电容式触摸软件库

MSP430™ 微控制器提供一定数量的外设，通过适当配置，这些外设可被用来执行一个电容值测量。 电容式触摸软件库的用途是为了创建一个可与 2xx 和 5xx MSP430 系列集成的单独接口并且外设在这些系列产品内进行设定。 本文档对电容式触摸软件库的配置和使用进行了解释说明。

本文档中描述的软件库可从以下网站下载http://focus.ti.com/docs/toolsw/folders/print/capsenselibrary.html。

Topic **Page**

1 简介 2
2 执行 2
3 配置 5
4 资源 25
5 API 调用 27
6 建立测量参数 33
附录 A 元件和传感器定义 37
附录 B 电容触摸传感器层详细说明 43
附录 C Beta 测试 55

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

1 简介

电容式触摸软件库是一款灵活的软件基础，此软件基础可在 MSP430 微控制器平台上快速启用几个不同电容式触摸感测算法中的一个。 虽然每个算法有其唯一的特性，通常有一定数量的应用专用因子，这些因子将使得一个算法实现更加容易地适应另外一个算法实现的需要。 这个软件库的一个主要用途就是提供与几个 MSP430 系列的接口以及那些系列产品内的不同外设集。

软件库提供几个层或者抽象程度。 抽象的更高级在较低级允许个性化定制和独特控制的同时为更快速且更简便开发提供标准控制。

为了使用这个软件库，对于测量方法有一个基本的了解十分重要，其中还包括如何为一个特定的方法配置此软件库，如何使用外设资源，以及应用程序接口 (API) 函数调用。

相关代码 [1] 被用作开发电容式触摸和其它电容式测量解决方案开发的出发点。 特性集可适应多种应用程序的需要，一个特定的应用也许并不需要所有这些应用程序。 提供了源代码之后，鼓励用户创建一个运行中的应用程序来删除代码中未使用的部分。 此外，由于对 ISR 的定义有可能与一个现有应用定义发生冲突，低功耗特性通常不在软件库中使用。 而且，由于应用允许 ISR 功能共用，鼓励用户升级所提供的源代码来产生 MSP430 的大多数低功耗功能。

2 执行

对于本文档中描述的软件执行，主要原则是比较两个独立的定时域。 一个定时域被固定，而另外一个作为一个电容值函数的变量。

2.1 张弛振荡器

张弛振荡器方法计算在一个固定周期（选通时间）内的张弛振荡器周期的数量，如图 1所示。

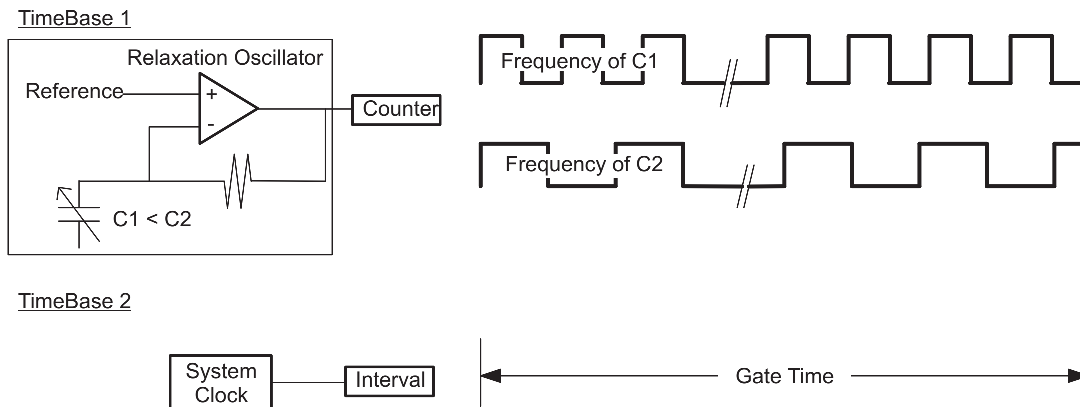

图 **1.** 张弛振荡器计量

张弛振荡器可由几个 MSP430 内的比较器或者 PinOSC 功能实现。 振荡频率是电路的电阻值和电容值的函数。 电容值为预期变量并且会随着触摸增加。 在时域内上升和下降时间增加，而在频域内频率减少。 随着电容值的增加，张弛振荡器周期的数量会在固定的选通时间内减少。

[1] 本文档中描述的软件库可从以下网站下载http://focus.ti.com/docs/toolsw/folders/print/capsenselibrary.html。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

软件库中针对 RO 方法的命名规范标明了张弛振荡器的机制、用于测量和计算振荡的定时器、以及用于定义选通周期的定时器（请见表 1）。

表 1. 张弛振荡器命名规范

名称	RO 机制	计数器	选通周期
RO_XXX_YYY_ZZZ	XXX	YYY	ZZZ
RO_COMPAp_TA0_WDTp	比较器 A+	Timer_A0	安全装置定时器（间隔模式）
RO_COMPB_TA1_WDTA	比较器 B	Timer_A1	安全装置定时器（间隔模式）
RO_Pinosc_TA0[1]	引脚振荡器	Timer_A0	'n' 个 ACLK 周期

[1] RO_PINOSC_TA0 是一个利用了 ACLK 和 Timer_A0 捕捉输入之间的内部连接的特殊情况。 用户可以选择在应用层将 ACLK 简单分频（2，4，8 分频并且将 n 设置为 1）或者输入一个 ACLK 周期数，或者上述两种操作。

2.2 阻容时间常数测量 (RC)

RC 方法是 RO 方法的互逆方法。 如图 2所示，现在选通时间为变量，电容值的一个函数，并且被计数的振荡器是固定的。

TimeBase 1

TimeBase 2

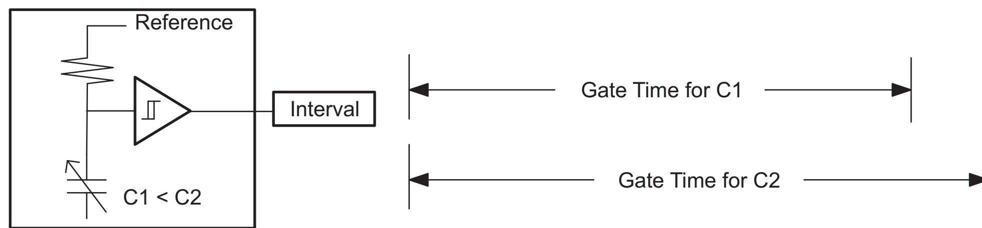

图 2. 阻容时间常数测量 (RC)

任一 MSP430 器件均可实现 RC 方法。 选通时间被定义为将电容充放电至端口 V_{IT+}和 V_{IT-}电平所需的时间。在这个变量选通时间内，计算固定振荡器周期的数量。 电容值的增加（发生了一个触摸）将导致选通时间的增加并因此增加被计数的周期数量。 由于只有对固定振荡进行计数的定时器需要被标识：RC_PAIR_TA0，命名规则比较简单。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

2.3 快速扫描张弛振荡器 (fRO)

除了变量选通周期由一个张弛振荡器创建而非充放电时间，fRO 方法与 RO 方法相似。 如图 3所示，被计数的振荡器频率仍然是固定的。

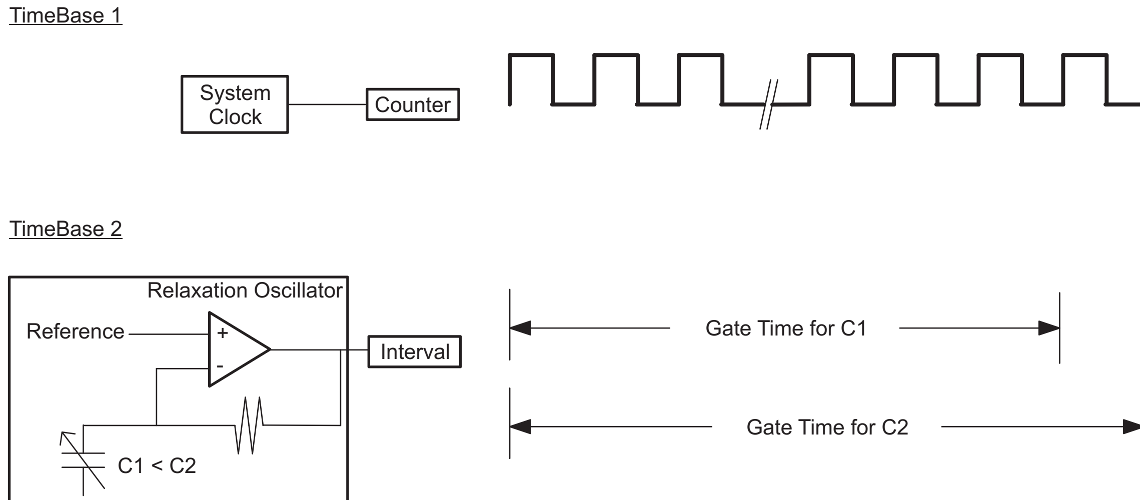

图 3. 快速扫描 RO 测量

此软件库中的 fRO 方法的命名规则标明了张弛振荡器机制、计算固定振荡周期的定时器、用于定义选通周期的定时器（作为一个可变张弛振荡器频率的函数）。

表 2. 张弛振荡器命名规则

名称	RO 机制	计数器	选通周期
fRO_XXX_YYY_ZZZ	XXX	YYY	ZZZ
fRO_COMPB_TA1_SW	比较器 B	Timer_A1	软件回路
fRO_PINOSC_TA0_SW	引脚振荡器	Timer_A0	软件回路

如名称所示，fRO 方法的用途是为了提供快速扫描速率；在灵敏度（灵敏度表示可以解决多小的电容变化）相似时具有比 RO 方法更快的速率。 在 RO 方法中，敏感度是选通周期的函数-选通周期的增加会增加灵敏度。 增加选通时间的负面影响就是减少的扫描速率：在一个单一测量期间将花费更多的时间。 为了保持 fRO 方法的灵敏度，固定时钟速率必须为选通时间内指定的变化提供足够的分辨率。 这通常意味着更短选通时间内的一个更快速的时钟源和更高的功耗。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

3 配置

有两个文件可作为配置软件库的主要方法：structure.c 和 structure.h 包含了所有元件和传感器（元件组）的定义。structure.h 在 structure.c 中生成软件库的其它部分可见的定义，还使用预编译程序定义来启用函数并限制代码大小。

3.1 元件定义

一个电容测量元件是一个线性结构，它的电容代表一个事件：一个触摸、湿度的变化、电介质的变化等。 可单独使用一个元件，诸如作为一个按钮、或者与其它元件组合使用来创建传感器、袖珍键盘、滑轮、或者滑盖，如图 4所示。

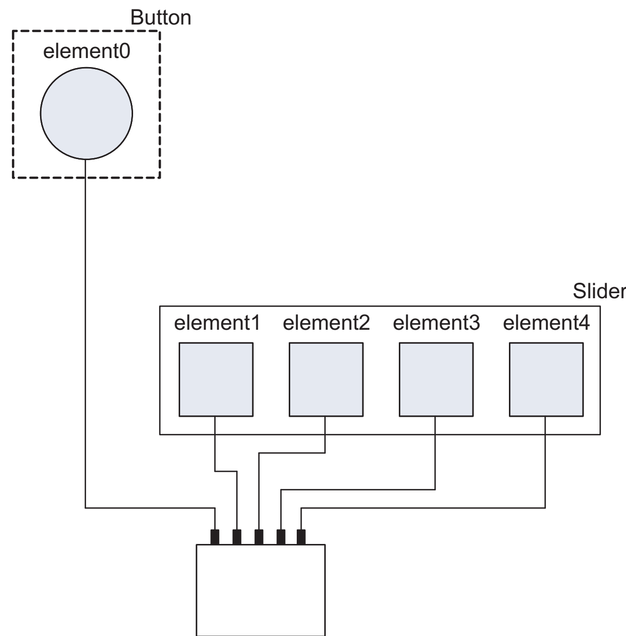

图 4. 元件

元件定义属于两个分类中的一个：端口定义或者测量参数。 端口定义包括数字 IO 外围寄存器、比较器外围寄存器、和位定义。 测量参数包括针对一个给定测量工具的一个元件的阀值和最大信号响应 (maxResponse)。 虽然测量工具要求测试，但端口定义可通过简单读取电路原理图来完成（请见6 节）。 为一个给定测量工具建立正确的测量参数会校准元件。

3.1.1 公用定义

InputBits（输入位）是一个公用定义，这个定义代表 GPIO 定义 Px.y 中的位 y 或者针对 COMPA+ 或者 COMPB 解决方案的比较器输入复用器。

threshold（阀值）定义了一个事件（通常为一个触摸发生时）被声明前电容变化必须超过的限值或者阀值。

maxResponse（最大响应）是一个传感器内元件最大预期响应并且只用于具有多个元件的传感器中：滑盖、滑轮、和按钮。 [1]。 maxResponse 参数的用途是为了将电容测量标准化为一个百分比，在这里阀值代表 0%，而 maxResponse 代表 100%。 这个百分比用于在多个元件产生阀值交叉时在传感器内识别主元件。

图 5在一个按钮应用中显示测量参数、阀值和 maxResponse 之间的相对关系。 阀值和 maxResponse 变量被限定为无符号 16 位整数（0 至 65535）。 当使用一个多元件抽象（按钮、滑盖、或者滑轮）时，这些值被以下的值进一步限定：maxResponse - 阀值 < 655。6 节提供了建立测量阀值的更多细节。

[1] 此按钮抽象是由两个或者更多元件组成的一个传感器。 此按钮抽象是由一个元件组成的一个传感器。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

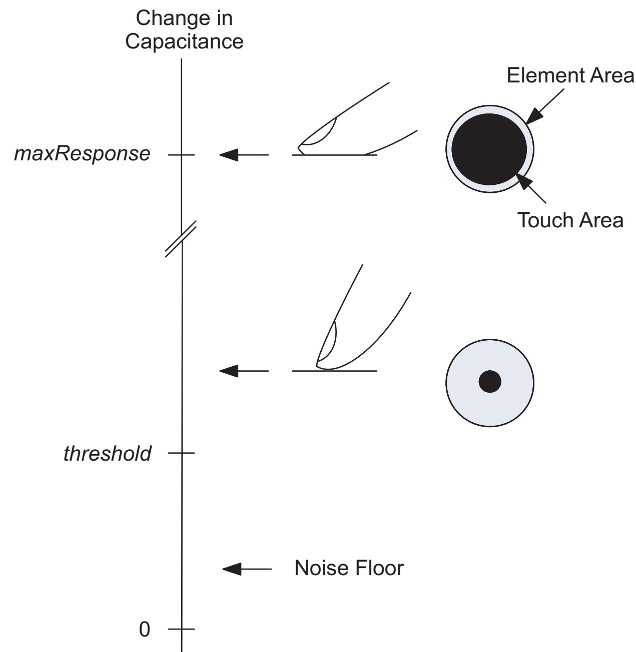

图 **5.** 元件测量参数：按钮示例

3.1.1.1 *Comp_A+* 工具

使用 COMPA+ 外设来创建一个张弛振荡器的工具使用同样的元件结构格式。

*InputBits*标示了 CACTL2 寄存器中的位 P2CA1，P2CA2，和 P2CA3。 这些位代表比较器的负输入。 这个输入被直接连接至电极。 基准输入在传感器部分进行了定义。

```
Const struct Element element_name = {
               .inputBits = P2CA2, // CA2
               .threshold = 100,
               .maxResponse = 200
};
```


版权 © 2011, Texas Instruments Incorporated

3.1.1.2 *Comp_B 工具*

*InputBits*标示了 CBCTL0 寄存器中的 CBIMSEL 位。

15	14	13	12	11	10	9	8
CBIMEN	被保留			CBIMSEL			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CBIPEN	被保留			CBIPSEL			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

CBIMEN	位 15	针对比较器 V- 端子的通道输入启用。 0 为 V- 端子选择的模拟输入通道被禁用。 1 为 V- 端子选择的模拟输入通道被启用。
被保留	位 14-12	被保留
CBIMSEL	位 11-8	如果 CBIMEN 被设定为 1 的话，为比较器 V- 端子选择通道输入。
CBIPEN	位 7	针对比较器 V+ 端子的通道输入启用。 0 为 V+ 端子选择的模拟输入通道被禁用。 1 为 V+ 端子选择的模拟输入通道被启用。
被保留	位 6-4	被保留
CBIPSEL	位 3-0	如果 CBIMEN 被设定为 1 的话，为比较器 V+ 端子选择通道输入。

图 **6. Comp_B** 控制寄存器 **0 (CBCTL0)**

```
const struct Element element_name = {
              .inputBits = CBIMSEL_2, // CB2
              .threshold = 100,
              .maxResponse = 200
 };
```

3.1.1.3 *PinOsc 工具*

**inputPxselRregister*和 **inputPxsel2Register*表示必须为引脚振荡器方法配置的适当寄存器。 与*inputBits*配合使用，这些寄存器可配置此结构。

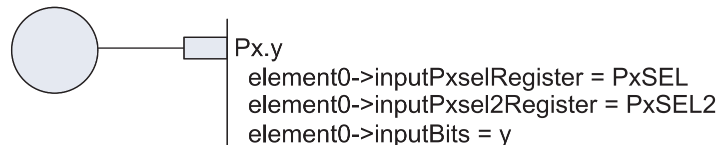

图 **7. PinOsc** 端口参数

```
const struct Element right = {
              .inputPxselRegister = (uint8_t *)&P2SEL,
              .inputPxsel2Register = (uint8_t *)&P2SEL2,

              .inputBits = BIT3,
              .maxResponse = 400,
              .threshold = 50
};
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

3.1.1.4 RC 工具

RC 工具由两个 GPIO 组成。 一个是输入，而另外一个为基准。 这个配置要求用于一个给定端口以及位定义的相关寄存器地址。

inputPxdirRegister，*inputPxoutRegister*，和 *inputPxinRegister* 表示端口方向、输出地址、和输入地址。 与*inputBits*配合使用，这些寄存器配置了结构的输入端口。

referencePxdirRegister 和 *referencePxoutRegister* 表示端口方向和输出地址。 与*referenceBits*配合使用，这些寄存器配置了结构的基准部分。

本说明的一个特性就是端口能够共用两个不同的函数。 也就是说，函数能够“翻转”，这样基准变成输入而输入成为基准来测量连接到基准输入的其它电极。 图 8中显示了这一特性以及以下的代码示例。

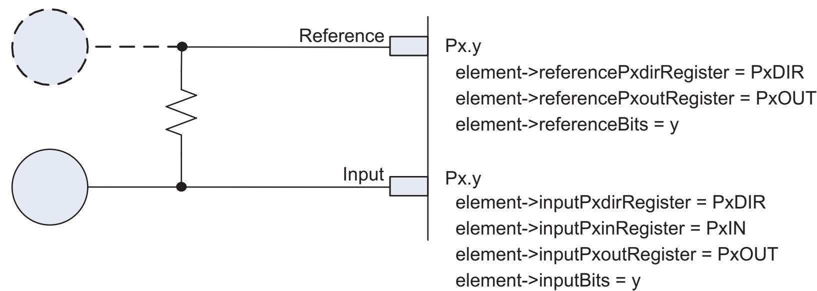

图 8. RC I/O 参数

