TEXAS INSTRUMENTS

Programmer's Guide

SLAA490A – April 2011 – Revised December 2012

Capacitive Touch Software Library

MSP430™ microcontrollers offer a number of peripherals that, when configured properly, can be used to perform a capacitance measurement. The purpose of the capacitive touch library is to create a single interface that can be integrated with the peripheral sets found in the Value Line, 2xx, 5xx, and FR59xx product families. This document explains the capacitive touch library's configuration and use.

The software library described in this document can be downloaded from http://www.ti.com/tool/capsenselibrary.

Contents

1 Introduction 4
2 Methods 4
3 Configuration 8
4 Resources 35
5 API Calls 37
6 Establishing Measurement Parameters 42
7 CTS_Layer.c and CTS_Layer.h Detailed Description 47

List of Figures

1 Relaxation Oscillator Measurement 5
2 Resistor-Capacitor Time Constant Measurement (RC) 6
3 RC Schematic 6
4 Fast Scan RO Measurement 7
5 Elements 8
6 Element Measurement Parameters: Buttons Example 9
7 CBCTL0 Register 10
8 RO_PinOsc Schematic 11
9 PinOsc Port Parameters 11
10 Capacitive Touch IO Implementation 11
11 CAPSIOxCTL Register 12
12 RC IO Parameters 13
13 Sensor Example 14
14 RO_COMPB Port Parameters 15
15 Port Parameters for RO_COMPAp_TAx Implementation 16
16 fRO_COMPAp_SW_TAx General Description 17
17 RO_COMPAp_TAx Schematic Description 19
18 Timing Parameters: Example WDTp 20
19 RO_COMPB Schematic 21
20 RO_PinOsc_TA0 Timing Parameter 25
21 RC Timing Parameters 27
22 fRO_CSIO_TA2_TA3 Timing Parameters 28
23 fRO_PinOsc_TA0_TA1 Timing Parameters 29
24 fRO_PinOsc_TA0 Timing Parameters 30

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

Copyright © 2011–2012, Texas Instruments Incorporated

25 fRO_COMPB_TA1_TA0 Timing Parameters 31
26 fRO_COMP_B_TAx_SW Timing Parameters 33
27 fRO_COMPAp_TA0_SW Timing Parameters 34
28 Slider Example 39
29 Wheel Example 40
30 Custom Slider Example 40
31 Measurement Example of a Four-Element Sensor 44
32 Valid Slider Locations as a Function of the Sensor Threshold 46
33 Status/Baseline Control Register (RAM) 47
34 Base Capacitance Update 49
35 Delta Measurement Block Diagram 50
36 Single Button Algorithm 51
37 Array of Buttons Algorithm 52
38 Slider and Wheel Algorithm 53
39 Slider Threshold Detection 54
40 Slider and Wheel Process Middle Algorithm 55
41 Slider Algorithm: Beginning of Slider 55
42 Slider Algorithm: End Of Slider 56
43 Wheel Threshold Detection 56
44 Wheel Algorithm: Beginning 57
45 Wheel Algorithm: Ending 58
46 Dominant Element Identification Function 59

List of Tables

1 Relaxation Oscillator Naming Convention 5
2 Fast RO Naming Convention 7
3 Comparison of fRO and RO Measurement Times 8
4 CBCTL0 Register Description 10
5 CAPSIOxCTL Register Description 12
6 baseOffset Description 17
7 halDefinition Descriptions 18
8 Watchdog Timer Source Select Definitions 19
9 Watchdog Timer+ Interval Select Definitions 20
10 Watchdog Source Select Definitions 22
11 Watchdog Interval Select Definitions 22
12 TimerA Source Select Definitions 23
13 TimerA Source Select Definitions 24
14 TimerA Source Select Definitions 26
15 measureGateSource Definitions for fRO_xxxx_SW_Txx 34
16 sourceScale Definitions for fRO_xxxx_SW_Txx 35
17 Gate Time Examples 36
18 Example Flash Resource Allocation 36
19 API Functions 38
20 Update Tracking Rate Format 42
21 Example Raw Results With RO Method 42
22 Example Change in Capacitance Results With RO method 43
23 Measurement Example of a Four-Element Sensor 45
24 Tracking Settings Against Direction of Interest 49
25 Tracking Settings in Direction of Interest 49

Copyright © 2011–2012, Texas Instruments Incorporated

26 Example HAL Definitions 51

Copyright © 2011–2012, Texas Instruments Incorporated

1 Introduction

The Capacitive Touch Software Library provides code that performs capacitance measurements on the MSP430 microcontroller platform using several combinations and permutations of peripherals. To simplify the use of capacitive touch sensing on the MSP430, the library provides a configuration structure that is specific to the implementation and API calls that are specific to the application.

The main purpose of this document is to explain the API calls and the configuration of the library. However, it is also important to have a basic understanding of the measurement implementations and how peripheral resources are used before using the library.

Section 2 describes the relaxation oscillator (RO), time constant (RC), and fast RO methods with the MSP430.

Section 3 describes the library configuration.

Section 4 describes peripheral resources.

Section 5 describes the APIs and the levels or degrees of abstraction with those APIs. The higher levels of abstraction provide standard controls for faster and easier development while the lower levels allow for customization and unique controls.

The associated code [(1)] is intended to be a starting point for developing capacitive touch and other capacitive measurement solutions. The library supports a wide variety of features and functions, all of which may not be required for a specific application. The source code is provided, and customers are encouraged to remove sections of code that are not used after creating a working application. Additionally, interrupt service routines (ISRs) are part of the source code, which may need additional code to support other applications. This is a common adjustment needed with WDT ISR. Again, as applications allow for shared ISR functionality, customers are encouraged to update the source code provided to support capacitive touch functionality in addition to other tasks.

2 Methods

For the methods described in this document, the fundamental principle is that two independent time bases are compared. One time base is fixed and the other is variable as a function of the capacitance. As long as the capacitance does not change, the relationship between the two remains constant. Timers are used to measure the two time bases, and the relationship is monitored in software. When the relationship changes, the software must decide if the change was an increase or decrease in the capacitance and if the change was significant enough to be considered a 'touch'.

The three methods discussed in this document are the relaxation oscillator (RO), the resistor capacitor (RC), and the fast relaxation oscillator (fRO).

(1) The software library described in this document can be downloaded from http://www.ti.com/tool/capsenselibrary.

Copyright © 2011–2012, Texas Instruments Incorporated

2.1 Relaxation Oscillator (RO) [2]

The RO method counts the number of relaxation oscillator cycles within a fixed period (gate time), as shown in Figure 1. The variable time base, TimeBase 1, is connected to a relaxation oscillator circuit. The relaxation oscillator can be realized with a comparator, the PinOsc feature found in the Value Line product family, or the Capacitive Touch IO found in the MSP430FR58xx and MSP430FR59xx family. The fixed time base, TimeBase 2, is connected to an internal MSP430 system clock such as the DCO or REFO. The gate time is a number of the system clock oscillations.

