用户指南
ZHCU200 – 2017 年 6 月

CapTIvate™ 技术指南 – 简化版

本文件是对 MSP430™ CapTIvate™ 触控技术的简要介绍。

最新和最齐全的文档（英文版）可通过登录以下网址获得：http://www.ti.com/CapTIvateTechGuide

目录

Chapter 1

Main Page

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments
13532 N. Central Expressway
Dallas, TX 75243
www.ti.com

2 准备开始

2.1 欢迎阅读 CapTIvate 准备开始

直接点击下面标题可以直接转到相应最受欢迎的章节。

2.1.1 起点

- 引言
- 开箱即用的体验
- CapTIvate 设计流程概要
- CapTIvate Design Center 快速入门
- CCS/IAR 项目快速入门

2.1.2 参考资料

- 电容式感测基础知识

2.1.3 专题练习

- 创建一个新的电容感测传感器设计项目
- 有关低功耗的实验
- 考察 CapTIvate 触控函数库

3 引言

3.1 德州仪器推出 CapTIvate 技术

CapTIvate 整合了一种令人兴奋的电容式触摸测量技术、一个功能强大的 CapTIvate Design Center GUI、一个多用途的硬件开发平台和一个全功能的电容式触控软件库，从而在电容式触控传感器设计流程演化的道路上迈进了一步。

本文件是技术、工具、软件和设计文件相关信息的集合，而且它提供了利用新型 CapTIvate 技术启动开发工作所需的一切。

3.2 本指南的构成

本指南被分为以下部分。

准备开始 — 从这里开始以创建一个新项目或尝试任何令人惊奇的电容式触摸演示。

电容式感测基础知识 — 本章概述了自电容和互电容以及电容式感测方法的基本原理。

技术介绍 — 详细描述了 CapTIvate 硬件模块，包括测量原理和测量信号链。另外，还讨论了低功耗特性和接近传感 (wake-on-proximity) 状态机。

设计指南 — 优良的传感器设计是造就成功触控产品的基础。本设计指南旨在为电容式触摸传感器的设计和布局提供指导，以使其能够实现性能的最大化。在最优传感器硬件设计的基础上，CapTIvate 触控软件库可以简化软件设计和优化以达到最低功耗。另外，Captivate Design Center 还可以用来在线调整电容触摸的参数和性能。

Design Center GUI — CapTIvate Design Center 是一款快速开发工具，其可加快面向那些支持 CapTIvate 技术的 MSP430FRxx 器件的电容式触摸设计。通过帮助指引产品开发人员逐步了解整个电容式触摸开发过程，CapTIvate Design Center 能够通过运用创新的用户图形界面、向导和控件来简化和加速任何的触摸设计。

器件家族 — 介绍了集成 CapTIvate 技术模块的 MSP430 器件。

软件库 — CapTIvate 电容式触摸软件库全面汇集了触摸、通信和传感器管理的函数。触摸函数库包含用于按钮、滑块和滚轮的高级传感器处理函数和对 CapTIvate 外设的直接访问函数。

MCU 开发套件 — MSP-CAPT-FR2633 MCU 开发套件是一款针对 MSP430FR2633 MCU 的易用型评估平台。它包含了在该 MCU 平台上学习开发所需的一切，包括用于编程、调试和 EnergyTrace 功耗测量的板极模拟。

专题练习 — 该专题练习逐步讲解了如何运用 CapTIvate 生态系统和 CapTIvate 电容式触摸 MCU 开发套件来启动开发工作。此专题练习的内容包括工具设置、开箱即用体验和三个动手操作练习。

4 电容式感测基础知识

这部分概述了自电容和互电容以及电容式感测方法的基本原理。

电容式感测是通过检测传感器元件的电容变化来实现的。传感器元件可以是任意的导电材料（例如：PCB 铺铜或导线）。

- 变化的原因可以是由于人机互动（比如：手指、耳朵或手）。
  这常常被称为电容式触摸或接近感应。
- 然而，电容式感测并不限于人机互动。其他的物体或材料也会改变传感器的电容。这可能是某种有机或无机材料，如金属或液体。在任一种场合中，电容如何变化的解释决定了应用。

电容是某种物体存储电荷能力的衡量尺度。任何两种被某个绝缘体隔离开来的导电材料只要它们的距离足够近，就会呈现出电容。

可以创建一个相似的系统，在该系统中，第二个电容实际上就是一个与系统互动的人。

- 铜电极构建在 FR4 PCB 材料上，并具有一个相对大地的自由空间耦合电容。
- 用户触摸覆盖物的顶部。覆盖物是一个绝缘体，并在用户与铜电极之间起电介质的作用。因而，额外的电容被添加至电极。

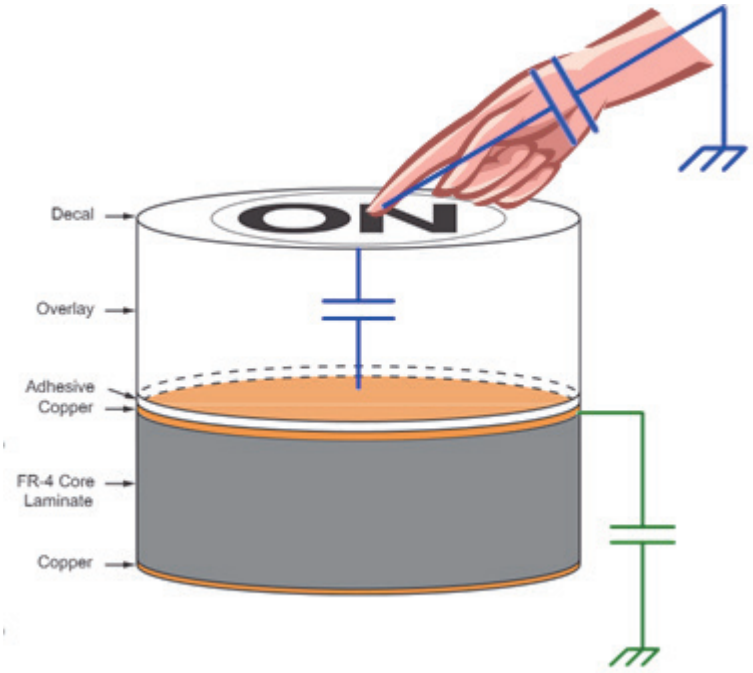


图 1：电容感测

4.1 自电容

这种测量相对于大地的电容变化的方法通常被称为自电容测量。有的时候它也被称为表面电容。在平行板模型中，电极规定了电容器的一个极板，另一个极板则是地（ C_electrode）或用户的手指（ C_touch）。一次触摸会导致电极的电容增加（通常为 1 pF 至 10 pF）。

当描述一款电容式触摸解决方案中使用的各种不同的电容时，一种等效电路模型会在表现不同电容的来源以及每种电容的影响的过程中有所帮助。图 2 是用于单个自电容按钮的等效电路实例。

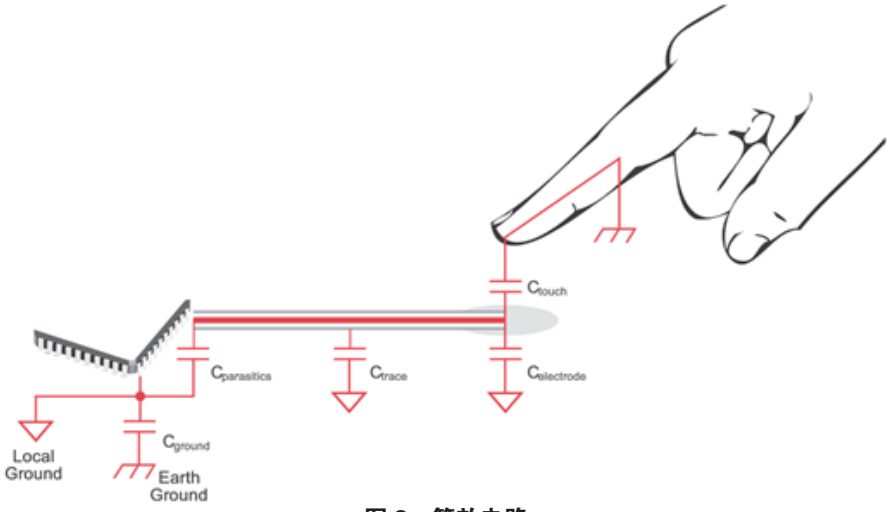


图 2：等效电路

图 2 中示出了五种不同的电容。

- C_ground 是被测器件 (DUT) 局部地与大地之间的电容。在某些应用中，当 DUT 使用主电源时，局部地和大地是连接的，但是通常情况下局部地是被容性耦合回大地的。
- C_trace 和 C_electrode 分别是走线和电极结构至局部地之间的电容。周边结构直接对该电容产生最严重的影响，通常是接地覆铜（位于同一层或相邻层）。图中未予示出的是走线和电极结构与大地之间的电容。这些电容并非一点影响没有；不过，为简单起见，在本文件的内容当中，设计指导是以影响局部电容（即：局部地与走线和电极之间的分离）的原则提供的。
- C_parasitics 是微控制器与电路内任何组件的内部寄生电容的组合。该电容也参考于局部地。
- C_touch 是在触摸互动和电极之间形成的平行板电容。在触摸的实例中，当手指挤压覆盖物时，变平的手指表面形成了上极板，而电极则形成了下极板。该电容是两块极板的面积、两块极板之间的距离以及分离两块极板之材料的介电常数的一个函数。

4.1.1 自电容寄生效应

除了任何 PCB 走线、导线、连接器和其他信号源之外，还有对局部地和大地的寄生电容。寄生电容会减弱用户触摸在系统中的影响，因为系统测量的是电容的变化。例如：

- 假设由于一次触摸引起的电容变化为 5 pF。在该场合中，如果寄生电容仅为 20 pF，则系统测量的电容增幅为 25%。
- 如果寄生电容为 100 pF，则触摸引起的电容增幅仅为 5%。因此，电容的变化就更难测量了。

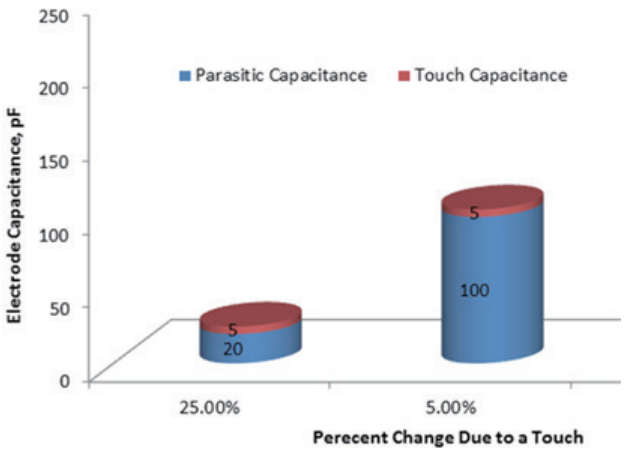


图 3：寄生电容

4.1.2 寄生效应和电场

寄生电容的影响并不限于只是导致电容变化百分比的下降。从物理学的角度来看，在靠近电极的地方设有接地结构会致使从电极投射出的电场线聚集在电极和接地之间，而不是通过覆盖物向上渗透至互动区域中。

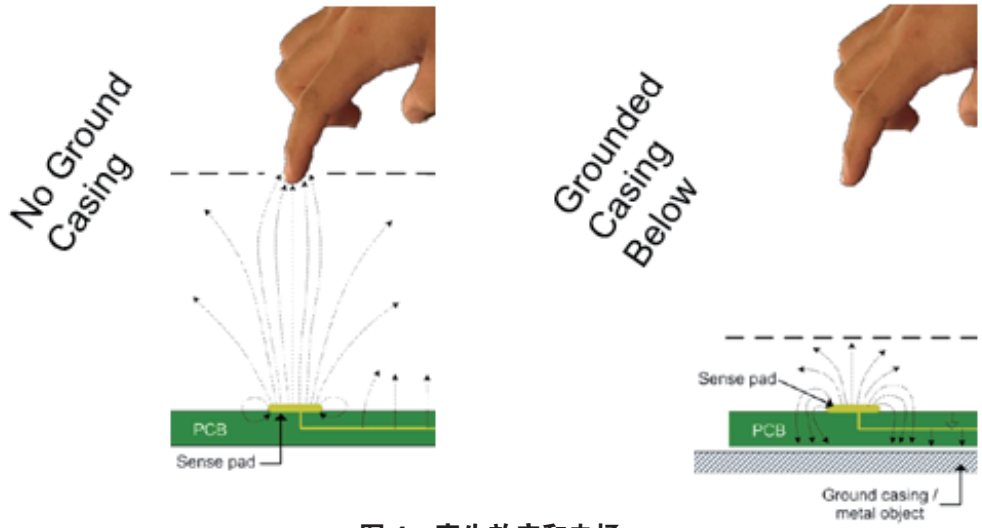


图 4：寄生效应和电场

4.1.3 电场传播

自电容电极以与电极成 360° 的角度把电场线投射出去。这意味着可以从 PCB 的底部并通过覆盖物材料来与电极实现互动。接地可用作一种屏蔽机制，如同由电压跟随器驱动的屏蔽所能做到的那样。

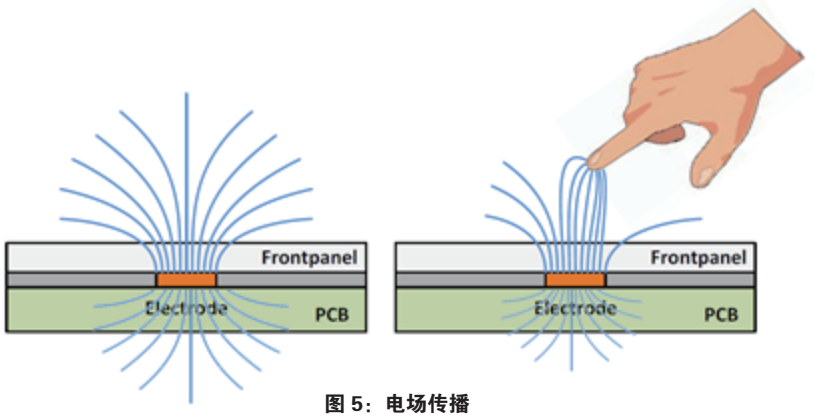


图 5：电场传播

另外，在自电容系统中投射的 360° 球体还对按钮的安置间距能够靠近到什么程度施加了限制。假如按键彼此靠得太近，那么用户很容易会错误地触发相邻的按键。对于按键相互之间能够挨得多近的限制常常取决于按键的大小和覆盖物材料的厚度。

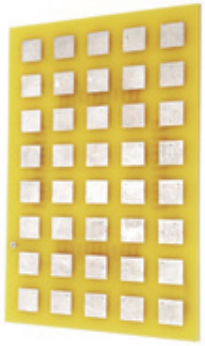


图 6：按钮密度

4.1.4 自电容总结

- 自电容是一种电容式感测测量方法，其测量了单个电极相对于大地接地的电容。
- 典型的用户触摸给电极增加了 1 pF 至 10 pF 的电容。
- 至接地的寄生电容有着降低对用户触摸的感受灵敏度的作用。
- 自电容电极以 360° 的角度向外投射电场线，并能在 PCB 的正反两面上与之实现互动（除非采用了接地屏蔽）。

4.2 互电容

就像自电容一样，互电容需要测量电容的变化，两者有一个重大的差异：互电容规定了电容器的两个极板，而不是像在自电容中那样使用大地接地作为第二个极板。

注：互电容有时被称为投射电容。

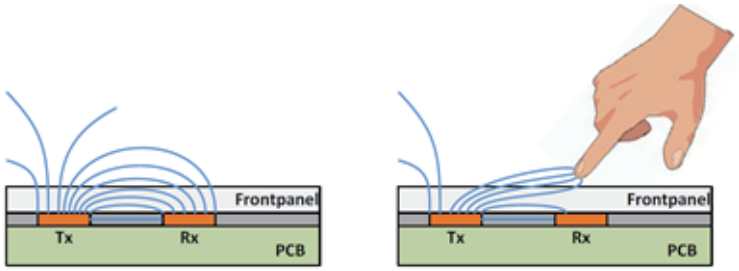


图 7：互电容

互电容电极实际上由两个分离的电极结构组成，因而需要从微控制器引出的两个引脚，一个发送电极和一个接收电极。当用户触摸了面板上的一个 Tx 与 Rx 相接触的区域时，这些 Tx 和 Rx 电极之间的互电容减小。这是因为用户互动扰乱了两个电极之间的电场传播。用户被耦合至大地接地，而且人体是一个导体。把一根手指放置在两个互电容电极之间与在这两者之间布设一个接地的作用大致相同；就是说，它减少了它们之间的电场耦合，从而减小了电容。由于触摸所引起的互电容的典型变化幅度很小，通常小于 1 pF。

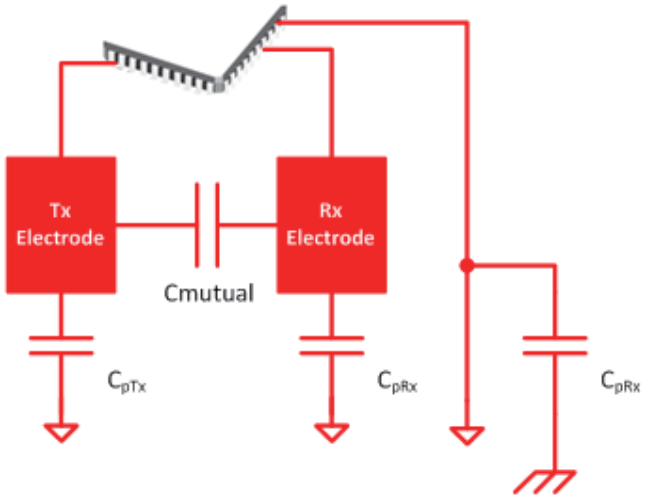


图 8：互电容示意图

4.2.1 互电容寄生效应

就像在自电容系统中一样，在互电容系统中也存在寄生电容。现在有两类寄生电容，即互电容和接地电容。

- 寄生互电容。在一根 Tx 走线靠近一根 Rx 走线的任何地方，这两根走线具有一个寄生互电容。这类似于自电容系统中的寄生接地电容。
- 至地的寄生电容（CpTx 或 CpRx）。在一个互电容系统中，Tx 和 Rx 电极具有至电路接地和大地接地的电容。与自电容的不同之处是现在寄生电容不影响测量。然而，它的确仍然影响着以高速度驱动电极的能力，因为该电容被充电和放电。

4.2.2 紧密耦合电场

由于互电容电极由两个导体（电容器的两个极板）限定，因此用户能够与之互动的电场被严格地限定在这两个导体之间。这是与自电容的一个显著差异，在后者中，互动场向外投射的方向是四面八方的。这会拥有许多的设计优势。例如，小键盘可以具有连成一片的密集按键，而不用担心当用户未直接按准某个键的中心时发生交叉耦合。另外，在按键之间排布保护槽和接近传感器也更加容易。

4.2.3 覆盖材料要求

覆盖材料是互电容触控面板所需要的。它为 Rx 和 Tx 电极之间的电场提供了一个传播区域，用户能够在这里与其实现互动。如果没有覆盖物，那么接触互电容电极将把 Rx 和 Tx 短接在一起，而且还增加了显著的接地寄生电容。

由于覆盖物起着向外投射交互电场的作用，因此互电容电极应根据覆盖材料的厚度来成形以优化灵敏度，这一点也是很重要的。如果 Rx 和 Tx 电极在一款采用厚覆盖物的设计中彼此靠得太近，则向上渗透至覆盖物顶部的电场微乎其微。

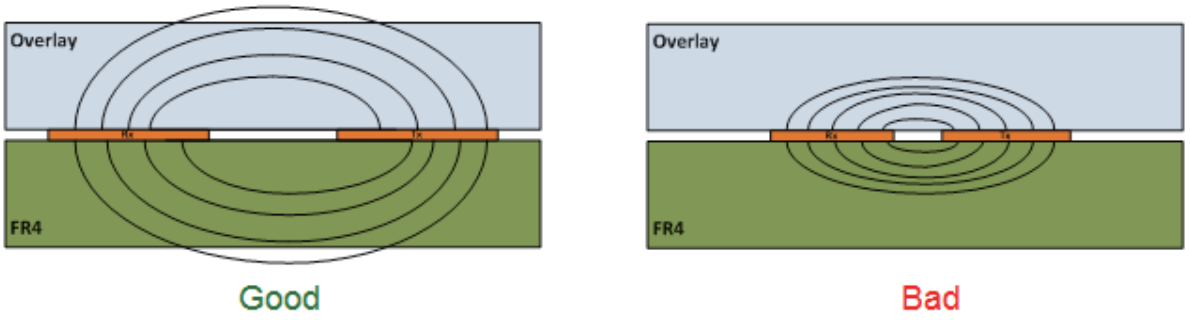


图 9：覆盖材料的厚度

4.2.4 互电容灵敏度

由于因触摸互电容电极引起的电容变化小于触摸自电容电极之时，所以测量较之自电容测量往往更加嘈杂。利用互电容电极难以构建大的滑块和滚轮，除非使用很多电极。因此，互电容电极最适合按钮矩阵和节点尺寸小但电极密度高的更先进传感器。

在互电容系统中电容是消减的。这意味着系统在起动时必须具有某个互电容。不过，需要互电容的是一个特定的位置：用户互动区域。

4.2.5 互电容总结

- 互电容是一种电容式感测方法，其测量两个电极之间电容的变化。
- 用户的触摸扰乱了两个电极之间的电场，因而减少了它们之间的耦合，并移除了互电容。
- 一次典型的用户触摸从电极移除的互电容小于 1 pF。
- 至接地的寄生电容虽不影响测量，但确实会影响系统驱动电极的能力。
- 互电容电场在 Tx 和 Rx 电极之间紧密耦合，因而可实现高密度触控面板。

4.3 真正的矩阵能力

一种采用少量的引脚来创建一个大的小键盘的方法是通过使用一个电极矩阵。矩阵也可以利用自电容来实现。图 10 示出了一个电容式触摸矩阵，其由 4 个行电极和 3 个列电极组成。这允许利用 7 个引脚导出 12 个按键。在一个自电容系统中，每行和每列作为一个独立的电极来扫描。个别按键通过聚合来自行电极和列电极的数据进行推导，以确定触摸的位置。自电容矩阵的缺陷是由于幻影效应而无法实现多点触控。

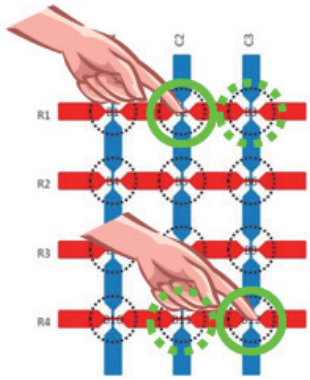


图 10：矩阵能力

在互电容方法中，列可被视作 Tx 电极，而行则可看作是 Rx 电极。这样，每个行 - 列交叉点就是一个独特的 Rx-Tx 组合和一个唯一的互电容。由于每个节点是一个处于测量之中的唯一电容，因此可实现全面的多点触控。

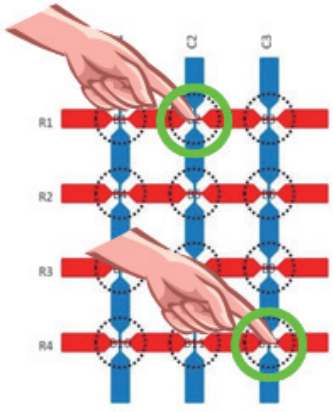


图 11：多点触控

因为每个 Rx 和 Tx 交叉点是一个唯一节点，所以一个 16 引脚器件能够支持 8 x 8 矩阵配置中的多达 64 个个别按钮。在面板上必须排布的引脚仅为 16 个，这在一块两层 PCB 上是非常容易实现的。等效的自电容设计实现相同的面板则将需要 64 个引脚。另外，拥有与众不同的节点还改善了系统的可靠性，因为每个节点（Rx 和 Tx 交叉点）是利用软件分开跟踪的（这与自电容方法所要求的聚合行和列测量数据以计算可能的按键操作是完全不同）。

5 技术介绍

关于 CapTIvate™ 外设的介绍，请参照英文最新版 CapTIvate™ Technology Guide 中相关章节的介绍。对于 CapTIvate™ Technology Guide v1.40.00.00，请参照章节“Technology”。

6 设计指南

6.1 引言

电容式触摸检测在设计的时候经常更多的考虑结构的美观。这常常导致在实现最优性能之前需进行多次修改设计。因此，必须了解用于电路布局的良好设计惯例和材料使用方面的原则，以最大限度地减少修改的次数。

优良的传感器设计是造就成功触控产品的基础。本设计指南旨在为电容式触摸传感器的设计和布局提供指导，以使其能够实现性能的最大化。通过在硬件中实现最高性能，CapTIvate 电容式触摸软件库能够执行电容式触摸测量，同时拥有极低的功耗。调试指南以及 CapTIvate Design Center 用于帮助调整电容式触摸应用的性能。

本指南中所提供的内容从产品设计，原理图到机械结构再到硬件布局。原理图和机械结构的设计将会影响 PCB 的布局。本指南首先说明了在产品的原理图和机械结构设计中要注意的事项。这些事项对布局有影响且必须首先予以考虑。在弄清了产品的要求之后，就能开始 PCB 布局工作了。

6.2 电容式触摸感测的概述

本节介绍了在电容式触摸检测中使用的品质因数。这些参数用于评估性能，并可作为一些设计的规范或注意事项。

如需了解自电容和互电容触摸检测理论中的背景，请参阅第 4 节。

6.2.1 信号和噪声

电容式触摸检测是一种模数转换器 (ADC)，具体而言是电容至数字转换器。和大多数 ADC 一样，重要的参数是分辨率、信噪比（SNR）和线性度（在滚轮和滑条的应用中）。在本文中，设计指导有助于实现信号的最大化和噪声的最小化，并在这两个目标有矛盾时提出相应的解决方案。

6.2.2 信号

如上所述，电容式触摸感测的基础是测量电容变化的能力。电容的变化是电容式触摸解决方案所识别的信号。术语“灵敏度”常用来描述信号强度，更灵敏的解决方案具有更强的信号。

灵敏度是用“电容 / 计数”来衡量的。在电容式触摸检测的背景下，由一次触摸所引起的电容变化幅度大约在几 pF 或几百 fF。对于一款解决方案来说，通过 300 次的计数完成 1pF 的电容变化检测很正常，这时候的灵敏度可能为 3.3 fF/ 计数，但是如果加上噪声的干扰，灵敏度就不是这样了。（见 6.2.6 节）。

灵敏度可以通过软件来设置，但是设计的时候应该首先利用硬件来提供优良的灵敏度，从而能在软件中采用最低的灵敏度设置，这样可以提供功耗最低的解决方案。请参阅用于自电容式传感器的等效电路（见图 2），必须最大限度地增加 C_touch，而其他四种电容则必须尽量地减小。走线和电极上形成的电容被近似为平行板电容。公式（1）说明了 PCB 布线时各种电容的形成。

$$C = Er \times E0 \times A/d \quad (1)$$

在本文中，我们从对触摸电容的正向影响来阐述介电常数 (Er)、面积 (A) 和距离 (d)。

6.2.3 寄生电容

虽然寄生电容是独立存在的，但它是灵敏度和信号的一部分。正如已经提到的那样，我们关注的是电容中的相对变化。电容变化基于触摸互动，不过该变化和系统的寄生电容相关。寄生电容也被称为稳态电容或基值电容。如果引起的电容变化为 100 fF，则灵敏度（这里指电容中的相对变化）可通过减小寄生电容来提高。

电容 C_trace、C_electrode 和 C_parasitics 一般都被称为寄生电容。

6.2.4 分辨率

术语“分辨率”在电容触摸的很多应用中都会使用到。在广义上说，分辨率也是灵敏度并可以用电容来定义。如果一种测量系统可以分辨电容的变化（0.1 pF 步进），它的分辨率为 0.1 pF。

在应用中，术语“分辨率”可以指滚轮、滑块和接近感应的距离。在这些应用中，目标是分辨在一个滑块或接近感应电极上的距离的变化。电容与距离成反比，但是在滑块、滚轮和接近传感器的环境中，分辨率用距离来表达比用电容表达更加清楚。

分辨率直接受限于灵敏度。硬件设计必须提供最大的灵敏度值，剩下的灵敏度提升就要靠软件算法来实现（通常以功耗增加为代价）。

6.2.5 线性

术语“线性”常用来描述电容式触摸滚轮和滑块的性能。与 ADC 或数模转换器 (DAC) 的线性性能相似，线性指的是检测出的位置与实际位置的匹配程度。由于电极的设计和形状之故，可能会存在过于平缓或者“死区”的现象，导致滚轮或滑块有不好的线性度。

线性与灵敏度有关。通过增加灵敏度，可减少或消除平缓区或死区，但是这通常是靠修改电极的形状来完成。

6.2.6 噪声

信号是在测量过程中检测到的电容变化。而噪声则是一种干扰，它没有改变电容但是影响了检测的数值。

6.2.7 检测距离

检测距离适用于接近传感器。范围或距离与灵敏度成正比。硬件设计的灵敏度越高，利用接近传感器可实现的检测距离或范围越大。

6.3 原理图

传感器布局的设计从了解传感器和 MCU 之间的连接开始。为了帮助实现该过程的自动化，CapTIvate Design Center 能够根据传感器的数量和类型自动地确定 MCU 和传感器之间的最优引脚配置。或者，对于那些有特定布线要求的应用，也可以手动的方式进行连接的配置。当配置完成时，可把引脚和传感器连接信息导出至一个文件，作为设计 PCB 过程中的一种辅助手段。

在开发过程的原理图阶段，有两个要素需要注意，与组件相关联的噪声和寄生电容。

6.3.1 器件

由于任何外置器件都会带来额外的寄生电容，外部器件在电容式触摸应用中应该尽量避免使用。于是，被选为电容式触摸电路一部分的器件应当具有尽可能小的占板面积。电容与面积成正比，因此占板面积的任何缩减都将减小电容。通常，这些外部器件与 ESD 保护有关，例如：布设在 MCU 和电极之间的限流电阻。另外，最大限度缩减占板面积的规则还适用于电容式触摸电路中的连接器或任何其他器件。

通常来说，随着温度的变化和时间的推移，任何器件都具有一个非常低的漂移因数。虽然这影响不大（因为软件具备漂移补偿功能），但最好还是使触控电路及所有相关器件随着时间的推移和温度的变化尽可能地保持稳定。

6.3.2 ESD 保护

ESD 保护的设计应尽量考虑在可能的 ESD 冲击和 PCB 之间添加充当隔离的层压材料。电荷会遍布于层压材料并找到一个释放点。要小心有时候静电不会穿透隔离材料但是会绕过它而打到 PCB 上。依靠覆盖物的高电介质击穿电压比那种给触控解决方案添加 ESD 组件的做法更为可取，因为这些 ESD 保护器件给电路增添了寄生电容并降低了灵敏度。

如果需要额外的 ESD 保护，TI 建议采用小的限流电阻器和低电容箝位。

6.3.3 电源

降低电源噪声最常用的方法是使用滤波器。π 型滤波器通常是首选的滤波器。另外，也可以采用低压降稳压器 (LDO) 来降低电源噪声（相比于开关电源）。

6.4 机械结构

机械结构是产品设计的机械特性。机械结构包括覆盖物材料、覆盖物顶部油墨、任何用于把电极结合至覆盖物或外壳的粘合剂、以及任何用来消减电极与覆盖物之间的空气间隙的过渡材料。另外，机械结构还包括用于电极的材料的类型。机械结构对信号和寄生电容均有影响。

本节的目的包括三个部分：

- 了解兼顾美学和坚固性的好处；
- 了解电极上的材料以及电极材料本身是如何影响电极布局的；
- 避免在机械结构中出现不利于电性能的错误。

6.4.1 典型叠层

图 12 示出了电容式触摸解决方案的一种典型叠层。该叠层的一个主要目的之一是减小（或者消除，如果可能的话）电极与触摸发生区域之间的任何低介电常数（空气）间隙。由叠层形成的电容对信号（由触摸引发的电容变化）具有非常强的影响。信号与材料的介电常数成正比。如果可行的话，应采用高介电材料，最重要的是，在设计中要避免任何空气间隙。

此外，空气间隙还会包含湿气，当温度变化以及空气间隙扩张和收缩时，它会影响性能乃至损坏叠层。

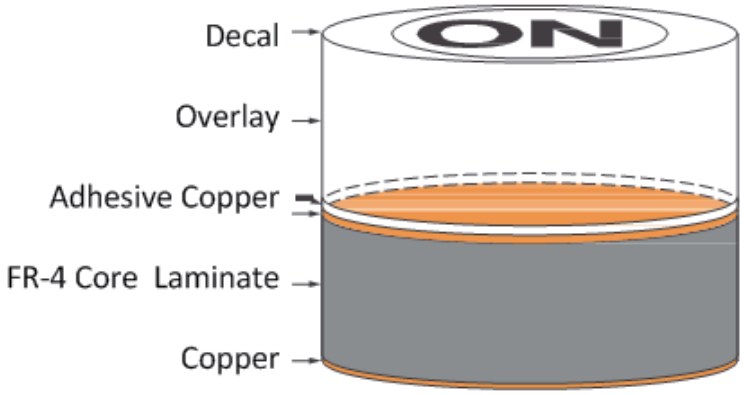


图 12：典型的材料叠层

叠层的另一种至关重要的属性是其应为非导电性的。对于覆盖物材料而言这通常并不是问题，但是在选择粘合剂、标签或油墨时则会被忽视。面向电容式触摸解决方案的常用粘合剂包括由 3M™ 提供的 200MP 产品，例如 467MP 和 468MP。

6.4.2 覆盖物

叠层的电容是所有材料的叠加，但是通常覆盖物的材料起决定因素。所用覆盖物材料的类型和尺寸由产品的外观设计以及外壳强度决定。一项常见的要求是坚固（抗擦伤和刺穿）而量轻。

图 13 示出了覆盖物的厚度和电路灵敏度之间的关系。从平行板电容方程可知，电容与材料的厚度成反比 (C ≈ 1/d)。

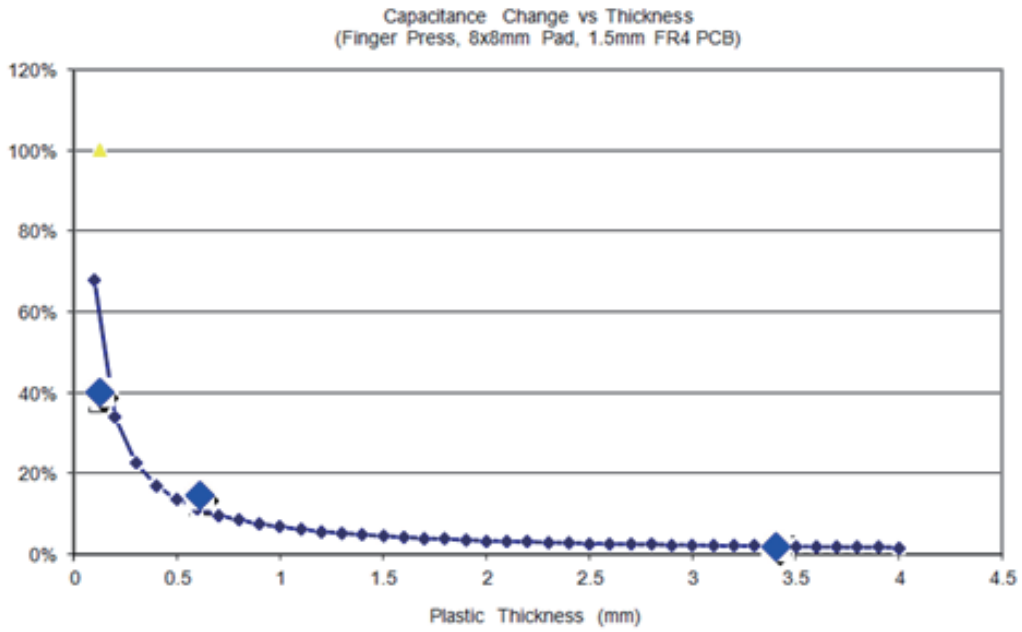


图 13：灵敏度与厚度的关系

材料的厚度和介电常数影响着电极的设计。电极是手指或手掌操作的区域，而间距（电极与相邻的电极或铺地）则与覆盖物的厚度有关。例如，当采用一个具有介电常数 = 3 的 2 mm 覆盖物时，间距应为约 1 mm（厚度的一半）。当采用一种较高介电常数的材料时（比如：Er = 6），在同样间距为 1mm，并保持同样触摸效果的情况下，覆盖物的厚度则可以加倍。表 1 列出了用作覆盖物的各种不同材料的介电常数值。

表 1：材料介电常数和击穿电压

材料	介电常数 (Er) (1)	击穿电压 (V/mm)
空气	1.0	3300 (STP)
FR-4	4.8	20000
玻璃	7.6 至 8.0	7900
强化玻璃	7.2 至 7.6	见 (2)
聚碳酸酯	2.9 至 3.0	16000
丙烯酸	2.8	13000
ABS	2.4 至 4.1	16000

(1) 相对介电常数
(2) http://www.corninggorillaglass.com

另外，表 1 还描述了不同覆盖材料的击穿电压。这一点应在针对 ESD 保护进行设计时予以考虑。ESD 解决方案是一个系统解决方案，因此除了考虑覆盖物，其他附加组件也应尽量考虑 ESD 保护性能，作为补充。

6.4.3 电极和走线材料

电极和介于电极和微控制器之间的走线所用的导电材料对性能有所影响。大多数的应用在 PCB 上使用铜，而铜具有 1.7 x 10-6 Ω-cm 的电阻率 (3)。当导体的电阻率增加时，把电荷移动至电极或从电极移出的能力下降。这与寄生电容增加所产生的影响是相同的。电阻率的这种增加（类似于寄生电容增加）降低了系统灵敏度。表 2 列出了触摸应用中常用材料的电阻率。

(3) 电阻率的单位为 Ω-cm，于是电阻等于电阻率乘以长度并除以横截面积：R = p x L / A。

表 2：材料的电阻率

材料	电阻率，p (Ω-cm)
铜	1.68×10^{-6}
银	1.59×10^{-6}
锡	1.09×10^{-5}
铟锡氧化物	1.05×10^{-3} (1)

(1) 该电阻率是针对 270 nm 的薄膜厚度。通常，对于 ITO，供应商提供的是薄膜电阻而不是电阻率，大约为每平方 10 Ω 至 100 Ω。

当采用高电阻率材料时，建议通常是增加走线的面积以减小电阻（以牺牲电容为代价）。ITO 解决方案提供的灵敏度较低，在电容测量算法中必须对此予以补偿（通过延长测量时间来实现）。

6.4.4 其他场合

在产品设计中往往会有一些特殊的应用，而本节举例说明了两种特殊的场合。第一种是有意地使电极与覆盖材料之间的空气间隙大于 2 mm，第二种则是戴着手套进行触摸操作。

6.4.4.1 间隙

在有些应用中，元器件与电极位于同一个电路层上。这造成覆盖物无法直接贴合在电极上。当一个 LED 被安装在电极附近时即为这种情况的一个常见的例子（见图 14）。另一种场景是当覆盖材料不是一个均匀的表面时，因此电极无法直接接触覆盖物。

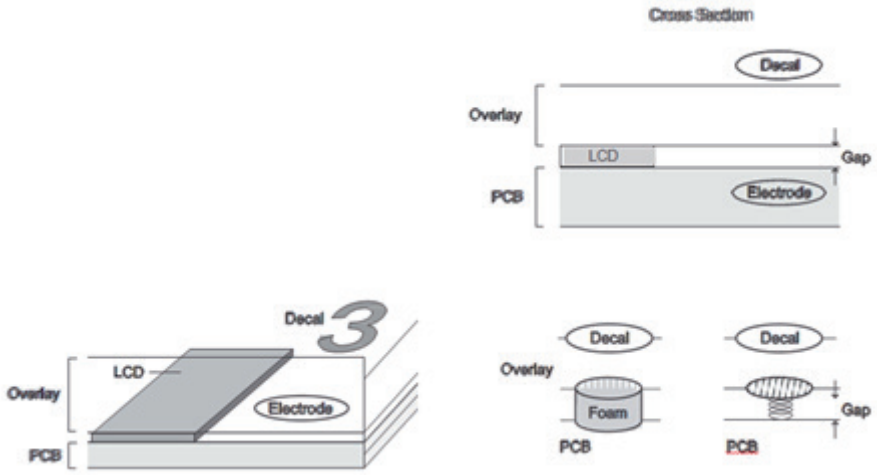


图 14：有意设置在电极与覆盖物之间的间隙

在任一种场合中，必须利用一种非导电填料（通常为粘合剂）或导电延伸物对间隙进行填充或桥接。当间隙超过 2 mm 时，应使用一种导电延伸物（泡沫或金属）。金属或泡沫必须具有延展性，以与表面的形状相符合并防止形成间隙。如图 14 所示，由泡沫或金属与覆盖物接触的部分现在就形成了待检测的触摸电极。

6.4.4.2 手套

手套就是介于电极和手指之间的另一个介质层，适用于相同的厚度和介电常数原理。使用手套进行触摸的应用难点在于如何同时支持戴手套和不戴手套的操作，以及如何适应各种不同材料的手套。典型的皮手套或塑料手套具有 2 至 4 的介电常数，而纺织手套和绝缘手套可具有小于 2 的介电常数。

6.5 常见的布局考虑因素

在了解了机械结构之后，就可以设计电极以提供最多的信号。PCB 布局设计与机械结构关系不大，对其有影响的因素包括微控制器和电极之间的距离、PCB 层数（例如：一层、两层或四层）和 PCB 上的其他电路。

首先需要考虑的是原理图和与触摸相关的元器件的布局。典型例子是 ESD 保护器件，如串联限流电阻器。在所有的场合中，都应使器件尽可能地靠近微控制器。当把器件移至远离微控制器时，可能会带来面积增大、噪声增加、引入 ESD 等风险。

6.5.1 布线

走线的寄生电容由几个主要和次要的因素组成。简单来说，C_trace 是在电极走线和底层的地之间，以及电极走线和周围的铺地之间形成的电容，它是寄生电容的主要来源。。典型双层 PCB 的顶视图和截面图示于图 15。电容 C_trace 由走线宽度 (W)、电介质厚度 (H)、走线粗度 (T) 和 PCB 材料的相对介电常数 (Er) 决定。

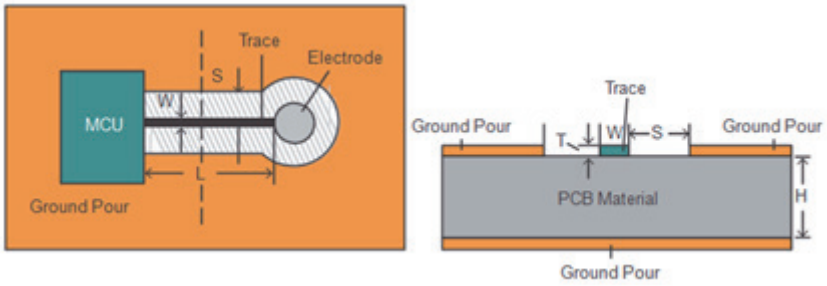


图 15：PCB 中走线的顶视图和截面图

使用尽量短的走线是因为走线上电容的形成和长度有关。增加走线的长度将增大与走线相关联的寄生电容。另外，增加走线的长度还会增加对噪声的敏感性。因此，微控制器与电极之间的走线排布尽可能地短。虽然很多情况下无法缩短走线长度，但是了解寄生电容与走线长度的关系还是很重要的。

6.5.1.1 单位长度走线的电容

应使单位长度走线的电容尽可能地小，以最大限度地减小寄生电容 (C_trace) 并最终实现灵敏度的最大化。如前文所述，C_trace 中的主导电容是位于走线和周围接地覆铜之间的平行板电容，这需要依靠 PCB 制造工艺来减小。较严格的容错和较小的最小尺寸（走线宽度和间隔）使得 PCB 布线可以采用较细的走线和较大的间距，从而产生较低的单位长度电容。此类制造工艺通常以成本升高为代价。

表 3 示出了单位长度的电容是如何随着不同尺寸的变化而改变的。这些数值取自《Coplanar Waveguide Analysis/Synthesis Calculators》中的“参考文献”部分。

表 3：单位长度电容的计算结果

W (mm)	S (mm)	T (mm)	H (mm)	Er	C (pF/cm)
0.152	0.152	0.036	1.6	4.6	0.633
0.152	0.254	0.036	1.6	4.6	0.555
0.152	0.381	0.036	1.6	4.6	0.496
0.203	0.152	0.036	1.6	4.6	0.692
0.203	0.254	0.036	1.6	4.6	0.602
0.203	0.381	0.036	1.6	4.6	0.543
0.254	0.152	0.036	1.6	4.6	0.740
0.254	0.254	0.036	1.6	4.6	0.641
0.254	0.381	0.036	1.6	4.6	0.578
0.254	0.152	0.036	2.54	4.6	0.736
0.254	0.254	0.036	2.54	4.6	0.637
0.254	0.381	0.036	2.54	4.6	0.566

表 3 显示：增加走线和地之间的间距 S 是一种减小寄生电容的有效方法。然而，增加间隔会带来负面影响，必须对此加以了解。一种影响就是增加板级空间。增加尺寸会导致 PCB 变大和成本升高。另一种影响则与噪声有关。较大的间隔使得走线对于触摸事件（触摸走线而不是电极）更加敏感，而且更容易遭受辐射干扰的影响。

在设计的时候，可以选择一个平衡的 S 值，这里通常建议取值为 1/8 的覆盖物厚度。此外，在靠近走线的地方常常采用网格地（而不是实心地）以缩减耦合面积，从而达到减小寄生电容的目的。

表 3 还显示：当 H 变小时，寄生电容增加，这导致灵敏度下降。在大多数应用中，采用的是双层 PCB，建议使用厚度为 1 mm 至 1.6 mm 的标准 FR4 PCB。如果采用多层电路板，建议使 H 尽可能地大。对于复杂的多层电路板（超过 6 层），要小心内层由于缺少铜会带来一些问题，而且顶层与底部之间的高度 H 可能比预期的小，这一点很重要。

图 16 示出了一块 4 层 PCB， 在空间不够的时候经常使用多层 PCB。为了减小寄生电容，地层通常放在走线下方的最底层，这样可以使 H 最大。如果结构受限，无法再加大 H，则可以使用较窄的 W、较大的 S、甚至改铺网格地来减小寄生电容。

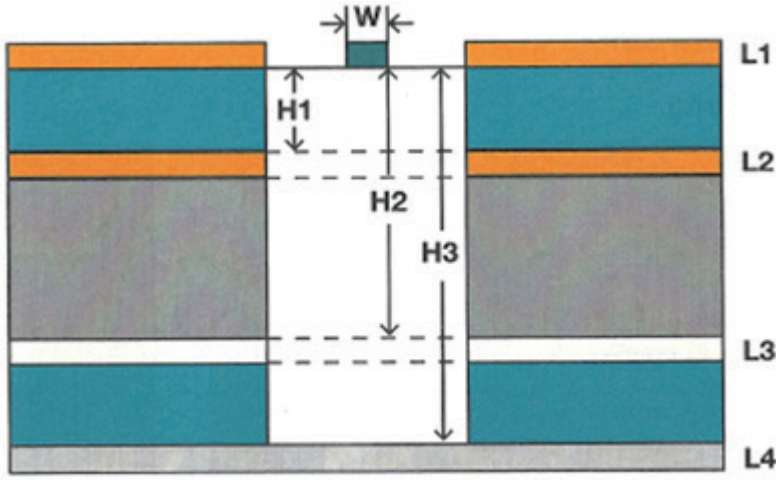


图 16：在多层 PCB 中下方没有覆铜的走线

如何通过单位长度的电容来计算允许的最长走线长度呢？下面的例子说明了如何计算单位长度电容为 0.58 pF/cm 的走线允许的最大走线长度

假设由一次触摸引起的电容 C_touch 约为 1 pF。为了使由一根手指感应的电容很大（相对于寄生电容），总寄生电容（C_parasitics、C_trace 和 C_electrode）应在 10 pF 至 20 pF 的范围内（触摸引起的变化率至少为 5%）。

假定 C_electrode 约为 3 pF，C_parasitics 为 5 pF，而且单位长度的电容为 0.58 pF/cm，则走线的长度 L 应不大于 210 mm。

如果电极较大（对于接近应用），电容本身较大（对于本例假设为 8 pF 左右），那么最大走线长度 L 缩减至 120 mm。

一般来说，在 PCB 工艺允许的情况下走线宽度 W 应尽可能地细，因为简短和纤细的走线更合乎需要。另外，走线的粗度 T 和材料的相对介电常数 Er 也对单位长度的电容具有显著的影响，但是它们是由 PCB 制造工艺决定的，并且常常难以改变。

6.5.1.2 连接器

在一些产品设计中电极可能处于电路板以外，因而采用一个连接器把走线从 PCB 连接到导线或者另一块 PCB 上。这种情况下不建议使用连接器，因为如同任何器件一样，连接器 PCB 焊脚和结构会产生寄生电容。

就寄生电容和灵敏度而言，连接器被视为一种降低触摸灵敏度的寄生电容。由于连接器等效于一个独立的寄生电容，因此连接器在 PCB 上的位置与灵敏度无关，但是和抗噪声能力却有很大关系。

图 17 显示：如果干扰源（噪声源，N1）位于 PCB 上，则连接器的推荐位置是放在靠近 MSP430 微控制器的地方。连接器引起的寄生电容可去除高频噪声 (Z 1/jwC) 并提高电路的抗噪声能力。相反，假如干扰源位于电路板之外（噪声源，N2），则应把连接器安置在远离微控制器但比较靠近电极的地方。这种布置方案可最大限度地缩减板外走线和排线长度（其作用类似于一个天线）。当如图 17 所示存在多个干扰源时，建议优先考虑抑制那些和电容检测频率最接近的干扰源。

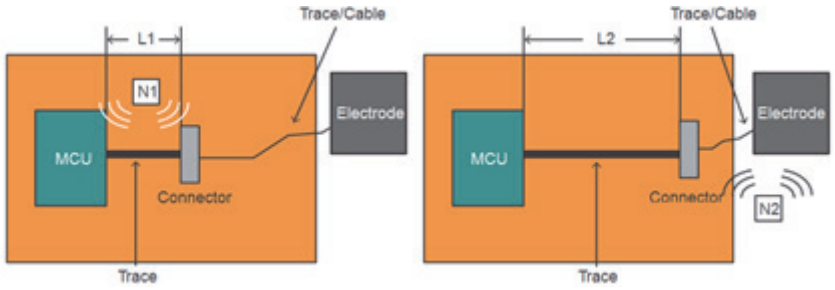


图 17：连接器和噪声源的顶视图

当决定连接器的位置时，设计人员必须平衡走线和排线的长度，同时考虑诸如背景噪声等参数。

6.5.1.3 走线材料

走线通常是用铜来完成的，但也可采用银和 ITO 等其他材料。银与铜相似，而且性能相近（在材料的厚度相同的条件下）。ITO 与铜差别比较大，而且电阻率的差异降低了触摸的灵敏度。减低走线的阻抗（通过增加宽度来实现）在设计中应具有高优先级，尽管这是以增加寄生电容和降低抗噪声能力为代价的。最后，使用像 ITO 这样的材料需要 MSP430 微控制器执行更多的处理，以针对灵敏度的下降和噪声的增加进行相应的调节。

6.5.2 电极材料

如在 6.5.1 节中讨论的那样，在诸如 ITO 等具有高电阻性的材料中，导电性变成了一个问题。虽然 ITO 的透明度非常好，但是与银和铜等材料相比时其电阻率很高。通常，物理尺寸的限制使得无法增加 ITO 电极的面积，因此灵敏度的任何下降必须在软件中进行补偿。这一般会导致测量时间略有延长，因此功耗有所增加。

6.5.3 电极设计

电极设计必须实现两个目标。首先，电极必须提供充足的信号（随触摸而发生的电容变化）。电极向上和向外投射电场，在想要达到的检测距离上有足够的灵敏度。一旦了解了叠层、厚度和电介质，即可进行电极的设计以提供最强信号。其次，电极设计必需具有最小的寄生电容。

在下面的部分中讨论的是电极的形状和面积，目的是在针对不同的实施方案（按钮、滑块和滚轮）实现信号的最大化。用于控制寄生电容的基本原理适用于不同的传感器设计，下文针对不同的设计进行了讨论。

图 18 举例说明了 PCB 的横截面和影响寄生电容的重要参数。高度、宽度和间距对电极的寄生电容 (C_electrode) 具有直接的影响。在图 18 中没有标识出一些基本的参数，因为基本参数同时对触摸电容 (C_touch) 和寄生电容 (C_electrode) 产生影响。本节说明了怎样改变高度和间距能够最大限度地减小寄生电容。在后续章节还会描述如何改变面积来尽量地增大 C_touch。

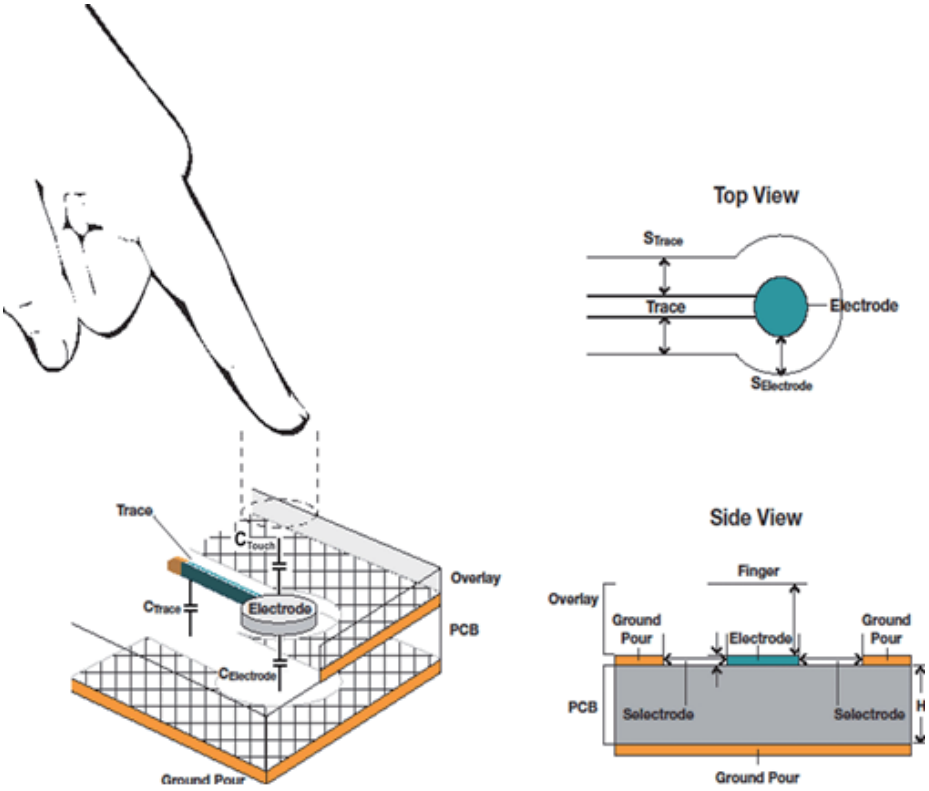


图 18：电极的电容

如 6.4 节中所述，间距 (S_electrode) 与覆盖物的厚度直接相关。而厚度和 PCB 的制作工艺有关，并且不是一个可以任意修改的参数。而间距通常是灵活性最大的参数。

表 4：基线电极电容

面积 (mm^2)	高度 (mm)	描述	间距 (mm)	电容 (pF)
10 x 10，FR-4 (Er = 4.4)	1.572	双层 PCB，L2	0.508	3.3
10 x 10，FR-4 (Er = 4.4)	1.572	双层 PCB，L2	1.02	3.2
10 x 10，FR-4 (Er = 4.4)	1.572	双层 PCB，L2	1.52	3.1
10 x 10，FR-4 (Er = 4.4)	1.30	四层 PCB，L2	0.508	3.8
10 x 10，FR-4 (Er = 4.4)	1.57	四层 PCB，L3	0.508	3.3

如表 4 所示，增加间距（电极与参考地之间的距离）将减小电容。通过减小寄生电容，由某个触摸事件引起的电容相对变化增大。例如，倘若由触摸引起的电容变化为 0.5 pF，则当基值电容为 11 pF（而非 12 pF）时，相对变化较大（5.5%，而不是 4.2%）。

6.5.4 形状

电极的电容和面积相关，所以电极的形状也很重要，因为形状会影响面积。设计电极形状的一个重要的细节是表面积不能太小。每个电极的面积必须提供最大的 C_touch，这将在出现触摸事件时产生最多的信号（电容的变化）。

6.5.5 串扰

在 6.5.1.2 节中，干扰源假定为一个点噪声源。在本节中，干扰源是十分靠近触摸按键走线的各种信号源。这些干扰源可以是另一个电容传感器走线或非电容传感器线路。非电容传感器线路包含数字信号、模拟信号和用于驱动 LED 的高电流信号。

6.5.5.1 邻近的电容式触摸信号

电容式传感器走线影响着相邻的电容式触摸传感器走线。电容传感器走线之间的间距 SCS 应保持在一个安全的距离（见图 19）。

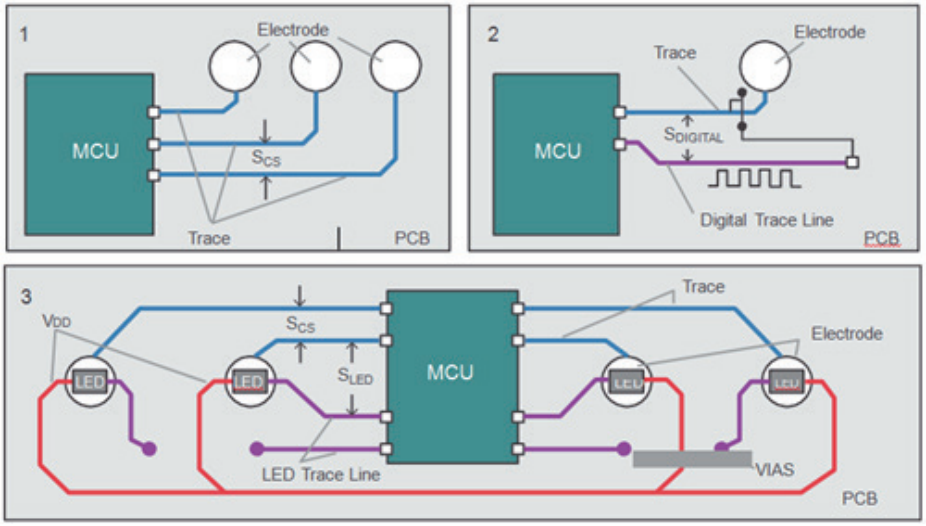


图 19：不同走线的顶视图

6.5.5.2 数字信号

数字信号通常是 PWM 信号或者诸如 I^2C 或 SPI 等通信信号。与电容式走线不同，这些信号的作用会像干扰源一样，而且在电容测量期间有可能正处于运行状态。建议使此类信号离开电容式触摸走线至少 4 mm。如果数字信号和电容式触摸走线必须交叉，则建议使交叉成 90° 角。

6.5.5.3 LED 背面照明

用于驱动 LED 的信号（除非 LED 需要大电流驱动器）与其他数字信号相似。因此，和数字信号一样，对于 SLED 强烈建议距离至少为 4 mm，如图 20 所示。

一般说来，不要使用高阻态来控制 LED。运用高阻抗状态以防止 LED 导电会导致通电状态下的电容与断电状态下的电容之间的显著差异。这种电容变化有可能被系统检测到并被看作是系统电容的变化，甚至更糟，发生误检。如果必须使用 LED 的高阻态来控制，那么建议在 LED 上并联一个分立的电容（通常使用 1nF 电容）。

- 绝对不要让 LED 呈高阻抗状态，因为当接通和关断时 LED 电容将发生变化。
- 增设一个小的旁路电容以控制 LED 电容的变化。
- 该电容的物理位置不需要靠近 LED，只要位于电路中即可。

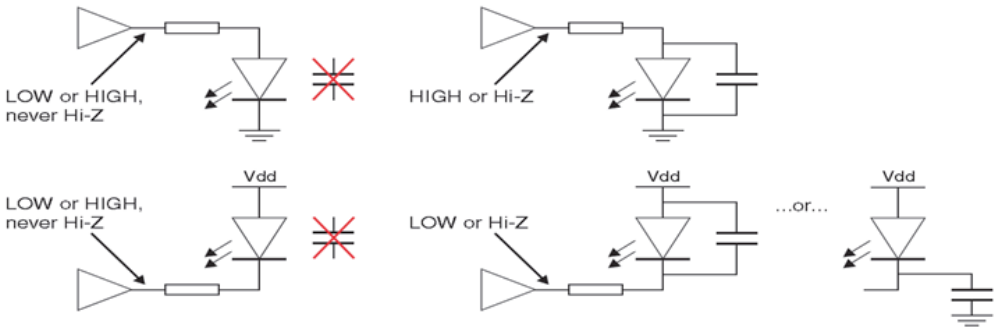


图 20：建议的 LED 驱动方法

LED 背面照明可在 PCB 的反面上使用一个反贴 LED 轻松地实现。

- 使孔洞尽可能地小，以消除传感器中可能的死点。
- LED 可能需要一个旁路电容（见上文）。

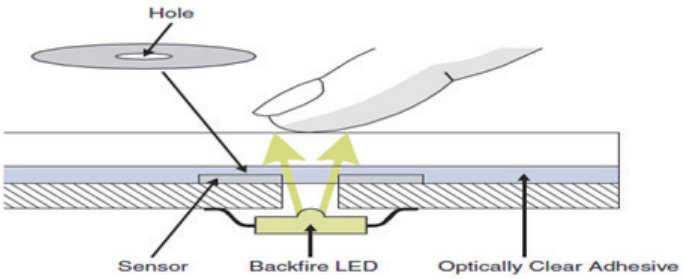


图 21：背面照明 LED

6.5.6 铺地

周围的铺地影响电极的灵敏度。当与地的间距减小和铺地的面积增大时，整个传感器的电容增加。（主要是寄生电容）。本节主要描述如何铺地以及覆铜的密度。

靠近电极和走线的铺地和覆铜必须连接至一个电位，而且不能悬空或置于某种高阻抗状态。如果悬空会使噪声容易耦合到电极上，因而强烈建议不要这么做。

6.5.6.1 间距

铺地（位于同一层的和在邻近 PCB 层上的）可降低噪声。这和 6.5.1 节（“布线”）中讨论的原理一致。布线的时候尽可能靠近地可以提高抗干扰能力，但也需保持足够远的距离（以最大限度地减小寄生电容）。保持多少间距与走线上方的材料厚度有关（包括面板和粘合剂）。如在 6.5.1 节中提到的那样，通常建议走线和接地之间的间隔为材料厚度的 1/8。电极和接地之间的间隔应至少为材料厚度的一半。

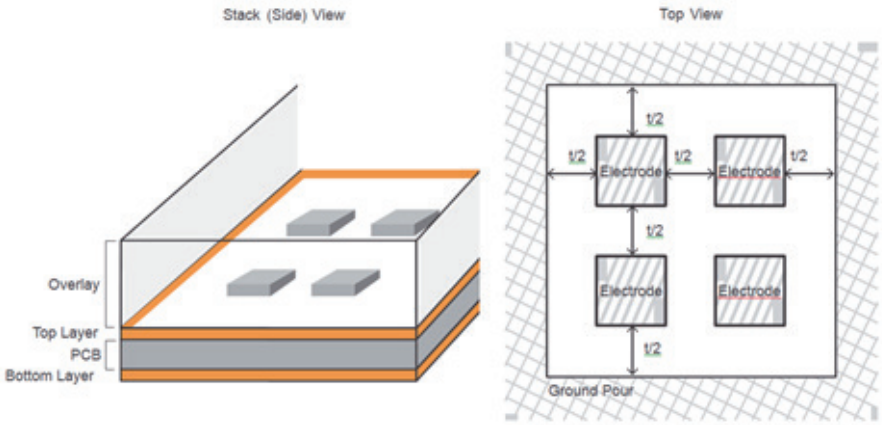


图 22：接地间隔

如图 22 所示，电极与地之间的间隔应至少为覆盖材料高度 t 的一半。不扫描的电极应保持在一个低逻辑电平（VSS 电位）且不能是悬空状态。这样，电极之间的间隔与接地和电极的间距相同。

6.5.6.2 覆铜

使用网格地（而不是实心接地覆铜）是一种不错的方法。这缩减了面积并因此减小了由 C_trace 和 C_electrode 引起的寄生电容。通常，25% 的网格地就够了，但可增大或减小该百分比以分别改善抗噪声能力或灵敏度。

