数设置一个数值，超出此值时将不更新基准线。

对于滑条传感器，它设置一个数值，那些低于它的扫描结果将会在计算中心位置的时候被忽略。

可能值为 3 到 255。

基准线更新阈值

当新的原始数值超过当前基准线且差值低于噪音阈值时，当前基准线与原始数值之间的差值将累计到一个 “ 水桶 ” 中。 当水桶已满时，基准线将递增并清空水桶。 此参数设置了基准线递增时水桶必须要达到的阈值。

可能值为 0 到 255。

SettlingTime

SettlingTime 参数控制等待 C_{mod} 电容上的电压稳定下来的软件延迟。 该循环每个迭代有 21 个 CPU 周期。 总延迟使用公式 6 计算：

Equation 6

$$Delay(\mu s) = \frac{6 + 21 \cdot (Settling\ Time)}{CPU_Speed(MHz)}$$

选择一个至少为 5 × R × C 的 SettlingTime，其中 R = 1 ÷ （时钟频率 × C_P），C = C_{mod}。

[+] Feedback

可能值为 2 到 255。 默认值为 20，对应于 12 MHz CPU 时钟频率下的 35.5 us 延迟。

ExternalCap

ExternalCap 参数用于选择 C_{mod} 电容连接的 PSoC 引脚。 建议的 C_{mod} 范围为 1200 pF 到 5600 pF。 C_{mod} 的值会影响手指灵敏度和转换时间。 有关详细信息，请参见 “ 直流和交流电气特性 ”。

可能值为 None、P0[1] 和 P0[3]。

迟滞

“ 迟滞 ” 参数会与手指阈值相加或相减，具体取决于传感器当前处于活动还是非活动状态。 如果传感器关闭，则差值必须超过手指阈值与迟滞之和。 如果传感器开启，则差值必须低于手指阈值减去迟滞后所得的差值。 它用于为手指检测算法增添平稳性和 “ 牢固性 ”。

可能值为 0 到 255。 但是，该设置必须低于 “ 手指阈值 ” 参数设置。

反跳

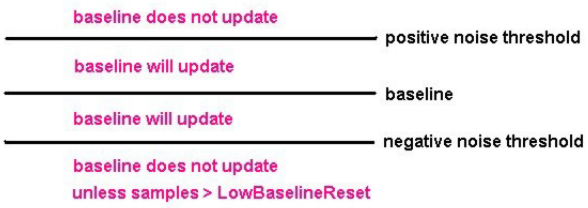
“ 反跳 ” 参数为传感器的活动状态切换添加了一个反跳计数器。 传感器要想从非活动状态切换到活动状态，差值必须在指定的采样数内保持超过手指阈值与迟滞之和。

可能值为 1 到 255。 设置 1 不提供平稳性。

NegativeNoiseThreshold

NegativeNoiseThreshold 参数会添加一个负的差值阈值。 如果当前原始数值低于基准线的计数超过 NegativeNoiseThreshold 参数，并且两者之差大于此阈值，则不会更新基准线。 但是，如果当前原始数值一直保持低状态 （差值大于阈值），并且次数超过 LowBaselineReset 参数指定的采样次数，基准线会被重置。

可能值为 0 到 255。



LowBaselineReset

LowBaselineReset 参数与 NegativeNoiseThreshold 参数配合使用。 如果采样到的数值低于基准线减去 NegativeNoiseThreshold，并且低于它的次数达到指定的采样次数，基准线会被重设成当前的原始数值。 它计算重设基准线所需的异常低的数值的次数， 它用来纠正 “ 启动时手指已经在（传感器上面）” 的情况。

可能值为 0 到 255。

传感器自动复位

此参数决定是在所有情况下都更新基准线还是仅当信号差低于噪音阈值时才更新基准线。 如果设置为 **Enabled**，基准线将持续更新。 该设置限定了传感器的最大持续时间长度（典型值为 5 - 10s），它会防止当没有任何东西触摸传感器而原始计数突然升高时传感器一直保持开启状态。 较大的供电电压波动、高能射频噪音源或极快的温度变化都会导致这种突然升高。

如果该参数设置为 **Disabled**，则仅当原始数值与基准线之差低于 “ 噪音阈值 ” 参数时才会更新基准线。

可能值为 Enabled 和 Disabled。

[+] Feedback

Freq Num

此参数允许以不同时钟频率扫描每个传感器三次，以此提高 EMI 性能。 Freq Num = 1 对应于标准扫描算法，Freq Num = 3 将启用高级算法。 启用高级扫描算法 (Freq Num = 3) 会将扫描时间和 RAM 消耗增加近三倍。

可能值为 1 和 3。

扩频

此参数允许在扫描过程中随机改变时钟频率值，以此提高 EMI 性能。 如果选择将 Freq Num 设置为 1，则启用此参数。

可能值为 Disable 和 Enable。

原始数据中值滤波器

该中值滤波器将查看传感器的最近三个样本并报告中值。 它用于消除短的噪音毛刺。 该滤波器会生成一个样本的延迟。 由于这种延迟以及 RAM 消耗，通常建议不要启用该滤波器。 启用该滤波器会消耗（传感器数 × 2 × Freq Num）字节的 RAM 以及 100 字节的闪存。 它默认为禁用。

原始数据 IIR 滤波器

此无限脉冲响应 (IIR) 滤波器可减少转换结果（原始数据）中的噪音。 对原始数据进行滤波会比对 XY 坐标进行滤波更有效，但需要更多 RAM。 启用此滤波器会额外消耗 100 字节的闪存。 它默认为禁用。