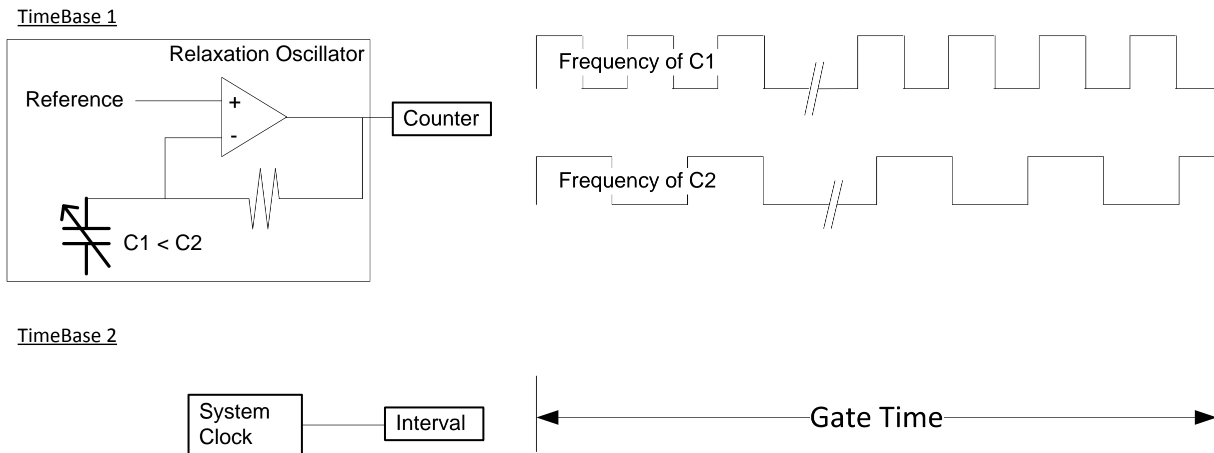


Figure 1. Relaxation Oscillator Measurement

The frequency of the relaxation oscillator is function of the capacitance of the circuit. As the capacitance increases (C1 to C2), the relationship between the TimeBase 1 and TimeBase 2 changes. The number of relaxation oscillator cycles counted within the gate time decreases. As the capacitance decreases (C2 to C1), the number of cycles within the gate time increases. Capacitance and counts have an inverse relationship in the RO method.

The naming convention for the RO method in the library identifies the relaxation oscillator mechanism, the timer that is used to measure or count oscillations, and the timer that is used to define the gate period.

(2) The RO method is described in more detail in *PCB-Based Capacitive Touch Sensing With MSP430* (SLAA363).

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+	TimerA0	Watchdog timer (interval mode)
RO_Pinosc_TA0	Pin Oscillator	TimerA0	'n' ACLK periods [1]
RO_CTIO_TA2_TA3	Touch Sense Enabled IO	TimerA2	TimerA3

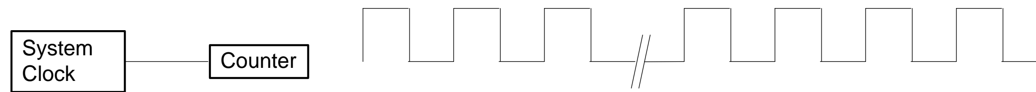
(1) The RO_PINOSC_TA0 is a special case that takes advantage of the internal connection between ACLK and the TimerA0 capture input. The user has the choice of simply dividing the ACLK in the application layer (by 2, 4, 8, and setting n to 1) or by entering a number of ACLK cycles, or both.

Copyright © 2011–2012, Texas Instruments Incorporated

2.2 Resistor-Capacitor Time Constant Measurement (RC) (1)

The RC method is the reciprocal of the RO method. As shown in Figure 2, the gate time is now variable, and the oscillator being counted is fixed. The fixed time base, TimeBase 1, is connected to an internal MSP430 oscillator, such as the DCO. The variable time base, TimeBase 2, is connected to a capacitor and resistor and the time it takes to charge and discharge the capacitor a number of times through the resistor is now the gate time. The RC method can be realized with any MSP430.

TimeBase 1



TimeBase 2

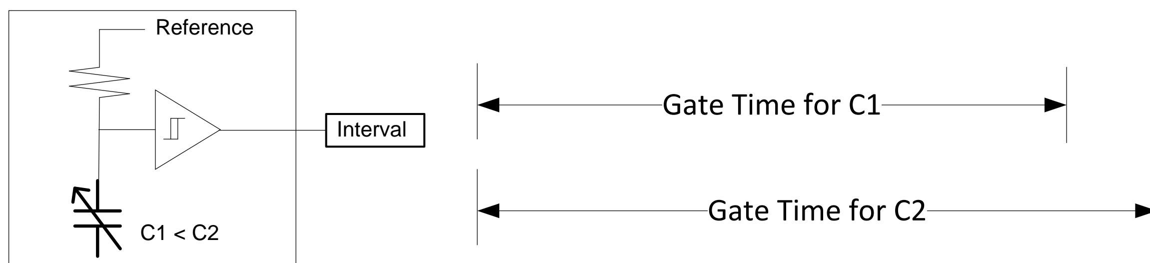


Figure 2. Resistor-Capacitor Time Constant Measurement (RC)

The gate time is a function of the capacitance of the circuit. As the capacitance increases (C1 to C2), the relationship between the two time bases changes. The number of system clock cycles counted within the gate time increases. As the capacitance decreases (C2 to C1), the number of cycles within the gate time decreases. For the RC method the relationship between capacitance and counts is direct.

This method uses a single timing resource (TIMER_A0) to measure the time it takes the capacitance to charge and discharge 'n' times. To measure both charge and discharge, two IO are needed, as shown in Figure 3. This is reflected in the function name within the library: RC_PAIR.

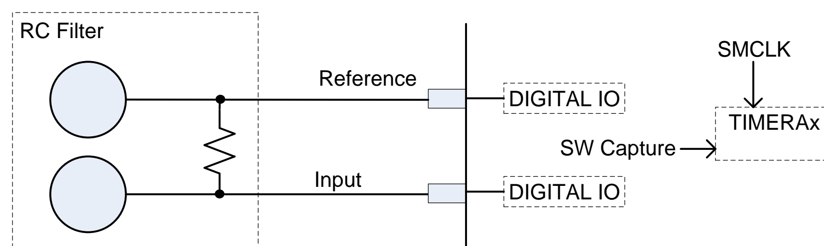


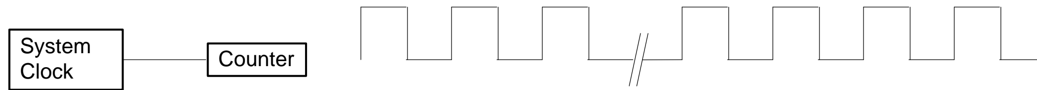
Figure 3. RC Schematic

(1) The RO method is described in more detail in *PCB-Based Capacitive Touch Sensing With MSP430* (SLAA363).

Copyright © 2011–2012, Texas Instruments Incorporated

2.3 Fast Scan Relaxation Oscillator (fRO)

The fRO method is similar to the RC method except that the variable gate period is created with a relaxation oscillator instead of the charge and discharge time. And as shown in Figure 4, the fixed time base, TimeBase 1, is connected to an internal MSP430 oscillator, such as the DCO. The variable time base, TimeBase 2, is connected to the relaxation oscillator.



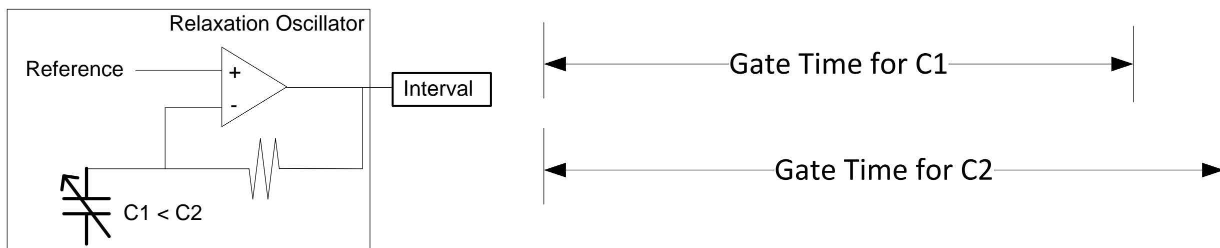


Figure 4. Fast Scan RO Measurement

The gate time is a function of the capacitance of the circuit. As the capacitance increases (C1 to C2), the relationship between TimeBase 1 and TimeBase 2 changes. The number of system clock cycles counted within the gate time increases. As the capacitance decreases (C2 to C1), the number of cycles within the gate time decreases. For the fRO method the relationship between capacitance and counts is direct just like the RC method. The fast relaxation oscillator (fRO) method is intended to bridge the gap between the RC and RO methods. The fRO method provides the fast scan rates of the RC method and the improved sensitivity of the RO method.