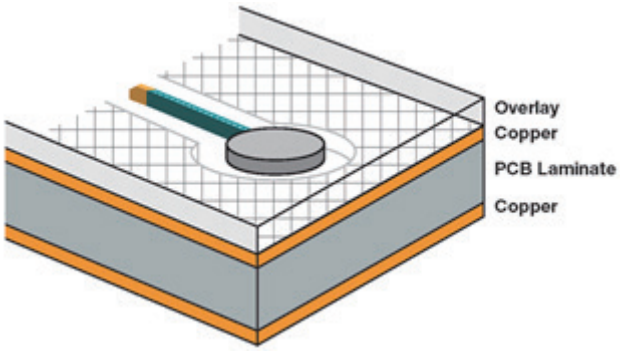


图 23：网格地示例

6.6 自电容传感器设计

如需了解自电容传感器的工作原理，请见第 4 节。

表 5：自电容传感器设计

参数	指导
电容耦合方式	位于电极和地之间
尺寸	等同于交互尺寸，比如手指大小
形状	各种各样：通常为圆形或方形
间距	覆盖物厚度的一半（最小值）

6.6.1 按键传感器

自电容式按键传感器是单个电极。

6.6.1.1 按键形状

对于按键而言，电极形状通常是圆形或矩形的。

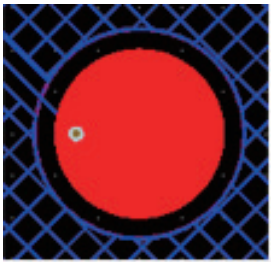


图 24：简单的按键设计

一个常见的错误是使电极的形状与用不导电的油墨印刷在覆盖物上的图标相同。如图 25 左侧所示，这会导致电极具有奇怪的形状，因而产生间断并缩减表面积。

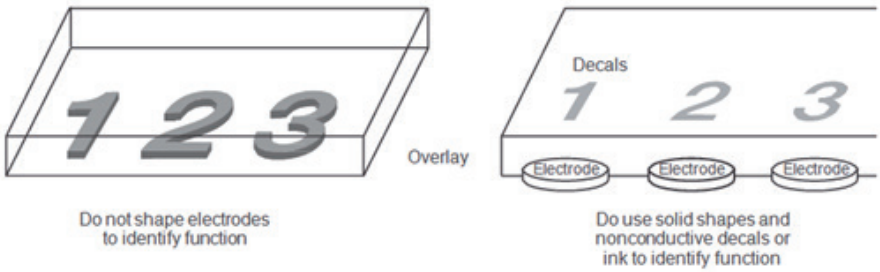


图 25：按键形状示例（注意事项）

6.6.1.2 按键感应区

按键感应区的设计目标是在用户触摸按钮电极上方的覆盖物时提供充足的信号。通常采用一种不导电的丝印或油墨来识别电极上方的触控感应区。丝印面积可以小于电极，这样触摸丝印外围也会被认为是有效触摸。相反，电极可以很小，以确保只有在触摸了丝印的中心时才会被识别为有效触摸。图 26 和图 27 示出了有效触控感应区与电极尺寸和手指大小之间具有怎样的关系。

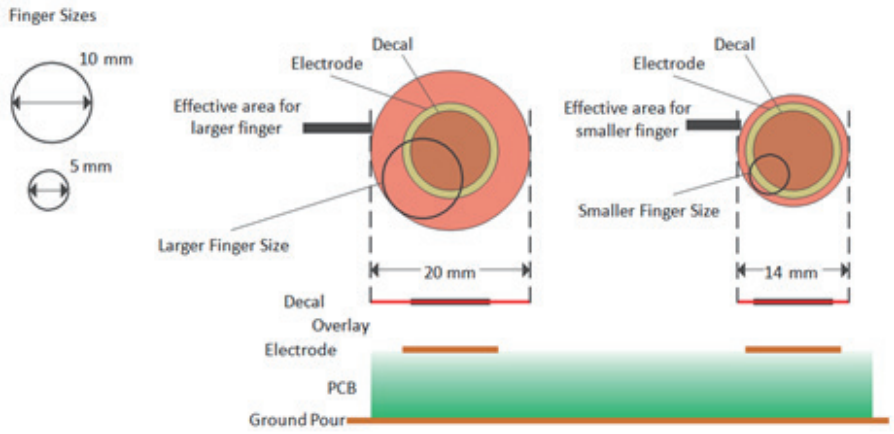


图 26：针对大于丝印的电极的有效感应区域示例

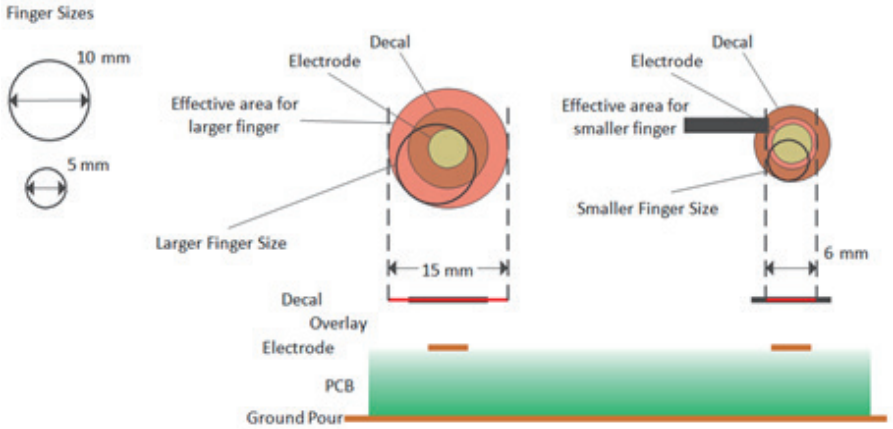


图 27：针对小于丝印的电极的有效感应区域示例

当覆盖物的间距增大时，有效触摸区域缩小。于是，应使按键电极的直径至少为面板厚度的三倍，这一点很重要。

6.6.2 滑条和滚轮传感器

一个典型的自电容式滑条传感器可由四个单独的电极组成，而一个滚轮传感器则可以由三个单独的电极组成。虽然可以具有更多的电极，但是 LAYOUT 变得更加困难，而且需要更多的 MCU RX 引脚。由于 MCU 支持一个周期内完成 4 个电极同时扫描，建议滑条和滚轮尽量用不超过 4 个电极进行设计，实现最高效，检测时间最短，功耗也最低。（就测量时间和功耗而言）。

6.6.2.1 滑条和滚轮形状

滑条和滚轮不同于按钮，因为触摸操作的时候涉及多个电极。如图 28 和图 29 所示，电极的面积变得不是最关键了，而是相邻电极之间的交叉区域如何设计更重要。

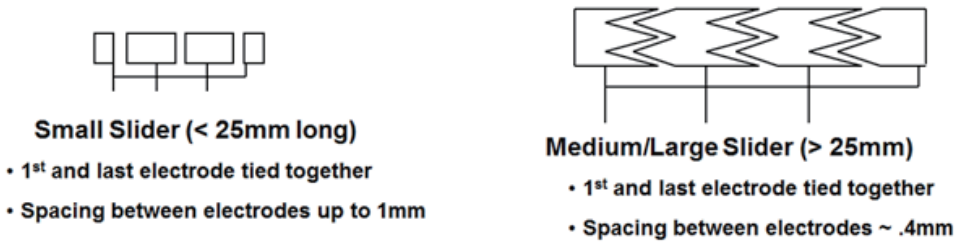


图 28：滑条设计

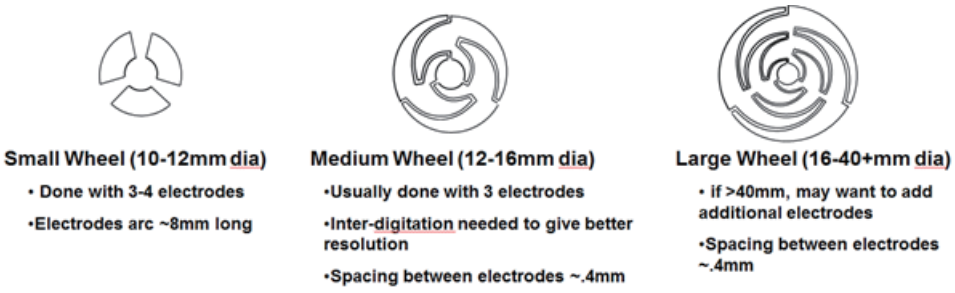


图 29：滚轮设计

6.6.3 接近传感器

6.6.3.1 接近传感器形状

接近传感器电极的形状有圆形或方形，这一点与按键相同。接近传感器需要高于按键传感器的灵敏度，而这种较高的灵敏度是通过增加感应面积以及拉大与其他导体的间距来实现的。

6.6.3.2 接近传感器感应面积

接近传感器通常具有较大的感应面积，从而可以从较远的距离检测大物体的靠近（手掌）。在大多数应用中电极的大小受限于最终产品的外形尺寸。提供足够的灵敏度需要较长的测量时间，因此会增加系统的功耗。

6.7 互电容传感器设计

如需了解互电容传感器的工作原理，请见第 4 节。

表 5：互电容传感器设计

参数	指导
电容耦合方式	位于 TX 和 RX 与地之间
尺寸	等同于交互尺寸，比如手指大小
形状	各种各样：建议为方形或带拐角的形状
间距	0.5 x 覆盖物厚度（最小值）

6.7.1 按键传感器

图 30 示出了一款简单的互电容按键设计。内部红色正方形为 RX 电极，外部红色正方形为 TX 电极，蓝色的电极走线在 PCB 底层。

图 30：常见的按键设计

- 简单的方形电极
  - 极易于 Layout
  - 极高的灵敏度
  - TX 在外侧，RX 在内侧
  - 对于单层设计，可在一侧打开 TX 并把 RX 走线从中穿过（断开的地方要尽量避免在拐角处）
  - 如果走线在双层 PCB 上则灵敏度更好

6.7.2 滑条传感器

图 31 示出了一个由四个电极组成的滑条。A 部分显示了滑条电极形状和走线。B 部分则突出显示了 TX（淡蓝色）和 RX（红色）电极。

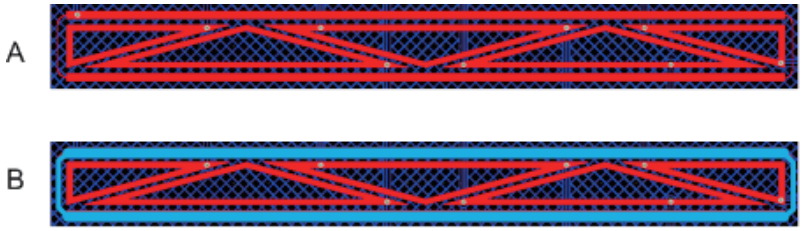


图 31：四电极滑条设计

- TX 在外侧（走线在顶层和底层）
- RX 在内侧（三角形状）
- 从 TX 到 RX 的间距为面板厚度的一半
- 第一个和最后一个 RX 三角形相互连接
- TX 和 RX 之间的间隔约为 0.5 mm

6.7.3 滚轮传感器

图 32 示出了一个由三个电极组成的滚轮设计。A 部分显示了滚轮电极形状和走线。B 部分则突出显示了 TX（淡蓝色）、RX（红色）和接地（绿色）电极。

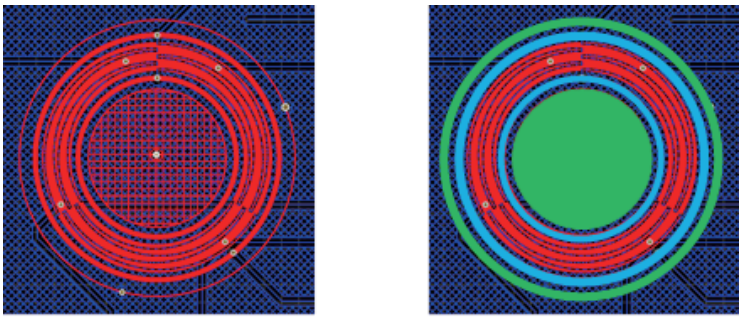


图 32：三电极滚轮设计

- 滚轮外侧和内侧上的圆圈是 TX（需要在电路板的反面把内侧和外侧连接在一起）
- 交叉锯齿状的是 RX（类似于自电容滚轮）
- TX 和 RX 之间的间距大约是面板厚度的一半
- 在滚轮内部增加的网格地 (ground hatch) 可以提高抗噪声能力。另外，在电路板的反面也可以铺网格地（注意总的对地电容不能太大）。
- 可在中心位置放 LED 或 LED 背光源
- 图形比较难画

图 33 示出了在中心有一个按键的滚轮。示意图 A 显示了滚轮和按键电极和走线。示意图 B 则突出显示了 TX（淡蓝色）、RX（红色）和地（绿色）电极。

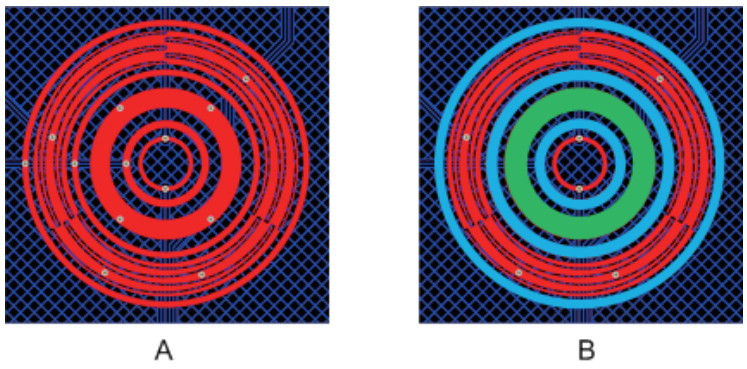


图 33：具有中心按键的三电极滚轮设计

6.7.4 互电容传感器的走线

- 可能的情况下应避免在靠近 RX 走线的地方排布 TX 走线
  - — RX 和 TX 的走线间距只要小于一个手指，在两个走线之间就形成了一个按键。
  - — 如果不能避免在同一层上，则在他们中间加一根地线隔开（这增加了寄生电容）
  - — 假如 TX 需要与 RX 交叉，则垂直交叉走线
  - —不同层的走线要避免 RX 与 TX 正好平行重叠
- 紧挨着其他的 TX 线路排布 TX 线路
- 紧挨着其他的 RX 线路排布 RX 线路
- 使附近的地远离（间距为面板厚度的一半）TX 和 RX 走线（这减小了寄生电容）

电极铺地

• 为了实现噪声抑制，需在以下位置采用 25% 的网格地：

— PCB 的反面

或

— PCB 的正面（与电极的间距至少为面板厚度的一半）

图 34 示出了一个采用两根 TX 和四根 RX 走线的 8 按键小键盘。在该布局中，TX 走线靠在一起，RX 也一样，但 TX 和 RX 之间分的很开。其中有一根 RX 和 TX 由于 PIN 脚原因走的比较近，在他们之间加了地做隔离。

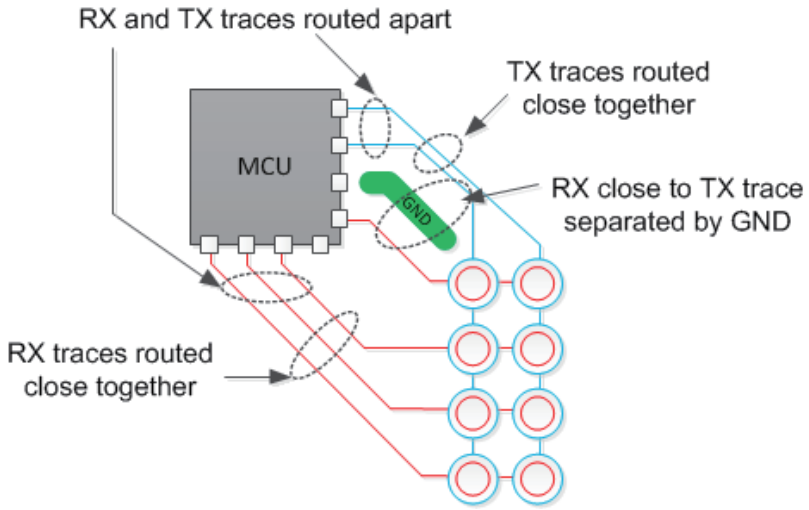


图 34：互电容传感器走线

6.8 参考文献

• http://wcalc.sourceforge.net/cgi-wcalc.html

6.9 超低功耗

本小节介绍了针对超低功耗的需求，如何从硬件和软件上逐步优化，进而实现超低功耗，并列举了一些测试数据， 具体内容请参考英文文档 CapTIvate™ Technology Guide 相关章节。

6.10 防水设计

本小节介绍了针对防水应用的相关设计注意事项，具体内容请参阅英文文档 CapTIvate™ Technology Guide 相关章节。

6.11 噪声抑制

本小节介绍了电容触摸应用中噪声的由来，以及为了增加系统抗噪声能力，如何从芯片、软件、硬件多方面改善相关设计来提高噪声免疫力。具体内容请参阅英文文档 CapTIvate™ Technology Guide 相关章节。

7 Design Center GUI

7.1 引言

CapTIvate Design Center 是一款快速开发工具，可加快面向支持 CapTIvate 技术的 MSP430 器件的电容式触摸设计。通过帮助产品开发人员逐步了解电容式触摸的开发过程，CapTIvate Design Center 能够通过运用创新的用户图形界面、向导和控件来简化和加速触摸设计。

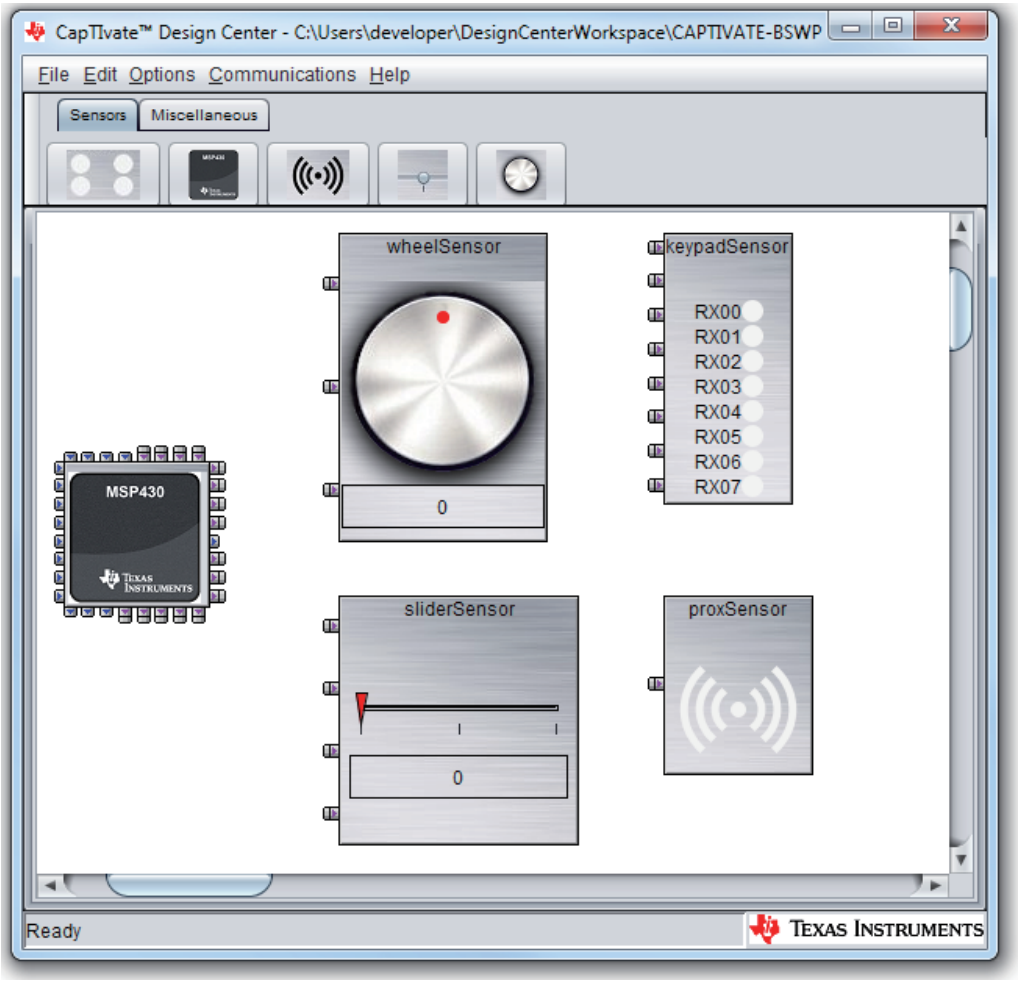


图 35：CapTIvate Design Center

7.1.1 主要特点

- 直观的图形用户界面 (GUI)，用于创建、配置和定义 MSP430 的传感器连接
- 支持滑条、滚轮、按键组和接近传感器
- 在同一款设计中支持互电容和自电容式传感器类型
- 用于 CCS 和 IAR IDE 的完整源代码项目的自动化生成
- 通过一个 HID 通信桥接器与目标板实现实时通信，用户利用实时通信能够：
  - 查看详细的传感器数据
  - 配置和调优传感器性能

7.1.2 硬件和软件要求

满足下列要求的 PC 可以执行 CapTIvate Design Center：

- Windows 7 操作系统和 Java version 1.6+
- Apple OS X 10.10.1+ 和 Java version 1.7+
- Linux OS 和 Java version 1.6+

该开发工具生成的源代码项目支持以下 IDE：

- TI Code Composer Studio™ (CCS) version 6.1 或更高的版本
- IAR System Embedded Workbench for MSP430 version 6.30 或更高的版本

7.1.2.1 额外的 Linux 要求

采用一个根账户来运行应用程序，以确保该应用程序拥有访问由 MSP 微控制器安装的 HID 设备的权限。

用户需执行以下一次性设置以启动访问：

- 创建文件 /etc/udev/rules.d/ti_hid.rules。
- 把下面的代码行添加至该文件并保存文件。

```
1 # Allow open access to any USB device with the default TI MSP device
2 # Vendor Id of 0x2047
3 ATTRS{idVendor}=="2047", MODE="0666"
```

7.1.3 许可

CapTIvate Design Center 是依据 BSD 3-clause 许可证发布的。详情请见安装目录中的许可清单。

7.2 CapTIvate™ Design Center 快速指南

本节提供了一个快速指南，描述怎样使用 CapTIvate Design Center 来建立一个典型的电容式触摸设计，生成源代码以及与目标板进行通信。

7.2.1 启动 Design Center

双击桌面 CapTIvateDesignCenter 快捷图标以启动该工具。

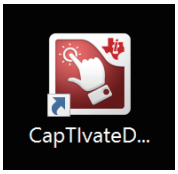


图 36：Design Center 图标

7.2.2 创建一个新项目

选择 File ➞ Project New 菜单来创建一个新项目。

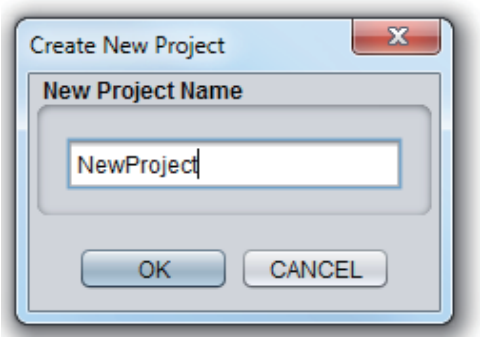


图 37：创建新项目

7.2.3 放置传感器

7.2.3.1 把一个滑条传感器放置于设计区域中

- 选择滑条传感器图标并移动光标将新的对象放置于设计区域中。

图 38：选择滑块传感器

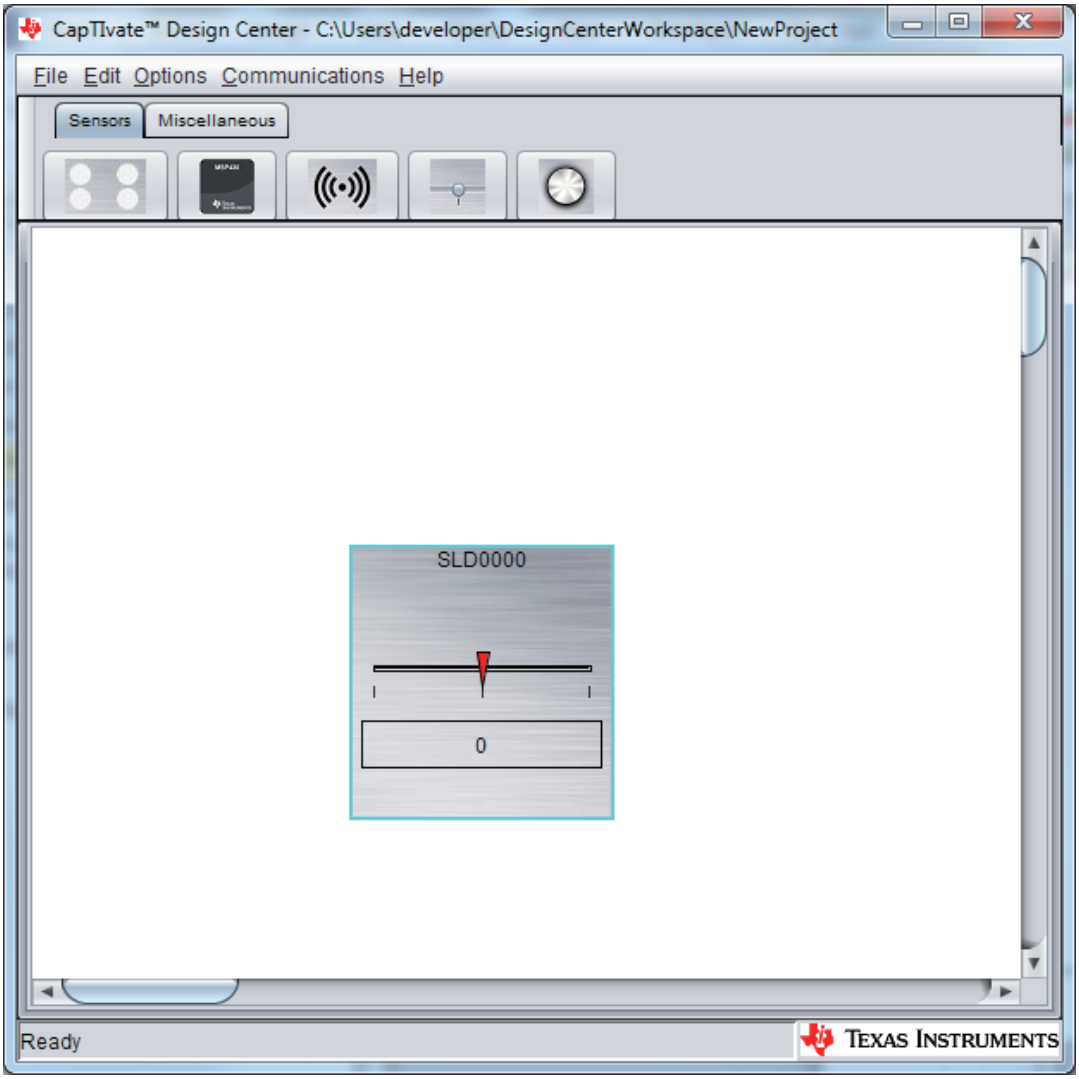


图 39：放置滑块传感器

- 显示传感器属性并将之配置为具有四个元件的自电容滑条。
  - 双击设计区域中的滑条对象以显示其属性对话框。
  - 配置该传感器为四个元件并关闭属性对话框。

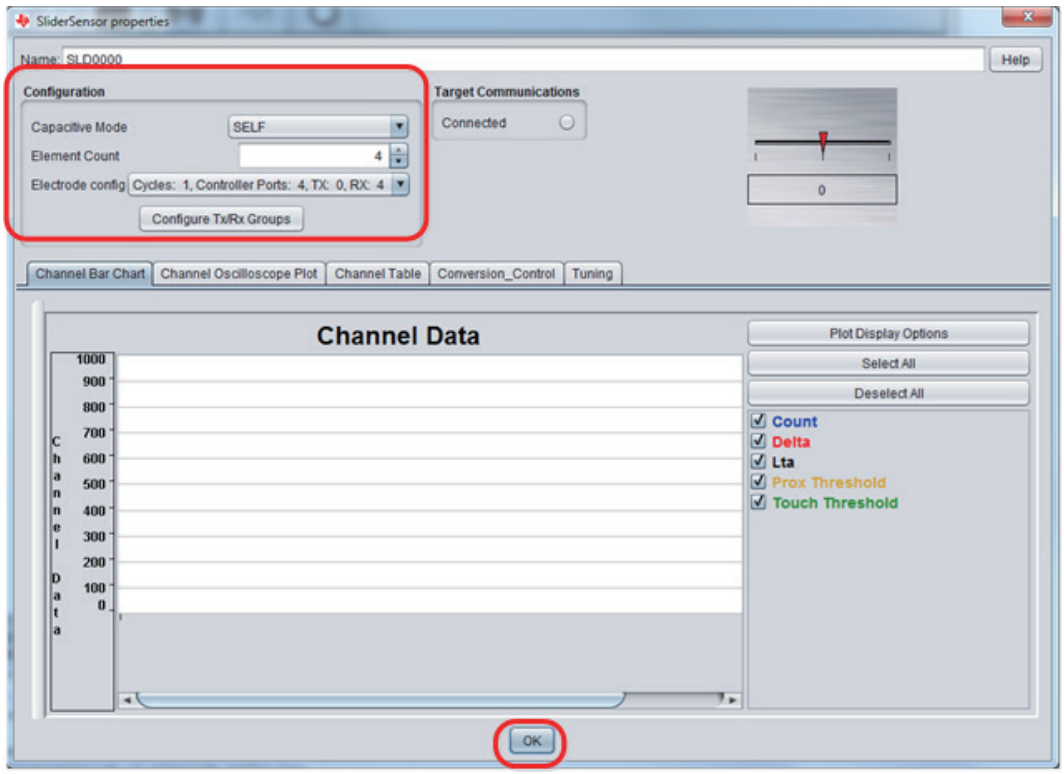


图 40：滑块传感器属性

7.2.3.2 把一个滚轮传感器放置于设计区域中

- 选择滚轮传感器图标并移动光标将新的对象放置于设计区域中。

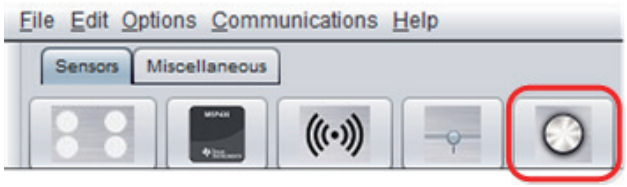


图 41：选择滚轮传感器

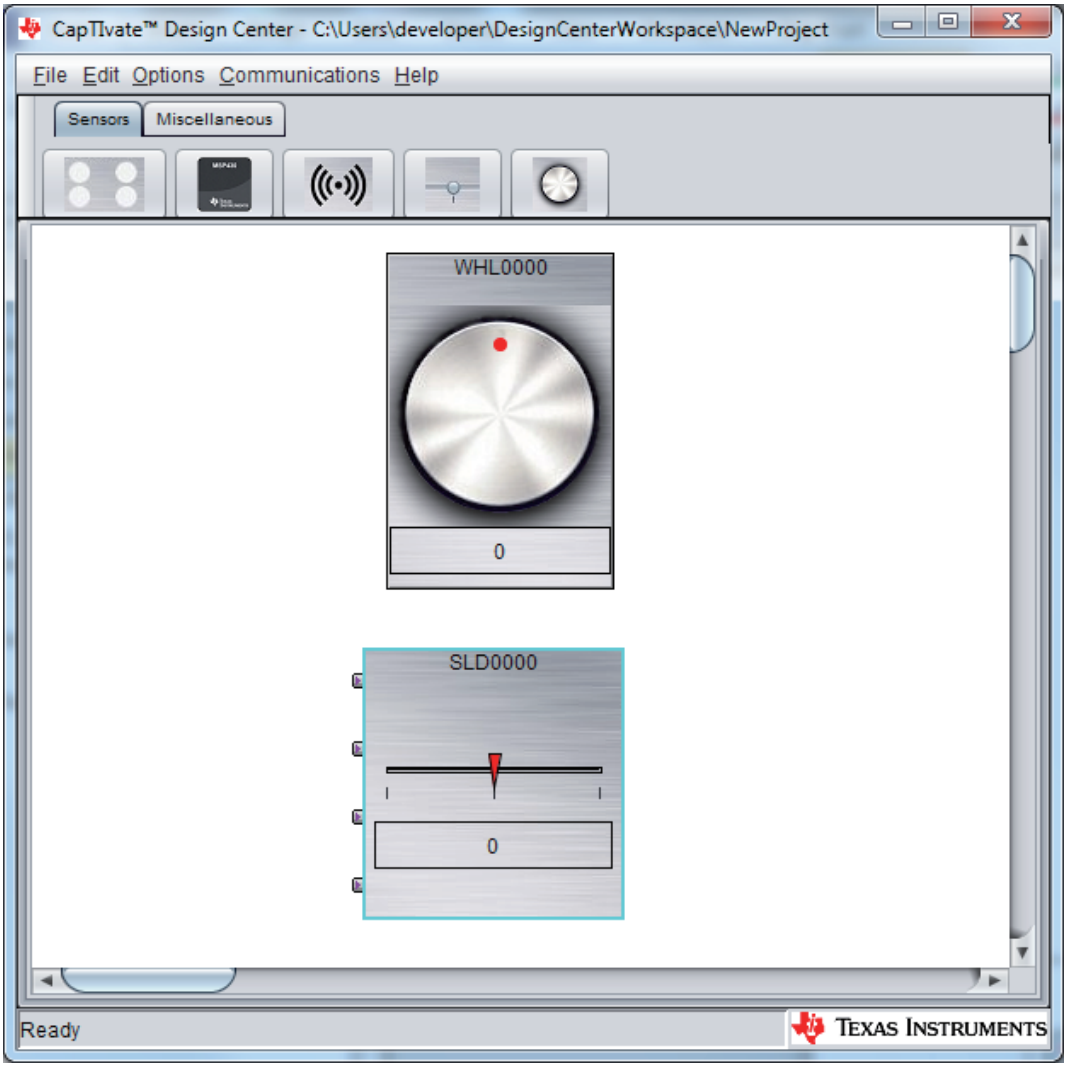


图 39：放置滑块传感器

- 显示传感器属性并将之配置为具有三个元件的自电容滚轮。
  - 双击设计区域中的滚轮对象以显示其属性对话框。
  - 配置该传感器为三个元件并关闭属性对话框。

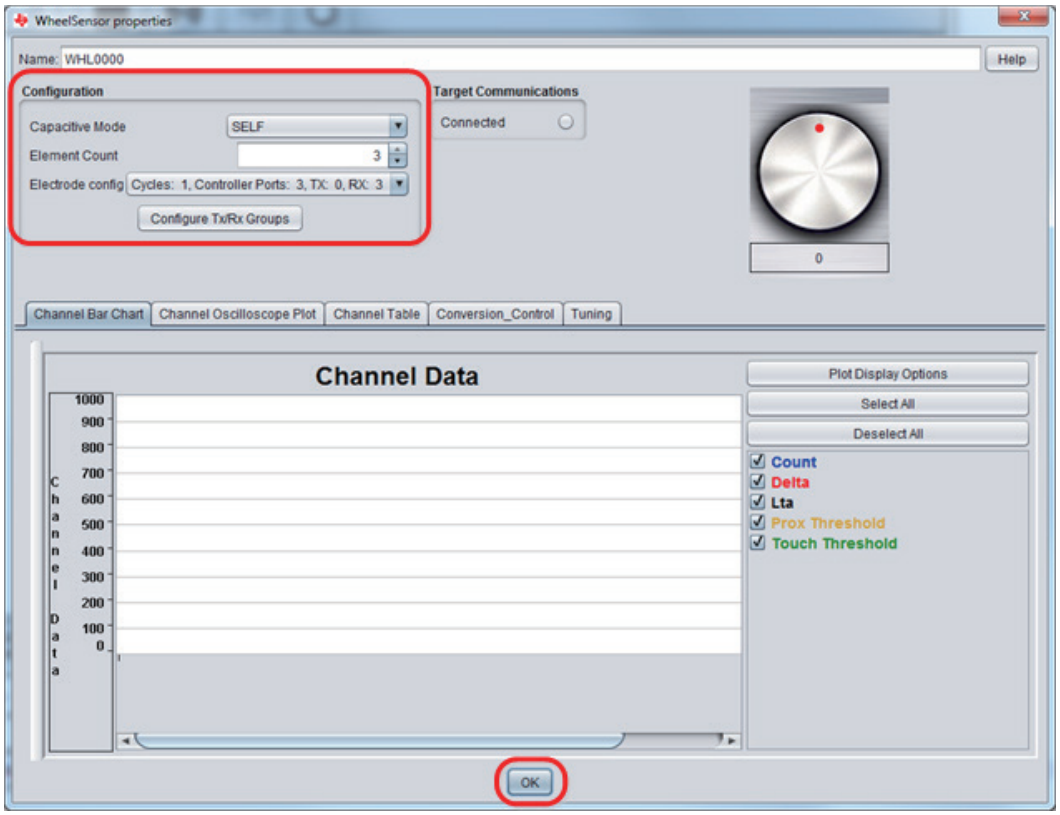


图 43：滚轮传感器属性

7.2.3.3 把一个按键组（键盘）传感器放置于设计区域中

- 选择按键组传感器图标并移动光标将新的对象放置于设计区域中。

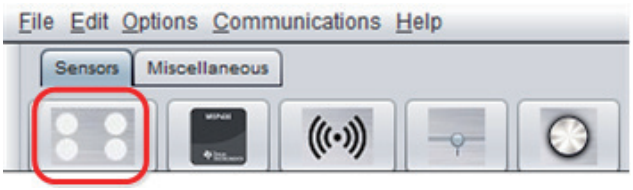


图 44：选择按键组传感器

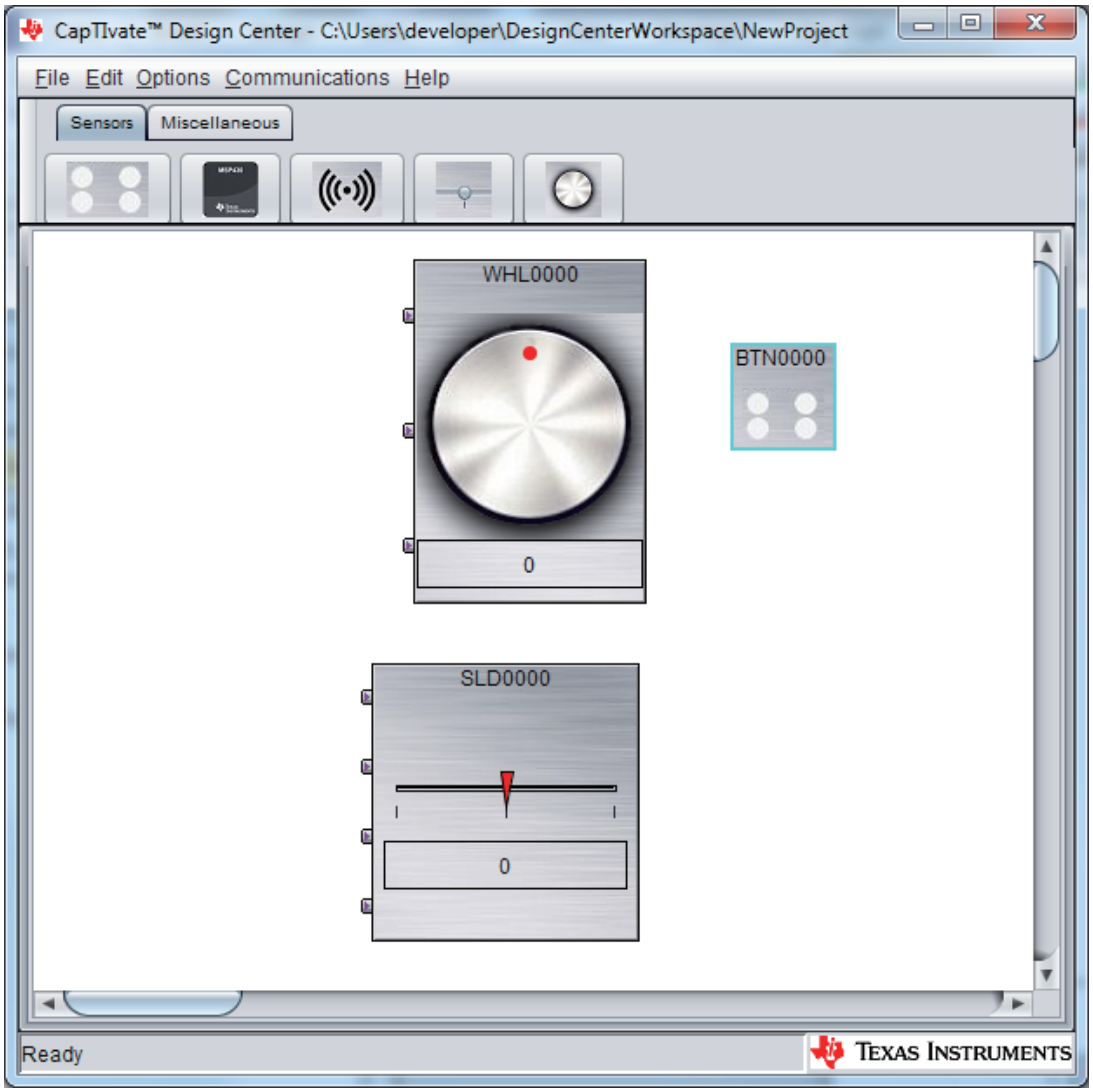


图 45：放置按键组

- 显示传感器属性并将之配置为具有 8 个元件的自电容按键组。
  - 双击设计区域中的按键组对象以显示其属性。
  - 配置该传感器为 8 个元件并关闭属性对话框。

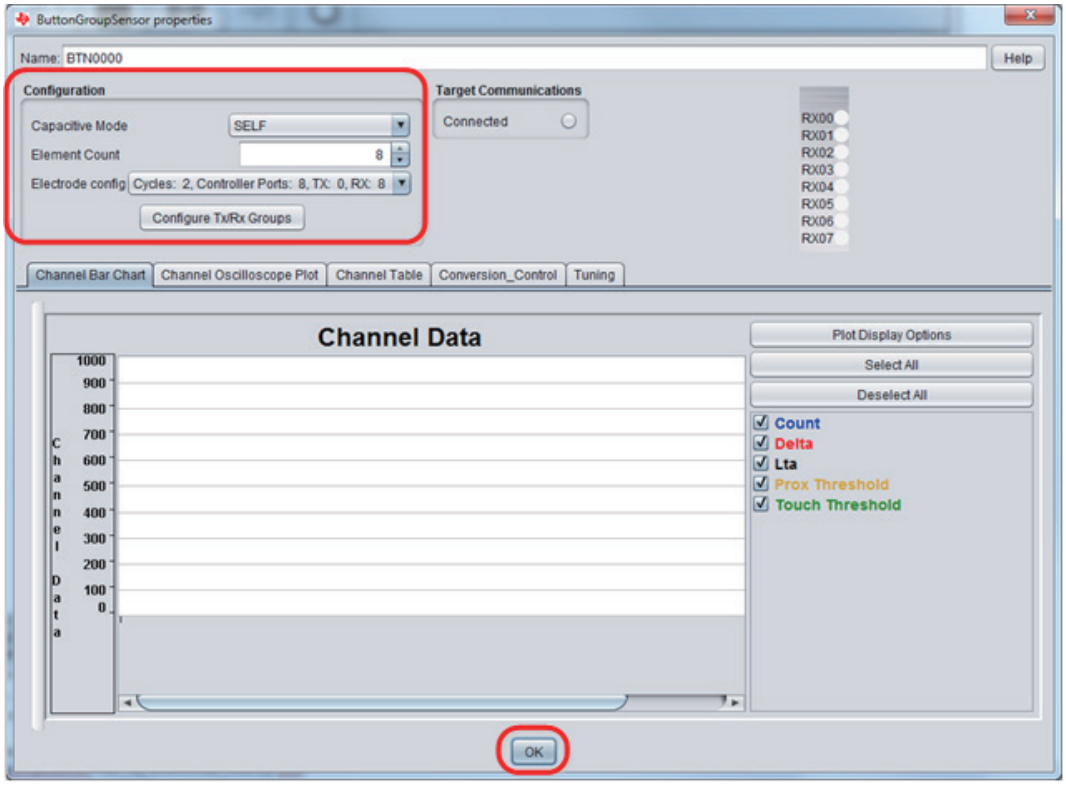


图 46：按钮组传感器属性

7.2.3.4 把一个接近传感器放置于设计区域中

- 选择接近传感器图标并移动光标将新的对象放置于设计区域中。

图 47：选择接近传感器

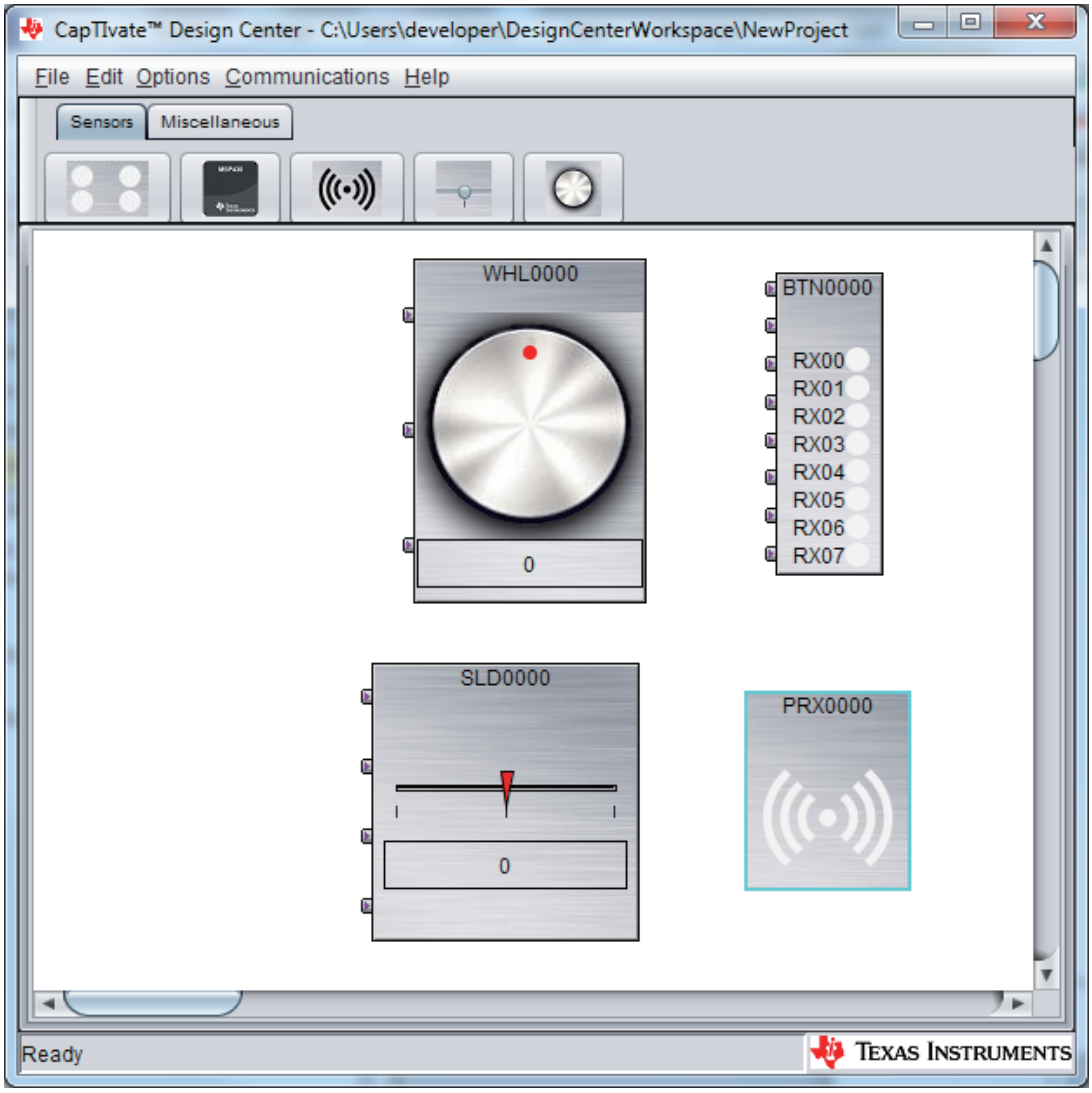


图 48：放置接近传感器

- 显示传感器属性并将之配置为具有 1 个元件的自电容接近传感器。
  - 双击设计区域中的接近传感器对象以显示其属性。
  - 配置该传感器为 1 个元件并关闭属性对话框。

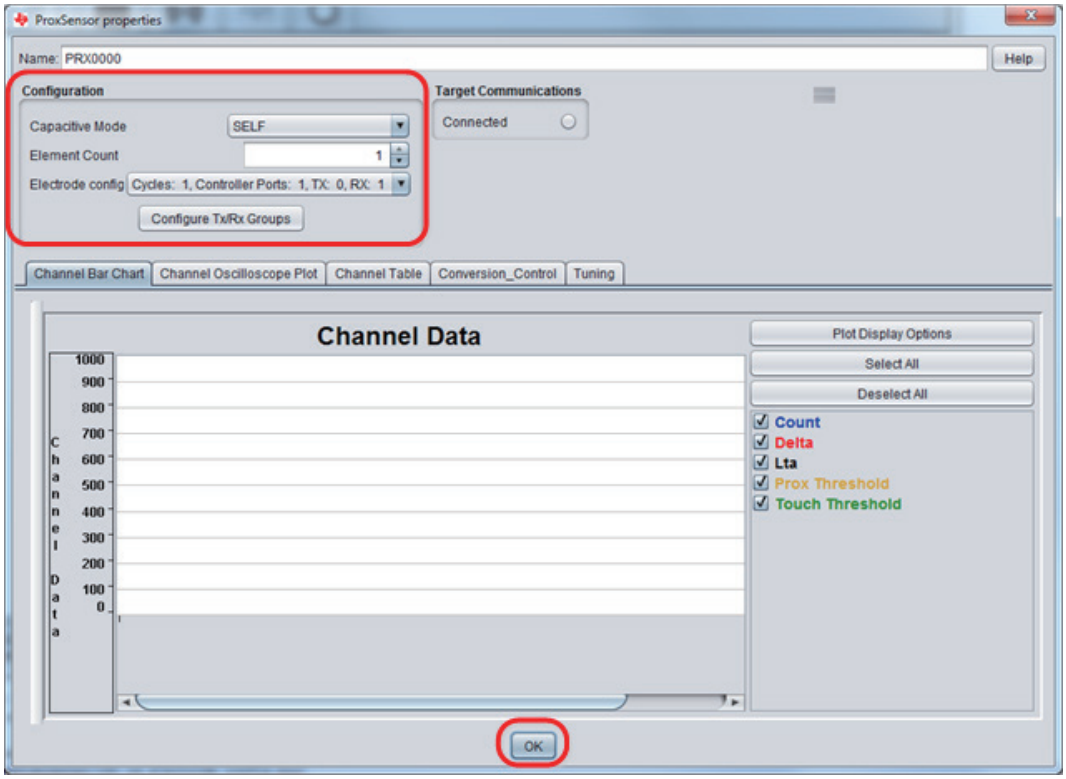


图 49：接近传感器属性

7.2.4 放置 MSP430 控制器

- 选择 MSP430 图标并移动光标将新的对象放置于设计区域中。

图 50：选择 MSP430 MCU

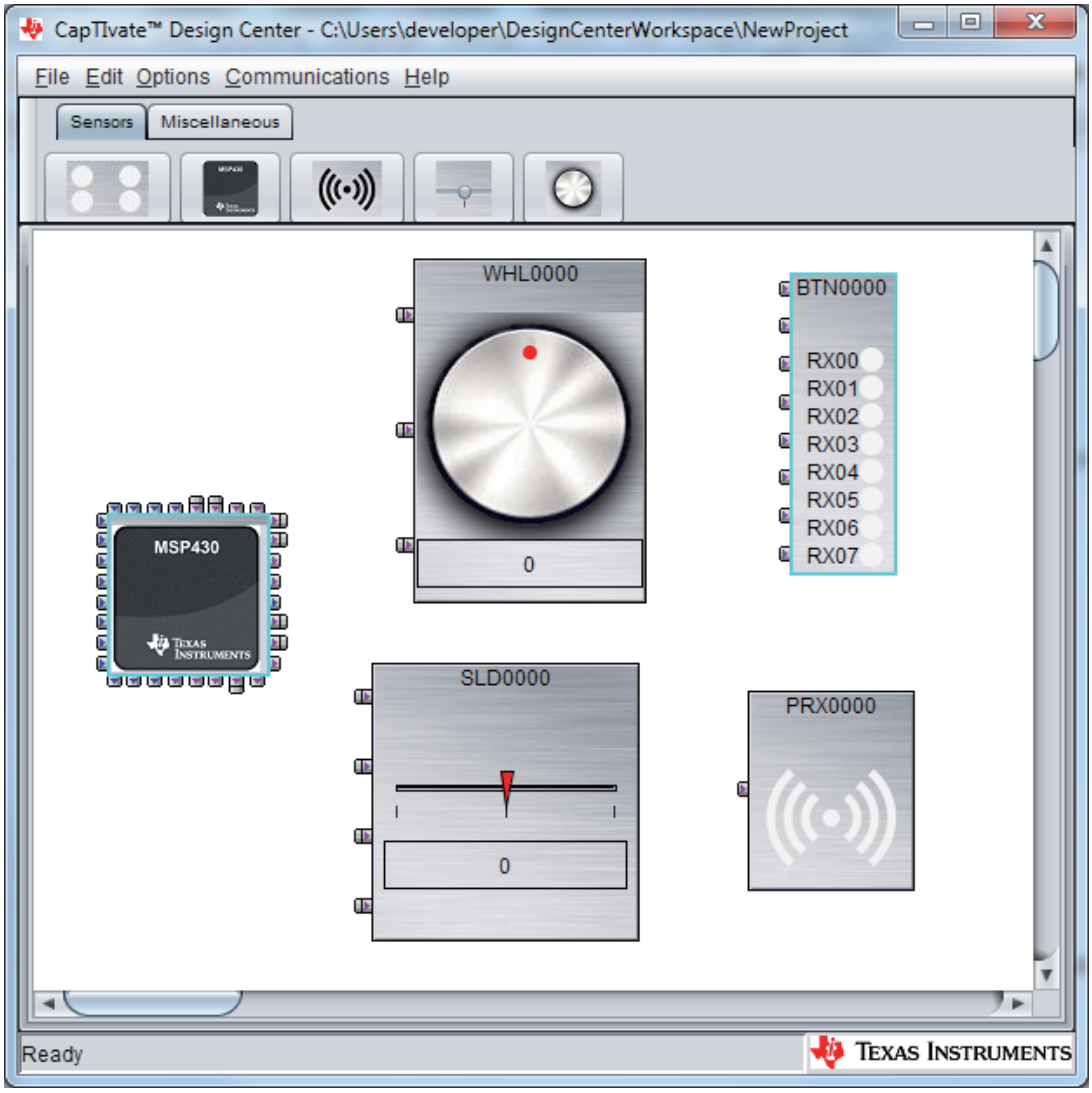


图 51：放置 MSP430 MCU

7.2.5 把传感器连接至 MSP430 端口

双击设计区域中的 MSP430 控制器对象以显示其属性。

- 把 MSP430 控制器配置为 MSP430FR2633IRHB（32 引脚 QFN 封装）。
- 注意 Errors LED 是红色的，这表示仍然有未连接的传感器端口。

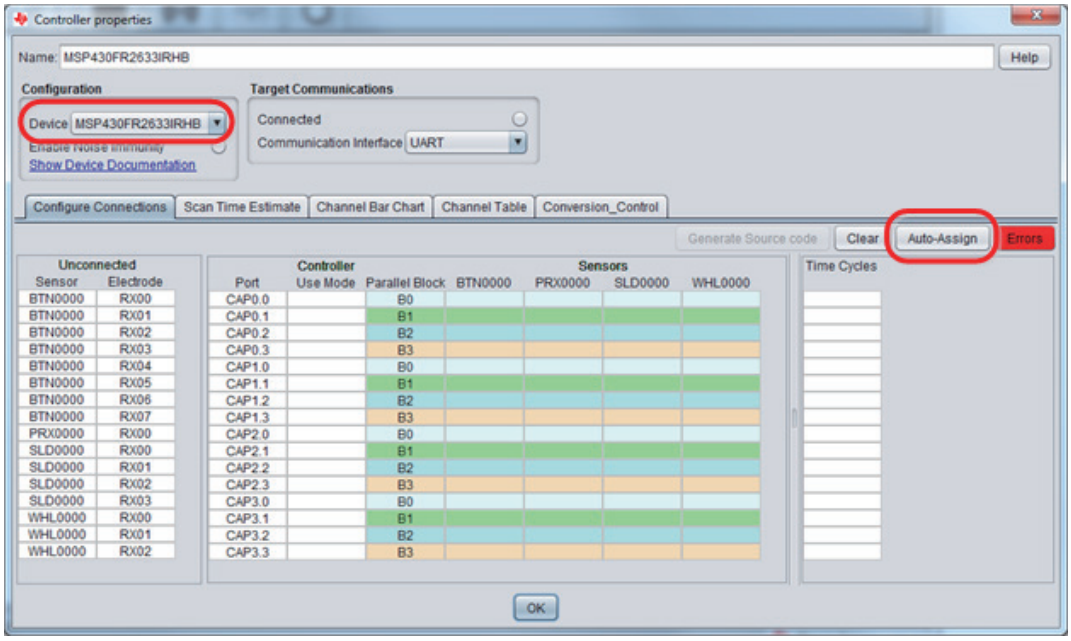


图 52：设定控制器属性

选择 Auto-Assign 按钮把所有的传感器端口自动分配给 MSP430 上合适的端口。请注意 Errors LED 是绿色的，这表示所有的传感器端口均已分配给控制器端口。

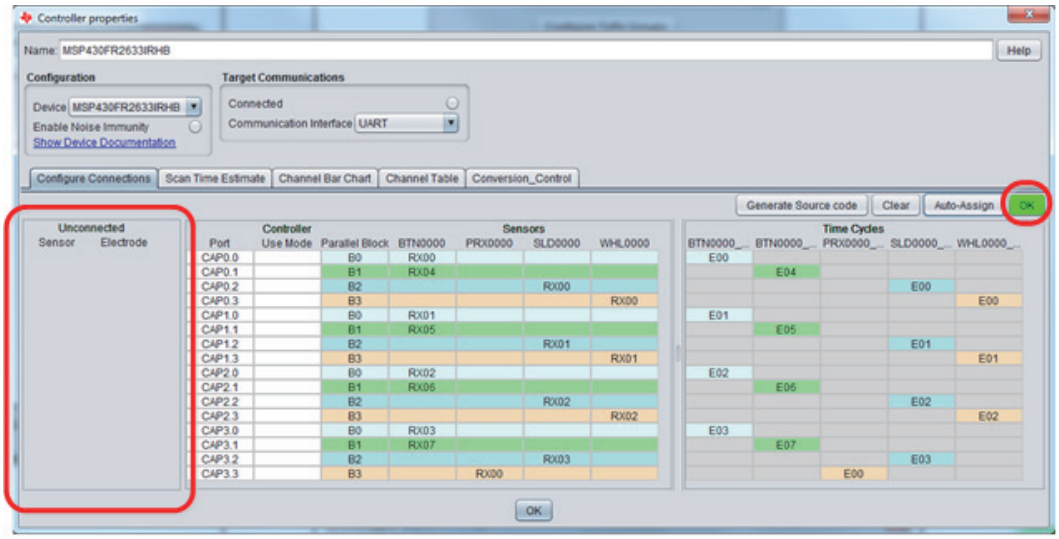


图 53：控制器属性完成

7.2.6 生成源代码

在 MSP430 控制器属性对话框上选择 Generate Code 按钮。

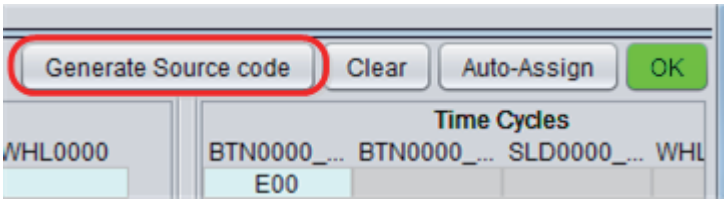


图 54：生成源代码

在对话框中选择 OK 以创建一个完整的项目。

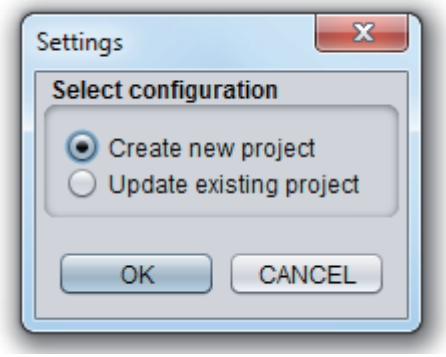


图 55：选择配置

保持用于生成代码的默认位置并选择 OK。

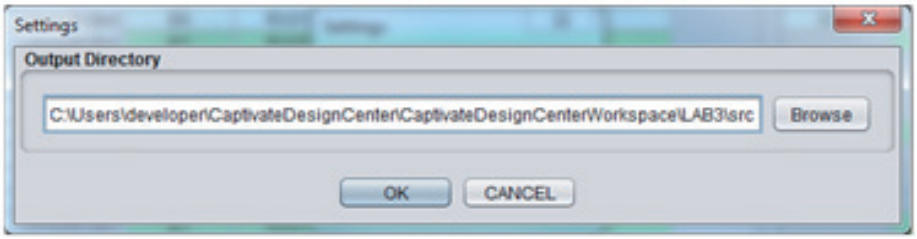


图 56：输出目录

7.2.7 使用 CCS 或 IAR 导入并运行针对 FR26xx 或 FR25xx 生成的固件项目

如需了解有关在 CCS 或 IAR 中导入或打开一个项目并运行生成项目的信息，请见 7.4 节。

7.2.8 与目标板进行通信

本节通过实例介绍如何通过 HID 读取并显示目标板的数据。

除非目标板连接至一个与在 Design Center GUI 中创建的配置相匹配的传感器面板，否则无法获得有效的传感器数据。

7.2.8.1 启动与目标板的 HID 通信

选择 Communications → Connect 菜单以启动 HID 通信。通过查看出现在主窗口左下角的消息来验证 HID 设备处于连接状态（见图 57）。