```
//RC P2.0 input, P2.1 Reference
const struct Element element1 = {

               .inputPxinRegister = (uint8_t *)&P2IN,
               .inputPxoutRegister = (uint8_t *)&P2OUT,
               .inputPxdirRegister = (uint8_t *)&P2DIR,
               .inputBits = BIT0,
               .referencePxoutRegister = (uint8_t *)&P2OUT,
               .referencePxdirRegister = (uint8_t *)&P2DIR,
               .referenceBits = BIT1,
               .threshold = 100,
               .maxResponse = 200
 };
 //RC P2.1 input, P2.0 Reference
const struct Element element2 = {

               .inputPxinRegister = (uint8_t *)&P2IN,
               .inputPxoutRegister = (uint8_t *)&P2OUT,
               .inputPxdirRegister = (uint8_t *)&P2DIR,
               .inputBits = BIT1,
               .referencePxoutRegister = (uint8_t *)&P2OUT,
               .referencePxdirRegister = (uint8_t *)&P2DIR,
               .referenceBits = BIT0,
               .threshold = 120,
               .maxResponse = 250
};
```


版权 © 2011, Texas Instruments Incorporated

3.2 传感器定义

传感器可以是一组独立元件，例如袖珍键盘、也可以是一组运行为一个传感器的元件，例如滑轮或者滑盖。传感器定义包含所有可用元件、用于测量所有元件电容的公用机制、以及针对给定机制的外设地址和位设置。 在滑轮和滑盖情况下，传感器定义还定义了连同滑盖一起的点或者位置的数量（请见图 9）以及传感器的灵敏度。

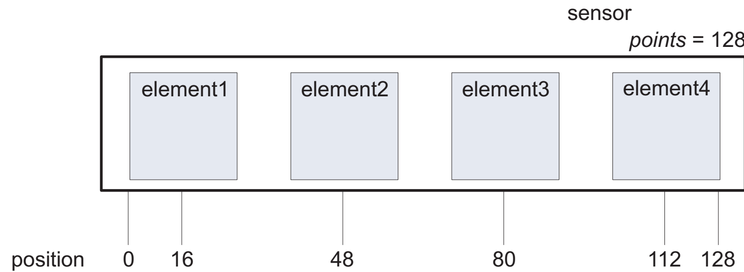

图 **9.** 传感器示例

3.2.1 公用定义

numElements（元件数量）表示了传感器内的元件数量。

*baseOffset*是在应用中被定义的元件数量的累积数量。 对于每个通道，有一个存储在 RAM 中的基线值。

```
//IN structure.h FILE
#define TOTAL_NUMBER_OF_ELEMENTS   8

//IN CTS_Layer.c FILE
uint16_t baseCnt[TOTAL_NUMBER_OF_ELEMENTS];
```

表 **3. baseOffset** 说明

传感器	**baseOffset**	元件	**RAM** 地址
Slider0	0	元件 0	baseCnt[0]
		元件 1	baseCnt[1]
		元件 2	baseCnt[2]
		元件 3	baseCnt[3]
滑盖 1	4	元件 4	baseCnt[4]
		元件 5	baseCnt[5]
		元件 6	baseCnt[6]
		元件 7	baseCnt[7]

*arrayPtr*表示了与一个传感器相关的所有元件。 在滑轮和滑盖情况下，元件的排列顺序十分重要，这是因为这个顺序被认为代表元件的物理顺序。

根据工具的不同，*measGateSource*定义了选通定时起源或者测量时钟源 (halDefinition)。 在 RC 和 fRO 工具中，measGateSource定义了测量时钟源。 在 RO 工具中，measGateSource 表示选通定时器源。

当定时器外设为输入时钟源提供一个输入定时器时，*sourceScale*被用于进一步将定时器源分频。 这一操作只应用于定时器外设，而不应用于安全装置定时器外设。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

*accumulationCycles*定义了对于不同工具的选通时间。 通常情况下，accumulationCycles 代表一个测量被重复的次数，但是在安全装置定时器被用作选通外设的情况下，accumulationCycles 代表安全装置定时器控制寄存器内的位设置。

*halDefinition*表示了传感器使用的测量工具。表 4列出了当前支持的不同工具。

表 4. halDefinition 说明

halDefinition	说明
RO_COMPAp_TA0_WDTp RO_COMPAp_TA1_WDTp	带有 COMPA+ 外设的张弛振荡器实现。 选通时间固定并且由被设定为间隔模式的 WDT+ 外设定义。 电容由 RO 周期的数量表示，此周期数量在固定选通时间内由 Timer_A0/A1 计数得出（请见节 3.2.2.1）。
RO_PINOSC_TA0_WDTp	带有数字 I/O 外设的张弛振荡器实现[1]，Timer_A0 被用来测量振荡器的频率，而 WDT 被用来设定选通时间（请见节 3.2.2.2）。
RO_PINOSC_TA0	带有数字 IO 外设的张弛振荡器实现，Timer_A0 用于测量振荡器频率，而 ACLK 源被用来设定选通时间（请见节 3.2.2.4）。
RO_COMPB_TA0_WDTA RO_COMPB_TA1_WDTA	带有 COMPB 外设的张弛振荡器实现，Timer_A0/A1 被用于测量振荡器频率，而 WDTA 外设被用来设定选通时间（请见节 3.2.2.2）。
RC_PAIR_TA0	用 Timer_A0 来测量 RC 时间常数。 选通时间为变量并随着充电/放电时间的变化而变化。 一个软件循环被用来建立充放电周期的数量，此数量定义了选通时间。 电容由针对选通时间的 Timer_A0 中的定时器计数的数量表示。 通常情况下，为了改进灵敏度，TA0 由一个高频时钟 (SMCLK) 供源。 电容元件按照对中定义的其它 IO 进行充放电（请见节 3.2.3.1）。
fRO_COMPAp_TA0_SW fRO_COMPAp_TA1_SW	带有 COMPA+ 外设的快速扫面张弛振荡器实现。 选通时间是变量并且随着张弛振荡器的周期变化而变化。 Timer_A0/A1 被用来建立振荡次数，而这个次数定义了选通时间。 电容由选通时间内的软件循环计数的数量表示。
fRO_PINOSC_TA0_SW	用 Timer_A0 来计算时间，用数字 IO 外设来执行张弛振荡器，在软件中计数振荡数量来建立选通时间（请见节 3.2.4.2）。
fRO_COMPB_TA0_SW fRO_COMPB_TA1_SW	由 COMPB 外设执行的快速扫描张弛振荡器。 选通时间为变量并随着张弛振荡器的周期变化而变化。 Timer_A0/A1 被用来建立振荡数量，而此数量定义了选通时间。 电容由选通时间内的软件循环计数的数量表示。
fRO_COMPAp_SW_TA0	由 COMPA+ 外设执行的快速扫描张弛振荡器。 此选通时间为变量并且随着张弛振荡器周期的变化而变化。 一个软件循环被用来建立振荡数量，而此数量定义了选通时间。 电容由 Timer_A0 内针对选通时间的定时器计数的数量表示。 为了改进灵敏度，TA0 通常由一个高频时钟 (SMCLK) 供源。

[1] 描述的数字 I/O 功能只在具有引脚振荡器特性的器件上提供。

版权 © 2011, Texas Instruments Incorporated

3.2.2 对于张弛振荡器 (RO) 方法的定义

3.2.2.1 RO_COMPAp_TAx_WDTp

张弛振荡器包含了 Comp_A+ 模块、一个基准、和 RC 滤波器。 此基准被连接至 Comp_A+ 的非反向输入上（通过输入复用器），而 RC 滤波器被连接至反向输入上（也通过一个输入复用器）。 基准是一个分压器，此分压器由一个到 GPIO 的连接和到接地以及比较器输出的连接组成。 RC 滤波器中的 'C' 是被测量的电容元件。

测量电容的一个方法就是将振荡器引至（通过 CAOUT）一个定时器输入 (TAxCLK)。 根据可用的时钟，提供了两个不同的 HAL 定义；RO_COMPAp_TA0_WDTp 和 RO_COMPAp_TA1_WDTp。

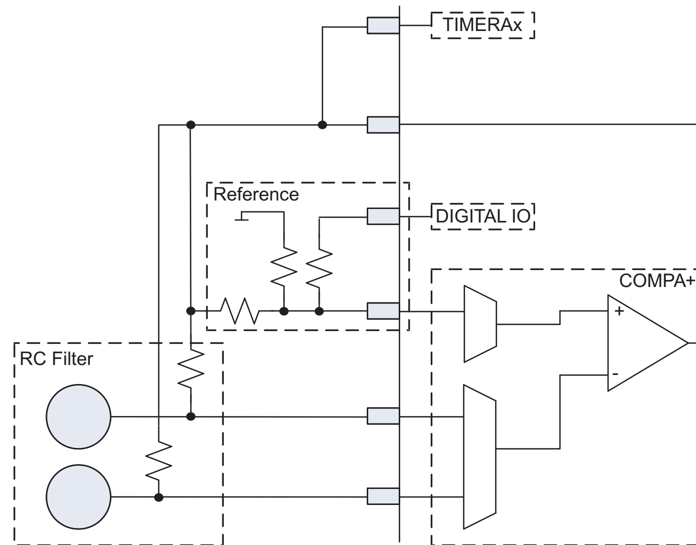

图 10. RO_COMPAp_TAx 电路原理图说明

3.2.2.1.1 端口参数

在元件结构中，为每个元件定义了比较器输入。 在传感器级上（请见图 11），比较器基准输入以及基准端口、比较器输出、和定时器输入被定义。

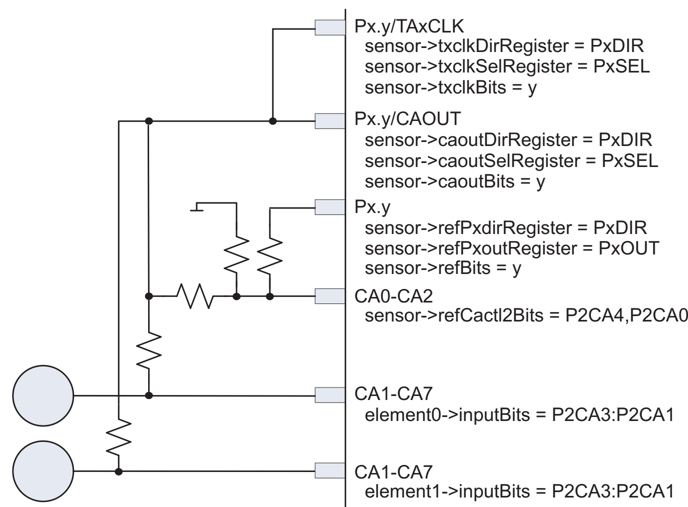

图 11. 针对 RO_COMPAp_TAx 实现的端口参数

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

*caoutDirRegister*和*caoutSetRegister*表示了端口方向地址和端口选择地址。 变量*caoutBits*定义了方向和选择寄存器内将被设定/复位的位来选择端口的 CAOUT 输出功能。 一些器件也使用 PxSEL2 寄存器来定义 CAOUT 使用用例。 在这些器件中，值*caoutSel2Register*也必须被设定（P1SEL2 或者 P2SEL2）。

*txclkDirRegister*和*txclkSelRegister*表示端口方向地址和端口选择地址。 变量*txclkBits*定义了方向和选择寄存器内将被设定/复位的位来选择端口的 TxCLK 输入功能。 一些器件也使用 PxSEL2 寄存器来定义 TxCLK 使用用例。 在这些器件中，值*txclkSel2Register*也必须被设定（P1SEL2 或者 P2SEL2）。

refDirRegister，*refPxOutRegister*，和*refBit*变量定义了图 11中所示外部基准电路的上拉部分。 这些位提供了打开和关闭基准的机制以达到节能的目的。*refPxDirRegister*和*refPxOutRegister*表示了端口方向地址和端口输出地址。 变量*refBits*定义了方向和选择寄存器内将被设定/复位的位以启用基准电路。

*refCact2Bits*表明了 Comp_A+ 的哪一个正输入被连接至电压基准。 基准应只通过 CA0，CA1，或者 CA2 被应用到正输入上。 这分别被表示为 P2CA0，P2CA4，和 P2CA0+P2CA4。

*capBits*定义了组成传感器的 I/O。 这个被应用到 Comp_A+ 控制寄存器 CAPD。 这个值是针对每个输入和基准输入的所有位定义的逻辑 OR 值（即，Px.y 的 y 值而非 CAy 中的 y 值）。

3.2.2.1.2 定时参数

两个定时参数定义了 WDTp 间隔，此间隔为针对 RO_COMPAp_TAx_WDTp 实现的选通时间。

*measureGateSource*表示 WDTp 源：SMCL 或 ACLK。 这个参数与安全装置 Timer+ 寄存器内的“安全装置 timer+ 时钟源选择”位等效。

表 **5.** 安全装置定时器时钟源选择定义

定义	值	源
GATE_WDT_ACLK	0x0004	ACLK
GATE_WDT_SMCLK	0x0000	SMCLK

*accumulationCycles*被用于定义 RO_COMPAp_TAx_WDTp 执行中的 WDTp 间隔。 此参数与安全装置 Timer+ 寄存器内的间隔选择位等效。

表 **6.** 安全装置 **Timer+** 间隔选择定义

定义	值	间隔
WDTp_GATE_32768	0x0000	32768 / 源
WDTp_GATE_8192	0x0001	8192 / 源
WDTp_GATE_512	0x0002	512 / 源
WDTp_GATE_64	0x0003	64 / 源

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

图 12显示了公用传感器参数 measGateSource 和 accumulationCycles 是如何被用来选择选通时间的。

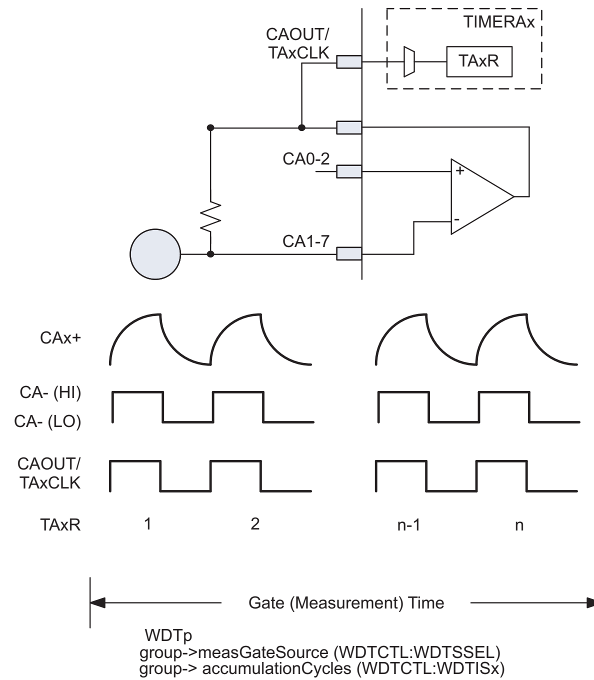

图 **12.** 定时参数：示例 **WDT+**

下面的示例显示了由一个四个元件组成的传感器 每个元件使用 RO 方法在一个 512/MCLK 周期内进行测量。

```
const struct Sensor slider =
{
                   .halDefinition = RO_COMPAp_TA0_WDTp,
                   .numElements = 4,
                   .baseOffset = 0,
                   .points = 80,
                   .sensorThreshold = 50,
                   // Pointer to elements
                   .arrayPtr[0] = &element0, // point to first element
                   .arrayPtr[1] = &element1,
                   .arrayPtr[2] = &element2,
                   .arrayPtr[3] = &element3,

                   // Reference Information
                   // CAOUT is P1.7
                   // TACLK is P1.0
                   .caoutDirRegister = (uint8_t *)&P1DIR, // PxDIR
                   .caoutSelRegister = (uint8_t *)&P1SEL, // PxSEL
                   .txclkDirRegister = (uint8_t *)&P1DIR, // PxDIR
                   .txclkSelRegister = (uint8_t *)&P1SEL, // SxSEL
                   .caoutBits = BIT7, // BITy
                   .txclkBits = BIT0,
                   // Reference is on P1.6
                   .refPxoutRegister = (uint8_t *)&P1OUT,
                   .refPxdirRegister = (uint8_t *)&P1DIR,
                   .refBits = BIT6,         // BIT6
                   .refCactl2Bits = P2CA4, // CACTL2-> P2CA4, CA1
                   .capdBits = (BIT1+BIT2+BIT3+BIT4+BIT5),
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

```
                    // Timer Information
                    .measGateSource= GATE_WDTp_SMCLK,      // 0->SMCLK, 1-> ACLK
                    .accumulationCycles = WDTp_GATE_512    // 512
};
```

3.2.2.2 RO_COMPB_TAx_WDTA

函数中的 RO_COMPB_TAx_WDTA 定义与 Comp_A+ 解决方案一样。 Comp_B 外设解决方案在执行中有所不同，如图 13所示，集成了基准电路和到 Timer_A 外设的连接。