默认 IIR 系数为 0.5。

原始数据 IIR 滤波系数

这是原始计数 IIR 滤波器的系数。 “2” 表示 ½ 以前值 + ½ 当前值。 “4” 表示 ¾ 以前值 + ¼ 当前值。 2 和 4 是仅有的允许设置。

时钟

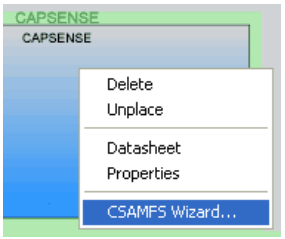
“ 时钟 ” 参数可用于增加传感器的有效电阻值。 如果传感器面积较大，有效电阻可能过高，以至于无法自动校准开关电容电路。 触摸板行 / 列或较大的接近传感器的灵敏度可能会下降。 在这种情况下，稳定电压会远远低于比较器阈值。 设置较大的 IMO 分频器可增加有效电阻，以补偿高电容。

可能值为 IMO、IMO/2、IMO/4 和 IMO/8。

向导

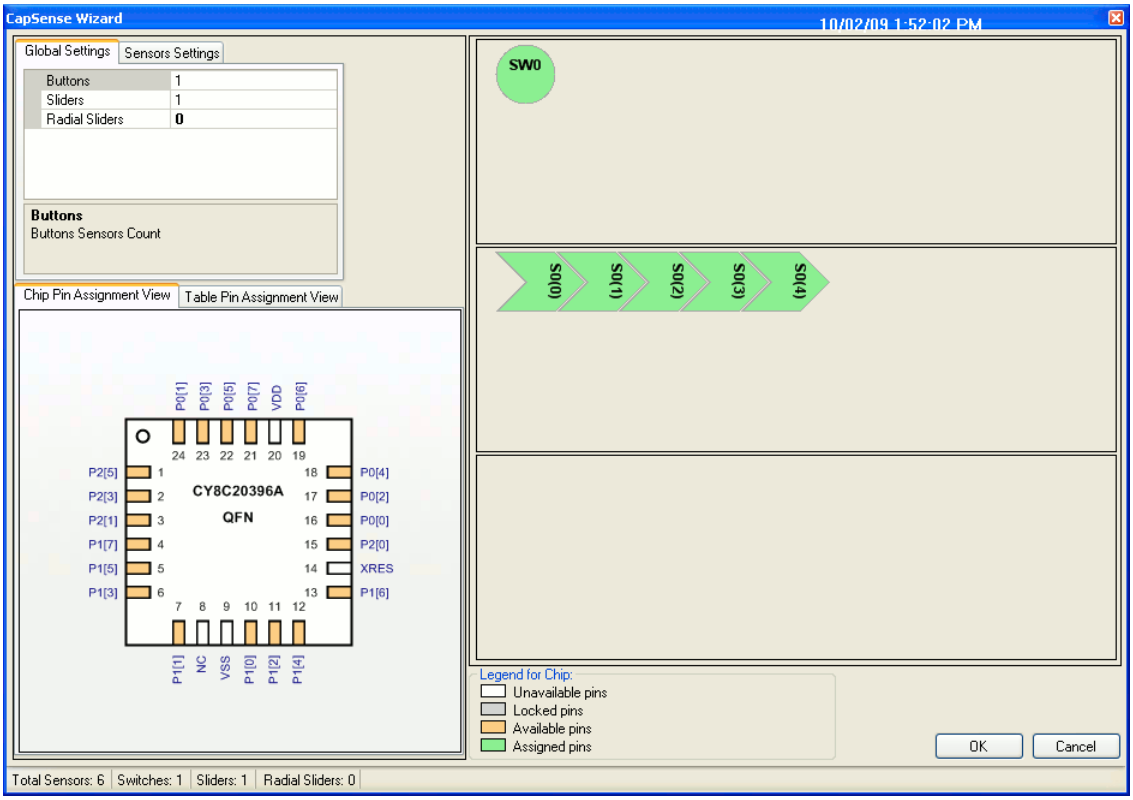
CSAMFS 向导用于设置 CapSense 按钮、滑条和接近传感器的引脚。 您可以使用一个拖放界面选择所需配置并分配按钮和段。

1. 要访问向导，请在 “ 设备编辑器互连视图 ”(Device Editor Interconnect View) 中右键单击任意 CSAMFS 模块，然后单击鼠标左键选择 “CSAMFS 向导 ”(CSAMFS Wizard)。



2. 向导将打开并显示按钮数和线形及放射形滑条数的数字输入框。

[+] Feedback



向导引脚图例

白色 – 该引脚不能用作 CapSense 输入。

灰色 – 该引脚处于锁定状态。 这可能是由于两个原因。 第一种可能性是另一个用户模块（如 LCD 或 I^2C）已经声明使用了该引脚。 第二种可能性是该引脚从其默认名称改成了其他名称。要将引脚恢复为默认名称，请在 “ 引脚 ”(Pinout) 视图中展开引脚，并从 “ **选择** ”(Select) 菜单中选择 “ **默认** ”(Default)。 这时该引脚在向导中便可用于分配。

