Freescale Semiconductor
应用笔记

Document Number: AN4637
Rev 0, 11/2012

# 使用 GPIO 方法在 Kinetis L 上集成触摸感应软件（TSS 3.0.1）

作者: Xianhu Gao

内容



## 1 简介

触摸感应软件（TSS）解决方案可将标准飞思卡尔微控制器转换为近距离电容触摸传感器控制器。在 TSS 3.0 和更高版本中，触摸感应库实现了电容式感应，不仅针对整个 Freescale S08、ColdFire V1 系列和 ARM®Cortex™-M4 Kinetis，还针对 ARM Cortex-M0+ Kinetis L 系列微控制器。

在 TSS 3.0.1 库中，我们提供名为 FRDMKL25Z_DEMO 的演示，使用基于 FRDM-KL25Z 板的 TSI 方法。FRDM-KL25Z 是一种超低成本的开发平台，通过基于 ARM Cortex-M0+处理器构建的 Kinetis L 系列 KL1 和 KL2 MCU 系列实现。

在此应用笔记中，多次使用了 FRDMKL25Z_DEMO，本文使用 GPIO 方法实现触控感应，而非 TSI 方法。这是为那些没有 TSI 模块的 MCU（例如 S08 或 KL02）集成 TSS 的很好示例。

### 1.1 Kinetis L 系列 Freedom 开发平台

FRDM-KL25Z 是一种超低成本的开发平台，通过基于 ARM Cortex-M0+处理器构建的 Kinetis L 系列 KL1 和 KL2 MCU 系列实现。FRDM-KL25Z 的特性包括轻松访问 MCU I/O、备用电池、低功耗工作、标准外形尺寸，它还

© 2012 Freescale Semiconductor, Inc.

提供扩展板选项和内置调试接口，用于 flash 编程和运行控制。一系列飞思卡尔和第三方开发软件都支持 FRDM-KL25Z。

有关 FRDM-KL25Z 的更多信息可从以下位置获取：**freescale.com/FRDM-KL25Z**。

## 1.2 触摸感应软件 3.0.1

触摸感应软件（TSS）解决方案可将标准飞思卡尔微控制器转换为近距离电容式触摸传感器控制器。与早期版本相比，TSS 3.0 和更高版本在各个部分中添加了新文本，包括“信号标准化”、“自动灵敏度校准”、“基线初始化”、“电极组”、“模拟方式实现滑条和旋钮解码 API”、“读取控件中的即时增量”、“接近函数”、“自动灵敏度校准”、“屏蔽功能和耐水性”、“耐水模式”、“模拟方式旋钮”、“模拟方式滑条”和“(按键）矩阵”。(查看 TSSAPIRM 的修订历史记录：TSSAPIRM，触摸感应软件 API 参考手册，可从 **freescale.com** 获取。)

最新 TSS 可在以下位置找到：**freescale.com/TSS**。

## 1.3 模拟方式滑条

在此应用笔记中，我们使用模拟方式滑条来控制 LED 的亮度。模拟方式滑条控制的工作方式与标准滑块相同，但电极较少，计算位置的分辨率更高。例如，两电极模拟方式滑条可在范围 128 中提供模拟位置。电极的形状必须符合在手指运动过程中增加和减小信号的条件，手指运动必须是直线的。图 1 显示了用于典型模拟方式滑条的电极的布置。