图 57：HID 已连接

7.2.8.2 显示传感器数据

- 双击滑条对象以显示属性对话框。选项卡用于显示实时传感器数据的图形，并于进行传感器配置的读取和修改以及参数的调优。
- 传感器的位置同样显示出来。

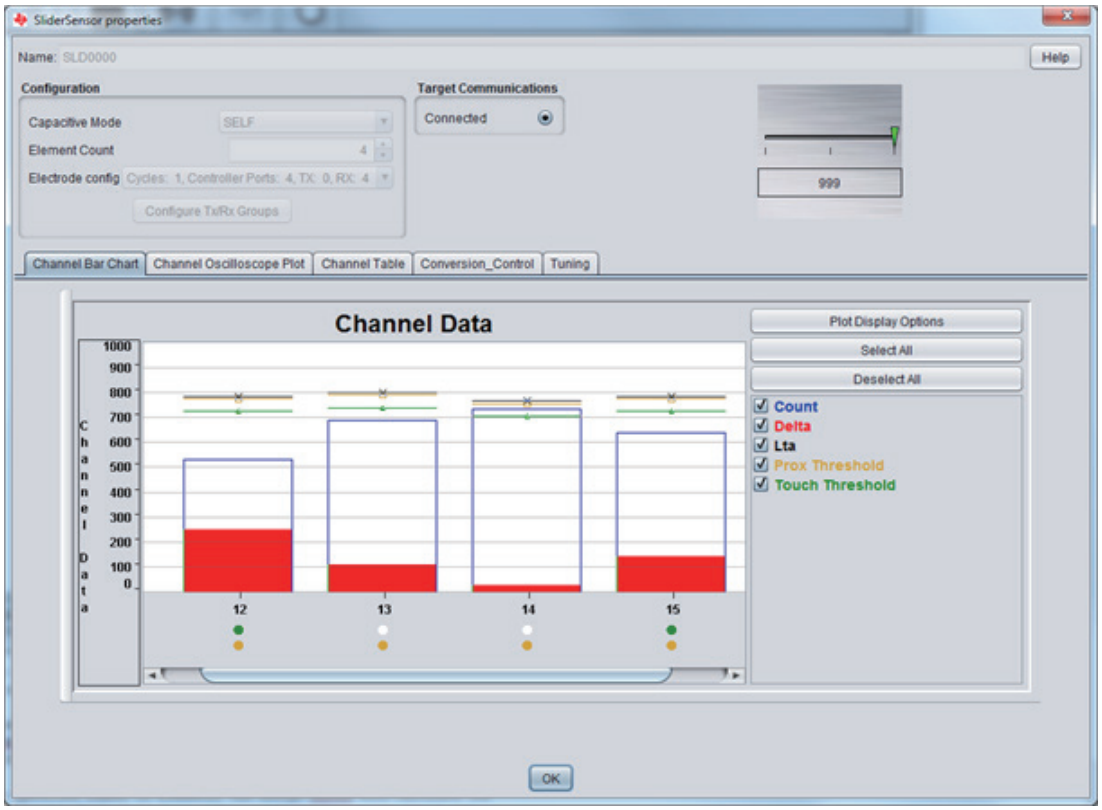


图 58：通道条形图

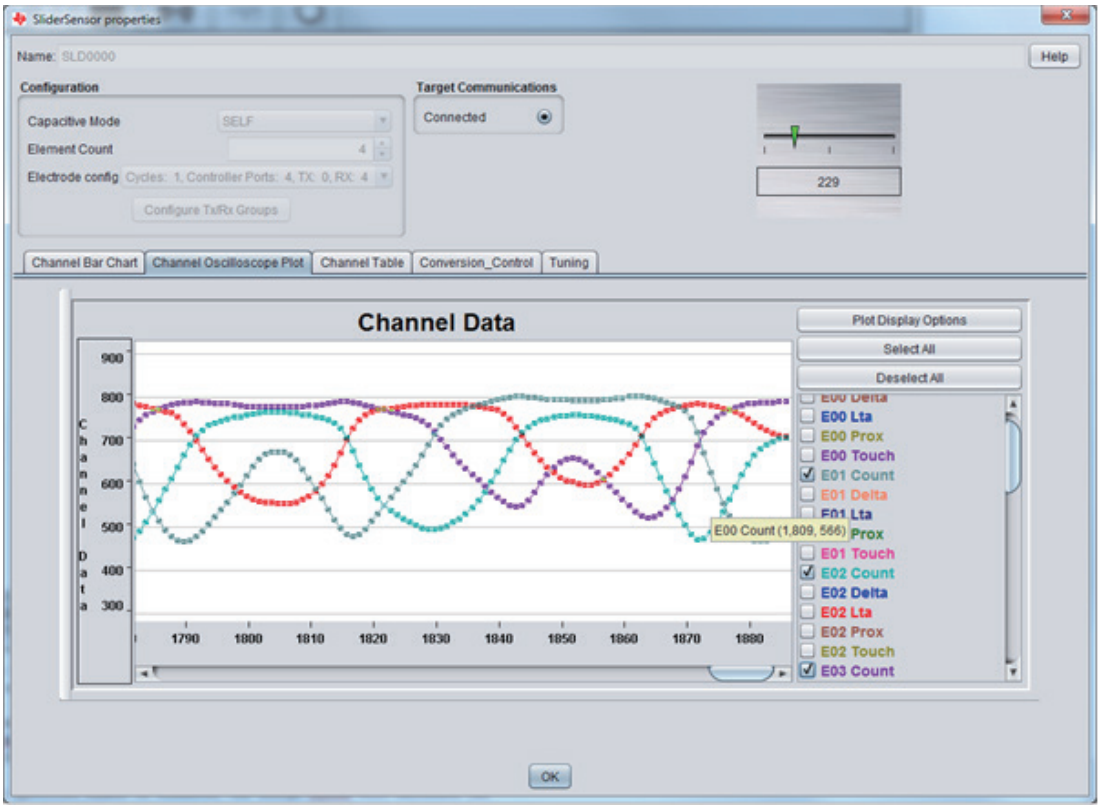


图 59：示波器图表

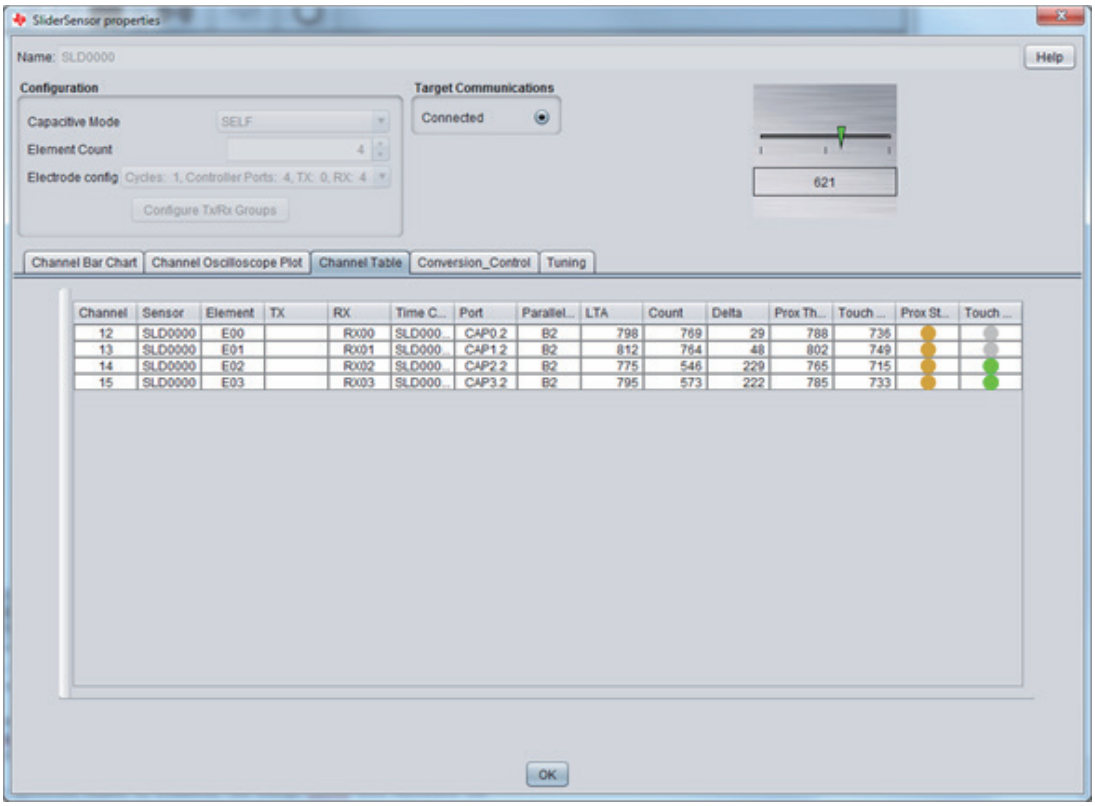


图 60：通道表

7.2.8.3 调优传感器性能

可采用 Tuning 面板对传感器进行实时配置，修改目标参数。所有参数均直接在该面板中说明，传感器参数仅在目标板已连接时才能修改。

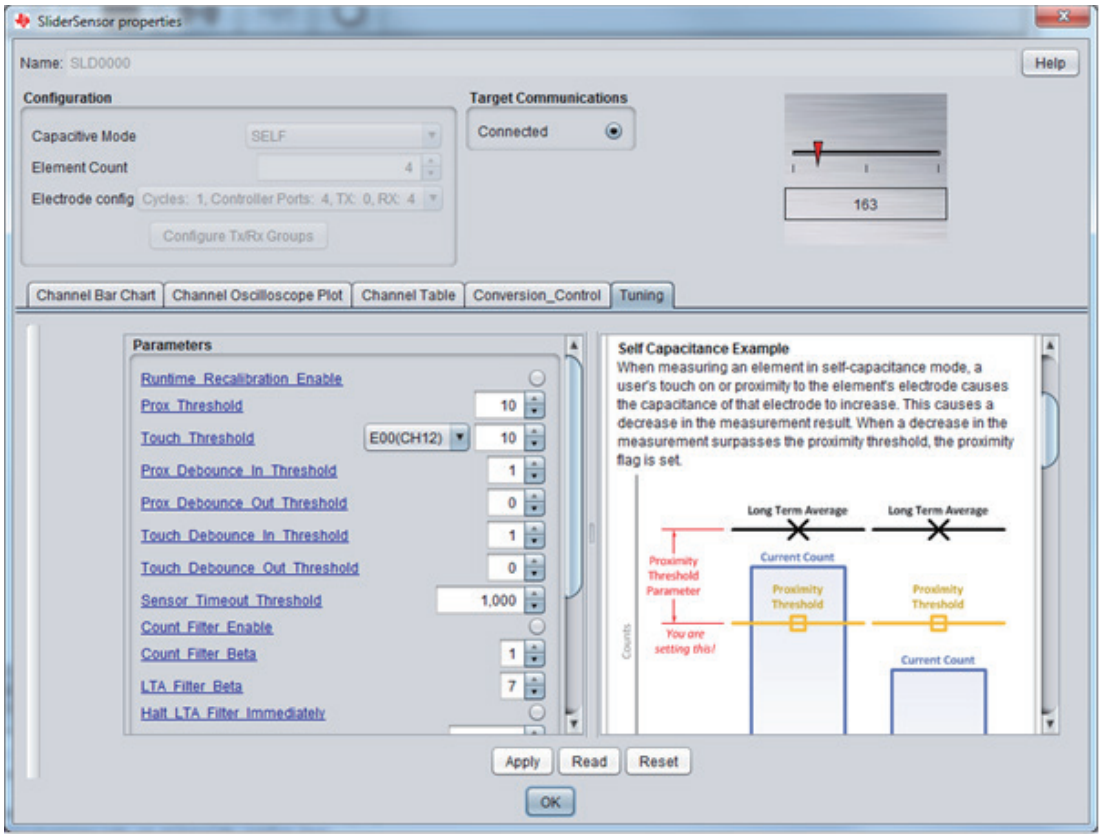


图 61：传感器调优参数

可以取消停靠任意调优视图以便更容易地一次查看一个或多个视图。点击并拖曳位于视图左边的句柄来移动它。关闭一个取消停靠的窗口可使其返回选项卡窗格。

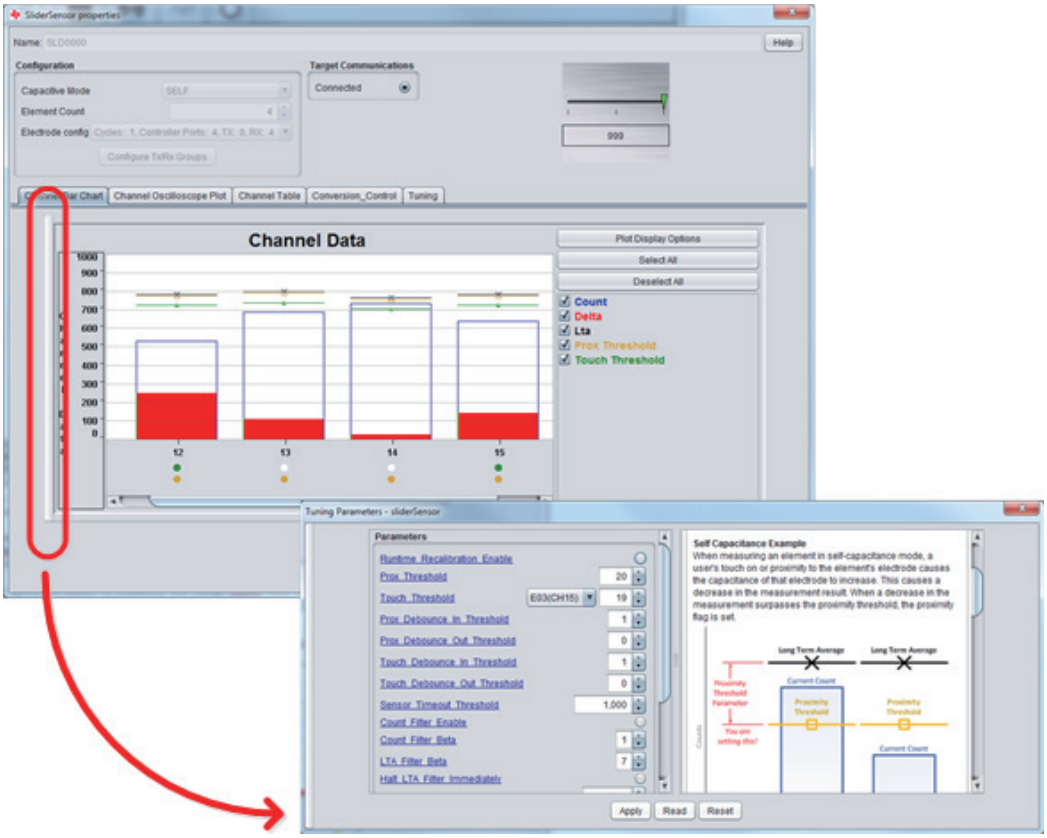


图 62：取消停靠视图

7.3 CapTIvateTM Design Center 用户指南

本节提供了有关 Design Center 特性的详细信息。

7.3.1 Design Center GUI 面板

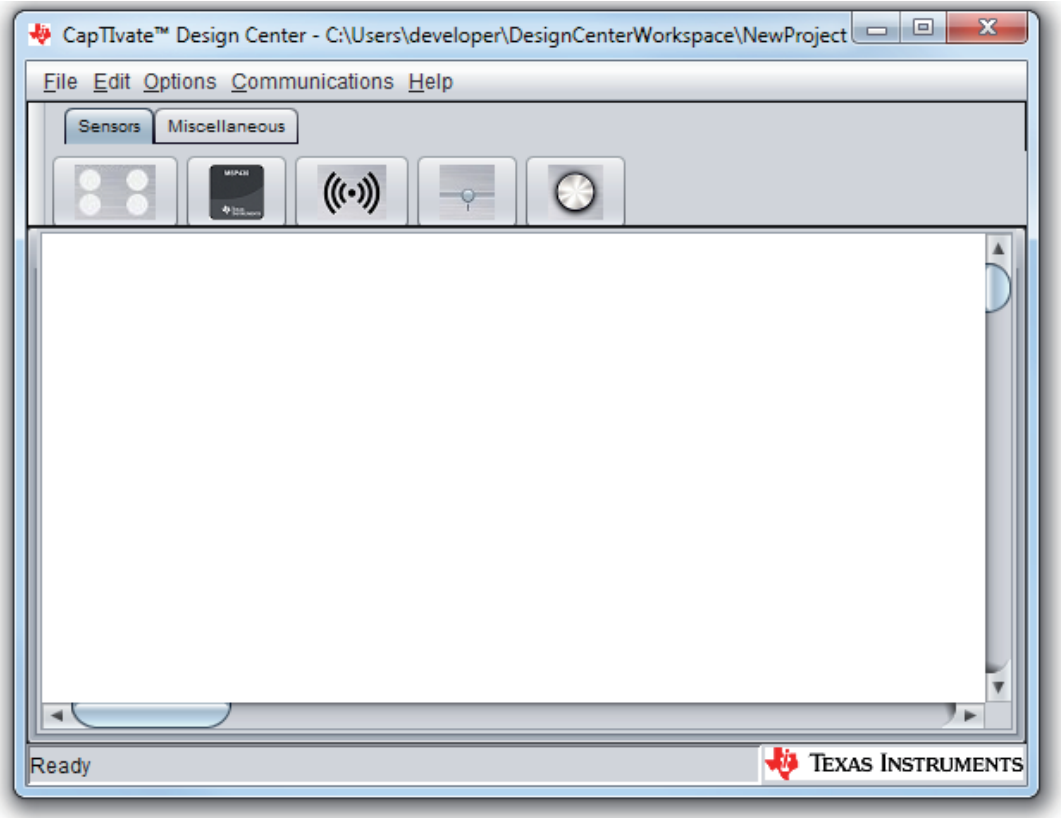


图 63：Design Center GUI 传感器选项卡

Design Center GUI 包括三个主要面板：

菜单栏

这是对 GUI 的功能和控件进行访问的菜单栏（见图 64）。

对象选择选项卡

在对象选择选择卡上可以选择对象放置于设计工作区。对象的类型包括：

- 传感器
- MSP430 控制器
- 诸如注释等各种各样的对象

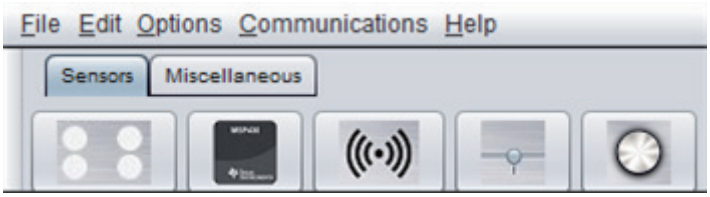


图 64：对象选择选项卡

设计工作区

- 在设计工作区中放置对象：
  - 利用鼠标左键从对象选项卡选择一个对象
  - 移动光标在设计区域中的位置并点击鼠标左键来放置对象
- 可以选择对象并在设计工作区将其移动至任意位置。
- 在选择了对象之后可通过点击鼠标右键来显示针对对象的附加操作。
- 如需显示属性并配置对象，则采用鼠标左键进行双击。
- 如需获得针对某种对象类型的在线帮助，则选择它并按 F1 键。

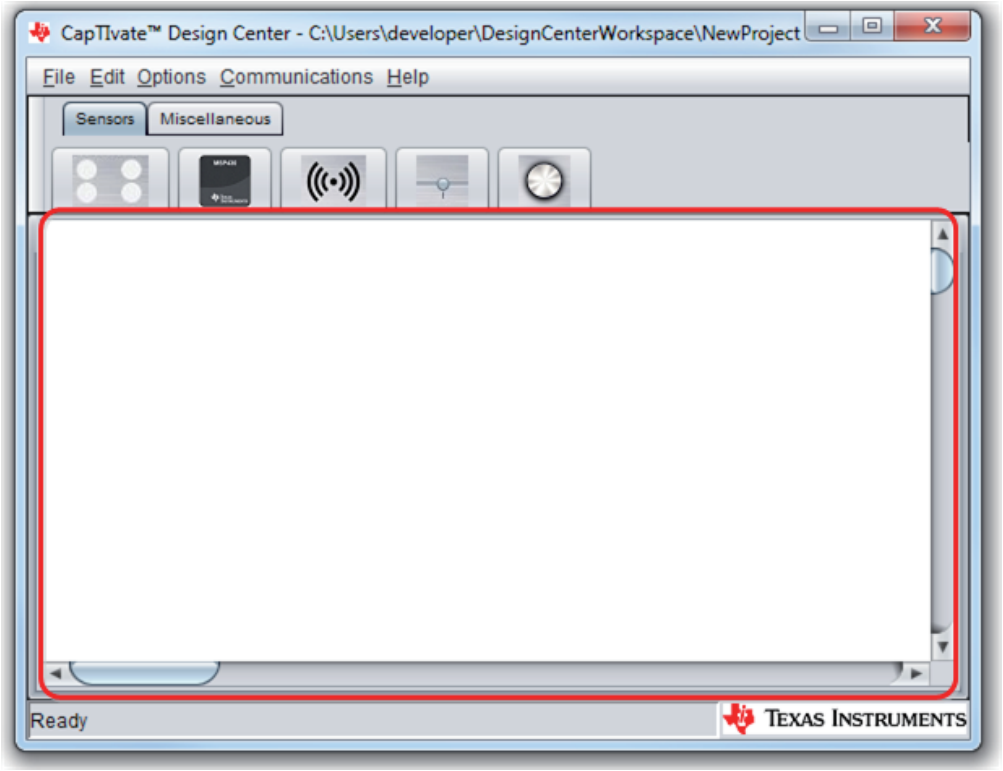


图 65：设计工作区

7.3.2 菜单说明

7.3.2.1 文件菜单

图 66 显示了文件 (File) 菜单。

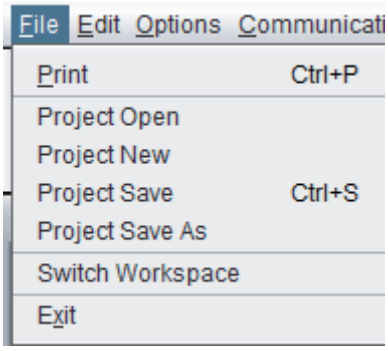


图 66：文件菜单

Print

采用该菜单来打印窗口。

Project Open、Save 或 Save As

采用该菜单以执行下列操作：

- 在当前的工作空间里打开一个新项目或已存在的项目
- 保存当前的项目
- 将当前的项目另存为一个不同名称的项目（见图 67）

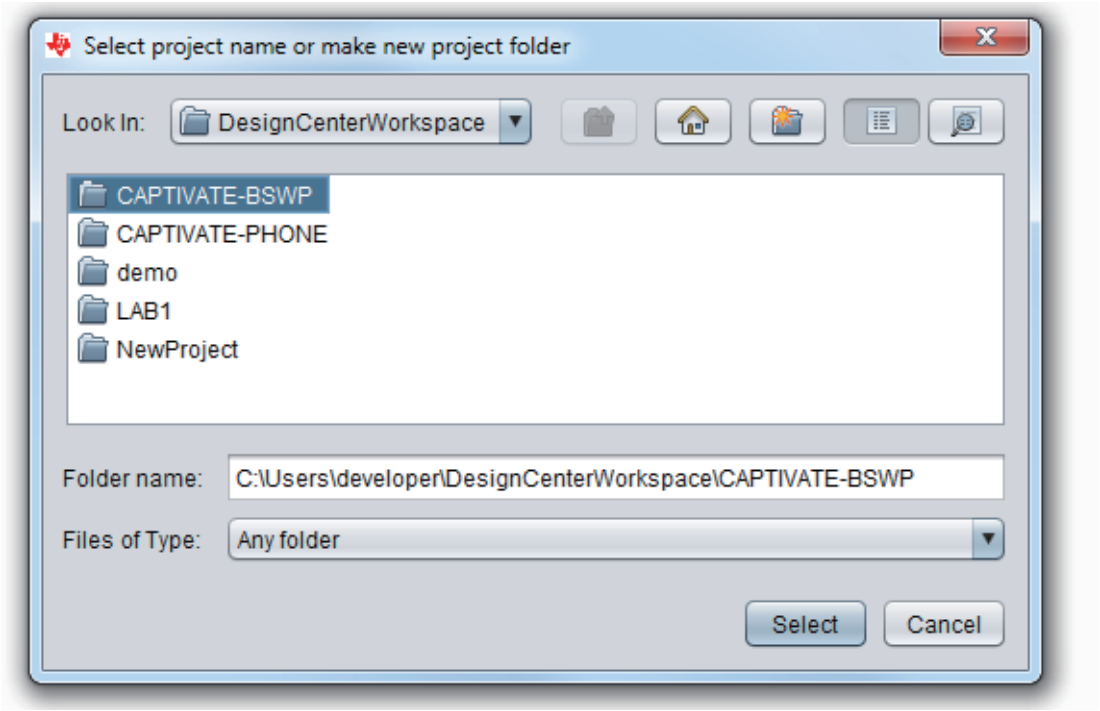


图 67：选择项目名称或建立新项目文件夹

Switch Workspace

该 Design Center 把项目存储在一个工作空间目录之内。采用 Switch Workspace 菜单来打开一个不同的工作空间目录。

默认的工作空间目录为 USERPROFILE%/CapTIvateDesignCenter/CapTIvateDesignCenterWorkspace。

Exit
退出该工具。

7.3.2.2 编辑菜单

图 68 示出了编辑菜单。

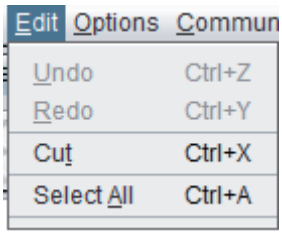


图 68：编辑菜单

Undo 或 Redo
撤消或重复上一次编辑菜单操作。

Cut
从设计中除去选定的对象。

Select ALL
选择设计工作区中的所有对象。

7.3.2.3 选项菜单

Display
使用该对话框定制设计工作区的显示选项。

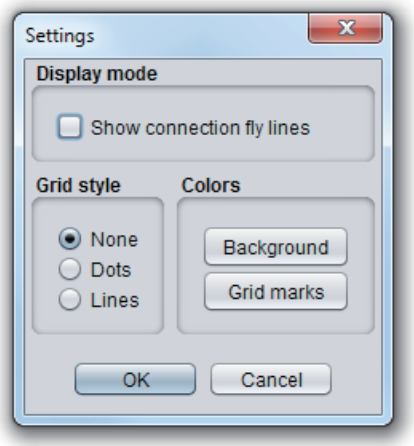


图 69：显示选项

Features

改变特性模式。在高级 (Advanced) 模式中，显示了更多的传感器调优和配置选项。

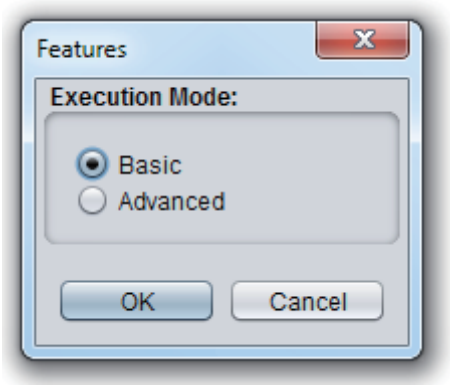


图 70：特性选项

7.3.2.4 通信菜单

图 71 显示了通信菜单。

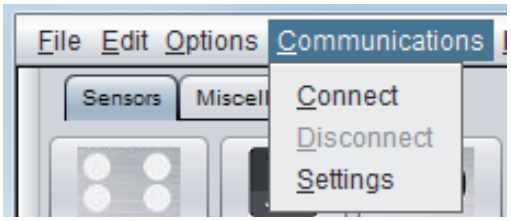


图 71：通信菜单

Connect/Disconnect

负责控制 Design Center 是否通过 HID 与目标板进行通信。

Settings

改变 HID 连接参数。

注： 这些参数默认至针对 TI CapTIvate 技术 EVM 的正确数值。

5.3.2.5 帮助菜单

图 72 显示了帮助菜单。

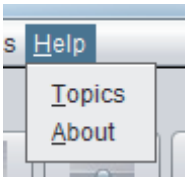


图 72：帮助菜单

Topics

在浏览器窗口中显示帮助文档。

About

显示软件版本信息。

7.3.3 控制器属性

双击控制器对象以显示属性对话框。

图 73：控制器属性图标

使用属性对话框对控制器对象执行以下操作：

- 配置器件；
- 将控制器引脚映射到传感器元件；
- 启用目标板通信并查看传感器数据。

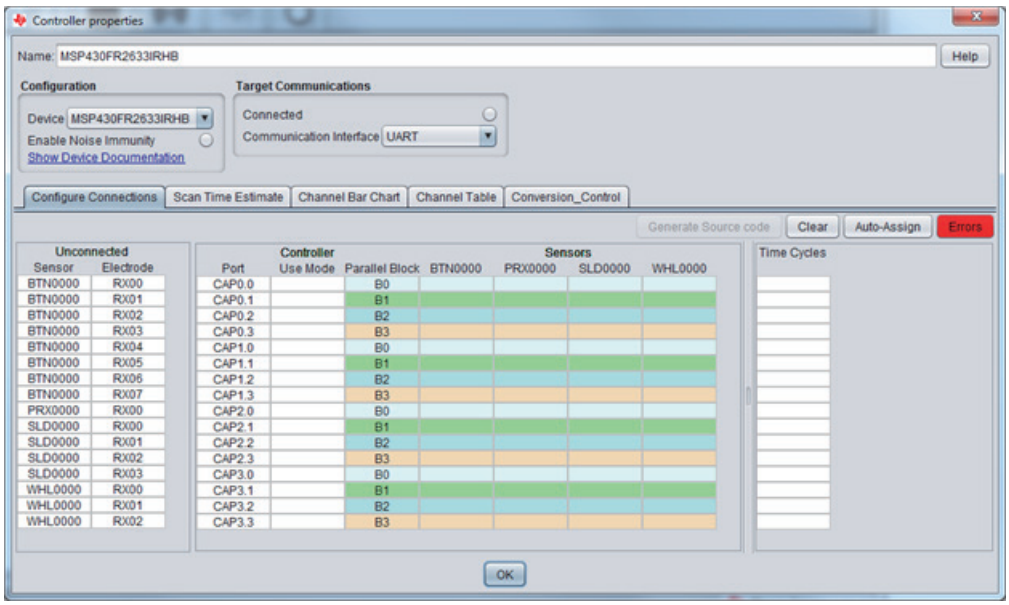


图 74：控制器属性

7.3.3.1 器件配置

- 从下拉列表中选择所需的 MSP430 CapTIvate 技术系列器件。
- 启用或停用抗噪声功能。

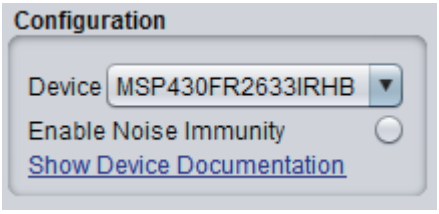


图 75：器件配置

7.3.3.2 从控制器属性对话框启用目标板通信

启用或停用目标板通信。该按钮等效于 Communications ➞ Connect/Disconnect 菜单项（见图 76）。

7.3.3.3 选择目标 板 MCU 通信接口

通信接口提供了用于 HID 桥接器 MCU 至目标板 MCU 通信接口的 UART 和 I^2C 接口选择。如需变更当前的接口，则选择一个新的接口并点击 Connected 按钮，HID 桥接器 MCU 立即开始使用新接口。当生成源代码时，生成的项目文件被更新以提供对新接口的支持。

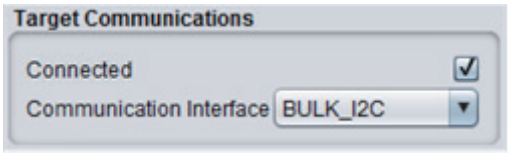


图 76：目标板通信

7.3.3.4 将传感器端口映射到控制器引脚

将传感器端口映射到控制器引脚有三种方法：

- 让 Design Center 采用控制器 Configure Connections 选项卡上的 Auto-Assign 功能自动地对所有的端口进行映射。
- 采用控制器 Configure Connections 选项卡中的连接表来对端口进行手动映射。
- 在设计工作区中的对象上以图形的方式连接传感器端口和控制器引脚。

7.3.3.4.1 自动分配 (Auto-Assign)

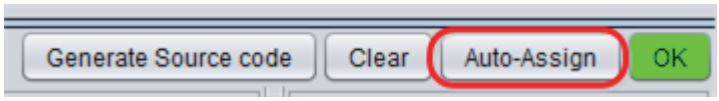


图 77：Auto–Assign 按钮

如果选择了 Auto-Assign 按钮，则 Design Center 尝试以最优的方式将所有未连接的传感器端口 (sensor ports) 连接至控制器引脚。任何已经存在的连接均不改变。可以采用手动方式建立部分连接，然后自动分配其余的连接。

在端口映射没有预先限制的情况下，自动分配是一种最优的选择。如果端口映射由于电路板布局的原因而受到约束，则需选择其他的选项。

如果端口映射成功，则 Errors 指示器变成绿色。假如该指示器是红色的，则仍然存在错误。点击该指示器显示对话框列出剩余错误。

7.3.3.4.2 采用配置连接表的手动映射

配置连接 (Configure Connection) 表（见图 78）用于实现传感器端口和控制器引脚映射方式的精确控制。该表含有三个主要部分：

- Unconnected 显示所有未被映射到控制器引脚的传感器端口。该列中的数据是只读的。
- Controller 显示 MSP430 上支持 CapTIvate 技术的所有端口、使用模式限制以及哪些端口属于同一个并行检测模块的信息。
- Time Cycles 显示哪些传感器端口在目标板上进行并行扫描。该数据为只读型，当检测到传感器连接或配置的改变时更新。

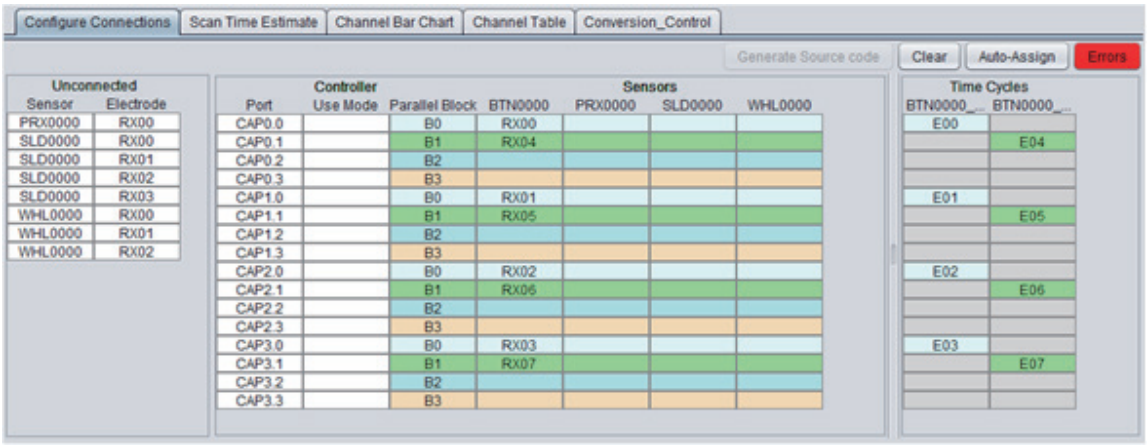


图 78：手动传感器映射

7.3.3.4.2.1 端口列编辑 (Port Column Edits)

选择端口列中的一个单元来打开下拉菜单，用户可以控制端口的使用模式。Use Mode 列显示用于该控制引脚的限制条件（见图 79）。

- Available：可以分配任何传感器端口
- Reserved：没有传感器端口可以分配给该引脚
- Rx only：仅传感器 Rx 端口可以分配给该引脚
- Tx only：仅传感器 Tx 端口可以分配给该引脚
- SELF only：仅自电容传感器端口可以分配给该引脚
- MUTUAL only：仅互电容传感器端口可以分配给该引脚

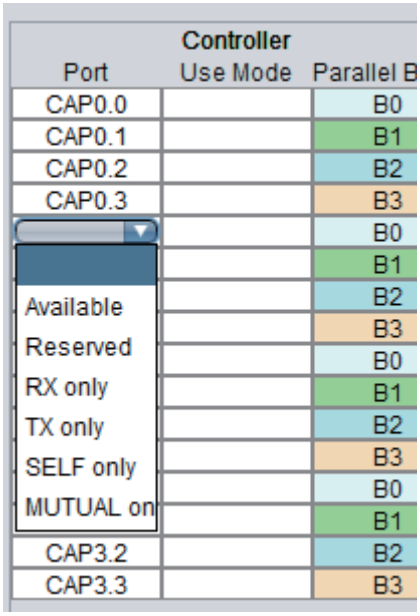


图 79：端口列编辑

7.3.3.4.2.2 并行块列编辑 (Parallel Block Column Edits)

选择端口列中的一个单元来打开下拉菜单，使用该菜单可以把引脚分配给每个并行检测块（见图 80）。

- **Sensors** 部分包含了设计中每个传感器对应的列，每个传感器端口可单独映射到用户指定的控制器引脚。

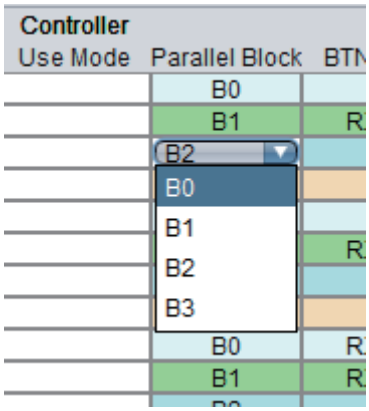
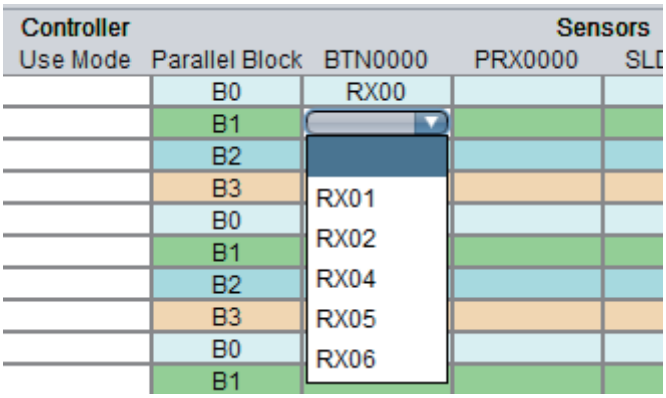


图 80：并行块编辑

7.3.3.4.2.3 传感器列编辑（Sensor Column Edits)

选择传感器列中的某一个单元以显示该传感器的所有未连接端口（这些端口在规则上允许被分配给该控制器引脚）的列表（见图 81）。当传感器端口已被分配时，Unconnected 和 Time Cycle 列表动态地更新。



7.3.3.4.3 以图形的方式建立手动连接

控制器和传感器之间的连接可通过点击传感器上的端口并拖曳至控制器上的指定的端口来建立。

注： Options ➞ Display Mode 菜单项用来显示该连接。

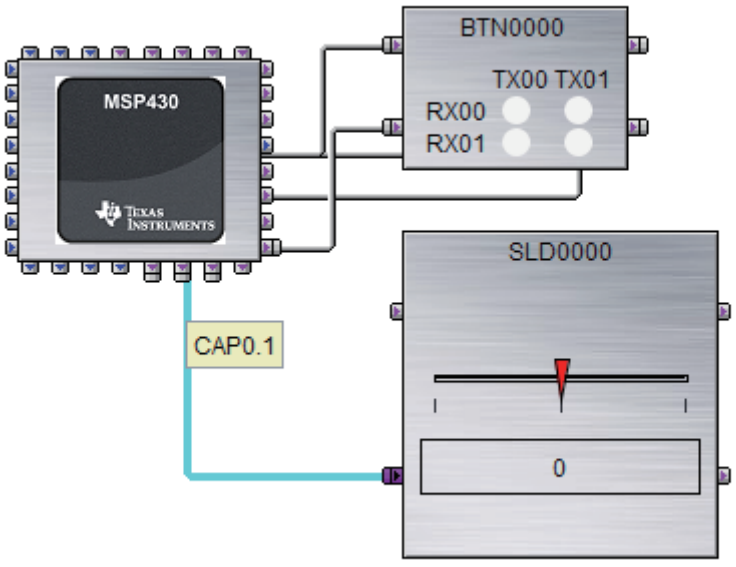


图 82：连接显示

7.3.3.5 源代码生成

当所有的传感器端口皆已连接至控制器时，可生成用于该设计的源代码。

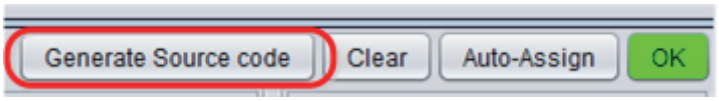


图 83：Generate Source Code 按钮

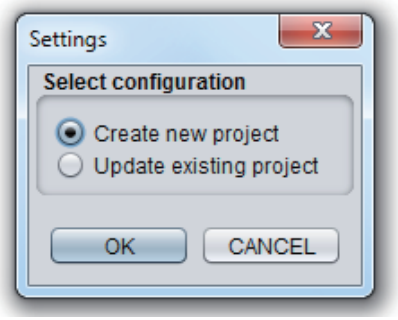


图 84：选择配置 (Select Configuration)

7.3.3.5.1 创建新项目 (Create New Project)

该选项创建一个包含设计定义源文件、关联代码源文件以及 CSS 或 IAR 项目文件的完整项目。

7.3.3.5.2 更新现有项目 (Update Existing Project)

该选项仅更新设计定义源文件。采用该选项来更新一个已经被导入某 IDE 的项目。

7.3.3.6 控制器目标数据显示器 (Controller Target Data Displays)

7.3.3.6.1 控制器通道条形图

见 7.3.5.1 节，通道条形图。

7.3.3.6.2 控制器通道表 (Controller Channel Table)

见 7.3.5.3 节，通道列表。

7.3.3.7 控制器转换控制参数 (Controller Conversion Control Parameters)

在 Conversion Control 选项卡中（见图 85），点击一个参数可以在 Description 视图窗格中显示该参数的说明。

必须启动目标通信才能将数值读取和写入到目标板。

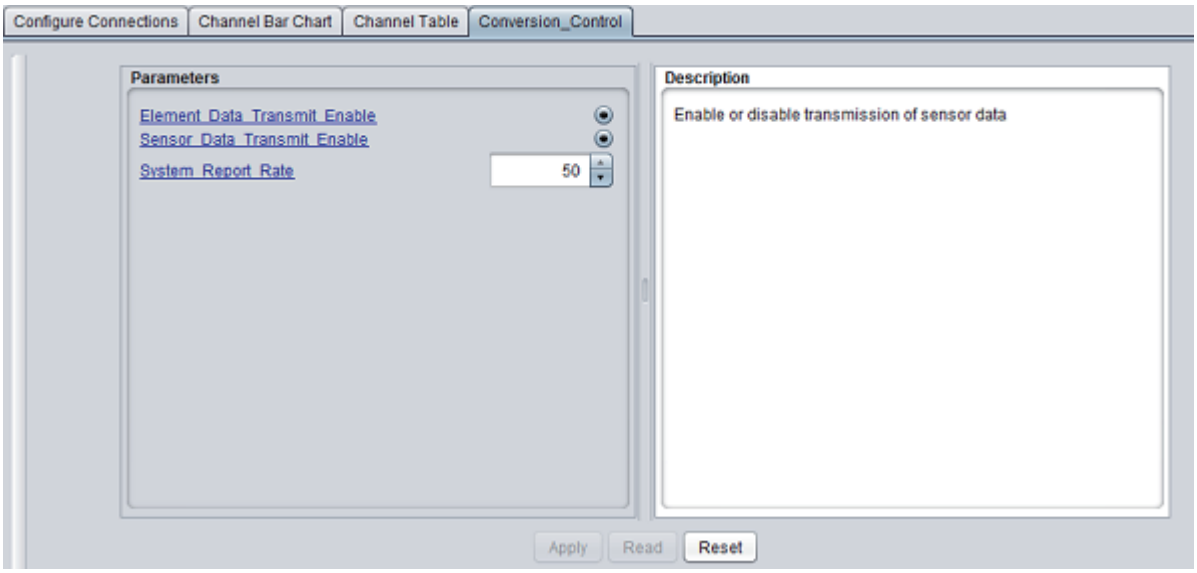


图 85：Conversion Control 选项卡

7.3.3.8 控制器扫描时间估算器 (Controller Scan Time Estimator)

扫描时间估算器显示测量每个传感器的标称时间量，其相对于总扫描周期来表示（见图 86）。总扫描周期可由 System Report Rate 参数来控制。

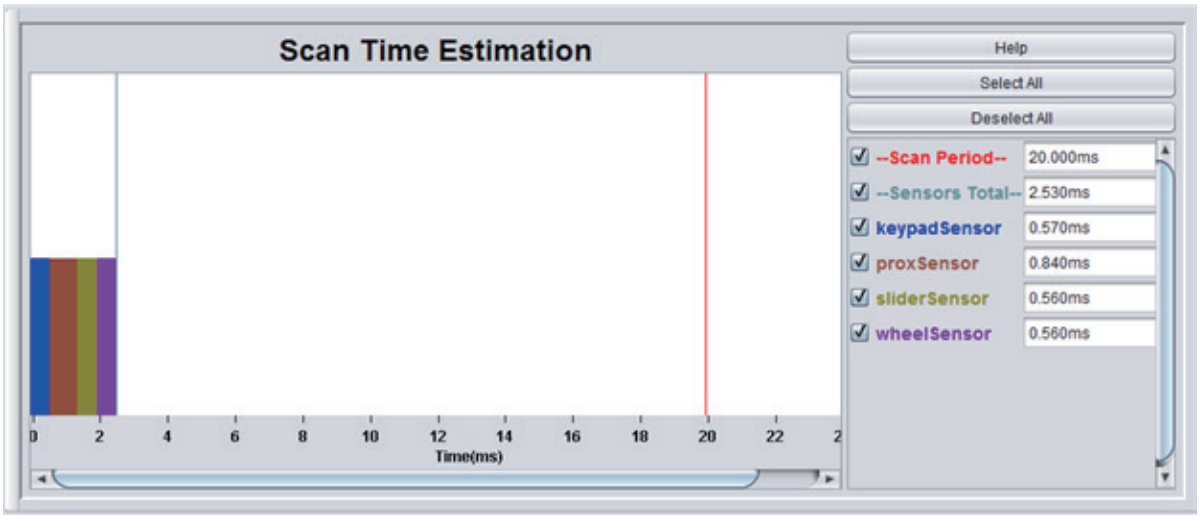


图 86：扫描时间估算器

7.3.4 传感器属性

双击一个传感器对象可以显示属性对话框（见图 87）。

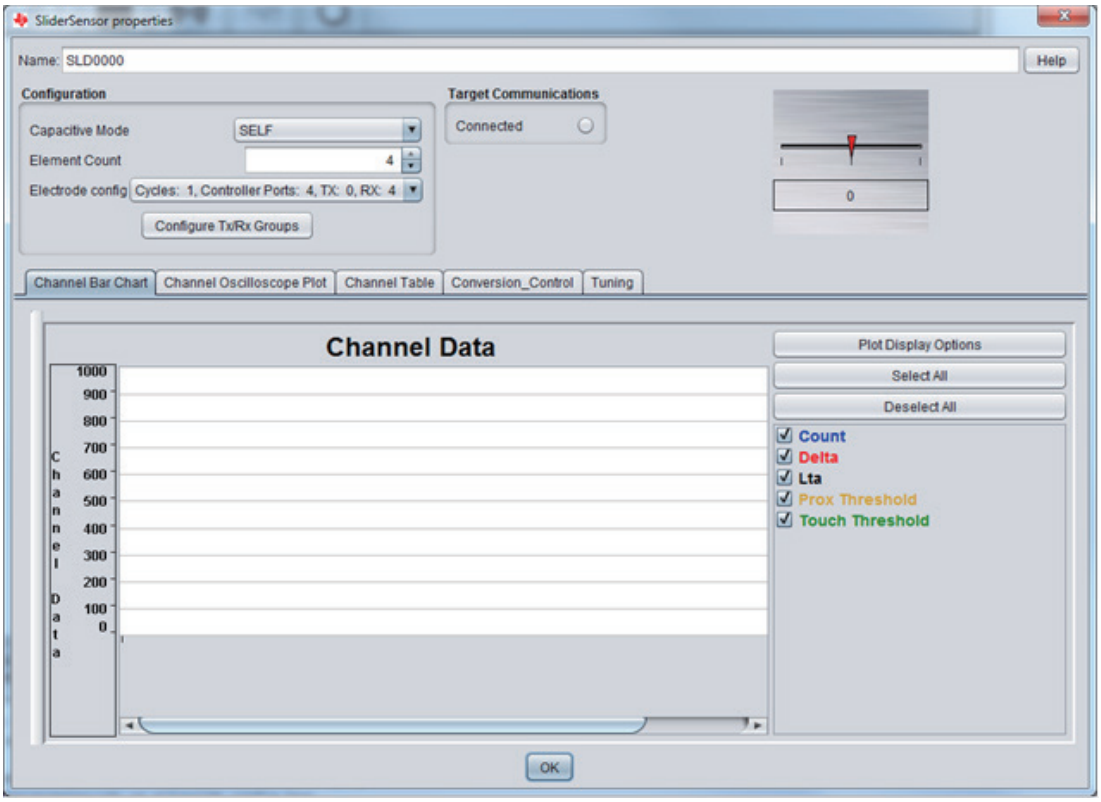


图 87：传感器属性对话框

在传感器对象属性对话框可以执行以下操作：

- 配置该传感器
- 调优该传感器
- 启动目标通信并查看传感器数据

7.3.4.1 位置显示器 (Position Display)

属性对话框可以显示传感器的位置，该信息仅限于特定的传感器类型。每当 Design Center 目标通信处于运行状态时，该显示器工作。

按键组传感器

- 按键组中的每个按键被显示在由传感器的 TX/RX 端口索引的网格列表中（见图 88）。
- 白色表示按钮处于未被触摸的状态。
- 黄色表示按钮的接近感应门限已经达到。
- 绿色表示触摸感应门限已经达到并检测到触摸动作。

图 88：按键组传感器位置显示器

滑条传感器

- 红色表示滑条处于未被触摸的状态（见图 89）。
- 绿色表示触摸门限已经达到并检测到触摸动作。
- 滑块位置在显示器上显示，并且在 GUI 的下方显示了数字值。

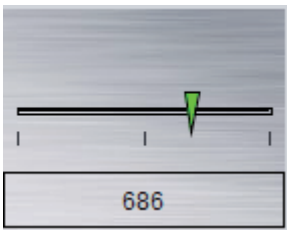


图 89：滑块传感器位置显示器

接近传感器

- 白色表示按钮处于未被触摸的状态（见图 90）。
- 黄色表示接近感应门限已经达到。
- 绿色表示触摸门限已经达到并检测到触摸动作。

图 90：接近传感器位置显示器

滚轮传感器

- 红色表示滚轮处于未被触摸的状态（见图 91）。
- 绿色表示触摸门限已经达到并检测到触摸动作。
- 滚轮位置在显示器上显示，并且在 GUI 的下方显示了数字值。

图 91：滚轮传感器位置显示器

7.3.4.2 传感器配置

传感器配置通过属性对话框来设定（见图 92）。

图 92：传感器配置

7.3.4.2.1 电容模式 (Capacitive Mode)

规定传感器的电容模式类型，选择包括：

- 互电容模式
- 自电容模式

7.3.4.2.2 元件数量 (Element Count)

规定用于传感器的元件数量。

7.3.4.2.3 电极配置 (Electrode Configuration)

根据规定的元件数量，Design Center 计算可能的 TX/RX 端口配置以及每种配置所需的 time cycles 的数量。Design Center 默认采取最小 time cycles 数量的配置。

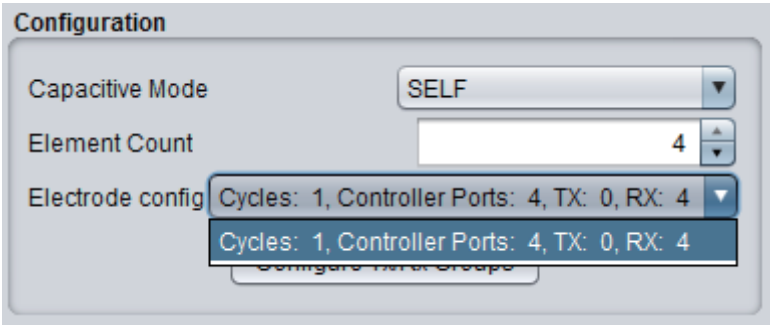


图 93：电极配置

7.3.4.2.4 配置 Tx 和 Rx 分组

在某些场合中，可能需要改变传感器元件与 TX 和 RX 端口的默认分配方式。选择 Configure TX/RX Groups 以显示一个用来变更分配映射的对话框。

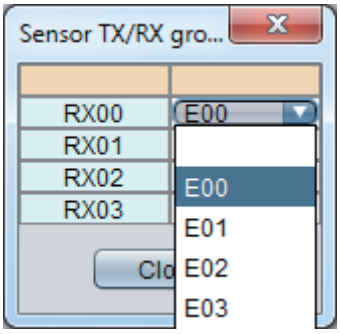


图 94：配置分组

7.3.4.3 从传感器属性对话框启动目标通信

Connected 选项， 该按钮等效于 Communications ➞ Connect/Disconnect 菜单项。

图 95：启用目标板通信

7.3.4.4 传感器目标数据显示器

7.3.4.4.1 传感器通道条形图

见 7.3.5.1 节。

7.3.4.4.2 传感器波形曲线图 (Sensor Oscilloscope Plot)

见 7.3.5.3 节。

7.3.4.4.3 传感器通道列表

见 7.3.5.3 节。

7.3.4.5 调优传感器性能

传感器参数通过 Tuning 和 Conversion Control 选项卡来设定。

当生成用于设计的源代码文件时，传感器参数值被写入相应的结构体。这些参数值在新的源代码经过编译并下载到目标板之后生效。

当在 Design Center 中启动了目标板通信时，可以将参数值实时地读取和直接写入目标板，实现了传感器性能的实时调优，并能够极大地减少实现最优系统性能所需要的时间。

注： 目标板被复位后，任何实时加载的数值都恢复原始值（除非新的源代码被生成、编译和下载至目标板）。

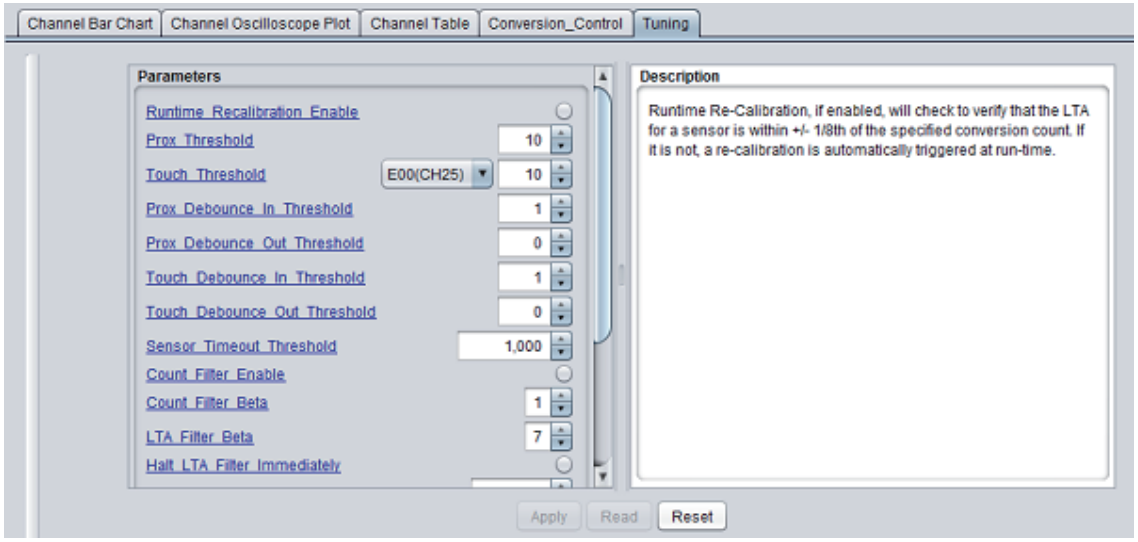


图 96：调优参数

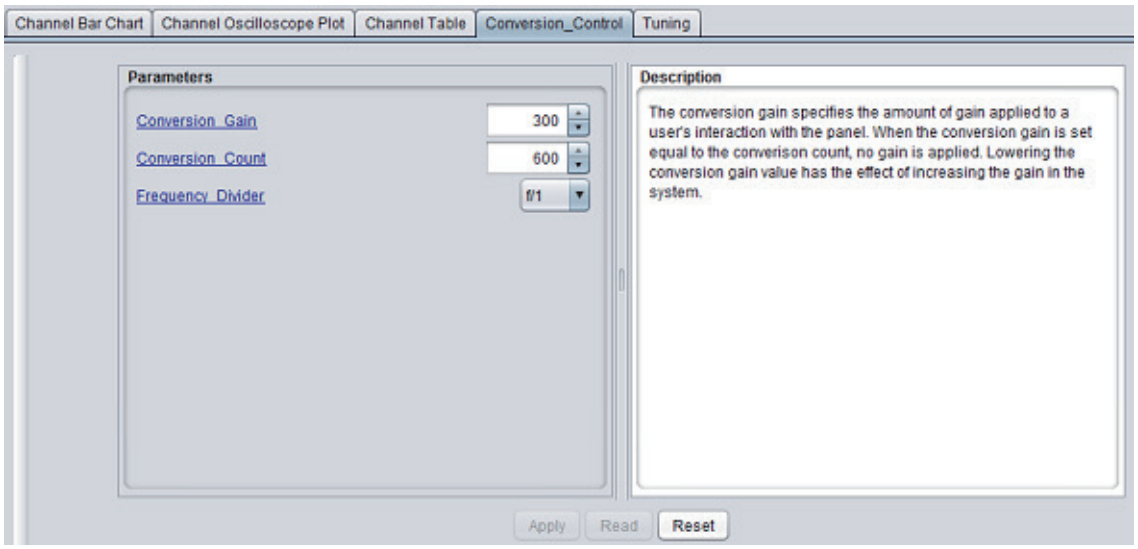


图 97：转换控制参数

7.3.5 显示目标板数据

该 Design Center 提供了丰富的工具组合，以查看来自目标板的传感器数据。

- 通道条形图 (Channel Bar Chart) 提供了用于显示与测量计数有关的数值和触摸 / 接近门限之间互动的简易方法。
- 波形曲线图 (Oscilloscope Plot) 提供了察看传感器数据测量结果历史轨迹的方法。
- 通道列表 (Channel Table) 提供了通道条形图中数据的表格显示 (tabular rendering) 以及针对每个传感器元件的传感器与控制器映射。
- 放大 / 缩小 / 全变焦
  - 如需放大，则点击和往下拉、并向右拖曳以划定放大区域
  - 如需缩小，则点击和往上拉、并向左拖曳以定义缩小量
  - 点击鼠标右键执行完整的缩小
- Plot Display 选项显示了改变曲线图显示的选项的对话框（见图 98）。

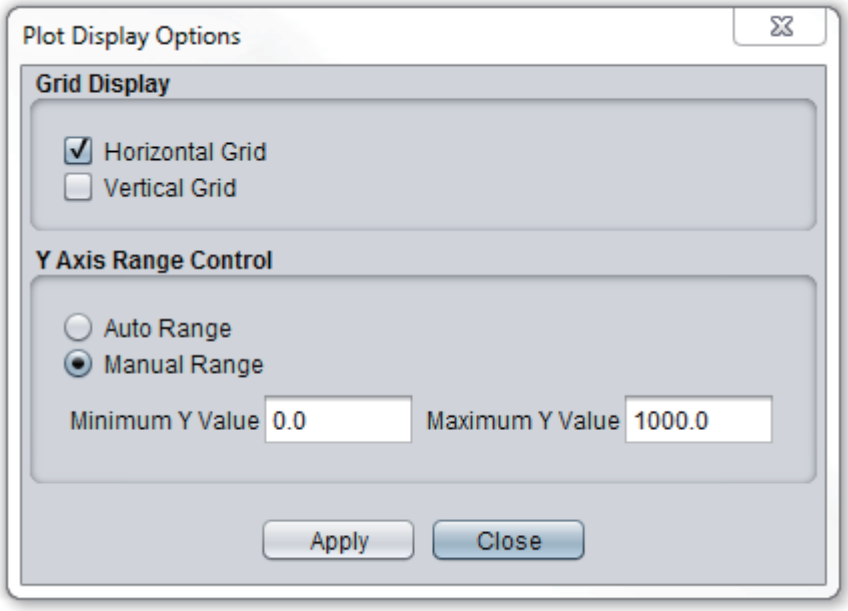


图 98：Plot Display 选项

7.3.5.1 通道条形图 (Channel Bar Chart)

通道条形图（见图 99）显示了每个通道的下列数据：

- 计数
- 变化量
- 接近感应门限
- 触摸门限

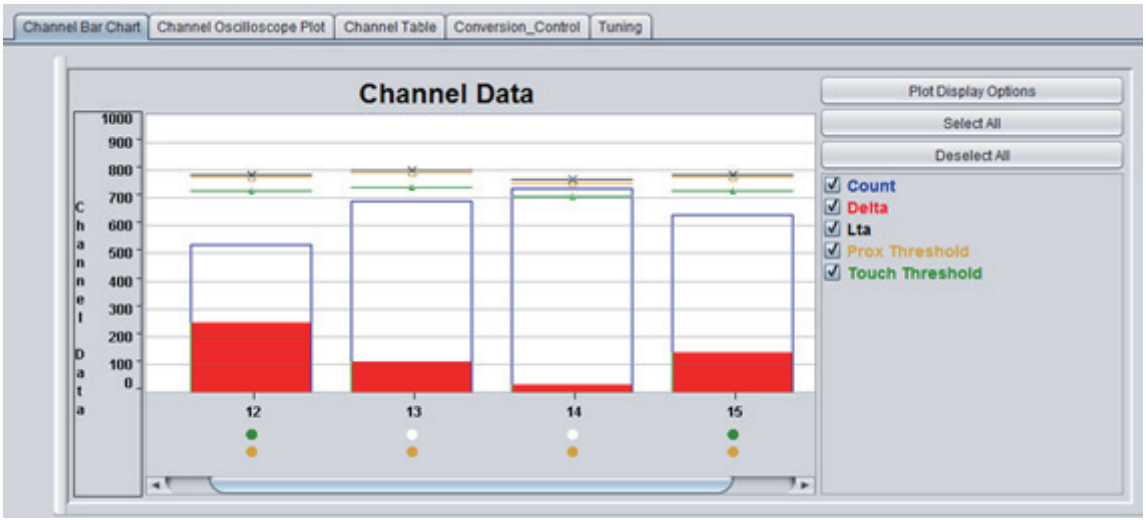


图 99：Channel Bar Chart

7.3.5.2 波形曲线图 (Oscilloscope Plot)

波形曲线图（见图 100）显示了每个通道的下列数据：

- 计数
- 变化量
- 接近感应状态 — 数值 100 表示接近感应门限 (Proximity Threshold) 已经达到
- 触摸状态 — 数值 100 表示触摸门限 (Touch Threshold) 已经达到

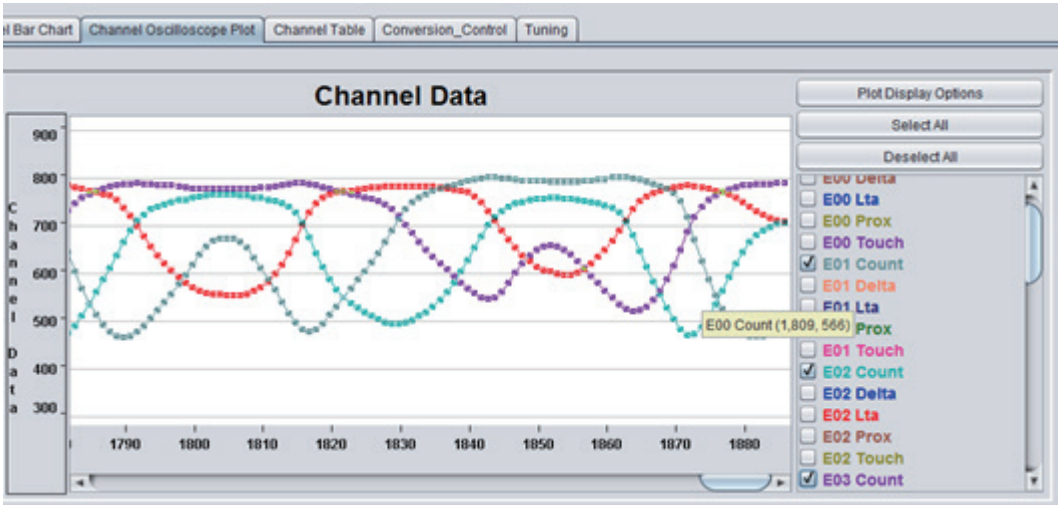


图 100：示波器曲线图

7.3.5.3 通道列表 (Channel Table)

通道列表（见图 101）显示了每个通道的下列数据：

- 连接数据
  - 通道
  - 传感器名称
  - 元件
  - TX
  - RX
  - 时间周期 (Time Cycle)
  - 控制器端口
  - 并行检测块 (Parallel Sense Block)
- 测量数据
  - 计数
  - 变化量
  - 接近感应门限
  - 触摸门限
  - 接近感应状态：黄色表示接近感应门限已经达到。
  - 触摸状态：绿色表示触摸门限已经达到。