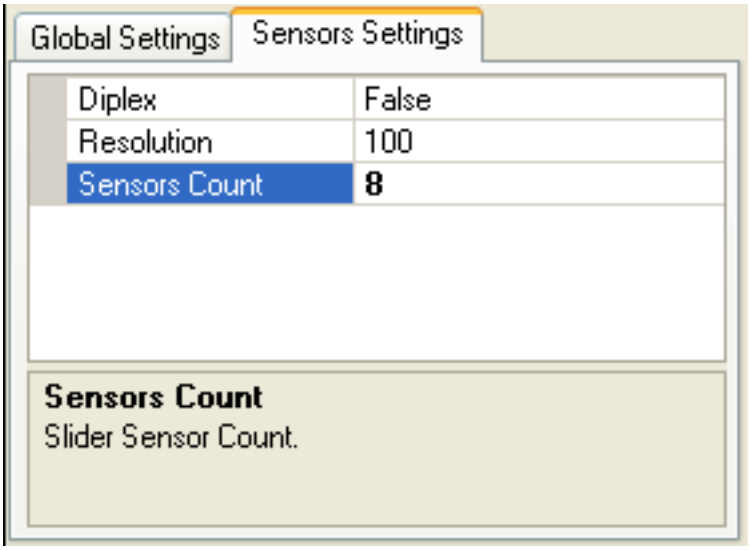
橙色 – 该引脚可用于分配。

绿色 – 该引脚已经分配为一个 CapSense 输入。

3. 键入按钮数。 传感器数受可用引脚数限制。 按 [Enter] 键为传感器数输入新值。
4. 键入线形和放射形滑条数。 X-Y 触摸板需要两个滑条。

[+] Feedback

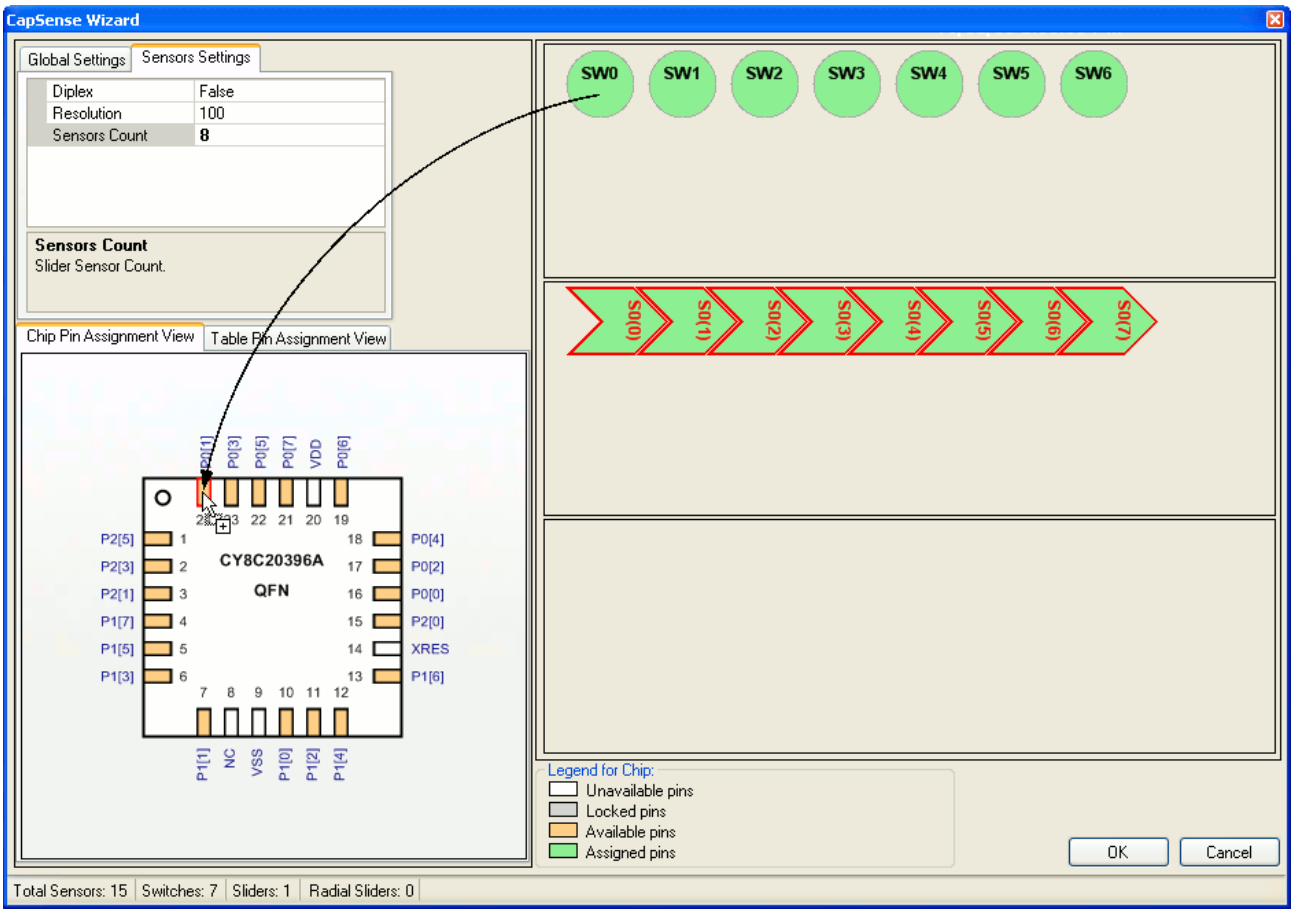
5. 单击其中一个滑条以启用该滑条的 “ 传感器设置 ”(Sensor Settings)。 键入每个滑条中的传感器元件数。 滑条传感器中的最低可行传感器数为 5，最大值由引脚数限定。 输入数据后，按 [Enter] 键输入新值。



6. 键入输出分辨率。 最小值为 5。 最大值为（用于传感器的引脚数 - 1）x 2^{16} - 1 或（2 x 用于传感器的引脚数 - 1）x 2^{16} -1（对于复用滑条）。
7. 如果需要，请选择 “ 复用 ”(Diplex)。 这会将选择用于传感器的引脚数映射到板上两倍数量的传感器位置， 但只会显示前半部分的复用传感器。 请参见 AN2292 中的复用表以了解引脚连接信息。