The naming convention for the fRO method in the library identifies the relaxation oscillator mechanism, the timer used to define the gate period (as a function of the variable relaxation oscillator frequency), and the timer used to count the fixed oscillation frequency.

Table 2. Fast RO Naming Convention

Name	RO Mechanism	Gate Period	Counter
fRO_XXX_YYY_ZZZ	XXX	YYY	ZZZ
fRO_COMPB_TA1_SW	Comparator B	TimerA1	Software Loop
fRO_PINOSC_TA0_TA1	Pin Oscillator	TimerA0	TimerA1

As the name implies, the purpose of the fRO method is to provide fast scan rates; faster than the RO method with similar sensitivity [(1)]. With the RO method, sensitivity is a function of the gate time. Increasing the gate time increases sensitivity. The negative consequence of increasing gate time is decreased scan rates: more time is spent during a single measurement. The sensitivity of the fRO method can also be improved with increased gate time, however, the sensitivity can also be improved by improving the system clock resolution (higher speed).

In comparison to the RO method the fRO method provides similar sensitivity (change in counts) in a shorter gate time. The theoretical exercise found in shows that similar sensitivity can be achieved with the fRO method in less time.

(1) In this context, sensitivity is a measure of how small a change in capacitance can be resolved.

Copyright © 2011–2012, Texas Instruments Incorporated

Table 3. Comparison of fRO and RO Measurement Times

	RO_PinOsc_TA0_WDTp			fRO_PinOsc_TAx_TA1		
	Gate Time (1-MHz SMCLK, WDT, 512)	Counter (RO)	Counts	Gate Time (45 RO cycles)	Counter (SMCLK)	Counts
Touched	512 µs	1 MHz	512	45/1 MHz = 45 µs	12 MHz	540
Untouched	512 µs	1.1 MHz	563	45/1.1 MHz = 41 µs	12 MHz	492
Difference			51			48

3 Configuration

There are two main files that serve as the means to configure the library: structure.c and structure.h. structure.c includes definitions of the elements and the sensors (groups of elements). The structure.h file makes the definitions in structure.c visible to the other portions of the library and also uses precompiler definitions to enable functions and limit code size.

3.1 Element Definition

A capacitive measurement element is a singular structure, whose capacitance represents an event; for example, a touch, a change in humidity, or a change in dielectric. An element can be used individually (for example as a button) or combined with other elements to create a keypad, wheel, or slider as shown in Figure 5.

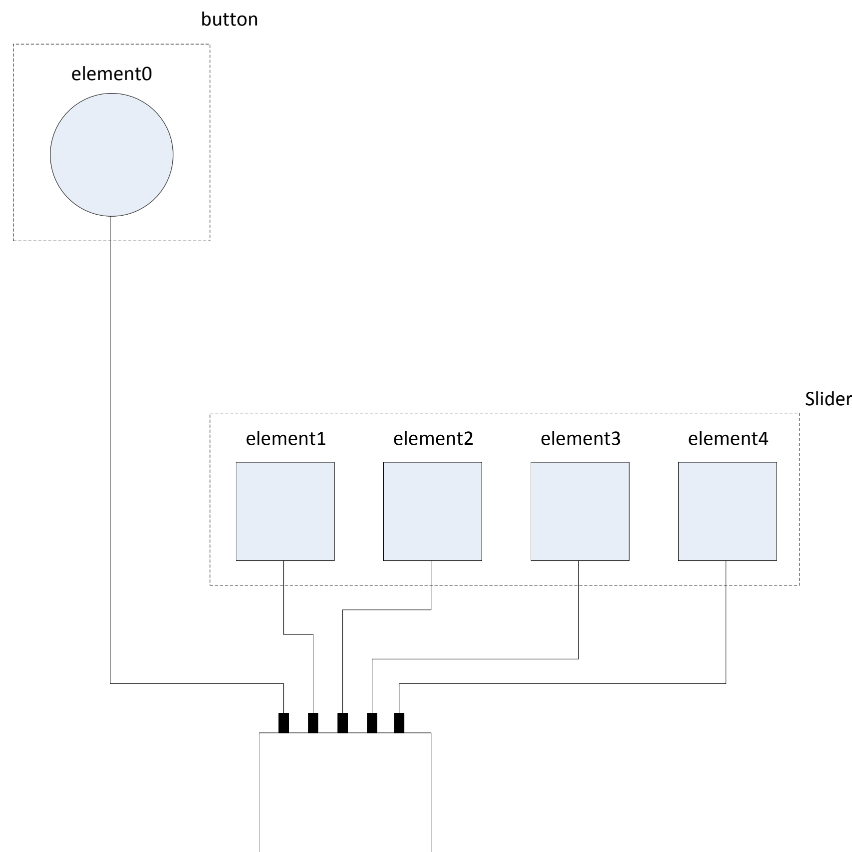


Figure 5. Elements

Copyright © 2011–2012, Texas Instruments Incorporated

The element definitions belong to one of two categories; port definition or measurement parameter. The ports definitions include digital IO peripheral registers, comparator peripheral registers, and bit definitions. The measurement parameters include the threshold and maximum signal response (maxResponse) of an element. The port definition can be completed by simply reading the schematic while the measurement parameters require testing (see Section 6). Establishing the correct measurement parameters calibrates the elements.

The element definition found in the configuration file structure.c uses a designated initializer list. This allows members to be initialized in any order and also enhances the readability of the element being initialized. This feature requires the GCC language extension found in Code Composer Studio (CCS). C99 is the default dialect found in IAR and, therefore, the default settings can be used.

3.1.1 Common Element Variables

InputBits is one common definition that can represent the bit y in the GPIO definition Px.y, the comparator input mux for either COMP_A+ or COMP_B solutions, or the capacitive touch IO port and bit selection in the capacitive touch IO control register.

threshold defines the limit or threshold that the change in capacitance must exceed before an event (typically a touch) is declared.

maxResponse is the maximum response expected from an element within a sensor and only used in sensors with multiple elements: slider, wheel, and buttons [(1)]. The purpose of the maxResponse parameter is to normalize the capacitive measurement to a percentage, where the threshold represents 0% and the maxResponse represents 100%. This percentage is used to identify the dominant element within the sensor if multiple elements have threshold crossings.

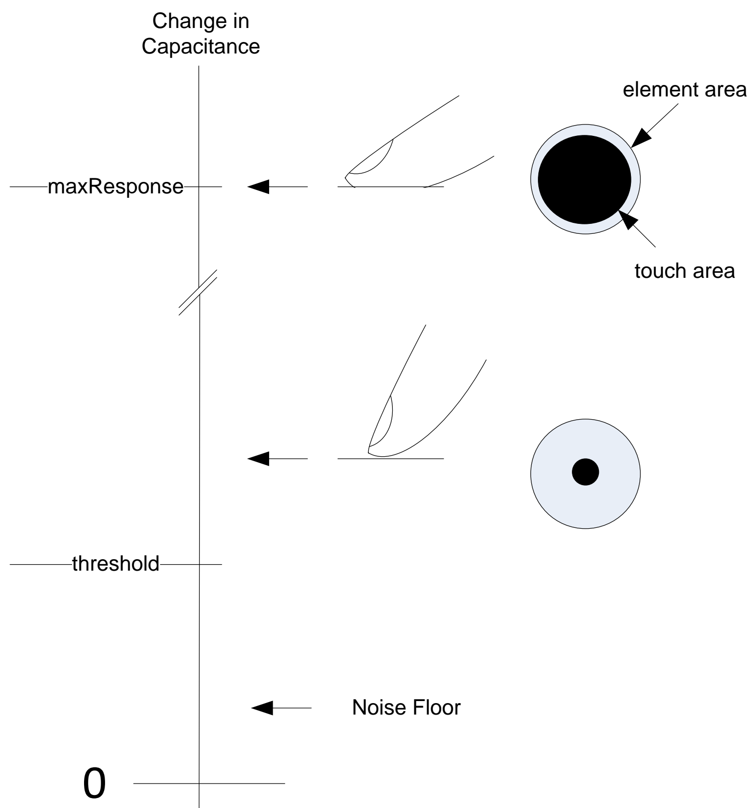


Figure 6. Element Measurement Parameters: Buttons Example

(1) The buttons abstraction is a sensor made from two or more elements. The button abstraction is a sensor made from one element.