**图 1. 电极模拟方式滑条**

# 2 基于 GPIO 的电容触摸感应方法

GPIO 低电平感应方法使用在触摸感应算法 中讨论的 GPIO 电容触摸感应方法来测量电容。

要测量电极的电容，必须使用具有 GPIO 和时间测量功能的飞思卡尔 MCU。

## 2.1 触摸感应算法

连接到 MCU 的电极充当电容器，外部上拉电阻限制为电极充电的电流。图 2 显示了电路示意图。

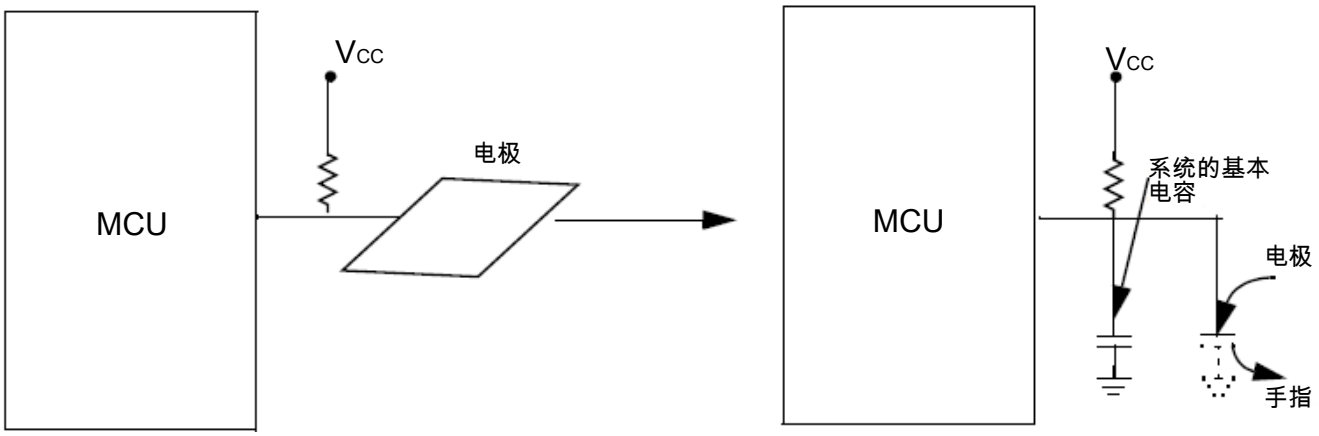


**图 2. 电极等效电路**

RC 电路充电时间常数（τ）按照以下公式定义：

$$\tau = RC$$

根据以上公式，如果上拉电阻保持不变，则电容的增加也会增加电路充电时间。MCU 测量充电时间，并使用此值确定电极是否被触摸。触摸电极时，手指电容将会增加到电极电容。这样会增加电路电容，进而增加定时器测量的充电时间。下图显示了包括手指增加电容的电容器的充电时间。

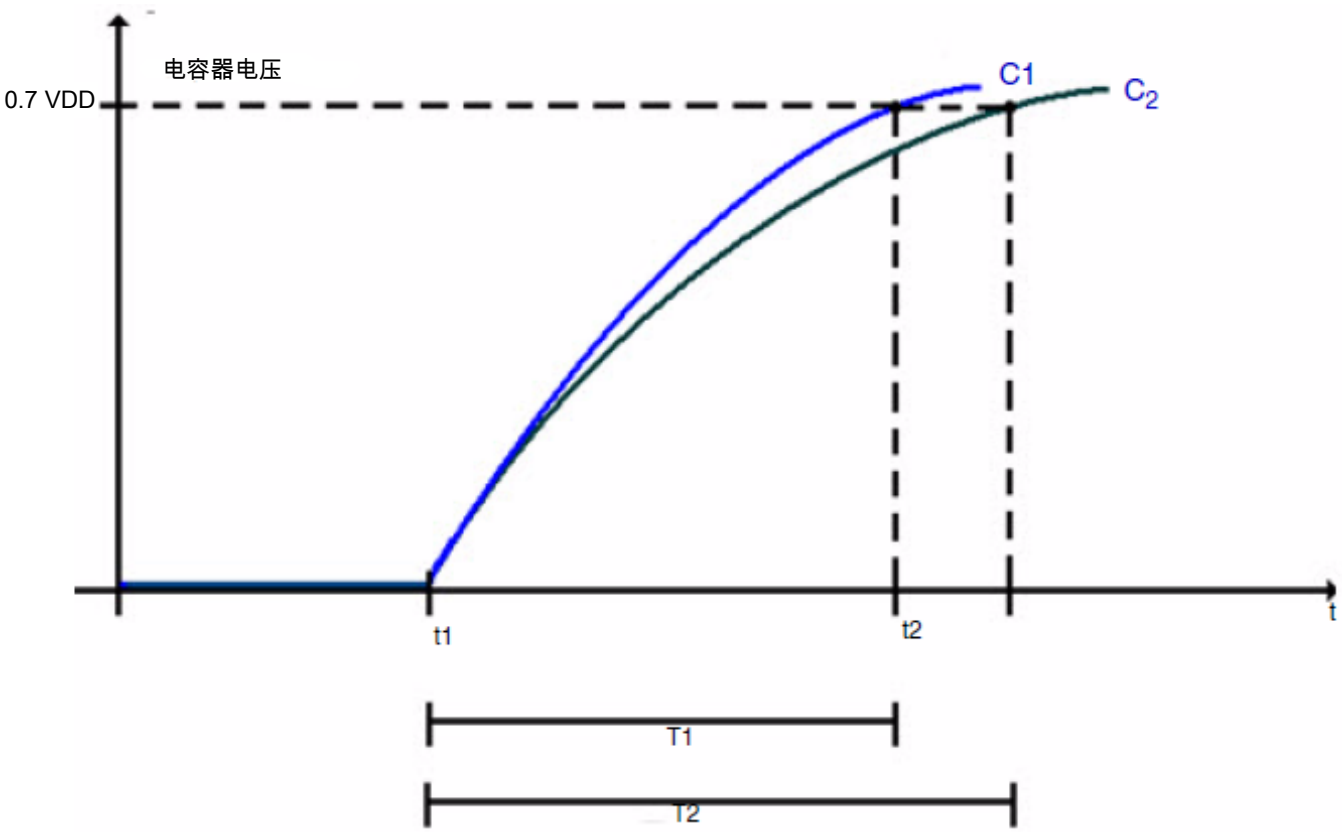


**图 3. 包括手指增加电容的电容器的充电时间**

如图 3 中所示：

- C1 — 没有额外电容时或尚未触摸电极时的充电时间曲线。
- C2 — 已触摸电极时的充电时间曲线。
- T1 — 尚未触摸电极时的充电时间。
- T2 — 已触摸电极时的充电时间。