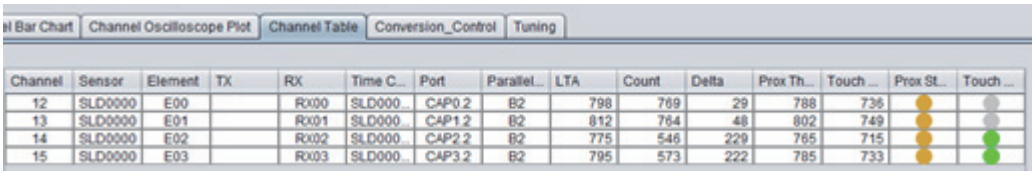


图 101：通道表

7.4 下载和运行生成的项目

7.4.1 导入或打开项目

7.4.1.1 Code Composer Studio

如需导入一个由 Design Center 生成的 CCS 项目，则：

1. 启动 CCS

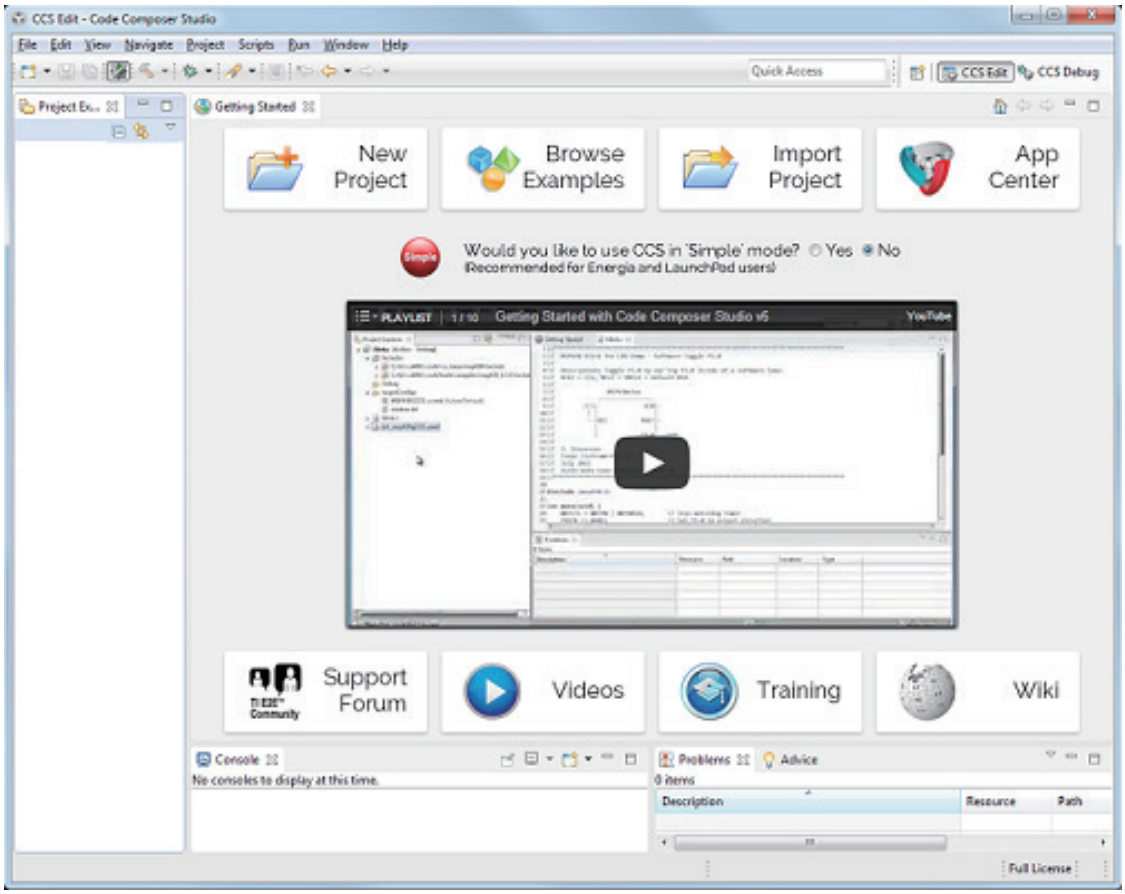


图 102：启动 CCS

2. 选择 Project → Import CCS Projects

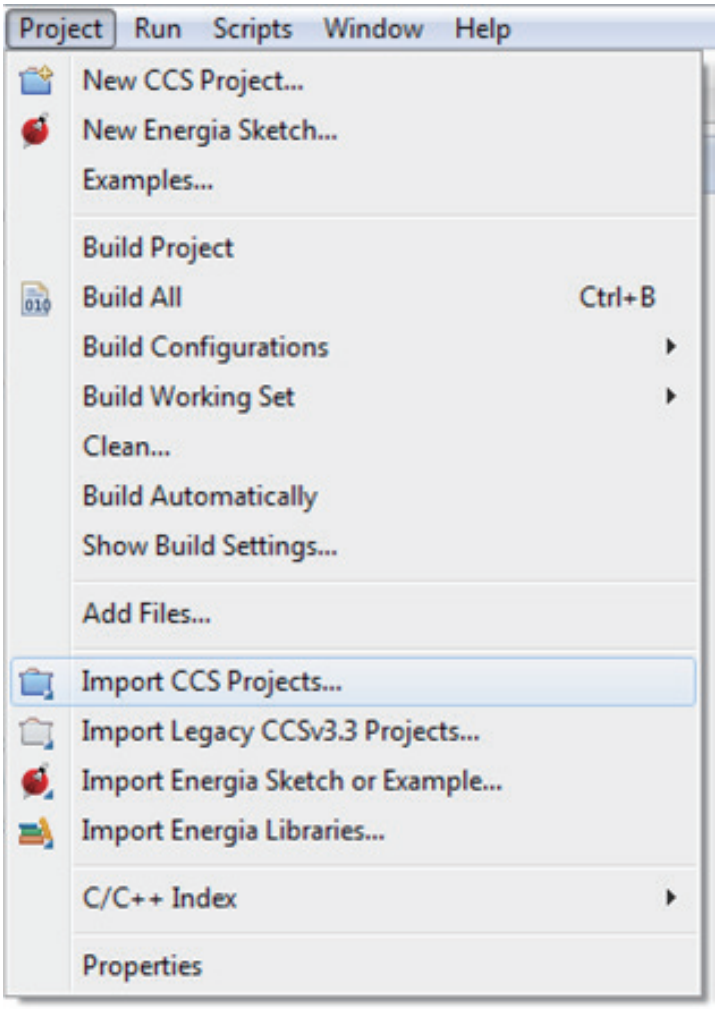


图 103：导入 CCS 项目

3. 提供生成的项目源的位置并选择项目

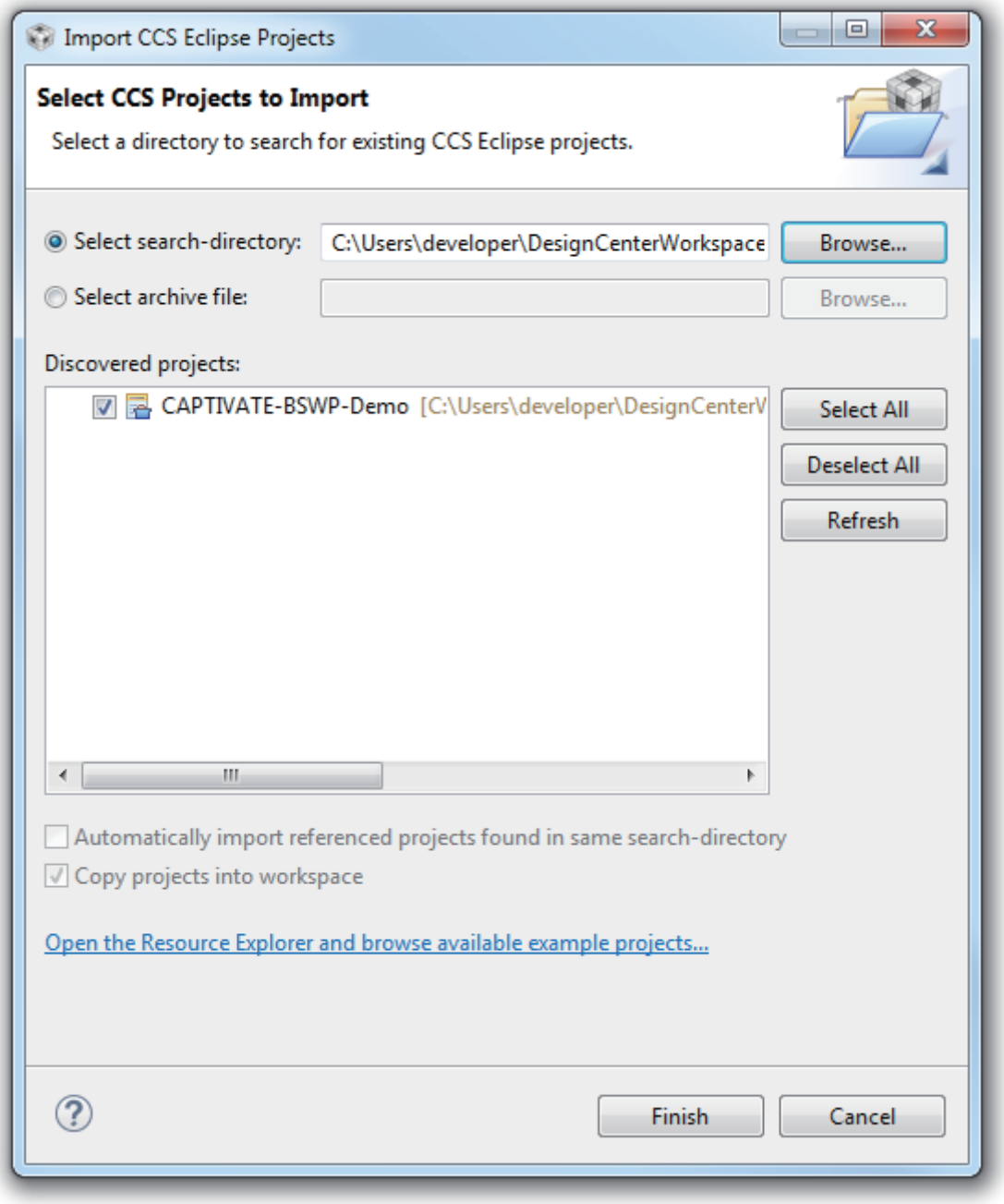


图 104：选择 CCS 项目以导入

4. 点击 Finish
5. 编译代码并将其下载到目标板 MSP430 上
   (a) 点击调试 (debug) 按钮（见图 105）。

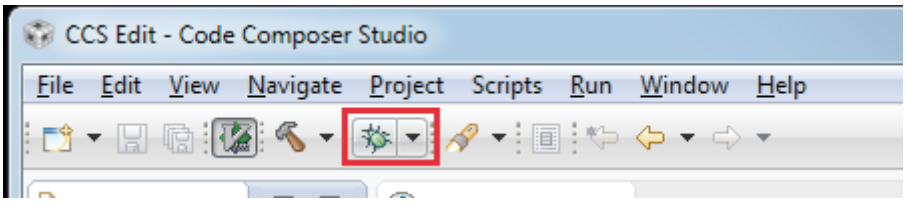


图 105：调试按钮

(b) 点击运行 (run) 按钮（见图 106）。

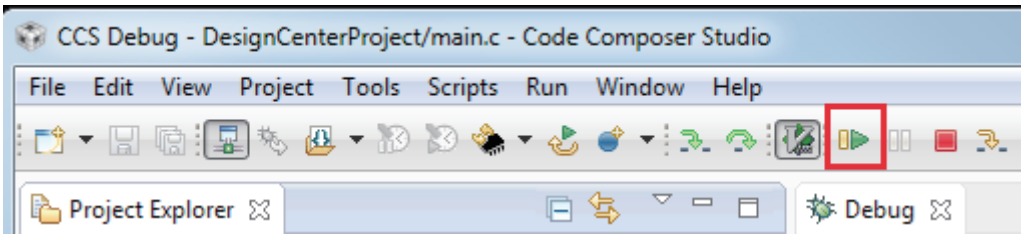


图 106：运行按钮

(c) 点击停止 (stop) 按钮以停止调试程序（见图 107）。

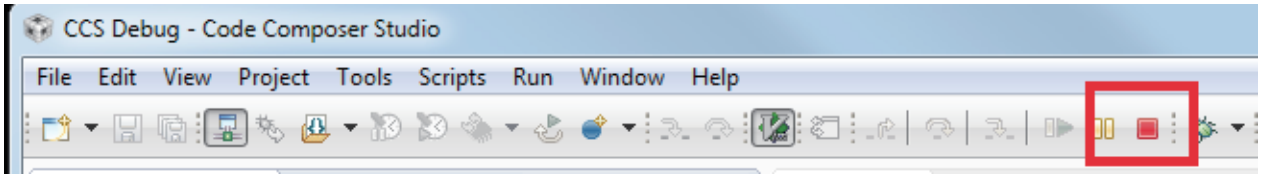


图 107：停止按钮

7.4.1.2 IAR Embedded Workbench

如需打开一个由 Design Center 生成的 IAR 项目，则：

1. 启动 IAR（见图 108）。

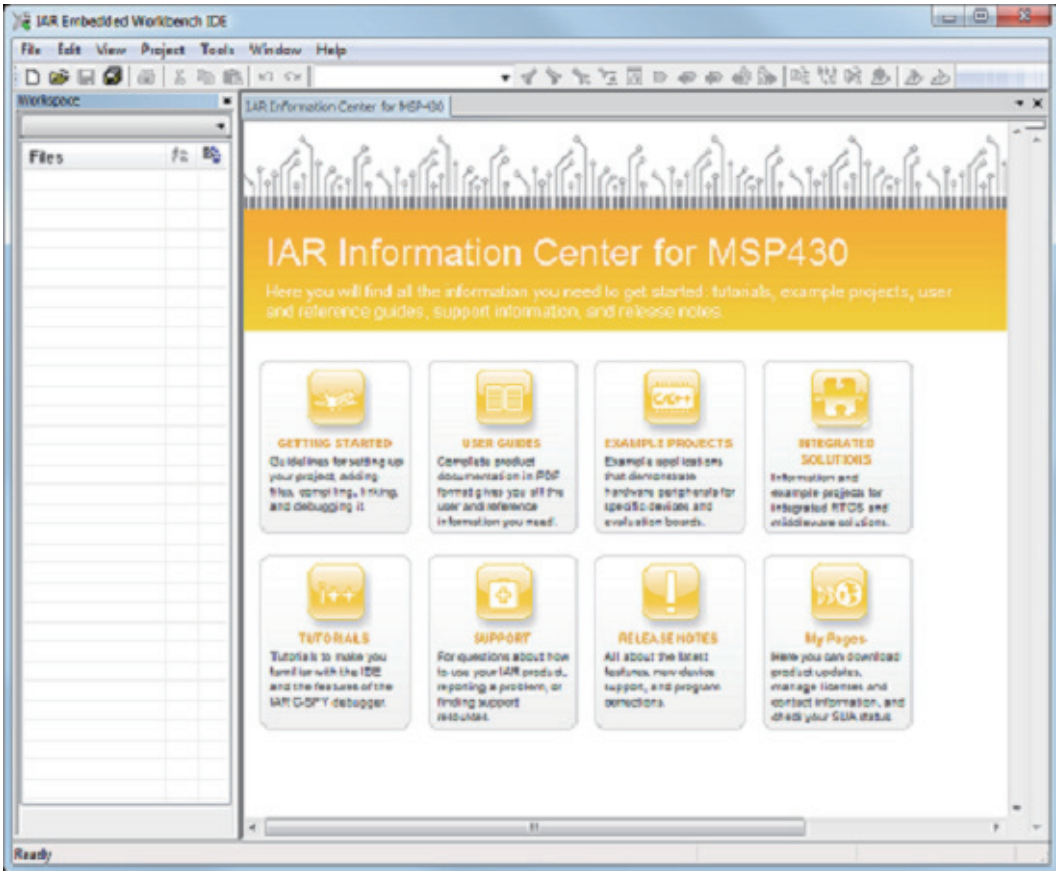


图 108：IAR 嵌入式工作平台 IDE

2. 转到 File ➞ Open Workspace 并导航至生成的源目录（见图 109）。

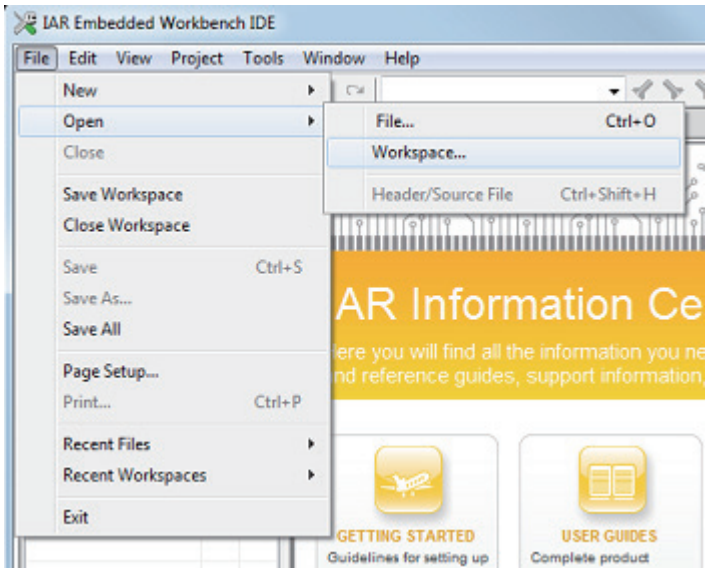


图 109：打开文件

3. 在 IAR/<PROJECT_NAME>.eww 下选择找到的工作空间文件（见图 110）。

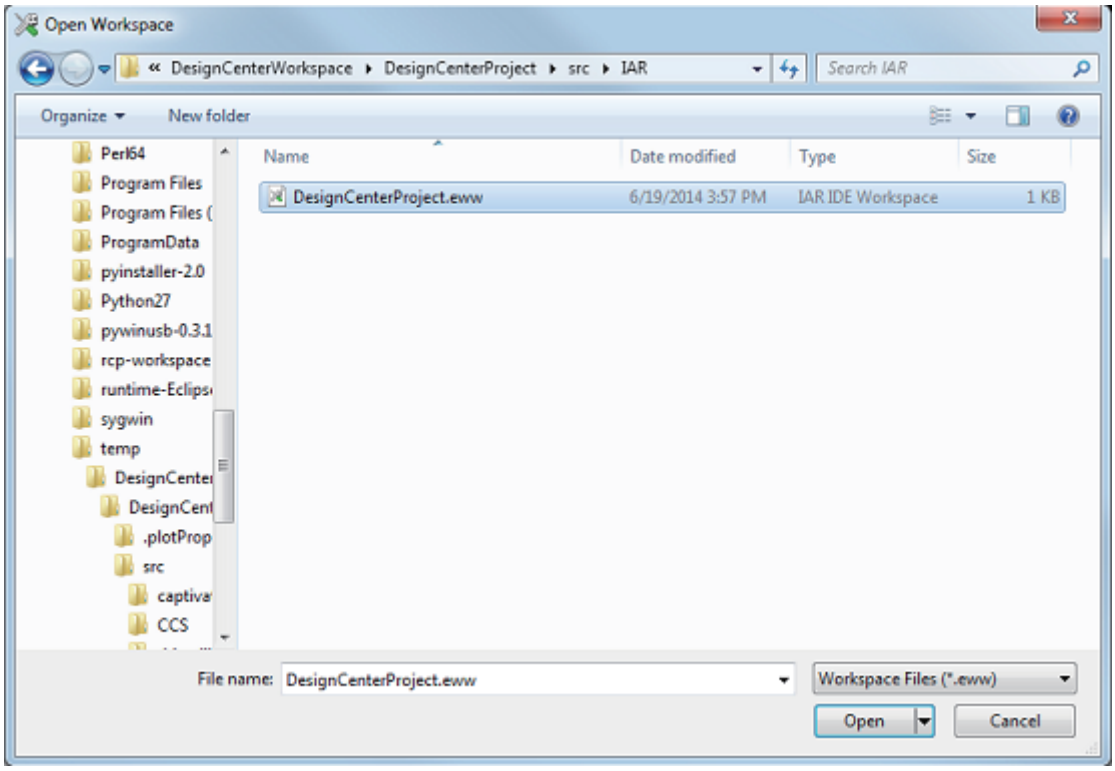


图 110：选择工作空间

4. 编译代码并将其下载到目标板 MSP430 上

(a) 启动调试 (Project → Download And Debug)（见图 111）。

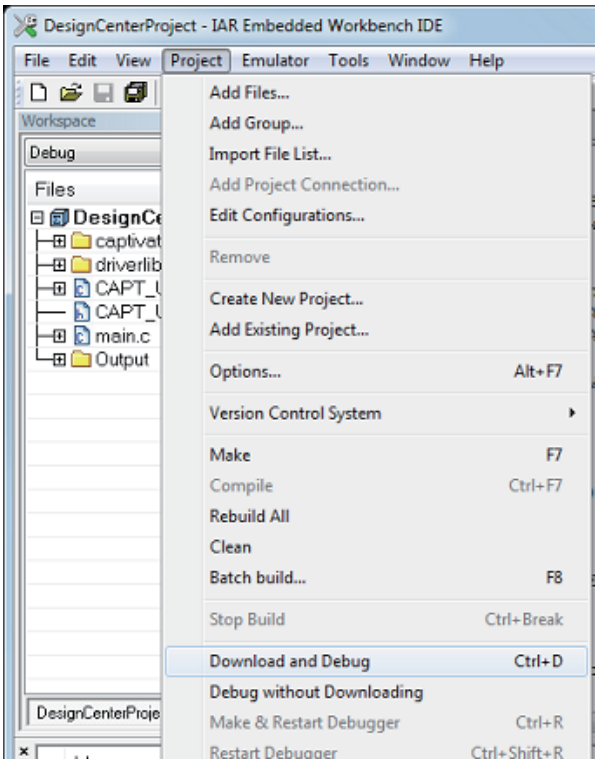


图 111：下载和调试

(b) 启动执行 (Debug → Go)（见图 112）。

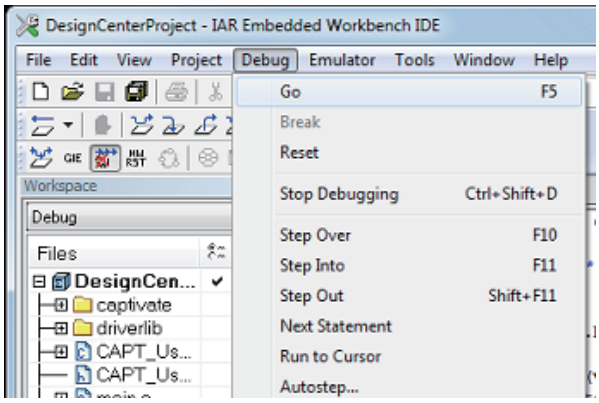


图 112：Go

(c) 停止执行 (Debug ➞ Stop Debugging) 以停止调试程序（见图 113）。

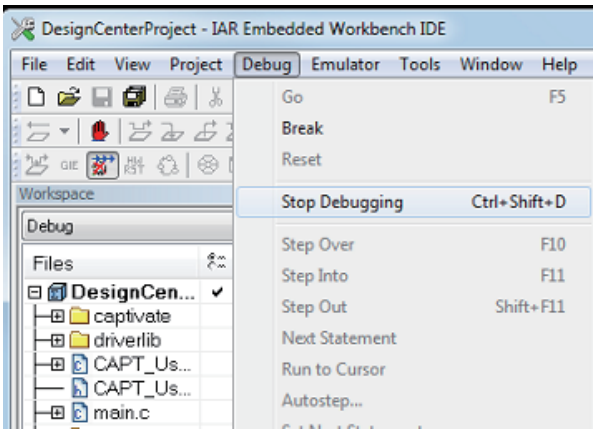


图 113：停止调试

7.4.2 启用 Design Center 中的目标板通信

当代码已经加载到目标板上后，选择 Design Center 中的 Communications ➞ Connect 菜单以启动 HID 通信（见图 114）。

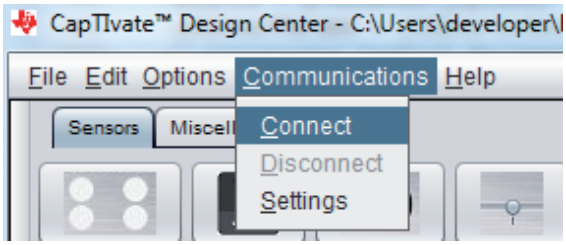


图 114：启用通信

7.5 CapTIvate 设计过程概览

创建任何电容式触摸设计都是一个复杂的过程。典型的步骤包括设计一块 PCB、编写大量的代码并进行多次代码修改和测试迭代的传感器调优过程，从而获得期望的性能。CapTIvate 技术硬件设计过程通过实现设计流程的自动化并帮助加快产品的上市进程而缩短了电容式触摸设计的开发周期。

7.5.1 创建新的传感器设计

第一步是采用 CapTIvate Design Center 启动一个的新的电容式触摸例程。当接收到用户的简单拖放输入时，CapTIvate Design Center 根据传感器的数量和类型自动地确定目标 MCU 和传感器之间的最优引脚配置，也可以对那些具有特定布线要求的应用进行手动分配。

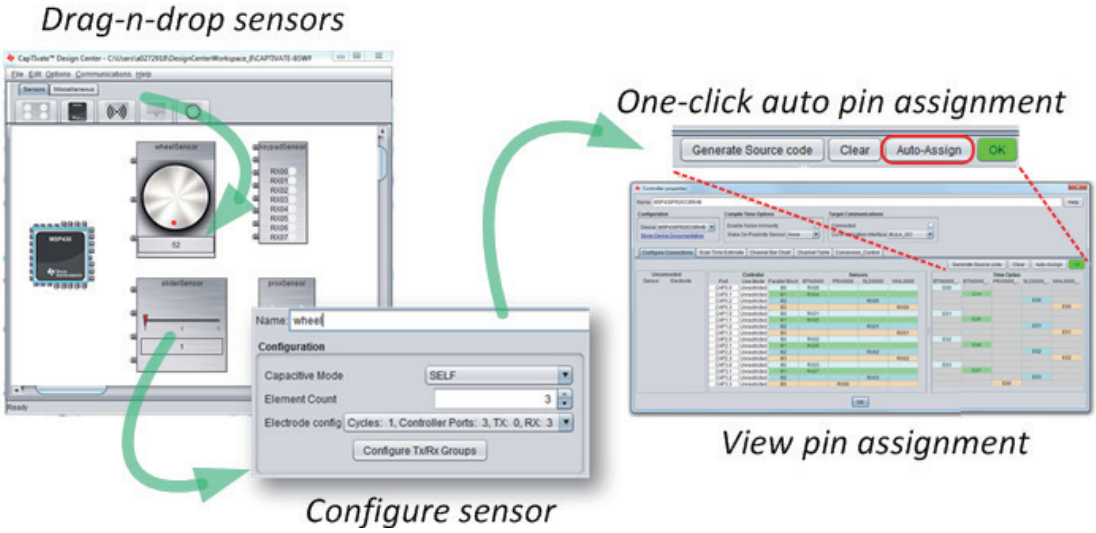
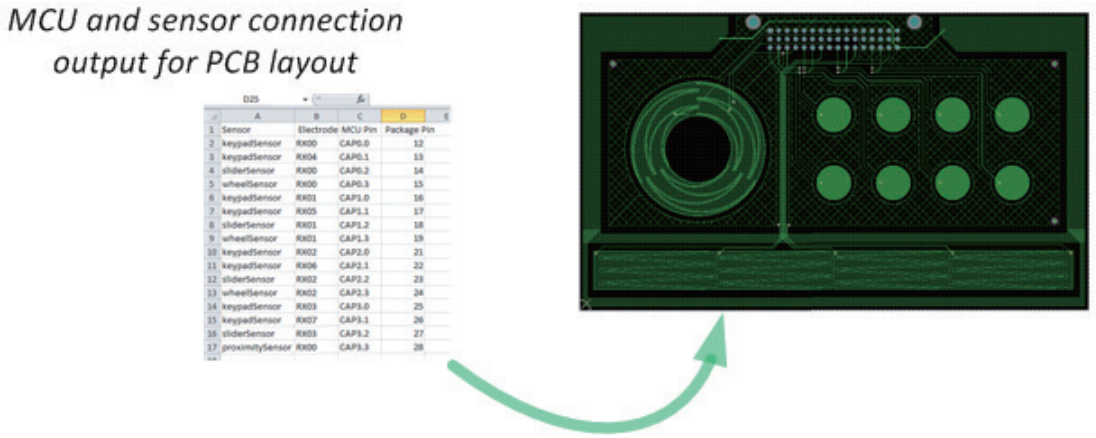


图 115：创建新的传感器设计

7.5.2 PCB 的布局

配置结束后，引脚和传感器的连接信息被方便地导入一个 .csv 文件，可在设计 PCB 的过程中作为一种辅助手段。



	A	B	C	D
1	Sensor	Electrode	MCU Pin	Package Pin
2	keypadSensor	RX00	CAP0.0	12
3	keypadSensor	RX04	CAP0.1	13
4	sliderSensor	RX00	CAP0.2	14
5	wheelSensor	RX00	CAP0.3	15
6	keypadSensor	RX01	CAP1.0	16
7	keypadSensor	RX05	CAP1.1	17
8	sliderSensor	RX01	CAP1.2	18
9	wheelSensor	RX01	CAP1.3	19
10	keypadSensor	RX02	CAP2.0	21
11	keypadSensor	RX06	CAP2.1	22
12	sliderSensor	RX02	CAP2.2	23
13	wheelSensor	RX02	CAP2.3	24
14	keypadSensor	RX03	CAP3.0	25
15	keypadSensor	RX07	CAP3.1	26
16	sliderSensor	RX03	CAP3.2	27
17	proximitySensor	RX00	CAP3.3	28

图 116：PCB 的布局

7.5.3 生成配置和初始项目文件

下一步是生成用于目标 MCU 的项目和传感器配置文件。CapTIvate Design Center 生成一整套针对德州仪器 Code Composer Studio (CCS) 和 IAR Embedded Workbench 的用户初始项目文件。除了应用项目和传感器配置源文件之外，还方便地提供了 CapTIvate 软件库和 MSP430 驱动程序库 (driver-lib) 文件，因为它们是开发周期中所必需的。当需要变更传感器设计时，CapTIvate Design Center 能够容易地生成新的传感器配置源文件，其将自动地替代项目中的前一个配置源文件。

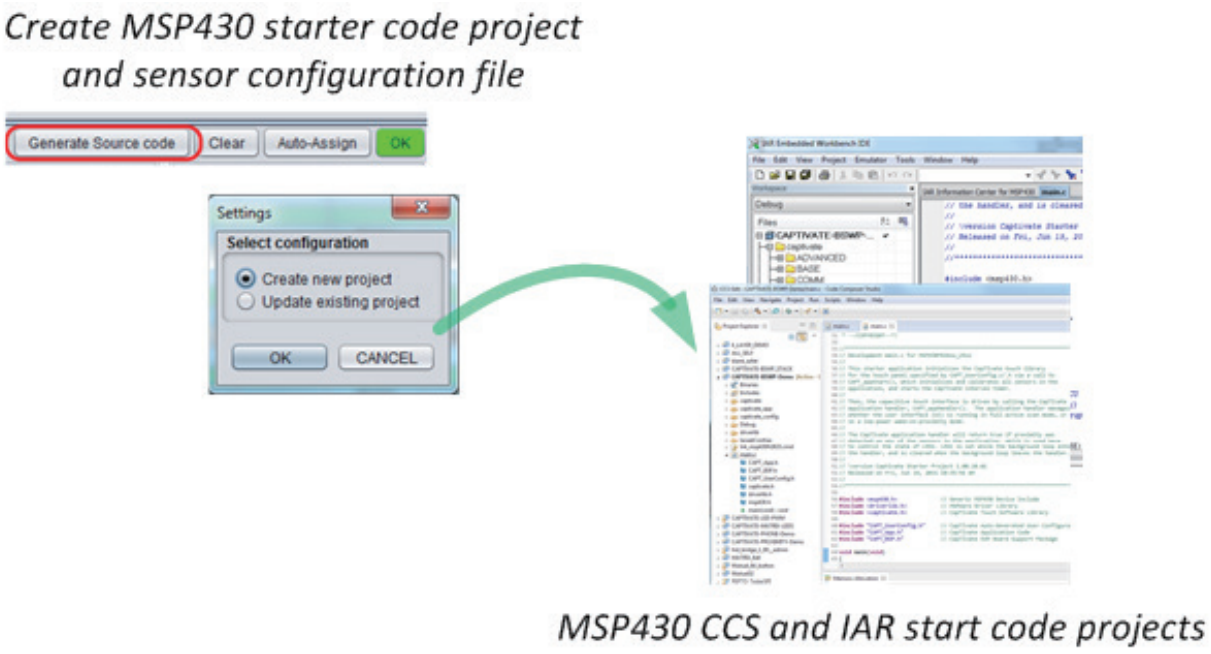


图 117：生成配置和起始项目文件

7.5.4 目标 MCU 的编程

在该步骤中，利用由 CapTIvate Design Center 生成的初始项目代码和传感器配置源文件对 MCU 进行编程。执行编程操作的具体步骤取决于开发环境，不过，由于所有的应用项目文件均已由 CapTIvate Design Center 创建，因此目标 MCU 无须编写任何代码就做好了运行首个电容式触摸应用程序的准备。

7.5.5 调优传感器

在这一步中，针对期望的触摸和感应进行传感器的调优。此时，PCB 已制造完成并做好了测试和调优的准备。当 MCU 运行时，传感器被扫描，实时数据和状态信息被传输至 CapTIvate Design Center。CapTIvate Design Center 具有几个不同的视图，这些视图为显示传感器响应给予了视觉帮助，并允许用户实时地修改传感器调优参数，从而提供了即时反馈。当实现了期望的触摸和感觉时，即可创建最终的传感器配置并将其写入目标 MCU。

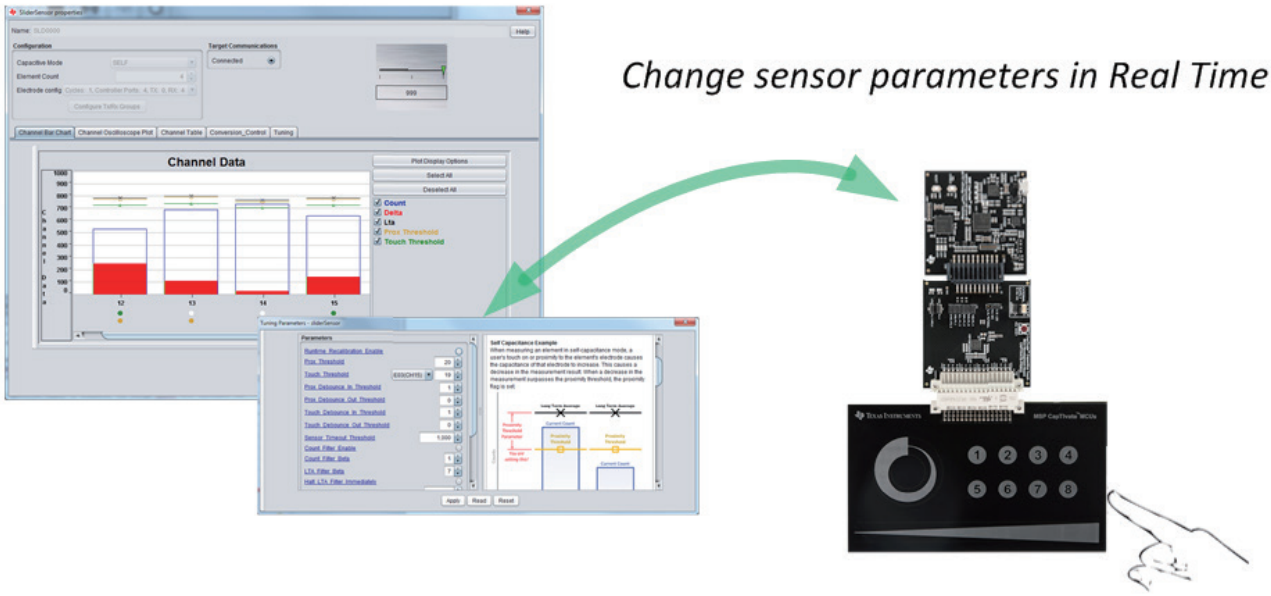


图 118：调优传感器

7.5.6 生成最终配置

最后一步是生成最终的传感器配置。在传感器调优之后，CapTIvate Design Center 生成更新的传感器配置（用户写入其目标 MCU）。目标 MCU ROM 中的 CapTIvate 软件库自动校准特性可确保性能随着时间的推移以及在产品与产品之间是相同的。通过采用上面概述的步骤，大幅度地缩减了最终产品的开发时间。

祝贺您！您完成了设计工作并为产品投放市场做好了准备。

8 器件家族

关于 CapTIvate™ 器件家族的介绍，请参照英文最新版 CapTIvate™ Technology Guide 中相关章节的介绍。

对于 CapTIvate™ Technology Guide v1.40.00.00，请参照章节“Device Family”。

9 软件库

9.1 引言

CapTIvate 电容式触摸软件库提供了支持自电容和互电容传感器所需的一切资源，旨在帮助用户缩短应用程序开发过程。

9.1.1 概述

该软件库包含了触摸、通信和传感器管理特性等功能。触摸功能不仅用于按键、滑条和滚轮的高级传感器处理，而且提供对 CapTIvate 外设的硬件直接访问。通信功能包含了 I^2C 和 UART 串行驱动程序的驱动，提供了支持 FR26xx 和 FR25xx 微控制器在传感器开发过程中将传感器数据发送至 CapTIvate Design Center 的通信协议。软件库管理器则简化了传感器测量、校准和通信功能。

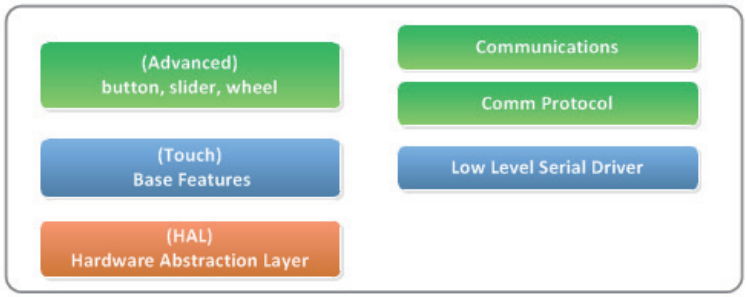


图 119：电容式触摸库

9.1.2 采用 CapTIvate 库的优势

- 简化传感器配置、测量、处理和通信
- 缩短应用程序开发周期

9.1.3 特性

- 在 FR26xx 和 FR25xx 的 ROM 中集成
- 简单的 API 接口
  - 使用该软件库仅需使用三个 API 接口调用
    - CAPT_initUI()
    - CAPT_calibrateUI()
    - CAPT_updateUI()
    - 通过回调机制向应用程序发送传感器更新的通知
- 触摸功能
  - 测量和报告过程是自动的
  - 支持 CapTIvate 硬件触摸唤醒 (wake-on-touch) 状态机
  - 具有触摸防抖动功能和接近触摸检测
  - 用于实现快速求平均和高分辨率的 IIR 数据滤波器
  - 采用内置 32 位硬件乘法器的 IQ Math
  - 按键处理
    - 提供了用于标示主要触摸按键的单一键值
    - 支持多达 64 个按键

- 滑条 / 滚轮处理
  - 支持 3 ～ 12 个元件
  - 高达 16 位的分辨率（视具体应用而定）
  - 滚轮位置准确度为 (±1)° （0° 至 360°）
  - 通信协议
- 支持传感器数据和命令
- 可以使用 CapTIvate™ Design Center 或主机处理器

- 串行驱动程序
  - 支持 I^2C 和 UART
- 软件库管理器
  - 管理传感器扫描和通信操作
- 利用 CapTIvate 特性以实现更好的性能和更低的功耗
  - 触摸唤醒状态机
  - 专用振荡器
  - 专用定时器

9.2 库文件结构

下面列出了 CapTIvate 软件库的文件结构。粗体标注的文件表示那些未在预编译库中提供而作为库源代码提供的组件。

- Advanced
  - CAPT_Buttons.h
  - CAPT_Slider.h
  - CAPT_EMC.h
  - **CAPT_Manger.c**
  - CAPT_Manger.h
- Base
  - CAPT_HAL.h
  - **CAPT_ISR.c**
  - CAPT_Touch.h
  - CAPT_Type.h
  - rom_captivate.h
  - rom_map_captivate.h
- COMM
  - CAPT_ByteQueue.h
  - CAPT_CommConfig.h
  - **CAPT_Interface.c**
  - CAPT_Interface.h
  - CAPT_PingPongBuffer.h
  - CAPT_Protocol.h
  - Serial Drivers

- **FunctionTimer.c**
- FunctionTimer.h
- FunctionTimer_Definitions.h
- **I2CSlave.c**
- I2CSlave.h
- I2CSlave_Definitions.h
- **UART.c**
- UART.h
- UART.Definitions.h

- captivate.h
- captivate.lib
- captivate.r43

9.3 软件库 API

用于 MSP430FRxx 系列的软件库 API 作为一个单独的文件提供。

9.4 用户指南 How-To

软件库的详细使用指南请参照英文最新版 CapTIvate™ Technology Guide 中相关章节的介绍。

对于 CapTIvate™ Technology Guide v1.40.00.00，请参照章节“Software Library->How-To”。

9.5 软件栈

该软件库专为在用户的应用程序和 CapTIvate 外设之间提供桥接而设计。在最低限度下，应用程序仅需实现与软件库中最上层的 Advanced 层的接口。

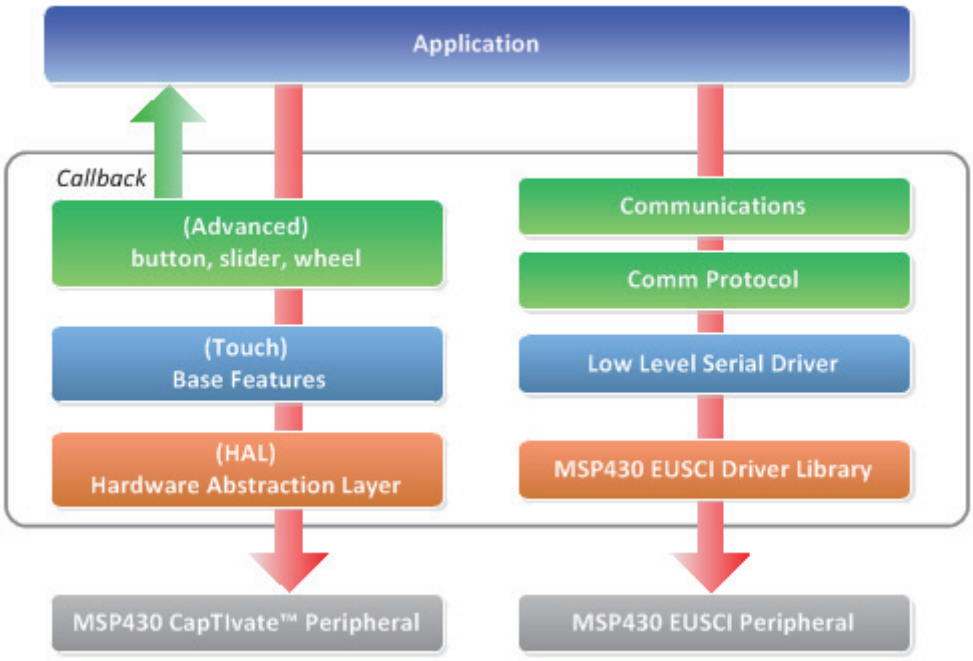


图 123：软件栈

9.5.1 Advanced 层

Advanced 层（高级层）函数提供按键、滑条和滚轮功能，针对每种传感器类型处理来自 Touch 层的结果（取决于应用中的传感器类型）。Advanced 层通过用户定义的回调函数来更新应用层。详情请见“回调”部分。

- 按键
  - 在一组按键（包括一个或多个按键）中标示出最主要触摸的按键
    - 如果触摸了多个按键传感器中一个以上的按钮，则应用程序需要负责检查其他的按键
  - 多达 256 个按键（取决于相应的器件）
    - MSP430FR26xx、FR25xx（32 引脚器件）支持 16 个自电容模式按键或 64 个互电容模式按键
- 滑条或滚轮
  - 提供经过滤波后的位置信息
  - 分辨率高达 13 位
  - 支持 3 ～ 12 元件传感器
  - (±1)° 准确度，360° （滚轮）

9.5.2 Touch 层

Touch 层（触摸层）函数执行原始传感器数据的处理和滤波，以及防抖和门限检测。结果被提供给 Advanced 层中的组件。

- 触摸和接近检测
- 触摸和接近防抖
- 长期基平均准线求 (LTA)
- IIR 和快速 IIR 测量及 LTA 滤波
- 传感器校准

9.5.3 HAL 层

HAL（硬件抽象层）软件功能可直接访问和操控 CapTIvate 模拟及硬件状态机。在典型的触摸应用中，不必直接从应用程序调用 HAL 函数。相反，HAL 函数是从软件库中较高级别的层调用，或者在初始化期过程中从软件库管理器调用。

9.5.4 软件库传感器管理器

为了帮助实现传感器管理和定时的抽象化，通常从一个应用程序的 main() 函数内部调用的三个高级函数是该软件库的库传感器管理器的一部分，此管理器负责管理较低级别库函数的执行。这些函数是：

- CAPT_initUI()
- CAPT_calibrateUI()
- CAPT_updateUI()

仅采用这三个函数即可构建一个基本的传感器测量框架，该框架提供传感器初始化、传感器校准和传感器更新。CapTIvate Design Center 可以生成演示这些函数用法的代码项目。

9.6 通信模块

CapTIvate 触摸库包括一个用于将 CapTIvate MCU 连接至外界的通信模块。本节讨论该通信模块的架构、特性和规格，以及从开发到生产的多种应用中如何使用。

本节假设读者熟悉以下内容：

- 作为用户接口的电容式触摸技术
- CapTIvate 生态系统和触摸软件库
- 基本的 MSP 微控制器架构

9.6.1 背景

设计电容式触摸接口是一种迭代过程。系统中的每个传感器必须分别调谐和优化以实现期望的灵敏度和感觉。 PC GUI 与目标 MCU 之间进行实时通信的能力可大幅度地减少调优所需的时间，并能提供更好的调优结果，这是由于内置于 CapTIvate 外设中的功能可以简便快捷地执行以确定最佳的配置。

在设计和调优阶段之后，电容式触摸微控制器在系统中承担两个作用。它可以作为人机接口 (HMI)，仅起到把电容式触摸板分解为可用信息（如触摸 / 无触摸，或一个滑块位置）的作用；或者可以作为主机处理器的双重作用，把诸如监视传感器或控制其他部件等其他功能集成到系统中。在第一种场合中（作为专用 HMI 的场合），控制器将几乎始终需要一个把接口的状态传输至某个其他主机（可以是另一个 MCU 或 MPU）的方法。该接口可能简单到就是一个 GPIO（其在某个按钮被触摸时被置位），或者复杂到是一个具有可寻址参数的完整 I^2C 协议。

9.6.2 概述

CapTIvate 软件库通信模块针对上述的两种需求提供了统一的解决方案。该通信模块是一组分层的固件，具有一个简单的顶层 API，专为通过标准的通用串行接口把 CapTIvate MCU 链接至 CapTIvate Design Center GUI 或其他主机处理器而设计。

在电容式触摸应用中 MCU 负责测量电容式传感器，处理测量结果以解释某种类型的结果，并把该结果传输至本地应用程序或主机处理器。通信模块层实现了该传输部分。下面的小节说明了在典型的 CapTIvate 应用中通信模块是如何与固件程序的其他部分配合工作的。

CapTIvate 通信模块是分层的，并且包含了几种互连的独立功能。这些功能在下面详细介绍。如需使用通信模块，只需要设置配置文件并调用顶层 API 接口。

通信模块包含四层。按照抽象级别的自上而下分别是：

- 接口层 [COMM/CAPT_Interface.c/.h]
  - 接口层实现了顶层 API 接口
- 协议层 [COMM/CAPT_Protocol.c/.h]
  - 协议层实现了 CapTIvate 协议包生成及包解析
- 串行驱动程序层 [COMM/Serial_Drivers/*.c/.h]
  - 串行驱动程序层包含几种可互换的串行驱动程序选项
  - 同一时间有且只有一个选项可以选择
  - 选择和配置在配置文件中进行处理
  - 串行驱动程序的构建基于 MSP430 DriverLib API
  - 提供了 UART 和 I^2C Slave 驱动程序
- 数据结构层 [COMM/CAPT_ByteQueue.c/.h, COMM/CAPT_PingPongBuffer.c/.h]
  - 数据结构层实现了基本的抽象数据类型，例如：先进先出 (FIFO) 队列和一个乒乓缓存（以帮助串行通信）。

9.6.3 接口层

CapTIvate 接口层是顶层通信层。它提供了应用程序所使用的顶层函数调用，并且起到了使协议层（其负责处理包生成和包解析）与串行驱动程序（其实际上把数据移入和移出微控制器）紧密结合的作用。提供的功能在下面的小节中进行说明。对应用模块的所有应用程序访问均应通过接口层。

9.6.3.1 使用通信模块：初始化接口

在启动时通信模块必须由应用程序通过调用 CAPT_initCommInterface() 进行初始化。该顶层初始化函数打开已选的串行驱动程序，并初始化通信所需的队列或缓冲区。

表 9：用于初始化通信的函数

说明	声明
初始化通信模块	extern void CAPT_initCommInterface(tCapTIvateApplication *pApp)

9.6.3.2 使用通信模块：处理输入数据

从主机输入的原始数据由串行驱动程序进行缓存，当应用程序空闲时对数据进行处理。应用程序必须周期性地调用 CAPT_checkForInboundPacket()，以查看是否已经从主机接收到数据包。该函数将在串行驱动程序的接收队列中检查数据包，如果发现了任何数据包，则根据它们的类型对其进行处理。通常，当应用程序可用时，在主循环中调用该函数。如果一个来自主机的数据即将送达，则串行驱动程序退出睡眠模式进入运行模式，这种机制可以被用来唤醒主循环从而调用该函数。

另外，还应当周期性地调用 CAPT_checkForRecalibrationRequest() 函数，以确认由 CAPT_checkForInboundPacket() 接收和处理的数据包是否要求应用程序重新校准用户接口。例如：如果接收到的数据包改变了转换计数，则要求重新校准系统中的传感器。

表 10：用于输入数据的函数

说明	声明
检查一个接收到的数据包，并对其进行处理	extern bool CAPT_checkForInboundPacket(void)
检查重新校准的请求	extern bool CAPT_checkForRecalibrationRequest(void)

9.6.3.3 使用通信模块：写出数据

接口层提供了三种用于把数据传输至主机的顶层结构。下面的三个函数负责处置合适数据包的生成、发送乒乓缓冲器的管理、以及通过串行接口的传输。如果串行外设是可用的（不在忙碌状态），则这些调用是无阻塞的，而且生成的数据包通过串行驱动程序中的中断服务程序传输至主机。

表 11：用于写数据的函数

说明	声明
写入元素数据	extern bool CAPT_writeElementData(uint8_t ui8SensorID)
写入传感器数据	extern bool CAPT_writeSensorData(uint8_t ui8SensorID)
写入通用数据	extern bool CAPT_writeGeneralPurposeData(uint16_t *pData, uint8_t ui8Cnt)

9.6.3.4 编译时间配置

编译时间配置选项在 CAPT_CommConfig.h 文件中设定。可用的编译时间选项说明于下。

9.6.3.4.1 接口选择定义

与其他的定义不同，CAPT_INTERFACE 位于用户配置 (User Confg) 文件 (CAPT_UserConfig.h) 中，用来选择通信模块所构建的接口。如果不需要通信模块，则应设定 CAPT_NO_INTERFACE。否则，应设定相应的通信模式。

注： Options ➞ Display Mode 菜单项用来显示该连接。

表 12：接口选择定义

参数	文件	有效值
CAPT_INTERFACE	CAPT_UserConfig.h	CAPT_NO_INTERFACE__，CAPT_UART_INTERFACE，CAPT_BULKI2C_INTERFACE，CAPT_REGISTERI2C_INTERFACE

9.6.3.4.2 传输缓冲区容量定义

CAPT_TRANSMIT_BUFFER_SIZE 定义传输缓冲区的容量。由于使用了乒乓缓冲，因此分配了该容量的两倍。另外，该容量还应是最大数据包大小的至少两倍，为字节填充留出余地。

表 13：传输缓冲区容量定义

参数	文件	有效值
CAPT_TRANSMIT_BUFFER_SIZE	CAPT_CommConfig.h	无符号整数

9.6.3.4.3 接收队列缓冲区容量定义

CAPT_QUEUE_BUFFER_SIZE 定义接收队列的大小。这是串行驱动程序用来缓冲接收数据（直到该数据被一个对 CAPT_checkForInboundPacket() 的调用所处理为止）的队列 。如果发现数据包正在被丢弃，则应该增加该缓冲区的容量。

表 14：接收队列缓冲区容量定义

参数	文件	有效值
CAPT_QUEUE_BUFFER_SIZE	CAPT_CommConfig.h	无符号整数

9.6.3.4.4 I^2C Slave 串行驱动程序接收缓冲区容量定义

表 15：I^2C Slave 串行驱动程序接收缓冲区容量定义

参数	文件	有效值
CAPT_I2C_RECEIVE_BUFFER_SIZE	CAPT_CommConfig.h	无符号整数

CAPT_I2C_RECEIVE_BUFFER_SIZE 定义了 I^2C Slave 驱动程序所使用的接收缓冲区的容量。该缓冲区容量应至少与预期的 I^2C 总线写任务最大长度一样大。

9.6.3.4.5 寄存器模式中的 I^2C Slave 串行驱动程序缓冲区容量定义

CAPT_I2C_REGISTER_RW_BUFFER_SIZE 定义了当通信接口配置在寄存器 I^2C 模式时 I^2C Slave 驱动程序所使用的缓冲区的容量。该缓冲区容量应至少与预期的 I^2C 总线写任务最大长度一样大。

表 14：接收队列缓冲区容量定义

参数	文件	有效值
CAPT_I2C_REGISTER_RW_BUFFER_SIZE	CAPT_CommConfig.h	无符号整数

9.6.4 协议层

CapTIvate 协议是一种用于发送电容式触摸专属数据的通信规范。它使得运用 MSP430 CapTIvate 技术的微控制器能够与主 PC 上的设计、调试和调优工具进行通信。除了该功能之外，它还能提供一种在较大系统中把一个 CapTIvate MCU 连接至另一个主机 MCU 或 SoC 的机制。本指南讨论该协议本身的详情：数据包类型和数据包结构。

CapTIvate 协议是一种基于数据包的串行消息传递协议。它包括了用于把实时电容式测量数据从一个 CapTIvate 目标 MCU 传递至另一个处理器的规定，以及用于对参数的读和写进行调优的规定。

9.6.4.1 用例

针对该协议的各种不同用例为：

- 电容式触摸开发与调优

  电容式触摸开发和调优需要察看来自触摸板的实时数据，并调节软件参数，以从该触摸屏上的传感器获得期望的响应和感觉。例如：调优一个电容式按键需要察看从微控制器返回的有关该按键的原始数据，并相应地调节门限、防抖和滤波器，以创建一个稳定的用户接口。拥有实时调节所有这些软件参数并察看传感器的能力（无需重新编译代码）是极其强大的，而且减少了开发时间。CapTIvate 协议是利用 CapTIvate Design Center 特别设计的，旨在满足该需要。

- 至主机处理器的接口

  大多数电容式触摸用户接口需要一个驱动触摸板的专用微控制器，其负责与主机处理器通信。CapTIvate 协议的灵活性允许其作为连接一个主机处理器的接口；它能够把触摸状态、接近状态和滑条 / 滚轮的位置传输至主机。

- 现场或在系统调试接口和调优

  由于 CapTIvate 协议支持电容式触摸调优参数的读和写，以及实时数据的传输，因此当与 CapTIvate Design Center PC 工具结合时其有可能在现场用作一种诊断工具。

9.6.4.2 数据包类型的介绍

CapTIvate 协议支持五种数据包类型：传感器数据包、周期数据包、参数数据包、通用数据包和触控板数据包。在电容式触摸系统中，通信链路中有两个端点：目标 MCU 和主机。主机也许是一个 PC 工具或某种类型的嵌入式处理器。传感器数据包、周期数据包、通用数据包和触控板数据包传递正在由目标 MCU 驱动的触摸板的当前的状态信息。这些数据包是单向的，仅从目标 MCU 传输至主机。参数数据包则是双向的，可以从目标板传输至主机或者从主机传输至目标板。

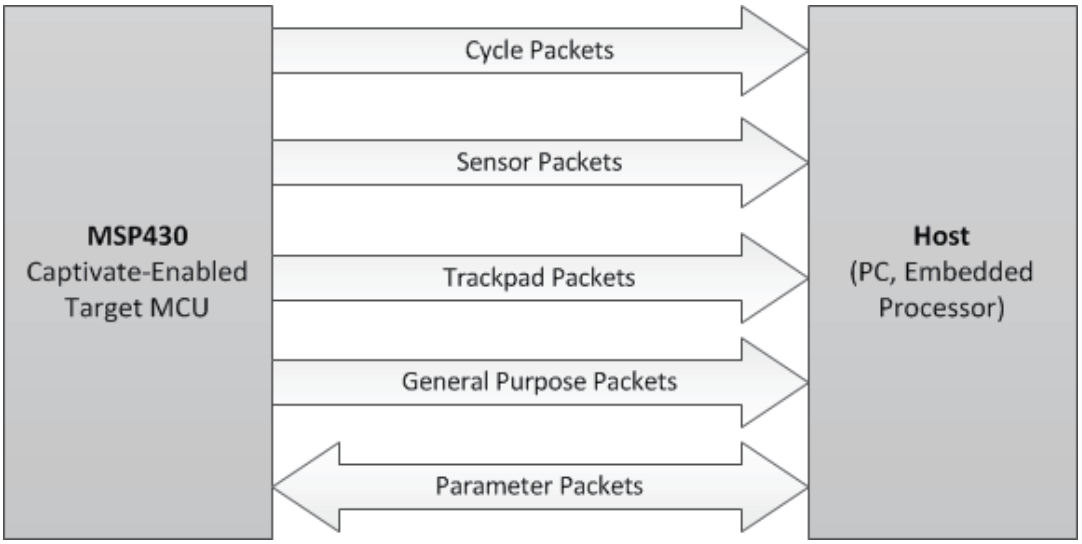


图 124：数据包方向性

9.6.4.2.1 传感器数据包

传感器数据包是从 CapTIvate MCU 至主机的单向数据包，它们提供有关传感器当前状态的信息。传感器状态信息包括诸如主导按键、滑条或滚轮位置、传感器总体接近状态和传感器总体触摸 / 先前触摸状态等。

9.6.4.2.2 周期数据包

周期数据包是从 CapTIvate MCU 至主机的单向数据包。它们提供低级元件信息，比如：元件触摸状态、元件接近状态、元件计数、以及一个周期之内所有元件的计数长期平均值（LTA）。这些数据包通常在调优阶段中使用，在该阶段中希望获得计数值以及设置门限和调谐滤波器的长期平均值的实时视图。

9.6.4.2.3 触控板数据包

触控板数据包是从触控板 MCU 至主机的单向数据包。它们提供触控板上的触摸的 X 和 Y 坐标。

9.6.4.2.4 通用数据包

通用数据包是从 CapTIvate MCU 至主机的单向数据包。它们充当一个通用容器，以发送可被格式化为一个 16 位无符号整数的任何信息。该通道能够作为一种调试工具，用于发送不能适合任何其他数据包类型的信息。在单个数据包中最多可以发送 29 个整数（58 字节）。

9.6.4.2.5 参数数据包

参数数据包是在主机和 CapTIvate MCU 之间的双向数据包。参数数据包允许主机在运行时调节目标板上的任一调优参数。例如：针对一个元件的触摸门限或者某个滑条的分辨率可通过发送适当的参数命令来调节。参数可以读取或写入。从主机至目标 MCU 的参数读和写始终触发回读最新的数值（在写操作的场合中，是在写入之后的数值）。

9.6.4.3 传输规则

上面讨论的数据包是通过一个介于目标和主机之间的某种串行接口传输的。全双工 UART 是典型接口。为了提供可靠和准确的数据包传输，对所有的数据包都应用了一组传输规则。这些规则在“HID 桥数据包模式”部分中讨论。

表 17 列出了在应用传输规则时相关的函数。

表 14：接收队列缓冲区容量定义

参数	文件
在一个数据包中填充同步字节	extern uint16_t CAPT_stuffSyncBytes(uint8_t *pBuffer, uint16_t ui16Length)
验证一个校验和	extern bool CAPT_verifyChecksum(const uint8_t *pBuffer, const uint16_t ui16Length, const uint16_t ui16Checksum)
获得一个校验和	extern uint16_t CAPT_getChecksum(const uint8_t *pBuffer, const uint16_t ui16Length)
在一个接收数据队列中识别和构建一个数据包	extern bool CAPT_processReceivedData(tByteQueue *pReceiveQueue, tParameterPacket *pPacket, tTLProtocolProcessingVariables *pVariables)

9.6.4.4 格式：传感器数据包

传感器数据包具有 6 字节的固定长度。有两个控制字节和四个数据负载字节。

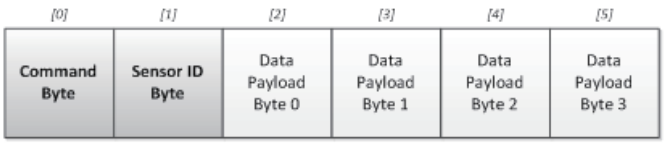


图 125：传感器数据包格式

- 命令字节（Command Byte） [0]
  用于传感器数据包的命令字节始终是 00h

• 传感器 ID 字节（Sensor ID Byte） [1]
传感器 ID 字节包含一个无符号 8 位整数，它指定了所传数据传感器的 ID。目标板的传感器 ID 通常是由全局传感器指针阵列中的指针的顺序建立的。传感器由 CapTIvate Design Center PC GUI 按字母顺序进行排序。

• 数据负载字节（Data Payload Byte） [2-5]
数据负载字节包含正在发送的传感器数据。负载中的信息基于传感器的类型。表 18 在传感器类型的基础上说明了负载信息。

表 18：传感器数据包

传感器类型	字节 0	字节 1	字节 2	字节 3
按键组	主导元件 （256 个元件，最大值）	前一个主导元件 （256 个元件，最大值）	保留	传感器状态
滑条	滑条位置 （16 位的低 8 位）	滑条位置 （16 位的高 8 位）	保留	传感器状态
滚轮	滚轮位置 （16 位的低 8 位）	滚轮位置 （16 位的高 8 位）	保留	传感器状态
接近感应	保留	保留	保留	传感器状态
触控板	保留	保留	保留	传感器状态

包括在按键组、滑条和滚轮传感器数据包中的传感器状态字节提供传感器状态的额外数据。状态标志全部是布尔标志，并且按如下方式分配：

表 19：传感器状态标志

位屏蔽（位置）	传感器状态标志
Bit 0 (01h)	全局传感器触摸标志
Bit 1 (02h)	全局传感器前一个触摸标志
Bit 2 (04h)	全局传感器接近标志
Bit 3 (08h)	全局传感器检测标志（接近检测预先防抖）
Bit 4 (10h)	全局传感器负触摸标志（反向触摸）
Bit 5 (20h)	全局传感器噪声状态
Bit 6 (40h)	全局传感器最大计数错误标志
Bit 7 (80h)	全局传感器校准错误标志

注： 对于滑条或滚轮传感器，当该传感器上的全局传感器触摸标志为“真”时，数据有效负载中的 16 个位置位包含有效的滑条或滚轮位置。如果没有全局触摸检测，则 16 个位置位均被置高（对于滑条或滚轮位置值为 0xFFFF）。

注： 保留的字段仍然被传输，但其不包含任何有意义的数据。不要使用或依靠在保留字段中传输的任何数据。

传感器数据包是通过调用协议层生成的。CAPT_getSensorPacket() 函数访问位于阵列 sensorArray 中的索引 ui8SensorID 上的传感器，并把生成的数据包存储在 pBuffer 所指向的缓冲空间中。此数据包的长度由该函数返回。

表 20：“获得一个传感器数据包”函数

说明	声明
获得一个传感器数据包	extern uint16_t CAPT_getSensorPacket(tSensor **sensorArray, uint8_t ui8SensorID, uint8_t *pBuffer)

9.6.4.5 格式：周期数据包

周期数据包的长度可变，取决于周期之内的元件数量。默认有三个控制字节和 3 个状态字节。除了这 6 个字节之外，周期中的每个元件有 4 个字节。

图 126：周期数据包格式

- 命令字节（Command Byte） [0]
  用于周期数据包的命令字节始终是 01h。

- 传感器 ID 字节（Sensor ID Byte） [1]
  传感器 ID 字节包含一个无符号 8 位整数，它规定了所传数据传感器的 ID。目标板的传感器 ID 通常是由全局传感器指针阵列中的指针的顺序建立的。传感器由 CapTIvate Design Center PC GUI 按字母顺序进行排序。

- 周期 ID 字节（Cycle ID Byte） [2]
  周期 ID 字节包含一个无符号 8 位整数，它规定了所传数据（相对于传感器）的周期的 ID。例如：在具有三个周期的传感器中，第一个周期 ID = 0，第二周期 ID = 1，而第三个周期 ID = 2。周期 ID 应该对应于传感器的周期指针阵列中的周期索引。

- 周期状态字节（State Byte） [3-5]
  周期状态字节规定了用于周期中每个元素的触摸和接近检测标志。每个周期最多支持 12 个元件。周期数据包的 Byte 3 ～ 5 包含一个 24 位周期状态，低 12 个位以按位的形式代表了每个元件的接近状态，高 12 个位则以按位的形式代表每个元件的触摸状态。

- 元件 LTA 和元件计数字节 [6-n]，n = 5 + 4（元件的数量）
  周期数据包的剩余部分包含元件 LTA 和计数值。LTA 和计数表示为 16 位无符号整数。因此，LTA / 计数部分中的每个元素需要 32 位。对于周期中每个额外的元件（最多 12 个元件），数据包的大小应以 32 位（4 个字节）的倍数递增。首先发送 LTA，然后发送计数。按照协议，当发送多字节的数值时，传输的顺序是从最低位字节到最高位字节。

周期数据包是通过调用协议层生成的。CAPT_getCyclePacket() 函数访问位于阵列 sensorArray 中的索引 ui8SensorID 上的传感器中的索引 ui8Cycle 上的周期，并把生成的数据包存储在 pBuffer 所指向的缓冲空间中。数据包的长度由该函数返回。

表 21：“获得一个周期数据包”函数

说明	声明
获得一个周期数据包	extern uint16_t CAPT_getCyclePacket(tSensor **sensorArray, uint8_t ui8SensorID, uint8_t ui8Cycle, uint8_t *pBuffer);

9.6.4.6 格式：触控板数据包

触控板数据包具有一个可变长度，该长度取决于触控板器件支持的同时触摸的最大数量。有三个控制字节，控制字节之后有一个触控板手势字节 (gesture byte)，用于指示某种手势的检测以及该手势是什么。对于触控板器件支持的每个同时触摸点，给该数据包增添了 4 个额外的有效负载字节。下面可以看到此类数据包的格式。