Copyright © 2011–2012, Texas Instruments Incorporated

Figure 6 shows the relative relationship between threshold and maxResponse in a buttons application. First, the maxResponse value is only needed for the buttons, wheel, and slider APIs. As mentioned in the definition the maxResponse is used to determine the dominant element. Second, the maxResponse represent a heavy touch where the finger flattens, by pressing heavily, to create a larger surface area. By contrast a light or normal touch creates a much smaller surface area. The threshold value represents a change in capacitance that is smaller than even the lightest touch but still significantly larger than any noise that might appear as a change in capacitance.

The threshold and maxResponse variables are limited to unsigned 16 bit integers (0 to 65535). These values are further limited by the following when a multi-element abstraction is used (buttons, slider, or wheel): maxResponse – threshold < 655. Section 6 provides more detail on establishing the measurement parameters.

3.1.1.1 Comparator_A+ (COMP_A+)

Implementations using the COMP_A+ peripheral to create a relaxation oscillator use the same element structure format.

InputBits identify the bits, P2CA1, P2CA2, and P2CA3, within the CACTL2 register which represent the negative terminal input of the comparator. This input is connected directly to the electrode. The input for the reference is defined in the sensor section.

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

3.1.1.2 Comparator_B (COMP_B)

InputBits identify the CBIMSEL bits in the CBCTL0 register.

Figure 7. CBCTL0 Register

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

Table 4. CBCTL0 Register Description

Bit	Field	Type	Reset	Description
15	CBIMEN	RW	0h	Channel input enable for the V– terminal of the comparator. 0b = Selected analog input channel for V– terminal is disabled. 1b = Selected analog input channel for V– terminal is enabled.
14-12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	CBIMSEL	RW	0h	Channel input selected for the V– terminal of the comparator if CBIMEN is set to 1.
7	CBIPEN	RW	0h	Channel input enable for the V+ terminal of the comparator. 0b = Selected analog input channel for V+ terminal is disabled. 1b = Selected analog input channel for V+ terminal is enabled.
6-4	Reserved	R	0h	Reserved. Always reads as 0.
3-0	CBIPSEL	RW	0h	Channel input selected for the V+ terminal of the comparator if CBIPEN is set to 1.

Copyright © 2011–2012, Texas Instruments Incorporated

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

3.1.1.3 Pin Oscillator (PinOsc)

The Pin Oscillator (PinOsc) implementation of the relaxation oscillator replaces the comparator and reference circuitry with the Schmitt trigger input found in the digital IO and an internal inverter. The PinOsc feedback path to the RC filter is accomplished with the integrated resistor, as shown in Figure 8.

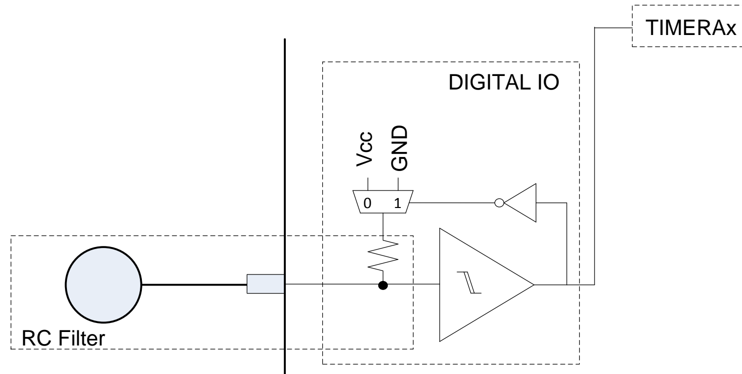


Figure 8. RO_PinOsc Schematic

**inputPxselRregister* and **inputPxsel2Register* identify the appropriate registers that need to be configured for the pin oscillator method. These registers in conjunction with *inputBits* configure the element as shown in Figure 9 and the following code snippet.

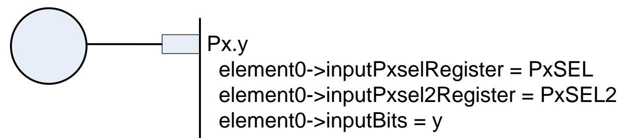


Figure 9. PinOsc Port Parameters

```
const struct Element right = {

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

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

3.1.1.4 Capacitive Touch IO (CSIO)

The capacitive touch IO (CSIO) implementation of the relaxation oscillator is the same as the PinOsc; however, the interface for controlling the digital IO is different.

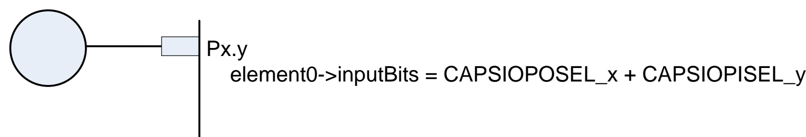


Figure 10. Capacitive Touch IO Implementation

Copyright © 2011–2012, Texas Instruments Incorporated

The relaxation oscillator is accomplished with internal circuitry and the port parameter only includes the *inputBits* which represent the capacitive touch IO port and pin select bits within the control register (CAPSIOxCTL). The port and pin select bits are defined at the element level while the control register is defined at the sensor level: inputCapsioctlRegister.

InputBits identify the port and pin select bits in the CAPSIOxCTL register.

Figure 11. CAPSIOxCTL Register

15	14	13	12	11	10	9	8
Reserved						CAPSIO	CAPSIOEN
r0	r0	r0	r0	r0	r0	r-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CAPSIOPOSELx				CAPSIOPISELx			Reserved
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0

Table 5. CAPSIOxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads 0.
9	CAPSIO	R	0h	Capacitive Touch IO state. Reports the current state of the selected Capacitive Touch IO. Reads 0, if Capacitive Touch IO disabled. 0b = Current state 0 or Capacitive Touch IO is disabled 1b = Current state 1
8	CAPSIOEN	RW	0h	Capacitive Touch IO enable 0b = All Capacitive Touch IOs are disabled. Signal toward timers is 0. 1b = Selected Capacitive Touch IO is enabled
7-4	CAPSIOPOSELx	RW	0h	Capacitive Touch IO port select. Selects port Px. Selecting a port pin that is not available on the device in use gives unpredictable results. 0000b = Px = PJ 0001b = Px = P1 0010b = Px = P2 ... 1110b = Px = P14 1111b = Px = P15
3-1	CAPSIOPISELx	RW	0h	Capacitive Touch IO pin select. Selects the pin within selected port Px (see CAPSIOPOSELx). Selecting a port pin that is not available on the device in use gives unpredictable results. 000b = Px.0 001b = Px.1 010b = Px.2 ... 110b = Px.6 111b = Px.7
0	Reserved	R	0h	Reserved. Always reads 0.

