Texas Instrument 电容触摸应用库

By izmao

1 介绍

电容触摸软件库是一个可以更改的软件底层，可以快速的适应各种电容触摸板。本库的主要目的是提供对几个系列的 430 微处理器进行统一。

这个库提供了几个层以抽象源代码。最高层提供了一些标准的快速的简单的开发。底层的驱动用来提供写比较具体的比较独特的控制。

要使用这个库要有一定的测试信号的基本知识，例如配置一个用于特定系统的库，外围器件的使用，以及 API（软件接口）的调用。

这些代码的意图是要为电容触摸和其他电容类测量提供一个跳板。这些功能可能不是为了某个单一的应用而产生，里面包含了各种各样的应用。我们鼓励将不必要的代码删除掉。另外低功耗因为要配置 ISR 所以在库里面没有典型的应用。可能与现有的应用程序配置相冲突。此外，我们同样鼓励向代码中加入低功耗的功能。

2 基本原理

基本原理是通过两个独立的波形发生比较。一个信号源是不变的，另一个是由电容变化而变化的。

2.1 弛缓振荡器（RO）

弛缓振荡器方法是通过计数延迟振荡器在一个门限时间内的次数。（figure1）

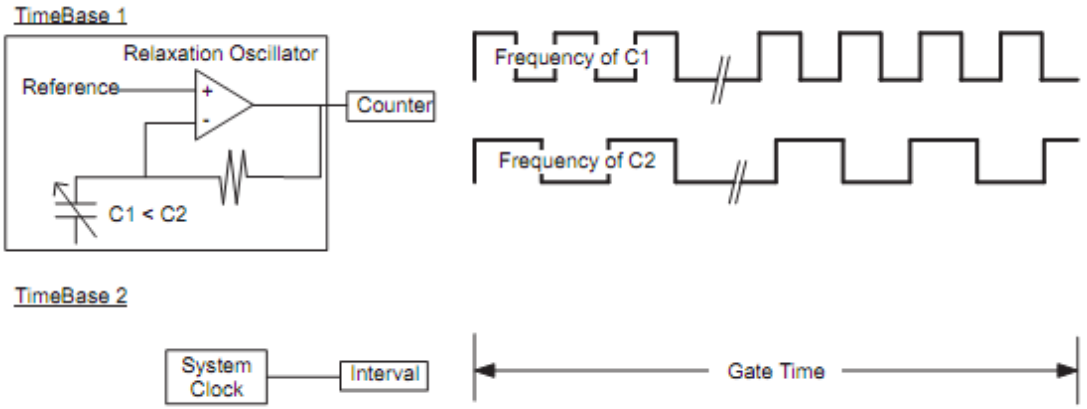

Figure 1. Relaxation Oscillator Measurement

迟缓振荡器法(RC)可以在有延迟振荡器或者比较器的 430 单片机上使用。频率是根据阻容震荡电路决定。电容会通过触摸而上升。时域内上升和下降时间都增加，频域内频率降低。电容的升高使得延迟振荡器的周期变短。

命名规则：

Table 1. Relaxation Oscillator Naming Convention

Name	RO Mechanism	Counter	Gate Period
RO_XXX_YYY_ZZZ	XXX	YYY	ZZZ
RO_COMPAp_TA0_WDTp	Comparator A+	Timer_A0	Watchdog timer (interval mode)
RO_COMPB_TA1_WDTA	Comparator B	Timer_A1	Watchdog timer (interval mode)
RO_Pinosc_TA0(1)	Pin Oscillator	Timer_A0	'n' ACLK periods

(1) The RO_PINOSC_TA0 is a special case that takes advantage of the internal connection between ACLK and the Timer_A0 capture input. The user has the choice of simply dividing the ACLK in the application layer (by 2,4,8 and setting n to 1) or by entering a number of ACLK cycles, or both.

2.2 阻容时间常量的测量（RC）

RC 法和 RO 法师相互关联的。门限时间是可变的，数其中的脉冲数。

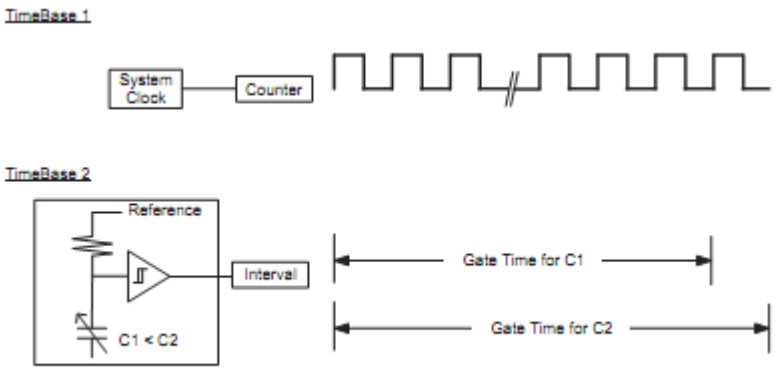

Figure 2. Resistor-Capacitor Time Constant Measurement (RC)