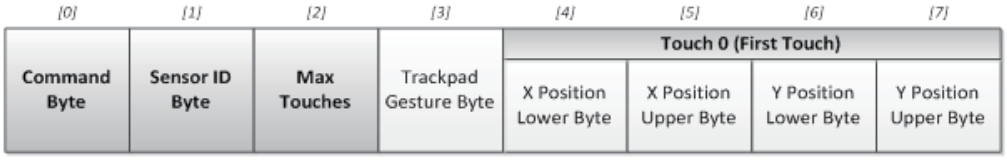


图 126：周期数据包格式

- 命令字节（Command Byte） [0]
  用于触控板数据包的命令字节始终是 02h。

- 传感器 ID 字节（Sensor ID Byte） [1]
  传感器 ID 字节包含一个无符号 8 位整数，它规定了所传数据传感器的 ID。目标板的传感器 ID 通常是由全局传感器指针阵列中的指针的顺序建立的。传感器由 CapTIvate Design Center PC GUI 按字母顺序进行排序。

- 最大触摸数（Max Touches） [2]
  最大触摸数字节包含一个无符号 8 位整数，它规定了触控板器件支持的同时触摸的数量，还提供了数据包其余部分的长度信息。在最大触摸字节之后，每一个同时存在的触摸均有对应的 4 个字节。触控板器件必须支持至少一个触摸数。

- 触控板手势字节（Trackpad Gesture Byte） [3]
  触控板手势字节指示是否检测到一个手势，以及该手势是什么。表 22 列出了数值 - 手势对应。如果未检测到手势，则手势字节被设定为 FFh。

表 22：触控板手势

数值	手势
00h	接近唤醒检测
01h	保留
02h	单触摸单点击
03h	单触摸双点击
04h	双触摸单点击
05h	双触摸双点击
06h	点击并按住
07h	双触摸点击并按住
08h	向左划动
09h	向右划动
0Ah	向上划动
0Bh	向下划动
0Ch-FEh	保留
FFh	未检测到手势

- 触摸坐标 [4-n]，n = 4 + 4（最大触摸数）
  触摸坐标指示触摸在触控板上的存在和位置。触摸位置是每个触摸通过两个 16 位无符号整数进行传送的：一个用于 X 轴上的坐标，一个用于 Y 轴上的坐标。对于触控板，X 或 Y 方向上的最大分辨率为 16 位。大多数应用的坐标小于 16 位的工作分辨率，但始终发送 16 位。如果不存在触摸，则 X 和 Y 坐标都读作 FFFFh。

注： 触控板数据包仅与触控板专用器件有关。

9.6.4.7 格式：通用数据包

通用数据包是专门用来充当数据流式传输机制的，用于某应用程序希望发送的任何数据。数据作为一个 16 位无符号整数组发送。通用数据包具有可变的长度，该长度取决于被发送的整数的数量。其默认有两个控制字节，在控制字节之后是数据有效负载（多达 29 个条目），每个条目两个字节（16 位）。

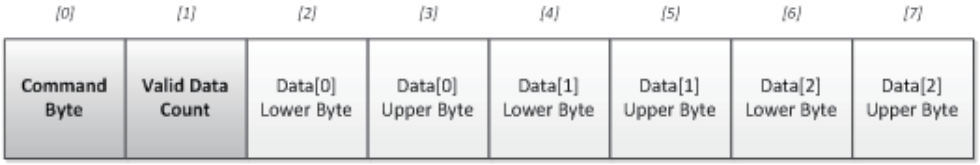


图 128：通用数据包格式

- 命令字节（Command Byte） [0]
  用于通用数据包的命令字节始终是 10h。
- 有效数据计数字节 (Valid Data Count) [1]
  有效数据计数字节指示在数据包中存在多少个有效的数据条目。这也表明了数据包的长度：每个有效的条目代表着两个有效负载字节（每个条目是一个 16 位无符号整数）。该字段的有效值为 1 ～ 29。
- 数据有效负载（Data Payload） [2-n]
  数据有效负载部分包含了即将传输的通用数据（其格式化为无符号的 16 位整数）。允许提供多达 29 个有效负载条目（58 个字节）。
  通用数据包通过调用协议层生成。CAPT_getGeneralPurposePacket() 函数生成一个用于 pData 所指向的长度为 ui8Cnt 的数组的数据包，并把生成的数据包存储在 pBuffer 所指向的缓冲空间中。

表 23：“获得一个通用数据包”函数

说明	声明
获得一个通用数据包	extern uint16_t CAPT_getGeneralPurposePacket(uint16_t *pData, uint8_t ui8Cnt, uint8_t *pBuffer)

9.6.4.8 格式：参数数据包

参数数据包具有 7 字节的固定长度，有 3 个控制字节和 4 个数据有效负载字节。下面可以看到该数据包的格式。

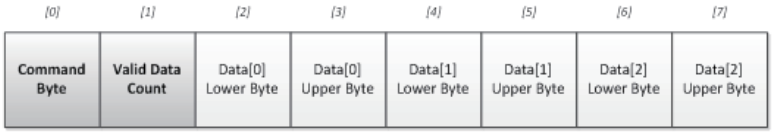


图 129：参数数据包格式

- 命令字节（Command Byte） [0]
  用于参数数据包的命令字节是可变的。并指示即将被执行读或写操作的参数的 ID。
- 读 / 写字节（Read/Write Byte） [1]
  读 / 写字节是一个 1 位布尔值，其表示操作是参数读取还是参数写入。1 = 写入，0 = 读取。
- 传感器 ID 字节（Sensor ID Byte） [2]
  传感器 ID 字节包含一个无符号 8 位整数，它规定了所传数据传感器的 ID。目标板的传感器 ID 通常是由全局传感器指针阵列中的指针的顺序建立的。传感器由 CapTIvate Design Center PC GUI 按字母顺序进行排序。

• 数据字节 (Data Bytes) [3-6]
数据字节包含了即将读取或写入的信息。在某些场合中，一个或多个数据字节被用于更多的 ID（例如：周期 ID 或元素 ID）。

协议层提供了若干用于构建和解释参数数据包的函数。调用函数 CAPT_processReceivedData() 使协议层在串行驱动程序排队等候的数据流中搜索潜在的数据包。当构建了一个有效的数据包时，可通过调用 CAPT_accessSensorParameter() 和 CAPT_accessSpecialSensorParameter() 来访问和更新 / 读取参数。

表 24：参数数据包函数

说明	声明
访问一个传感器参数	extern tTLParameterAccessResult CAPT_accessSensorParameter(tSensor **sensorArray, tParameterPacket *pPacket)
访问一个特殊的传感器参数	extern tTLParameterAccessResult CAPT_accessSpecialSensorParameter(tSensor **sensorArray, Parameter tParameterPacket *pPacket)
在接收数据队列中识别和构建一个数据包	extern bool CAPT_processReceivedData(tByteQueue *pReceiveQueue, tParameterPacket *pPacket, tTLProtocolProcessingVariables *pVariables)
验证一个校验和	extern bool CAPT_verifyChecksum(const uint8_t *pBuffer, const uint16_t ui16Length, const uint16_t ui16Checksum)

表 25 ～ 28 罗列了可通过使用参数数据包来调节的参数。

表 25：传感器参数

Parameter Name	Byte 0: CMD	Byte 1: RW	Byte 2: ID	Byte 3	Byte 4	Byte 5	Byte 6	MCU Task	SW Containing Structure	SW Containing Variable
Conversion Gain	80	RW	Sensor ID	Don't Care	Don't Care	ATI Base Lower Byte	ATI Base Upper Byte	Re-Cal	tSensor	ui16ConversionGain
Conversion Count	81	RW	Sensor ID	Don't Care	Don't Care	ATI Target Lower Byte	ATI Target Upper Byte	Re-Cal	tSensor	ui16ConversionCount
Prox Threshold	82	RW	Sensor ID	Don't Care	Don't Care	Prox Threshold Lower Byte	Prox Threshold Upper Byte	NA	tSensor	ui16ProxThreshold
Prox Debounce-In Threshold	84	RW	Sensor ID	Don't Care	Don't Care	Prox Db In	Don't Care	NA	tSensor	ProxDbThreshold .DbUp
Prox Debounce-Out Threshold	85	RW	Sensor ID	Don't Care	Don't Care	Prox Db Out	Don't Care	NA	tSensor	ProxDbThreshold .DbDown
Touch Debounce-In Threshold	86	RW	Sensor ID	Don't Care	Don't Care	Touch Db In	Don't Care	NA	tSensor	TouchDbThreshold .DbUp
Touch Debounce-Out Threshold	87	RW	Sensor ID	Don't Care	Don't Care	Touch Db Out	Don't Care	NA	tSensor	TouchDbThreshold .DbDown
Sensor Time-out Threshold	88	RW	Sensor ID	Don't Care	Don't Care	Sensor Time-out Lower Byte	Sensor Time-out Upper Byte	NA	tSensor	ui16TimeoutThreshold
Count Filter Enable	89	RW	Sensor ID	Don't Care	Don't Care	Count Filter Enable bit	Don't Care	NA	tSensor	bCountFilterSelect
Count Filter Beta	8A	RW	Sensor ID	Don't Care	Don't Care	Count Filter Beta	Don't Care	NA	tSensor	ui8CntBeta
LTA Filter Beta	8B	RW	Sensor ID	Don't Care	Don't Care	LTA Filter Beta	Don't Care	NA	tSensor	ui8LTABeta
Halt LTA Filter Immediately	8C	RW	Sensor ID	Don't Care	Don't Care	LTA Filter Halt	Don't Care	NA	tSensor	bSensorHalt
Runtime Recalibration Enable	8D	RW	Sensor ID	Don't Care	Don't Care	Runtime Re-Cal Enable	Don't Care	NA	tSensor	bReCalibrateEnable
Force Re-Calibrate	8E	N/A	Sensor ID	Don't Care	Don't Care	Don't Care	Don't Care	Re-Cal	tSensor	N/A
Bias Current	8F	RW	Sensor ID	Don't Care	Don't Care	Bias Current	Don't Care	Re-Cal	tSensor	ui8BiasControl
Sample Capacitor Discharge	95	RW	Sensor ID	Don't Care	Don't Care	Cs Discharge	Don't Care	Re-Cal	tSensor	bCsDischarge
Modulation Enable	96	RW	Sensor ID	Don't Care	Don't Care	Mod Enable	Don't Care	Re-Cal	tSensor	bModEnable
Frequency Divider	97	RW	Sensor ID	Don't Care	Don't Care	Freq Div	Don't Care	Re-Cal	tSensor	ui8FreqDiv
Charge/Hold Phase Length	98	RW	Sensor ID	Don't Care	Don't Care	Charge Length	Don't Care	Re-Cal	tSensor	ui8ChargeLength
Transfer/Sample Phase Length	99	RW	Sensor ID	Don't Care	Don't Care	Transfer Length	Don't Care	Re-Cal	tSensor	ui8TransferLength

表 25：传感器参数（续）

Parameter Name	Byte 0: CMD	Byte 1: RW	Byte 2: ID	Byte 3	Byte 4	Byte 5	Byte 6	MCU Task	SW Containing Structure	SW Containing Variable
Error Threshold	9A	RW	Sensor ID	Don't Care	Don't Care	Error Threshold Lower Byte	Error Threshold Upper Byte	NA	tSensor	ui16ErrorThreshold
Negative Touch Threshold	9B	RW	Sensor ID	Don't Care	Don't Care	Negative Touch Threshold Lower Byte	Negative Touch Threshold Upper Byte	NA	tSensor	ui16NegativeTouchThreshold
Idle State	9C	RW	Sensor ID	Don't Care	Don't Care	Idle State	Don't Care	NA	tSensor	bIdleState
Input Sync	9D	RW	Sensor ID	Don't Care	Don't Care	Input Sync	Don't Care	NA	tSensor	ui8InputSyncControl
Timer Sync	9E	RW	Sensor ID	Don't Care	Don't Care	Timer Sync	Don't Care	NA	tSensor	bTimerSyncControl
Automatic Power-Down Enable	9F	RW	Sensor ID	Don't Care	Don't Care	Power Down Control	Don't Care	NA	tSensor	bLpmControl
Halt LTA on Sensor Prox or Touch	A0	RW	Sensor ID	Don't Care	Don't Care	Sensor Prox/Touch Halt	Don't Care	NA	tSensor	bPTSensorHalt
Halt LTA on Element Prox or Touch	A1	RW	Sensor ID	Don't Care	Don't Care	Element Prox/Touch Halt	Don't Care	NA	tSensor	bPTElementHalt

9.6.4.8.2 元件参数

表 26：元件参数

Parameter Name	Byte 0: CMD	Byte 1: RW	Byte 2: ID	Byte 3	Byte 4	Byte 5	Byte 6	MCU Task	SW Containing Structure	SW Containing Variable
Touch Threshold	83	RW	Sensor ID	Cycle No. (relative to sensor)	Lower 4 Bits: Element No. (relative to cycle)	Touch Threshold	Don't Care	NA	tElement	ui8TouchThreshold [Element No.]
Coarse Gain Ratio	A2	R	Sensor ID	Cycle No. (relative to sensor)	[UPPER 4 Bits: Frequency No.] [LOWER 4 Bits: Element No. relative to cycle]	Coarse Gain	Don't Care	NA	tCapTIvateElementTuning	ui8GainRatioCoarse
Fine Gain Ratio	A3	R	Sensor ID	Cycle No. (relative to sensor)	[UPPER 4 Bits: Frequency No.] [LOWER 4 Bits: Element No. relative to cycle]	Fine Gain	Don't Care	NA	tCapTIvateElementTuning	ui8GainRatioFine
Parasitic Offset Scale	D0	R	Sensor ID	Cycle No. (relative to sensor)	[UPPER 4 Bits: Frequency No.] [LOWER 4 Bits: Element No. relative to cycle]	Offset Scale (2 bit selection)	Don't Care	NA	tCapTIvateElementTuning	ui16OffsetTap Upper Byte
Parasitic Offset Level	D1	R	Sensor ID	Cycle No. (relative to sensor)	[UPPER 4 Bits: Frequency No.] [LOWER 4 Bits: Element No. relative to cycle]	Offset Level (8 bit selection)	Don't Care	NA	tCapTIvateElementTuning	ui16OffsetTap Lower Byte

9.6.4.8.3 滑条和滚轮参数

表 27：滑条和滚轮参数

Parameter Name	Byte 0: CMD	Byte 1: RW	Byte 2: ID	Byte 3	Byte 4	Byte 5	Byte 6	MCU Task	SW Containing Structure	SW Containing Variable
Slider/Wheel Position Filter Enable	90	RW	Sensor ID	Don't Care	Don't Care	Slider Filter Enable bit	Don't Care	NA	tSliderSensorParams	SliderFilterEnable
Slider/Wheel Position Filter Beta	91	RW	Sensor ID	Don't Care	Don't Care	Slider Filter Beta	Don't Care	NA	tSliderSensorParams	SliderBeta
Desired Slider/Wheel Resolution	92	RW	Sensor ID	Don't Care	Don't Care	Slider Resolution Lower Byte	Slider Resolution Upper Byte	NA	tSliderSensorParams	ui16Resolution
Slider Lower Trim	93	RW	Sensor ID	Don't Care	Don't Care	Slider Lower Trim Lower Byte	Slider Lower Trim Upper Byte	NA	tSliderSensorParams	SliderLower
Slider Upper Trim	94	RW	Sensor ID	Don't Care	Don't Care	Slider Upper Trim Lower Byte	Slider Upper Trim Upper Byte	NA	tSliderSensorParams	SliderUpper

9.6.4.8.4 控制器和管理参数

表 27：滑条和滚轮参数

Parameter Name	Byte 0: CMD	Byte 1: RW	Byte 2: ID	Byte 3	Byte 4	Byte 5	Byte 6	MCU Task	SW Containing Structure	SW Containing Variable
Element Data Transmit Enable	C0	RW	Don't Care	Don't Care	Don't Care	Element Transmit Enable	Don't Care	NA	tCapTIvateApplication	bElementDataTxEnable
Sensor Data Transmit Enable	C1	RW	Don't Care	Don't Care	Don't Care	Sensor Transmit Enable	Don't Care	NA	tCapTIvateApplication	bSensorDataTxEnable
Active Mode Scan Rate (ms)	C2	RW	Don't Care	Don't Care	Don't Care	Active Report Period (ms) Lower Byte	Active Report Period (ms) Upper Byte	NA	tCapTIvateApplication	ui16ActiveModeScanPeriod
Wake-on-Prox Mode Scan Rate (ms)	C3	RW	Don't Care	Don't Care	Don't Care	WoP Report Period (ms) Lower Byte	WoP Report Period (ms) Upper Byte	NA	tCapTIvateApplication	ui16WakeOnProxModeScanPeriod
Wakeup Interval	C4	RW	Don't Care	Don't Care	Don't Care	Wakeup Interval	Don't Care	NA	tCapTIvateApplication	ui8WakeupInterval
Inactivity Time-out	C5	RW	Don't Care	Don't Care	Don't Care	Time-out Lower Byte	Time-out Upper Byte	NA	tCapTIvateApplication	ui16InactivityTimeout

9.6.5 UART 驱动程序

MSP430 eUSCI_A UART 驱动程序为 MSP430 应用程序提供了一种简单的 UART API，可实现中断驱动型发送和接收操作以及错误处理。本节概述了该驱动程序的特性、架构和 API。本文件假定读者熟悉 MSP430 MCU 架构以及嵌入式 C 语言编程概念。

9.6.5.1 相关文件

本文件应与下列的其他支持性文档配套使用。MSP430 驱动程序库 API 并不专门用于于该驱动程序，其具有单独的文档描述。

- MSP430 驱动程序库 (DriverLib) 用户指南
- 相关的 MSP430 系列用户指南

9.6.5.2 驱动程序的目的

eUSCI_A UART 驱动程序使得开发人员能够通过一个与在 PC 上使用的应用程序接口相似的 API 实现快速启动并运行 UART 通信。该 API 提供了诸如 UART_openPort() 和 UART_transmitBuffer() 等简单的函数，而且它允许开发人员注册用于接收数据和错误情况的事件处理程序。为了实现移植性，该驱动程序基于 MSP430 DriverLib 。此驱动程序专为与 MSP430 DriverLib build 1.90.00.00 及更高版本配合工作而设计。和任何其他的串行接口一样，UART 具有优点和缺陷。它是否为嵌入式接口的最佳选择取决于诸多因素。UART 的优势与缺陷在下面讨论。

UART 的优势

- 易于实施。UART 是最常见的串行通信协议之一。虽然存在着诸如 I^2C 和 SPI 等速度和可靠性更高的协议，但没有哪种协议能在实施的简易性上与 UART 相提并论。UART 可容易地连接至主机 PC 或微控制器，因此它非常适合用作调试操控台。
- 点对点简单性。由于只存在两个节点，因此该接口不是一根连接了多个器件的总线，避免了增加寻址或芯片选择开销的需要。
- 全双工通信。在该 UART 驱动程序实施方案中，发送和接收操作是相互独立的，可以并行执行。两个连接节点均可把自己视为一个主控器，并可随时开始向另一个节点传输。

UART 的缺陷

- 严苛的位时序。由于 UART 传输是异步的（没有位时钟），因此在接口的两侧都必须生成准确的时序。
- 位速率限制。由于上述的苛刻位时序限制，与 SPI 和 I^2C 等同步接口相比，UART 通常在最大位时钟方面更加受限。
- 严苛的 ISR 时序。该驱动程序未采取将 DMA 通道用于数据从发送和接收缓冲区移动至 RAM 的做法，因为并不是所有的器件都有 DMA 。所有的数据传输都是通过中断服务程序（而不是 DMA）进行处理的，要求 CPU 至少在每个字节周期可用于处理接收中断。一个字节周期被定义为一个字节的传输时间，其等于位周期的 10 倍。如果采用了奇偶校验或多于一个的停止位，则字节传输时间增加。例如：位时钟为 250 kHz，则一个字节的传输时间为 4 μs x 10 位，即 40 μs，当采用 8 MHz CPU 时钟时，每 40 μs 只有 320 个可用指令周期，假如 CPU 不可用于处理接收中断，则接收字节有可能丢失。该驱动程序提供了一种错误检测机制，在数据丢失和数据正在丢失时向应用程序发出警示信号。
- 异步开销。在该驱动程序中配置的 UART 需要至少一个起始位和一个停止位。因此，每个 8 位字节需要传送 10 个数位。在 SPI 接口中将不存在此开销，而 I^2C 加入了一个 ACK/NACK 位和寻址开销。

9.6.5.3 驱动程序特性

在 UART 驱动程序中实现的主要特性列举如下。

- 全双工双向通信
- 无需流量控制
- 全中断驱动型
- 非阻塞传递函数
- 接收事件回调
- 错误事件回调
- 在编译的时候选择使用哪个 eUSCI_A 实体

9.6.5.4 驱动程序概述

UART 驱动程序在源代码中提供。该驱动程序包括三个源文件：

- UART_Definitions.h（配置头文件）
- UART.h（API 头文件）
- UART.c（API 实现文件）。核心驱动程序在 UART.h 和 UART.c 中实现。UART_Definitions.h 文件使得开发人员能够调整驱动程序的编译选项。

UART 驱动程序需要下面的硬件资源：

- 一个 eUSCI_A 外设实体
- 两个器件引脚（UCAxTXD 和 UCAxRXD），其中的“x”代表所选的 eUSCI_A 实体

UART 驱动程序 API 由函数和数据类型组成。下面的部分说明了如何配置 UART 驱动程序并用它来执行发送和接收操作。UART 驱动程序采用一个基本的软件状态机来管理操作。图 130 示出了三种驱动程序状态及其互连。

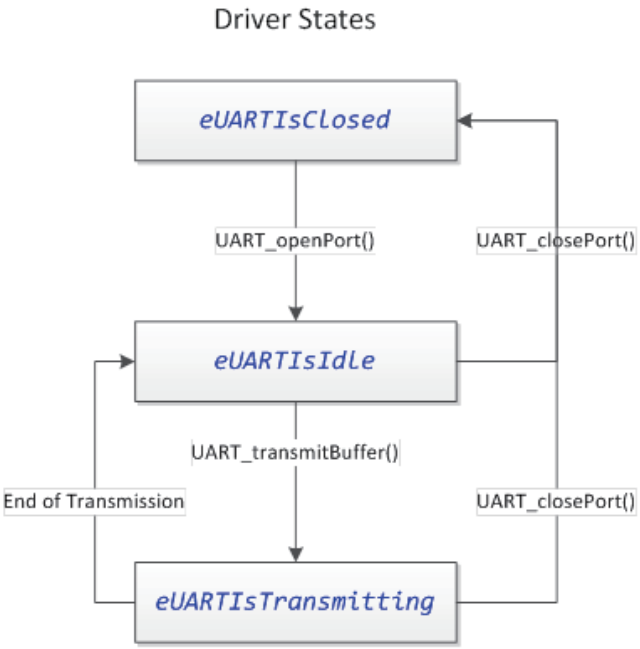


图 130：UART 驱动程序状态机

9.6.5.5 驱动程序编译配置选项

UART 驱动程序具有编译选项及运行配置选项。在编译的时候有些东西必须是确定的，比如哪个 eUSCI_A 外设实体是与驱动程序相关联的。而诸如波特率等其他选项可在运行的时候控制。

编译时间配置选项在 UART_Definitions.h 文件中设定。可用的编译时间选项在下面说明。

9.6.5.5.1 UART 使能定义

表 29：UART 使能参数

参数	文件	有效值
UART__ENABLE	UART_Definitions.h	true, false

UART Enable 编译选项用于选择 UART 的使能或停用，提供了把该驱动程序排除在编译过程之外的机制。如需包含该驱动程序，则把 UART__ENABLE 定义为“true”，其他情况下则将其定义为“false”。

9.6.5.5.2 UART eUSCI_A 外设选择定义

表 29：UART 使能参数

参数	文件	有效值
UART__EUSCI_A_PERIPHERAL	UART_Definitions.h	EUSCI_A0_BASE, EUSCI_A1_BASE

UART eUSCI_A 外设选择使得可以选择哪一个 eUSCI_A 实体与 UART 驱动程序相关联。如果必需进行引脚选通的变更，这样可以在设计过程中提供灵活性。必须提供一个有效的基地址。当做出一个 eUSCI_A 外设选择时，UART 驱动程序 ISR 地址被自动地链接至合适的 eUSCI 中断矢量。如果选择了一个无效的地址，将引发编译程序错误。

9.6.5.5.3 UART 低功耗模式 (LPM)

表 31：UART 低功耗模式参数

参数	文件	有效值
UART__LPMx_bits	UART_Definitions.h	0，LPM0_bits，LPM1_bits，LPM2_bits，LPM3_bits

UART 通信常常在低功耗模式中进行。UART LPM 模式配置以三种方式使用，如下文所述。

- 接收回调和错误回调函数可以触发从低功耗模式主动退出，以唤醒 CPU 执行进一步的动作。如果某个回调函数返回了一个布尔值 true，则 UART 驱动程序将在从回调返回之后唤醒 CPU。其从 UART ISR 退出时清零在 UART LPM 模式配置中指示的状态寄存器位以实现该功能。
- 当全部传输都已完成时，发送操作将退出 UART LMP 模式配置所规定的低功耗模式，从而通知应用程序传输已完成。
- 传输函数一般情况下是非阻塞函数。当传输开始时，该函数返回。然而，当调用传输函数时前一个传输仍在进行之中，有一种选项是等待前一个传输在某种低功耗模式中完成。如果选择了该选项，则在等待的同时进入由 UART LPM 模式配置所规定的低功耗模式。传输函数将在前一个传输退出运行时开始传输（依据上面的第二点）。

9.6.5.6 驱动程序运行配置选项

UART 驱动程序具有编译选项及运行配置选项。运行配置选项是通过在应用程序中添加 tUARTPort 结构体并在运行驱动程序时把该结构体传递至驱动程序来实现的。当运行 UART 驱动程序时需要一种 tUARTPort 架构，tUARTPort 结构在存储器中都必须是可用的，驱动程序在运行的时候必须参考该该结构体以找到用于接收处理和错误处理的回调函数。不过，驱动程序在任何时候都不修改此结构体中的数据，因此，可以把该结构体存储在一个只读存储区段（例如：C 常数存储器）中。这些参数被认为是在运行的时候可调用的，因为如果 UART 驱动程序先关闭然后重新打开，那么可由应用程序来变更参数。例如，应用程序可以通过关闭端口、改变 tUARTPort 结构上的波特率选项、以及重新打开端口来更改 UART 波特率。tUARTPort 结构的 .peripheralParameters 是一种取自 MSP430 驱动程序库 (Driver Library) 的 EUSCI_A_UART_initParam 结构体。

表 32：驱动程序运行配置选项

Member	Description	Valid Values
bool (*pbReceiveCallback)(uint8_t)	pbReceiveCallback is a function pointer that may point to a receive event handler. If no receive handling is required, initialize this member to 0 (null).	Null, or a valid function address.
bool (*pbErrorCallback)(uint8_t)	pbErrorCallback is a function pointer that may point to an error event handler. If no error handling is required, initialize this member to 0 (null).	Null, or a valid function address.
.peripheralParameters.selectClockSource	This member specifies the clock source for the eUSCI_A peripheral.	EUSCI_A_UART_CLOCKSOURCE_SMCLK, EUSCI_A_UART_CLOCKSOURCE_ACLK
.peripheralParameters.clockPrescalar	This member specifies the eUSCI_A clock prescalar. This affects the baud rate.	0-65535
.peripheralParameters.firstModReg	This member specifies the eUSCI_A first stage modulation. This affects the baud rate.	0-15
.peripheralParameters.secondModReg	This member specifies the eUSCI_A second stage modulation. This affects the baud rate.	0-255
.peripheralParameters.parity	This member specifies the UART parity mode.	EUSCI_A_UART_NO_PARITY, EUSCI_A_UART_ODD_PARITY, EUSCI_A_UART_EVEN_PARITY
.peripheralParameters.msborLsbFirst	This member specifies the transmission bit order.	EUSCI_A_UART_LSB_FIRST, EUSCI_A_UART_MSB_FIRST
.peripheralParameters.numberofStopBits	This member specifies the number of stop bits.	EUSCI_A_UART_ONE_STOP_BIT, EUSCI_A_UART_TWO_STOP_BITS
.peripheralParameters.uartMode	This member specifies the UART mode.	EUSCI_A_UART_MODE, EUSCI_A_UART_IDLE_LINE_MULTI_PROCESSOR_MODE, EUSCI_A_UART_ADDRESS_BIT_MULTI_PROCESSOR_MODE, EUSCI_A_UART_AUTOMATIC_BAUDRATE_DETECTION_MODE
.peripheralParameters.overSampling	This member specifies whether UART oversampling is enabled.	EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION, EUSCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION

9.6.5.7 使用驱动程序：打开和关闭驱动程序

表 33 列出了用于打开和关闭 UART 驱动程序的函数。

表 33：用于打开和关闭驱动程序的函数

说明	声明
打开 UART 端口	extern void UART_openPort(const tUARTPort *pPort)
关闭 UART 端口	extern void UART_closePort(void)

UART 驱动程序由 UART_openPort() 函数调用传递完整的 tUARTPort 结构体来打开和初始化。tUARTPort 结构体必须由应用程序添加，它可以被存储在只读存储器中，因为 UART API 在任何时候都不会修改此结构体，而是仅仅参考该结构体。重要的是，应把该结构体存储于 UART_openPort() 调用时所给定的地址中，因为 UART API 将在端口打开时参考该结构以访问回调函数。如果必须释放存储器，则需首先调用 UART_closePort() 以关闭 UART 端口。

对 UART_closePort() 的调用将停用 UART 端口及其关联的 eUSCI_A 外设，并停止所有的 Rx/Tx 中断活动。在该端口关闭之后，不再调用事件处理程序，并可以把 tUARTPort 结构体内存释放给应用程序。

9.6.5.8 使用驱动程序：传输数据

数据传输是通过下面的函数调用来完成的。

表 34：数据传输声明

说明	声明
传输一个缓冲区	extern void UART_transmitBuffer(const uint8_t *pBuffer, uint16_t ui16Length)
传输一个字节	extern void UART_transmitByteImmediately(uint8_t ui8Data)
获得端口状态	extern uint8_t UART_getPortStatus(void)

传输操作是中断驱动型的，如需在打开 UART 端口之后启动传输，则需调用 UART_transmitBuffer()。这个函数传递一个指向传输缓冲区的指针，以及传输缓冲区的长度（用字节来定义）。如果 eUSCI_A 外设可用，则传输立即开始且函数返回。如果外设仍忙于发送前一个传输，则其阻塞（在一种 UART_Definitions.h 中规定的低功耗模式下），直到前一个传输完成为止。

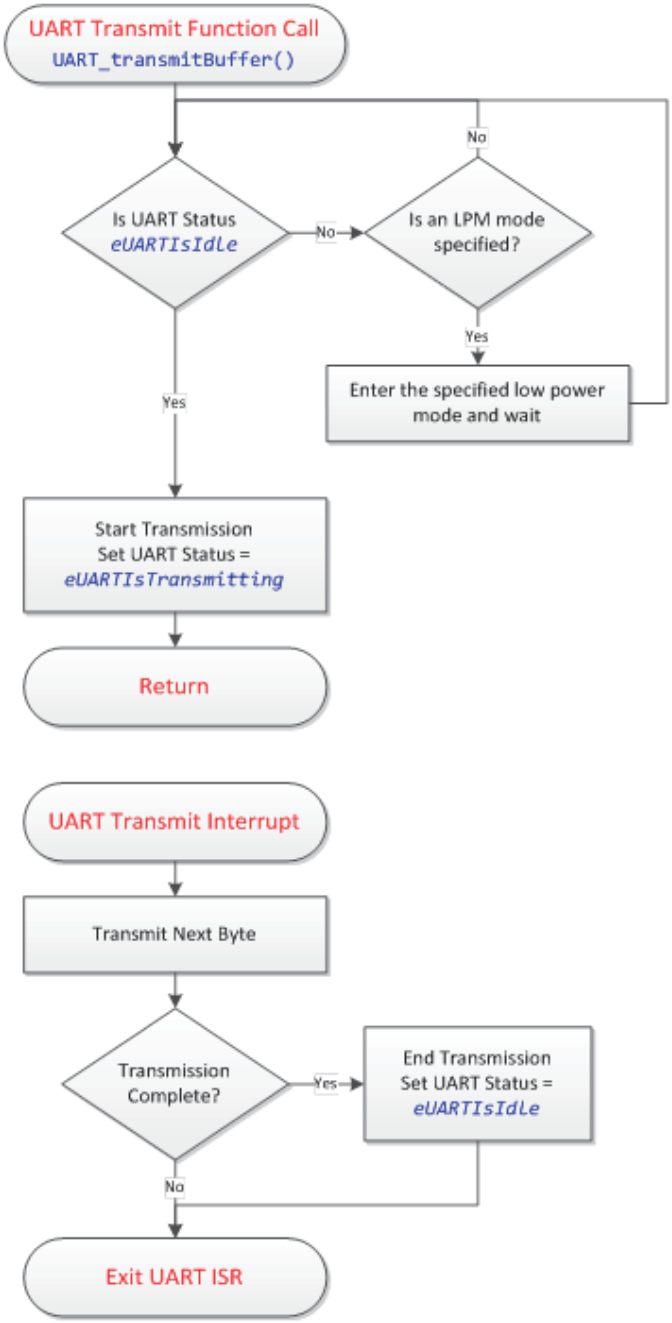


图 131：UART 传输流程图 T

发送操作使用在 UART_transmitBuffer() 函数中被指向的缓冲区内存 。数据拷贝并不高效，因此，在传输期间不得更改内存，否则传输无效。为了检测传输完成且内存再次可用，可以通过调用 UART_getPortStatus() 来检查 UART 端口状态。如果某个应用程序正在复用缓冲区空间，最好运用乒乓缓存策略，这样就可以在前一个数据包正在发送的同时装载新的数据包。

调用 UART_getPortStatus() 将回送由 tUARTStates 枚举的数值之一选项说明如下。

表 35：端口状态选项

端口状态选项	描述
eUARTIsClosed	UART 驱动程序当前未打开，而且既不接收数据也不发送数据。
eUARTIsIdle	UART 驱动程序当前是打开的，但是不处于传输一个缓冲区的过程之中。
eUARTIsTransmitting	UART 驱动程序当前是打开的，而且处于传输一个缓冲区的过程当中。

如果仅发送单个字节，则可通过 UART_transmitByteImmediately() 函数立即发送。一旦 eUSCI_A 外设发送缓冲区可用，该函数立即发送单个字节，并将其锁定直到缓冲区重新可用。如果某个由 UART_transmitBuffer() 来启动的传输在进行之中，则该传输可能首先完成，因为它是中断驱动型的。UART_transmitByteImmediately() 函数主要在终端仿真应用程序中使用，已发送的 ASCII 字符在这里被回送给用户查看。

9.6.5.9 使用驱动程序：接收数据

接收到的数据通过接收回调传递至应用程序。每次从 UART 外设接收一个新的字节，UART 驱动程序都将调用一个用户定义的函数，并把该字节传递至接收回调函数，然后由应用程序来决定如何处理数据。建议采用 FIFO 队列来处理缓冲输入数据，随后可以在后台进程中提取和处理数据。接收事件处理程序被 UART 驱动程序中的 UART ISR 调用，因此事件驱动程序应尽可能地短。建议遵循写入事件驱动程序时用于写入 ISR 的相同做法。

接收回调函数通过将其地址存储于 tUARTPort 结构体的 pbReceiveCallback 元素中以链接至驱动程序。如果应用程序不希望监听接收事件，则 tUARTPort 结构中的接收回调函数指针可预置为零。接收操作全部在 ISR 之外执行，下面演示了接收操作 ISR 流程，处理接收中断时将检查错误。

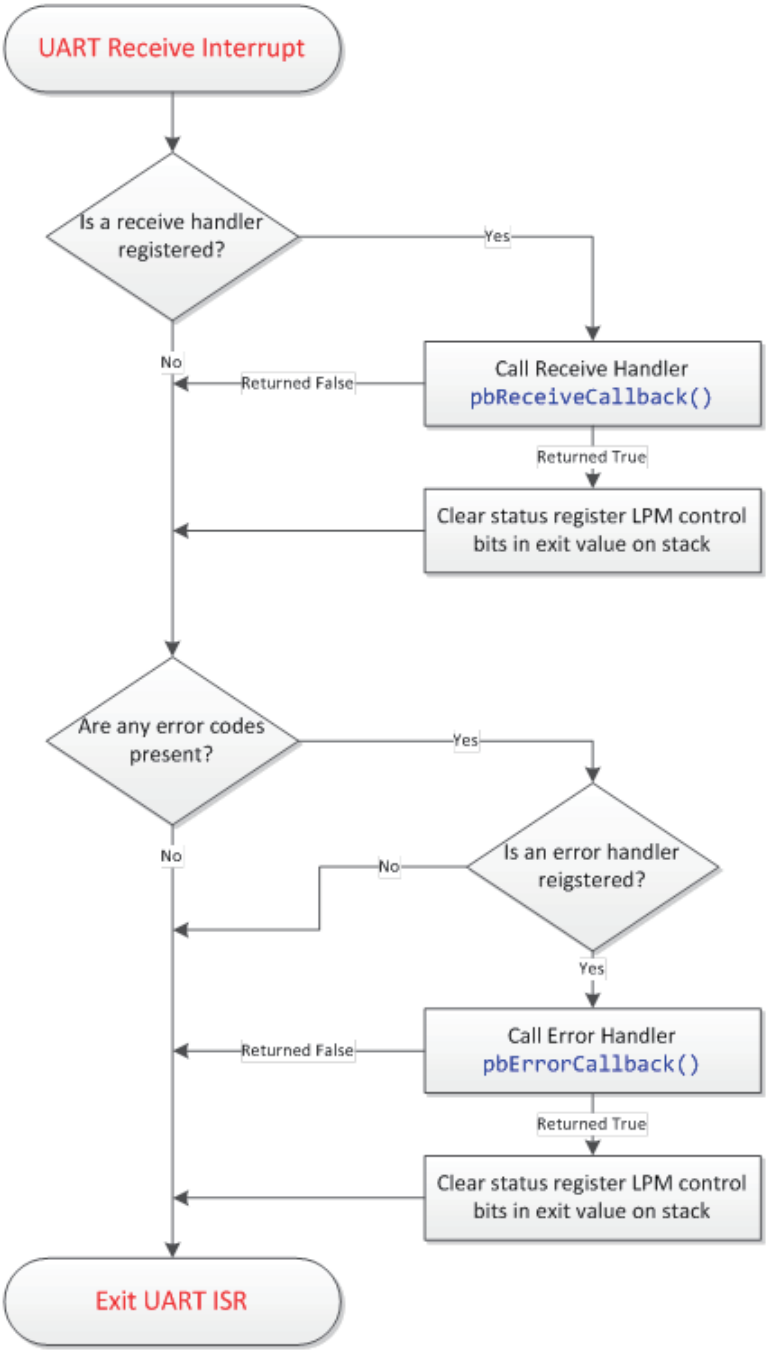


图 132：UART 接收流程图

9.6.5.10 使用驱动程序：错误处理

UART 驱动程序运用基本的 UART 错误检测在出现故障时向应用程序发出警示信号。由驱动程序实施检测的两种错误是：

- UART 接收缓冲区溢出。如果一个新的字节在前一个字节读出之前被置于 eUSCI_A UART 接收缓冲区中，就会出现这种情况。当无法提供以数据输入外设的速率读出数据的足够 CPU 带宽时，这种情况将发生。为了在开发过程中排除该错误，有必要减缓接收数据的速率（不管是通过字节之间的延迟还是降低波特率），或者通过缩短中断处理程序或增高 CPU 时钟频率来提升 CPU 的处理能力。枚举错误标志是 eUARTOverrunError。

- UART 成帧错误。该错误通常出现于器件之间存在波特率失配的情况下，枚举错误标志是 eUARTFramingError。驱动程序通过错误回调函数向应用程序发出警示信号。其工作原理类似于接收回调函数，唯一的差别在于传递的是错误代码而不是接收的字节。错误回调函数从 UART ISR 调用，因此应使其尽可能地短。错误回调函数通过将其地址置于 tUARTPort 结构体的 pbErrorCallback 元素中从而链接至驱动程序。如果应用程序不希望监听错误事件，则 tUARTPort 结构中的错误回调函数指针可预置为零。

9.6.6 I²C Slave 驱动程序

MSP430 eUSCI_B I²C Slave 驱动程序提供了连接 MSP430 应用程序的简单 I²C Slave API，从而实现了中断驱动型 I²C 读 / 写操作以及总线超时检测和驱动程序错误处理。本用户指南概述了此驱动程序的特性、架构和 API。本文件假定读者熟悉 MSP430 MCU 架构、以及嵌入式 C 语言编程概念和基本的 I²C 原理。

注： 在本文件的剩余部分里，I²C Slave 驱动程序被称为驱动程序。

9.6.6.1 相关文件

本文件应与下列的其他支持性文档配套使用。MSP430 驱动程序库 API 并不专门用于于该驱动程序，其具有单独的文档描述。

- MSP430 驱动程序库 (DriverLib) 用户指南
- 相关的 MSP430 系列用户指南

9.6.6.2 驱动程序的用途

该驱动程序可实现 MSP430 应用程序的快速开发，这里，MSP430 本身是更大嵌入式系统中的某个其他主控器的从属器件，MSP430 微控制器常常是一个较大的主机 MCU 或主机 MPU 的辅助 MCU。

该驱动程序的结构设计以拥有灵活性为目标，从而可以实现不同的应用。其能够提供一个连接主机处理器的寄存器文件接口，类似于一个传感器或 EEPROM。另外，它还可以用作一个连接主机的批量传输接口。

9.6.6.3 驱动程序特性

该驱动程序所提供的主要特性如下。

- 中断驱动型 I²C Slave 软件状态机
- I²C 写（master 发送器，slave 接收器）事件回调
- I²C 读（master 接收器，slave 发送器）缓冲区分配
- 支持 I²C 起动 / 停止 / 起动以及 I²C 起动 / 重复起动序列
- 通过一个错误回调函数实现的 I²C 错误 / 警告报告
- 在 I²C 中断之间工作于 LPM3（当使用超时特性时）
- 在 I²C 中断之间工作于 LPM4（当未使用超时特性时）
- 漏极开路 slave 请求信号（以提醒 master 处理 slave）
- slave 请求超时功能用以防止应用程序停止运行
- I²C 总线处理超时功能用以防止应用程序停止运行

9.6.6.4 驱动程序概述

该驱动程序以 C 语言源代码提供。它包括 3 个基本驱动程序文件以及 3 个与 I^2C 超时检测特性（其为一种编译时间选项）相关的源文件。

基本驱动程序：
- 2CSlave_Definitions.h（配置头文件）
- I2CSlave.h（API 头文件）
- I2CSlave.c（API 实现文件）

函数定时器（用于 I^2C 超时检测）
- FunctionTimer_Definitions.h（配置头文件）
- FunctionTimer.h（API 头文件）
- FunctionTimer.c（API 实现文件）

I2CSlave_Definitions.h 和 FunctionTimer_Definitions.h 文件包含用于每个组件的编译选项。

该驱动程序需要下面的 MCU 硬件资源：
- 基本驱动程序
  - 一个 eUSCI_B 外设实体
  - 两个器件引脚（UCBxSDA 和 UCBxSCL），其中的“x”代表选定的 eUSCI_B 实体
- slave 请求特性（可选）
  - 一个额外的器件引脚（用于 slave 请求信号线的数字 IO 口）
- 超时特性（可选）
  - 具有至少两个捕获比较单元（CCR0 和 CCR1）的 TIMER_A 外设实体

该驱动程序的操作基于一个负责追踪当前驱动程序状态的软件状态机。有四种可能的状态，即：关闭、空闲、读和写。由于该驱动程序实现了 I^2C Slave，所以应当明确针对读和写的命令命名规则。从 I^2C slave 的角度来看，I^2C 总线写入是一种接收操作，因为总线 master 对 slave 进行写操作，同样，I^2C 总线读取从 slave 的角度来看是一种发送操作。图 133 表示了由该驱动程序实现的状态机。

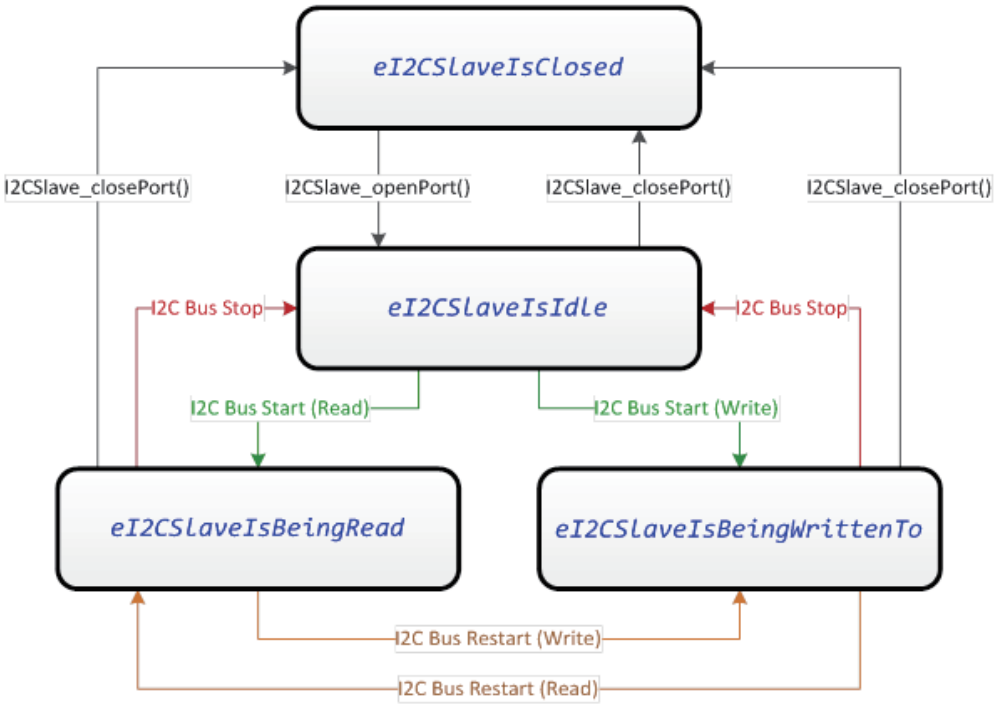


图 133：I²C Slave 驱动程序状态机

如图 133 所示，空闲、读和写之间的状态变更仅受控于 I²C 总线 master。不过，slave 可以随时关闭驱动程序。

9.6.6.4.1 驱动程序概述：I²C 接收操作（I²C 总线写入）

每当总线 master 把一个启动或重启动条件发送至驱动程序的 7 位 I²C 地址时（R/_W 位清零），该驱动程序进入写入状态 (eI2CSlaveIsBeingWrittenTo)。在此状态中，驱动程序将接收数据数据被接收到由用户在打开驱动程序时所指定的缓冲存储器中。如果总线 master 试图在接收缓冲区的长度以外写入数据，则数据被忽略，而且错误回调函数会向应用程序发出警示信号。当发布了一个停止条件或重启动条件时，写入状态被清除，此时驱动程序将调用一个用户指定的回调函数以处理接收的数据。

9.6.6.4.2 驱动程序概述：I²C 发送操作（I²C 总线读取）

每当总线 master 把一个启动或重启动条件发送至驱动程序的 7 位 I²C 地址时（R/_W 位设定），该驱动程序进入读取状态 (eI2CSlaveIsBeingRead)。在此状态中，驱动程序通过从链接至驱动程序的发送缓冲存储器加载数据以把数据传输至 master，该缓冲存储器必须在进入读取状态之前完成预加载。如果总线 master 试图在缓冲区被指定之前读取数据，抑或试图读出多于缓冲区中可用的数据，则驱动程序将由时钟同步输出一个无效字节，其可被规定为一种编译选项。当发布了一个停止条件或重启动条件时，读取状态被清除。在发生该情况时，驱动程序把 CPU 从低功耗模式中唤醒（假如它正在等待某个读取操作的完成）。

9.6.6.4.3 驱动程序概述：Slave 请求特性

该驱动程序提供了一种用于向其希望与之通信的总线 master 发出警示信号的机制，在许多应用中是有帮助的，因为 slave 没有办法在 I^2C 总线上启动通信。Slave 请求特性作为一种漏极开路请求线路来实现，该请求信号线可以是 MCU 上的任一数字 IO 口。此请求信号线应通过上拉电阻上拉至 VCC，驱动程序控制请求信号线 IO 置于高阻态还是下拉至逻辑低电平。驱动程序 API 包含一个用于下拉请求信号线电平的函数，其在返回之前等待来自 master 的 I^2C 总线响应（或超时）。

9.6.6.4.4 驱动程序概述：超时特性（函数定时器）

该驱动程序提供了用于避免应用程序锁定的超时机制。如需以低功耗方式启用超时特性，则要用专用的硬件定时器来设定针对特定的驱动程序操作的时间长度限值。可以设有超时的操作是：

- I^2C Slave 请求超时。slave 请求特性使用用信号线通知总线 master 与 slave 进行通信，slave 请求 API 函数专为锁定至 master 起动通信（通过 I^2C 总线读取启动条件）为止而设计。如果超时特性被启用，则 slave 请求函数将在 master 未在规定的超时周期之内起动读取操作的情况下自动超时。
- I^2C 事务处理超时。事务处理超时对所有 I^2C 总线事务处理可以花费的时长设有限值。事务处理时间被定义为发布一个启动或重启动条件到发布一个重启动或停止条件之间的时间，事务处理超时的目标是检测总线主控器在传输途中停止的场景或 I^2C 总线断连。

超时特性是由函数定时器 (Function Timer) 模块实现的，不受软件的影响。事务处理监测可能是应用层的一部分（也许带有一个标准的看门狗定时器），因此在编译的时候可以完全排除超时特性。Function Timer 模块在定时器周期中预设的延时之后会在前台程序中调用一个预定义函数，该函数会采取行动，如取消 slave 请求或复位驱动程序。

9.6.6.5 驱动程序编译配置选项

编译配置选项在 I2CSlave_Definitions.h 和 FunctionTimer_Definitions.h 文件中设定。这些选项说明如下。

9.6.6.5.1 I^2C Slave 使能定义

I^2C slave 使能编译选项用于选择启用或停用驱动程序（见表 36），其提供了把该驱动程序从编译过程中排除的机制。如需包含该驱动程序，则需把 I2CSlave__ENABLE 定义为“true”，否则将其定义为“false”。

表 36：I^2C Slave 使能定义

参数	文件	有效值
I2CSLAVE__ENABLE	I2CSlave_Definitions.h	true，false

9.6.6.5.2 I^2C eUSCI_B 外设选择定义

I^2C eUSCI_B 外设选择可选择哪个 e_USCI_B 实体与 I^2C 驱动程序相关联（见表 37）。这样可以在需要进行引脚选通变更的情况下提供设计灵活性。为此必须提供一个有效的基地址。

表 37：I^2C e_USCI_B 外设选择定义

参数	文件	有效值
I2CSLAVE__EUSCI_B_PERIPHERAL	I2CSlave_Definitions.h	EUSCI_B0_BASE

9.6.6.5.3 I²C Slave 地址定义

I²C slave 地址规定了与该器件相关联的 7 位 I²C 总线地址（见表 38）。

表 38：I²C Slave 地址定义

参数	文件	有效值
I2CSLAVE__ADDRESS	I2CSlave_Definitions.h	0x00 至 0x7F

9.6.6.5.4 I²C Slave LPM 模式定义

I²C 通信可以在低功耗模式中运行（见表 39）。当处于 I²C slave 模式时，在 e_USCI_B 外设上发送或接收字节不必在 启动 MSP430 时钟，这正是由总线 master 提供位时钟的一个优势。I²C slave LPM 模式控制以下面的方法来使用。

- 接收回调函数可以在一个接收事件结束之后触发 I²C ISR 的主动退出（由一个重启动条件或停止条件触发）。如果接收回调函数回送“真”至驱动程序，则在退出时 I²C slave LPM 模式所规定的状态寄存器位被清零。
- 如果前一个读取正在被请求或在进行之中，则对 I2CSlave_setTransmitBuffer() 的调用将导致阻塞。假如通过 I²C slave LPM 模式定义指定了一种低功耗模式，则阻塞将发生在该低功耗模式中，否则其将阻塞在工作模式中。

表 39：I²C Slave LPM 模式定义

参数	文件	有效值
I2CSLAVE__LPMx_bits	I2CSlave_Definitions.h	LPM0_bits，LPM1_bits，LPM2_bits，LPM3_bits，LPM4_bits

9.6.6.5.5 I²C Slave 无效字节定义

无效字节定义了当 master 试图在驱动程序的发送缓冲区的长度以外进行读取操作时传输至 master（在 I²C 读操作期间）的字节（见表 40）。

表 40：I²C Slave 无效字节定义

参数	文件	有效值
I2CSLAVE__INVALID_BYTE	I2CSlave_Definitions.h	0x00 至 0xFF

9.6.6.5.6 I²C Slave 超时使能定义

Slave 超时使能控制其是否被纳入驱动程序的超时特性（见表 41）。超时特性用来设定 I²C slave 任务在失败之前所需的最大时间。可以利用超时特性来监测的两项任务是 I²C slave 请求及任何的 I²C 事务处理。启用超时特性需要包含 FunctionTimer 源文件、一个 Timer_A 实体及额外的内存。

表 41：I²C Slave 超时使能定义

参数	文件	有效值
I2CSLAVE__TIMEOUT_ENABLE	I2CSlave_Definitions.h	true, false

9.6.6.5.7 I²C Slave 请求超时周期定义

I²C slave 请求超时周期规定了针对 I²C 请求超时的超时周期，单位为函数定时器时钟周期（见表 42）。这是驱动程序在把 slave 请求信号线拉低后（在事务处理失败之前）将等待的时间。总线 master 必须在此时间内响应 slave 请求。

表 42：I²C Slave 请求超时周期定义

参数	文件	有效值
I2CSLAVE__REQ_TIMEOUT_CYCLES	I2CSlave_Definitions.h	0x0000-0xFFFF

9.6.6.5.8 I²C Slave 传输超时周期定义

I²C slave 传输超时周期规定了任何 I²C 事务处理的超时周期，单位为函数定时器时钟周期（见表 43）。I²C 事务处理的定时区间从启动条件至停止条件，或启动条件至重启动条件。

表 43：I²C Slave 传输超时周期定义

参数	文件	有效值
I2CSLAVE__TXFR_TIMEOUT_CYCLES	I2CSlave_Definitions.h	0x0000-0xFFFF

9.6.6.5.9 I²C Slave 请求使能定义

请求使能控制着在驱动程序中是否包含 I²C slave 请求特性（见表 44）。I²C 请求信号线提供了一种让 slave 用信号通知总线 master 发起通信的机制。

表 44：I²C Slave 请求使能定义

参数	文件	有效值
I2CSLAVE__REQ_ENABLE	I2CSlave_Definitions.h	true, false

9.6.6.5.10 I²C Slave 请求信号线定义

I²C 从属请求信号线由三个数值来定义：端口输出寄存器、端口方向寄存器和引脚屏蔽（见表 45）。如果 slave 请求使能被设定为“false”，则这些不是必须定义的。

表 45：I²C Slave 请求线路定义

参数	文件	有效值
I2CSLAVE__REQ_POUT	I2CSlave_Definitions.h	PxOUT
I2CSLAVE__REQ_PDIR	I2CSlave_Definitions.h	PxDIR
I2CSLAVE__REQ_MASK	I2CSlave_Definitions.h	BIT0-BIT7

9.6.6.5.11 函数定时器使能定义

函数定时器使能控制着在驱动程序中是否包含函数定时器（见表 46）。如果不使用 I²C slave 超时特性，则函数定时器被停用时（设定为“false”）保存内存。

表 46：函数定时器使能定义

参数	文件	有效值
I2CSLAVE__REQ_POUT	I2CSlave_Definitions.h	PxOUT
I2CSLAVE__REQ_PDIR	I2CSlave_Definitions.h	PxDIR
I2CSLAVE__REQ_MASK	I2CSlave_Definitions.h	BIT0-BIT7

9.6.6.5.12 函数定时器外设定义

函数定时器外设定义存储着与函数定时器相关联的 Timer_A 实体的基地址（见表 47）。该实体必须具有至少两个捕获比较单元（CCR0 和 CCR1）。

表 47：函数定时器外设定义

参数	文件	有效值
FUNCTIONTIMER__PERIPHERAL	FunctionTimer_Definitions.h	TIMER_A0_BASE，TIMER_A1_BASE，TIMER_A2_BASE，TIMER_A3_BASE

9.6.6.5.13 函数定时器时钟源定义

函数定时器时钟决定了函数定时器的分辨率，以及最大延时（见表 48）。时钟源包括 SMCLK 和 ACLK。如果时钟源改变，函数定时器的有效延时可能发生变化。

表 48：函数定时器时钟源定义

参数	文件	有效值
FUNCTIONTIMER__CLOCK	FunctionTimer_Definitions.h	TASSEL__SMCLK，TASSEL__ACLK

9.6.6.5.14 函数定时器时钟分频器定义

函数定时器时钟分频器对源时钟（在上文中做了规定）进行下分频。如果时钟分频器改变，则函数定时器的有效延时可能发生变化（见表 49）。

表 49：函数定时器时钟分频器定义

参数	文件	有效值
FUNCTIONTIMER__DIVIDER	FunctionTimer_Definitions.h	ID__1，ID__2，ID__4，ID__8

9.6.6.5.15 函数定时器扩展时钟分频器定义

函数定时器扩展时钟分频器允许增加与标准分频器相串联的第二个分频器。如果扩展时钟分频器改变，则函数定时器的有效延时可能发生变化。

表 50：函数定时器扩展时钟分频器定义

参数	文件	有效值
FUNCTIONTIMER__EXDIVIDER	FunctionTimer_Definitions.h	TAIDEX_0 至 TAIDEX_7

9.6.6.5.16 函数定时器 LPM 清零定义

函数定时器 LPM 清零定义控制着从前台程序中调用的函数退出时由函数定时器清零哪些 LPM 位。该设定可以在一个超时事件之后唤醒 CPU ，函数定时器从不使用这些位来进入低功耗模式，仅在退出 LPM 时使用它们。

表 51：函数定时器 LPM 清零定义

参数	文件	有效值
FUNCTIONTIMER__LPM_CLEAR	FunctionTimer_Definitions.h	LPM0_bits，LPM1_bits，LPM2_bits，LPM3_bits，LPM4_bits

9.6.6.6 驱动程序运行配置选项

驱动程序的运行配置选项是通过在应用程序中添加一个 tI2CSlavePort 结构体，并在打开驱动程序时把这个结构体传递至驱动程序来实现的。tI2CSlavePort 结构体在下文中讨论。

表 51：函数定时器 LPM 清零定义

Member	Description	Valid Values
bool (*pbReceiveCallback)(uint16_t)	pbReceiveCallback is a function pointer that may point to a receive event handler in the application. If no receive handling is required, initialize this member to 0 (null).	Null, or a valid function address.
void (*pvErrorCallback)(uint8_t)	pvErrorCallback is a function pointer that may point to an error event handler in the application. If no error handling is required, initialize this member to 0 (null).	Null, or a valid function address
ui16ReceiveBufferSize	This member stores the size of the receive buffer pointed to by pReceiveBuffer.	0x00 to 0xFF
pReceiveBuffer	This member is a pointer to the I²C receive buffer.	A valid pointer
bSendReadLengthFirst	When set to true, this flag configures the driver to always load the length of the transmit buffer as the first data byte read by the bus master. This is useful when variable length bulk packets are being read out by the master, and the master needs to know how many bytes to read from the slave.	true, false

假如在驱动程序构包含了超时特性，还会产生函数定时器运行配置，但是它在 I²C 端口被打开时将由驱动程序自动地处理。为了完整性，表 53 说明了函数定时器运行配置结构体 (tFunctionTimer)，但是应用程序无需了解函数定时器的任何详情。

表 53：定时器运行配置结构体

Member	Description	Valid Values
ui16FunctionDelay_A	This member specifies the length of the delay (in timer clock cycles) before function A is called.	0x0000 to 0xFFFF
bool (*pbFunction_A)(void)	pbFunction_A is a function pointer to function A.	A valid function pointer, else null
ui16FunctionDelay_B	This member specifies the length of the delay (in timer clock cycles) before function B is called.	0x0000 to 0xFFFF
bool (*pbFunction_B)(void)	pbFunction_B is a function pointer to function B.	A valid function pointer, else null.