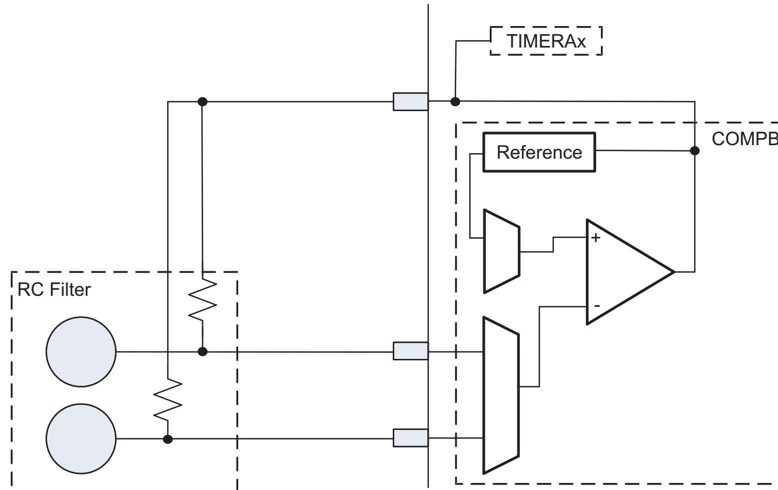

图 13. RO_COMPB 电路原理图

3.2.2.2.1 端口参数

如果图 14所示，Comp_B 的不同执行需要一个针对传感器定义的替代参数集。

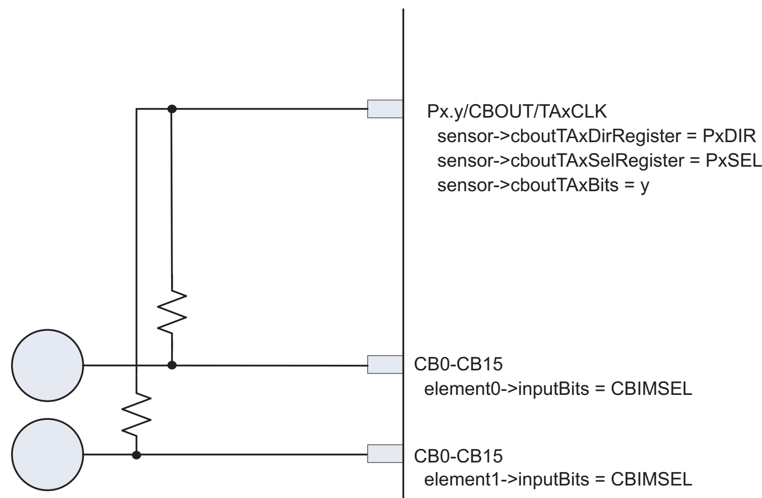

图 14. RO_COMPB 端口参数

*cboutTAxRegister*和*cboutTAxRegister*确定了端口方向地址和端口选择地址。 变量*cboutTAxBits*定义了方向和选择寄存器内将被设定/复位的位以选择端口的 CBOUTx 输出和 TxCLK 输入功能。 请注意这些端口共用 5xx 系列器件上的同一个 I/O。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

*cbpBits*禁用了端口引脚（也可被用作比较器输入）上的数字 I/O 功能。 这个被应用到 Comp_B 控制寄存器 CBCTL3。 CBCTL3 中的 CBPDy 位禁用比较器通道 y 的端口（即，CBPDy 禁用 CBy 而非 Px.y）。

3.2.2.2.2 定时参数

两个定时参数定义了 WDTA 间隔，此间隔为针对 RO_COMPB_TAx_WDTA 执行的选通时间。 WDTA 模块提供了四个不同的源设置和八个安全装置时间间隔。

*measureGateSource*表明 WDTA 源：SMCLK，ACLK，VLO，或者 XCLK。 这个参数与安全装置定时器控制寄存器 (WDTCTL) 内的安全装置定时器时钟源选择位等效。

表 **7.** 安全装置 **Timer_A** 源选择定义

定义	值	源
GATE_WDTA_SMCLK	0x0000	SMCLK
GATE_WDTA_ACLK	0x0020	ACLK
GATE_WDTA_VLO	0x0040	VLO
GATE_WDTA_XCLK	0x0060	XCLK

*accumulationCycles*被用来定义 RO_COMPB_TAx_WDTA 执行中的 WDTA 间隔。 这个参数与安全装置定时器控制寄存器 (WDTCTL) 中的间隔选择位等效。

表 **8.** 安全装置 **Timer_A** 间隔选择定义

定义	值	间隔
WDTA_GATE_2G	0x0000	2G / 源
WDTA_GATE_128M	0x0001	128M / 源
WDTA_GATE_8192K	0x0002	8192k / 源
WDTA_GATE_512K	0x0003	512k / 源
WDTA_GATE_32768	0x0004	32768 / 源
WDTA_GATE_8192	0x0005	8192 / 源
WDTA_GATE_512	0x0006	512 / 源
WDTA_GATE_64	0x0007	64 / 源

下面的示例描述了一个由四个元件组成的传感器，每个元件用 RO 方法在 512000/SMCLK 周期内测量。

```
const struct Sensor sliderA =
               {
                   .halDefinition = RO_COMPB_TA0_WDTA,
                   .numElements = 4,
                   .baseOffset = 0,
                   .cbpdBits = (BITC+BITD+BITE+BITF),
                   // Pointer to elements
                   .arrayPtr[0] = &element0,       // point to first element
                   .arrayPtr[1] = &element4,
                   .arrayPtr[2] = &element8,
                   .arrayPtr[3] = &elementC,
                   .cboutTAxDirRegister = (uint8_t *)&P3DIR,        // PxDIR
                   .cboutTAxSelRegister = (uint8_t *)&P3SEL,        // PxSEL
                   .cboutTAxBits = BIT4, // P3.4
                   // Timer Information
                   .measGateSource= GATE_WDTA_SMCLK,
                   .accumulationCycles= WDTA_GATE_512K //
                };
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

5xx 系列内的不同成员器件在 CBOUT 和 TA0 之间或者 CBOUT 和 TA1 之间，在某些情况下，在上述两个情况下提供一个内部连接。 除了 HAL 定义名称，对于 TA0 和 TA1 的说明和参数一样。

3.2.2.3 *RO_PINOSC_TA0_WDTp*

张弛振荡器的引脚振荡器 (PinOsc) 工具用数字 I/O 和一个内部变极器内的施密特触发器输入取代了比较器和基准电路。 到 RC 滤波器的 PinOsc 反馈路径由集成电阻器完成。 在图 15的 RC 滤波器中集成的电阻器为 'R'。

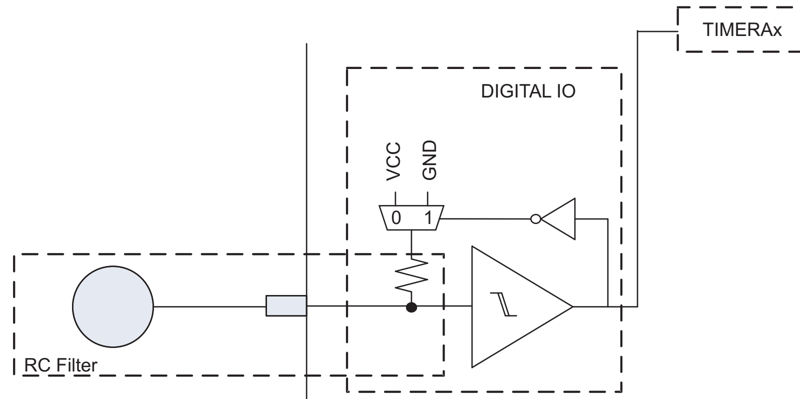

图 **15. RO_PinOsc** 电路原理图

3.2.2.3.1 *端口参数*

由于张弛振荡器由内部电路实现，端口参数只包括端口选择寄存器 (PxSEL)，端口选择 2 寄存器 (PxSEL2)，和输入位定义。 所有这些参数被定义在元件级并且在传感器级上没有相关的端口定义。

3.2.2.3.2 *定时参数*

公用定时参数，measGateSource 和 accumulationCycles，与 Comp_A+ 工具一致（请见节 3.2.2.1.2）。

下面的传感器定义描述了由一个元件组成的传感器，此元件由 RO 方法在一个 8192/SMCLK 周期内测量。

```
const struct Sensor middle_button =
               {
                  .halDefinition = RO_PINOSC_TA0_WDTp,
                  .numElements = 1,
                  .baseOffset = 4,
                  // Pointer to elements
                  .arrayPtr[0] = &middle_element, // point to first element
                  // Timer Information
                  .measGateSource= GATE_WDT_SMCLK, // 0->SMCLK, 1-> ACLK
                  //.accumulationCycles= WDTp_GATE_32768        //32768
                  .accumulationCycles= WDTp_GATE_8192         // 8192
                  //.accumulationCycles= WDTp_GATE_512          //512
                  //.accumulationCycles= WDTp_GATE_64           //64
               };
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

3.2.2.4 RO_PINOSC_TA0

选择 MSP430 器件作为一个 RO_PinOsc 的替代工具 (1) 为了使用到定时器捕捉输入的内部 ACLK 连接。 选通时间为捕捉事件的数量（与 ACLK 周期等效），而频率计数器仍然是源自张弛振荡器的外设 Timer_A0。由于捕捉中断代表一个单一振荡，一个软件循环对几个外部中断进行计数来创建等效选通时间。 与在低功耗模式下完成测量的 WDT 方法不同，这个软件循环方法将会由于 CPU 保持在激活模式下而消耗更多的电能。

3.2.2.5 端口参数

由于张弛振荡器由内部电路实现，端口参数只包括端口选择寄存器 (PxSEL)，端口选择 2 寄存器 (PxSEL2)，和输入位定义。 所有这些参数被定义在元件级并且在传感器级上没有相关的端口定义。

3.2.2.6 定时参数

如图 16所示，针对 RO_PINOSC_TA0 实现的唯一定时参数定义为 ACLK 周期的数量：传感器 -> accumulationCycles。

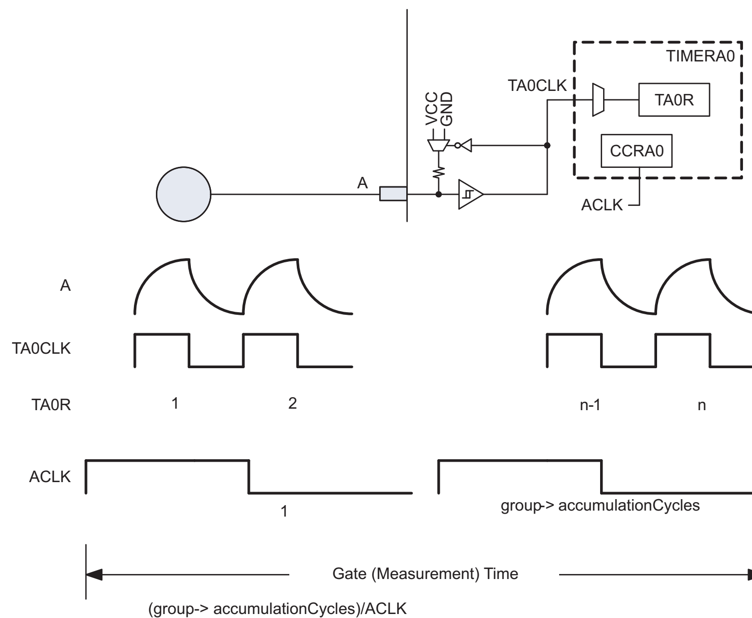

图 16. RO_PINOSC_TA0 定时参数

下面的传感器定义说明了一个由单原件组成的传感器，使用 RO 方法在 100/ACLK 周期内测量。

```
const struct Sensor volume =
               {
                  .halDefinition = RO_PINOSC_TA0,
                  .numElements = 2,
                  .baseOffset = 0,
                  // Pointer to elements
                  .arrayPtr[0] = &up,          // point to first element
                  .arrayPtr[1] = &down,        //
                  // Timer Information
                  .accumulationCycles= 100     // 100 ACLK cycles
                };
```

(1) 请见器件专用表以确定定时器捕捉输入是否支持这个到 ACLK 的连接。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

3.2.3 RC 方法的定义

3.2.3.1 RC_PAIR_TA0

RC 方法测量 RC 时间常数，在这里 R 代表电阻器，而 C 代表电极的电容值。 这个方法使用一个单一计时源（外设或者软件）来测量电容 'n' 次充放电所花费的时间。 为了测量充电和放电的时间，需要两个 IO。这一要求反映在软件库内函数名称上：RC_PAIR。

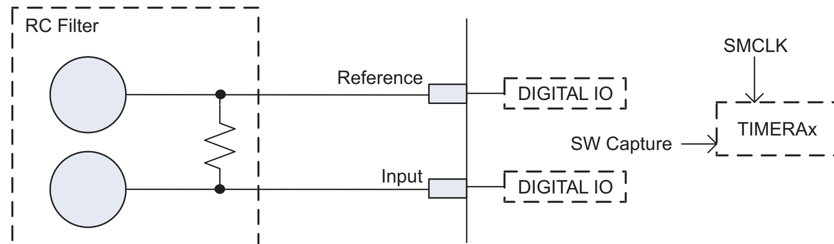

图 **17. RC** 电路原理图

3.2.3.1.1 端口参数

定义输入和基准的端口参数在元件级而非传感器级上进行了说明（请见节 3.1.1.4）。

3.2.3.1.2 计时参数

RC 方法使用定时器外设来测量 RC 电路的充放电时间。 可通过累积几个图 18中所显示的充电/放电周期来增加这个测量（在时间和数量方面增加）。 在参数中定义了周期数量：传感器 -> 累积周期。

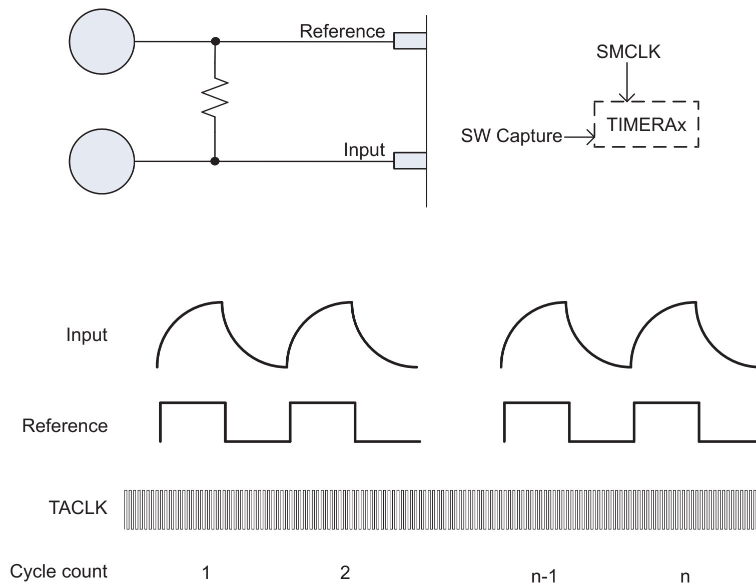

图 **18. RC** 计时参数

版权 © 2011, Texas Instruments Incorporated

下面的传感器定义描述了一个由两个元件组成的传感器，这两个元件由 RC 方法测量。 每个元件的选通时间为四个充放电周期。

```
const struct Sensor scroll =
               {
                  .halDefinition = RC_PAIR_TA0,
                  .numElements = 2,
                  .baseOffset = 0,
                  // Pointer to elements
                  .arrayPtr[0] = &left,          // point to first element
                  .arrayPtr[1] = &right,         //
                  // Timer Information
                 .accumulationCycles= 4          // 4 charge/discharge cycles
               };
```

3.2.4 对于快速扫面张弛振荡器 (RO) 方法的定义

快速张弛振荡器 (fRO) 方法用于缩小 RC 和 RO 方法间的差距。 fRO 方法提供了 RC 方法的快速扫描速率以及 RO 方法的提升的灵敏度。 相对于 RO 方法，现在选通时间（选通时间）为变量，根据张弛振荡器变化，而不是固定的。 选通时间的变化是一个被测量的电容元件的函数。 此外，在之前的 RO 方法中频率计数器是变化的，但是现在在 fRO 执行中，频率计数器时固定的。 频率计数器可以是一个基于 MCLK 的软件循环或者是一个基于系统时钟的定时器（通常为 SMCLK）。 现在由于电容增加，选通时间增加并且选通时间内的频率计数器数量也因此增加。

3.2.4.1 fRO_COMPAp_TAx_SW

fRO_COMPAp_TAx_SW 执行具有与 RO_COMPAp_TAx_WDTp 执行一样的硬件说明（请见图 10)。 之前已经提过，RO 和 fRO 方法间的关键区别时频率计数器和选通定时器是交换的。 现在选通定时器是被测量的电容的函数，而频率计数器由一个固定频率（一个系统时钟）供源。 在 fRO_COMPAp_TAx_SW 执行情况下，可变的选通定时器由张弛振荡器和外设 Timer_Ax 创建。 选通时间是一个带有 MCLK/10 频率的软件循环。

与 RO 方法相比，fRO 方法在更短的测量或选通时间内提供了相似的灵敏度（数量发生改变）。 表 9中的理论应用显示在更少的时间内使用 fRO 方法可实现相似的灵敏度。

表 9. 16MHz 时，fRO 和 RO 测量时间的比较

	RO_COMPAp_TAx_WDTp			fRO_COMPAp_TAx_SW		
	选通时间：(1MHz SMCLK，WDT，512)	计数器：(RO)	总数	选通时间：（160 个 RO 周期）	计数器：(MCLK/10)	总数
被触摸	512µs	600kHz	307	150/600kHz=250µs	16MHz/10	400
未触摸	512µs	700kHz	358	150/700kHz=214µs	16MHz/10	342
			51			58

快速 RO 方法通常用在提供多个定时器的器件中，这样频率计数器功能将与其它硬件定时器而非一个软件循环一起执行。 这样不但减少了功耗（运行在 LPM0 模式而不是激活模式下）而且删除了与软件指令有关的 10x 因子。

3.2.4.1.1 端口参数

fRO_COMPAp_TAx_SW 和 RO_COMPAp_TAx_WDTp 执行的端口参数是一样的（请见节 3.2.2.1和A.1 节）。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

3.2.4.1.2 计时参数

只有一个计时参数：accumulationCycles。图 19显示了在 fRO_COMPAp_TA0_SW 方法中如何使用参数 accumulationCycles 来累积多个张弛振荡器周期。