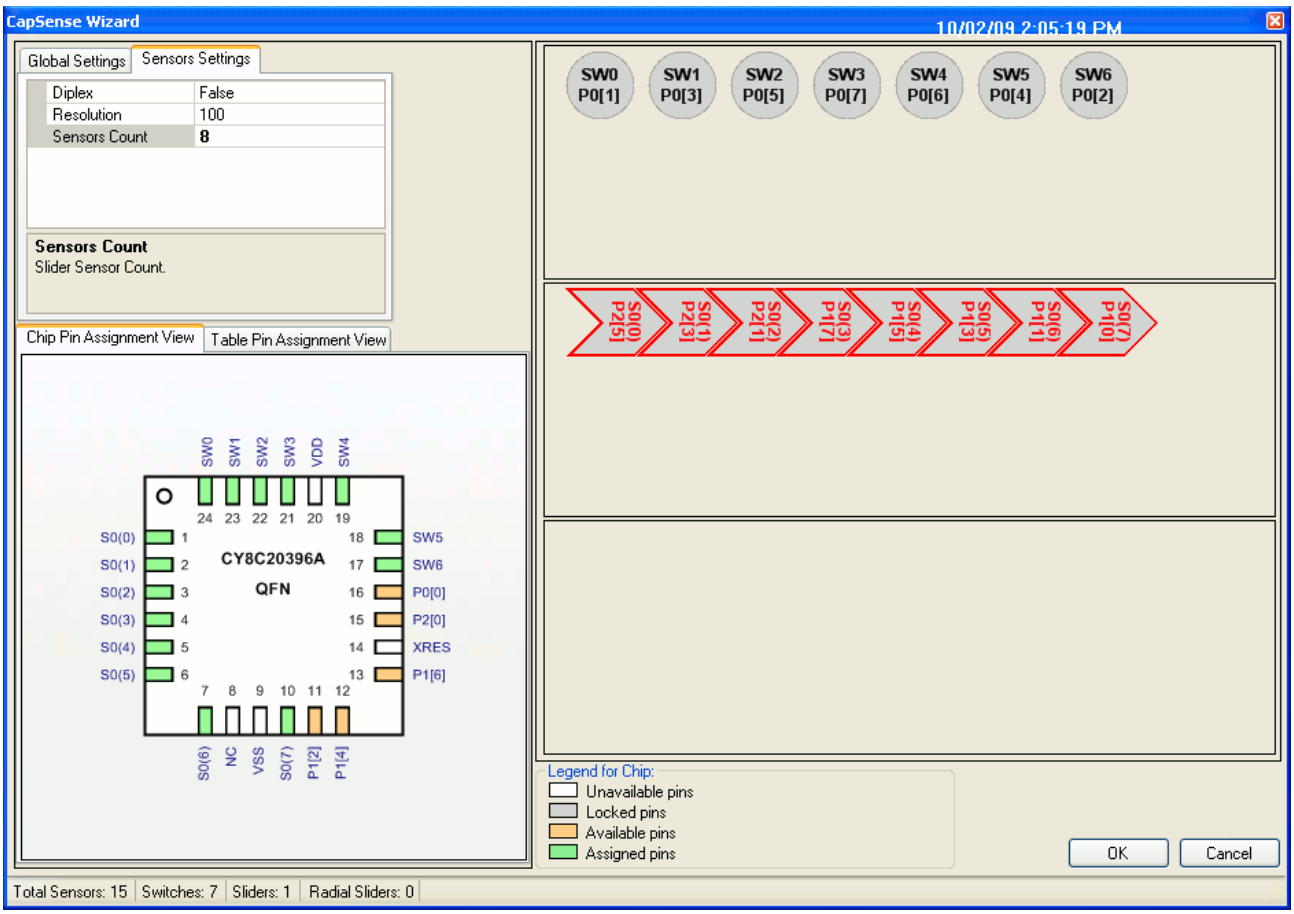
[+] Feedback

8. 通过在 “ 引脚分配视图 ”(Pin Assignment View) 中将开关或传感器拖放到引脚上将开关或传感器分配给引脚。 您可以选择在 “ 芯片引脚分配视图 ”(Chip Pin Assignment View) 或 “ 表引脚分配视图 ”(Table Pin Assignment View) 中将开关或传感器拖放到引脚上。 选定后，端口引脚将变为绿色，且不能再用于分配。 可以通过将传感器拖离端口以更改传感器的分配。



[+] Feedback

9. 为其余独立传感器重复分配操作。



将单个滑条传感器映射到物理端口引脚的操作与单个传感器相同。 单击 “ **确定** ”(OK) 以接受数据并返回到 PSoC Designer。

至此就完成了传感器的布置。 在 “ 设备编辑器 ”(Device Editor) 窗口中右键单击并选择 “ **刷新** ”(Refresh) 以更新引脚连接。

设置用户模块参数并生成应用程序。 如果愿意，现在您可以改编一个样本项目。

如果想要更改引脚分配，请将光标放在分配的引脚上，单击引脚并将其拖放到开关框之外。 该引脚将取消分配，您可以重新分配它。

完成向导后，单击 “ 生成应用程序 ”(Generate Application)。 根据您输入的传感器个数、引脚分配、复用和分辨率，将生成一组表。 这些表位于 CSAMFS_Table.asm 中。

[+] Feedback

应用程序编程接口

应用程序编程接口 (API) 例程作为用户模块的一部分提供，允许您编写用于与用户模块进行交互的代码而无需考虑其实现细节。 本节将说明接口的每个函数以及包含文件提供的相关常量。

Note 在这里，与所有用户模块 API 中的情况相同，调用 API 函数会改变 A 和 X 寄存器的值。 如果在调用之后仍需要这些值，则调用函数需负责在调用之前保留 A 和 X 的值。 选择这种寄存器易变策略是出于效率原因，该策略是从 PSoC Designer 1.0 版本开始实施的。 C 编译器会自动考虑这一要求。 汇编语言编程人员需确保其代码考虑到这一策略。 虽然某些用户模块 API 函数不会改变 A 和 X，但无法保证这些函数将来不会改变它们。