9.6.6.7 使用驱动程序：打开和关闭驱动程序

I²C slave 驱动程序的打开和关闭是通过表 54 所列的函数来完成的。

表 54：打开和关闭驱动程序

说明	声明
打开 I2C slave 端口	extern void I2CSlave_openPort(const tI2CSlavePort *pPort)
关闭 I2C slave 端口	extern void I2CSlave_closePort(void)

该驱动程序通过调用 I2CSlave_openPort() 进行初始化并做好使用准备，其传递了一个完整的 tI2CSlavePort 结构体。该 tI2CSlavePort 结构体必须由应用程序添加，驱动程序在任何时候都不修改此结构体，因此可以把该结构体存储在一个只读存储器（例如：C 常数存储区段）中。请注意，应把该结构体存储在存储器中传递至 I²CSlave_openPort() 的原始地址，因为驱动程序将参考该结构体并在端口打开时访问回调函数。如果必须释放存储器，首先需要调用 I2CSlave_closePort() 来关闭驱动程序，对 I2CSlave_closePort() 的调用将停用驱动程序及其关联的 eUSCI_B 外设，并停止所有的 I²C slave 活动。如果在程序中包含了超时特性，则 I2C_closePort() 将停用负责驱动超时特性的函数定时器模块。在该端口关闭之后，则不再调用事件处理程序，并可以把 tI2CSlave 结构体内存释放到应用程序。

9.6.6.8 使用驱动程序：发送数据（I²C 读操作）

数据发送是通过下面的函数调用来完成的：

表 55：传输数据（I²C 读操作）

说明	声明
设定传输缓冲区	extern void I2CSlave_setTransmitBuffer(uint8_t *pBuffer, uint16_t ui16Length)
设定请求标志	extern void I2CSlave_setRequestFlag(void)
获得端口状态	extern uint8_t I2CSlave_getPortStatus(void)

一个 I²C slave 不能独立地把数据推送到总线上，master 必须对 slave 进行寻址并由时钟同步输出数据，因此，在总线 master 试图对其进行读取操作之前必须提供数据缓冲区供驱动程序使用。应用程序可以把一个指针和一个长度量（以字节为单位）传递至 I2CSlave_setTransmitBuffer() 函数，该函数将使其阻塞（如果一个当前事务处理正在进行之中）和返回（当传输缓冲区指针和长度量已被更新时）。如果未提供发送缓冲区且总线 master 试图从 slave 读取数据，则其将读出无效的字符（一种编译选项）。

I2CSlave_setTransmitBuffer() 函数并不执行缓冲器的复制，而是仅仅存储其位置和长度量。这对于寄存器应用是非常有价值的，在此类应用中，应用程序只更新缓冲区， I2CSlave_setTransmitBuffer() 在初始化期间仅需要调用一次。从总线 master 重复地的读取数据将重读同一个缓冲区，直到 I2CSlave_setTransmitBuffer() 被再次调用为止。在传输完成之前应用程序并不重写传输缓冲区空间，这一点很重要。

如果 slave 请求特性被使能，则可以通过调用 I2CSlave_setRequestFlag() 函数向 master 发送信号。该函数立即拉低请求信号线，然后在其返回（或超时）之前等待 master 开始执行一个读操作。

驱动程序状态可以通过调用 I2CSlave_getPortStatus() 由应用程序随时获得。该函数返回当前的 I²C slave 状态。可能的状态由 tI2CSlaveStates 枚举。表 56 列出了可能的枚举。

表 56：tI2CSlaveStates 枚举

端口状态选项	说明
eI2CSlaveIsClosed	驱动程序关闭。除了 I2CSlave_openPort() 和 I2CSlave_getPortStatus() 以外的所有函数都是无效的。
eI2CSlaveIsIdle	驱动程序打开，但是当前没有 I²C 事务处理在进行。
eI2CSlaveIsBeingRead	驱动程序打开，而且总线 master 正在读取数据。驱动程序从发送缓冲区加载数据，直到所有数据皆已发送为止，然后它装入无效的字节。
eI2CSlaveIsBeingWrittenTo	驱动程序打开，而且总线 master 正在写入数据。写入的数据被置于接收缓冲区中（如果有空间的话）。在该状态结束时，调用应用程序中的接收回调函数。

9.6.6.9 使用驱动程序：接收数据（I²C 写操作）

总线 master 可以随时启动一个写操作，驱动程序将把输入数据缓存至接收缓冲区中。当事务处理完成时，调用应用程序接收回调函数，该回调函数传递从 master 接收的数据的大小（以字节为单位）。由于它是被驱动程序中断服务程序调用的，因此不处理其他的中断，而且 eUSCI_B 可能会延伸总线时钟，直到驱动程序从接收回调返回为止，为回调函数提供了处理接收数据及在 master 有可能继续通信之前更新发送缓冲区的机会。

9.6.6.10 使用驱动程序：错误处理

驱动程序提供了一个错误回调函数，以在驱动程序出现问题的情况下向应用程序发出警示信号。该错误回调函数在被调用时传递一个指示错误的数值。可能的错误列举如下。

表 57：错误处理

端口状态选项	说明
eI2CSlaveTransmitRequestTimeout	该代码指示：向总线主控器发送的一个 slave 请求未在超时窗口之内得到服务，而且该请求超时。
eI2CSlaveTransactionTimeout	该代码指示：一项 I^2C 事务处理所花费的时间长于超时窗口，而且事务处理超时。另外，该错误还导致一次驱动程序复位。
eI2CSlaveReceiveBufferFull	该代码指示：接收缓冲区在最后一个读事务处理期间已满，而且来自总线 master 的数据丢失。
eI2CSlaveWasReadBeyondTransmitBuffer	该代码指示：总线 master 试图在有效传输缓冲区长度以外从 slave 读取数据（表示 master 在读取无效的字节）。

9.7 性能比较

详细的性能比较数据请参照英文最新版 CapTIvate™ Technology Guide 中相关章节的介绍。

对于 CapTIvate™ Technology Guide v1.40.00.00，请参照章节“Software Library-> Benchmarks”。

10 MSP-CAPT-FR2633 开发套件

10.1 引言

电容式触摸快速原型设计从未象现在这样更快和更容易。使用 MSP430FR2633 MCU 模块化套件和 CapTIvate Design Center 来评估 MSP430FR2633 MCU 的性能（采用电容式触摸演示板），或者开发 PCB 并体验 CapTIvate Design Center 的强大功能，以及实时在线的触摸传感器参数调整的简单易行，所有这些都不需要编写一行代码。另外，还提供了免费的软件开发工具，包括 TI 基于 Eclipse 的 Code Composer Studio (CCS) 和 IAR Embedded Workbench® IAR-KICKSTART。 当 与 MSP430FR2633 MCU 开发套件配合使用时，这两种集成型开发环境 (IDE) 均支持 EnergyTrace 技术。

10.2 概述

MSP-CAPT-FR2633 MCU 开发套件是一款面向支持 CapTIvate 电容式触摸技术的 MSP430FR2633 微控制器的易用型评估平台。其包含了在基于 MSP430™ 超低功耗 (ULP) FRAM 的微控制器 (MCU) 平台上启动开发工作所需的一切资源，包括用于编程、调试和能耗测量的板载仿真。MSP430FR2633 MCU 运用了一种新型 CapTIvate 电容式触摸技术，可支持自电容和互电容式传感器，以及一种因其超低功耗、高耐用性和高速写访问等特点而著名的非易失性存储器 — 嵌入式 FRAM（铁电随机存取存储器）。

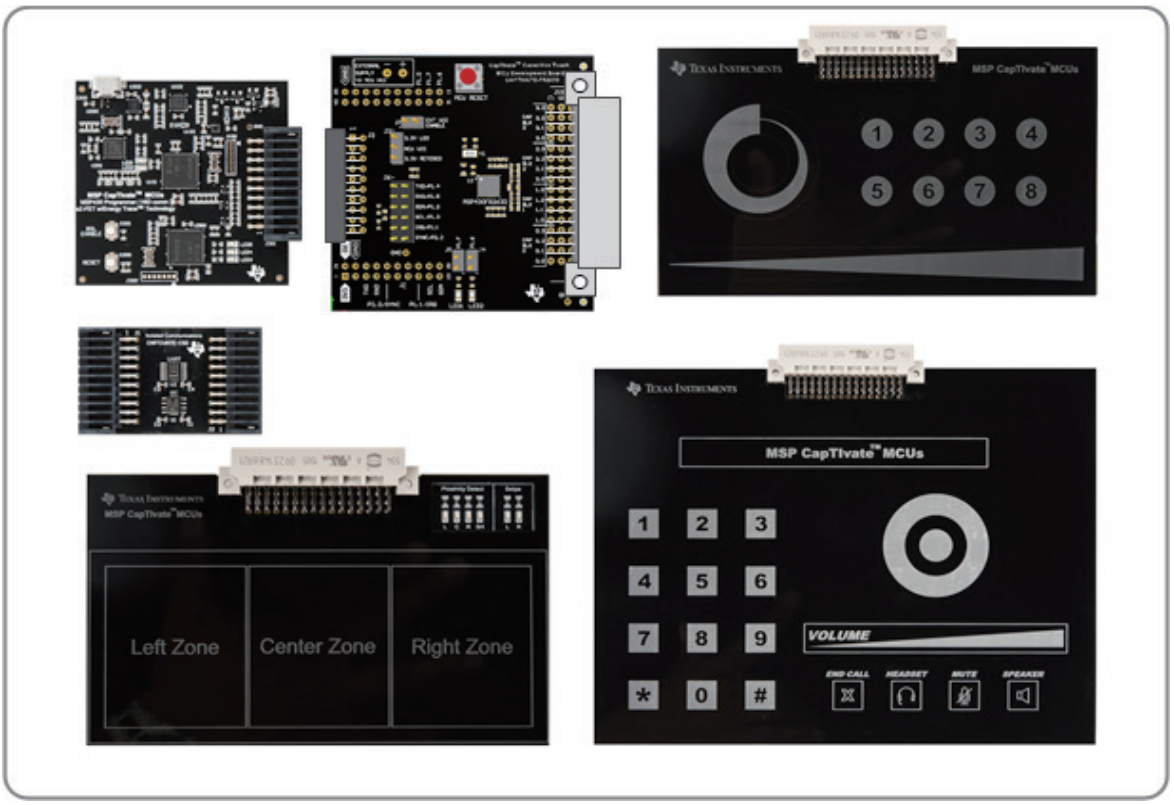


图 134：套件的组成

10.3 MCU 开发套件的主要特性

- MCU PCB
  - 支持 CapTIvate 电容式触摸技术的 MSP430FR2633 超低功耗 FRAM 微控制器
  - 多达 16 个 CapTIvate I/O 支持自电容和互电容式传感器的任意组合
    - 16 个自电容电极
    - 64 个互电容电极
- 编程器 PCB
  - eZ-FET — 一种支持 EnergyTrace 技术的开源板载调试器
  - 专用 HID 串行通信桥接器 — 无需驱动程序
- 隔离 PCB
  - 用于电池供电或传导噪声测试中的隔离式通信
- 三个电容传感器演示 PCB
  - 自电容按钮、滑块、滚轮
  - 互电容按钮、滑块和滚轮
  - 高分辨率自电容接近传感器

10.3.1 套件 PCB 的组成

- CAPTIVATE-FR2633 MCU
- CAPTIVATE-PGMR — 具有 HID 通信功能的编程器
- CAPTIVATE-ISO — 隔离式通信板
- CAPTIVATE-BSWP — 自电容传感器演示（“开箱即用”体验）
- CAPTIVATE-PHONE — 互电容传感器演示
- CAPTIVATE-PROXIMITY — 接近传感器演示

10.3.2 软件示例

在 CapTIvateDesignCenterWorkspace 目录中提供了用于每个演示面板的一个 CCS 和 IAR 项目示例。

10.4 硬件

10.4.1 CAPTIVATE-FR2633 处理器 PCB 概述

图 136 和图 137 分别示出了 CAPTIVATE-FR2633 PCB 及其方框图。

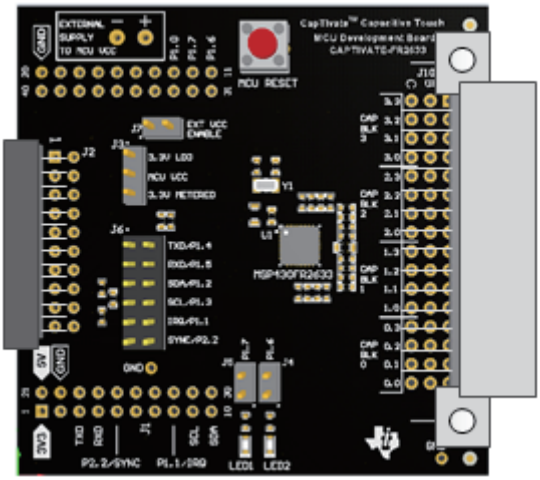


图 136：CAPTIVATE–FR2633 处理器 PCB

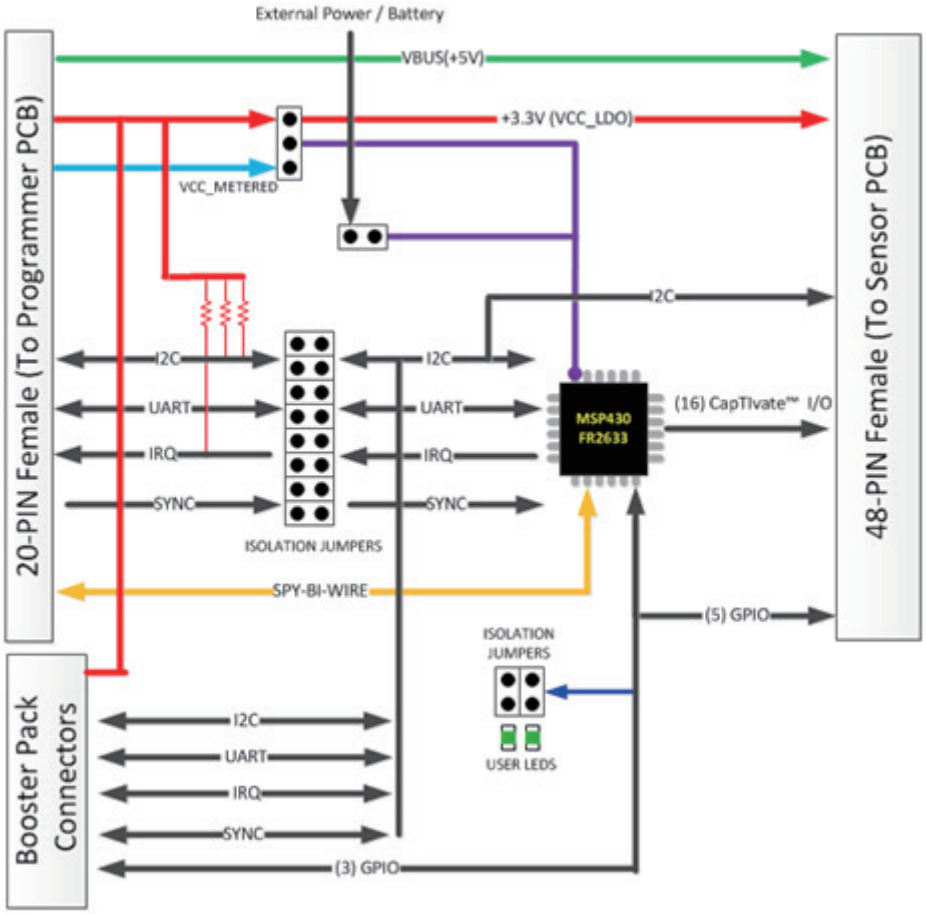


图 137：CAPTIVATE–FR2633 处理器 PCB 方框图

10.4.1.1 特性

- 支持 CapTIvate 电容式触摸技术的 MSP430FR2633 MCU
  - 15.5 kB FRAM，4 kB RAM
  - 12 kB ROM（BSL、CapTIvate 软件库、HID 通信、以及 MSP430 driverlib I^2C 和 UART 驱动程序）
  - 16 个 CapTIvate I/O
- 48 针传感器 PCB 连接器
  - 16 个 CapTIvate I/O
  - 5 个 GPIO
  - I2C
  - 3.3 V LDO
  - 5 V (VBUS)
  - 支持未来具有 32 个 CapTIvate I/O 的 CapTIvate 器件
- 20 针编程器、电源和通信连接器
  - 两线制 (Spy-Bi-Wire) 目标 MCU 编程
  - 采用 HID 桥接器的串行通信
  - 两个 3.3 V 电源轨
  - 5 V USB

- 40 引脚 BoosterPack 连接器
  - 电源、UART、I^2C 和 3 个 GPIO
- 可选的 VCC 电源跳线 J3
  - 从编程器 PCB 引出的可选 VCC 电源
    - 3.3 V（用于利用 EnergyTrace 技术进行的能耗测量）
    - 3.3 V LDO 向目标 MCU 和传感器 PCB 连接器提供高达 250 mA 的电流
  - 外部电源或电池

10.4.1.2 支持 CapTIvate 电容式触摸技术的 MSP430FR2633 MCU

MSP430FR2633 能采用 1.8 V 至 3.6 V 的电压工作，具有高达 16 MHz 的系统时钟和 8 MHz FRAM 存取能力、15.5 kB 非易失性 FRAM 和 4 kB 数据 RAM。为了帮助最大限度地增加可提供给应用程序使用的 FRAM 代码存储器容量，CapTIvate 软件触摸库、HID 通信、以及 MSP430-Driverlib I^2C 和 UART 驱动程序在 12 kB ROM 中提供，并可通过一个简单的 API 进行访问。更多详情请查阅“软件库 API”。

CapTIvate 电容式触摸技术支持 16 个 I/O 引脚上的自电容和互电容式测量，并具有并行扫描能力和多达 64 个电极。这种灵活性为设计人员在单一设计中混合使用自电容和互电容型按钮、滑块、滚轮和接近传感器提供了自由度。

10.4.1.3 传感器 PCB 连接器（48 针凹型）

凹形连接器（见图 138）使得 MSP430FR2633 所有的 16 个 CapTIvate I/O 通道、I^2C、3.3 V、5 V 及若干个通用 I/O 可用。该连接器的中间一排保留，以用于将来支持多达 32 个 CapTIvate I/O 通道的器件。在该 PCB 上，B 排上的所有插针均接地。

如需了解有关该连接器之 48 针凸形版本的信息，请参阅 10.4.7 节。

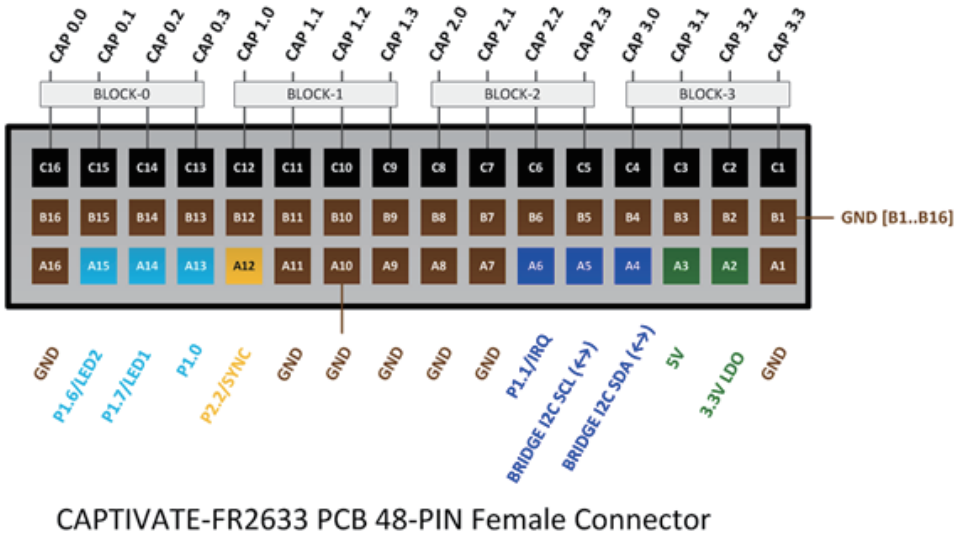


图 138：48 针凹形传感器 PCB 连接器输出针脚

10.4.1.4 编程、电源和通信连接器（20 针凹形）

该凹形连接器专为与 CAPTIVATE-PGMR 编程器 PCB 相连接而设计。此连接器在两个 PCB 之间提供电源、编程和通信。

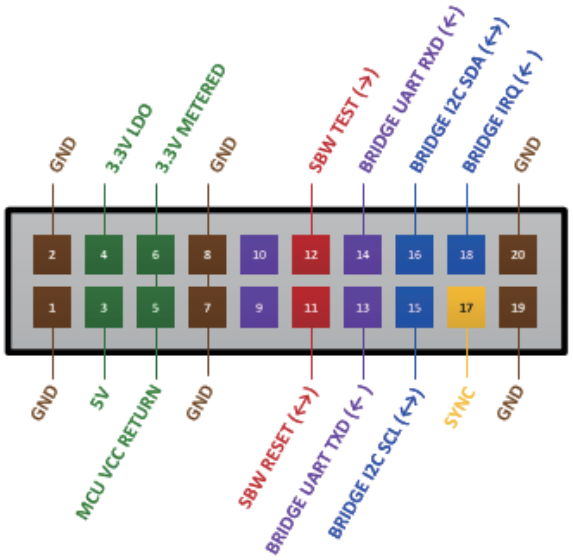


图 139：20 引脚凹形编程器连接器

10.4.1.5 BoosterPack 连接器焊脚

CAPTIVATE-FR2633 提供了标准的 40 针 BoosterPack 连接器焊脚，并可提供 3.3 V、5 V、UART、I^2C 和 5 个 GPIO 信号（见图 140）。

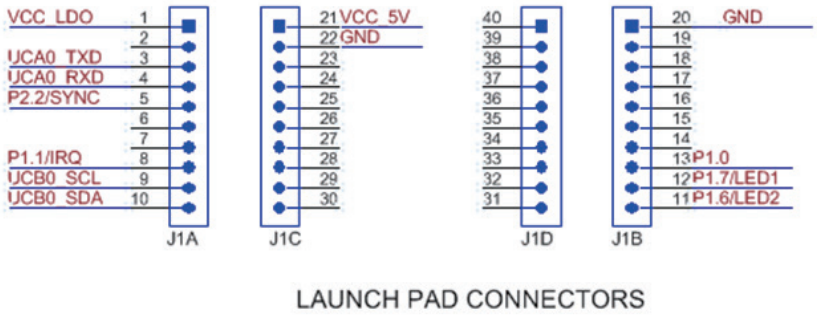


图 140：40 针 BoosterPack 连接器

建议的 BoosterPack 排针 (header) 和连接器：

- SAMTEC DW-10-15-F-D-210，排针，凸形
- SAMTEC SSQ-110-23-G-D，连接器，凹形

10.4.1.6 电源

CAPTIVATE-FR2633 PCB 具有两个可通过 20 针连接器 J2 获得的电源（当连接至 CAPTIVATE-PGMR PCB 时）。一个电源是负责给用于诸如 LED 驱动器或传感器 PCB 上的触觉振动驱动器的 48 针连接器 J10 供电的 3.3 V (VCC_LDO)。第二个电源是支持 EnergyTrace 技术 DC/DC 3.3 V 输出 (VCC_METERED) 的 eZFET。

10.4.1.6.1 目标 MCU 电源

把跳线 J3 定位到引脚 (1-2) 以选择 3.3 V LDO 电源用于目标 MCU。

10.4.1.6.2 测量目标 MCU 电源

当采用 CCS 或 IAR IDE 时，EnergyTrace 技术使得能够测量目标 MCU 所消耗的功率。把跳线 J3 定位到引脚 (2-3) 以选择 3.3 V METERED 电源用于目标 MCU。

10.4.1.7 编程和调试

该 CAPTIVATE-FR2633 PCB 上的 MSP430FR2633 MCU 专为通过其两线制 (Spy-Bi-Wire) 接口进行编程和调试而设计。在该 PCB 上不能提供四线 JTAG 连接。

注： 在 CAPTIVATE-FR2633 上不支持 eZFET 反向通道 UART 功能。

10.4.1.8 通信

MSP430FR2633 MCU 与一个位于 CAPTIVATE-PGMR PCB 上的专用 USB HID 桥接器 MCU 进行通信，其采用 UART 或 I^2C 通信以把传感器数据和状态发送至 CapTIvate Design Center，作为传感器设计和参数调整过程的一部分。作为 CapTIvate 软件库的组成部分，与 UART 和 I^2C 驱动程序一起提供了一种紧凑型通信协议。两者皆位于 MSP430FR2633 ROM 中，以最大限度地减轻对 FRAM 内存占用的影响。通信协议在 10.6 节中说明。

当与 CapTIvate 协议一起使用时，UART 工作于一种全双工模式（采用 RX 和 TX 引脚），而 I^2C 则起一个 I^2C 受控器的作用（采用 SDA 和 SCL 引脚及一个额外的引脚 P1.2/IRQ 以生成中断请求）。

提供了跳线，以在 MSP430FR2633 与 20 针编程和通信连接器之间提供隔离（如果一个或多个 I/O 引脚必须改作他用）。在跳线的隔离侧上提供了用于 I^2C、SDA、SCL 和 IRQ 信号的上拉电阻器（见图 141 和表 58）。

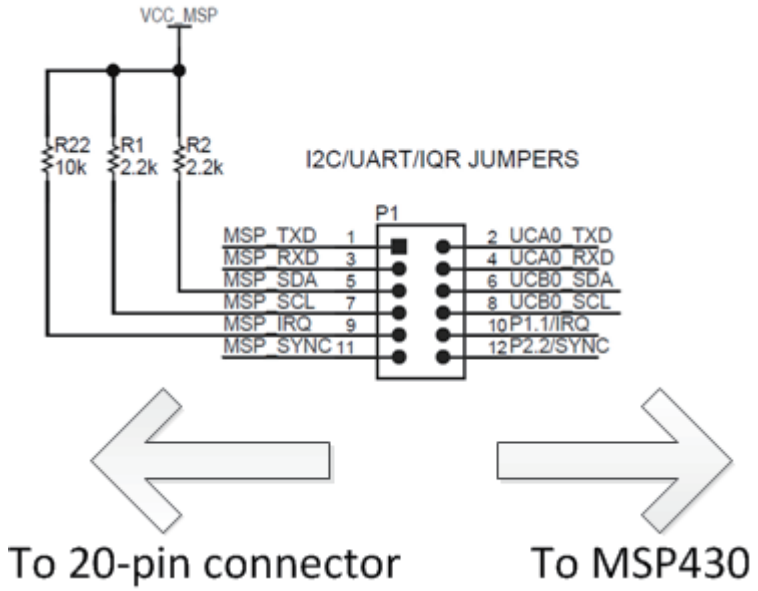


图 141：跳线

表 58：通信隔离跳线

跳线	说明	交替使用
TXD	MSP430FR2633 UART 数据输出	P1.4
RXD	MSP430FR2633 UART 数据输入	P1.5
SDA	MSP430FR2633 I2C 数据输入 / 输出	P1.2
SCL	MSP430FR2633 I2C 时钟输入	P1.3
IRQ	MSP430FR2633 I2C 从属中断输出	P1.1
SYNC	MSP430FR2633 CapTIvate 同步输入	P2.2

10.4.1.9 扩展 I^2C 总线

在 48 针传感器 PCB 连接器上也存在 I^2C 信号。I^2C 使得 MSP430FR2633 能够控制 I^2C 从属器件，比如有可能安装在传感器 PCB 上的 LED 或触觉振动驱动器。

10.4.1.10 电源选择

VCC_LDO 和 VCC_METERED 均布线至一个选择跳线 J3，后者负责给 MSP430FR2633 MCU 供电（见图 142）。对于大多数应用，建议采用 VCC_LDO，因为它可提供一个经过优良调节的输出。当选择 VCC_METERED 输出时，MCU 所消耗的功率可运用 CCS IDE 中的 EnergyTrace 技术进行测量。

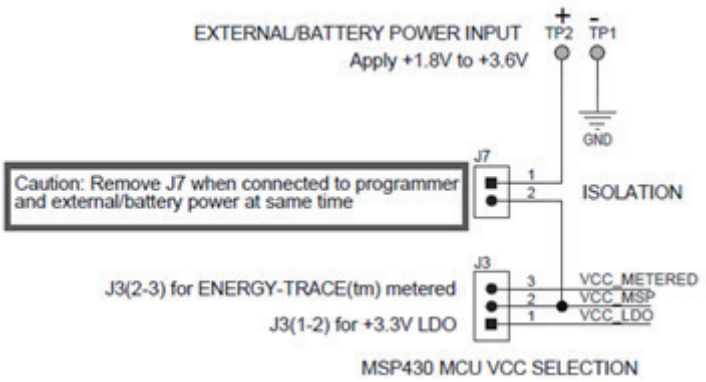


图 142：MSP430 VCC 选择

10.4.1.11 外部电源或电池电源

对于电池供电型应用，MSP430FR2633 可依靠一个外部 2.0 V 至 3.6 V 电源（位于在 PCB 上提供的外部电源组）供电。这允许在一种从任何系统或大地接地点浮动的环境中对 CAPTIVATE-FR2633 PCB 和传感器 PCB 进行评估。

警告

当评估采用一个外部电源的 MSP430FR2633 时，必须完全去除 CAPTIVATE-FR2633 PCB 上的跳线 J3。这可避免在取自 CAPTIVATE-PGMR PCB 的 VCC_LDO 或 VCC_METERED 与外部电源之间发生任何冲突。然而，作为任何标准开发或调试过程的一部分，CAPTIVATE-PGMR 和 CAPTIVATE-FR2633 PCB 可通过 CAPTIVATE-ISO PCB 进行连接。采用 CAPTIVATE-ISO PCB 可提供编程和调试以及与目标 MCU 的 I2C 和 UART 通信。

10.4.2 CAPTIVATE-PGMR 编程器概述

图 143 和图 144 分别示出了 CAPTIVATE-PGMR 编程器 PCB 和方框图。

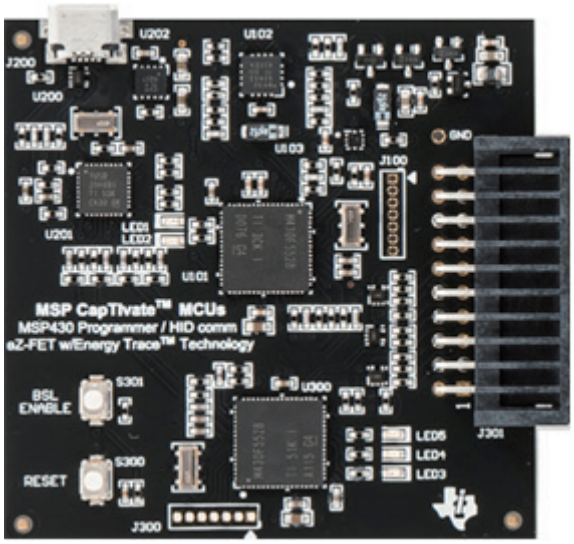


图 143：CAPTIVATE–PGMR PCB

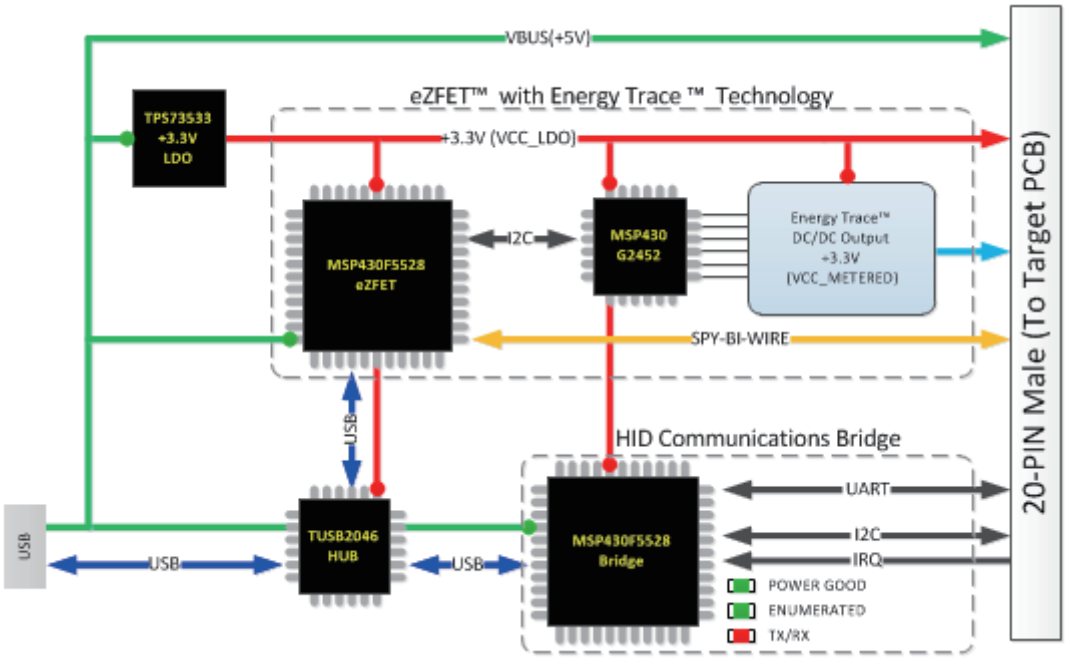


图 144：CAPTIVATE–PGMR PCB 方框图

10.4.2.1 特性

- 支持 EnergyTrace 技术的 eZFET
  - 简单的两线制 (Spy-Bi-Wire) 目标 MCU 编程
  - 可用于对任何 MSP430 进行编程
  - 单独的 ＋ 3.3 V 输出可用于 MCU
    - EnergyTrace
    - 专用 LDO
- 20 针编程器 / 电源 / 通信连接器
  - 两线制 (Spy-Bi-Wire) 接口
  - 与目标的 UART 和 I^2C 串行通信
  - 两个 3.3 V 电源轨
  - 5 V USB
- USB HID 串行桥接器
  - 专用 MSP430F5528
  - 在 CapTIvate Design Center 和目标 MCU 之间提供了接口
  - 支持 UART 和 I^2C
  - HID — 无需安装驱动程序
  - 支持高达 250k 的波特率
  - 可通过 BSL 实现简易的更新

10.4.2.2 编程、电源、通信连接器（20 针凸形）

图 145 示出了与 CAPTIVATE-FR2633 MCU PCB 相连接的凸形连接器。该连接器在两个 PCB 之间提供电源、编程和通信

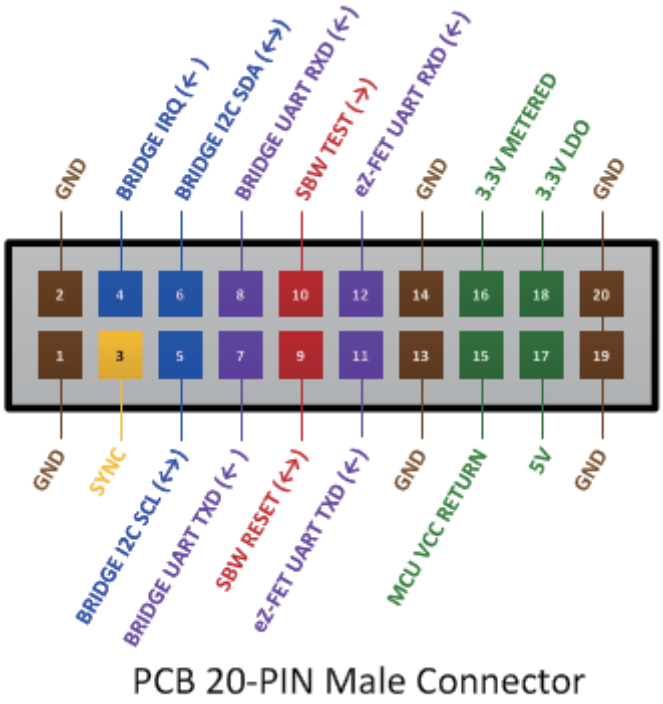


图 145：20 针凸形编程器连接器针脚分配

10.4.2.2.1 电源

至 CAPTIVATE-PGMR PCB 的电源通过 USB 连接器提供（在大约 5VDC）。一个 TI TPS73533 3.3 V、500 mA LDO 负责给 CAPTIVATE-PGMR 上的所有器件供电。该 3.3 V 输出参考于 VCC_LDO，并可在连接时供 CAPTIVATE-FR2633 PCB 上的 MSP430FR2633 使用。另外，支持 EnergyTrace 技术的 eZFET 还提供了一个 DC/DC 3.3 V 输出。该输出参考于 VCC_METERED，而且仅可供目标 MCU 用于能耗测量。CAPTIVATE-FR2633 PCB 上的一根跳线在这两个电源之间进行选择以用于 MSP430FR2633。

注： CAPTIVATE-PGMR 被视为低功耗 USB 器件，因此所吸收的电流应至多为 100 mA。CAPTIVATE-PGMR PCB 的标称吸收电流为 60 mA。这允许任何目标吸收高达 40 mA 的电流。

10.4.2.2.2 编程 / 调试

eZFET 通过其两线制 (Spy-Bi-Wire) 接口提供编程和调试。eZFET 反向通道 UART 功能可在该 PCB 上使用，然而，CAPTIVATE-FR2633 并不支持此项功能。

10.4.2.2.3 借助 CAPTIVATE-ISO PCB 来使用 Spy-Bi-Wire (SBW) 编程

请参阅“JTAG 和 SBW 局限性”。

10.4.2.3 USB HID 串行桥接器

这里是该桥接器的要点：

- 支持 UART 和 I^2C 接口
- 运用 CapTIvate 协议进行出厂编程
- 无需 USB 驱动程序
- 可采用 USB BSL 实现简易的固件更新

10.4.2.3.1 通信

CAPTIVATE-PGMR PCB 具有一个 MSP430F5528 桥接器 MCU，其作为一个 USB HID 器件进行枚举，且不需要在 PC 上安装任何驱动程序。MSP430F5528 的工作频率为 25 MHz，支持 I^2C 和 UART 接口，并利用一个紧凑型通信协议进行了出厂编程，用以在目标 MCU 和 CapTIvate Design Center 之间发送传感器数据和状态。如需了解有关该通信协议的详情，请参阅 10.6 节。

10.4.2.3.2 BSL

MSP430F5528 桥接器 MCU 支持采用 USB BSL 的固件更新。如欲更新现有的固件或加载一个用户定义的固件映像，则使用与 MSP430ware 一起提供的 Python Firmware Upgrade 实用程序。根据 MSP430ware 在计算机上的安装位置，导航至 ...\MSP430ware...\usblib430\Host_USB_Software\Python_Firmware_Upgrader 目录。

- 把 MSP430F5528 置于 BSL 模式中
  - 按下并保持 RESET 铵钮 (S300)
  - 按下 BSL 按钮 (S301)
  - 释放 RESET 按钮
  - 释放 BSL 按钮
- 启动固件升级实用程序
  - 双击该实用程序图标（该实用程序扫描 USB 总线，以寻找 MSP430F5528 在 BSL 模式中的某种特定的 VID/PID 组合）。
  - 如果未发现器件，则重新上面的步骤以把 MSP430F5528 置于 BSL 模式并在实用程序菜单中选择 File → Rescan 总线。
  - 一旦完成，即从实用程序菜单选择 File → Open 用户固件，以选择固件映像并开始更新过程。

10.4.3 CAPTIVATE-ISO（通信隔离 PCB）

图 146 和图 147 分别示出了 CAPTIVATE-ISO PCB 和方框图

图 146：CAPTIVATE–ISO PCB

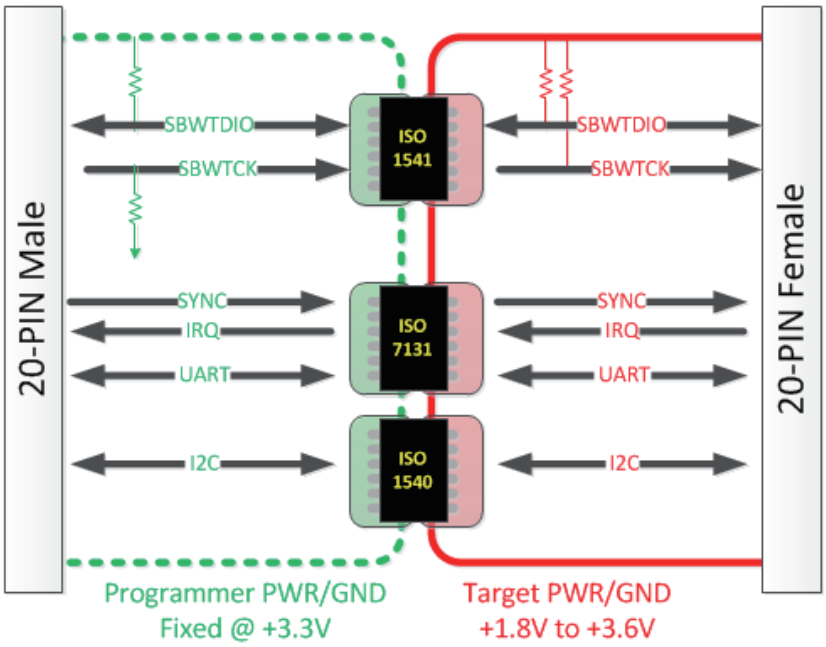


图 147：CAPTIVATE–ISO PCB 方框图

10.4.3.1 特性

- 为 SBW 编程和调试、I^2C 和 UART 通信提供了电流隔离。
  - TI 隔离器件
- 用于两线制 (Spy-Bi-Wire) 的 ISO1541
- 用于 UART 的 ISO7131
- 用于 I2C 的 ISO1540
  - 没有共用的电源或接地
- 当执行以下操作时使用
  - 调试电池供电型应用
  - 传导噪声测试

10.4.3.2 隔离式 JTAG Spy-Bi-Wire

CAPTIVATE-ISO PCB 提供了隔离式 Spy-Bi-Wire 编程和调试。由于在 SBW 定时中增加了延迟，因此 TI 建议采用默认的中等 JTAG/SBW 速度（或更低）。此时不支持 JTAG/SBW 速度 = FAST。

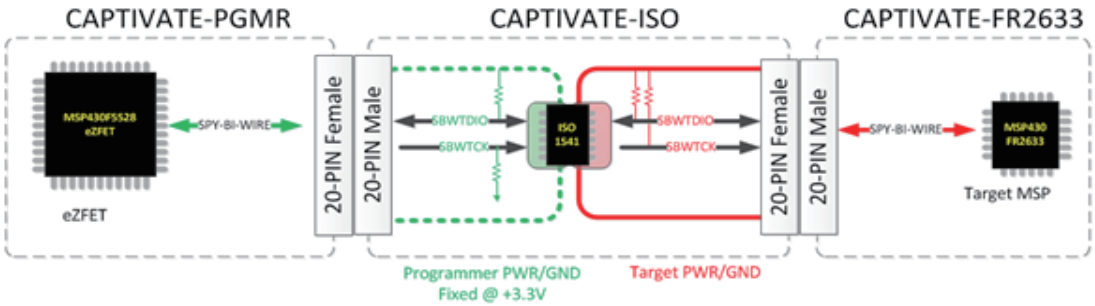


图 148：SBW 信号

10.4.4 CAPTIVATE-BSWP（自电容演示 PCB）

图 149 示出了 CAPTIVATE-BSWP 自电容演示 PCB。

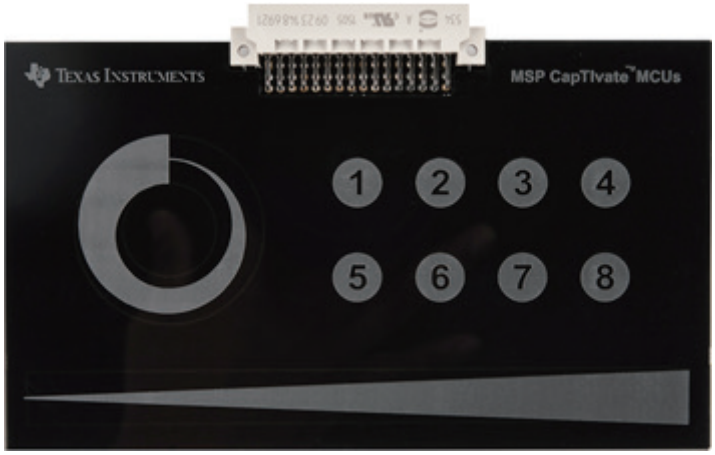


图 149：CAPTIVATE–BSWP 自电容演示 PCB

- 演示可利用自电容滑块和滚轮实现的较高灵敏度和分辨率
- 运用 MSP430FR2633 上的所有 16 个 IO
- 为客户提供一种用于滑块和滚轮的参考布局
- 特性：
  - 8 按钮小键盘
  - 4 按键滑块
  - 3 按键滚轮
  - 接近环形传感器

10.4.5 CAPTIVATE-PHONE（互电容演示 PCB）

图 150 示出了 CAPTIVATE-PHONE 互电容演示 PCB。

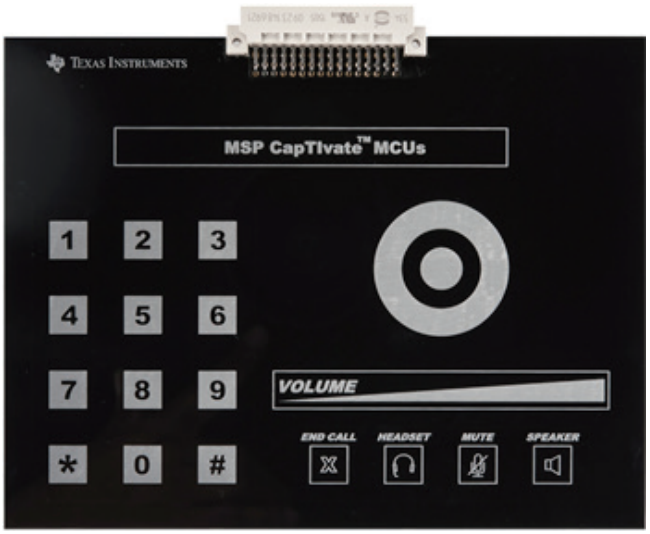


图 148：SBW 信号

- 演示互电容在某种应用中的优势
- 使得客户能够评估 MSP430FR2633 MCU 互电容性能
- 演示集成的触觉振动效果
- 演示一个组合面板 (complex panel)，充分展现软件库和 CapTIvate Design Center
- 特性：
  - 12 按钮数字小键盘
  - 4 按钮模式小键盘
  - 4 按键应用滑块
  - 4 按键音量滑块
  - 具有中心按钮的 3 按键滚轮
  - 接近 / 屏蔽传感器
  - 触觉振动反馈（通过 DRV2605L + LRA）

10.4.6 CAPTIVATE-PROXIMITY（接近演示 PCB）

图 151 示出了 CAPTIVATE-PROXIMITY 接近演示 PCB。

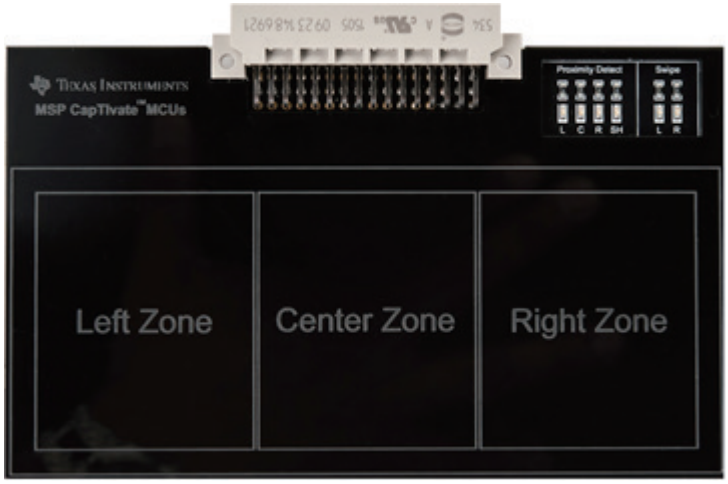


图 151：CAPTIVATE–PROXIMITY 接近演示 PCB

- 演示接近感测能力
- 演示使用一个并行通道作为屏蔽电极
- 运用 3 个专用 IO
- 演示使用复用 IO 作为数字输出
- 特性：
  - 3 个接近区传感器
    - 左
    - 中心
    - 右
  - 1 个屏蔽按键
    - 使接近电极免遭不希望的交互作用（下方或两侧）
    - 另外还报告某种接近状态

10.4.7 48 针凸形传感器 PCB 连接器信息

所有的演示传感器 PCB 均采用相同的 48 针凸形连接器，此类连接器可从不同的制造商和供应商那里采购。如果设计一个连接至 CAPT-FR2633 PCB 的 PCB，那么下面的机械结构信息是有用的。表 59 列出了制造商器件型号。

表 59：连接器信息

制造商	器件型号	Digi-Key 零件编号
FCI	86093487313H55ELF	609-4951-ND
Harting	09 23 148 6921	1195-1856-ND
Harting	09 23 148 2921	1195-1854-ND

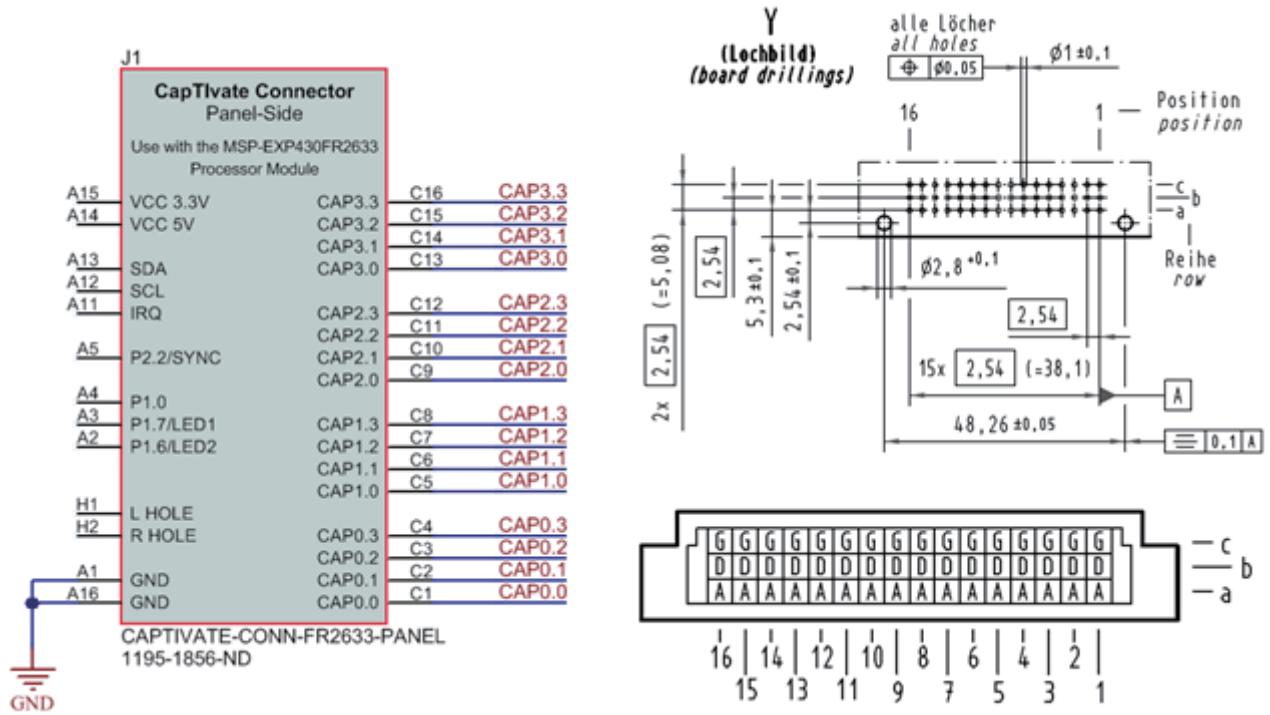


图 153：48 针凸形传感器连接器示意图

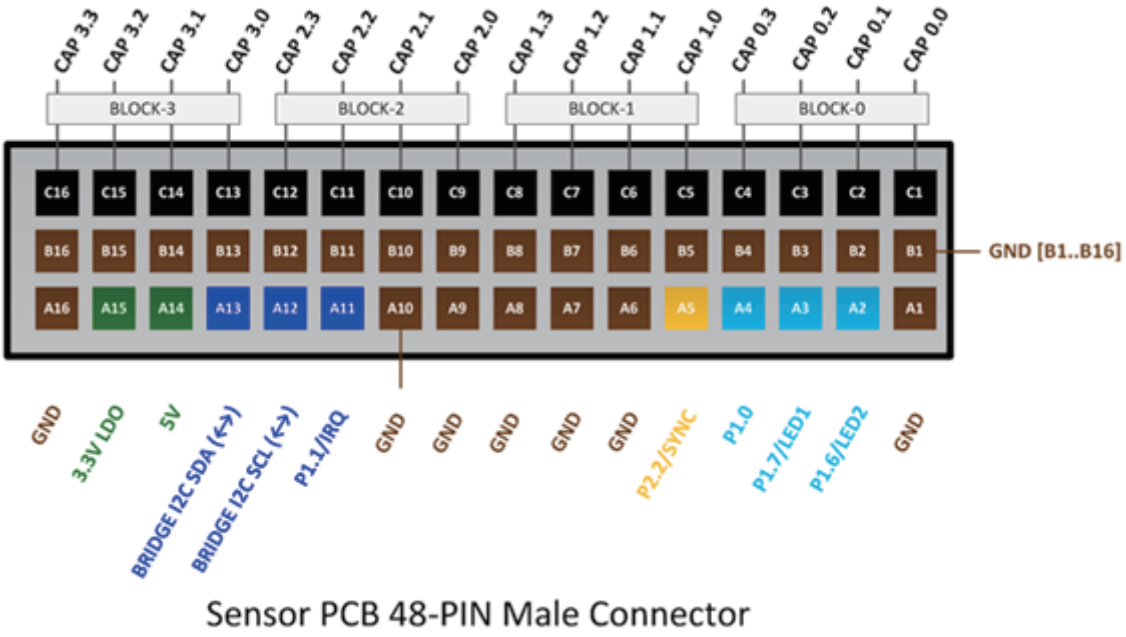


图 154：48 针凸形传感器连接器针脚分配

10.5 利用 MCU 开发套件启动开发工作

10.5.1 在启动开发工作之前

在 PC、Linux 或 MAC 电脑上安装 CapTIvate Design Center GUI（注意要求）。在安装过程中，CapTIvate Design Center 项目示例和针对每个演示 PCB 的 MSP430FR2633 固件项目被置于对应子目录中 CapTIvateDesignCenter\CapTIvateDesignCenterWorkspace 下的用户主目录中。

10.5.2 硬件设置

完成下列工作：

- 把示例传感器演示 PCB 连接至 CAPTIVATE-FR2633 MCU 和 CAPTIVATEPGMR PCB
- 连接介于 CAPTIVATE-PGMR 编程器 PCB 和计算机之间的微型 USB 电缆
- 验证绿色 LED（电源良好）是导通的，而绿色 LED（USB 枚举）是闪烁的。

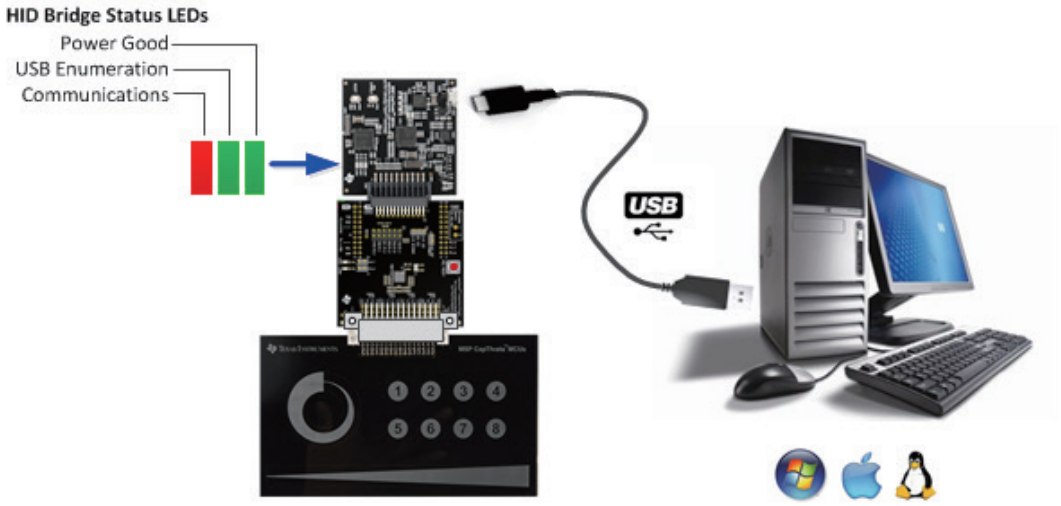


图 155：典型设置

10.5.3 有关接近唤醒运行方式的重要提示

为了弄清接近唤醒模式是如何影响“开箱即用”体验的，下面用实例说明了其运行方式。

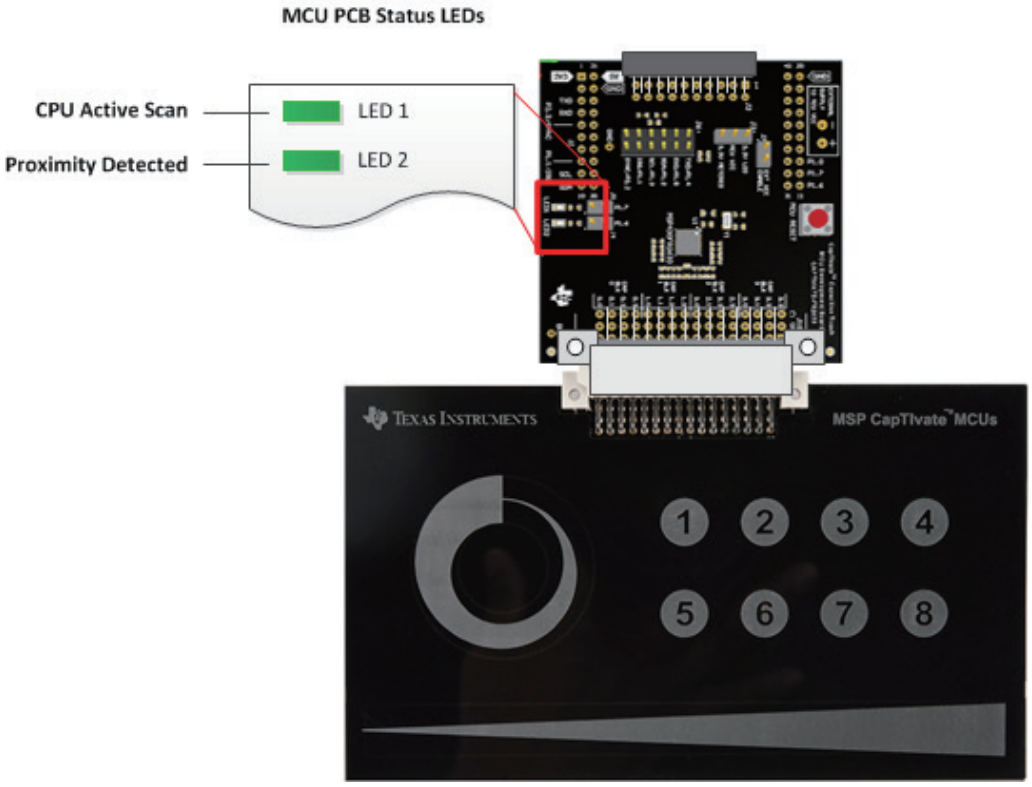


图 156：开箱即用状态 LED

10.5.3.1 接近唤醒低功耗模式

在接近唤醒模式期间 MCU 板有可能看似不在工作之中。与 Design Center 没有通信，而且 MCU 板上的两个 LED 都是关闭的。在该状态中，CPU 处于低功耗模式 3，硬件状态机以 10 Hz 频率主动地仅仅扫描接近传感器，直至检测到一个接近事件为止。

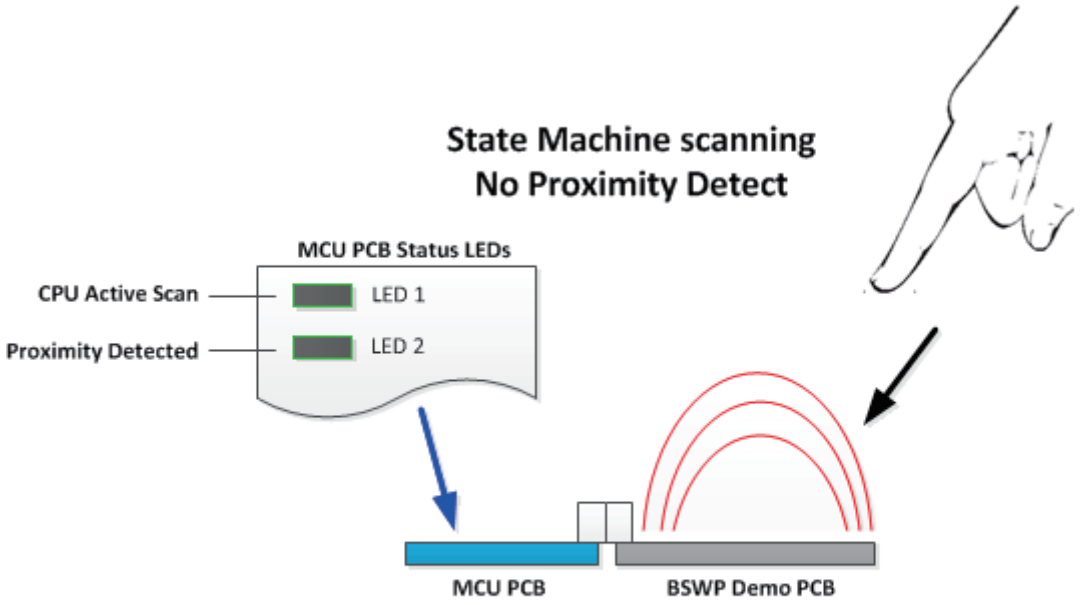


图 154：48 针凸形传感器连接器针脚分配

10.5.3.2 主动模式中的 CPU 扫描

把手放到 MCU 板的附近将引起一种接近条件，从而导致 CPU 退出低功耗模式并恢复以 30 Hz 频率主动地扫描所有的电容传感器（只要您的手靠近或触摸面板）。LED1 将以 CPU 扫描传感器的速率闪烁。当接近检测为“真”时 LED2 导通。

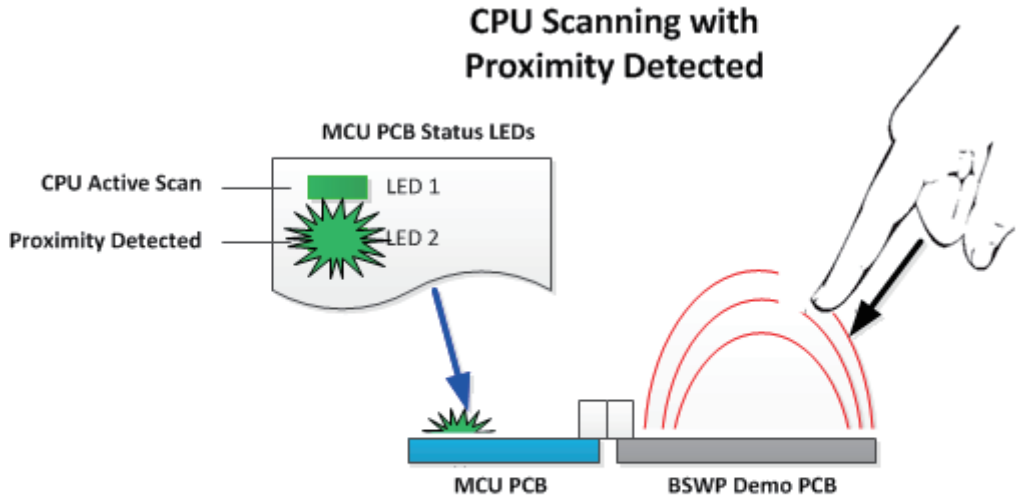


图 158：主动模式

10.5.3.3 进入接近唤醒低功耗模式

在把手从 MCU 板移走之后，LED2 将关闭，不过 LED1 将保持导通，表示面板仍在由 CPU 进行短时间的扫描。在一秒之后 CPU 进入低功耗模式 3，LED1 关闭，且硬件状态机现在主动地仅扫描接近传感器。

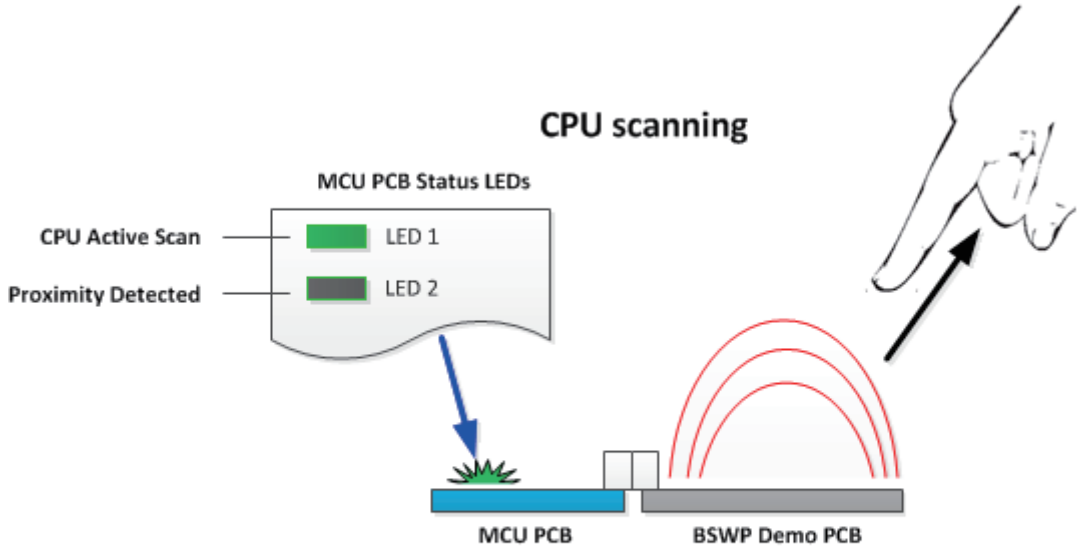


图 159：进入睡眠模式

10.5.4 CAPTIVATE-BSWP 演示（开箱即用体验）

该演示采用 BSWP（按钮、滑块、滚轮和接近）面板来考察 CapTIvate Design Center 的主要特性。在这里点击 11.7.1 节以开始该演示。

10.5.5 CAPTIVATE-PHONE 演示

本例演示了 CAPTIVATE-PHONE PCB 的互电容式按钮、滑块、滚轮和接近特性。

10.5.5.1 启动演示

- 遵循上面的“演示硬件设置”说明
- 采用安装在桌面上的图标来启动 CapTIvate Design Center。选择 Project → Open 并选择 CAPTIVATE-PHONE 项目目录。
- 该项目的主窗口是设计工作区，并预先植入了微控制器和与用于本演示的硬件相匹配的选定传感器。
- 启用 CapTIvate Design Center 中的 Communications（请见 7.4.2 节），以开始演示。

10.5.6 CAPTIVATE-PROXIMITY 演示

该示例演示了 CAPTIVATE-PROXIMITY PCB 的自电容和接近特性。

10.5.6.1 启动演示

- 遵循上面的“演示硬件设置”说明
- 采用安装在桌面上的图标来启动 CapTIvate Design Center。选择 Project → Open 并选择 CAPTIVATE-PROXIMITY 项目目录。
- 该项目的主窗口是设计工作区，并预先植入了微控制器和与用于本演示的硬件相匹配的选定传感器。
- 启用 CapTIvate Design Center 中的 Communications（请见 7.4.2 节），以开始演示。

注： MCU 处于接近唤醒模式，而且也许看似没有活动。把手靠近或触摸电路板以启动 MCU。

10.5.7 目标 MCU 的重新编程

为了演示在 MCU 开发套件中提供的每个传感器 PCB，必须利用对应的固件对 CAPTIVATE-FR2633 MCU PCB 进行重新编程。用于每个演示的源代码在根目录中 DesignCenterWorkspace 下的其自有目录中提供。按照这些步骤执行以加载和运行生成的固件项目。

10.6 使用 HID 桥接器

HID 桥接器是一款多用途开发工具，其提供了一种把平台软件（比如：PC GUI）与包含基本串行接口（如 UART 或 I^2C）的嵌入式系统相连接的方法。HID 桥接器是一种专为在支持 USB 的 MSP430F5xx MCU 上运行而设计的固件产品。典型的实施方案（例如：CAPTIVATE-PGMR）使用了一个 MSP430F5528 MCU。

HID 桥接器实现了从一个串行接口经由 USB HID 设备类别至一个主机平台的双向数据传输。相比于其他的同类竞争接口，HID 提供了一项独特的优势。与 USB CDC 设备不同，在主机上不需要进行 COM 端口识别，而且在大多数新版操作系统上无需安装驱动程序。此外，HID 设备对于热插拔（断接和重新连接）的适应性也好得多。在嵌入式系统开发期间，开发工具的断接和重新连接是极为常见的；采用 HID 可使该过程无缝连接。

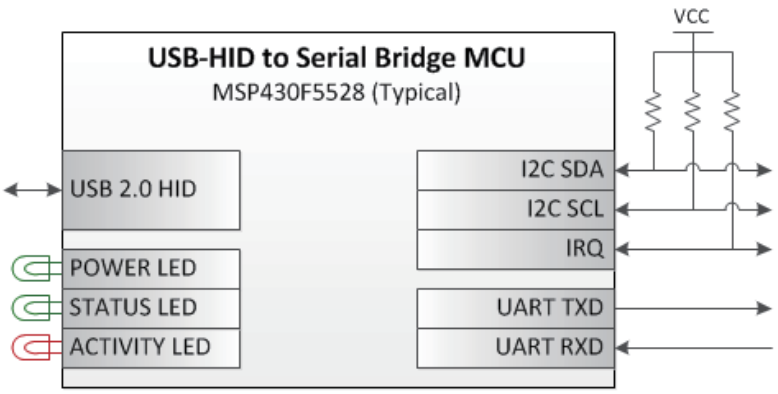


图 160：HID 桥接器示意图

10.6.1 支持的操作模式

HID 桥接器提供了下面的操作模式：

- HID 至非格式化原始 UART（高达 250k 波特的双向全双工）
- HID 至格式化 UART（高达 250k 波特的双向全双工，具有智能数据包识别和成帧功能）
- HID 至非格式化原始 I^2C 主控器（高达 400kbps I^2C 主控器）
- HID 至格式化 I^2C 主控器（高达 400kbps I^2C 主控器，具有智能数据包识别、成帧和从属 IRQ 处理能力）