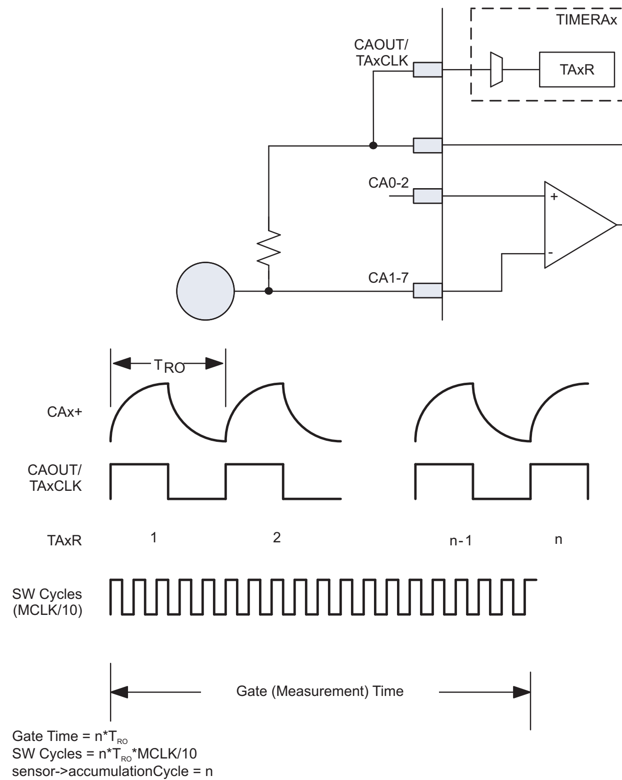

图 19. fRO_COMPAp_TA0_SW 计时参数

3.2.4.2 fRO_PINOSC_TA0_SW

fRO_PINOSC_TA0_SW 执行具有与 RO_PINOSC_TA0_WDTp 执行一样的硬件说明（请见节 3.2.2.3）。如上所述，RO 和 fRO 方法的关键区别是频率计数器和选通时间被交换。 现在选通定时器是被测量的电容的函数，而频率计数器由一个固定频率（一个系统时钟）供源。 在 fRO_PINOSC_TA0_SW 执行情况下，可变的选通定时器由张弛振荡器和外设 Timer_Ax 创建。 选通时间是一个带有 MCLK/10 频率的软件循环。

与 RO 方法相比，fRO 方法在更短的测量或选通时间内提供了相似的灵敏度（数量发生改变）。 表 7 中的理论应用显示在更少的时间内使用 fRO 方法可实现相似的灵敏度。

表 10. 16MHz 时，fRO 和 RO PinOsc 测量时间的比较

	RO_PINOSC_TA0_WDTp			fRO_PINOSC_TAx_SW		
	选通时间：**(1MHz SMCLK，WDT，512)**	计数器：**(RO)**	总数	选通时间：（**160 个 RO 周期**）	计数器：**(MCLK/10)**	总数
被触摸	512µs	1MHz	512	320/1MHz=320µs	16MHz/10	512
未被触摸	512µs	1.1MHz	563	320/1.1MHz=290µs	16MHz/10	465
			51			47

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

3.2.4.2.1 端口参数

与 RO_PinOsc 执行相似，在传感器级说明中没有端口参数。 PxSEL 和 PxSEL2 定义出现在元件级说明中。

3.2.4.2.2 计时参数

由于计时源是 hal 定义的一部分，因此唯一被定义的参数是选通时间内振荡的数量。 这个数，在图 20中显示为 'n'，由系统级上的变量 accumulationCycles 定义：传感器 -> accumulationCycles。

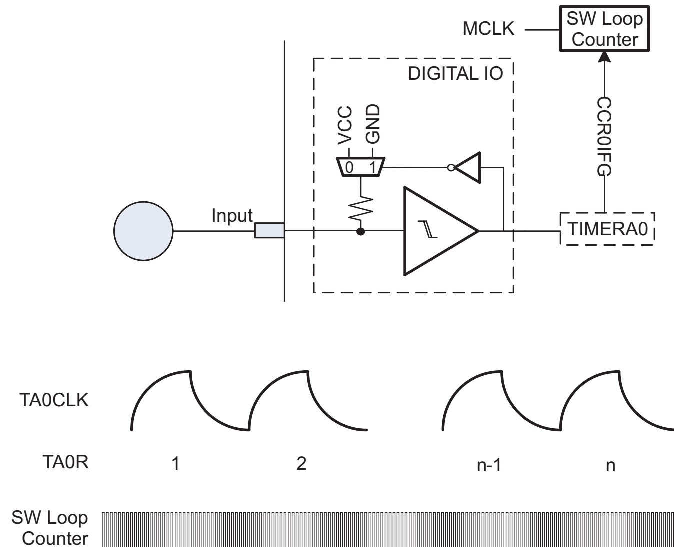

图 **20. fRO_PINOSC_TA0** 计时参数

3.2.4.3 *fRO_COMPB_TAx_SW*

fRO_COMPB_TAx_SW 执行具有与 RO_COMPB_TAx_WDTA 执行一样的硬件说明（请见节 3.2.2.2）。 如上所述，RO 和 fRO 方法的关键区别是频率计数器和选通时间被交换。 现在选通定时器是被测量的电容的函数，而频率计数器由一个固定频率（一个系统时钟）供源。 在 fRO_COMPB_TAx_SW 执行情况下，可变的选通定时器由张弛振荡器和外设 Timer_Ax 创建。 选通时间是一个带有 MCLK/10 频率的软件循环。

与 RO 方法相比，fRO 方法在更短的测量或选通时间内提供了相似的灵敏度（数量发生改变）。 表 11中的理论应用显示在更少的时间内使用 fRO 方法可实现相似的灵敏度。

表 **11. 25MHz** 时，**fRO** 和 **RO** 测量时间的比较

	RO_COMPB_TAx_WDTA			fRO_COMPB_TAx_SW		
	选通时间：**(1MHz SMCLK，WDT，512)**	计数器：**(RO)**	总数	选通时间：（**160** 个 **RO** 周期）	计数器：**(MCLK/10)**	总数
被触摸	512µs	600kHz	307	80/600kHz=133µs	25MHz/10	333
未被触摸	512µs	700kHz	358	80/700kHz=114µs	25MHz/10	285
			51			48

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

快速 RO 方法通常用在提供多个定时器的器件中，这样频率计数器功能将与其它硬件定时器而非一个软件循环一起执行。 这样不但减少了功耗（运行在 LPM0 模式而不是激活模式下）而且删除了与软件指令有关的 10x 因子。

3.2.4.3.1 端口参数

fRO_COMPAp_TAx_SW 和 RO_COMPAp_TAx_WDTp 执行的端口参数是一样的（请见节 3.2.2.2和）。

3.2.4.3.2 计时参数

只有一个计时参数；accumulationCycles。图 21显示了在 fRO_COMPB_TAx_SW 方法中如何使用参数 accumulationCycles 来累积多个张弛振荡器周期。

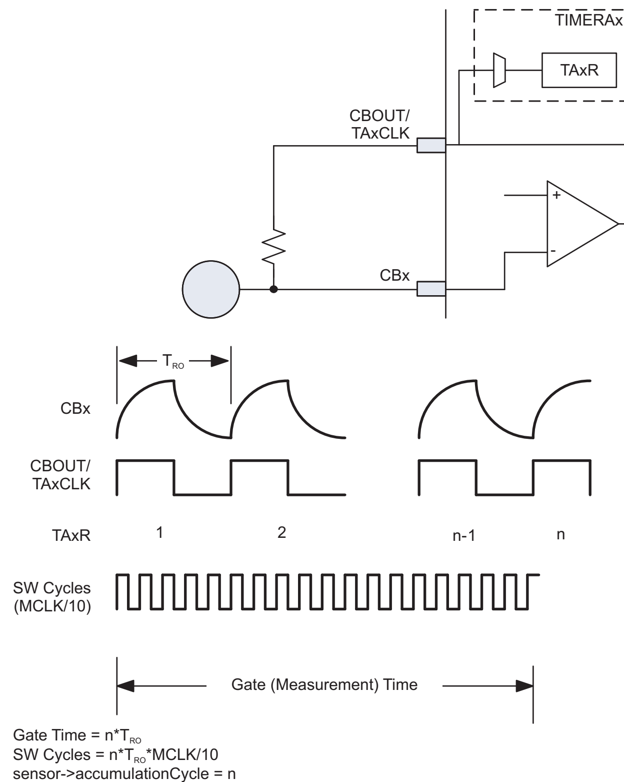

图 21. fR0_COMPB_TAx_SW 计时参数

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

3.2.4.4 fRO_COMPAp_SW_TAx

如名称所示，fRO_COMPAp_SW_TAx 执行使用一个软件定时器来创建选通时间，而将定时器外设作为频率计数器，在这里频率源为一个系统时钟。 由于这个执行不使用定时器来为张弛振荡计数，所以不再需要如图 22所示的物理连接。

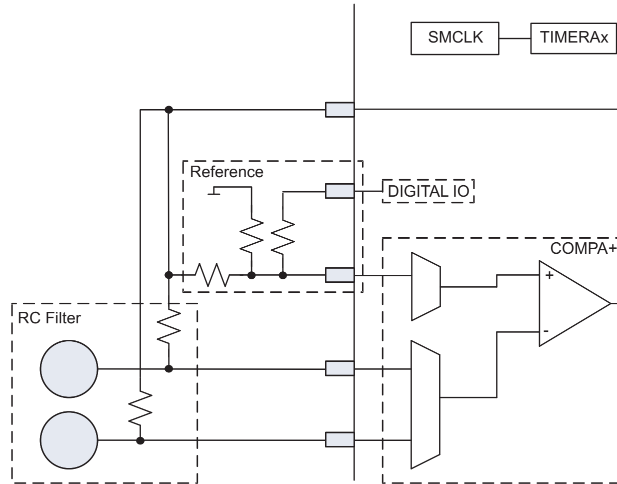

图 22. fRO_COMPAp_SW_TAx 总体说明

与 RO 方法相比，fRO 方法在更短的测量或选通时间内提供了相似的灵敏度（数量发生改变）。 表 12中的理论应用显示在更短的选通时间内使用 fRO 方法可实现相似的灵敏度。

表 12. 16MHz 时，fRO 和 RO 测量时间的比较

	RO_COMPAp_TAx_WDTA			fRO_COMPAp_SW_TAx		
	选通时间：(1MHz SMCLK，WDT，512)	计数器：(RO)	总数	选通时间：（160 个 RO 周期，MCLK 为 16MHz）[(1)]	计数器：(SMCLK)	总数
被触摸	512μs	600kHz	307	80/600kHz=133μs	16MHz	333
未被触摸	512μs	700kHz	358	80/700kHz=114μs	16MHz	285
			51			48

(1) 具有一个缓慢的 MCLK 会导致张弛振荡数量的计数误差。 也就是说，如果软件轮询循环太慢的话，有可能丢失振荡周期，从而导致比预计时间更长的选通时间。 软件循环为 14 周期，因此，MCLK 频率必须比最大张弛振荡频率快 14 倍。

RO 方法通常用在具有多个定时器的器件中，这样频率定时器功能和选通定时器将由硬件完成。 这将减少功耗（运行在 LPM0 模式而非激活模式下）

3.2.4.4.1 端口参数

除了 txclkDirRegister，txclkSelRegister，和 txclkBits 未被使用外，端口参数与节 3.2.2.1中描述的那些参数相似。

3.2.4.4.2 计时参数

有三个计时参数；measureGateSource，sourceScale，和 accumulationCycles。 定义选通时间的选通定时器由 accumulationCycles 定义。 在达到 accumulationCycles 之前软件循环计算张弛振荡器的周期，并在此时读取定时器，TAx。 measureGateSource 和 sourceScale 配置 TAx 外设。 这些参数明确地定义了源（通常为 SMCLK）和定时器分频器（通常为 1 分频）。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

*measureGateSource*确定 TAx 的时钟源。 这个参数与 Timer_A 控制寄存器 TACTL 中的 TASSELx 位等效。

表 13. 针对 fRO_xxxx_SW_Txx 的 measureGateSource 定义

名称	定义	时钟源
TIMER_TxCLK	0x0000	TxCLK
TIMER_ACLK	0x0100	ACLK
TIMER_SMCLK	0x0200	SMCLK
TIMER_INCLK	0x0300	INCLK

*sourceScale*用于定时器源分频。 这个参数与 Timer_A 控制寄存器 TACTL 中的输入分频器位 (IDx) 等效。

表 14. 针对 fRO_xxxx_SW_Txx 的sourceScale 定义

名称	定义	时钟源
TIMER_SOURCE_DIV_0	0x0000	TxCLK
TIMER_SOURCE_DIV_1	0x0040	ACLK
TIMER_SOURCE_DIV_2	0x0080	SMCLK
TIMER_SOURCE_DIV_3	0x00C0	INCLK

*accumulationCycles*定义了每个选通周期内张弛振荡器周期的数量。 在 fRO_COMPAp_SW_TAx 方法中，张弛振荡器周期的计数由一个软件轮询循环完成，此循环寻找一个比较器中断标志来表明一个振荡已经发生。

3.2.5 滑盖和滑轮特定定义

只有 API 函数 TI_CAPT_Wheel 和 TI_CAPT_Slider 需要以下定义。

为了在软件库中包含滑轮或者滑盖 API，需要在 structure.h 中做出以下定义：

```
/Are wheel or slider representations used?
//#define SLIDER
#define WHEEL

// Illegal slider position.  This value is returned
// when no touch on the wheel or slider is detected.
#define ILLEGAL_SLIDER_WHEEL_POSITION                       0xFFFF
```

在 structure.c 中，必须添加针对*points*（点）和*sensorThreshold*（传感器阀值）的定义。

变量*points*定义了一个滑盖或者滑轮边沿的点的数量。

*sensorThreshold*定义了传感器声明一个有效触摸所需的累积响应。 这个变量的用途是为了区分一个与传感器的真正交互与一个无意交互，无意交互只激活一个元件。

滑轮或者滑盖 sensorThreshold 与主元件和其相邻元件的响应相比较（x-1，x，x+1 的和）。 一个滑盖的端点是一个特殊情况，在这种情况下，只需比较一个端元件（主元件）和一个相邻元件。 如果响应超过 sensorThreshold，那么传感器的有效使用情况已生效并且位置被计算。 如果没有检测到有效使用情况，那么返回 ILLEGAL_SLIDER_WHEEL_POSITION（在 structure.h 中定义）。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

4 资源

根据配置的不同，这个软件库可消耗几个不同的资源，从而使这些资源不可用于主应用或者只在实际测量周期内不可用。 完全不可用的资源通常为安全装置定时器外设、数字 I/O、和被分配的内存资源。

这个软件库确实执行一个所有被使用寄存器的简单环境存储来大大减少对于复位外设的需要。 应该注意的是，环境存储并不是广泛存在的，一个好的做法就是在启用中断前清除 IFG 标志。 [1]。

4.1 时间（测量时间）

软件库中的 API 调用正在阻断所有调用并且在测量完成之前不将 CPU 归还给应用。 决定 CPU 不可用时间长短的主要因素是电容测量时间。 这个时间可以是来自一个固定时钟源的周期数量（系统时钟）或者一个来自可变（张弛振荡器）时钟源的周期数量。

表 15显示了一些针对多种电容测量方法和设置的示例选通时间。 有必要注意的是，灵敏度与选通时间直接相关。 缩短选通时间将会导致灵敏度的降低。 在 fRO 执行中，可通过增加馈入计数器的固定频率时钟来增加灵敏度（同时保持较短的选通时间）（请见节 3.2.4）。

(1) 一个明显的例子就是使用 Timer_A3 时，在这里软件库并不使用所有三个捕捉和控制寄存器；然而，当定时器被使用时，CCIFG 可被设定。

表 15. 测量时间示例

方法	选通时钟源 **(.measGateSource)**	时间间隔定义 **(.accumulationCycles)**	时间 **(ms)**
RO_XXX_YYY_WDTp/A	ACLK=VLO~12kHz	64 (WDTp_GATE_64)	5.33
	SMCLK=2MHz	8192 (WDTp_GATE_8192)	4.1
	SMCLK=1MHz	512 (WDTp_GATE_512)	0.512
RO_PINOSC_TA0	ACLK=VLO~12kHz	100	8.3
RC_PAIR_XXX	上升+下降（未被触摸）~1.4μs	20	0.028+TBD[1]
	上升+下降（被触摸）~1.6μs	20	0.032+TBD
fRO_COMPx_YYY_SW	RO（未被触摸）~700kHz	4000	5.71
	RO（被触摸）~600kHz	4000	6.67
fRO_PINOSC_YYY_SW	RO（未被触摸）~1.2MHz	4000	3.33
	RO（被触摸）~1MHz	4000	4
fRO_COMPx_SW_YYY	RO（未被触摸）~700kHz	500	0.714
	RO（被触摸）~600kHz	500	0.833

(1) 这个额外的时间是与使用软件来在几个周期内设立充电和放电相关的开销。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

4.2 存储器：闪存和 RAM

软件库消耗的代码空间的数量是元件数量、传感器数量、测量方法、和抽象级别的直接函数。表 16显示了一个代码尺寸是如何随着抽象级别的升高而增加的示例。

表 16. 示例闪存资源分配

API 调用	软件库（字节）	配置结构：6 个元件、3 个结构 (RO_PINOSC_TA0_WDTp)	注释
TI_CAPT_Raw	396 (0x18C)	114 (0x72)	优化级别 0 (CCSv4，CGT v3.3.2)
TI_CAPT_Init_Baseline TI_CAPT_Update_Baseline TI_CAPT_Custom	1840 (0x730)	114 (0x72)	优化级别 0 (CCSv4，CGT v3.3.2)
TI_CAPT_Init_Baseline TI_CAPT_Update_Baseline TI_CAPT_Custom TI_CAPT_Button TI_CAPT_Wheel	2828 (0xB0C)	120 (0x78)	优化级别 0 (CCSv4，CGT v3.3.2)

RAM 可被静态分配以保持基线跟踪特性。 所需的 RAM 数量是元件总数量的函数：每元件 2 字节。 软件库使用 TOTAL_NUMBER_OF_ELEMENTS 定义来表示需要为基线跟踪分配 RAM 以及需要分配的量。 当只使用 TI_CAPT_RAW API 时，那么 TOTAL_NUMBER_OF_ELEMENTS 不应被定义并因此无需消耗 RAM 资源。

可静态或者动态分配 RAM 来执行测量以确定电容的变化（TI_CAPT_Custom 和传感器抽象）。 如果 RAM 被静态分配，必须在 structure.h 对 RAM_FOR_FLASH 进行定义。 分配的 RAM 空间的数量取决于每个传感器的元件最大数量，每个元件 2 字节。