阻容式（RC）方法可以在任何 430 单片机上实现。门限时间通过电容充放电时间来决定。在这个可变的门限时间内计数器计数。电容增大（触摸时）门限时间增大。

2.3 快速扫描延迟振荡器（fRO）

fRO 模式类似于 RC 模式，但是门限时间是由弛缓振荡器产生替代充电和放电时间。如图三所示，被计数的脉冲仍然是固定不变的。

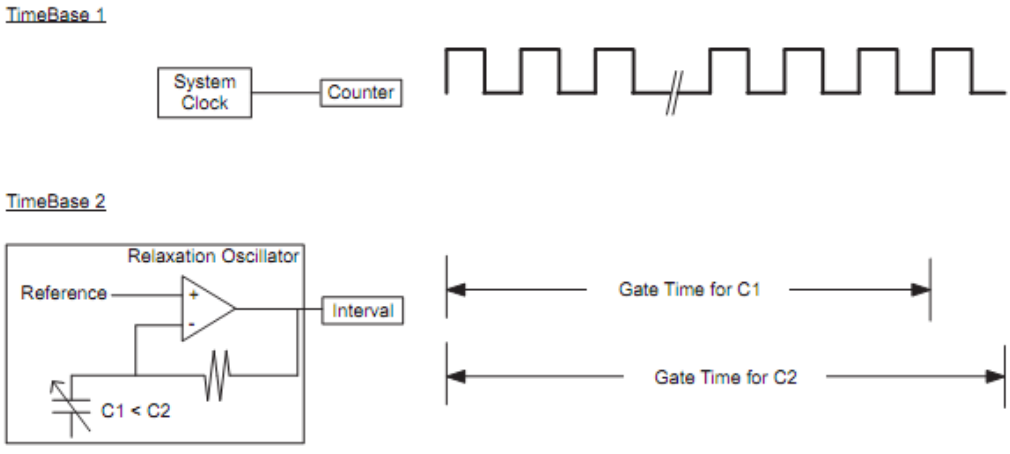

Figure 3. Fast Scan RO Measurement

库中命名规则：

Table 2. Relaxation Oscillator Naming Convention

Name	RO Mechanism	Counter	Gate Period
fRO_XXX_YYY_ZZZ	XXX	YYY	ZZZ
fRO_COMPB_TA1_SW	Comparator B	Timer_A1	Software loop
fRO_PINOSC_TA0_SW	Pin Oscillator	Timer_A0	Software loop

形如其名，fRO 的速度扫描很快速；同样灵敏度的情况下比 RO 模式快很多（由电容大小决定）。由 RO 模式原理可知，灵敏度与阀门时间有关——增加阀门时间可以增加灵敏度。但是增加阀门时间会减小扫描速度；单次测量需要更多的时间。为了保证 fRO 的灵敏度，单位时间脉冲必须有足够的分辨率。也就意味着在采样时间段内要有足够快的时钟源和更多的电量消耗。

3 配置

有两个文件用来配置主要的库：structure.c 和 structure.h。structure.c 包含了所有的元素和感应器（元素组）的定义。structure.h 用来为其他文件提供一个桥梁，并且里面包含了预处理，用以缩小代码和使能功能块。

3.1 元素定义

一个电容的感应元素是一个结构体，例如哪个电容代表那个事件（一次触摸，改变湿度，改变电解质等等）。一个元素可以被单独使用，例如用作一个单独的按键，或者联合其他的元素来穿件一个感应器；键盘，滑动条或者滑动轮（类似于 ipod classic 上面的感应按键）。如图 4