10.6.2 HID 桥接器接口

10.6.2.1 主机接口：USB HID 设备类别实施方案

该桥接器作为一种复合类人机接口设备 (HID) 在 USB 主机上枚举。两个 HID 设备枚举：

- HID0，数据传输接口
  - 用于目标和主机之间的数据传输
  - 对数据进行了二进制格式化
- HID1，HID 桥接器配置控制台接口
  - 用于通过 HID 桥接器 (HB) 命令集实现的 HID 桥接器本身的配置
  - 对命令进行了 ASC II 字符串格式化

10.6.2.2 目标接口：UART 实施方案

HID 桥接器目标 UART 接口是支持若干常用波特率的双向全双工两线式 UART 接口。

支持的波特率为：250 kBaud、115.2 kBaud、38.4 kBaud、19.2 kBaud 和 9.6 kBaud。当配置在 UART 模式中时，HID 桥接器持续不断地聆听目标 UART 端口并可以随时接收数据，即使在向目标传输数据的时候也不例外。UART 波特率可通过 HB 命令集进行配置。

另外，UART 实施方案还支持一种针对慢速目标的字节延迟特性。该特性在每个字节的传输起始点之前插入一个延迟（单位为毫秒）。这能够允许使用一个快速 (250 kBaud) 速率，但是字节被分隔开了，以使目标能拥有响应其 UART 中断服务例程的时间。当目标在发送数据时，它仍然能以 250 kBaud 的全速率发送，但是它不必以该速率响应一个数据流，因为字节是隔开的。UART 延迟可通过 HB 命令集进行配置。

10.6.2.3 目标接口：I²C 实施方案

当配置在 I^2C 模式中时，HID 桥接器充当一个与 I^2C 从属设备进行通信的 I^2C 总线主控器。

HID 桥接器通过发送一个至选定从属地址的起动 / 写入条件、写入数据并发布一个 I^2C 停止条件来启动对目标的写操作。

从属设备可通过把 I^2C IRQ 线路拉至低电平（直到 HID 桥接器从目标读出数据为止）来请求执行读取操作。当 HID 桥接器向受控器发布一个起动 / 读取条件时，受控器必须释放 IRQ 线路。由主控器读取的第一个字节预计是一个向 HID 桥接器指示其应进一步从受控器读取多少个字节的长度字段。主控器随后将继续执行读取操作，读出那么多数量的字节，然后发布一个停止条件。

10.6.3 HID 桥接器配置命令集

HID 桥接器是由主机平台通过经由配置控制台 HID 接口发送和接收 HID 桥接器 (HB) 命令来配置的。表 60 说明了可用的配置命令。

表 60：HID 桥接器配置命令集

命令	说明	有变元吗?	有效变元 (1)	有效变元 (2)
HB VERSION	获得版本字符串	无	—	—
HB PLATFORM	获得平台字符串	无	—	—
HB UPGRADE	进入 USB 引导装载程序	无	—	—
HB REBOOT	重启 HID 桥接器	无	—	—
HB MODE	获得或设定操作模式	格式，接口	原始，数据包	UART，I^2C
HB SAVECONFIG	把现行配置或默认配置保存到引导存储器	配置以保存	现行，默认	—
HB UARTBAUD	获得或设定 UART 波特率	新的波特率	250000，115200，38400，19200，9600	—
HB UARTDELAY	获得或设定 UART 字节延迟周期	新的延迟周期	0 至 1000	—
HB I2CADDRESS	获得或设定目标 I^2C 从属地址	新的地址	0 至 127	—
HB I2CCLOCK	获得或设定目标 I^2C 时钟频率	新的频率	100000，400000	—

对于那些没有变量的命令，简单地发送“HB x”（其中的 x 是命令）。对于有变量的命令，发送“HB x”（其中的 x 是命令）将回传该命令的当前设置（如果支持的话）。如果传递的是有效的参数（例如：“HB UARTBAUD 250000”），则参数被更新至传递的新数值。

CapTIvate Design Center 根据打开的当前 Design Center 项目在该接口上自动地发送配置命令。

10.6.4 HID 桥接器操作模式

如在 10.6.1 节中介绍的那样，HID 桥接器支持几种不同的操作模式。本节说明的是每种模式具体运作方式的详情。操作模式可以通过 HID 桥接器命令集来设定。

10.6.4.1 操作模式：数据包模式

数据包模式 (packet mode) 要求根据下面的传输规则组把通过目标接口发送的数据以数据包的形式进行格式化。HID 桥接器将缓存从目标接收的所有数据。它随后将对即将发送至主机的数据包进行识别和成帧。当向主机发送数据包时，每个数据包在其自己的 HID 报告中传输。HID 桥接器缓存来自主机的所有数据包，并在把数据包发送到目标之前应用相关的传输规则。

如果任何事务处理属于下面的类型，则它们遵从传输规则：

- 数据包以 PACKET/UART 模式从目标传输至 HID 桥接器
- 数据包以 PACKET/UART 模式从 HID 桥接器传输至目标
- 数据包以 PACKET/I^{2}C 模式从 HID 桥接器传输至目标

用于数据包模式中的串行接口的传输规则如下：

- 所有的数据包以一个 3 字节标头为起点，其包括：
  - [0] 一个 SYNC 字节位于位置 0，等于 55h
  - [1] 一个 BLANK 字节位于位置 1，不等于 55h。通常采用 AAh。
  - [2] 一个 LENGTH 字节位于位置 2，其以字节为单位指示有效负载和校验和的长度（有效负载长度 + 2）。
- 所有的数据包可能包含一个长达 60 字节的有效负载，其可以包含任何二进制数据
  - 如果 SYNC 字节 (55h) 应出现在有效负载部分，则它必须发送两次（字节填充）以将其与一个真实的 SYNC 字节区别开来。这个重复的字节并不计算在 60 字节的最大有效负载长度中。
- 所有的数据包将以一个 2 字节校验和为结束。
  - 校验和作为有效负载部分之总和的较低 16 位减去任何填充的字节（重复的 55h 字节）来计算。
  - 校验和首先发送低位字节，其次发送高位字节。

依据这些规则，一个数据包将具有以下的格式：

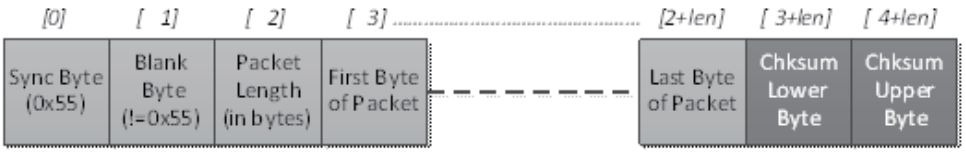


图 161：应用了传输规则的串行接口数据包

所有其他的数据包类型（其余的字节罗列于下）只需要把一个校验和添加至有效负载。

- 数据包以 PACKET/I²C 模式从目标传输至 HID 桥接器
- 数据包以任何模式从 HID 桥接器传输至主机

当把有效负载和校验和发送到主机时，HID 桥接器去除了传输规则开销（如果应用的话）。SYNC、BLANK 和 LENGTH 字节被去除，而且有效负载部分的任何填充字节也被去掉。校验和被保留并转送。

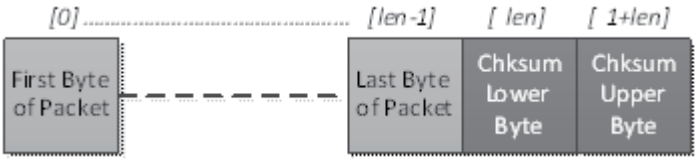


图 162：未应用传输规则的基础数据包

在所有的场合中，当 HID 桥接器处于 PACKET 模式时，校验和字段必须有效，否则数据将不被转送。例如，倘若一个数据包从目标发送至 HID 桥接器，而且校验和是无效的，则该数据包不被发送到主机。同样，如果主机向 HID 桥接器发送一个数据包，而且校验和无效，则该数据包不被发送至目标。

10.6.4.2 操作模式：原始模式

当在原始 (RAW) 模式中操作时，HID 桥接器基本上起一个通信缓冲器的作用。无需进行格式化。从目标接收的任何数据均被转送至主机，而从主机接收的任何数据则被转送到目标。在主机侧没有 HID 帧定位。

11 专题练习

11.1 MSP CapTIvate 微控制器简介

该专题练习逐步讲解如何运用 CapTIvate 生态系统和 CapTIvate 电容式触摸 MCU 开发套件来启动开发工作。内容包括工具设置、开箱即用体验和三个实验。实验是相辅相成的，旨在教授电容式触摸调试技巧、推荐的调试顺序、低功耗优化和利用 CapTIvate 电容式触摸库的软件开发。

11.2 专题练习概要

该专题练习通过动手实验介绍了 CapTIvate 生态系统。此专题练习被分为四个部分：

- 开箱即用体验
  - 探究 CapTIvate 电容式触摸 MCU 开发套件的组件
  - 对 MSP430FR2633 CapTIvate MCU 的特性和性能进行实验（无需编写任何代码）
- 实验 1 (Lab 1)：使用 CapTIvate Design Center 以创建和调优传感器
  - 学习如何使用 CapTIvate Design Center
  - 学习 CapTIvate Design Center 和 CCS 之间的交互
  - 学习基本的电容式触摸调优原理
- 实验 2 (Lab 2)：体验低功耗设计方法
  - 学习如何使用“接近唤醒”特性
  - 学习如何调优以实现尽可能低的功耗
  - 运用 EnergyTrace 测量调优对于功耗的影响
- 实验 3 (Lab 3)：使用 CapTIvate 软件库
  - 了解该软件库中的传感器管理特性
  - 学习怎样运用回调函数以获得接近、触摸和位置数据

当该专题练习结束之时，用户将通晓如何使用 CapTIvate Design Center、CapTIvate 触摸软件库和 CapTIvate 电容式触摸 MCU 开发套件的基础知识。

11.3 假设

该专题练习做了下面的假设：

- 用户熟悉 MSP430 微控制器架构
- 用户熟悉 Code Composer Studio v6 及其特性，并已经安装了 Code Composer Studio 并获得了使用许可
- 用户熟悉基本的电容式触摸原理

11.4 所需的工具

完成该专题练习的学习需要以下工具：

- CapTIvate Design Center PC GUI 工具
  - CapTIvate Design Center 是一款用于配置和调优 CapTIvate 触摸的 PC GUI 工具。可从如下网址下载安装程序：
  - 需要 Java 运行引擎 (JRE) version 1.7 或之后发布的版本。
- 支持 MSP430FR2633 的 TI Code Composer Studio v6.10 或更高版本。
  - Code Composer Studio 可从如下的 Wiki 网页下载：
    http://processors.wiki.ti.com/index.php/Download_CCS

11.4.1 CAPTIVATE-FR2633 核心板 和 CAPTIVATE-PGMR 编程 / 调试板

CAPTIVATE-FR2633 和 CAPTIVATE-PGMR 编程 / 调试板分别提供了 MSP430FR2633 目标 MCU 和编程 / 调试功能。

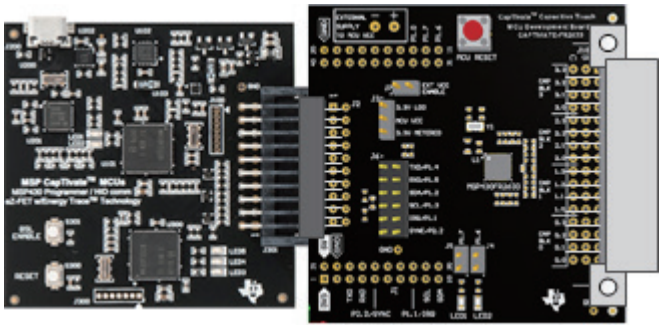


图 163：编程器和 MCU PCB

11.4.2 CAPTIVATE-BSWP（开箱即用体验演示面板）：

CAPTIVATE-BSWP 是一个插入 EVM 的电容式触摸评估面板。其在此次专题练习中用于演示电容式触摸接口。该面板演示的是自电容。

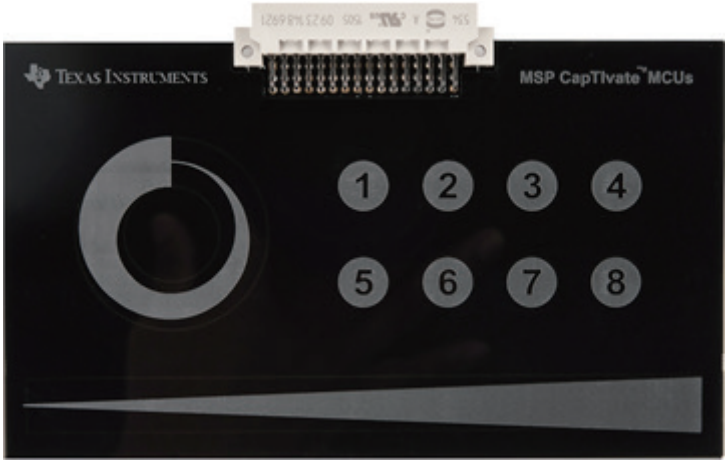


图 164：BSWP 演示面板

11.5 硬件设置

请完成以下工作：

- 把示例传感器演示面板连接至 CAPTIVATE-FR2633 MCU 和 CAPTIVATE-PGMR PCB
- 通过 USB 电缆连接 CAPTIVATE-PGMR 编程器 • 验证绿色 LED（指示电源良好）是导通的，而且绿色 LED（指示 USB 枚举）在闪烁。

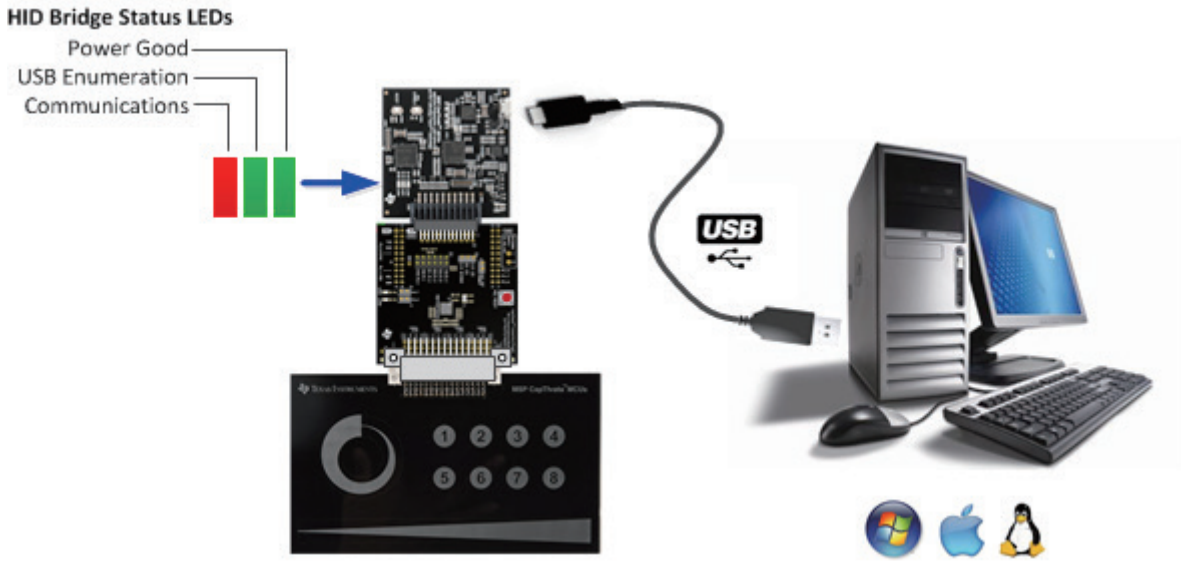


图 165：典型设置

11.6 有关“接近唤醒”运行方式的重要提示

为了解“接近唤醒”模式是如何影响“开箱即用”体验的，在图 166 中示出了运行方式。

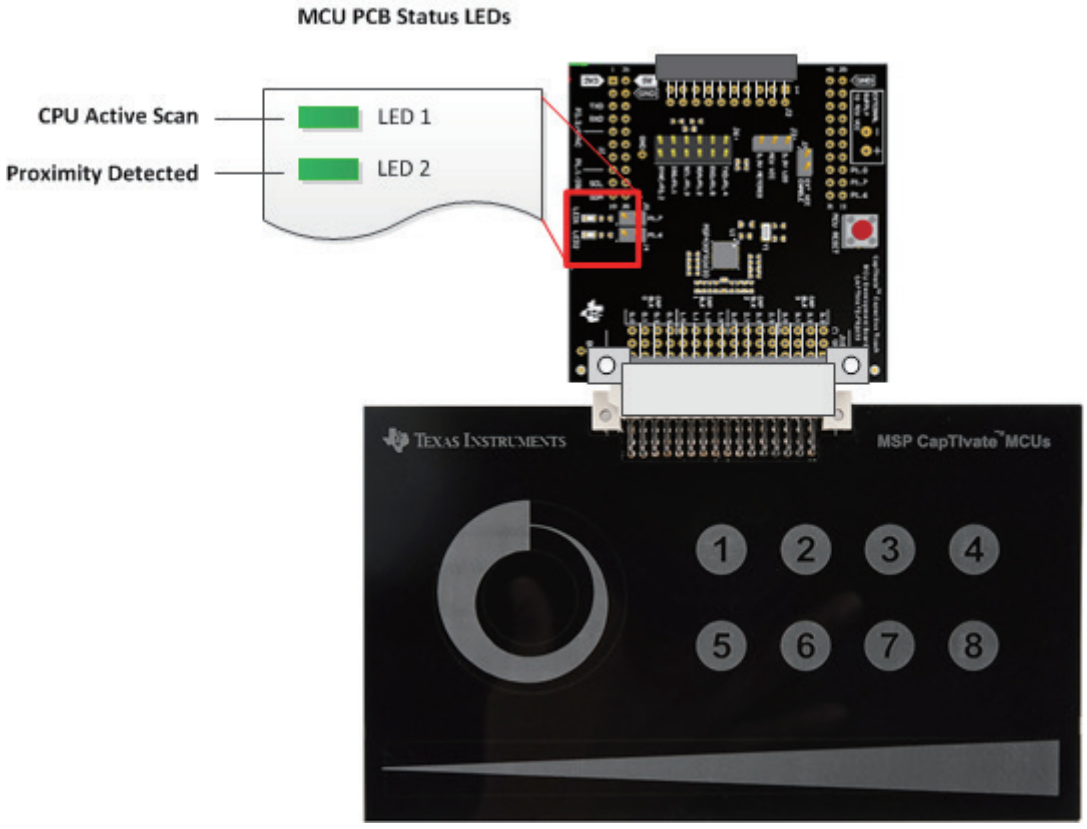


图 166：开箱即用状态 LED

11.6.1 接近唤醒低功耗模式

在接近唤醒模式中 MCU 板也许看似处于非工作状态。没有与 Design Center 的通信，而且 MCU 板上的两个 LED 均关闭。在该状态中，CPU 处于 LPM3，硬件状态机以 10 Hz 频率仅主动地扫描接近传感器，直至检测到一个接近事件为止。

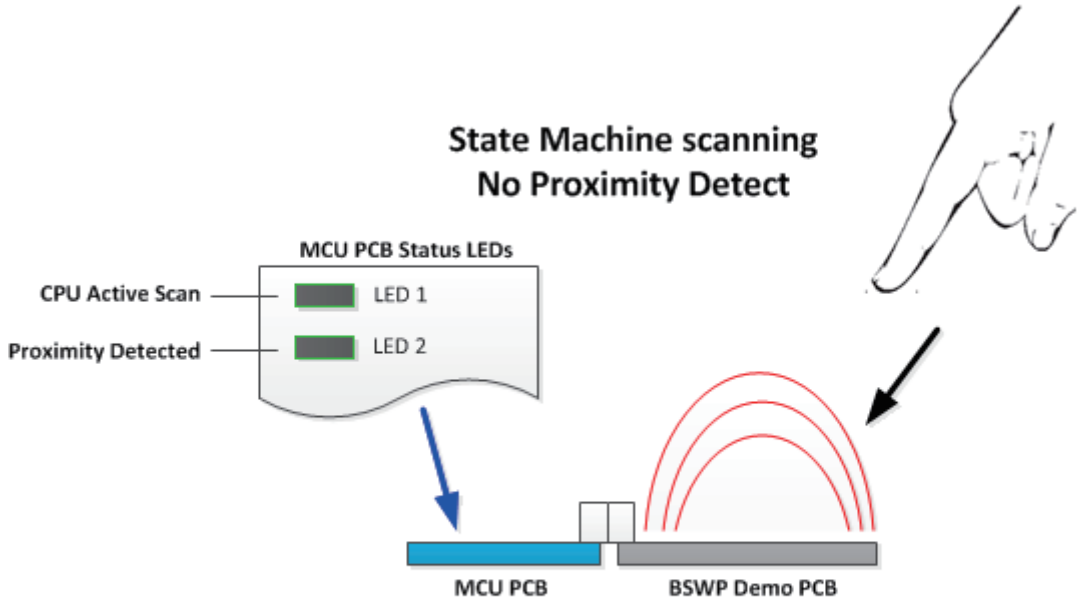


图 167：接近唤醒模式

11.6.2 主动模式中的 CPU 扫描

把手靠近 MCU 板将引发一个接近事件，从而导致 CPU 退出低功耗模式并恢复以 30 Hz 频率主动地扫描所有的面板传感器（只要手靠近或者接触面板）。LED1 以 CPU 扫描传感器的速率闪烁。LED2 在接近检测为“真”时处于导通状态。

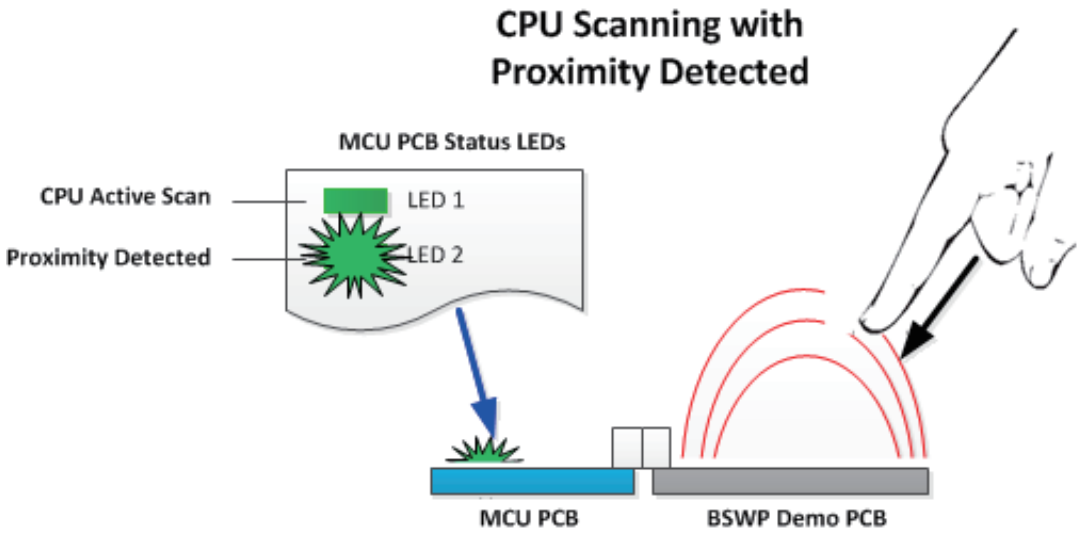


图 168：主动模式

11.6.3 进入接近唤醒低功耗模式

在把手从 MCU 板移开之后，LED2 关闭，而 LED 1 处于导通状态，表示面板仍然由 CPU 进行短时间的扫描。在一秒钟之后，CPU 进入 LPM 3，LED 1 关闭，且硬件状态机仅主动地扫描接近传感器。

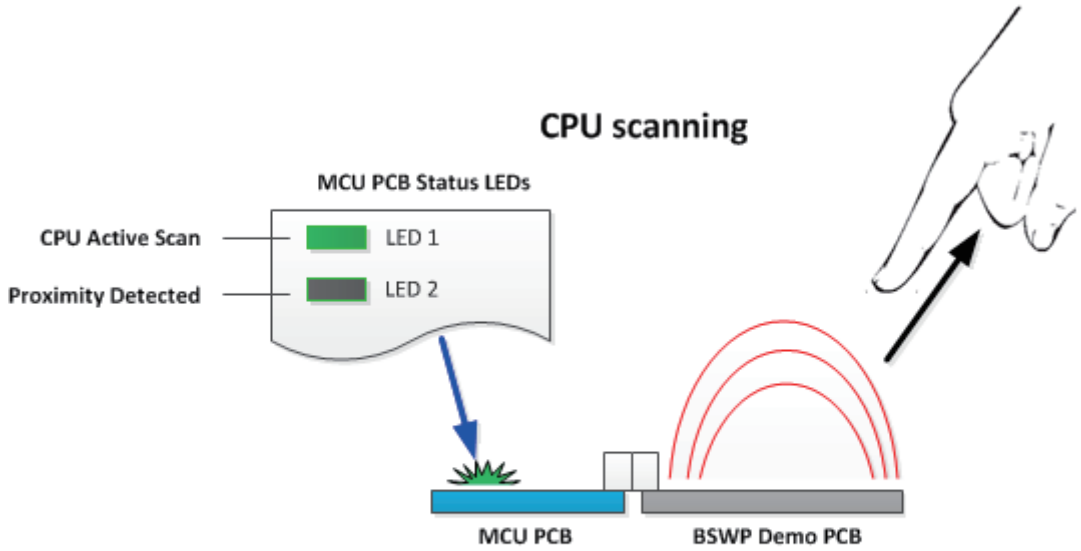


图 169：进入睡眠模式

11.7 CAPTIVATE-BSWP 演示

11.7.1 开箱即用体验

该演示使用自电容式 BWSP（按键、滑块、滚轮和接近传感器）面板以演示 CapTIvate Design Center 的主要特性。

- 设计和传感器调优视图
- 对自电容式按键、滑块、滚轮和接近传感器进行实验

在结束了这种开箱即用体验之后，继续参加专题练习以学习更多的东西。

- 如何针对期望的触摸和感觉来创建和调优传感器
- 低功耗传感器调优考虑因素
- 使用 CapTIvate 触摸库

注： CAPTIVATE-FR2633 MCU PCB 在出厂时预先写入了 CAPTIVATE-BSWP 开箱即用体验演示固件。但是，如果 CAPTIVATE-FR2633 MCU PCB 需要重新编程，则在继续工作之前对目标 MCU 进行重新编程。

11.7.2 启动演示

- 遵循上面的“演示硬件设置”说明。
- 双击安装在桌面上的图标来启动 CapTIvate Design Center。选择 Project → Open 并选择 CAPTIVATE-BSWP 项目目录。
- 该项目的主窗口是设计工作区，并预先导入了微控制器和与用于本演示的硬件相匹配的选定传感器。
- 使能 CapTIvate Design Center 中的 Communications 以开始演示。

注： MCU 处于接近唤醒模式，也许看似没有活动。把手靠近或触摸电路板以启动 MCU。

Design Center 工作区在触摸屏上显示传感器的当前状态。触摸位于面板上的各种不同的传感器，并在工作区上观察响应。Design Center 为按键组绘制一个圆形按键指示符，并为滑块、滚轮和接近传感器显示了一幅滑块、滚轮和接近勾勒图。

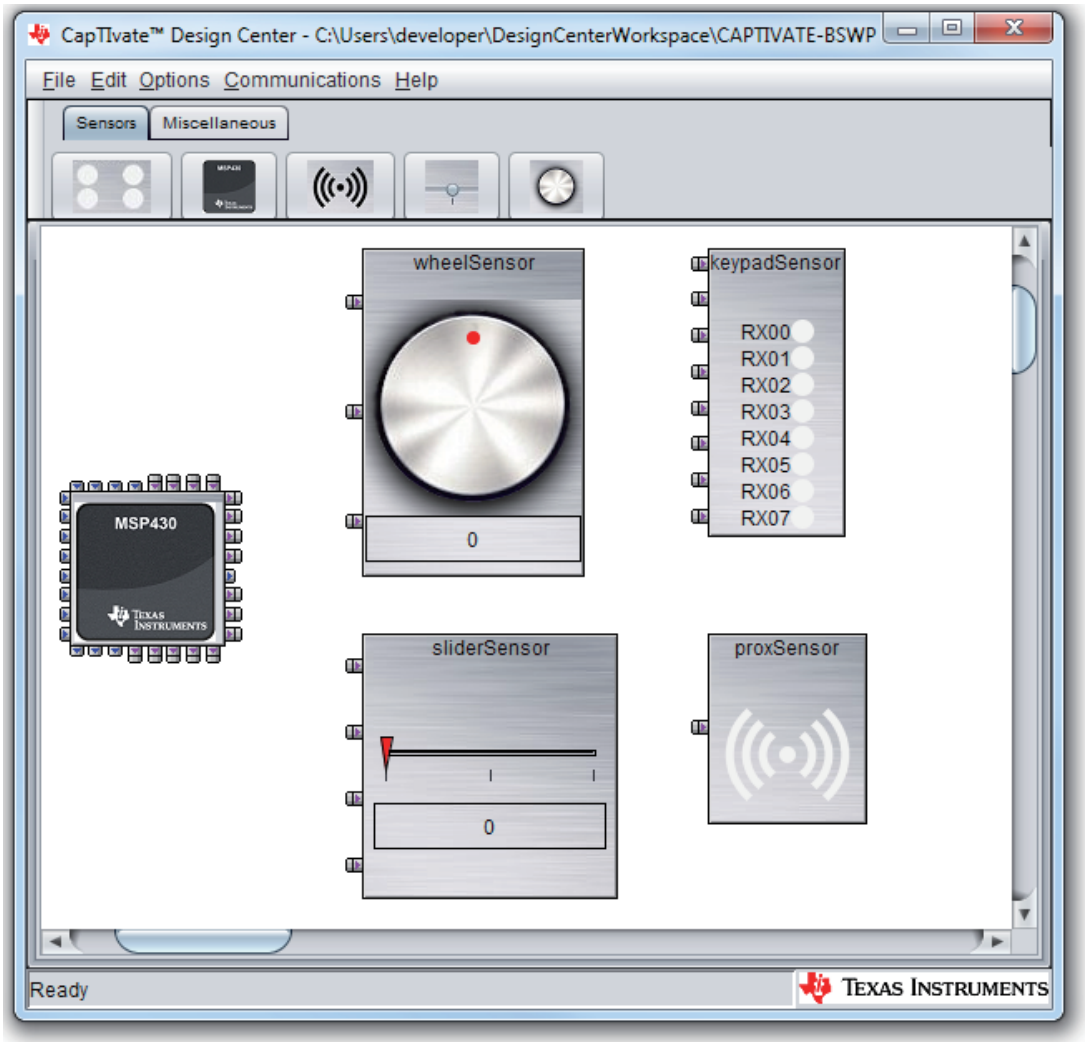


图 169：进入睡眠模式

首先要注意的是 Design Center 中的传感器上有不同颜色的指示灯根据其所代表的状态而闪烁。

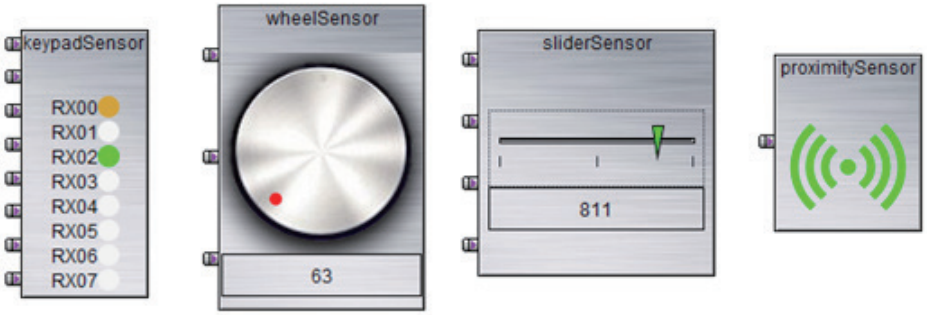


图 171：传感器彩色状态

滑块或滚轮传感器上的红色表示滑块或滚轮不处在被触摸的状态，而指针或圆圈则显示的是最后一个有效位置。滑块或滚轮传感器上的绿色表示滑块或滚轮正处在被触摸的状态，而指针或圆圈指示器则显示位置。按键组指示器或接近传感器上的白色意味着在各自的元件上未进行触摸或接近检测。绿色表示元件正处于被触摸的状态。琥珀色指示元件正处在接近检测中。绿色将始终取代琥珀色，因为当某个元件处于触摸检测状态时，其通常也处在接近检测状态。如果想同时查看接近检测 LED 和触摸检测 LED，以及原始数据，则打开控制器定制器。

双击工作区视图中的 MSP430 窗口小部件。控制器定制器窗口打开。

图 172：MSP430 窗口小部件

控制器定制器窗口可同时提供 MCU 配置控制以及来自整个面板的元件数据的实时查看。这些不同的功能位于该定制器的不同选项卡上。用于控制器定制器的默认选项卡是“配置连接”(Configure Connections) 选项卡。该选项卡规定了传感器至端口映射。

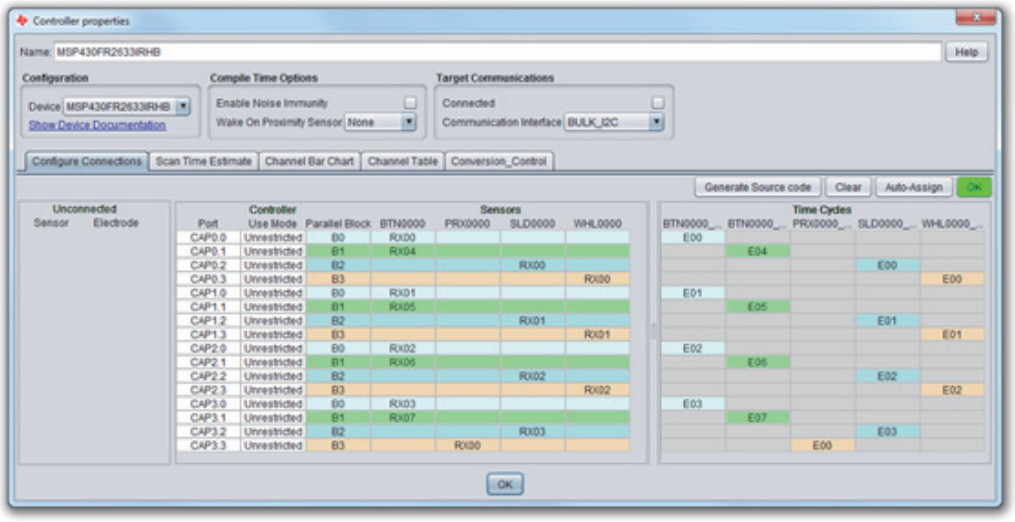


图 173：控制器定制器视图

打开“通道条形图”(Channel Bar Chart) 选项卡。通道条形图选项卡显示了系统中每个元件的通道数据。接近和触摸指示器（分别为琥珀色和绿色）显示在每个通道的条形图的下方。通道的即时计数值在用于每个通道的曲线图中被绘制为蓝色条形图。绿色条形线指示触摸门限，而黄色条形线指示的是接近门限。LTA 被绘制为黑色条形线，而且常常隐藏在计数条形图的背后出现，除非元件处于被触摸的状态。

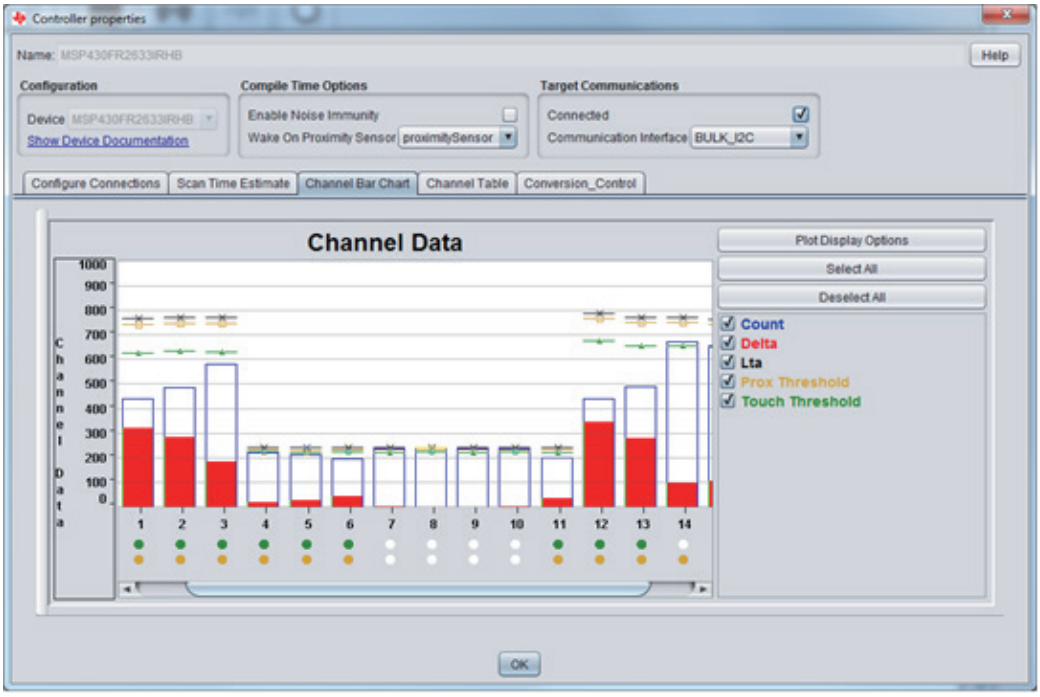


图 174：控制器定制器视图

本节介绍了开箱即用演示。在 8.8 节中继续该专题练习的学习以了解怎样启动一款传感器设计。

11.8 创建一个新的电容感测设计项目

11.8.1 LAB1

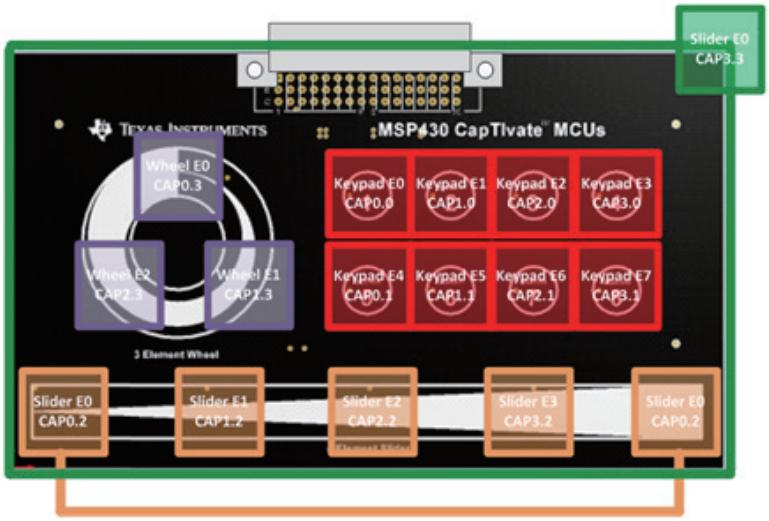
本节的内容超出了开箱即用体验的范围，旨在说明以下操作：

- 如何运用 CapTIvate Design Center 创建新的传感器项目
- 如何生成 CCS 和 IAR 目标代码
- 如何调优传感器

11.8.2 第一部分 – 创建一个新的设计

配置传感器是一个重要的步骤。为了利用目标 MCU 的并行扫描能力，应用必须在可用的 CapTIvate 端口引脚和每个传感器中的电极之间建立最优的连接。由于具备并行扫描电极的能力，因此会对扫描时间和功耗皆产生影响。本节采用 CapTIvate Design Center 的自动分配特性以生成一种适用于 CAPTIVATE-BSWP 面板的最优引脚配置。

BSWP（按键、滑块、滚轮和接近）面板具有几种不同的自电容传感器：一个按键组、一个滚轮、一个滑块和一个接近传感器。图 175 示出了引脚配置和传感器布局。



Sensor	Element	RX	Port
keypad	E00	RX00	CAP0.0
keypad	E01	RX01	CAP1.0
keypad	E02	RX02	CAP2.0
keypad	E03	RX03	CAP3.0
keypad	E04	RX04	CAP0.1
keypad	E05	RX05	CAP1.1
keypad	E06	RX06	CAP2.1
keypad	E07	RX07	CAP3.1
prox	E00	RX00	CAP3.3
slider	E00	RX00	CAP0.2
slider	E01	RX01	CAP1.2
slider	E02	RX02	CAP2.2
slider	E03	RX03	CAP3.2
wheel	E00	RX00	CAP0.3
wheel	E01	RX01	CAP1.3
wheel	E02	RX02	CAP2.3

图 175：演示面板电极连接

11.8.2.1 创建一个新的 CapTIvate Design Center 项目

从 Design Center 工具栏选择“File, Project New”，并给项目名称提供 LAB1。应出现一个空白的工作空间版面。如果 Design Center 询问您是否保存对于示例 BSWP 项目的任何修改，则选择“放弃”(Discard)，这样示例项目之后就能起一种参考作用。

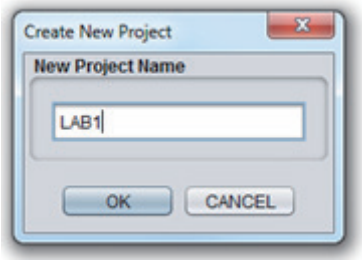


图 176：新项目名称 (New Project Name)

11.2.2.2 实例化 控件控件 — 增设一个 MSP CapTIvate 控制器

在 Design Center 中的菜单工具栏下面是 控件控件 栏。Design Center 包含了被称为“控件控件”的对象，其代表了电容触摸系统中的实际事物；例如：一个 MCU、一个按键组、一个滑块、或者一个滚轮。

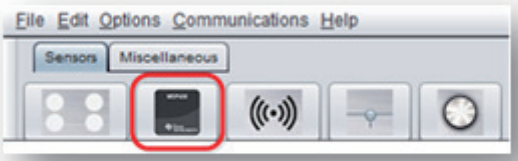


图 177：控件 对象菜单

所有的设计必须具有且仅有一个控制器。如欲给工作空间版面增添一个控制器，则从控件控件栏选择 MSP Controller 控件 图标，并在版面上点击以放置。工作空间版面现在应该显示一个控制器控件控件。

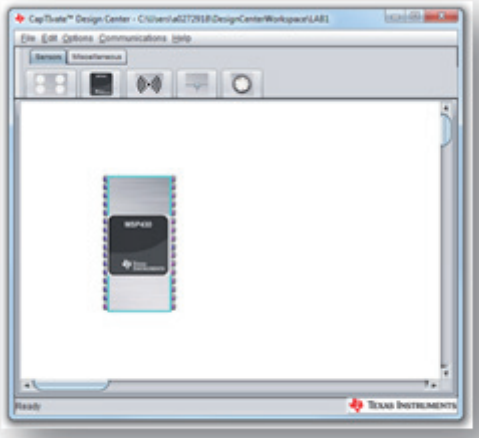


图 178：增添 MCU 控件控件

双击 MSP 控件控件以打开控制器定制器。在控制器属性视图中，从配置器件下拉菜单中选择 MSP430FR2633IRHB（32 引脚 QFN 封装）器件。

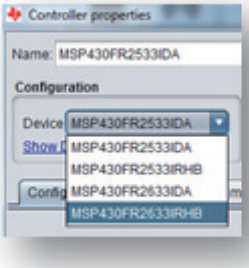


图 179：选择器件

必须对用于和目标 MCU 进行通信的通信接口进行配置。I^2C 提供了低于 UART 接口的功耗。因此，从控制器定制器中的通信接口下拉菜单中选择 BULK_I2C（见图 180）。

图 180：选择通信模式

关闭控制器定制器。控件 发生了变化，表示在该专题练习中使用的器件和封装（见图 181）。

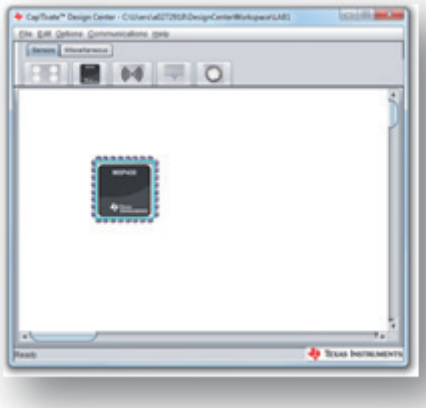


图 181：被配置的 MCU 控件

11.8.2.3 增添传感器

该面板具有一个按键组（具有 8 个按键的小键盘）、一个 4 元件滑块、一个 3 元件滚轮和单元件接近传感器。现在给这些传感器各增加一个 控件。就和控制器一样，用于这些传感器类型中每一种的各自 控件 可通过点击 控件 图标并把该 控件 拖放到版面上进行添加。传感器都添加好之后，版面看起来应与下图相似

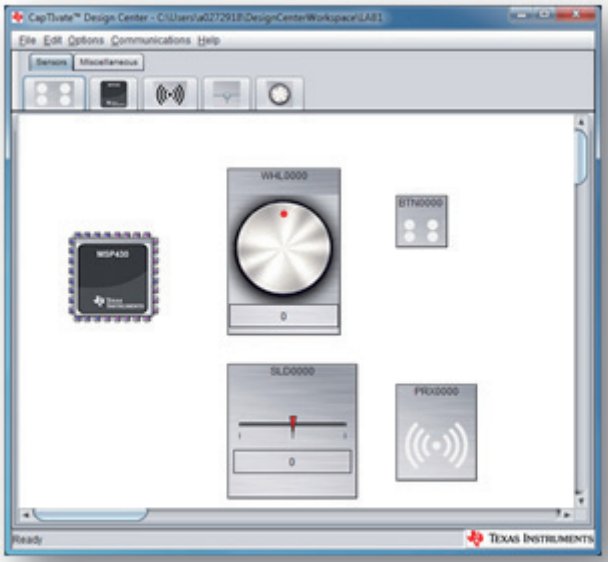


图 182：增添传感器 控件

11.2.2.4 配置传感器

版面中已经包含了系统中的所有传感器和控制器。Design Center 现在需要每个传感器相关的更多信息。从按键组开始。双击添加至版面的按键组以打开其定制器窗口。该定制器在其顶端上具有一个配置 (Configuration) 部分。由于 BSWP 面板仅限于自电容，因此对于 Capacitive Mode 选项指定了 SELF。由 8 个按键构成了小键盘，所以把按键计数设定为 8。Electrode Config 下拉菜单只显示了一个选项：2 个周期、0 个 Tx 引脚和 8 个 Rx 引脚，因为对于 8 按键自电容传感器而言这是唯一的可能性。该定制器应与图 183 相匹配。

图 183：小键盘配置

给按键组一个有意义的名称。在定制器顶部的 Name 文本框中输入名称。由 Design Center 提供的独特默认名称为 BTN00。对于该实验室动手操作把该名称改为 keypad。

对于滑块、滚轮和接近传感器需要执行相同的流程。现在，设置这些传感器的配置以与面板配置相匹配（请查阅本节开始处的图像作为参考）。Design Center 假设一个滚轮有 3 个元件，而一个滑块有 4 个元件，在此开发板上恰好就是这种配置。因此，仅需进行的配置是给传感器命名并把电容式感测模式设定为 SELF。赋予这些传感器新的名称和配置，如图 184 所示。

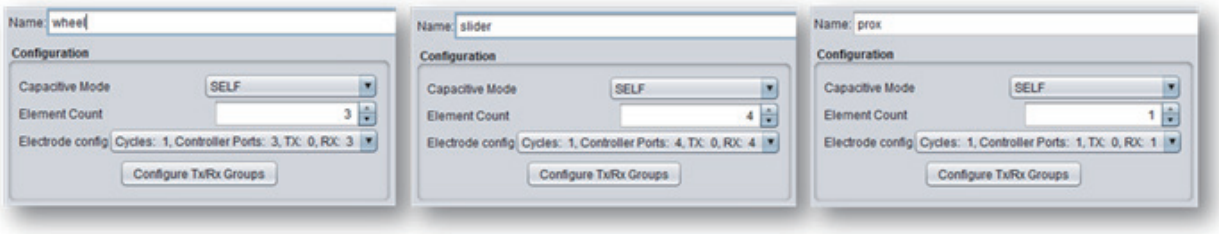


图 184：传感器配置

现在，版面视图在四个传感器的左侧显示小的连接点：8 个用于小键盘、4 个用于滑块、3 个用于滚轮、1 个用于接近传感器。每个连接点表示该传感器所需的一个 CapTIvate 端口。这些连接点映射至 MSP430FR2633 MCU 控制器上的实际端口。

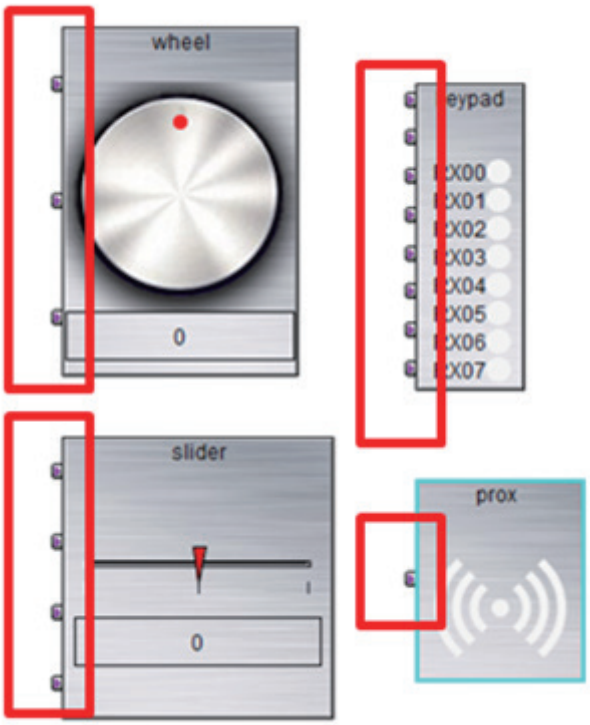


图 185：未分配的传感器电极

11.8.2.5 自动分配传感器连接

Design Center 能够自动地把传感器映射至端口。这是推荐采用的端口映射方法，因为 Design Center 运用一种算法来决定最优的配置。端口分配是在控制器定制器中完成的。打开控制器定制器（这是看似一个器件的 控件）。如图 186 所示，该定制器打开时选择的是 Configure Connections 选项卡。在前一个步骤中添加的传感器罗列在该选项卡最左边的 Unconnected 方框中，表示什么都尚未连接。需注意的是，Generate Source Code 按键是灰色的且不可用。这是因为生成源代码需要一个有效的端口映射。Configure Connections 选项卡上位于中间的方框每个可用 CapTIvate 端口显示为一行。在默认情况下该表的 Use Mode 列将每个引脚设定为 Unrestricted。这意味着自动分配功能可以使用这里列出的所有端口。通过把 Unrestricted 选项改为 Reserved 可以保留一个端口（例如，将其用于一个 SPI 接口）。BSWP 面板需要全部 16 个端口，于是把所有的端口置为 Unrestricted。

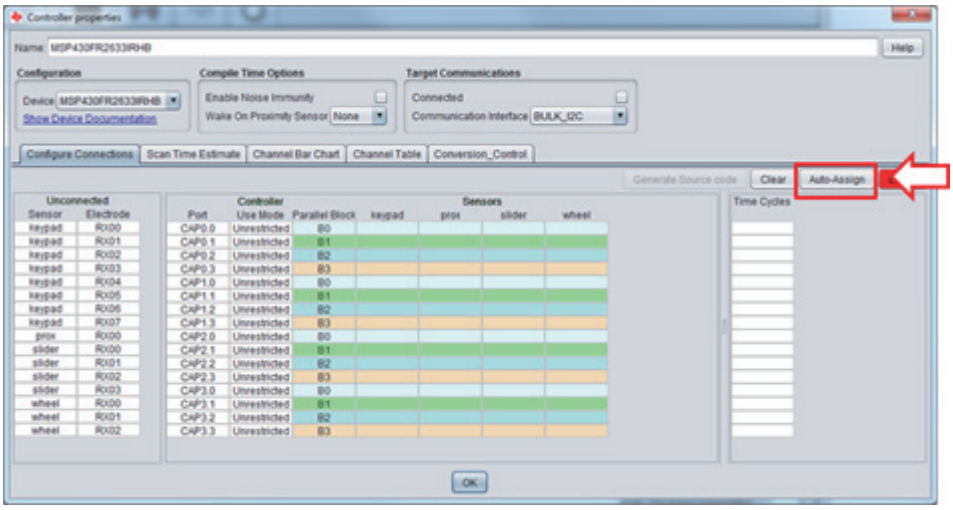


图 186：自动分配功能

当完成时，配置看起来像图 187 那样。指示器为绿色并显示 OK，表示所有的传感器皆已连接至控制器端口引脚，并且没有冲突。所有的 16 个 CapTIvate I/O 引脚均已被使用。

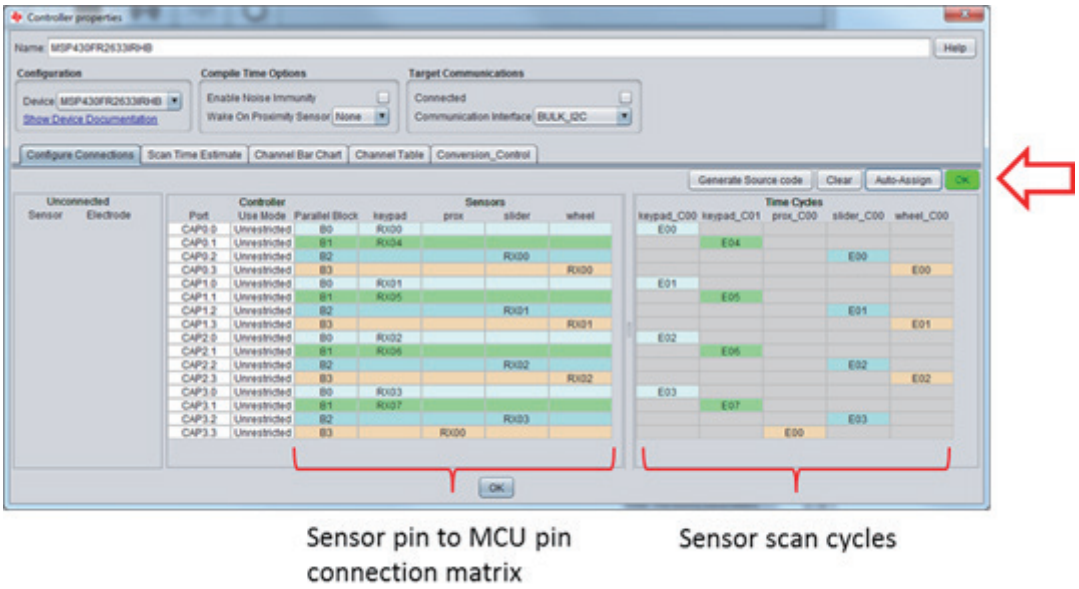


图 187：自动分配成功

既然端口映射已知，那接下去就计算时间周期。Configure Connections 选项卡上的第三个方框显示了时间周期的概要。一个时间周期是一组以并行方式进行测量的元件。请注意，小键盘传感器在两个时间周期中测量，而接近、滑块和滚轮传感器则各需要一个时间周期。如果器件仅支持顺序测量，则此面板将花 16 个周期来测量每个传感器。该器件仅需 5 个周期就能测量所有的传感器，从而缩减了测量时间和功耗。

注：自动分配功能可用于为一个已经存在的开发板映射端口，因为自动分配功能是用于为 CAPTIVATE-BSWP 面板生成原始端口映射。

保存当前的 Design Center 项目。选择 Project ➞ Save As 并把项目文件夹的名称改为 LAB1，如图 188 中所示。

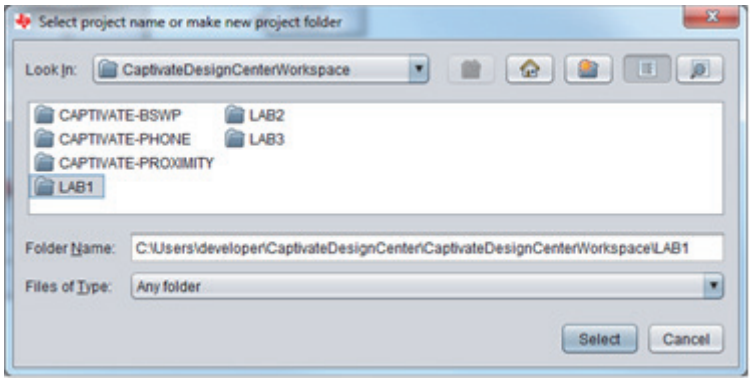


图 188：保存 LAB1

11.8.2.6 生成启动项目目标 MCU 代码

既然针对传感器的传感器配置已经就绪，下一步便是生成一个启动代码项目 (starter code project)。启动代码项目是一个针对 MSP430FR2633 的全功能 Code Composer Studio 和 IAR 固件项目，可用于对目标 MCU 进行编程并开始调优过程。

11.8.2.6.1 生成启动代码项目

在控制器定制器中，在 Configure Connections 选项卡的下方选择 Generate Source Code 按钮。

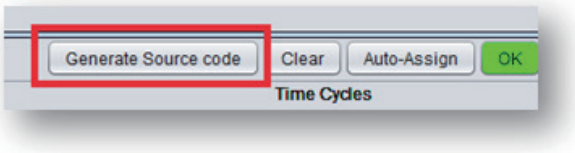


图 189：生成启动代码项目

保存启动代码项目。一个对话框提示您创建一个新的项目或者更新一个现有的项目。选择“Create new project”，然后点击 OK 以接受用于新启动项目的默认位置。

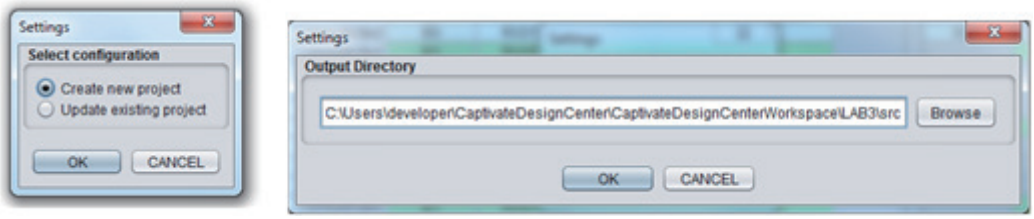


图 190：保存启动代码项目

Design Center 在 Design Center Workspace 目录下创建启动代码项目。在下一步中，该项目的一个副本被导入 CCS 工作空间目录。

11.8.2.6.2 构建 LAB1 启动项目代码

把启动代码项目导入 CCS。将 LAB1 启动项目源代码导入到一个新项目并至 CCS 工作空间。在导入该项目之后，点击调试器图标以构建代码并将代码下载至目标。

注： 在 CCS 中第一次构建一个 CapTIvate 源代码项目时，如果需要构建小代码小数据运行库，则编译程序有可能显示如下的消息。视所采用 PC 的不同，这或许需要几分钟的时间。

```
warning #10366-D: automatic library build: using library
"C:\ti\ccsv6\tools\compiler\ti-cgt-msp430_4.4.3\lib\rts430x_sc_sd_eabi.lib" for the
first time, so it must be built.  This may take a few minutes.
```

图 191：小内存运行库构建消息

把目标设定为在自由运行模式中运行，或简单地停止调试会话。点按 MCU PCB 上的复位按键。

注： 在一个 POR 或 RESET 之后，MCU 在通信开始之前需要一秒或两秒的时间以校准传感器。

与 Design Center 进行通信。在 Design Center 中，关闭所有的定制器窗口以使仅主版面可见，并显示所有的传感器。从 Design Center 菜单栏选择 Communications → Connect。来自目标的传感器数据流现在出现于 BSWP 演示板的主视图中。

11.8.3 第二部分 — 调优传感器

传感器尚未得到调优。每个传感器以 Design Center 分配的默认值来操作。这些默认值是良好的起点，但是有改善的余地。这就是本节的目标 — 改进默认值并选择最适合该应用的数值。

察看一下用于系统中 16 个通道的个别元件数据。打开 Controller 定制器并选择 Channel Bar Chart 选项卡。所有 16 个通道在这里显示其元件数据。当触摸各种不同的传感器时，蓝色 (BLUE) 长条（绝对计数测量）减少，因为这是一个自电容面板。实心的红色 (RED) 长条指示长期平均值 (LTA) 和当前计数之间的差异。当某个元件被触摸时，这些数值增加。当您与面板互动时，请注意默认门限相对于计数的位置。

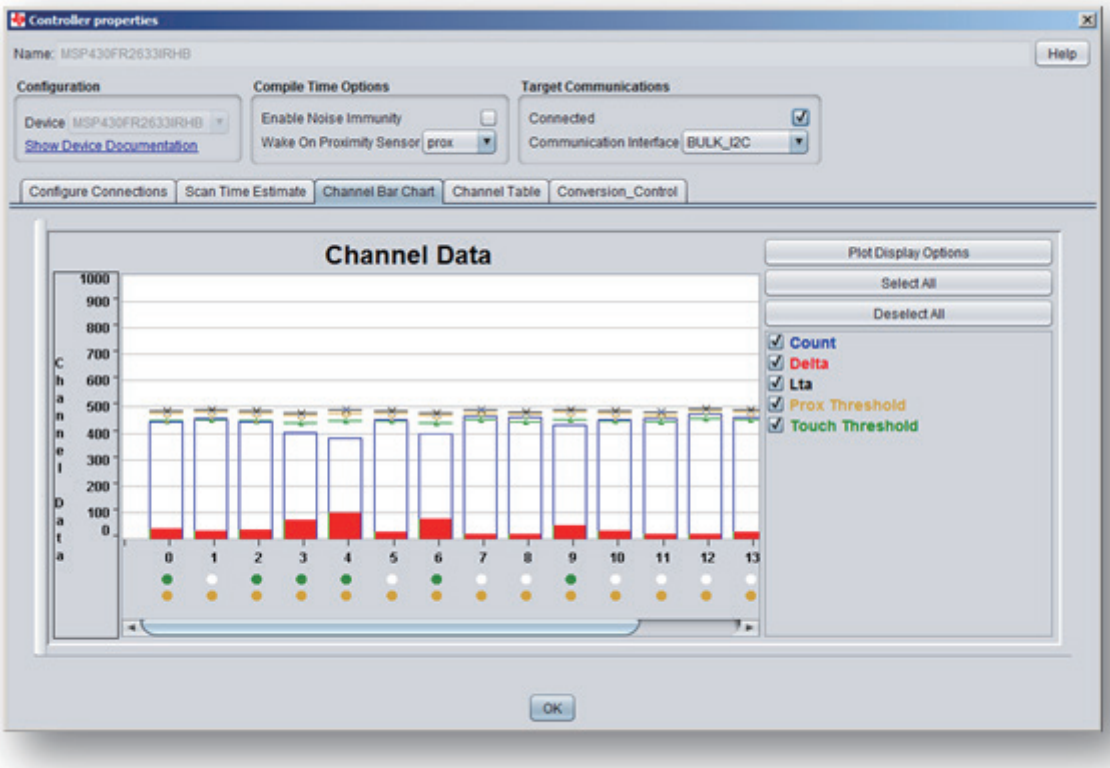


图 192：通道条视图

Design Center 为所有的传感器分配了默认的接近和触摸门限。这些数值必须针对每个元件进行微调，因为每个元件具有一个稍有不同的敏感度和寄生电容。通过打开 CCS 中的 CAPT_UserConfig.c 文件来检查这些默认值。搜索 .ui8TouchThreshold 或 .ui16ProxThreshold。接近门限被分配给传感器，而触摸门限则被分配给传感器中的每个单独元件。

注： 接近门限值是相对于长期平均值 (LTA) 的一个绝对偏差。触摸门限值是动态的，且被表示为 LTA 的一个百分数。如需了解更多，则双击一个传感器，点击 Tuning Tab，然后点击 Prox Threshold 或 Touch Threshold Parameters 以提出综合帮助主题。