以闪存空间被占用为代价，这个 RAM 可从一个堆 (HEAP) 中被动态分配。 为了动态分配 RAM，RAM_FOR_FLASH 定义必须被省略。 按照最大传感器中的元件数量，HEAP 的大小必须被设定（在 IDE 中）为 2 字节加上 2 字节。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

4.3 系统时钟

软件库不对系统时钟（MCLK，SMCLK，或者 ACLK）做任何调整并且如针对电容测量的应用层中定义的那样使用它们。 有必要了解应用环境中软件库的时钟使用。 例如，如果使用安全装置定时器间隔将电容测量时间设定改为 8192/SMCLK，那么在应用中改变 SMCLK 的频率也将改变电容测量期间的测量时间。 如果在一个应用中改变针对电容测量的时钟源，那么有必要相应地重新初始化基线跟踪。

4.4 外设

外设的不同组合可被用于测量电容变化。 虽然这些外设在电容测量期间不可被应用使用，大多数外设可在此时与其它应用或者函数共用或复用。

4.4.1 定时器：A，B，D

每次测量时定时器外设被重新配置和初始化，因此，当没有发生电容测量时，定时器外设可被用于其它函数。

4.4.2 安全装置定时器

如果外设已经在软件库中被选择使用时，安全装置定时器 ISR 不可用。

4.4.3 比较器：A，B

每次测量时比较器外设被重新配置和初始化，因此，当没有发生电容测量时，比较器外设可被用于其它函数。 建议将其它输入连接到电容传感器元件，这是因为这有可能与电容测量对接。

4.4.4 数字 I/O

建议在用作电容测量的 I/O 引脚上共用或者复用功能。

5 API 调用

软件库提供三个不同的抽象层。 最低级的抽象为 TI_CAPT_Raw API 函数调用。 这个函数调用测量适当的传感器并且为应用层提供“原始”电容测量。 TI_CAPT_RAW 函数是最强大的，这是因为它可以在电容测量的解释和应用中实现最大灵活性。

抽象的下一个级别是 TI_CAPT_Custom API 函数调用。 这个 API 调用 TI_CAPT_Raw 函数并且还包含一个基线跟踪算法。 TI_CAPT_Custom API 为应用层提供来自基线电容的已测得电容的变化幅度。 如果变化的方向与所需的方向一致，变化值只提供给应用层。 如果变化的方向与所需方向相反，这个信息被基线跟踪所使用，但是并不提供给应用层。 提供了几个 API 函数调用来调整运行时间内的基线跟踪。 TI_CAPT_Custom API 用于与“定制”元件和传感器设计一起使用。 基线跟踪仍可使用，但是电容变化中的解释和应用必须在应用层被处理。

TI_CAPT_Custom API 之上的抽象级别，包括一个按钮、按钮组、一个滑轮、和一个滑盖的传感器表示。 相关的 API 包括 TI_CAPT_Custom 函数的解释和应用。

表 17. API 函数

类别	API 函数
电容测量	uint8_t TI_CAPT_Button(const struct Sensor *);
电容测量	const struct Element * TI_CAPT_Buttons(const struct Sensor *);
电容测量	uint16_t TI_CAPT_Slider(const struct Sensor*);
电容测量	uint16_t TI_CAPT_Wheel(const struct Sensor*);
电容测量	void TI_CAPT_Custom(const struct Sensor *, uint16_t*);

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

表 17. API 函数 (continued)

类别	API 函数
电容测量	void TI_CAPT_Raw(const struct Sensor*, uint16_t*);
基线跟踪	void TI_CAPT_Init_Baseline(const struct Sensor*)
基线跟踪	void TI_CAPT_Update_Baseline(const struct Sensor*,uint8_t)
基线跟踪	void TI_CAPT_Reset_Baseline_Tracking(void);
基线跟踪	void TI_CAPT_Update_Tracking_DOI(uint8);
基线跟踪	void TI_CAPT_Update_Tracking_Rate(uint8_t, uint8_t);

5.1 *uint8_t TI_CAPT_Button(Sensor *);*

输入：到传感器的指针，定义了代表一个按钮的元件

输出：0/1，

功能：测量按钮。 一个 0 意味着电容的变化少于或者等于元件中设定的阀值，而 1 表示电容的变化已经超过了阀值。

5.2 *element * TI_CAPT_ Buttons(Sensor *);*

输入：到传感器的指针，定义了一组元件，其中每个元件代表一个按钮

输出：到一个元件或者 0 的指针

功能：测量传感器（按钮）并确定哪个按钮，如果有的话，正在被触摸。 这个函数的返回值为指向元件的结构指针，此元件比其阀值高最大裕量（阀值值的 %）。 如果没有按钮超过其阀值（在元件结构中设定），那么这个函数返回一个 0 或者 'Null Pointer' （空指针）。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

5.3 uint16_t TI_CAPT_Slider(Sensor *);

输入：到传感器的指针，定义了组成滑盖的一组元件

输出：位于滑盖上；ILLEGAL_SLIDER_WHEEL_POSITION -> 无触摸；0 - 最大 -> 触摸的位置为传感器结构定义的最大值。

功能：这个函数在触摸被检测到时返回滑盖的位置，而在未检测到触摸时返回一个无效值。

传感器结构中元件的顺序应该代表沿着滑盖的元件的顺序。 在传感器位置内确认的第一个元件代表最低值：传感器排列中第一个元件的外沿为位置 0。最后一个元件代表最高值：排列中最后一个元件的外沿代表传感器内的分辨路数量（点）。

```
const struct Sensor group =
                {
                   .numElements = 5,
                   .baseOffset = 0,
                   // Pointer to elements
                   .arrayPtr[0] = &element0,   // point to first element
                   .arrayPtr[1] = &element1,
                   .arrayPtr[2] = &element2,
                   .arrayPtr[1] = &element3,
                   .arrayPtr[2] = &element4,
                   .points = 100,
                   .sensorThreshold = 50
                  };
```

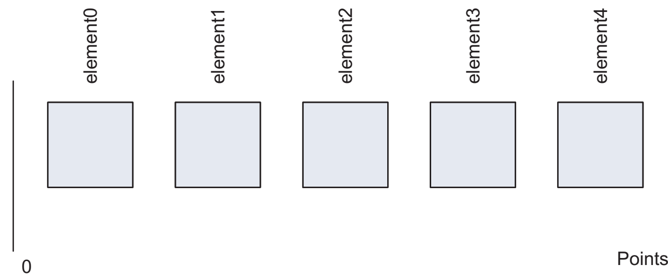

图 **23.** 滑盖执行

如果没有元件超过其阀值（在元件结构中设定），那么这个函数返回值 ILLEGAL_SLIDER_WHEEL_POSITION，这个值在 structure.h 中进行了定义。

5.4 uint16_t TI_CAPT_Wheel(Sensor *);

```
uint16_t TI_CAPT_Wheel(Sensor *);
```

输入：到传感器的指针，定义了组成一个滑轮的一组元件。

输出：滑盖上的位置：

ILLEGAL_SLIDER_WHEEL_POSITION -> 无触摸

0 - 最大值 -> 触摸在传感器结构定义“点”所定义的最大值的位置上

版权 © 2011, Texas Instruments Incorporated

功能：测量传感器内的元件。 这个函数或者返回一个无效数来表示没有触摸被测量到或者一个代表滑轮上位置的有效值。

传感器结构内的元件顺序应该表示滑轮周围的元件顺序。 在传感器位置内确认的第一个元件代表最低值：传感器排列中第一个元件的外沿为位置 0。最后一个元件代表最大值：排列中最后一个元件的外沿代表这个值所包围的点。

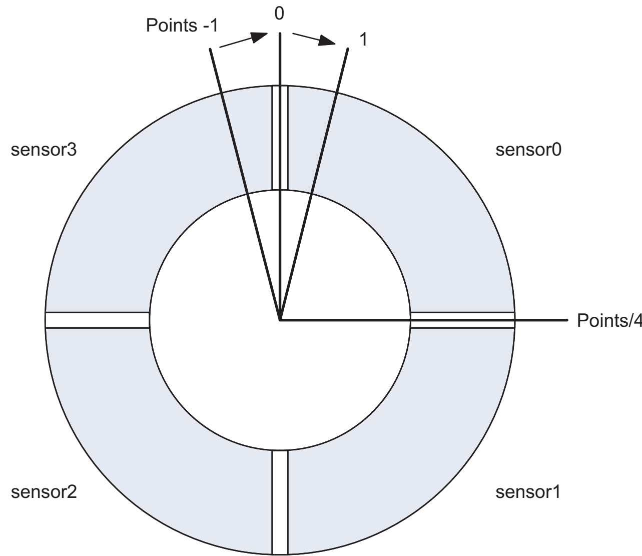

图 **24.** 滑轮示例

如果没有元件超过其阀值（在元件结构中设定），那么这个函数返回值 ILLEGAL_SLIDER_WHEEL_POSITION，这个值在 structure.h 中进行了定义。

5.5 *void TI_CAPT_Custom(Sensor *, uint16_t *);*

输入：到传感器的指针，定义了组成一个定制接口的一组元件，以及到排列的指针，随测量结果更新。

输出：无

功能：测量传感器内针对每个元件相对于基线（电容历史记录）的电容变化。

传感器结构内的元件顺序可随意安排，但是应用和配置之间必须了解元件顺序。 排列中的第一个元件与传感器结构中的第一个元件相对应。

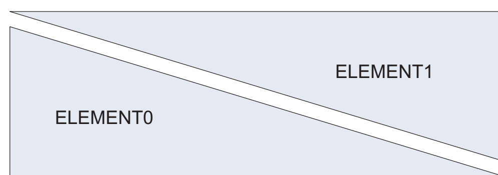

图 **25.** 定制滑盖示例

这个功能要求应用分配一个数组，当被调用时，这个函数能够填充此数组。 当需要在应用层控制传感器的功能，而低级别函数或者测量和基线跟踪仍由软件库管理时，这个类型的 API 就很实用。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

5.6 void TI_CAPT_Raw(Sensor*, uint16_t*);

输入：到传感器的指针，定义了组成一个定制接口的一组元件，以及到排列的指针，随测量结果更新。

输出：无

功能：测量传感器内每个元件的电容。 这个函数用电容的定时器表示法来更新输入数组。

传感器结构内的元件顺序可随意安排，但是必须由应用和配置管理。 通过的排列中第一个元件与传感器结构中的第一个元件相对应。

这个函数要求应用分配一个数组，当被调用时，这个函数可以填充此数组。 当需要在应用层内控制传感器和基线跟踪的功能，而测量仍由软件库管理时，这个类型的 API 就很实用。

5.7 void TI_CAPT_Init_Baseline(Sensor *);

输入：到传感器的指针

输出：无

功能：测量传感器并且直接将测得的值置入相关的基线变量中。 操作开始时，存储在 RAM 的基线值有可能处于位置的状态。 对于传感器内的每个元件，使用这个函数将测量载入 RAM 空间。 不同的函数自动地将当前的测量与已有的基线函数进行平均并且有可能在跟踪算法达到一个代表环境的稳定状态值之前产生错误性能。

5.8 void TI_CAPT_Update_Baseline(Sensor *, uint8_t);

输入：到传感器的指针以及与基线进行平均的测量的数量。

输出：无

功能：将基线与输入中定义的测量数平均。 这个函数的用途是为了进行单独测量来更新传感器内每个元件的基线值。

5.9 void TI_CAPT_Reset_Tracking (void);

输入：无

输出：无

功能：复位基线跟踪，这样所需的方向为电容的增加。 还复位了跟踪速率，这样基线跟踪低设置 (01) 朝所需方向变化并快速设置 (00) 上朝所需方向相反的方向发生电容变化；也就是说，跟踪在快速设置上电容减少并且在慢设置上电容增加。

5.10 void TI_CAPT_Update_Tracking_DOI (uint8_t);

输入：所需的方向。

输出：无。

功能：如果输入为真（非零），那么所需的方向为电容的增加。 如果输入为 0x00，那么所需方向为电容的减少。 在大多数应用中，所需的方向为电容的增加，这是因为在一个域内引入一个对象会导致电容的增加。在一些情况下，当一个对象出现时对一个对象的识别是有益的，然后当对象被删除后，检测所需方向的改变。 这通常在对象长时间保持不变时有用。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

5.11 void TI_CAPT_Update_Tracking_Rate (uint8_t);

输入：基线的速率调整来改变电容，朝所需方向变化并朝所需方向相反的方向变化。

表 18. 更新跟踪速率格式

输入值	所需方向的跟踪速率	所需方向相反的跟踪速率
0000 0000b	极慢	快
0001 0000b	慢（缺省值）	快（缺省值）
0010 0000b	中	快
0011 0000b	快	快
0000 0000b	极慢	快
0100 0000b	极慢	中
1000 0000b	极慢	慢
1100 0000b	极慢	极慢

输出：无。

功能：根据表 18更新跟踪速率。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

6 建立测量参数

测量参数，maxResponse，threshold，和 sensorThreshold 受到传感器定义内所选计时参数的影响。 校准是一个迭代过程，在这个过程中传感器参数被改变以在测量参数被选择前提供适当的响应。

6.1 测量函数

TI_CAPT_Raw 函数不使用任一测量参数并且可被用于为 TI_CAPT_Custom 函数建立一个阀值。 当一个传感器内的一个或者多个元件超过阀值时，TI_CAPT_Custom 要求一个阀值参数来禁用基线跟踪。 请注意，使用原始函数时，电容的增加由 RC 和 fRO 方法中计数的增加表示，由 RO 方法中技术的减少表示。

```
#include "CTS_Layer.h"

// Need to Allocate at least 10 bytes to HEAP in IDE
// threshold set to '0' in structure.c

unsigned int delta_data[4];

void main(void)
{

   TI_CAPT_Init_Baseline(&group);
   TI_CAPT_Update_Baseline(&group,30);

   while(1)
   {
    TI_CAPT_Custom(&group,delta_data);
    __no_operation();       // set breakpoint here
   }
}
```

表 **19.** 使用 **RO** 方法的原始结果示例

有源元件	raw_data[0]	raw_data[1]	raw_data[2]	raw_data[3]
无	394	435	426	367
0（轻）	257	424	427	369
0（正常）	137	410	428	371
0（重）	110	304	420	371
无	390	435	426	367
1（轻）[(1)]	367	223	408	367
1（正常）	361	165	401	366
1（重）	226	117	332	365
无	389	435	425	368
2（正常）	382	349	146	341
无	390	435	426	267
3（正常）	388	421	255	111

(1) 一个轻、正常、和重按压的之间的差异是接触的表面积不同。 在使用一个手指的应用中，由于施加了更多的压力，指端变平，从而产生一个更大的表面积。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

从表 19，交互和无交互之间的差异可建立针对元件 0 和元件 1 的阀值。 凭经验，一个好方法将阀值设定为差异的一半。 例如，在这个配置中，为了保证在传感器 0 和传感器 1 上检测到一个轻触摸，阀值应该分别为 137/2 和 212/2。

6.2 按钮和按钮组

为 TI_CAPT_Button 和 TI_CAPT_Buttons 抽象定义阀值值由 TI_CAPT_Custom 函数完成。TI_CAPT_Custom 函数测量自基线（由软件库跟踪）的变化的幅度。 只为所需的方向返回变化的幅度。 相反方向的变化由 0 表示。在下面的代码示例中，所需的方向是一个增加的电容。 有关改变所需方向的说明请见 TI_CAPT_Update_Tracking_DOI API。

```
#include "CTS_Layer.h"

// threshold set to '0' in structure.c

unsigned int delta_data[4];

void main(void)
{

   TI_CAPT_Init_Baseline(&group);
   TI_CAPT_Update_Baseline(&group,30);

   while(1)
   {
     TI_CAPT_Custom(&group,delta_data);
     __no_operation(); // set breakpoint here
   }
}
```

表 **20.** 使用 **RO** 方法的电容结果变化的示例

有源元件	delta_data[0]	delta_data[1]	delta_data[2]	delta_data[3]
无	0	0	0	0
0（轻）	130	11	0	0
0（正常）	188	16	0	0
0（重）	287	71	0	0
无	0	0	0	0
1（轻）	14	205	13	1
1（正常）	30	288	35	2
1（重）	222	328	91	2
无	0	0	0	0
2（正常）	3	49	292	24
无	0	0	0	0
3（正常）	0	5	52	243

阀值计算与表 19中显示的相似。

使用一组元件的 API，与 TI_CAPT_Buttons API 类似，除了阀值之外，要求maxResponse 参数的定义。借助于一个传感器内的多个元件，maxResponse 被用于使每个元件的响应标准化并且识别哪个元件具有主响应。 标准化的用途是为了解决元件性能的可能差异。 将表 20作为一个示例，maxResponse 将只是重交互的结果。 请牢记，阀值和 maxResponse 间的关系如节 3.1.1中所描述的那样。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

6.3 传感器组：滑轮和滑盖

对于滑轮和滑盖 API，阀值和 maxResponse 测量参数的意思具有细微的不同。 当交互首次“滑动”进元件区域内时，阀值代表预计的最小响应。 当交互滑过元件时，maxResponse 代表最大返回值。 这通常为元件的中心，在中心区域在元件和交互之间有最大的区域重叠。图 26显示了如何使用定制函数来测量一个滑盖的性能并且为阀值和 maxResponse 变量确定值。