所提供的入口点用于初始化、启动和停止 CSAMFS 用户模块。 在所有情况下都要将以下入口点中所示的 CSAMFS 前缀替换为模块的实例名称。 未能使用正确的实例名称是造成语法错误的一个常见原因。

软件控制参数

传递给 API 的控制参数包括：

bSnsGroup

引用用作滑条的一组特定传感器。 CSAMFS_bGetCentroidPos 使用它来选择要更新的传感器组。

按钮包含在组 0 中。 滑条包含在组 1 及更高编号的组中。

bSensor

CSAMFS_wGetPortPin 使用传感器编号为选定活动传感器确定端口和位掩码 （bPort 和 bMask）。 CSAMFS_wGetPortPin 会返回 bMask 和 bPort。 CSAMFS_EnableSensor 和 CSAMFS_DisableSensor 使用这两项选择特定传感器。 CSAMFS_wReadSensor 还会使用它们设置要返回哪个传感器的计数。

CSAMFS 数据数组

API 函数使用多个全局数组。 您不能手动改变这些数组。 您可以检查这些值以进行调试。

CSAMFS_waSnsResult

该数组存储每个传感器的 16-bit 原始计数值。 其维度为 [频率数][传感器数]

CSAMFS_waIIR

该数组存储每个传感器的 16-bit 滤波原始数值 （如果启用了 IIR 滤波器）。 其维度为 [频率数][传感器数]

CSAMFS_waSnsBaseline

该数组存储每个传感器的 16-bit 基准线值。其维度为 [频率数][传感器数]。

CSAMFS_waSnsDiff

该数组存储每个传感器的 16-bit 差值 （原始数 - 基准线）。

CSAMFS_baSnsOnMask

该 8-bit 数组存储传感器的开 / 关数据 （对于按钮或滑条）。 数组中每个元素可存储最多 8 个传感器的传感器状态。CSAMFS_baSnsOnMask[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）。CSAMFS_baSnsOnMask[1] 包含传感器 8 到 15 （如果需要它们）的掩码位，以此类推。 该字节数组包含囊括所有布置的传感器所需的元素。 如果传感器开启，则位值为 1；如果传感器关闭，则位值为 0。

[+] Feedback

CSAMFS_baDACCodeScan

该数组存储 8-bit IDACSetting 参数值，这些参数值将设定 Cmod 上的 ADC 线性斜坡电压的斜率。只要在调用 CSAMFS_Start 之后立即加载该数组 （在初始化基准线或扫描任何传感器之前），便可以分别为每个传感器配置此参数。

CSAMFS_baDACCodeBaseline

该数组存储在 CSAMFS_Start. 中自动确定的每个传感器的 8-bit 校准设置。该数组中的值为负载每个 CapSense 输入的寄生电容提供了一个相对度量。

基本 API

基本 API 用于启动和停止用户模块。

CSAMFS_Start()

说明：

为每个传感器校准 CSAMFS 用户模块以及所有传感器斜坡 IDAC 值的通用值。 断开所有传感器引脚与模拟复用器总线的连接。 所有传感器引脚将转为接地。 将 C_{mod} 电容连接到系统。

C 原型：

```
void  CSAMFS_Start
```

汇编：

```
lcall  CSAMFS_Start
```

参数：

无

返回值：

无

副作用：

**

CSAMFS_Stop

说明：

禁用 CapSense 模块。 调用 CSAMFS_ClearSensors 以断开所有传感器引脚与模拟复用器总线的连接并将其转为接地。

C 原型：

```
void  CSAMFS_Stop()
```

汇编：

```
lcall  CSAMFS_Stop
```

参数：

无

返回值：

无

[+] Feedback

副作用：

**

CSAMFS_Calibrate

说明：

校准所有传感器斜坡 IDAC 值的通用值。此函数在调用 CSAMFS_Start() 时运行。 可以随时调用此函数以重新校准传感器。 该函数将调整 iDAC 电流以获得尽可能接近 wLevel 的原始计数。

C 原型：

```
void  CSAMFS_Calibrate(WORD wLevel)
```

汇编：

```
mov  A, <wLevel
mov X, >wLevel
lcall  CSAMFS_Calibrate
```

参数：

wLevel - 所需原始计数值

返回值：

无

数据表

向导会基于您输入的传感器个数、引脚分配、复用和分辨率生成一组数据表。 传感器表位于 CSAMFS_table.asm 中。 分组和复用表位于 CSAMFSHL.asm 中。

CSAMFS_Sensor_Table

传感器表针对每个传感器包含一个 2 字节条目。 第一个字节是端口号，第二个字节是位的位掩码（而不是位编号）。 该表包含所有的独立传感器，然后是依序排列的每个滑条传感器。 下面是一个包含六个传感器的示例表：

```
CSAMFS_Sensor_Table:
_CSAMFS_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
```

CSAMFS_Group_Table

分组表定义每个传感器组或滑条传感器组。 表中每一行的第一个条目是该组的起始传感器编号。 第二个条目是该组中的传感器数。 第三个条目是 0（如果未实现复用）。 第四、第五和第六个条目组合在一起形成用于中心位置计算的固定点乘数值。 下面是一个包含七个传感器的示例：

```
CSAMFS_Group_Table:
CSAMFS_Group_Table:
// Group Table
//        Origin Count  Diplex?  SliceMultiplier
      db    0,    0x7,    0x0,    0x00     // Buttons
```