```
const struct Element element_name = {
              .inputBits = CAPSIOPOSEL0+CAPSIOPOSEL1+CAPSIOPISEL2, // P3.4
              .threshold = 100,
              .maxResponse = 200
}
```


Copyright © 2011–2012, Texas Instruments Incorporated

3.1.2 Element Variables Specific to RC

The RC method is comprised of two GPIO. One is the input and the other is the reference. The configuration requires the pertinent register addresses for a given port as well as the bit definition.

inputPxdirRegister, inputPxoutRegister, and inputPxinRegister identify the port direction , output, and input addresses. These registers in conjunction with *inputBits* configure the input portion of the element definition.

referencePxdirRegister and referencePxoutRegister identify the port direction and output addresses. These registers in conjunction with *referenceBits* configure the reference portion.

One feature of this description is that one pin can have two different functions. That is the pin can be a reference in one element definition and an input in another element definition. Figure 12 and the following code snippet show the reference and input definitions.

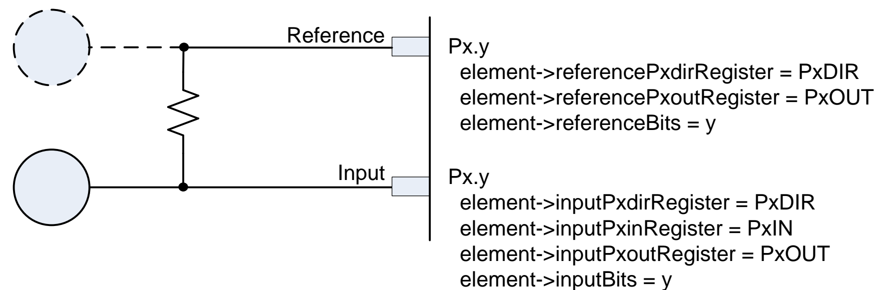


Figure 12. RC IO Parameters