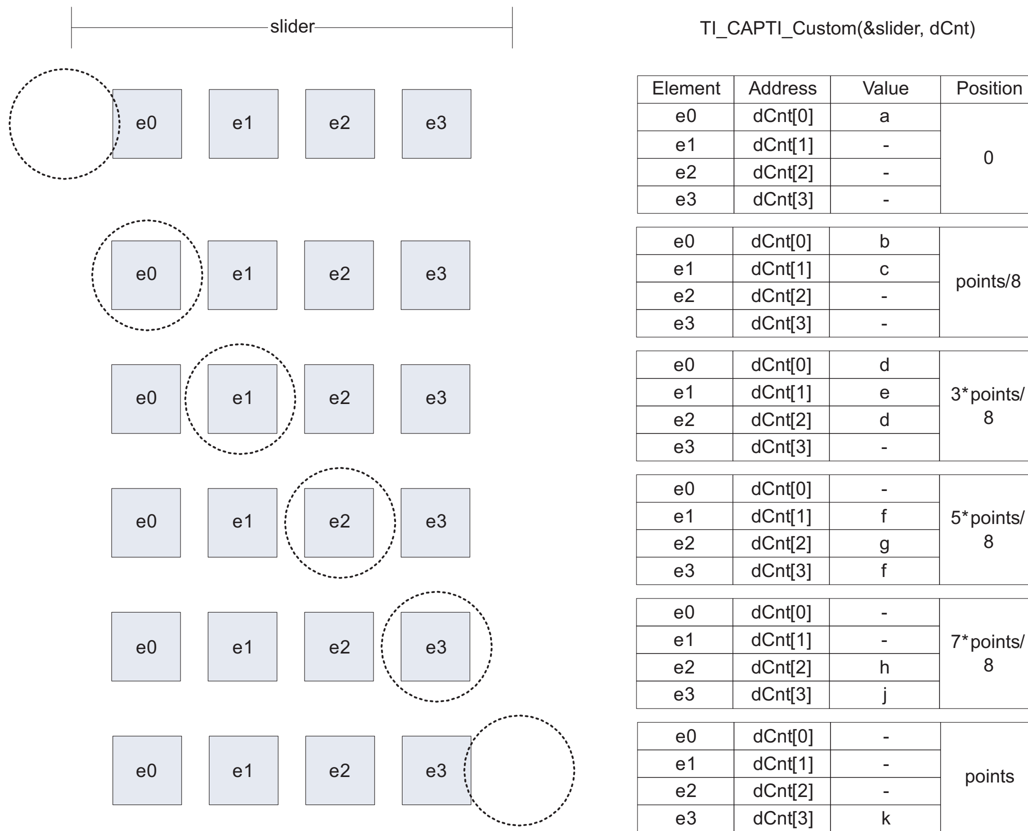

图 **26.** 一个四元件传感器的测量示例

理想情况下，电极的几何形状将引起针对 a，b，c，d，f，h，和 k 的等效的、非零响应。 更加重要的是，响应应该大于相应的最大返回值，b，e，g，或者 j 的 10%。

表 **21.** 一个四元件传感器的测量示例

元件	阈值	**maxResponse** [(1)]
e0	(a+d)/2	b 或者（阀值 + 655），其中的较小值
e1	(c+f)/2	e 或者（阀值 + 655），其中的较小值
e2	(d+h)/2	g 或者（阀值 + 655），其中的较小值
e3	(f+k)/2	j 或者（阀值 + 655），其中的较小值

(1) 在某些几何形状中，maxResponse 内的值并不是电极真正最大的返回值，但是返回值被记录在电极的中心。 重要的标准是相邻的元件（对于一个滑盖或者滑轮）有相等的返回值。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

如果设计满足这些标准，那么应该考虑在应用层中使用 TI_CAPT_Custom 函数并执行位置计算。 如果使用 TI_CAPT_Custom 函数，那么所需的阀值值和之前提到的一样。

滑轮和滑盖还要求一个第三测量参数，此参数是传感器结果的一部分，sensorThreshold。 如图 27中所描述的那样，sensorThreshold 定义了滑盖的有效区域。 一个好的起始值为 75。 在损失位置精度的前提下，减少这个值，增加滑盖区域。 相反地，增加整个值增加位置精度，但是交互必须更加严密地沿着滑轮或者滑盖的中心线。

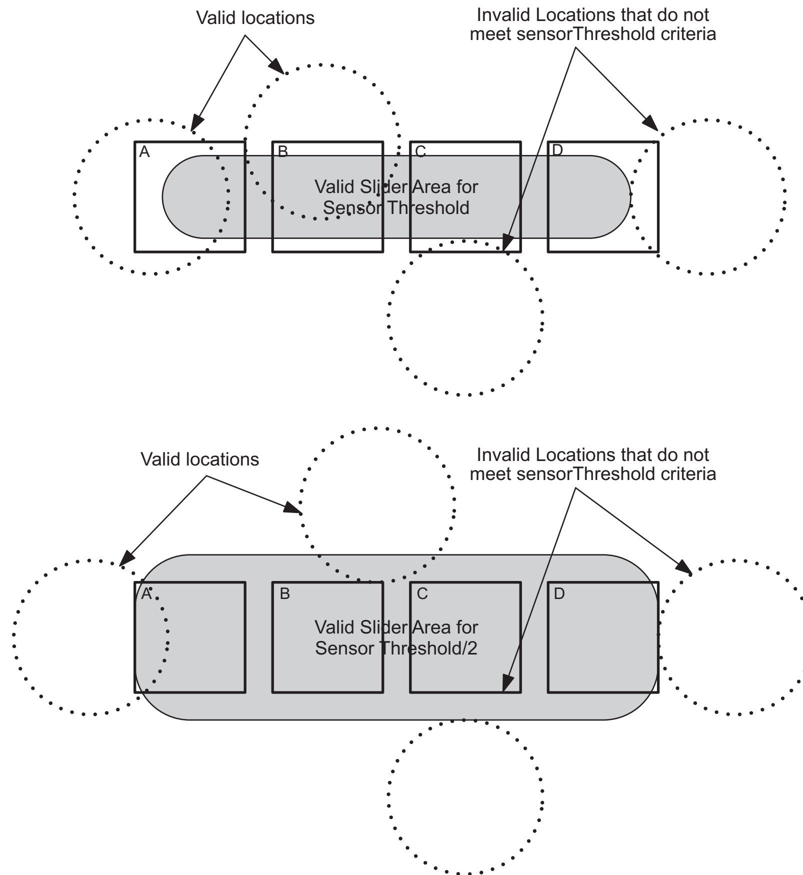

图 **27.** 有效滑盖位置作为一个传感器阀值的函数

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

附录 A 元件和传感器定义

A.1 RO_PINOSC_TA0_xx

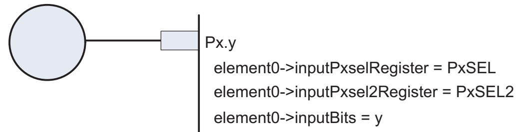

图 28. RO_PINOSC_TA0 元件和传感器定义

A.1.1 RO_PINOSC_TA0_WDTp

```
const struct Element element0 = {
               .inputPxselRegister = (uint8_t *)&P2SEL,
               .inputPxsel2Register = (uint8_t *)&P2SEL2,
               .inputBits = BIT3,
               .maxResponse = 400,
               .threshold = 50
 };

 const struct Sensor group =
                {
                   .halDefinition = RO_PINOSC_TA0_WDTp,
                   .numElements = 1,
                   .baseOffset = 0,
                   // Pointer to elements
                   .arrayPtr[0] = &element0, // point to first element
                   // Timer Information
                   .measGateSource= GATE_WDTp_SMCLK,      // 0->SMCLK, 1-> ACLK
                   .accumulationCycles= WDTp_GATE_8192    // 8192 SMCLK cycles in gate time
                 };
```

A.1.2 RO_PINOSC_TA0

```
const struct Element element0 = {
               .inputPxselRegister = (uint8_t *)&P2SEL,
               .inputPxsel2Register = (uint8_t *)&P2SEL2,
               .inputBits = BIT3,
               .maxResponse = 200,
               .threshold = 80
};

const struct Sensor group =
               {
                  .halDefinition = RO_PINOSC_TA0,
                  .numElements = 1,
                  .baseOffset = 0,
                   // Pointer to elements
                  .arrayPtr[0] = &element0,  // point to first element
                   // Timer Information
                  .accumulationCycles = 20   // Number of ACLK cycles in gate time
               };
```


版权 © 2011, Texas Instruments Incorporated

A.1.3 fRO_PINOSC_TA0_xx

```
const struct Element element0 = {
                .inputPxselRegister = (uint8_t *)&P2SEL,
                .inputPxsel2Register = (uint8_t *)&P2SEL2,
                .inputBits = BIT3,
                .maxResponse = 200,
                .threshold = 80
 };

const struct Sensor group =
                {
                   .halDefinition = fRO_PINOSC_TA0_SW,
                   .numElements = 1,
                   .baseOffset = 0,
                    // Pointer to elements
                   .arrayPtr[0] = &element0,     // point to first element
                    // Timer Information
                   .accumulationCycles = 1000    // Number of RO cycles in gate time
                };
```

A.2 *xx_COMPAp_TAx_xx*

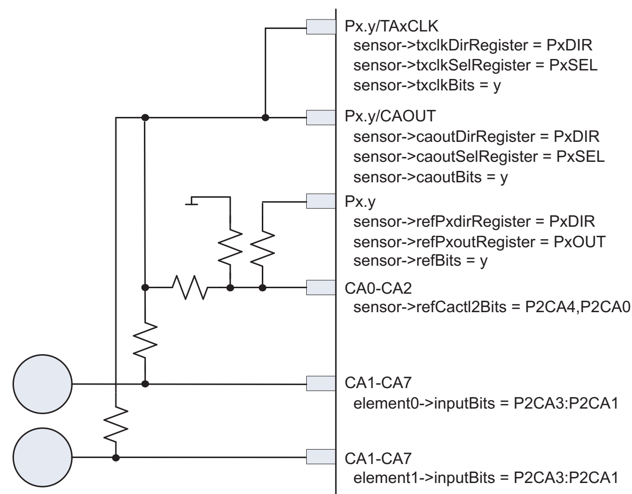

图 29. RO_COMPAp_TAx 元件和传感器定义

版权 © 2011, Texas Instruments Incorporated

A.2.1 RO_COMPAp_TAx_xx

```
const struct Element element0 = {
               .inputBits = P2CA2, // CA2
               .maxResponse = 250,
               .threshold = 100
 };

const struct Sensor group =
               {
                 .halDefinition = RO_COMPAp_TA0_WDTp,
                 .numElements = 1,
                 .baseOffset = 0,
                  // Pointer to elements
                 .arrayPtr[0] = &element0,
                  // Reference Information
                  // CAOUT is P1.7
                  // TACLK is P1.0
                 .caoutDirRegister = (uint8_t *)&P1DIR, // PxDIR
                 .caoutSelRegister = (uint8_t *)&P1SEL, // PxSEL
                 .txclkDirRegister = (uint8_t *)&P1DIR, // PxDIR
                 .txclkSelRegister = (uint8_t *)&P1SEL, // SxSEL
                 .caoutBits = BIT7, // BITy
                 .txclkBits = BIT0,
                 .refPxoutRegister = (uint8_t *)&P1OUT,
                 .refPxdirRegister = (uint8_t *)&P1DIR,
                 .refBits = BIT6,             // BIT6
                 .capdBits = BIT1+BIT2,       // BIT1,2
                 .refCactl2Bits = P2CA4,      // CACTL2-> P2CA4 , CA1
                  // Timer Information
                 .measGateSource= GATE_WDTp_SMCLK,    // 0->SMCLK, 1-> ACLK
                 .accumulationCycles= WDTp_GATE_512   // gate time is 512 SMCLK cycles
               };
```

A.2.2 fRO_COMPAp_TAx_xx

```
const struct Element element0 = {
.inputBits = P2CA2, // CA2
 .maxResponse = 230,
.threshold = 80
 };

const struct Sensor group =
{
.halDefinition = fRO_COMPAp_TA0_SW,
.numElements = 1, .baseOffset = 0,
 // Pointer to elements
.arrayPtr[0] = &element0,
// Reference Information
// CAOUT is P1.7
// TACLK is P1.0
.caoutDirRegister = (uint8_t *)&P1DIR, // PxDIR
 .caoutSelRegister = (uint8_t *)&P1SEL, // PxSEL
 .txclkDirRegister = (uint8_t *)&P1DIR, // PxDIR
.txclkSelRegister = (uint8_t *)&P1SEL, // SxSEL
.caoutBits = BIT7, // BITy
.txclkBits = BIT0,
.refPxoutRegister = (uint8_t *)&P1OUT,
 .refPxdirRegister = (uint8_t *)&P1DIR,
.refBits = BIT6, // BIT6
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

```
 .capdBits = BIT1+BIT2, // BIT1,2
 .refCactl2Bits = P2CA4, // CACTL2-> P2CA4 , CA1
// Timer Information
 .accumulationCycles = 1000
 };
```

A.3 fRO_COMPAp_SW_xx

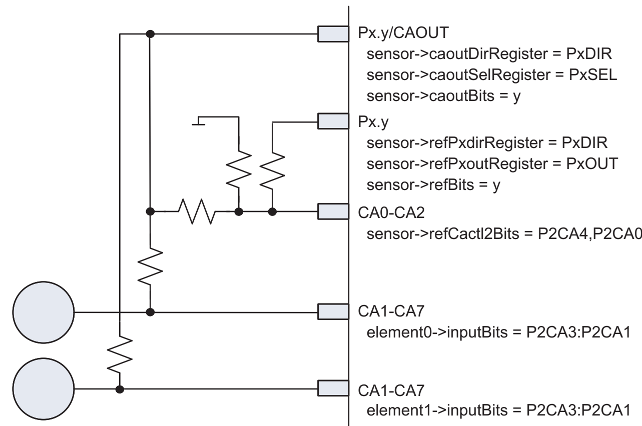

图 30. fRO_COMPAp_TAx 元件和传感器定义

```
const struct Element element0 = {
               .inputBits = P2CA2, // CA2
               .maxResponse = 180,
               .threshold = 60
};

const struct Sensor group =
               {
                  .halDefinition = fRO_COMPAp_SW_TA0,
                  .numElements = 1,
                  .baseOffset = 0,
                   // Pointer to elements
                  .arrayPtr[0] = &element0,
                  // Reference Information
                  // CAOUT is P1.7
                  .caoutDirRegister = (uint8_t *)&P1DIR, // PxDIR
                  .caoutSelRegister = (uint8_t *)&P1SEL, // PxSEL
                  .caoutBits = BIT7, // BITy
                  .refPxoutRegister = (uint8_t *)&P1OUT,
                  .refPxdirRegister = (uint8_t *)&P1DIR,
                  .refBits = BIT6,         // BIT6
                  .capdBits = BIT1+BIT2,   // BIT1,2
                  .refCactl2Bits = P2CA4, // CACTL2-> P2CA4 , CA1
                   // Timer Information
                  .accumulationCycles = 1000 // number of RO cycles in gate time
               };
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

A.4 xx_COMPB_TAx_xx

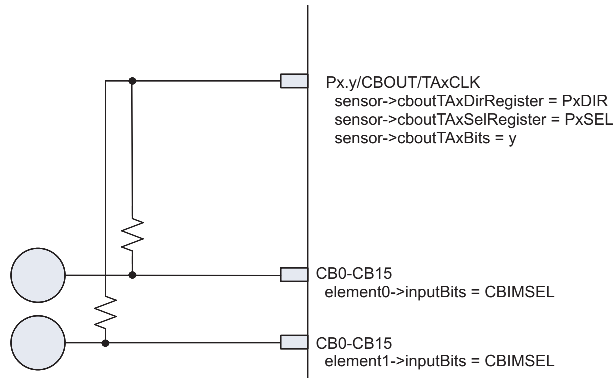

图 **31. RO_COMPB_TAx** 元件和传感器定义

A.4.1 RO_COMPB_TAx_xx

```
const struct Element element0 = {
               .inputBits = CBIMSEL_2, // CB2
               .maxResponse = 310,
               .threshold = 240
 };

 const struct Sensor group =
                {
                   .halDefinition = RO_COMPB_TA0_WDTA,
                   .numElements = 1,
                   .baseOffset = 0,
                   .cbpdBits = (BIT2),
                   // Pointer to elements
                   .arrayPtr[0] = &element0, // point to first element
                   .cboutTAxDirRegister = (uint8_t *)&P3DIR, // PxDIR
                   .cboutTAxSelRegister = (uint8_t *)&P3SEL, // PxSEL
                   .cboutTAxBits = BIT4, // P3.4
                   // Timer Information
                   .measGateSource= GATE_WDTA_SMCLK, // 0->SMCLK, 1-> ACLK
                   .accumulationCycles= WDTA_GATE_512K //
                 };
```

A.4.2 fRO_COMPB_TAx_xx

```
const struct Element element0 = {
                .inputBits = CBIMSEL_2, // CB2
                .maxResponse = 250,
                .threshold = 160
};

 const struct Sensor group =
                {
                   .halDefinition = fRO_COMPB_TA0_SW,
                   .numElements = 1,
                   .baseOffset = 0,
                   .cbpdBits = (BIT2),
                   // Pointer to elements
                   .arrayPtr[0] = &element0, // point to first element
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

```
        .cboutTAxDirRegister = (uint8_t *)&P3DIR, // PxDIR
        .cboutTAxSelRegister = (uint8_t *)&P3SEL, // PxSEL
        .cboutTAxBits = BIT4, // P3.4
        // Timer Information
        .accumulationCycles= 1000 /
    };
```

A.5 RC_PAIR_TAx

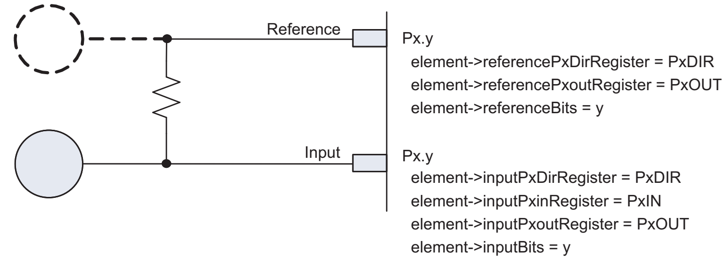

图 **32. RC_PAIR_TAx** 元件和传感器定义

```
const struct Element element0 = {
               .inputPxinRegister = (uint8_t *)&P2IN,
               .inputPxoutRegister = (uint8_t *)&P2OUT,
               .inputPxdirRegister = (uint8_t *)&P2DIR,
               .inputBits = BIT0,
               .referencePxoutRegister = (uint8_t *)&P2OUT,
               .referencePxdirRegister = (uint8_t *)&P2DIR,
               .referenceBits = BIT1,
               .threshold = 100,
               .maxResponse = 200

};

 const struct Sensor group =
                {
                    .halDefinition = RC_PAIR_TA0,
                    .numElements = 1,
                    .baseOffset = 0,
                    // Pointer to elements
                    .arrayPtr[0] = &element0,              // point to first element
                    // Timer Information
                    .accumulationCycles= 4                 // 4 charge/discharge cycles
                };
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