[+] Feedback

在具有独立传感器和滑条传感器的项目中，独立传感器组以及每个滑条传感器在分组表中都有一个单独的条目，如下所示：

```
CSAMFS_Group_Table:
CSAMFS_Group_Table:
; Group Table
;         Origin Count  Diplex?  DivBtwSns(wholeMSB, wholeLSB, fractByte)
      db    0,    0x7,    0x0,    0x00,    0x00,    0x00 ; Buttons
      db    0x6,  0xA,    0x4,    0x0,     0x7,     0xE5 ; Slider 1
```

CSAMFS_Diplex_Table

复用表定义每个滑条传感器的传感器全范围映射。 该表包含两部分：每个滑条的传感器映射，以及每个单独滑条对其表的引用。 下面是一个包含 10 个传感器的滑条的典型示例。

```
DiplexTable_0:
; This group is not a diplexed slider

DiplexTable_1:
    db    0,1,2,3,4,5,6,7,8.9,0,3,6,9,1,4,7,2,5,8   // 10 switch slider

CSAMFS_Diplex_Table:
_CSAMFS_Diplex_Table:
    db >DiplexTable_0, <DiplexTable_0
    db >DiplexTable_1, <DiplexTable_1
```

低层 API

低层 API 用于获取传感器数据。

CSAMFS_ScanSensor

说明：

扫描单个传感器以确定代表其电容的原始数值。 该例程假定在执行之前调用了 CSAMFS_Start 函数。 该例程是一个模块级的调用，即它会等待 CapSense 模块中断发生后 CSAMFS_ISR 才完成。 然后，它会将来自 16-bit 计数器的数据传递给 CSAMFS_wADCResult 全局变量。

C 原型：

```
void  CSAMFS_ScanSensor(BYTE bSensor)
```

汇编：

```
mov  A,  bSensor
lcall CSAMFS_ScanSensor
```

参数：

bSensor： 范围为 0 到 n-1，其中 n 是在 CSAMFS 向导中设置的传感器总数与包含在滑条传感器中的传感器数之和。

返回值：

无

副作用：

**

[+] Feedback

CSAMFS_ScanAllSensors

说明：

通过为每个传感器索引调用 CSAMFS_ScanSensor 扫描所有配置的传感器。 然后，它将所有启用的滤波器应用于原始数据和更新后的所有基准线。

C 原型：

```
void  CSAMFS_ScanAllSensors()
```

汇编：

```
lcall  CSAMFS_ScanAllSensors
```

参数：

无

返回值：

无

副作用：

**

CSAMFS_ClearSensors

说明：

针对每个传感器执行 CSAMFS_DisableSensor。 传感器引脚将断开与模拟复用器总线的连接并转为接地。

C 原型：

```
void  CSAMFS_ClearSensors()
```

汇编：

```
lcall  CSAMFS_ClearSensors
```

参数：

无

返回值：

无

副作用：

**

CSAMFS_wGetPortPin

说明：

返回指定传感器的端口号和引脚掩码。 所传递的参数将指引并选择 CSAMFS_Sensor_Table. 中的数据。

C 原型：

```
WORD  CSAMFS_wGetPortPin(BYTE bSensor)
```


[+] Feedback

汇编：

```
mov  A,  bSensor
lcall     CSAMFS_wGetPortPin
```

参数：

bSensor：范围为 0 到 n-1，其中 n 是在 CSAMFS 向导中设置的传感器总数与包含在滑条传感器中的传感器数之和。

返回值：

bPort 和 bMask：用于确定所选择的特定传感器的端口号和位掩码。

副作用：

**

CSAMFS_EnableSensor

说明：

为下一个测量周期中的扫描配置选定的传感器。传感器的端口和引脚是使用 CSAMFS_wGetPortPin 例程选择的，其中端口号和传感器位掩码分别加载到 X 和 A 中。驱动模式将修改以便设定选中的端口和引脚为“Analog High Z”模式，并启用正确的模拟复用器总线输入。

C 原型：

```
void  CSAMFS_EnableSensor(BYTE bMask, BYTE bPort)
```

汇编：

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

参数：

bPort 和 bMask：用于确定所选择的特定传感器的端口号和位掩码。

返回值：

无

副作用：

**

CSAMFS_DisableSensor

说明：

断开传感器的连接，使其不再是一个输入。通常，该调用与 CSAMFS_wGetPortPin. 配合使用。这将断开端口引脚与模拟复用器总线的连接。驱动模式将更改为 Strong (01)，数据寄存器位设置为 0。这会将传感器接地。

C 原型：

```
void  CSAMFS_DisableSensor(BYTE bMask, BYTE bPort)
```

汇编：

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


[+] Feedback

参数：

bPort 和 bMask： 用于确定所选择的特定传感器的端口号和位掩码。

返回值：

无

副作用：

**

高层 API

高层 API 用于处理低层 API 获取的传感器数据。

CSAMFS_UpdateSensorBaseline

说明：

为一个传感器更新传感器基准值。 为每个传感器单独计算的历史数据称为传感器的基准线。 该基准线值使用 “ 水桶方法 ” 进行更新。

水桶方法使用以下算法。

1. 每次调用 CSAMFS_UpdateSensorBaseline 时，通过从原始数值中减去前一个基准线值，得到差值。该差值存储在 waSnsDiff 数组中。
2. 每次调用 CSAMFS_UpdateSensorBaseline 时，都会将该差值与噪音阈值进行比较。 如果差值低于噪音阈值，则会将差值的一半加到 / 累加到一个虚拟水桶中 . 如果差值高于噪音阈值，则不更新基准线值。
3. 当虚拟水桶中的累计差值达到 BaselineUpdateThreshold 时，基准线值将递增，同时水桶将复位为 0。
4. 如果差值低于噪音阈值，waSnsDiff 数组中存储的值将设为 0。