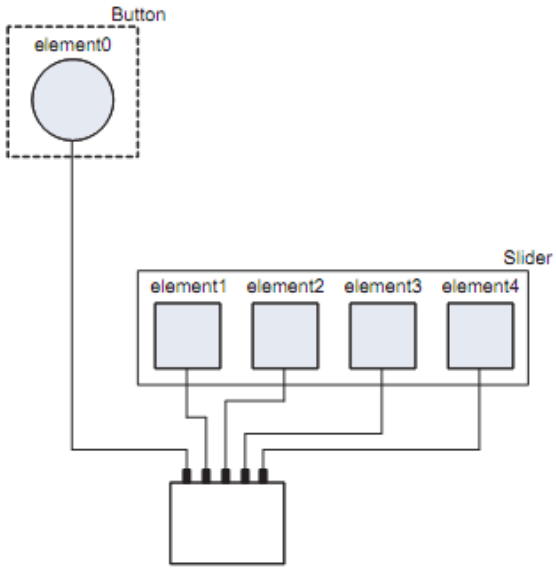

Figure 4. Elements

3.1.1 一般性定义

输入位（InputBits）是一个用来描述 GPIO Px.y 中的 y 的一般性定义或者比较器 A 比较器 B 中的 mux 端。

门限（threshold）限制了电容的变化量不会超过触摸发生时的量。

最大反应（maxResponse）仅用于带有多元素的感应器所希望的最大反应值，例如按键，滑动轮，滑动条等。maxResponse 主要是用来将感应量量化成百分数，用来识别一个感应器中有门限交叉的元素。

图五表示了在触摸按钮应用中测量参量，门限和最大反应（maxResponse）之间的关系。门限和最大反应间被限制于一个 16 位数（0-65535）。这些值在按钮滑动条和滑动轮使用时被进一步限制为：maxResponse-threshold<655。第六章有更多的细节介绍关于建立一个测量参量。

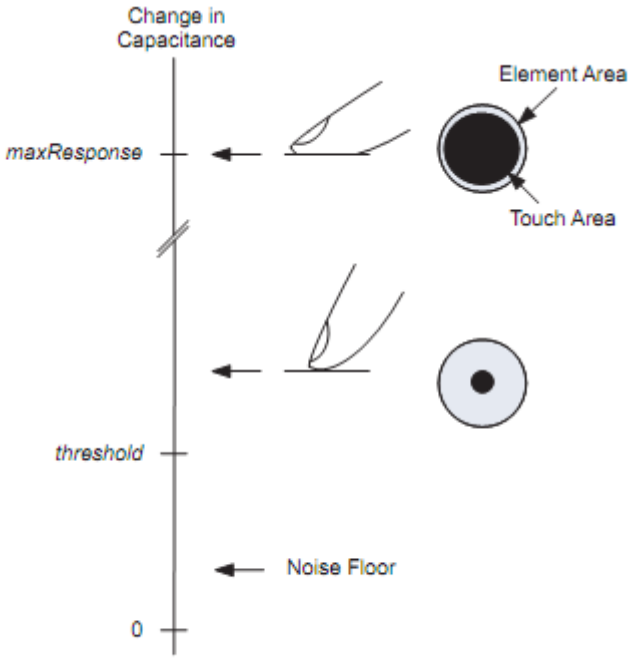

Figure 5. Element Measurement Parameters: Buttons Example

3.1.1.1 使用比较器 A+

用比较器 A+创建一个 RO（延迟振荡器），用同样的元素结构体格式定义。

InputBits 可以识别 在 CACTL2 寄存器内的 P2CA1,P2CA2,和 P2CA3。这些位代表了比较器反向输入端。输入直接接入电极。参考输入定义在感应器章节。

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

3.1.1.2 使用比较器 B

InputBits 识别 CBCTL0 中的 CBIMSEL 位。

15	14	13	12	11	10	9	8
CBIMEN	Reserved			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	Reserved			CBIPSEL			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

CBIMEN	Bit 15	Channel input enable for the V– terminal of the comparator. 0 Selected analog input channel for V– terminal is disabled. 1 Selected analog input channel for V– terminal is enabled.
Reserved	Bits 14-12	Reserved
CBIMSEL	Bits 11-8	Channel input selected for the V– terminal of the comparator if CBIMEN is set to 1.
CBIPEN	Bit 7	Channel input enable for the V+ terminal of the comparator. 0 Selected analog input channel for V+ terminal is disabled. 1 Selected analog input channel for V+ terminal is enabled.
Reserved	Bits 6-4	Reserved
CBIPSEL	Bits 3-0	Channel input selected for the V+ terminal of the comparator if CBIPEN is set to 1.

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

3.1.1.3 使用 PinOsc

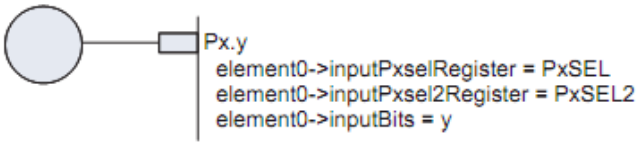

Figure 7. PinOsc Port Parameters

```
const struct Element right = {

              .inputPxselRegister = (uint8_t *)&P2SEL,
              .inputPxsel2Register = (uint8_t *)&P2SEL2,

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

3.1.1.4 使用 RC

使用 RC 要比较两个 GPIO。一个是输入另一个是参考。

By izmao

```
//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
};
```