调优是按逐个传感器的方式进行的。每个传感器具有一个定制器窗口，该窗口通过双击版面上的传感器 控件 来打开。

11.8.3.1 调优一个按键组传感器

首先从按键组传感器开始。

提示：为了在传感器调优过程中提供帮助，可移出任何一个传感器选项卡，以在一个选项卡中查看实时数据，同时在另一个选项卡中修改某个调优参数（见图 193）。

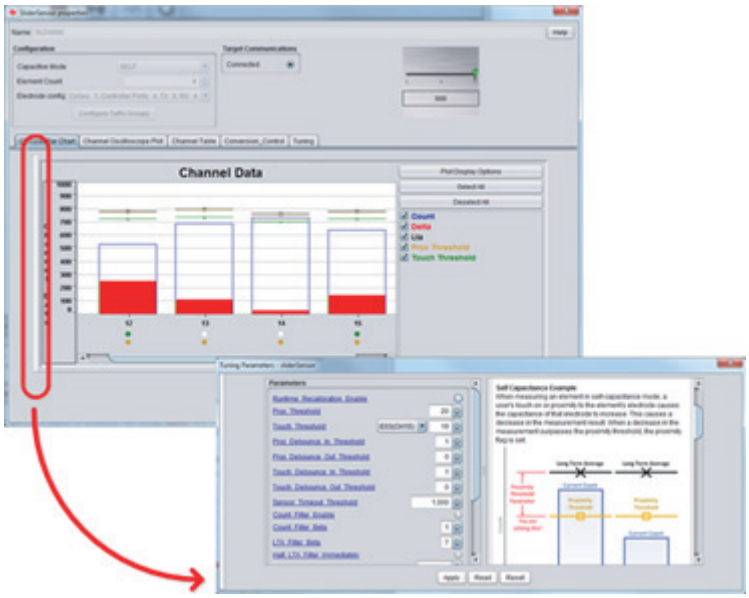


图 193：移出选项卡

11.8.3.1.1 转换计数和转换增益

对转换计数 (Conversion Count) 和转换增益 (Conversion Gain) 进行实验。关闭控制器定制器，并通过双击版面上的小键盘传感器打开小键盘传感器定制器。该定制器示出了与控制器视图相同的条形图，不过这里仅示出了用于小键盘传感器中的 8 个元件的 8 个长条。

触摸面板上的按键 1，同时查看 Design Center 中的数据。请注意，一个触摸在测量中产生了大约 75 个计数的变化 — 当被触摸时，计数从约 500 个（也是默认的转换计数规格）变至大约 425 个。

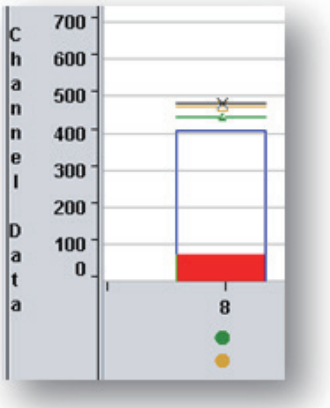


图 194：按键条

获得 75 的计数变化量是很方便的，但是确定某种触摸条件并不需要那么多的信息。系统是相当稳定的（只有几项噪声），因此额外的分辨率不是必要的，而且分辨率的提高以增加扫描时间和功耗为代价，这在 LAB3 节中讨论。

把转换计数改为 250（而不是默认值 500），并测试是否仍然可提供用于 确定某种触摸条件的足够信息。打开按键传感器定制器中的 Conversion Control 选项卡。如果任何默认设置都未改变，则转换控制窗口应看起来像图 195 那样。

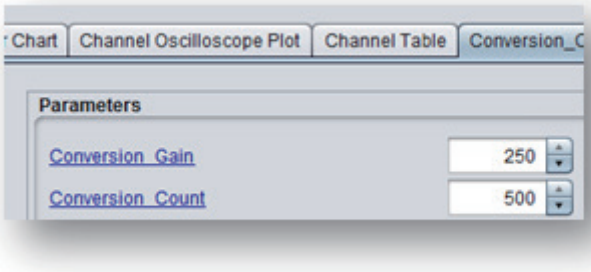


图 195：转换控制

Design Center 选择了一个 500 的转换计数 (Conversion Count) 和一个 250 的转换增益 (Conversion Gain)。这为确定一个触摸提供了足够的信息，但是更低的数值也是可以奏效的。把转换计数变为 250，并将转换增益改至 100。转换增益是 MCU 在不使用寄生电容偏移减除时所需的基值。总转换计数是在增添了偏移减除之后所需的计数。

图 196：转换控制

在一个调优参数于 Design Center 中调整了之后，Apply 按键点亮为黄色，提示您把参数变更应用到目标微控制器。选择 Apply 按键以提交变更。切换回 Channel Bar Chart 选项卡时，所做的变更应立即可见。基线（无触摸）测量规格化在大约 250 个计数，而触摸按键将把计数减少约 200，从而提供了一个 45 至 50 个计数的变化量。

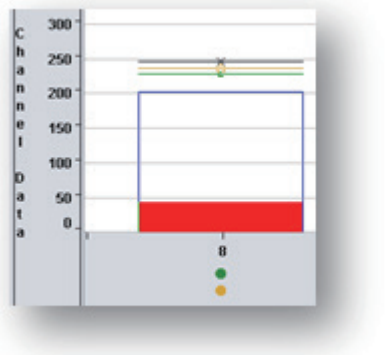


图 197：关闭按键条

11.8.3.1.2 接近和触摸门限

既然选择了一个适当的测量分辨率，那么就把接近门限和触摸门限调节至可提供期望的按键触感的数值。当触摸面板上的按键 1 时，条形图上的绿色条代表触摸门限的位置，而琥珀色条则代表接近门限。由于蓝色条（当前计数）超过了接近门限（琥珀色）和触摸门限（绿色），因此接近和触摸指示灯均被点亮。

因为一个完整的触摸提供了大约 50 的计数变化量，因此默认的触摸门限使得元件有点过于敏感了。应用需要一种以按键为中心的结实触摸。通过编辑触摸门限以要求更加结实的触摸。

触摸门限针对某给定元件定义了使该元件进入触摸状态所需的变化量大小。触摸门限被规定为长期平均值 (LTA) 的一个百分数。因此，实际的绝对值在运行的时候动态地计算。在 Design Center 中，该门限以长期平均值 (LTA) 的 1/128 为单位输入。通过将触摸门限规定为长期平均值 (LTA) 的一个百分数，使得该门限与 LTA 成比例，即使在 LTA 漂移的情况下也不例外。默认的触摸门限值为 10。由于设定了一个 250 的转换计数，因此有效触摸门限为 250 / 128 x 10 = 19。现在查看 Channel Table 中的触摸门限可见：绝对门限为 250 − 19 = 231。

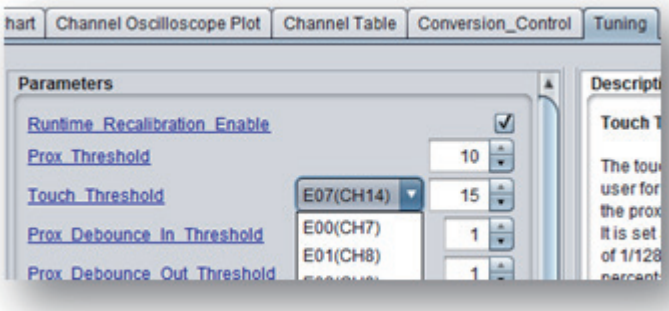


图 198：接近和触摸门限

对于该系统，把触摸门限增加至 15。数值为 15 的触摸门限提供了一个 250 / 128 x 15 = 29 的有效触摸门限。这需要一种更结实的触摸以便检测到该触摸。触摸门限在传感器定制器的 Tuning 选项卡中设定，就像 Conversion Count 和 Conversion Gain 一样。由于小键盘最多由编号从 E00 至 E07 的 8 个元件构成，按键 1 对应于电极 E00 (CH8)，因此需要修改针对所有 8 个元件（E00 至 E07）的触摸门限。这些变更改善了按键的触感。

如需更多地了解触摸门限的工作原理，则在编辑某个门限时阅读 Design Center 中的 Description 方框。可随意地调节接近门限。只有一个接近门限。该门限适用于整个传感器。

11.8.3.1.3 计数滤波

在条形图中，当一个按键被触摸时，计数值几乎立即下降 50 个计数。把计数滤波器调节得更加积极。在 Tuning 选项卡中，将计数滤波器 β 值设定为 2（默认值为 1）。这意味着对于每个新数值具有一个 25% 的权重，而不是一个 50% 的权重。因此，传感器的响应速度将减缓一半。另外，滤波器 β 值的增大还略微地降低了系统上的噪声。

图 199：计数滤波器

在做出变更之后选择 Apply 按键。现在条形图显示了一个速度稍慢的响应。察看滤波器响应的另一条途径是“通道示波器曲线图” (Channel Oscilloscope Plot) 选项卡。这是显示历史的条形图的一种替代视图，而且它是一种察看针对滤波器变更之响应的方便手段。

该曲线图示出了一个计数滤波器 β 值和一个 LTA 滤波器 β 值。LTA 滤波器 β 值影响着在无人触摸面板时（没有出现超越门限的情况）LTA 跟踪基础计数变化的速度。针对长期平均滤波器的默认滤波器 β 值为 7。计数滤波器 β 值和 LTA 滤波器 β 值的可用选项均为 0 ～ 7。β 值每增加 1，滤波器的长度都将倍增。例如，β 值从 1 变为 2 将使滤波器对一个阶跃输入的响应速度提高一倍。

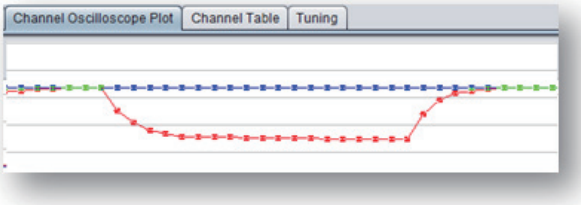


图 200：Oscillscope 选项卡

该按键现在应获得了可接受的初始调优。

11.8.3.2 调优一个滚轮传感器

关闭按键传感器定制器，并通过双击版面上的滚轮传感器 控件 以打开滚轮传感器定制器。

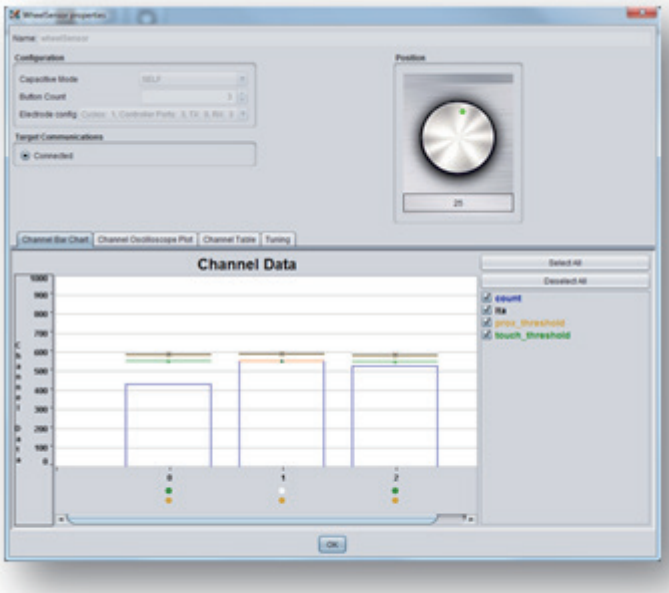


图 201：滚轮传感器视图

11.8.3.2.1 滚轮转换计数和转换增益

和按键传感器一样，滚轮传感器具有一个 500 的默认转换计数和一个 250 的默认转换增益。当一根手指跨越滚轮移动时，这些设置将产生一个大约 100 的计数变化量。与预先调优的按键不同，滚轮采用每个元件的计数变化量作为位置函数的一个输入，以计算滚轮正在被触摸的位置。因此，增加转换计数以提供额外的分辨率是有益的。把转换计数设定为 800，并将转换增益设定为 125。

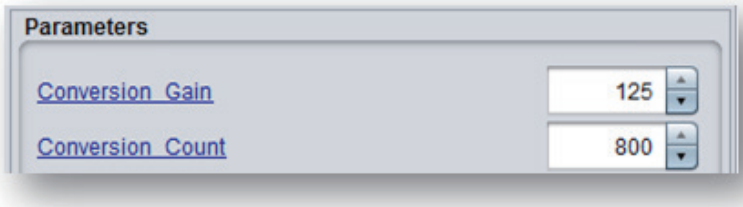


图 202：转换控制

计数和增益的增加使每个元件上的最大变化量增加了约 2 倍。位置计算可从分辨率的这种增加中极大地受益。

11.8.3.2.2 滚轮分辨率

默认的滚轮分辨率为 1000 点。大多数应用不需要这么多的分辨率，所以将其减少至一个更加可用的数值 100。把期望的分辨率调优参数从默认值 1000 变更为 100。在应用了变更之后，请注意滚轮的响应仍然和其在 1000 个计数时具有完全一样的线性，但是现在数据实际上可用性高得多了，因为可以容易地使用转盘来选择任何特定的数字。算法支持高达 16 位的分辨率。

11.8.3.2.3 滚轮传感器位置滤波

位置滤波器系数可以 255 种不同的步进实施调节。255 代表了一个单位响应（无滤波），而 1 则代表了最强的滤波器。通过把滚轮滤波器系数从默认的 100 变更为 150 以进一步增加滚轮的响应时间。滚轮响应现在相当地快了，而且仍然有足够的滤波以提供抖动极小的稳定位置报告。作为测试，将传感器滤波器设定为 25 并查看滚轮是怎样响应过度滤波的。

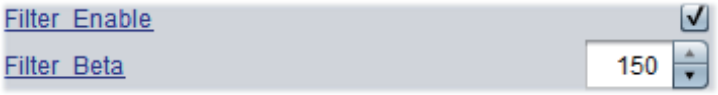


图 203：转换控制

11.8.3.2.4 滚轮触摸门限

由于敏感度在 Conversion Count 和 Conversion Gain 步长中显著地增加，因此滚轮对于触摸极其敏感。需要注意的是，当您把手指悬停在滚轮上方时，元件早在您实际触摸面板之前就进入了一种触摸状态。为使系统仅对触摸做出响应，就像按键组传感器一样更新数值。27 或 28 的设定值远远好于默认值 10。和平时一样，记住要更新用于所有 3 个元件的触摸门限。

11.8.3.3 调优一个滑块传感器

现在已经配置了滚轮，接着就可以配置滑块了。

11.8.3.3.1 滑块转换计数和转换增益

就转换计数 (Conversion Count) 和转换增益 (Conversion Gain) 而言，滑块与滚轮是完全一样的。把转换计数设定为 800，并将转换增益设定为 125，以增加敏感度及为位置算法提供更多的分辨率。双击滑块传感器以打开其定制器，并点击 Conversion Control 选项卡。

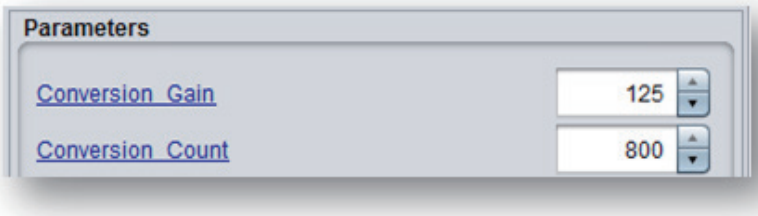


图 204：转换控制

11.8.3.3.2 滑块端点微调

除了具有两个端点之外，滑块在其他方面与滚轮都是非常相似的。所有相同的原理均适用，但是存在两个新的参数：滑块下限校正 (Slider Lower Trim) 和滑块上限校正 (Slider Upper Trim)。

选择定制器中的 Tuning 选项卡。上限校正 (Upper Trim) 和下限校正 (Lower Trim) 是参数窗口中的最后两个参数。

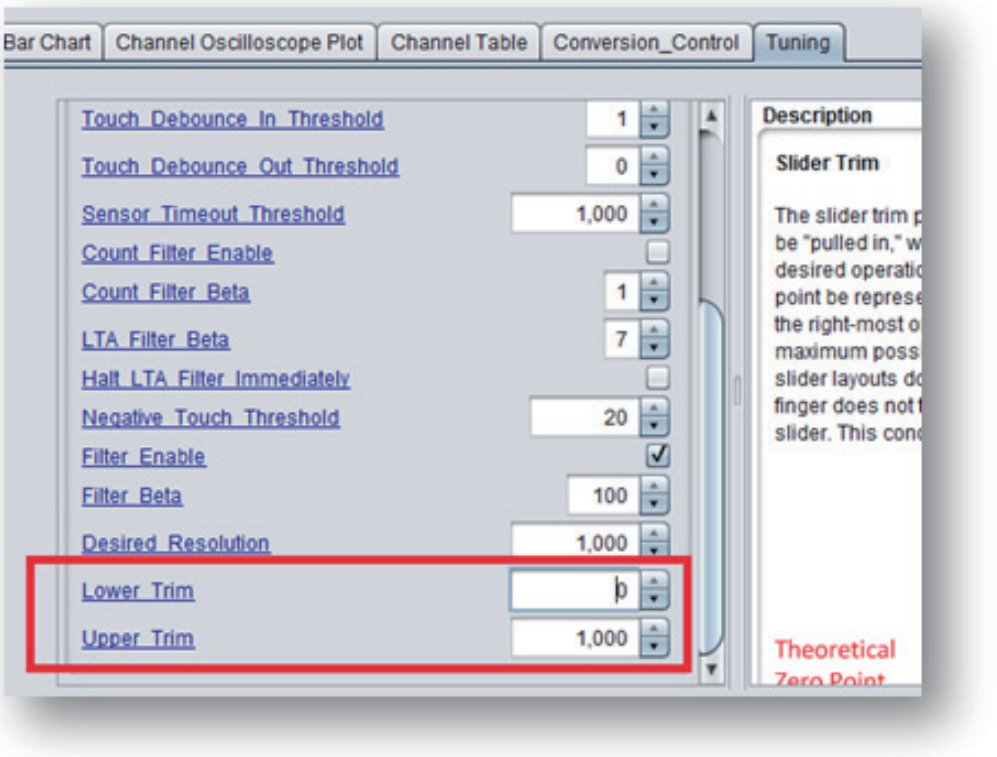


图 205：滑块校正

当用您的手指上下移动滑块时，请注意滑块值范围从大约 40 至 960，尽管对于该传感器需要一个 0 至 1000 的分辨率。滑块值绝对不会达到 0 或 1000，因为可以把滑块看作是切成两半的滚轮，而且端点绝不会被其对面的元件所牵拉。为消除这一困扰，有一种通过修整边缘来拉伸滑块的机理。目前，下限校正值被设定为 0，而上限校正值被设定为 1000。

通过把校正值提供给滑块来应用拉伸，以重新对齐零点和最大值点。如欲设定滑块校正值，则把手指放置在滑块最左边的部分上，并察看响应。数值通常高于 0 但小于 100。该位置需被定义为零，这样用户在使用滑块时能够完全到达 0。把您触摸滑块最左边部分时显示的测量值输入 Tuning 选项卡中的 Lower Trim 参数框里。

接着，把手指放在滑块最右边的部分里。一个介于 900 至 1000 的数值是典型的（因为最大值为 1000 点）。把该数值输入 Upper Trim 参数框中。把更改应用到控制器并审查结果。对于该滑块，图 206 中的数值可与已经规定的其他设置很好地配合工作。

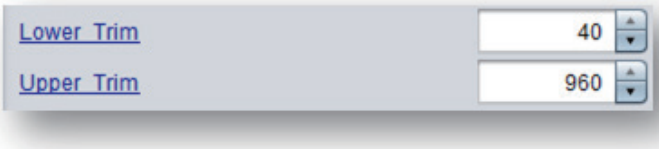


图 206：选择滑块微调值

对于滑块传感器不需要其他的新参数。除了校正值以外，滑块的运作与滚轮完全相似。

11.8.3.3.3 滑块触摸门限

由于敏感度在 Conversion Count 和 Conversion Gain 步长中显著地增加，因此滑块对于触摸极其敏感。当您把手指悬停在滑块上方时，元件早在您实际触摸面板之前就进入了一种触摸状态。在这里，就像按键组传感器和滚轮传感器一样更新数值。23 至 25 的设定值远远好于默认值 10。确保更新用于滑块中所有 4 个元件的触摸门限。

11.8.3.4 接近传感器调优

接近传感器具有与按键组传感器完全一样的参数，但是只有一个元件。设置接近传感器以观察能够实现什么类型的接近距离。

11.8.3.4.1 接近转换计数和转换增益

500 的默认转换计数和 250 的默认转换增益没有提供非常大的敏感度。当采用默认接近门限 10 时，传感器仅可获得 1 cm 的检测距离。通过把转换计数设定为 800 和把转换增益设定为 100 来增加敏感度。

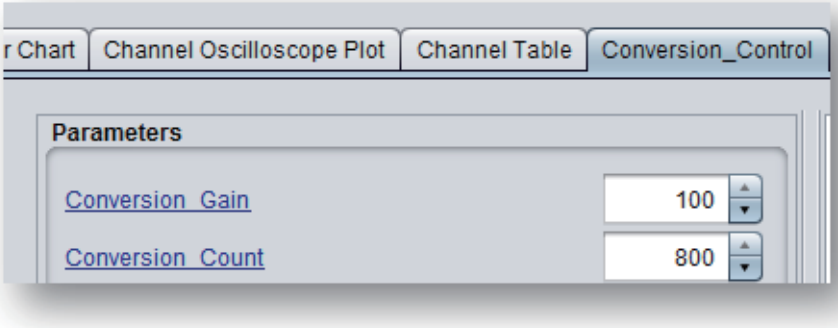


图 207：选择接近微调值

请注意敏感度的改善。检测距离增加到了 5 cm 以上。

11.8.3.4.2 关于改变触摸门限的用途

由于接近门限是用来检测什么时候有人靠近面板的，因此触摸门限可用于执行另一项任务。因为 CAPTIVATE-BSWP 面板上的接近传感器布局遍及 PCB 的各个位置，所以其可用于检测是否有人倚靠在面板上或者把整只手放在面板的顶部。此类信息可用于阻断其他按键的输出。设想一种安装在墙上的安全小键盘应用。如果有人倚靠在它上面，则检测到这种情况将是件好事，可防止所有其他的传感器同时触发。可设定触摸门限以检测此类场景，而接近门限则用于检测距离。

11.8.3.5 其他的传感器参数

本节说明的是一些尚未讨论过的通用传感器参数。第一个是传感器超时门限。

11.8.3.6 传感器超时

传感器超时门限提供了一种用于避免发生卡键的情况。设想墙壁式恒温器上的触摸屏。想象一下，如果有人用扫帚棒抵住了恒温器，这就呈现为一种接近检测。接近检测具有中止长期平均 (LTA) 的作用。因此，LTA 将不再跟踪环境漂移。这影响了触摸屏的可用性。通过规定（在采样中）传感器在互动被作为无效而废弃之前能够在检测（接近或触摸）状态中停留多久，超时功能避免了这种情景的出现。

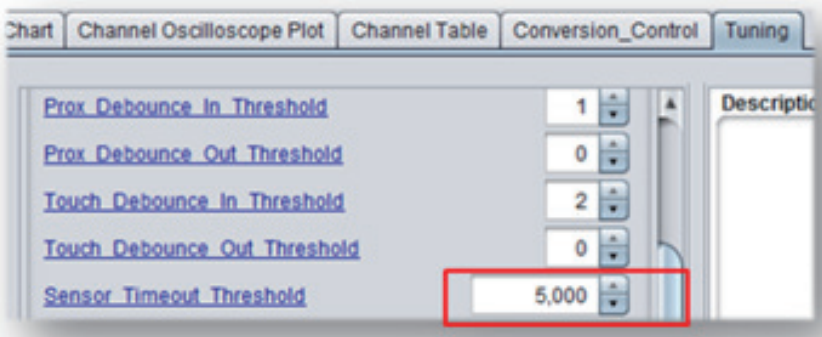


图 208：选择滑块超时值

用户通常是简短地触摸传感器，不会持续 30 秒或更长的时间。通过设定一个较低的传感器超时门限，系统能够在由于某种原因而长时间地困在检测状态时复位。对于测试，通过设定极短的超时进行该工具的仿真；在检测了 200 个样本之后（< 10 秒），系统使触摸标志复位并补种长期平均值。

11.8.3.6.1 传感器去抖 (Sensor Debounce)

该触摸库提供了一种用于转入和转出接近及触摸状态的内置去抖机制。去抖可在传感器水平上进行调节。

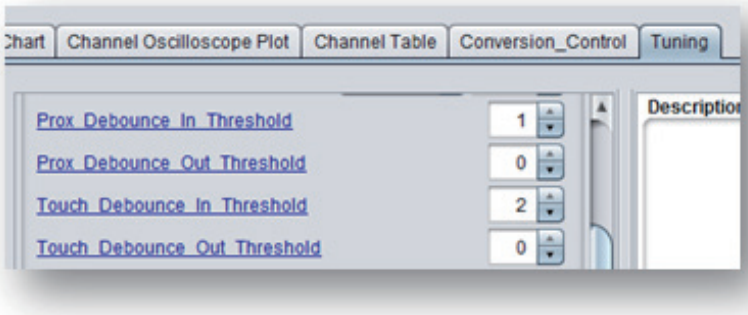


图 209：选择滑块防去抖

用于 Debounce 的默认设置为 0（未应用去抖）。把去抖应用到系统可确保实际上存在一个有效的触摸（而不是手的掠过或噪声事件）。不过，去抖将延迟引入到了系统之中。

对于测试，设定越来越长的去抖设置以观察其对传感器性能的影响。

11.8.3.7 扫描时间估计 (Scan Time Estimation)

基于针对本项目配置的传感器，一种时间表或扫描时间估计显示了总扫描周期、扫描每个传感器的时间、以及用于所有传感器的组合时间。处理数据和把数据传输至 Design Center 所需的时间不含于此，只包括了传感器测量扫描时间。如欲查看扫描时间估计，则打开 Controller Customizer 并选择 Scan Time Estimate 选项卡。

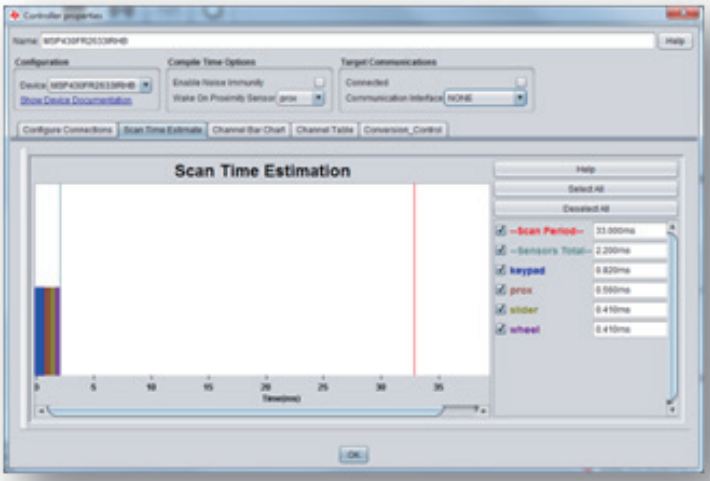


图 210：Scan Time Estimation 视图

11.8.4 第三部分 – 生成更新的传感器配置

现在调优了传感器，下面就把这些新的调优值应用到源代码。该更新并不需要再生一个全新的项目，简单地采用 Design Center 中的 Update Code 特性来更新 CAPT_UserConfig.c 和 CAPT_UserConfig.h 文件即可。首先，将 Design Center model (*.ser) 的一个更新副本另存为调优的一个已确认良好副本。然后，打开 Controller 定制器，并选择 Generate Source Code 按键。选择“更新现有项目”(Update existing project) 配置。

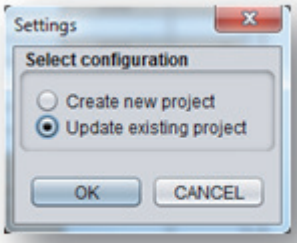


图 211：Update existing project 配置

选择在 FIND SECTION 部分（其位于 CCS 工作空间，而不是生成启动项目的工作文件夹）中产生的 CCS 项目的根目录。

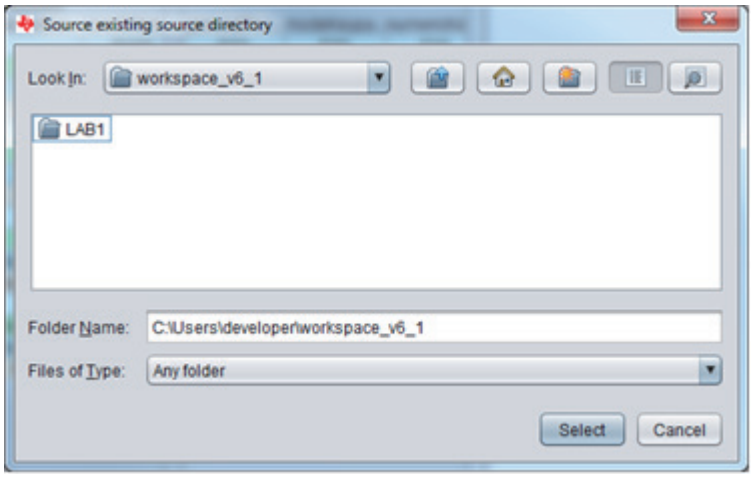


图 212： CCS 项目的选择目录

重建 CCS 项目并重新运行它，所有新的调优值全部生效。

祝贺您！您已经成功地完成了自己的首个 CapTIvate 传感器设计。11.9 节将说明如何使用低功耗接近唤醒功能来为传感器设计测量电流。

11.9 有关低功耗的实验

11.9.1 LAB2

LAB2 演示接近唤醒功能，其能够通过采用 CapTIvate 硬件状态机以扫描接近传感器（与此同时 CPU 处于低功耗模式 3）而实现极低的功耗。该 LAB 说明了传感器调优和扫描速率如何能够影响低功耗操作。EnergyTrace™ 功能在此实验室动手操作中测量功耗。

11.9.1.1 先决条件

LAB2 基于 LAB1 中所完成的工作，所以如果您跳过了那一步，则查阅 11.8 节。

11.9.1.2 目标

- 学习如何使用 EnergyTrace 进行功耗测量
- 观察调优传感器如何影响着低功耗

11.9.1.3 硬件设置

继续使用前一个实验室动手操作 (LAB1) 所采用的相同硬件设置。在 MCU PCB 上，在 MCU_VCC 和 3.3V METERED 之间的移动跳线 J3。

图 213：跳线移动至 3.3 V METERED

11.9.1.4 Design Center 项目

在着手进行该实验室动手操作之前，把当前的 CapTIvate Design Center 项目另存为 LAB2。选择 Project ➞ Save As 并将文件夹名称变更为 LAB2。

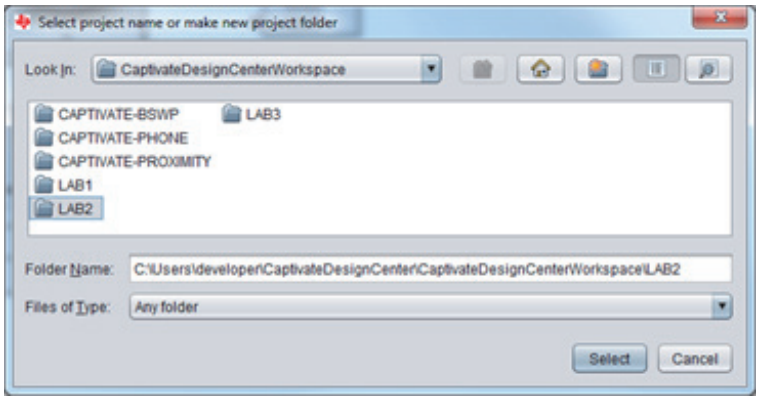


图 214：另存为 LAB2

11.9.1.5 增添触摸唤醒功能

CapTIvate Design Center 可以在源代码中使能某些特定的功能。此类功能之一是“接近唤醒”，也被称为“触摸唤醒”，其允许 CPU 在 CapTIvate 硬件状态机扫描某种接近情况时处于一种低功耗模式。

11.9.1.6 选择接近唤醒传感器

使用控制器定制器 Compile Time Options 框，从下拉菜单选择一个接近唤醒传感器。这种选择可挑选在接近唤醒期间扫描哪个传感器。对于本实验室动手操作，选择 prox sensor。

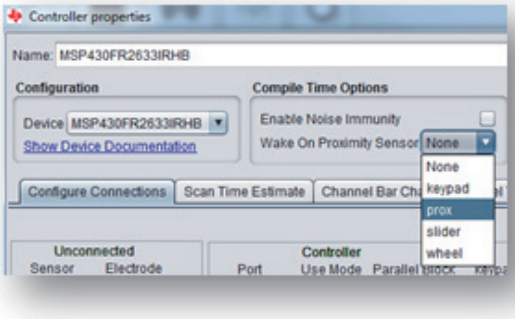


图 215：选择接近唤醒传感器

11.9.1.7 修改接近唤醒参数

在控制器的 Conversion Control 选项卡中，若干参数定义了 CPU 和接近唤醒操作。

唤醒间隔 (Wakeup Interval) 规定了 CPU 应自动唤醒的频度，这与接近或触摸事件无关。这是一种用于接近唤醒状态机的内置安全机制。对于本实验室动手操作，停用 Wakeup Interval 以使接近唤醒状态机无限期地运行，这为实施低功耗测量提供了所需的时间。

11.9.1.8 生成 LAB2 启动项目代码

生成启动代码项目。在控制器定制器中 Configure Connections 选项卡的下方选择 Generate Source Code 按键。选择 Create new project 并使用默认的输出目录。

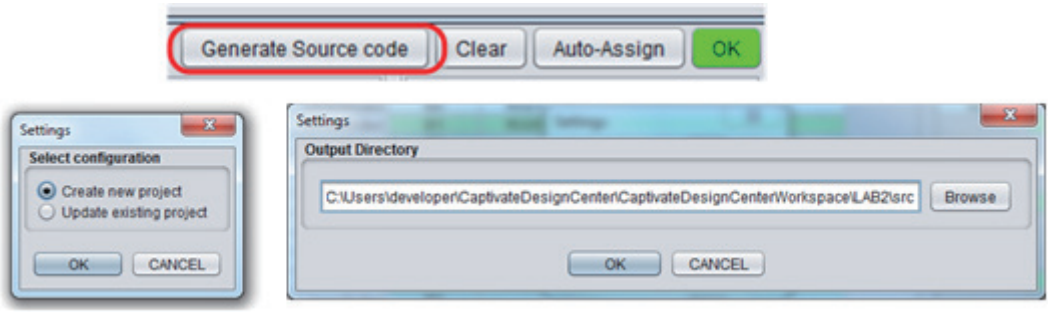


图 216：创建和保存启动代码项目

11.9.1.9 启动目标 MCU

把 LAB2 启动代码项目导入 CCS。

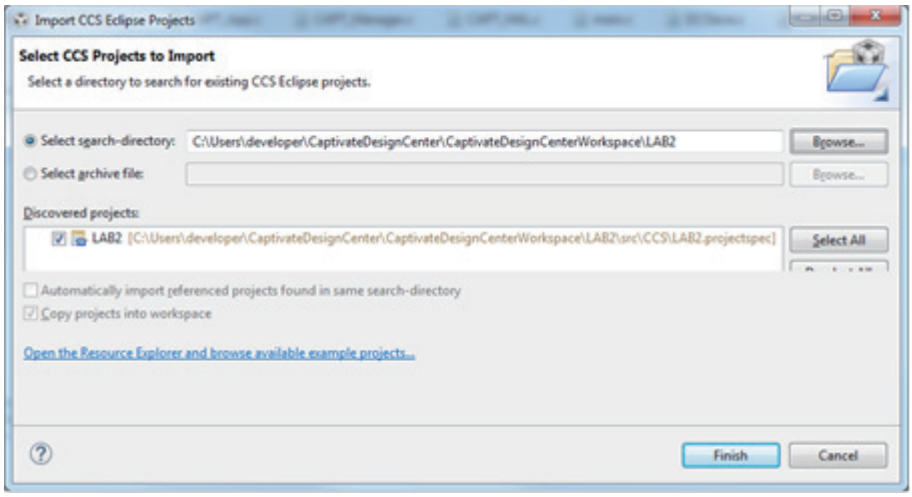


图 217：把启动代码导入 CCS

在导入了项目之后，点击调试器图标以构建代码并将代码下载至目标。把目标设定为在自由运行模式中运行 (Run → Free Run) 或终止调试会话。停用调试器将停用 MCU 中的 EEM 模块，从而减小了由 MCU 所吸收的电流。

11.9.2 采用 EnergyTrace 技术

EnergyTrace 技术内置于 CAPTIVATE-PGMR PCB 上的 eZFET 编程器。EnergyTrace 技术提供了一个 DC/DC 稳定输出，并监视一个负载（目标 MCU）所消耗的能量。使用 EnergyTrace 功能来测量目标 MSP430FR2633 中的功耗和电流。

重要说明：虽然 EnergyTrace 技术能以合理的准确度测量非常小的电流，并提供了一种用于低功耗演示的简单工具，但强烈建议使用一个工作台电源、高准确度仪表、以及与 eZFET 编程器隔离的目标来进行低功耗测量。

如欲启用 EnergyTrace 技术，则点击图 218 中示出的图标。假如该图标未出现在菜单栏上，则请查阅“用于 MSP430 的 Code Composer Studio v6.1 用户指南 (Code Composer Studio v6.1 for MSP430 User's Guide)”(SLAU157) 中的 EnergyTrace 技术章节。

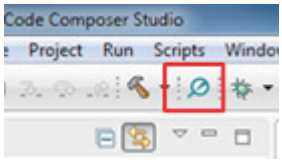


图 218：启用 EnergyTrace

一个类似于图 219 所示的新选项卡在 CCS 中打开。

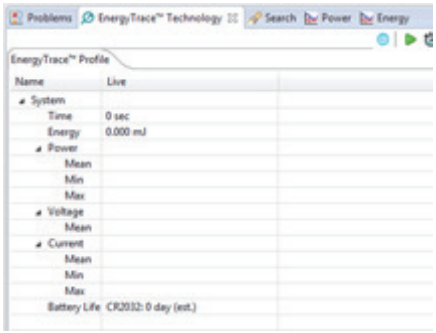


图 219：EnergyTrace 选项卡

设定了 30 秒的 EnergyTrace 捕获长度。这为捕获该实验室动手操作的能耗轨迹提供了足够的时间。

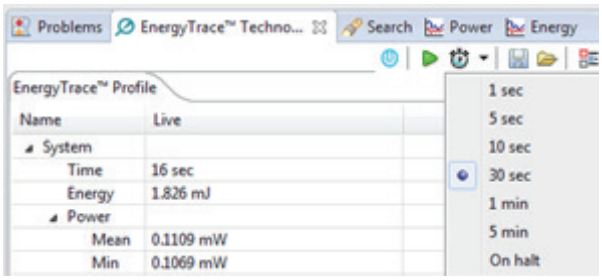


图 220：EnergyTrace 捕获

目标 MCU 运行于没有调试器的自由运行模式。如欲使目标 MCU 复位，则揿按 MCU PCB 上的 RESET 按键，然后等待 LED2 闪烁一次。通过点击绿色播放箭头来启动 EnergyTrace 捕获（见图 221）。

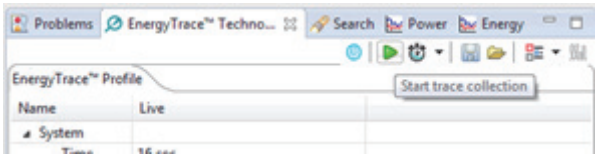


图 221：启动 EnergyTrace

11.9.2.1 第一部分 – 测量接近传感器低功耗电流

首先，测量接近唤醒平均电流以建立一根基线。在实施此项测量时一定要避免手接触 PCB。EnergyTrace 平均电流测量值通常小于 10 µA。这是采用默认转换计数 (800) 以每秒 10 次的频度测量接近传感器所需的电流。如果测量值远大于 10 µA，则揿按 RESET 按键，等待 LED2 闪烁一次，然后重复进行该测量。如果测量值为 0，则确保跳线 J3 被设定在 MCU PCB 上的 MCU_VCC 和 3.3V METERED 之间。记录测量结果。

11.9.2.2 第二部分 – 传感器敏感度对低功耗电流的影响

其次，测量传感器调优会对器件电流消耗所产生的影响。BSWP 演示板具有一个沿着 PCB 的边沿排布的接近传感器。该接近传感器被接近唤醒状态机用来检测何时有手靠近了 PCB。接近传感器的调优就像按键一样，但是具有更高的敏感度。在 LAB1 中，接近转换计数被设定为 800，因而提供了充足的分辨率和大约 2 cm 开外的接近检测。通过增加传感器的分辨率可延伸检测的距离。

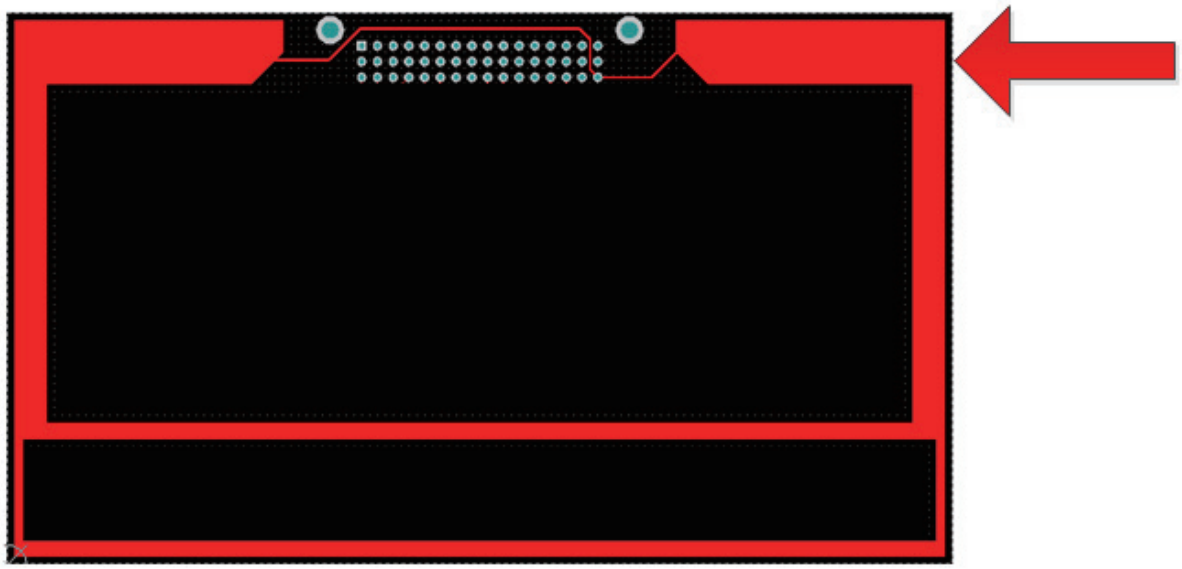


图 222：BSWP 接近轨迹

首先把转换计数增加至 1600。在 Design Center 中，双击接近传感器以打开其定制器，然后点击 Conversion Control 选项卡。把转换计数更新至 1600。然后点击 Apply 按键。点击 Channel Bar Chart 以查看新的转换计数数据并确认 LTA 接近 1600。验证一下：当您的手接近 PCB 时，现在可更早地检测到物体的接近（即在离 PCB 更远的地方就能检测）。接近传感器如今更加敏感，并能从更远的距离检测到物体的接近。如果在 转换计数为 1600 时数值不可见，则调节曲线图范围以查看通道条数据。采用曲线图显示选项 (Plot Display Options) 来调节曲线图参数，以适合通道条的新尺寸。

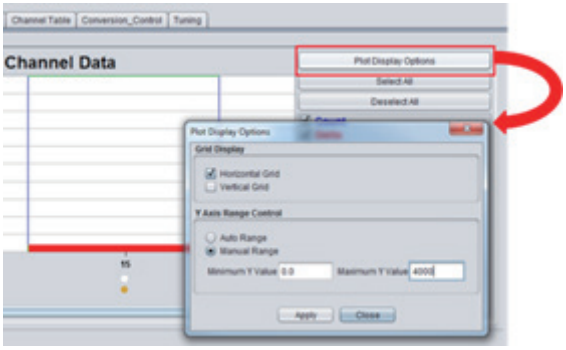


图 223：曲线图选项

鉴于这种较高的分辨率，当蓝色长条超越接近门限时也许更难看到。Design Center 具有一种放大功能，可提供选定区域的扩大视图。

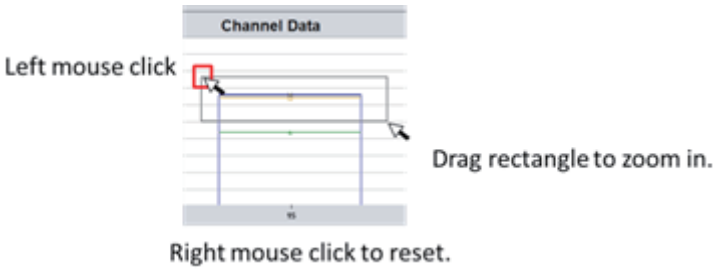


图 224：放大

接着，更改接近门限以防止发生误检测。选择接近传感器的 Tuning 选项卡并把 Proximity Threshold 和 Negative Touch Threshold 变更为 30。点击 Negative Touch 参数以在 Design Center 描述框中了解有关该参数的更多信息。

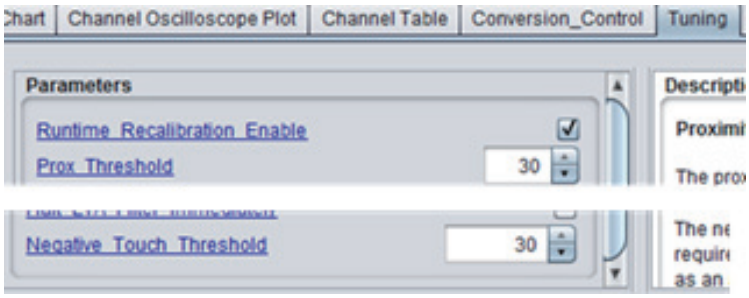


图 225：接近门限

由于灵敏度的增加，接近检测距离也增加了。如欲观察这如何影响了低功耗电流，则不要使 MCU 复位，因为这将把转换计数复位至 800，以及把门限复位至 15。启动 EnergyTrace 并检查平均电流。它通常从先前的基线测量值倍增。记录该测量值。

11.9.2.3 第三部分 – 传感器扫描速率对低功耗电流的影响

另外，对传感器进行扫描的速率也会影响器件的低功耗电流。

通过揿按 MCU 复位按键可使所有的参数复位至其默认值。如欲确认参数被复位至其原始编程值，则点击接近传感器 Conversion 控制选项卡，然后点击 READ 按键。这读取了目标 MCU 中的电流值。转换计数应为 800。

在 Design Center 中，双击控制器以打开其定制器并选择 Conversion Control 选项卡。把接近唤醒模式扫描速率 (Wake On Prox Mode Scan Rate) 修改为 50 ms，然后点击 Apply。这实际上使得对接近传感器的扫描速率倍增。

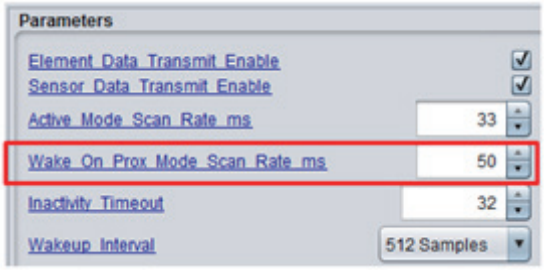


图 226：接近扫描速率

不要使 MCU 复位，因为这把扫描速率复位至 100。启动 EnergyTrace 并检查平均电流。该电流通常从基线测量值提高了一倍多。

11.9.3 总结

像传感器的扫描速率一样，对于更高敏感度和更高分辨率的需求直接影响着低功耗电流。所以在针对终端应用设计传感器时应始终考虑权衡折衷的问题。

祝贺您！您已经成功地对低功耗接近唤醒功能进行了试验，并获知了传感器调优和扫描速率会怎样对传感器设计产生直接的影响。

11.10 CapTIvate 触控函数库

11.10.1 LAB3

11.10.1.1 先决条件

LAB3 基于 LAB2 中所完成的工作，所以如果您跳过了那一步，则查阅 11.9 节。

11.10.1.2 目标

- 学习如何使用传感器管理器来简化传感器测量。
- 学习如何使用回调函数来处理传感器事件。
- 学习如何把接近、触摸和位置状态的采集作为回调函数的一部分。

11.10.1.3 硬件设置

继续使用前一个实验室动手操作 (LAB2) 所采用的相同硬件设置。在 MCU PCB 上，在 MCU_VCC 和 3.3V LDO 之间移动跳线 J3，并检查跳线对于两个 LED 处在适当的位置。

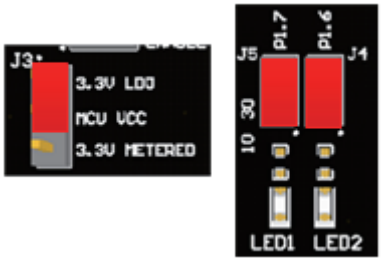


图 227：跳线

11.10.1.4 Design Center 项目

在着手进行该实验室动手操作之前，把当前的 CapTIvate Design Center 项目另存为 LAB3。选择 Project ➞ Save As 并将文件夹名称变更为 LAB3。

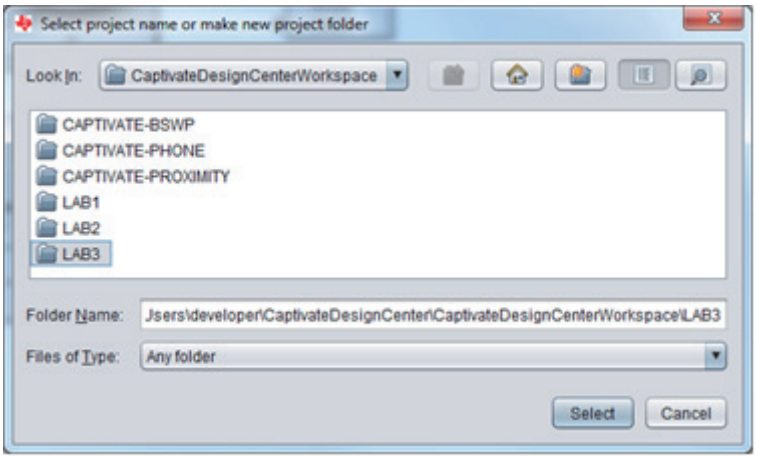


图 228：另存为 LAB3

11.10.1.5 生成 LAB3 启动代码项目

生成启动代码项目。在控制器定制器中 Configure Connections 选项卡的下方选择 Generate Source Code 按键。选择 Create new project 并使用默认的输出目录。

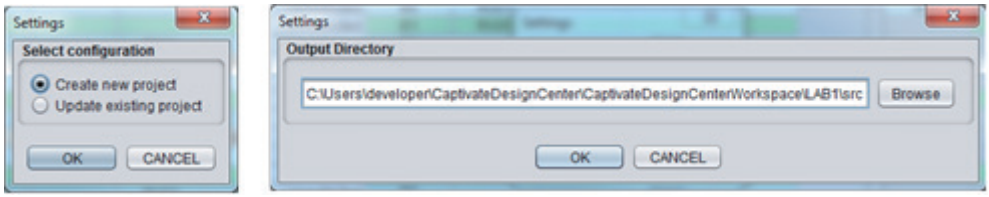


图 229：保存 LAB3 启动代码

选择 Design Center 通信接口。双击控制器以使定制器视图出现。在 Target Communications 中，选择 Connected 通信复选框并验证 Target Communications 接口被设定至 BULK_I^{2}C。

与在前一个实验室动手操作中一样，把启动代码项目导入 CCS。构建 LAB3 启动代码项目并对目标 MCU 进行编程。

11.10.1.6 第一部分 – CapTIvate 触控技术库概要

该软件库全面汇集了提供触摸、通信和传感器管理的功能和组件，如图 230 中描绘的软件栈所示。与触摸相关的主要软件层是 Advanced、Touch（或 Base）和 Communications。功能涵盖的范围从高级传感器处理到负责提供对 CapTIvate 外设的直接访问功能均在其列。

用于 I^{2}C 和 UART 串行驱动程序的通信功能和一种通信协议使得 FR26xx 或 FR25xx 微控制器能够在传感器开发过程中将传感器数据发送至 CapTIvate Design Center。

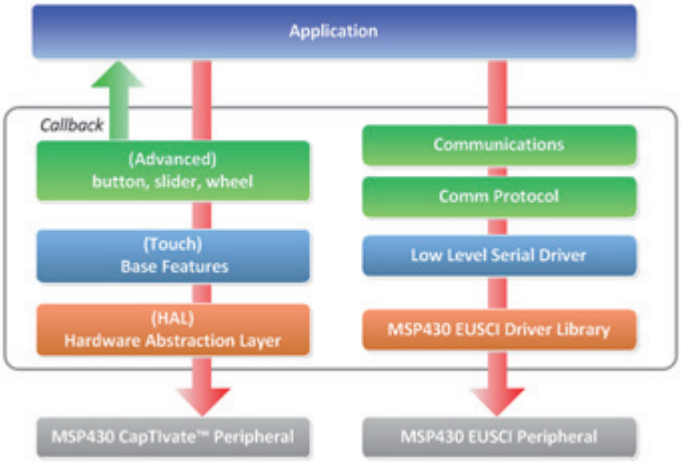


图 230：CapTIvate 触摸库

大多数库组件预编译在 MSP430FR2633 ROM 中提供。另外，它们还分别以针对 CCS 或 IAR 的 captivate.lib 或 captivate.r43 文件格式提供。未给这些组件提供源代码。采用 CapTIvate API 来访问这些组件。

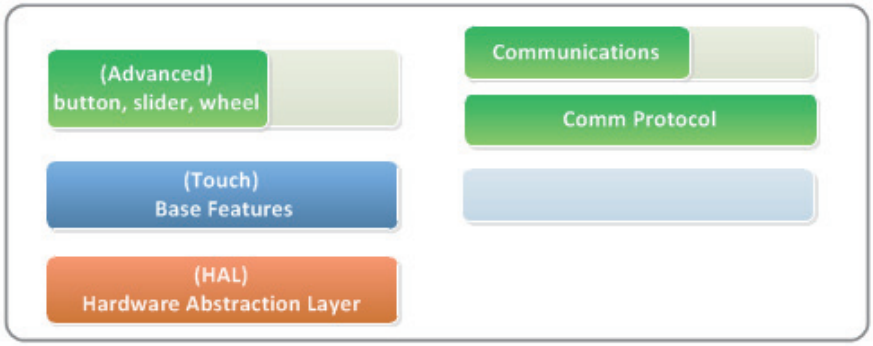


图 231：预编译库

其他组件作为源代码来提供，其可修改以适合应用的需要。库管理器 CAPT_Manager.c 提供了三个高级函数以管理主要的传感器活动：CAPT_initUI()、CAPT_updateUI() 和 CAPT_calibrateUI()。这极大地简化了与软件库的应用接口，并可用作一种针对应用的框架。库管理器是作为资源提供的。

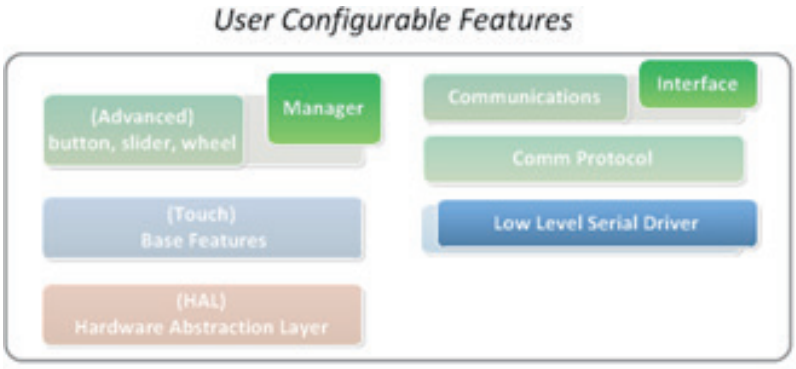


图 232：用户可配置的功能

11.10.1.7 第二部分 – 考察 CCS 项目结构和细节

考察 CCS 项目和库功能。

- CapTIvate 库组件位于 captivate 目录中。
  - 用于软件库的 API 头文件
  - ROM 头文件
  - 预编译库
  - 应用级传感器管理器
- CapTIvate 配置文件位于 captivate_config 目录中
  - 配置文件由 Design Center 生成
  - **不要人工编辑这些文件**，因为它们是由 Design Center 更新的
- CapTIvate 应用程序框架文件位于 captivate_app 目录中
  - 为使用该软件库提供了应用级框架示例
  - 电路板支持文件

11.10.1.8 第三部分 – 创建和使用回调函数

CapTIvate 触摸库采用回调函数以实现与用户应用程序的通信。回调函数提供了一种用于在某个传感器被更新时向应用程序发出通知的机制。应用程序必须首先寄存其用于每个传感器的回调函数，然后才能接收更新。当寄存一个应用程序的回调函数时，每次对应的传感器被扫描和处理时都执行回调函数，这与检测的类型（接近或触摸）无关。在回调函数期间，应用程序能够查询传感器的数据结构以确定传感器的状态。下面的插图示出了一个序列示例，在该序列中，先对小键盘进行扫描和处理，接着执行小键盘回调函数，然后把传感器数据和状态信息传送至 CapTIvate Design Center。对于其他的传感器，该序列继续，然后根据编程扫描速率重复进行。

图 233：在传感器处理之后执行回调函数

对于本实验室动手操作，创建了一个在任何小键盘按键被掀按时接通 LED1 的回调函数。库函数 CAPT_registerCallback() 负责提供回调函数注册。

把名为 my_keypad_callback 的应用程序回调函数寄存至一个小键盘传感器采用下面的语法：
MAP_CAPT_registerCallback(&keypad, &my_keypad_callback);

LAB3 行动

在 LAB3 代码项目中，把上述的函数调用刚好插入在函数调用 CAPT_appStart() 之前（在主程序中的第 80 行附近）。这使得该回调函数能够在软件库启动之前获得注册。

注： 由 Design Center 为本项目创建的传感器名称位于 CAPT_UserConfig.h 文件中，这里为方便起见予以示出。该列表中的第一个传感器是小键盘传感器。

```
101 //
102 // Captivate Sensor Prototypes
103 // These prototypes enable access to sensors
104 // from the application.
105 //
106 extern tSensor keypad;
107 extern tSensor prox;
108 extern tSensor slider;
109 extern tSensor wheel;
110
```

图 234：传感器代码

接着，在回调函数的内部，应用程序能够对传感器数据和状态实施任何处理，如检查物体接近、触摸检测、或者滑块或滚轮位置。当小键盘中的任何按键被触摸和释放时，下面的示例可加以检测，并触发一个 LED。

LAB3 行动

把下面的回调函数代码片段添加至 main() 之前的某个地方：

```
void my_keypad_callback(tSensor* pSensor)
{
    if ((pSensor->bSensorTouch == true) && (pSensor->bSensorPrevTouch == false))
    {
        LED2_ON;
    }
    else if ((pSensor->bSensorTouch == false) && (pSensor->bSensorPrevTouch == true))
    {
        LED2_OFF;
    }
}
```

图 235：回调函数代码

在回调函数中，只有当检测到触摸时 LED2 才接通，而当检测到释放时则 LED2 关闭。这最大限度地缩短了 CPU 花费在回调函数中的时间。如欲允许回调函数控制 LED2，则对 main() 中以下代码行外的原始代码注释做一个额外的变更。

LAB3 行动

如这里所示修改 main() 的部分·

```
/*
LED1_ON;
if(CAPT_appHandler()==true)
    LED2_ON;
else
    LED2_OFF;
LED1_OFF;
*/
CAPT_appHandler();
```

图 236：修改主程序

点击调试器图标以构建代码并将代码下载到目标。点击播放图标以启动演示。当小键盘按键被触摸时 LED2 接通。

11.10.1.9 第四部分 – 确定传感器状态和位置

滑块和滚轮是多元件传感器，通常具有三个或更多的电极。像按键一样，滑块和滚轮也会具有接近和触摸状态，但是它们还具有一个位置。位置是依靠个别元件的测量来确定的。CapTIvate 软件库算法根据所有传感器元件提供的测量值来确定传感器位置（手指正在触摸的地方），并在传感器结构中提供结果。

本节说明如何创建一个用于滑块的回调函数，该函数在滑块位置大于 500 时接通 LED1，并在滑块位置小于或等于 500 时关闭 LED1。之所以选择 500，是因为它位于滑块的中间（依据 LAB1，把滑块分辨率设定为 1000）。

把名为 my_slider_callback 的应用程序回调函数寄存至一个滑块传感器采用下面的语法：
MAP_CAPT_registerCallback(&slider, &my_slider_callback);

LAB3 行动

在 LAB3 代码项目中，把 MAP_CAPT_registerCallback() 函数调用插入在 main() 中的函数调用 CAPT_appStart() 之前。

下面的回调函数示例读取传感器数据结构，并根据滑块位置的数值来接通和关闭 LED1。

LAB3 行动

把下面的回调函数代码添加在 main() 之前：

```
void my_slider_callback(tSensor* pSensor)
{
    tSliderSensorParams *pSliderParams;

    if (pSensor->bSensorTouch == true)
    {
        pSliderParams = (tSliderSensorParams*)(pSensor->pSensorParams);
        if (pSliderParams->SliderPosition.ui16Natural > 500)
        {
            LED1_ON;
        }
        else
        {
            LED1_OFF;
        }
    }
}
```

图 237：my_slider_callback

点击调试器图标以构建代码并将代码下载到目标。点击运行图标以启动演示运行。

当在 Design Center 中查看滑块时，LED1 在滑块位置大于 500 时接通，并在滑块位置低于 500 时关闭。如果您在滑块位置大于 500 时把手指从滑块移开，则 LED1 保持导通，因为代码程序仅在滑块被揿按时才改变 LED 的状态。

对于该回调函数的以下修改将在滑块不处于被触摸的状态时关闭 LED1。

```
void my_slider_callback(tSensor* pSensor)
{
    tSliderSensorParams *pSliderParams;

    if (pSensor->bSensorTouch == true)
    {
        pSliderParams = (tSliderSensorParams*)(pSensor->pSensorParams);
        if (pSliderParams->SliderPosition.ui16Natural > 500)
        {
            LED1_ON;
        }
        else
        {
            LED1_OFF;
        }
    }
    else
    {
        LED1_OFF;
    }
}
```

图 238：对 my_slider_callback 所做的更改

祝贺您！这是 CapTIvate 讲习班的结束。

有关 TI 设计信息和资源的重要通知

德州仪器 (TI) 公司提供的技术、应用或其他设计建议、服务或信息，包括但不限于与评估模块有关的参考设计和材料（总称“TI 资源”），旨在帮助设计人员开发整合了 TI 产品的 应用； 如果您（个人，或如果是代表贵公司，则为贵公司）以任何方式下载、访问或使用了任何特定的 TI 资源，即表示贵方同意仅为该等目标，按照本通知的条款进行使用。

TI 所提供的 TI 资源，并未扩大或以其他方式修改 TI 对 TI 产品的公开适用的质保及质保免责声明；也未导致 TI 承担任何额外的义务或责任。TI 有权对其 TI 资源进行纠正、增强、改进和其他修改。

您理解并同意，在设计应用时应自行实施独立的分析、评价和 判断， 且应全权负责并确保 应用的安全性， 以及您的 应用 （包括应用中使用的所有 TI 产品）） 应符合所有适用的法律法规及其他相关要求。你就您的 应用声明，您具备制订和实施下列保障措施所需的一切必要专业知识，能够 (1) 预见故障的危险后果，(2) 监视故障及其后果，以及 (3) 降低可能导致危险的故障几率并采取适当措施。您同意，在使用或分发包含 TI 产品的任何 应用前， 您将彻底测试该等 应用 和该等应用所用 TI 产品的 功能。除特定 TI 资源的公开文档中明确列出的测试外，TI 未进行任何其他测试。

您只有在为开发包含该等 TI 资源所列 TI 产品的 应用时， 才被授权使用、复制和修改任何相关单项 TI 资源。但并未依据禁止反言原则或其他法理授予您任何TI知识产权的任何其他明示或默示的许可，也未授予您 TI 或第三方的任何技术或知识产权的许可，该等产权包括但不限于任何专利权、版权、屏蔽作品权或与使用TI产品或服务的任何整合、机器制作、流程相关的其他知识产权。涉及或参考了第三方产品或服务的信息不构成使用此类产品或服务的许可或与其相关的保证或认可。使用 TI 资源可能需要您向第三方获得对该等第三方专利或其他知识产权的许可。

TI 资源系“按原样”提供。TI 兹免除对 TI 资源及其使用作出所有其他明确或默认的保证或陈述，包括但不限于对准确性或完整性、产权保证、无屡发故障保证，以及适销性、适合特定用途和不侵犯任何第三方知识产权的任何默认保证。

TI 不负责任何申索，包括但不限于因组合产品所致或与之有关的申索，也不为您辩护或赔偿，即使该等产品组合已列于 TI 资源或其他地方。对因 TI 资源或其使用引起或与之有关的任何实际的、直接的、特殊的、附带的、间接的、惩罚性的、偶发的、从属或惩戒性损害赔偿，不管 TI 是否获悉可能会产生上述损害赔偿，TI 概不负责。

您同意向 TI 及其代表全额赔偿因您不遵守本通知条款和条件而引起的任何损害、费用、损失和/或责任。

本通知适用于 TI 资源。另有其他条款适用于某些类型的材料、TI 产品和服务的使用和采购。这些条款包括但不限于适用于 TI 的半导体产品 (http://www.ti.com/sc/docs/stdterms.htm)、评估模块和样品 (http://www.ti.com/sc/docs/sampterms.htm) 的标准条款。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122