## 2.2 模拟方式滑条控制 API

在 TSS 库中，存在 TSS_SensorGPIO.c 和 TSS_SensorGPIO.h 文件，它们包含用于执行电极感应和设置每个电极状态的函数。

对于模拟方式滑条，提供了 8 个配置和状态寄存器。请参见表 1 。

**表 1. 控制配置和状态寄存器**

| 寄存器编号 | 大小（字节） | 寄存器名称 | 初始值 | 简要描述 |
|---|---|---|---|---|
| 0x00 | 1 | ControlId | 取决于应用 | R—显示控件类型和控件编号 |
| 0x01 | 1 | ControlConfig | 0x00 | RW—此寄存器配置对象的所有使能器。 |
| 0x02 | 1 | DynamicStatus | 0x00 | R—显示运动、方向和位移信息 |
| 0x03 | 1 | 位置 | 0x00 | R—显示触摸和绝对位置信息。保留无效位置状态标志。 |
| 0x04 | 1 | 事件 | 0x00 | RW—配置调用回调函数的事件 |
| 0x05 | 1 | AutoRepeatRate | 0x00 | RW—配置当按住按键且未检测到运动时，报告按键的速率。 |
| 0x06 | 1 | MovementTimeout | 0x00 | RW—在报告保持事件且没有运动之前，解码器必须达到的未检测到位移的次数。 |
| 0x07 | 1 | 范围 | 0x40 | RW—定义报告位置的绝对范围。 |

但不支持对这些寄存器进行直接操作，否则数据可能被损坏。TSS_SetASliderConfig()和 TSS_GetASliderConfig()是在 TSS_API.h 文件中为用户准备的 2 个 API。

在表 1 中提及的所有寄存器都由以下两个 API 函数读取或写入。

```
UINT8 TSS_SetASliderConfig(TSS_CONTROL_ID u8ControlId,UINT8 u8Parameter,UINT8 u8Value)
UINT8 TSS_GetASliderConfig(TSS_CONTROL_ID u8ControlId,UINT8 u8Parameter);
```

## 2.3 应用中断管理

使用 GPIO 方法时，所有用户中断处理程序必须通过调用 TSS_SET_SAMPLE_INTERRUPTED()宏，向 TSS 库注册。在进行电极测量时，TSS 将中断保持使能状态。电极测量程序有可能会被用户中断函数给打断，导致测量值不准确，需要用户来添加相应的处理机制。

# 3 GPIO 方法的设置

FRDMKL25Z_DEMO 是 TSS3.0 和更高版本的库中专门面向 KL25 Freedom 板的演示，默认的触摸感应方法是 TSI 方法。

下载代码（参见 **freescale.com/FRDM-KL25Z**）之后，可通过触摸模拟方式滑条，控制板上的三个 LED 的亮度。此外，演示中还展示了低功耗和唤醒控制。

但在此应用中，我们重点介绍 GPIO 方法，因此必须挑选和修改一些要点。

## 3.1 电极定义

有两种类型的电极：GPIO 和 TSIx_CHy。由于默认演示配置是针对 TSI 引脚的，因此模拟方式滑条引脚在 TSS_SystemSetup.h 文件中使用以下代码定义：

```
#define TSS_E0_TYPE                 TSI0_CH9
#define TSS_E1_TYPE                 TSI0_CH10
```

但在 GPIO 方法中，我们定义了以下宏：

```
#define TSS_E0_P                    B
#define TSS_E0_B                    16
#define TSS_E1_P                    B
#define TSS_E1_B                    17

#define TSS_E0_TYPE                 GPIO
#define TSS_E1_TYPE                 GPIO
```

TSS_E0_P 和 TSS_E0_B 指示端口以及此端口中的位。例如，在以上配置中，E0 为 PTB16，E1 为 PTB17。虽然 PTB16 和 PTB17 与 TSI0_CH9 和 TSI0_CH10 使用相同的引脚，但它们现在只是普通的 GPIO，不再处于 TSI 控制模式下。

此外，如果用户需要设置 GPIO 输出驱动强度和压摆率，可以定义以下宏：

```
#define TSS_USE_GPIO_STRENGTH           1
#define TSS_USE_GPIO_SLEW_RATE          1
```

这是从 TSI 方法变换到 GPIO 方法的第一步。

## 3.2 硬件定时器

触摸感应控制的算法基于计数器；基于计数值的大小来判断是否检测到触摸，或者电极上新增的电容有多大。对于 GPIO 方法，必须从片上定时器选择一种硬件定时器：TPMx、FTMx、MTIMx。

在此应用中，我们选择 TPM1 作为定时器，因此在 TSS_SystemSetup.h 文件中定义了以下宏：

```
#define TSS_HW_TIMER                            TPM1
```

但在原始 demo 中，我们使用 TPM1 作为延迟定时器，因此必须修改延迟函数 DelayMS()。在此应用中，延迟函数被删除。

必须针对应用，对定时器预分频进行调整，更改用于测量电容的计时器速度。定时器频率取决于 MCU 总线频率。定时器配置使用预分频值，调整相对于 MCU 总线频率的时间频率。

可以通过改变 TSS_SystemSetup.h 文件中的宏来完成对预分频值的修改：

```
#define TSS_SENSOR_PRESCALER        4    // adjust in different applications
```