```
//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
};
```

3.2 Sensor Definition

The sensor can simply be a single element, a group of independent elements like a keypad, or a group of elements functioning as one sensor like a wheel or slider. The sensor definition includes all of the applicable elements, the implementation that is used to measure the capacitance of all the elements, and the peripheral addresses and bit settings for the given mechanism. In the case of wheels and sliders, the sensor definition also defines the number of points or positions along with the slider, as shown in Figure 13, and the sensitivity of the sensor.

Copyright © 2011–2012, Texas Instruments Incorporated

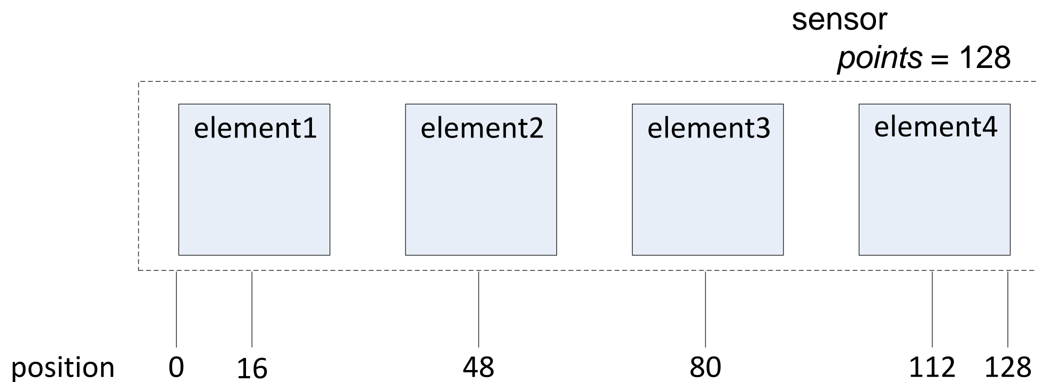


Figure 13. Sensor Example

This section is divided into three sub-sections: port variables, common variables, and variables specific to sliders and wheels. The port variables are only applicable to the CSIO and comparator implementations of capacitive touch sensing. These implementations require either additional GPIO or register settings to configure the MSP430. The common variables are variables that are common in name but different in meaning dependent upon the method. This sub-section describes through each implementation, providing code examples. The wheels and sliders sub-section describes the variables associated with wheel and slider sensor.

The sensor definition found in the configuration file structure.c uses a designated initializer list. This allows members to be initialized in any order and also enhances the readability of the sensor being initialized. This feature requires the GCC language extension found in Code Composer Studio (CCS). C99 is the default dialect found in IAR and therefore the default settings can be used.

3.2.1 Port Variables

The port variables within the sensor definition are only applicable to the COMP_A+, COMP_B, and CSIO implementations.

3.2.1.1 CSIO Port Variable

The port and pin select bits are defined at the element level while the control register is defined at the sensor level: *inputCapsioctlRegister*.

The *inputCapsioctlRegister* is different from the other parameters found in the library because the association between the CAPSIOxCTL register and timer is device dependent. It is important that the control register selected works with the HAL also indicated within the sensor level definition. For example if CAPSIO0CTL is associated with TimerA2, then the *inputCapsioctlRegister* must point to CAPSIO0CTL when the RO_CSIO_TA2_WDTA HAL definition is used.

3.2.1.2 COMP_B Port Variables

The implementation of the COMP_B solution requires an alternative set of parameters for the sensor port variables, shown in Figure 14.

Copyright © 2011–2012, Texas Instruments Incorporated

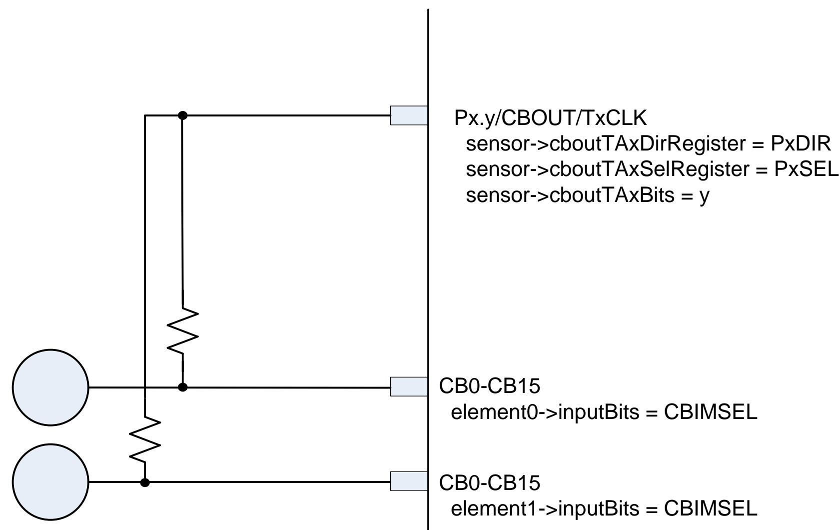


Figure 14. RO_COMPB Port Parameters

cboutTAxRegister, cboutTAxRegister identify the port direction address and port selection address. The variable *cboutTAxBits* define the bits that are to be set or reset within the direction and selection register to select the CBOUTx output and TxCLK input function of the port. Note that on 5xx family devices, these ports share the same IO.

Some MSP430 devices allow the digital IO to be driven from a different voltage rail (DVIO) than the voltage rail (VCC) that the Comp_B peripheral uses. If the comparator output, CBOUT, uses DVIO and DVIO is different from VCC, then the input high and input low values set within the library need to be updated. These values are set in the Comp_B control register 2, CBCTL2, found in the CTS_HAL.c file.

cbpdBits is used to disable the Digital IO on the port pins also used as the comparator inputs. This is applied to the COMP_B control register CBCTL3. The bit CBPDy, in CBCTL3, disables the port of the comparator channel 'y' (that is, CBPDy disables CBy and not Px.y)

3.2.1.3 COMP_A+ Port Variables

In the element structure the comparator input is defined for each element. At the sensor level, as shown in Figure 15, the comparator reference input is defined as well as the reference port, comparator output, and timer input.

Copyright © 2011–2012, Texas Instruments Incorporated

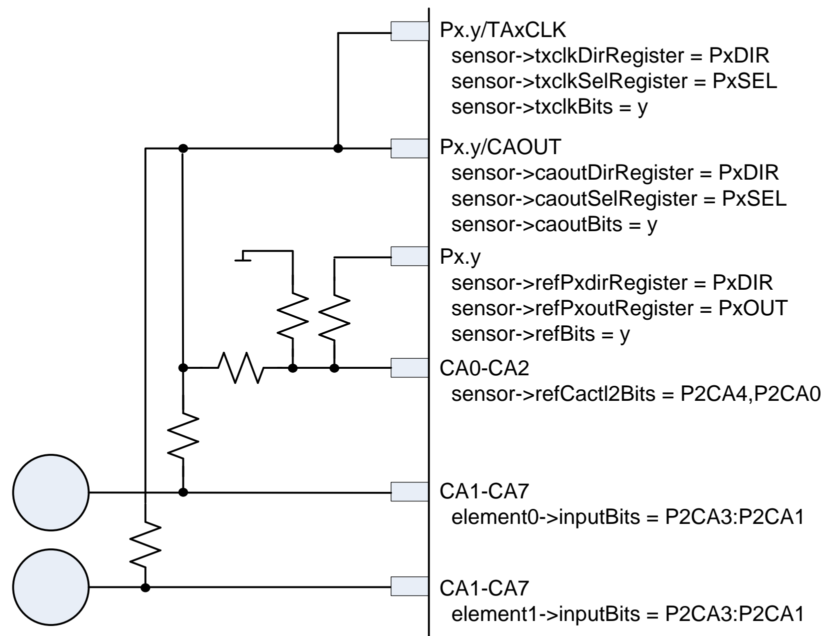


Figure 15. Port Parameters for RO_COMPAp_TAx Implementation

caoutDirRegister, caoutSelRegister identify the port direction address and port selection address. The variable *caoutBits* define the bits that are to be set or reset within the direction and sel register to select the CAOUT output function of the port. Some devices also use the PxSEL2 register to define the CAOUT use case. In the case of these devices the value *caoutSel2Register* must also be defined (either as P1SEL2 or P2SEL2).

txclkDirRegister, txclkSelRegister identify the port direction address and port selection address. The variable *txclkBits* define the bits that are to be set or reset within the direction and selection register to select the TxCLK input function of the port. Some devices also use the PxSEL2 register to define the TxCLK use case. In the case of these devices, the value *txclkSel2Register* must also be defined (either as P1SEL2 or P2SEL2).

The *refPxDirRegister, refPxOutRegister* and *refBit* variables define the pullup portion of the external reference circuit shown in Figure 15. These bits provide the mechanism to turn on and off the reference for power savings. *refPxDirRegister, refPxOutRegister* identify the port direction address and port output address. The variable *refBits* define the bits that are to be set or reset within the direction and selection register to enable the reference circuit.

refCactl2Bits indicates which positive input of COMP_A+ is connected to the voltage reference. The reference should only be applied to the positive input via CA0, CA1, or CA2. This is represented as P2CA0, P2CA4, and P2CA0+P2CA4, respectively.

capdBits is used to define the IO which make up the sensor. This is applied to the COMP_A+ control register CAPD. This value is the logical 'OR' of all the bit definitions for each input and the reference input (that is, the 'y' of Px.y and NOT the 'y' in CAy).

A slight variation of the COMP_A+ implementation is the fRO_COMPAp_SW_TAx implementation which uses a software timer to create the gate time and the timer peripheral as the frequency counter. Because this implementation does not use the timer to count the number relaxations oscillations the physical connection is no longer needed as shown in Figure 16.

Copyright © 2011–2012, Texas Instruments Incorporated

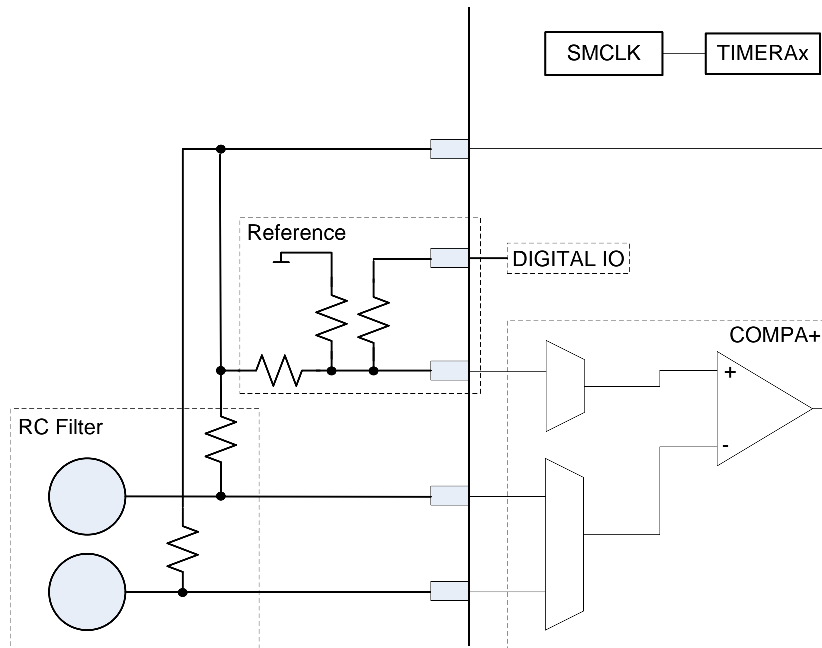


Figure 16. fRO_COMPAp_SW_TAx General Description

The port parameters are the same as those described above with the exception of *txclkDirRegister, txclkSelRegister,* and *txclkBits,* which are not used.

3.2.2 Common Sensor Variables

numElements identifies the number of elements that are within the sensor.

baseOffset is a cumulative count of the number of elements that are defined in the application. There is a baseline value stored in RAM for each element.

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

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

Table 6. baseOffset Description

Sensor	.baseOffset	Element	RAM Address
Slider0	0	element0	baseCnt[0]
		element1	baseCnt[1]
		element2	baseCnt[2]
		element3	baseCnt[3]
Slider1	4	element4	baseCnt[4]
		element5	baseCnt[5]
		element6	baseCnt[6]
		element7	baseCnt[7]

arrayPtr identifies all of the elements associated with a sensor. In the case of the wheel and slider the order of the array is important because it is assumed that the order represents the physical order of the elements.