该函数是从 CSAMFS_ScanAllSensors() 中自动调用的。

C 原型：

```
void CSAMFS_UpdateSensorBaseline(BYTE bSensor)
```

汇编：

```
mov   A, bSensor
lcall  CSAMFS_UpdateSensorBaseline
```

参数：

bSensor： 范围为 0 到 n-1，其中 n 是在 CSAMFS 向导中设置的传感器总数与包含在滑条传感器中的传感器数之和。

返回值：

无

副作用：

**

[+] Feedback

CSAMFS_bIsSensorActive

说明：

检查指定传感器（上的电容变化量）与其手指阈值相比的差值。 其中会考虑 “ 迟滞 ”。 根据传感器的状态，“ 迟滞 ” 值将与手指阈值相加或相减。 如果传感器处于活动状态，则降低阈值。 如果处于非活动状态，则升高阈值。 该函数还会更新 CSAMFS_baSnsOnMask 数组中的传感器的位。

C 原型：

```
BYTE  CSAMFS_bIsSensorActive(BYTE bSensor)
```

汇编：

```
mov  A,  bSensor
lcall CSAMFS_bIsSensorActive
```

参数：

bSensor： 范围为 0 到 n-1，其中 n 是在 CSAMFS 向导中设置的传感器总数与包含在滑条传感器中的传感器数之和。

返回值：

如果处于活动状态，则返回值为 1，如果处于非活动状态，则返回值为 0。

副作用：

**

CSAMFS_bIsAnySensorActive

说明：

检查所有传感器（采到的变化量）与其手指阈值相比的差值。 为每个传感器调用 CSAMFS_bIsSensorActive 以便在调用此函数之后使 CSAMFS_baSnsOnMask 数组保持最新状态。

C 原型：

```
BYTE  CSAMFS_bIsAnySensorActive()
```

汇编：

```
lcall CSAMFS_bIsAnySensorActive
```

参数：

无

返回值：

如果处于活动状态，则返回值为 1，如果处于非活动状态，则返回值为 0。

副作用：

**

CSAMFS_SetDefaultFingerThresholds

说明：

为 CSAMFS_baBtnFThreshold 数组加载 FingerThreshold 参数值。 如果没有为 CSAMFS_baBtnFThreshold 数组手动加载定制值，则必须在扫描之前调用此函数。

[+] Feedback

C 原型：

```
BYTE  CSAMFS_SetDefaultFingerThresholds()
```

汇编：

```
lcall CSAMFS_SetDefaultFingerThresholds
```

参数：

无

返回值：

无

副作用：

**

CSAMFS_InitializeBaselines

说明：

通过扫描每个传感器为 CSAMFS_waSnsBaseline 数组加载初始值。

C 原型：

```
void  CSAMFS_InitializeBaselines()
```

汇编：

```
lcall CSAMFS_InitializeBaselines
```

参数：

无

返回值：

无

副作用：

**

CSAMFS_InitializeSensorBaseline

说明：

为 CSAMFS_waSnsBaseline 数组加载特定传感器的初始值。 用于为特定传感器复位基准线。

C 原型：

```
void  CSAMFS_InitializeSensorBaseline(BYTE bSensor)
```

汇编：

```
lcall CSAMFS_InitializeSensorBaseline
```

参数：

bSensor： 范围为 0 到 n-1，其中 n 是在 CSAMFS 向导中设置的传感器总数与包含在滑条传感器中的传感器数之和。

返回值：

无

[+] Feedback

副作用：

**

CSAMFS_wGetCentroidPos

说明：

检查差值数组的中心。 如果存在一个中心，则会在临时变量中存储偏差和长度并根据 CSAMFS 向导中指定的分辨率计算中心位置。

C 原型：

```
WORD  CSAMFS_wGetCentroidPos(BYTE bSnsGroup)
```

汇编：

```
mov  A,  bSnsGroup
lcall CSAMFS_wGetCentroidPos
```

参数：

bSnsGroup： 引用用作滑条的一组特定传感器。

返回值：

滑条的位置值，LSB 位于 A 中，MSB 位于 X 中。

副作用：

该例程通过减去噪音阈值来修改差值。 该例程只应在每次扫描后调用一次以避免得到负差值。 如果您的应用要监控差值信号，请在传输差值数据之后调用此例程。

注： 如果滑条段的噪音数值大于噪音阈值，此子例程可能会生成错误的中心位置结果。 噪音阈值应小心设置 （高出噪音水平足够程度），以确保噪音不会生成错误的中心位置。

[+] Feedback

固件源代码样本

扫描按钮以及打开和关闭 LED

此代码是为 CSAMFS UCC 板 (CY3280-20x34) 和线性滑条模块板 (CY3280-SLM) 编写的。

```
//------ Sample code for CSAMFS buttons that LEDs On and Off   ----------
//------     pin assignments for Linear Slider Module plugged   ---------
//-------     into CSAMFS UCC board, CY3280-20x43+SLM            --------

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

void main(void)
{
//initialize LED states
  PRT0DR |= 0b00100010;  //turn-off LED on P0[5],P0[1]
  PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
  PRT2DR |= 0b10100000;  //turn-off LED on P2[7],P2[5]

//Set port drive modes for LEDs
  PRT0DM0 |= 0b00100010;  //strong on P0[5],P0[1]
  PRT0DM1 &=~0b00100010;

  PRT1DM0 |= 0b10100100;  //strong on P1[2]
  PRT1DM1 &=~0b10100100;

  PRT2DM0 |= 0b10100000;  //strong on P2[5],P2[7]
  PRT2DM1 &=~0b10100000;

  M8C_EnableGInt;  //enable global interrupts for use with CSAMFS

  CSAMFS_Start();     //initialize the CSAMFS User Module
  CSAMFS_SetDefaultFingerThresholds();   //Load finger thresholds
  CSAMFS_InitializeBaselines();  //Set baselines to current count

  while(1)  //infinite loop scanning buttons
  {
    CSAMFS_ScanAllSensors();    //sample all buttons and compute baselines

// control the LEDs using the sensor states.
// LED ON if active, OFF if not active.
// Check buttons in sequence.
    if (CSAMFS_bIsSensorActive(0))
    {
      PRT1DR &= ~0b00000100;   //turn-on LED on P1[2]
    }
    else
    {
      PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
    }
    if (CSAMFS_bIsSensorActive(1))
    {
      PRT0DR &= ~0b00100000;  //turn-on LED on P0[5]
    }
```