有时，应用可能使用非标准电极尺寸，具有特定的电容值。如果电极始终未充电，需要在定时中断函数中完成出错处理，在定时器超时后跳出循环函数。这样可以确保电极触控函数不会阻塞其他应用程序的执行。定时器溢出超时值的调整使用 TSS_SystemSetup.h 文件中的以下宏进行：

```
#define TSS_SENSOR_TIMEOUT        65535 // adjust in different applications
```

## 3.3 外设初始化

TSS 库提供 OnInit Callback 函数。执行 TSS_Init 函数调用时，必须使用此回调。此函数用于初始化外设时钟和设置引脚复用器等。

在此函数中，定义如下所示：

```
#define TSS_ONINIT_CALLBACK        TSS_fOnInit  // in TSS_SystemSetup.h file

/* in events.c file */
void TSS_fOnInit(void)
{
    SIM_MemMapPtr sim = SIM_BASE_PTR;

    /* Modules Clock enablement */
    sim->SCGC5 |= SIM_SCGC5_PORTB_MASK;  /* Port B clock enablement */

    // HW Timer clock must be opened here, or enter hard fault
    sim->SCGC6 |= SIM_SCGC6_TPM1_MASK;

    /* Set Electrodes for GPIO function */
    PORTB_PCR16 = PORT_PCR_MUX(MUX_ALT1);
    PORTB_PCR17 = PORT_PCR_MUX(MUX_ALT1);
 }
```

这对于 TPM1 和 PORTB 非常重要，缺少定义将在运行代码时进入 HardFault 异常中断。

## 3.4 上拉电阻

按照 GPIO 方法算法，上拉电阻是必需的，它可对电极的充电和放电产生很大影响。通常，电阻值越大，分辨率越高，电容测量范围越小。使用正确的电阻值，用户可以修改电容范围，以及电容对适合应用的范围的灵敏度。表 2 显示在从 500 kΩ 至 2 MΩ 的不同上拉电阻值下的最大电容范围和电容分辨率。

**表 2. 在不同上拉电阻值下的最大电容范围和电容分辨率**

| 电压电平 | 上拉电阻（Ω） | 电容范围（最大值）（pF） | 电容分辨率（最小值）（pF） |
|---|---|---|---|
| VDD > 2.3 V | 500 k | 169.44 | 0.6645 |
| VDD > 2.3 V | 680 k | 124.59 | 0.4886 |
| VDD > 2.3 V | 810 k | 104.59 | 0.4102 |
| VDD > 2.3 V | 1 M | 84.72 | 0.3322 |
| VDD > 2.3 V | 1.5 M | 56.48 | 0.2215 |
| VDD > 2.3 V | 2 M | 42.36 | 0.1661 |
| 1.8 V < VDD < 2.3 V | 500 k | 107.53 | 0.4217 |
| 1.8 V < VDD < 2.3 V | 680 k | 79.07 | 0.3101 |
| 1.8 V < VDD < 2.3 V | 810 k | 66.38 | 0.2603 |
| 1.8 V < VDD < 2.3 V | 1 M | 53.77 | 0.2108 |

下一页继续介绍此表...

表 2. 在不同上拉电阻值下的最大电容范围和电容分辨率 (继续)

| 电压电平 | 上拉电阻（Ω） | 电容范围（最大值）（pF） | 电容分辨率（最小值）（pF） |
|---|---|---|---|
| 1.8 V < VDD < 2.3 V | 1.5 M | 35.84 | 0.1406 |
| 1.8 V < VDD < 2.3 V | 2 M | 26.88 | 0.1054 |

在此应用中，在板和电极上的跳线 J9（引脚 8）中的 P3V3 之间，连接了 500 kΩ 电阻。如果 10 kΩ 电阻上拉，充电和放电时间过短，可能检测不到触摸；如果 10 MΩ 电阻上拉，动态响应会比较差。因此，必须根据应用来调整适当的上拉电阻。

## 3.5 低功耗功能

在 FRDMKL25Z_DEMO 演示中，我们增加了低功耗功能：如果在大约 8 秒内没有检测到触摸，则系统将进入在 main.h 文件中定义的低功耗模式 LOW_POWER_MODE。如果检测到触摸，则 CPU 将会唤醒，因为 TSI 越界中断可以在所有类型的低功耗模式下唤醒 MCU。

当前的 KL25 芯片存在一个问题：越界中断功能与扫描结束中断密切关联。用户无法单独通过越界中断来唤醒 CPU，因此演示提供了一种解决方法。LPTMR 也用作 LLWU 源。每次发生 LPTMR 中断时，将对 TSI0_GENCS 寄存器进行轮询，“扫描结束”标记将被清除，否则电极扫描将不会执行。

TSS 无法完全管理 MCU 低功耗模式。TSS 只让 TSS 系统和所选的低功耗控制源器件做好准备，以便进入 MCU 的低功耗模式。然后，用户自行启动进入低功耗模式。

以下步骤展示低功耗功能的典型使用示例。

1. 负责低功耗控制和同步的外围模块由包含在 TSS_SystemSetup.h 文件中的 TSS_USE_LOWPOWER_CONTROL_SOURCE 函数定义。

```
#define TSS_USE_LOWPOWER_CONTROL_SOURCE          TSI0
```