附录 B　电容触摸传感器层详细说明

B.1　电容触摸传感器层

电容触摸传感器层执行几个函数并可被分成几个层。 顶部（与真实应用最接近）为表示层。 根据被调用的 API，元件信息被转化成所需的格式或者“呈现”给应用层。 与表示层密切相关的是主键检测。 这个函数与一个 API 调用无关，但是它被应用层用来确定主元件。 最后的两个层是定制或者变量增量层以及原始电容层。 API 可直接调用进定制层或者原始层，并且细化和解释被提升至应用层。 在定制 API 调用的情况下，函数管理基极电容跟踪并提供电容变化或者当前测量和跟踪基极电容之间的变量增量。 RAW API 调用只提供原始即时电容测量，而无需基极电容的任何调整或者考虑。

B.1.1　状态/基线控制寄存器

```
uint16 ctsStatusReg=0;
```

15	14	13	12	11	10	9	8
未使用							

7	6	5	4	3	2	1	0
TRADOI		**TRIDOI**		未使用	**PAST_EVNT**	**DOI**	**EVNT**

未用	位 15-8	未使用
TRADOI	位 7-6	跟踪速率与所需方向相反 00　极慢 01　慢 10　中 11　快
TRIDOI	位 5-4	跟踪速率与所需方向相同 00　极慢 01　慢 10　中 11　快
未使用	位 3	未使用
PAST_EVNT	位 2	之前的事件。 在之前扫描上发生的一个事件 0　在之前的扫描上无事件出现 1　在之前扫描上发生的一个事件
DOI	位 1	所需方向 0　增加的电容 1　减少的电容
EVNT	位 0	事件。 组中的一个元件已经检测到一个阈值交叉 0　无事件出现 1　一个事件发生

图 33. 状态/基线控制寄存器的说明 (RAM)

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

B.1.2 基线跟踪

增量计算是当前测量和之前电容测量或者基线电容（代码环境中的 baseCnt）之间的差异。 基线电容被监视或者跟踪以解决任一环境变化，这些环境变化将影响进行电容测量的机制。 这包括但不只限于 Vcc，温度，和湿度。

B.1.2.1 所需方向

如之前提到的那样，电容增加的表示是 RC 和 快速 RO 方法的计数增加，和 RO 方法的计数减少。 如果应用正在寻找电容的增加或者减少，识别一个所需方向的目的是为了建立。 在大多数人机接口应用中，所需方向是电容的增加。 一个手指或者触摸的出现会增加一个元件的电容。 电容的增加也可由环境因素引起，但是认为这些变化与人机交互相比相对较慢。 电容的变化与所需方向一致，但是变化幅度并未大到足够超过此阀值，此阀值也许会由于环境发生变化。 这将要求基极电容的更新。 为了确保这些变化并不是由一个缓慢移动对象引起的，建议非常缓慢的调整所需方向。 选择调整速率的折中方法是为了解决缓慢移动对象和快速环境变化的问题。

与所需方向相反的电容变化通常只表示一个环境变化。 由于变化绝对是由环境导致的，可以更加大幅度地调整基线来解决此变化。 变量电容函数将结果设定为 0 来表明与所需方向相反的电容的变化。

B.1.2.2 所需方向的示例

一个应用需要检测何时一个木块被放置，然后被移除。 这个块通常在几天内被保留在这个位置。 所需方向的电容增加表明何时木块就位，然后所需方向的电容减少表明何时木块被移除。 一旦木块就位，任何电容的额外增加被当成一个与所需方向相反的变化并且基线被相应地更新。 用同样的方法，在木块被移除后，如果电容减少，则此减少被当成与所需方向相反的变化。

B.1.2.3 更新基线电容

增量计算会产生一个零值（代表一个与所需方向相反或者相对的变化）、一个少于阀值的非零值、或者一个大于阀值的值。 如图 34所示，按照 TRADIO 设置，一个零值会引起一个基线更新。

超过阀值的增量值表明一个事件。 当一个事件发生时，传感器内的其它元件有可能被激活，即使数量很少。因此当一个传感器内一个事件发生时，有必要暂停所需方向的基线更新。

只有当增量计算结果为非零并且少于阀值时，与所需方向一致的基线更新才会发生。 之前的事件标志，PAST_EVNT，表明一个传感器内的其中一个元件已经经历了一个阀值交叉。 如果 PAST_EVNT 标志为真，那么基线不应被更新，这是因为增加可能是由一个相邻器件上的触摸引起的。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

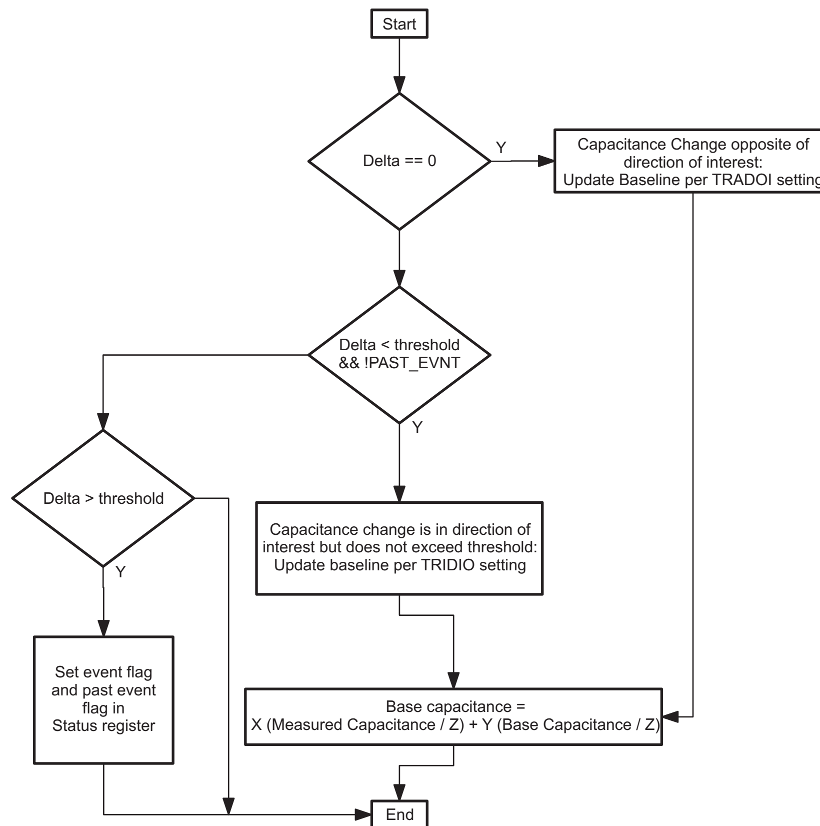

图 **34.** 基极电容更新

表 **22.** 与所需方向相反的跟踪设置

正在设置	说明	RO（测量值<基值）	RC，快速 RO（测量值>基值）
3	快	基值=测量值 / 2 + 基值 / 2	基值=测量值 / 2 + 基值 / 2
2	中	基值=测量值 / 4 + 3 x 基值 / 4	基值=测量值 / 4 + 3 x 基值 / 4
1	慢	基值=测量值 / 64 + 63 x 基值 / 64	基值=测量值 / 64 + 63 x 基值 / 64
0	极慢（缺省值）	基值=测量值 / 128 + 127 x （基值 / 128）	基值=测量值 / 128 + 127 x （基值 / 128）

表 **23.** 所需方向的跟踪设置

设置	说明	RO（测量值<基值）	RC，快速 RO（测量值>基值）
3	快	基值 = 3 x 测量值 / 4 + 基值 / 4	基值 = 3 x 测量值 / 4 + 基值 / 4
2	中	基值=测量值 / 2 + 基值 / 2	基值=测量值 / 2 + 基值 / 2
1	慢（缺省值）	减量基值	增量基值
0	极慢	只有测量值和基值之间的差异大于 15 时减少基值	只有测量值和基值之间的差异大于 15 时增加基值

B.1.3 测量函数

B.1.3.1 增量测量 + 基值电容跟踪：定制 API 调用

“定制”API 测量一个给定结构的元件的电容的变化。 针对定制 API 函数的输入是到传感器的指针以及到排列的第一个元件的指针，在此排列中电容变化被记录。 定制 API 调用用“原始”函数测量每个元件的电容。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

B.1.3.2 增量计算

HAL 定义 (hal_definition) 和所需方向确定了增量计算。 hal_definition 被有序排列，这样所有少于 64 的值所用方法的计数值与电容的变化直接相关（也就是说，计数的增加意味着电容的增加），当 hal_definition 大于 64 时情况相反（即，电容的增加会导致计数减少）。 当使用 RC 和快速扫描 RO 方法时，电容的增加由一个计数增加表示。 相反地，当使用 RO 方法时，电容的增加由一个计数减少表示。

在定制 API 中执行的增量计算产生一个 0 或者非零值。 非零值只是已测得电容和一个给定元件基线电容间的差异。 值为 0 的 deltaCnt 表示与所需方向相反（相对）的电容变化。

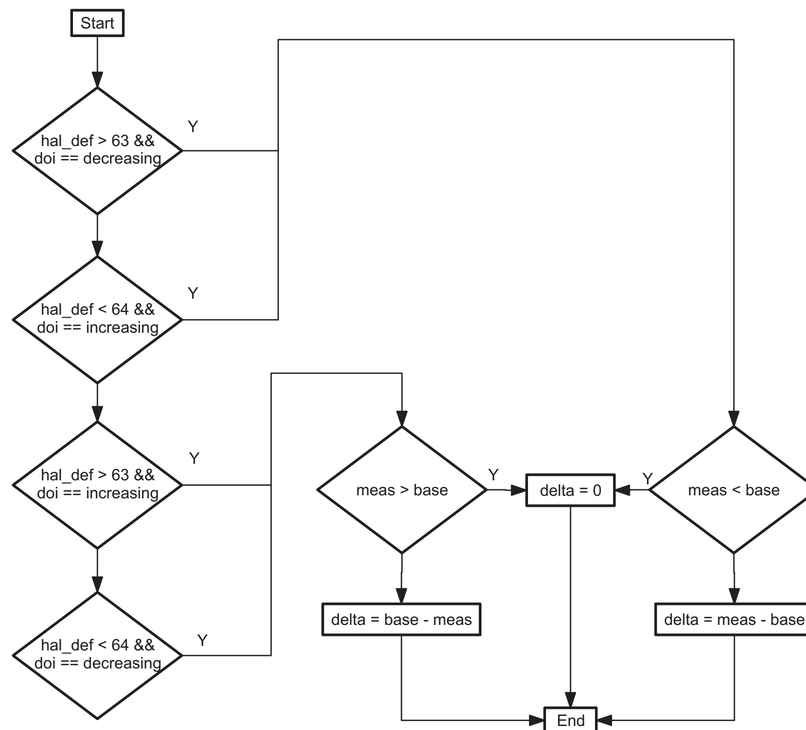

图 **35.** 增量测量

B.1.3.3 原始电容测量：原始 API 调用

RAM 测量函数的唯一用途是为了根据用户配置来调用适当的 HAL 函数。 这个函数更新函数调用内提供的 RAM 变量，由更高级的调用函数所使用。

“原始”特性调用适当的 HAL 函数。 针对一组元件的 HAL 定义可在传感器结构中找到。

```
sensor0.halDefinition
```


SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

hal_definition 代表一个完成电容式触摸功能的 MSP430 外设的组合。

表 **24. HAL** 定义

名称	类型	编号	测量硬件	测量时间	选通时间
RC_PAIR_TA0	RC	1	数字 I/O	Timer_A0	不可用
RC_SINGLE_TA0	RC	2	数字 I/O	Timer_A0	不可用
RC_PAIR_TA1	RC	3	数字 I/O	Timer_A1	不可用
RC_PAIR_TD0	RC	4	数字 I/O	Timer_D0	不可用
fRO_PINOSC_TA0_SW	fRO	25	数字 I/O	Timer_A0	软件
fRO_COMPB_TA0_TA1	fRO	26	Comp_B	Timer_A1	Timer_A0
fRO_COMPB_TA0_TD0	fRO	27	Comp_B	Timer_D0	Timer_A0
RO_COMPAp_TA0_WDTp	RO	64	Comp_A+	Timer_A0	WDT+
RO_PINOSC_TA0_WDTp	RO	65	数字 I/O	Timer_A0	WDT+
RO_PINOSC_TA0	RO	66	数字 I/O	Timer_A0	ACLK
RO_COMPAp_TA1_WDTp	RO	67	Comp_A+	Timer_A1	WDT+
RO_COMPB_TA0_WDTA	RO	68	Comp_B	Timer_A0	WDTA
RO_COMPAp_TA0_SW	RO	69	Comp_A+	Timer_A0	软件
RO_PINOSC_TA0_SW	RO	70	数字 I/O	Timer_A0	软件

B.1.4 传感器抽象

B.1.4.1 按钮

```
uint8_t TI_CAPT_Button(Sensor *);
```

输入：到传感器的指针，定义了按钮

输出：0/1，

功能：测量按钮。 一个 0 意味着电容的变化少于或者等于传感器内设定的阀值，而 1 意味着电容改变已经超过阀值。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

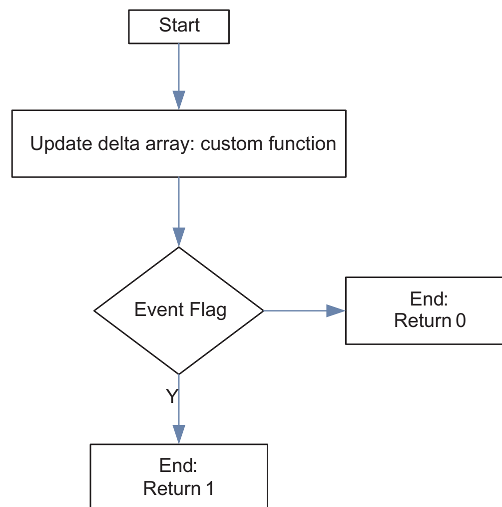

图 **36.** 单按钮算法

Element * TI_CAPT_ Buttons(Sensor *);

输入：到传感器的指针，定义了一组元件，在这里每个元件代表一个按钮

输出：到一个元件结构的指针

功能：这个函数返回值为指向元件的结构指针，元件超过了其阀值最大裕量：对于每个元件被标准化至（maxResponse - 阀值值）。 如果没有按钮超过其阀值（在元件结构中设定），那么这个函数返回一个 0 或者 'Null Pointer'。

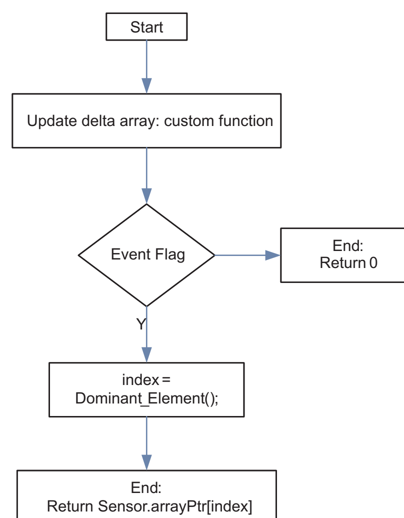

图 **37.** 按钮算法的排列

B.1.4.2 滑盖/滑轮

一个滑轮或者滑盖是一个由一组元件组成的传感器类型。 传感器被分成由用户定义的一定数量的点。 排列（第一个到最后一个）的方向被留给应用的解释。 从软件库的角度来讲，排列定义内的第一个元件与滑盖上的 '0' 值相对应，最后一个元件与用户定义的点数量相对应。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

针对滑盖和滑轮功能的算法显示在图 38中。 TI_CAPT_Custom 函数被用于测量传感器内定义的每个元件的电容的变化。 这个函数还更新基线跟踪和事件标志状态（请见节 B.1.3.1）。

为了滑轮和滑盖在传感器级提供额外的检测机制。 事件标志是确定元件级上阀值交叉的方法，而*sensorThreshold*变量提供了一个传感器级上的阀值。 这个机制的用于区分一个和传感器的真正交互与一个无意交互，而此无意交互也许只激活一个元件。

最终滑盖和滑轮函数计算交互（触摸）发生的位置。 滑盖和滑轮要求四种类型的配置参数来表示一个位置。 这四个参数为可分辨点的数量、传感器级阀值、每个元件的元件级阀值、和每个元件的最大响应。

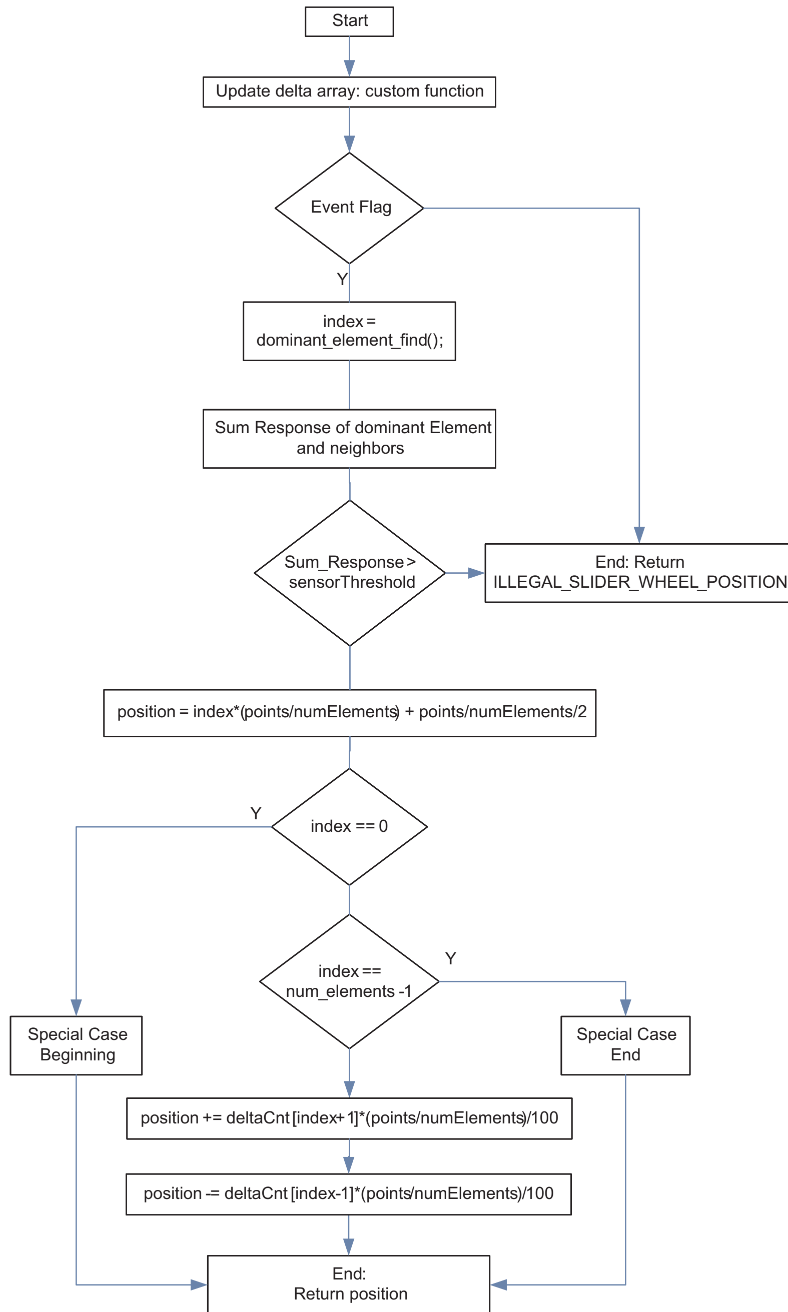

图 **38.** 滑盖**/**滑轮算法

版权 © 2011, Texas Instruments Incorporated

B.1.4.2.1 滑盖检测

滑盖 sensorThreshold 是一个主元件和其相邻元件响应的比较值。 如图 39所示，端点是一个特殊情况，这个情况只需要比较端元件（主元件）与一个相邻元件相比较。