[+] Feedback

```
    else
    {
      PRT0DR |= 0b00100000;  //turn-off LED on P0[5]
    }
    if (CSAMFS_bIsSensorActive(2))
    {
      PRT0DR &= ~0b00000010;  //turn-on LED on P0[1]
    }
    else
    {
      PRT0DR |= 0b00000010;  //turn-off LED on P0[1]
    }
    if (CSAMFS_bIsSensorActive(3))
    {
      PRT2DR &= ~0b10000000;  //turn-on LED on P2[7]
    }
    else
    {
      PRT2DR |= 0b10000000;  //turn-off LED on P2[7]
    }
    if (CSAMFS_bIsSensorActive(4))
    {
      PRT2DR &= ~0b00100000;  //turn-on LED on P2[5]
    }
    else
    {
      PRT2DR |= 0b00100000;  //turn-off LED on P2[5]
    }
  }
}
```

使用线性滑条控制 LED 亮度

此代码是为 CSAMFS UCC 板 (CY3280-20x34) 和线性滑条模块板 (CY3280-SLM) 编写的。

```
//------ Sample code for CSAMFS slider controlling LED intensity -------
//------   pin assignments for Linear Slider Module plugged   ----------
//-------  into CSAMFS UCC board, CY3280-20x43+SLM               -------

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

int wCentroid = 0;  //estimated finger position; 0xffff for no finger
int wPos = 0;  //estimated finger position
int wLED_PWM; //controls LED intensity

void main(void)
{
//initialize LED states
  PRT0DR |= 0b00100010;  //turn-off LED on P0[5],P0[1]
  PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
  PRT2DR |= 0b10100000;  //turn-off LED on P2[7],P2[5]

//Set port drive modes for LEDs
```


[+] Feedback

```
  PRT0DM0 |= 0b00100010;  //strong on P0[5],P0[1]
  PRT0DM1 &=~0b00100010;

  PRT1DM0 |= 0b10100100;  //strong on P1[2]
  PRT1DM1 &=~0b10100100;

  PRT2DM0 |= 0b10100000;  //strong on P2[5],P2[7]
  PRT2DM1 &=~0b10100000;

  M8C_EnableGInt;  //enable global interrupts for use with CSAMFS

  CSAMFS_Start();     //initialize the CSAMFS User Module
  CSAMFS_SetDefaultFingerThresholds();   //Load finger thresholds
  CSAMFS_InitializeBaselines();  //Set baselines to current count

  while(1)  //infinite loop scanning slider
  {
    CSAMFS_ScanAllSensors(); //sample all sensorsand and compute baselines

    wCentroid = CSAMFS_wGetCentroidPos(1); //estimated position
    if (wCentroid != 0xffff)  //0xffff means finger off slider
    {
      wPos = wCentroid;  //get position, range is 0 to 100
    }

    if (wPos > 0)  //if position>0, then pulse all LEDs ON
    {
      PRT1DR &= ~0b00000100;  //turn-on LED on P1[2]
      PRT0DR &= ~0b00100000;  //turn-on LED on P0[5]
      PRT0DR &= ~0b00000010;  //turn-on LED on P0[1]
      PRT2DR &= ~0b10000000;  //turn-on LED on P2[7]
      PRT2DR &= ~0b00100000;  //turn-on LED on P2[5]

      for (wLED_PWM = 0; wLED_PWM < wPos*wPos/100; wLED_PWM++)
      {   //control LED pulse width by position^2
      }  //this control function looks nice

    }
    // LED pulse ON is over for this period, turn all off
    PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
    PRT0DR |= 0b00100000;  //turn-off LED on P0[5]
    PRT0DR |= 0b00000010;  //turn-off LED on P0[1]
    PRT2DR |= 0b10000000;  //turn-off LED on P2[7]
    PRT2DR |= 0b00100000;  //turn-off LED on P2[5]


  }  //do next scan (while loop)
}
```


[+] Feedback

更多参考资料

建议在了解 CDAMFS 用户模块文档之后进一步阅读以下应用笔记。 这些应用笔记可在赛普拉斯半导体公司的网站 (www.cypress.com) 上找到：

- *CapSense 最佳实践* – AN2394
- *CapSense 应用的信噪比要求* – AN2403
- *用于调试 CapSense 应用的绘图工具* – AN2397
- *PSoC CapSense 应用的 EMC 设计考虑事项* – AN2318
- *电容式感应应用中的能耗和睡眠考虑事项* – AN2360
- *PSoC CapSense 布线指南* – AN2292
- *通用异步发送器的软件实现* – AN2399
- *电容式感应的防水* – AN2398

版本历史记录

版本	创作者	说明
1.1	DHA	在用户模块和配置向导中添加了放射形滑条功能。
1.1.a	DHA	将内联函数移到库中。

Note PSoC Designer 5.1 在所有用户模块数据手册中提供了一个版本历史记录。 这一部分概要说明了当前与以前的用户模块版本之间的差异。

Copyright © 2009-2010 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.