2. 用户按照 TSS_SetSystemConfig 定义 LowPowerScanPeriod、LowPowerElectrode 和 LowPowerElectrodeSensitivity 寄存器。

```
TSS_SetSystemConfig(System_LowPowerScanPeriod_Register, 0x08);
TSS_SetSystemConfig(System_LowPowerElectrode_Register, 29u);
TSS_SetSystemConfig(System_LowPowerElectrodeSensitivity_Register, 0x1A);
```

3. 如果用户希望进入 MCU 低功耗模式，则必须由 TSS_SetSystemConfig 函数使能系统配置寄存器中的低功耗使能位。

```
(void)TSS_SetSystemConfig(System_SystemConfig_Register,(TSS_SYSTEM_EN_MASK |
TSS_DC_TRACKER_EN_MASK | TSS_LOWPOWER_EN_MASK));
```

4. 用户现在可以通过与使用的 MCU 平台相关的指令，强制 MCU 进入低功耗模式。

```
LowPowerControl();    // in the main loop in main.c
```

5. 如果所选的低功耗控制源器件检测到触摸，则 MCU 将会唤醒，程序继续运行。低功耗使能位被自动禁用。

但在此应用中，由于选择了 GPIO 方法，TSI 模块完全没有使用，因而必须删除 TSI 相关配置。因此，必须删除原始演示代码中的以下宏：

```
/* in isr.h */
  #undef  VECTOR_042
  #define VECTOR_042 TSS_TSI0Isr

/* in TSS_SystemSetup.h */
  #define TSS_USE_LOWPOWER_CONTROL_SOURCE    TSI0

/* TSI Autocalibration Settings */
  #define TSS_TSI_RESOLUTION                 6
  #define TSS_TSI_EXTCHRG_LOW_LIMIT          1
```

```
  #define TSS_TSI_EXTCHRG_HIGH_LIMIT          7
  #define TSS_TSI_PS_LOW_LIMIT                0
  #define TSS_TSI_PS_HIGH_LIMIT               7

/* Active Mode Clock Settings */
#define TSS_TSI_AMCLKS                      0
  #define TSS_TSI_AMPSC                       0
  #define TSS_TSI_AMCLKDIV                    1

/* Low Power TSI definition */
  #define TSS_TSI_LPCLKS                      0

/* in TSS_fOnInit(void) in events.c file */
  sim->SCGC5 |= SIM_SCGC5_TSI_MASK;
/* Set Electrodes for TSI function */
  PORTB_PCR16 = PORT_PCR_MUX(MUX_ALT0);
  PORTB_PCR17 = PORT_PCR_MUX(MUX_ALT0);
```

# 4 FreeMASTER

FreeMASTER 软件提供了 MCU 和 PC 之间通信的驱动。通过 FreeMaster 软件，可以对 TSS 库中使用到内部变量进行图形化的观察和配置。此工具还让用户能够观察信号行为、调节灵敏度并设置 TSS 系统控制寄存器。此应用程序的 GUI 已在 TSS 3.0\examples\FRDMKL25Z_DEMO\gui 中生成和保存。以下步骤讨论 GUI 的使用。

1. 首先，打开 FRDMKL25Z_DEMO\gui 下的 GUI 文件并建立连接。选择“Project（项目）”>“Options（选项）”并选择“Comm（通信）”选项卡。将端口设置为计算机上的虚拟串行端口，并将波特率设置为 115,200。参见图 4 作为示例。