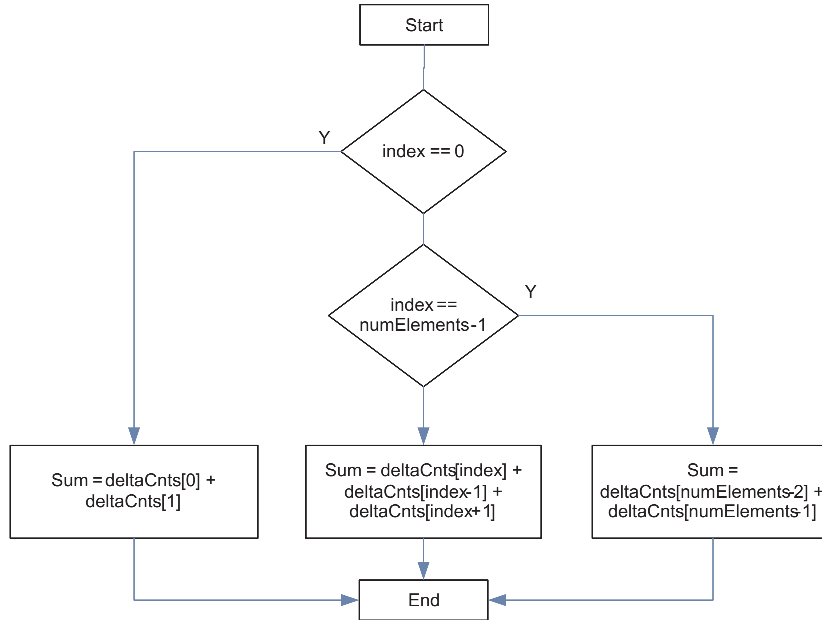

图 39. 滑盖阀值检测

由于交互移动（滑动）超过最后一个元件的中心，相邻元件的变化值为 '0' 并且传感器的阀值只是一个最后元件的函数。 因此 sensorThreshold 定义了手指可以从端器件的中心位置偏离多远并且仍然是滑盖的一部分。 只要传感器响应超过 sensorThreshold，位置被计算。

B.1.4.2.2 滑盖位置

在符合传感器阀值标准时，滑盖位置的计算被预测。 如果标准未被满足，那么函数只是返回一个预定义的值来表示没有检测到交互。 当有一个有效交互时，函数确定了来自主器件和其最近相邻元件的响应的位置。主元件功能为建立一个“基本”位置确定了中间元件，而一个或者两个相邻元件被用于拉或者倾斜至最终位置。 在图 40中的示例中，滑盖有 64 个位置并且在滑盖排列中有 4 个元件。

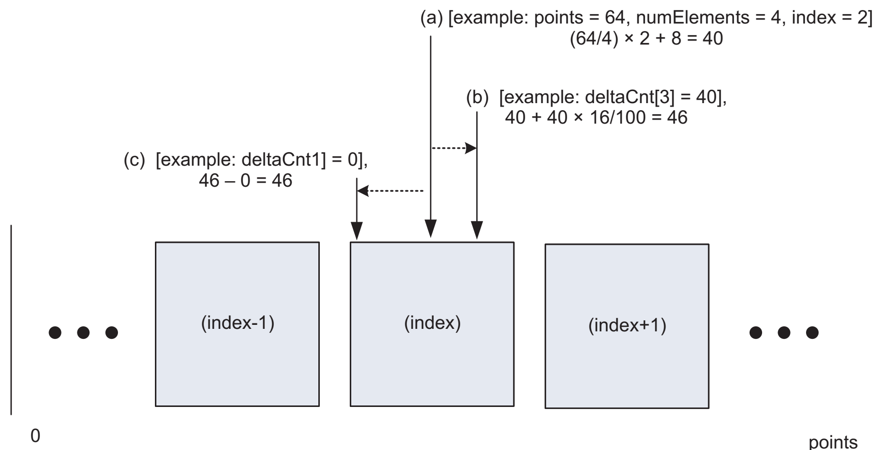

图 40. 滑盖/滑轮过程中算法

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

在特殊情况下，主元件为滑盖的开始或者末端元件，那么只有最近的相邻元件被用来倾斜或者影响位置。

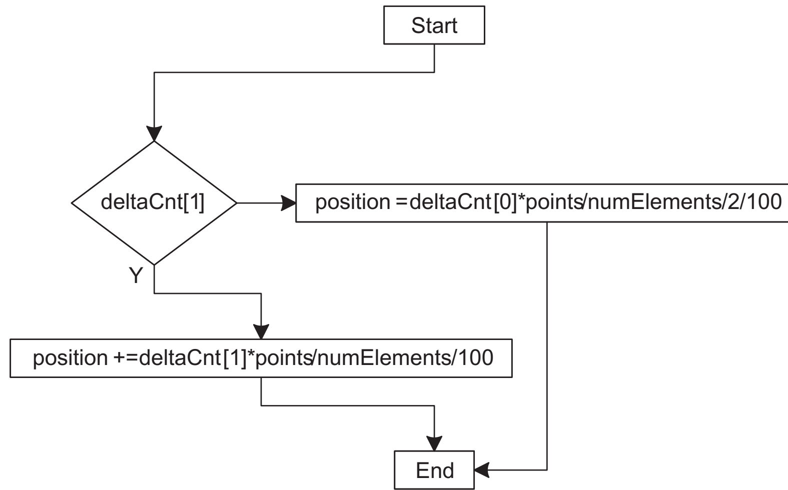

图 **41.** 滑盖算法：滑盖的起点

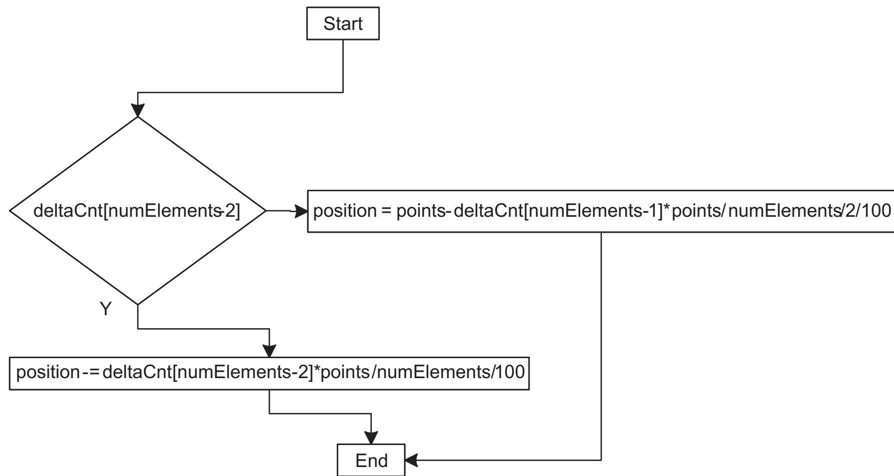

图 **42.** 滑盖算法：滑盖终点

版权 © 2011, Texas Instruments Incorporated

B.1.4.2.3 滑轮位置

滑轮 sensorThreshold 是与主元件和其相邻元件响应的相比较的值（x-1，x，和 x+1 的和）。 端点是一个特殊情况，要求将“环绕”考虑在内，请见图 43。 一旦被初始化，这三个元件的响应被相加，然后得出的值与 sensorThreshold 相比较。 如果超过阀值，那么函数继续计算位置。

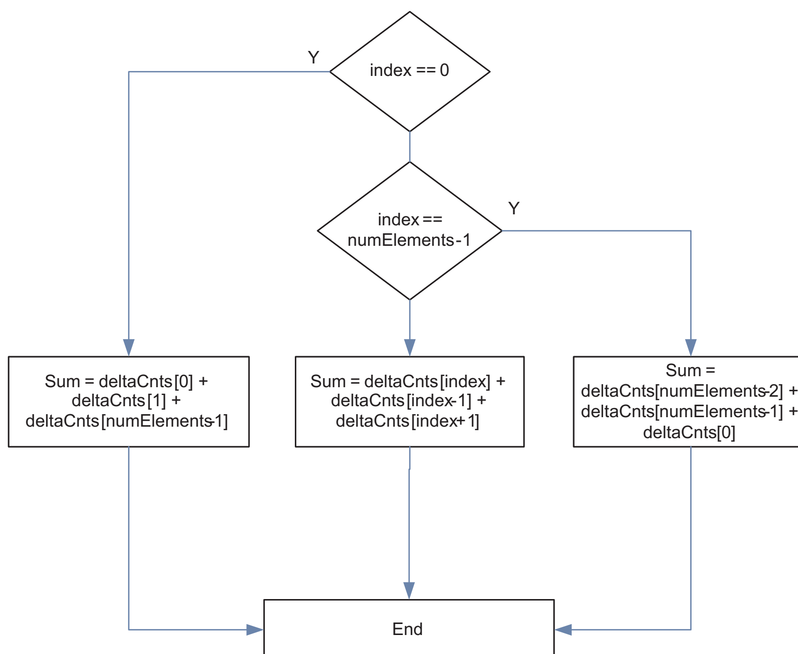

图 **43.** 滑轮阀值检测

版权 © 2011, Texas Instruments Incorporated

B.1.4.2.4 滑轮位置

入职前所提到的，滑轮只是滑盖的特殊情况。 需要采取额外的处理来解决从排列末端“环绕”回开始位置。当主元件为排列的开始和终止元件时，图 44和图 45显示了用来计算位置的算法。

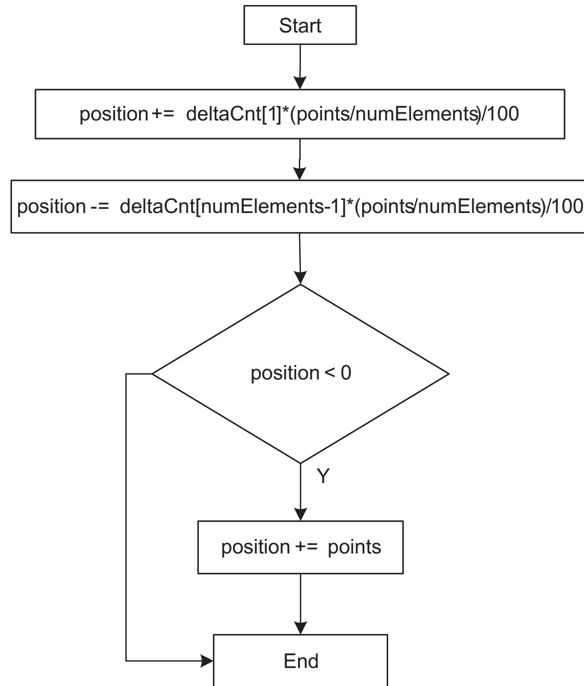

图 **44.** 滑轮算法：开始

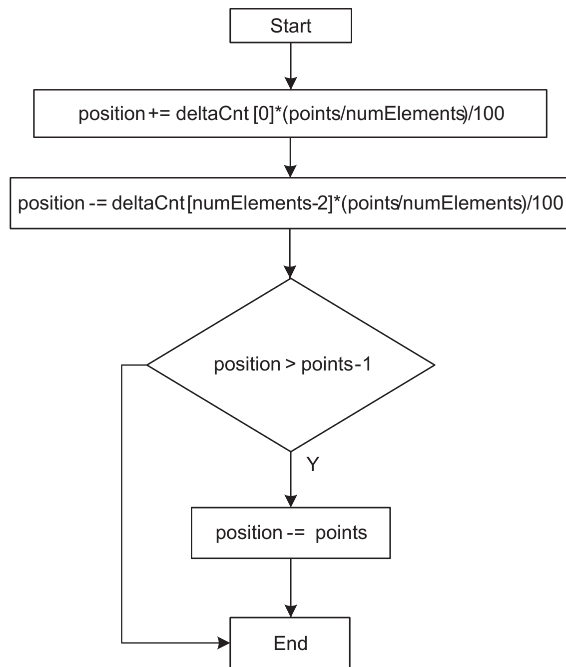

图 **45.** 滑轮算法：终止

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

B.1.5 主元件识别

一个阀值交叉的识别实际发生在基极电容更新函数内（请见节 B.1.3.1）。 当一个阀值交叉事件已经发生，那么下面的值用来确定传感器结构中的主元件并将响应在 0 至 100 的范围内缩放。 一个零将表示响应等于或者少于阀值，而 100 表示一个响应等于 maxResponse。

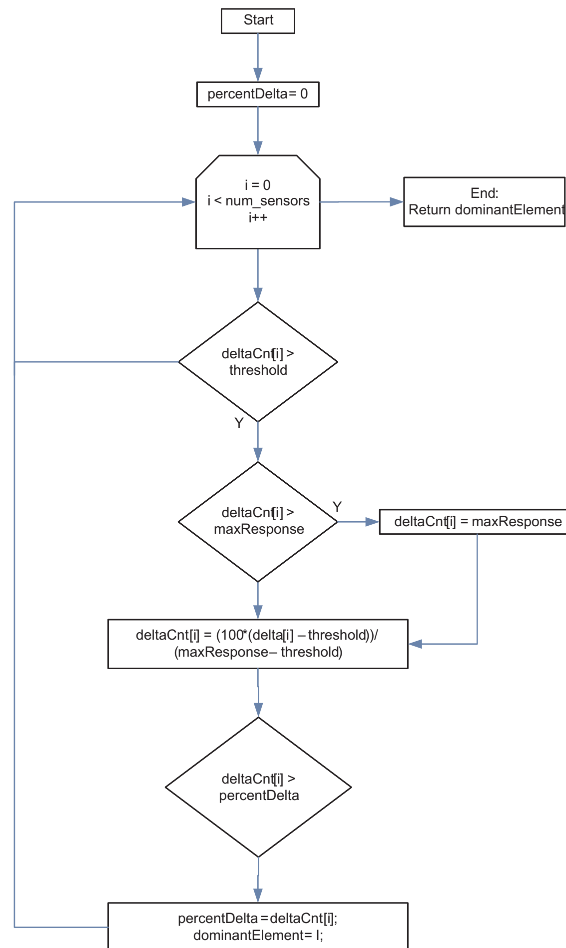

图 **46.** 主元件识别

版权 © 2011, Texas Instruments Incorporated

附录 C　Beta 测试

组合的数量、设置和应用的排列使一个完全综合性的测试计划变得不切实际。 此时，不能保证所有可能使用情况的功能。如果他们的应用还未经与软件库的兼容性测试的话，表 25被用作一个代理来帮助用户识别。例如，RO_PINOSC_TA0_WDTp 测量方法用定制的和滑轮 API 进行了测试。 这个方法也使用按钮 API 进行测试，而另外一个测量方法被用来支持滑轮 API。 按钮示例用于显示与不同测量方法和 API 调用的互操作性。

表 25. Beta 测试

	HAL	可用	传感器	额外 HAL [(1)]	额外传感器 [(2)]
1	TI_CTS_RO_COMPAp_TA0_WDTp_HAL	TI_CAPT_Custom	TI_CAPT_Slider		
2	TI_CTS_fRO_COMPAp_TA0_SW_HAL	TI_CAPT_Custom	TI_CAPT_Slider		
3	TI_CTS_fRO_COMPAp_SW_TA0_HAL	TI_CAPT_Custom	TI_CAPT_Slider		
4	TI_CTS_RO_COMPAp_TA1_WDTp_HAL	TI_CAPT_Custom			
5	TI_CTS_fRO_COMPAp_TA1_SW_HAL	TI_CAPT_Custom			
6	TI_CTS_RC_PAIR_TA0_HAL	TI_CAPT_Custom			
7	TI_CTS_RO_PINOSC_TA0_WDTp_HAL	TI_CAPT_Custom	TI_CAPT_Wheel	6	按钮
8	TI_CTS_RO_PINOSC_TA0_HAL	TI_CAPT_Custom	TI_CAPT_Wheel	7	按钮
9	TI_CTS_fRO_PINOSC_TA0_SW_HAL	TI_CAPT_Custom	TI_CAPT_Wheel	8	按钮
10	TI_CTS_RO_COMPB_TA0_WDTA_HAL	TI_CAPT_Custom	TI_CAPT_Slider	10	次滑盖
11	TI_CTS_fRO_COMPB_TA0_SW_HAL				
12	TI_CTS_RO_COMPB_TA1_WDTA_HAL	TI_CAPT_Custom	TI_CAPT_Button		
13	TI_CTS_fRO_COMPB_TA1_SW_HAL	TI_CAPT_Custom	TI_CAPT_Button		

(1) 额外的 HAL 显示 HAL 函数是独立的并且不会影响基线跟踪或者其它共用函数调用。

(2) 额外的传感器显示传感器函数是独立的，并且不会损坏其它也使用基线跟踪和其它函数的传感器。

SLAA490 — http://www-s.ti.com/sc/techlit/SLAA490
版权 © 2011, Texas Instruments Incorporated

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2015, Texas Instruments Incorporated