measGateSource is used to define either the gate timer source or the measurement clock source depending upon the method (halDefinition). In RC and fRO methods, measGateSource defines the measurement clock source. In the RO implementation measGateSource identifies the gate timer source.

Copyright © 2011–2012, Texas Instruments Incorporated

sourceScale is used to further divide down the source (either the measurement source or the gate source depending upon the context). This only applies to timer peripherals and not the watch dog timer peripherals.

accumulationCycles is used to define the gate time. Typically accumulationCycles is the number of oscillations of the gate time, but when the watch dog timer is used as the gate peripheral the accumulationCycles represents the bit settings in the watch dog timer control register.

halDefinition identifies which measurement implementation is being used for the sensor. Table 7 lists the different implementations currently supported.

Table 7. halDefinition Descriptions

halDefinition	Description
RO_COMPAp_TA0_WDTp RO_COMPAp_TA1_WDTp	Relaxation oscillator implemented with COMP_A+ peripheral. The gate time is fixed and defined by the WDT+ peripheral set to interval mode. The capacitance is represented by the number of RO cycles counted by TimerA0 or TimerA1 during the fixed gate time (see Section 3.2.2.1).
RO_PINOSC_TA0_WDTp	Relaxation oscillator implemented with PinOsc, TimerA0 is used to measure frequency of oscillator, and the WDTp is used to set the gate time (see Section 3.2.2.3).
RO_PINOSC_TA0	Relaxation oscillator implemented with PinOsc, TimerA0 is used to measure frequency of oscillator, and the ACLK source is used to set the gate time (see Section 3.2.2.4).
RO_PINOSC_TA0_TA1	Relaxation oscillator implemented with PinOsc, TimerA0 is used to measure frequency of oscillator, and TimerA1 is used to set the gate time (see Section 3.2.2.3).
RO_COMPB_TA0_WDTA RO_COMPB_TA1_WDTA RO_COMPB_TB0_WDTA	Relaxation oscillator implemented with COMP_B peripheral, TimerA0, TimerA1, or TimerB0 is used to measure frequency of oscillator, and the WDTA peripheral is used to set the gate time (see Section 3.2.2.2).
RO_COMPB_TA1_TA0	Relaxation oscillator implemented with COMP_B peripheral, TimerA1 is used to measure frequency of oscillator, and TimerA0 is used to set the gate time (see Section 3.2.2.2).
RO_CSIO_TA2_WDTA	Relaxation oscillator implemented with capacitive touch IO peripheral, TimerA2 is used to measure frequency of oscillator, and the WDTA peripheral is used to set the gate time (see Section 3.2.2.5).
RO_CSIO_TA2_TA3	Relaxation oscillator implemented with capacitive touch IO peripheral, TimerA2 is used to measure frequency of oscillator, and TimerA3 is used to set the gate time (see Section 3.2.2.5).
RC_PAIR_TA0	Measure RC time constant with TimerA0. The gate time is variable and changes with the charge and discharge time. A software loop is used to establish the number of charge and discharge cycles that define the gate time. The capacitance is represented by the number of TimerA0 counts within the gate time. Typically TA0 is sourced from a high frequency clock (SMCLK) for improved sensitivity. The capacitive element is charged and discharge with the other IO defined in the pair (see Section 3.2.2.6).
fRO_COMPAp_TA0_SW fRO_COMPAp_TA1_SW	Fast Scan Relaxation oscillator implemented with COMP_A+ peripheral. The gate time is variable and changes with the period of the relaxation oscillator. The TimerA0 or TimerA1 is used to establish the number of oscillations that define the gate time. The capacitance is represented by the number of software loops counted within the gate time (see Section 3.2.2.12).
fRO_COMPAp_SW_TA0	Fast Scan Relaxation oscillator implemented with COMP_A+ peripheral. A software (SW) loop is used to establish the number of oscillations that define the gate time. The capacitance is represented by the number of timer counts in TimerA0 for the gate time. Typically TA0 is sourced from a high frequency clock (SMCLK) for improved sensitivity (see Section 3.2.2.13).
fRO_PINOSC_TA0_SW	Measure the time with TimerA0, Relaxation oscillator implemented with PinOsc, several oscillations are counted in SW to establish the gate time (see Section 3.2.2.9).
fRO_PINOSC_TA0_TA1	Fast Scan Relaxation oscillator implemented with PinOsc. The TimerA0 is used to establish the number of oscillations that define the gate time. The capacitance is represented by the number of TimerA1 counts within the gate time. Typically TA1 is sourced from a high frequency clock (SMCLK) (see Section 3.2.2.8).
fRO_COMPB_TA0_SW fRO_COMPB_TA1_SW	Fast Scan Relaxation oscillator implemented with the COMP_B peripheral. The TimerA0 or TimerA1 is used to establish the number of oscillations that define the gate time. The capacitance is represented by the number of software loops counted within the gate time (see Section 3.2.2.11).
fRO_COMPB_TA1_TA0	Fast Scan Relaxation oscillator implemented with COMP_B. The TimerA1 is used to establish the number of oscillations that define the gate time. The capacitance is represented by the number of TimerA1 counts within the gate time. Typically TA1 is sourced from a high frequency clock (SMCLK) (see Section 3.2.2.10).

Copyright © 2011–2012, Texas Instruments Incorporated

Table 7. halDefinition Descriptions (continued)

halDefinition	Description
fRO_CSIO_TA2_TA3	Fast Scan Relaxation oscillator implemented with the capacitive touch IO. The TimerA2 is used to establish the number of oscillations that define the gate time. The capacitance is represented by the number of TimerA3 counts within the gate time. Typically TA3 is sourced from a high frequency clock (SMCLK) (see Section 3.2.2.7).

3.2.2.1 RO_COMPAp_TAx_WDTp

The relaxation oscillator is comprised of the COMP_A+ module, a reference, and RC filter. The reference is connected to the non-inverting input of COMP_A+ (via the input mux) while the RC filter is connected to the inverting input (also via an input mux). The reference is a voltage divider made up of one connection to a GPIO and the other connections to ground and the comparator output.

One way to measure the capacitance is to route the oscillator (via CAOUT) to a timer input (TAxCLK). Two different HAL definitions are provided depending upon which clock is available; RO_COMPAp_TA0_WDTp and RO_COMPAp_TA1_WDTp.

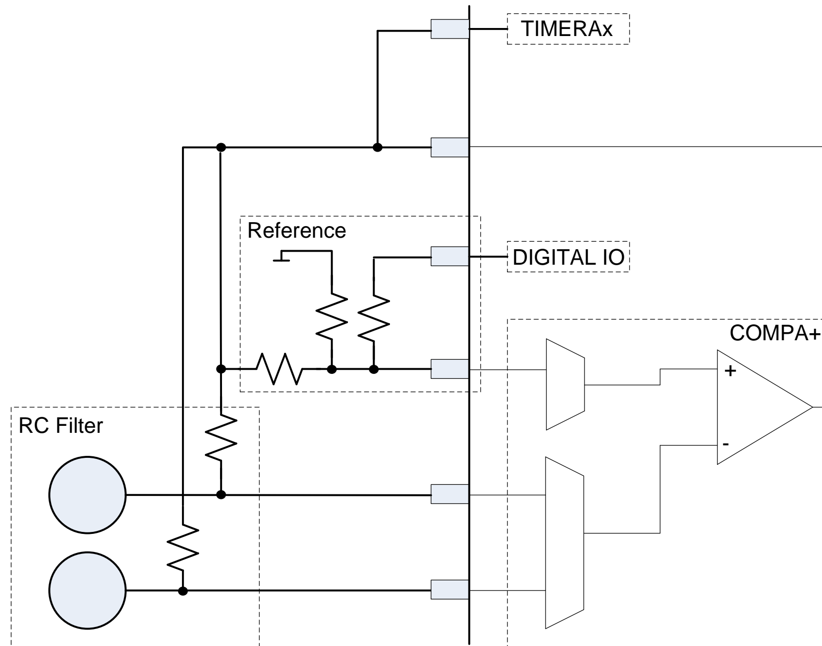


Figure 17. RO_COMPAp_TAx Schematic Description

The two timing parameters define the WDTp interval that is the gate time for the RO_COMPAp_TAx_WDTp implementation.

measureGateSource indicates the WDTp source: SMCLK or ACLK. This parameter is equivalent to the 'Watchdog timer+ clock source select' bits in the Watchdog Timer+ register.

Table 8. Watchdog Timer Source Select Definitions

Definition	Value	Source
GATE_WDTp_ACLK	0x0004	ACLK
GATE_WDTp_SMCLK	0x0000	SMCLK

accumulationCycles is used to define the WDTp interval in the RO_COMPAp_TAx_WDTp implementation. This is equivalent to the interval select bits in the Watchdog Timer+ register.

Copyright © 2011–2012, Texas Instruments Incorporated

Table 9. Watchdog Timer+ Interval Select Definitions

Definition	Value	Interval (s)
WDTp_GATE_32768	0x0000	32768/source
WDTp_GATE_8192	0x0001	8192/source
WDTp_GATE_512	0x0002	512/source
WDTp_GATE_64	0x0003	64/source

Figure 18 shows how the common sensor parameters measGateSource and accumulationCycles are used to select the gate time.

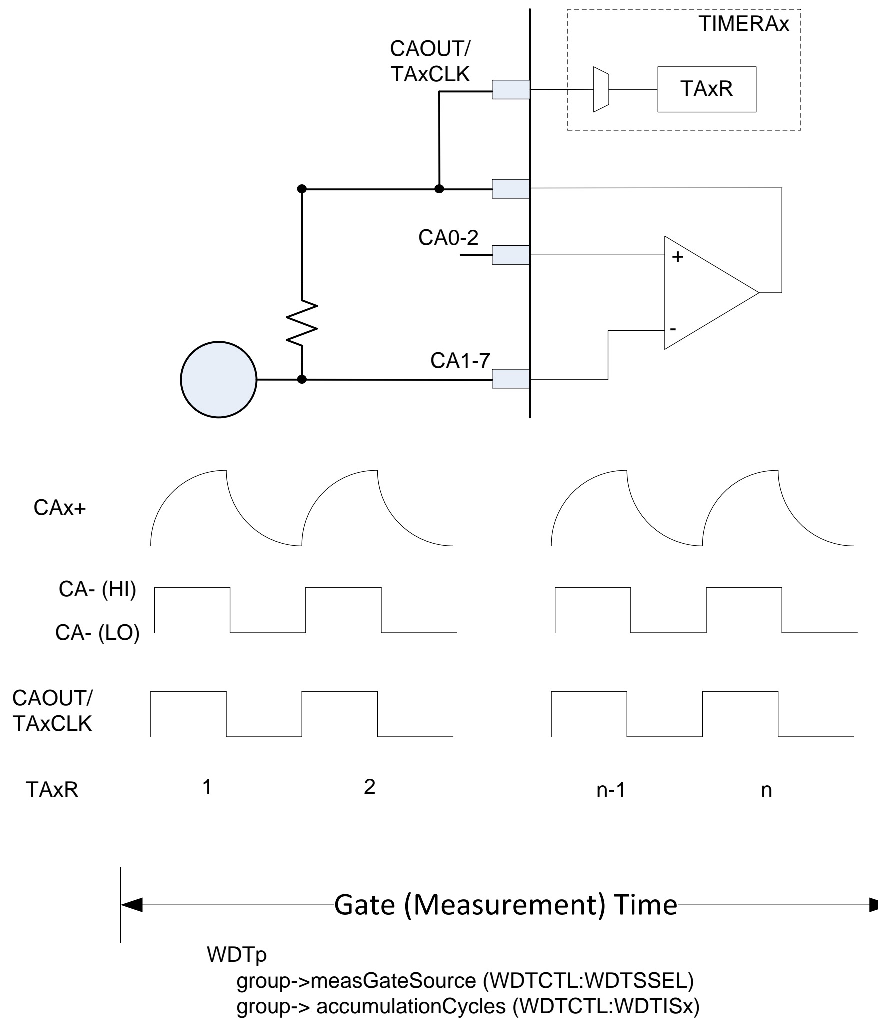


Figure 18. Timing Parameters: Example WDTp

The following example describes a sensor made up of four elements, and each element is measured with the RO method for a period of 512/SMCLK.

Copyright © 2011–2012, Texas Instruments Incorporated