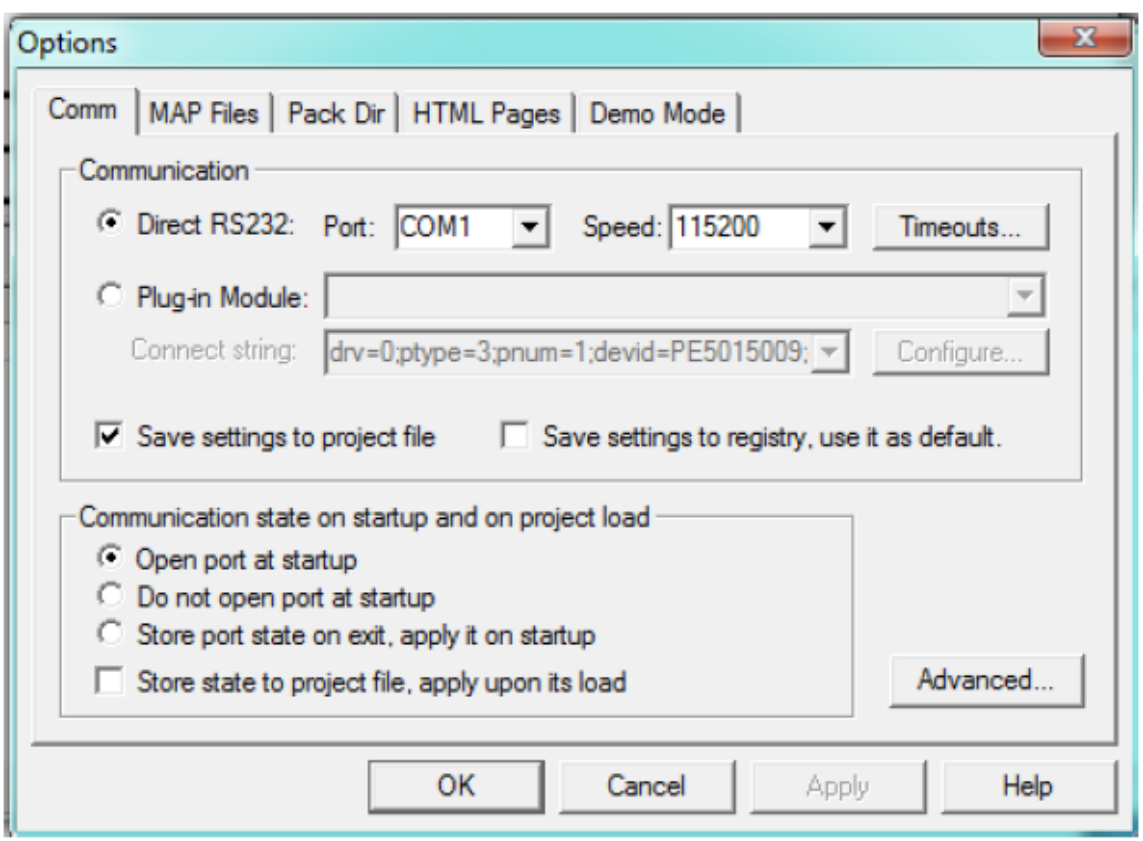


**图 4. 建立通信**

2. 然后，选择“File（文件）”>“Start Communication（开始通信）”。GUI 可通过串行端口从 MCU 读取数据，用户可以检查系统配置和电极信号变化曲线。图 5 显示用户在代码中设置的系统配置，图 6 显示电极信号数据，图 7 显示控制状态，图 8 显示电极信号数据曲线。

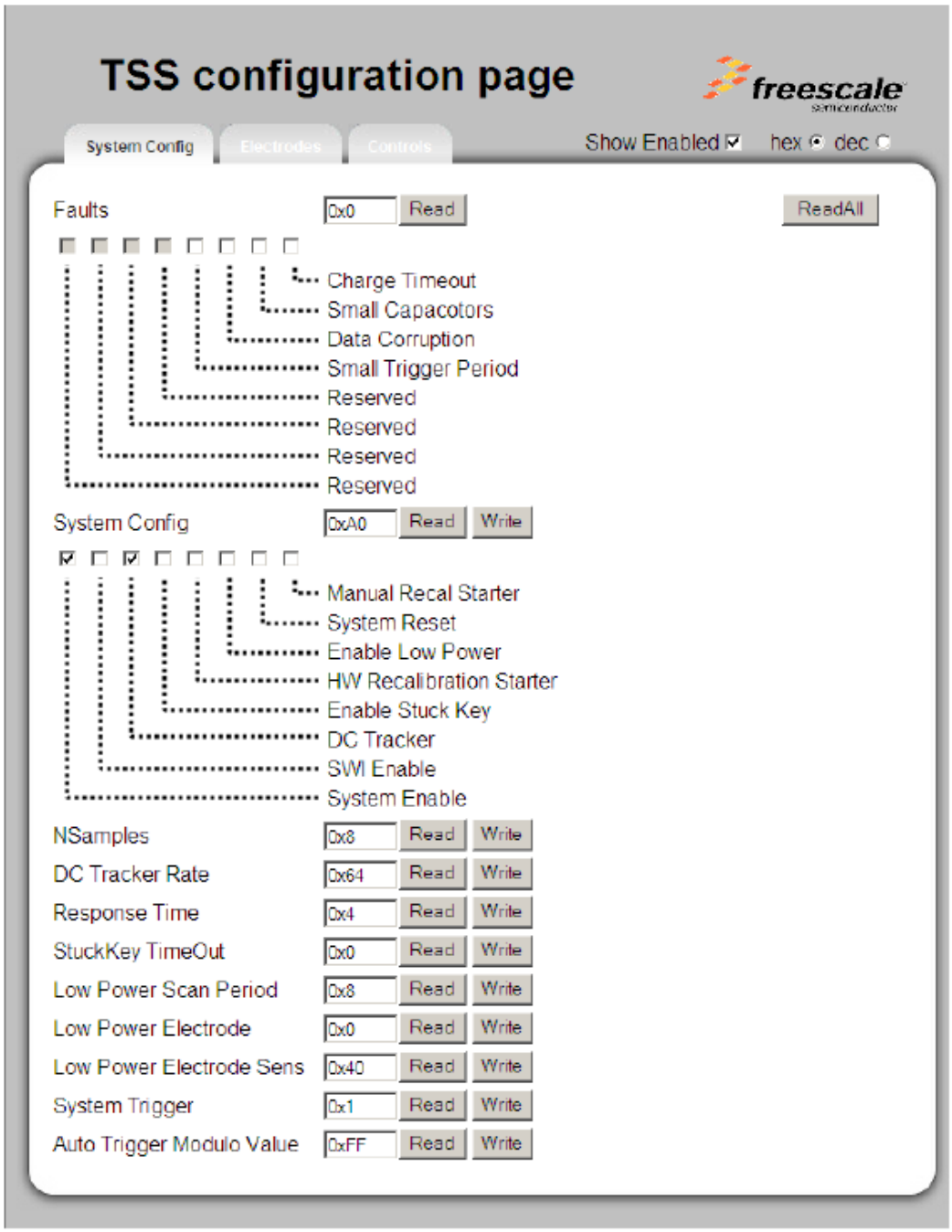


**图 5. 系统配置**

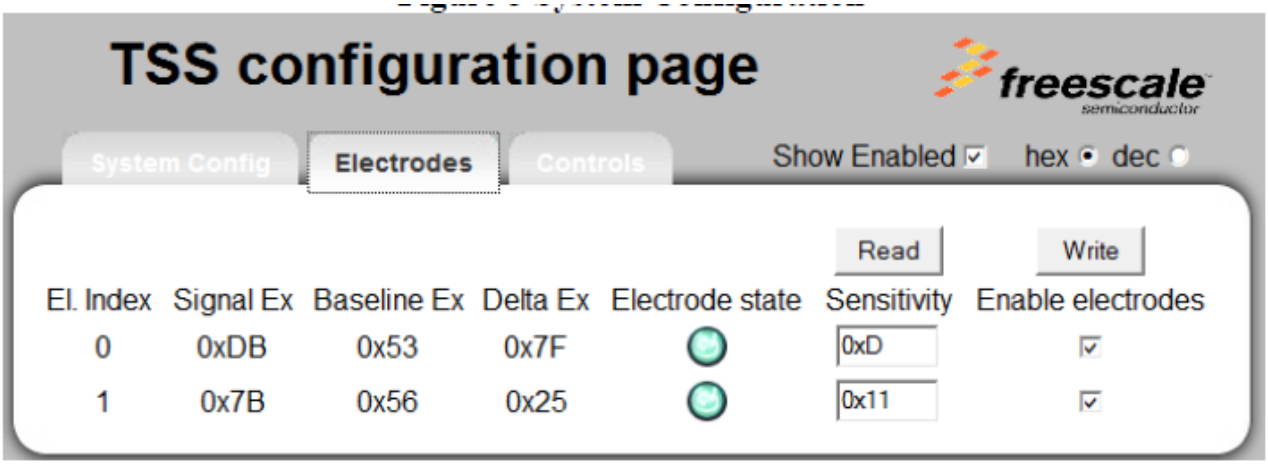


**图 6. 电极数据**

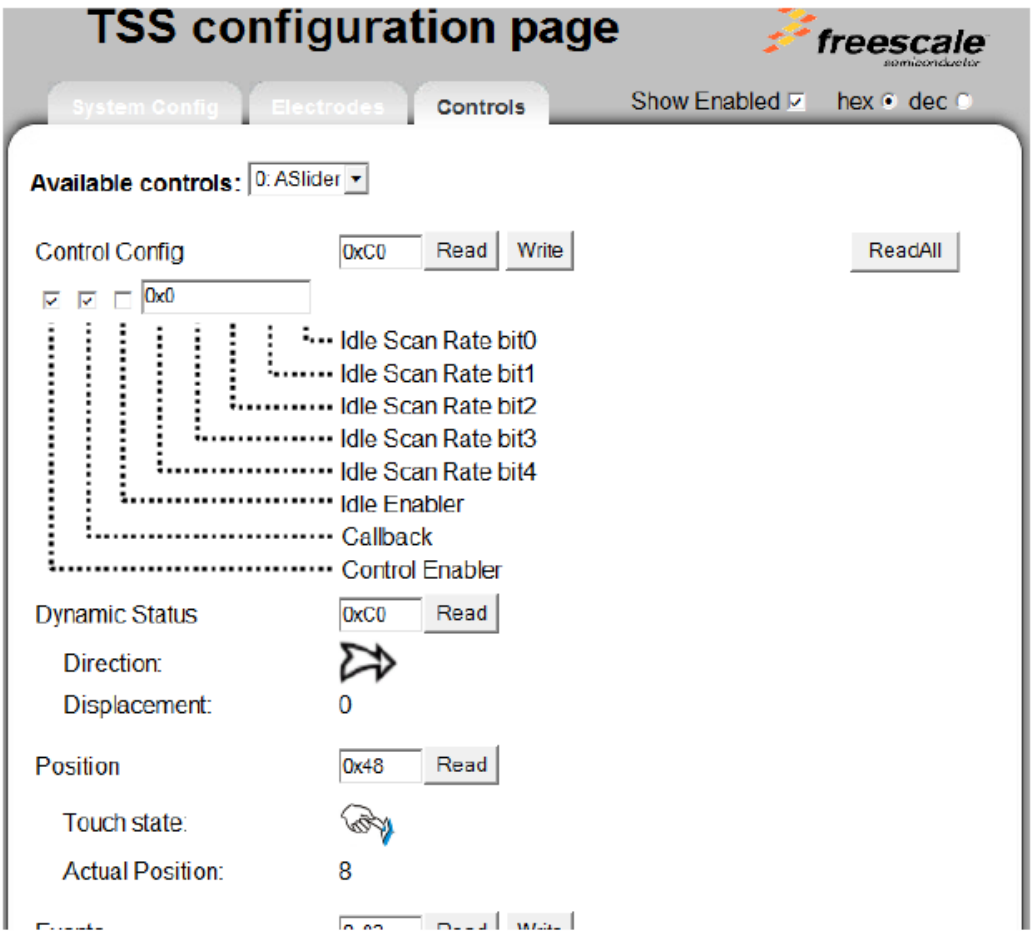


**图 7. 控制状态**

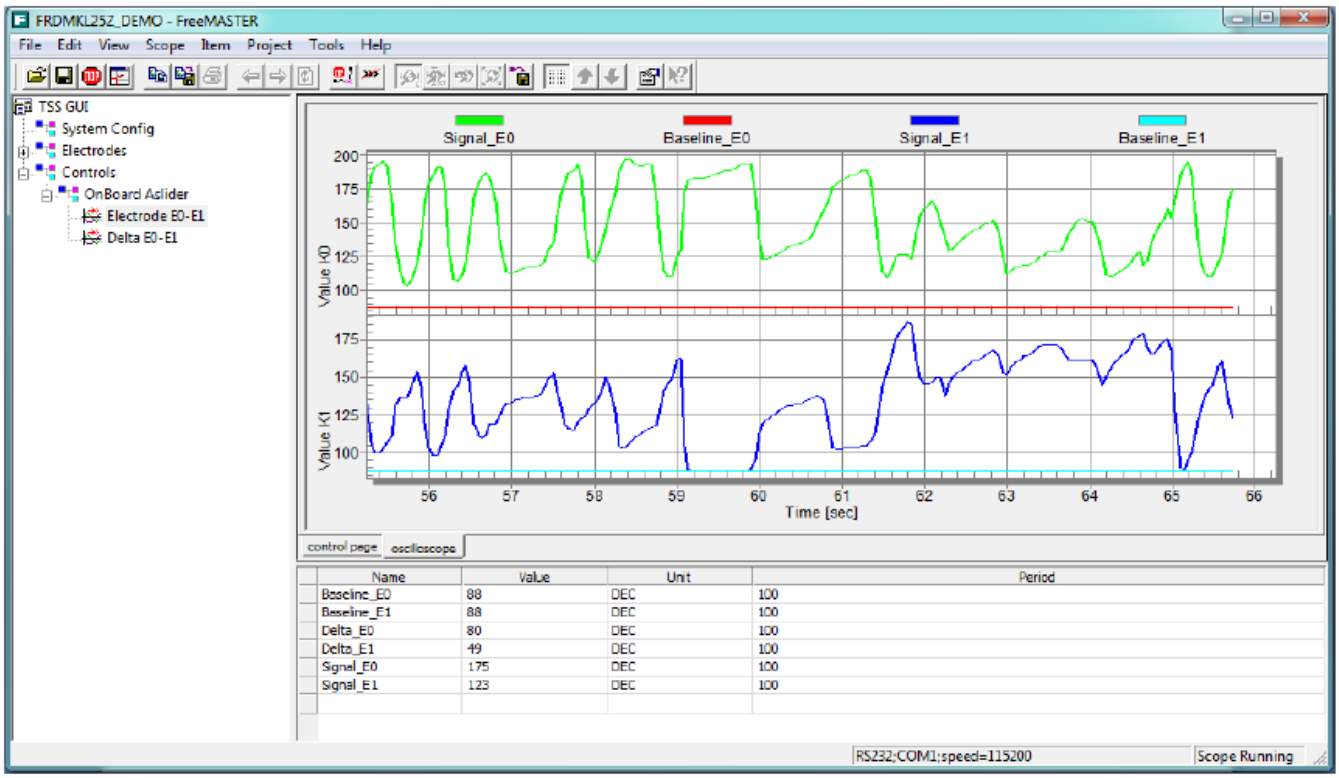


**图 8. 电极数据曲线**

# 5 结论

此应用基于 KL25 演示；差异在于触摸感应方法，原始演示基于 TSI 模块，而此应用基于 GPIO。它为用户提供了很好的例子，帮助他们将 TSS 库集成到现有项目中，或者使用 GPIO 方法修改现有 TSS 项目。使用 GPIO 作为触摸感应电极只需要 1 个空闲 I/O 和 1 个硬件定时器。这对于没有 TSI 模块的芯片（例如 KL02 和 S08）非常有用，还有利于降低成本。此应用已在 KL25 和 KL02 Freedom 板上成功执行，其结果与 TSI 方法相同。

# 6 参考

可从 **freescale.com** 获取以下文档，以备进一步参考。

- TSSUG：TSSUG，触摸感应软件用户指南（修订版 5）
- TSSAPIRM：TSSAPIRM，触摸感应软件 API 参考手册（修订版 7）
- FRDM-KL25Z：Kinetis L 系列 Freedom 开发平台

***How to Reach Us:***

**Home Page:**
freescale.com

**Web Support:**
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。 本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。 Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。 Freescale 的数据表和/或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。 所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。 Freescale 未转让与其专利权及其他权利相关的许可。 Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, and Kinetis, are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2012 Freescale Semiconductor, Inc.

© 2012 飞思卡尔半导体有限公司

Document Number AN4637
Revision 0, 11/2012