```
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),

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

3.2.2.2 RO_COMPB_yyy_zzz

The relaxation oscillator solution with COMP_B is the same in function as the COMP_A+ solution. The COMP_B peripheral solution is different in implementation, integrating the reference circuitry and connection to the TimerA or TimerB peripheral, as shown in Figure 19.

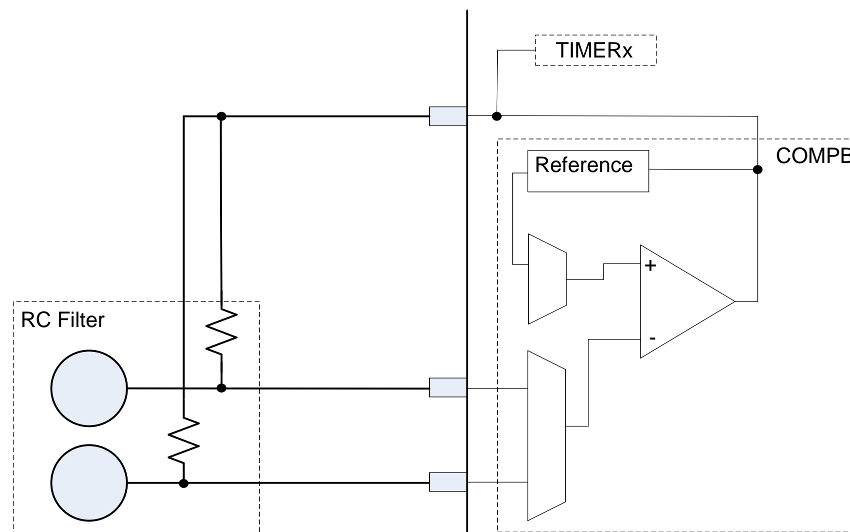


Figure 19. RO_COMPB Schematic

Copyright © 2011–2012, Texas Instruments Incorporated

3.2.2.2.1 RO_COMPB_TAx_WDTA, RO_COMPB_TBx_WDTA

The two timing parameters define the WDTA interval that is the gate time for the implementation. The WDTA module provides 4 different source settings and 8 watch dog timer intervals.

measureGateSource indicates the WDTA source: SMCLK. ACLK, VLO, or XCLK. This parameter is equivalent to the Watchdog timer clock source select bits in the Watchdog Timer Control Register (WDTCTL).

Table 10. Watchdog Source Select Definitions

Definition	Value	Source
GATE_WDTA_SMCLK	0x0000	SMCLK
GATE_WDTA_ACLK	0x0020	ACLK
GATE_WDTA_VLO	0x0040	VLO
GATE_WDTA_XCLK	0x0060	XCLK

accumulationCycles is used to define the WDTA interval. This is equivalent to the interval select bits in the Watchdog Timer Control Register (WDTCTL).

Table 11. Watchdog Interval Select Definitions

Definition	Value	Interval (s)
WDTA_GATE_2G	0x0000	2G/source
WDTA_GATE_128M	0x0001	128M/source
WDTA_GATE_8192K	0x0002	8192k/source
WDTA_GATE_512K	0x0003	512k/source
WDTA_GATE_32768	0x0004	32768/source
WDTA_GATE_8192	0x0005	8192/source
WDTA_GATE_512	0x0006	512/source
WDTA_GATE_64	0x0007	64/source

The following example describes a sensor made up of four elements and each element is measured with the RO method for a period of 512000/SMCLK.

```
const struct Sensor sliderA =
               {
                  .halDefinition = RO_COMPB_TA0_WDTA,
                  .numElements = 4,
                  .baseOffset = 0,
                  .points = 64,
                  .sensorThreshold = 5,
                  .cbpdBits = (BITC+BITD+BITE+BITF),
                  // Pointer to elements
                  .arrayPtr[0] = &element0,
                  .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  //
               };
```

Different members within the 5xx family provide an internal connection between CBOUT and a timer. The description and parameters are the same for timers TA0, TA1, and TB0, with the exception of the HAL definition name.

Copyright © 2011–2012, Texas Instruments Incorporated

3.2.2.2.2 RO_COMPB_TA1_TA0

The timing parameters define the timer interval that is the gate time for the implementation. The timer module provides four different source settings, a divider to scale the input source, and the interval, which is a 16 bit integer loaded into the timer compare register.

measureGateSource indicates the timer source: TxCLK, ACLK, SMCLK, or INCLK. This parameter is equivalent to the timer source select bits (TxSSEL) in the timer control register (TxCTL).

Table 12. TimerA Source Select Definitions

Definition	Value	Source
TIMER_TxCLK	0x0000	TxCLK
TIMER_ACLK	0x0100	ACLK
TIMER_SMCLK	0x0200	SMCLK
TIMER_INCLK	0x0300	INCLK or inverted TxCLK

sourceScale is used to divide the timer source. This parameter is equivalent to the input divider select bits (ID) found in the timer control register (TxCTL).

accumulationCycles is used to define the timer interval. This value is loaded into the capture/compare register (TxCCR0).

The following example describes a sensor made up of four elements and each element is measured with the RO method for a period of 50/ACLK.

```
const struct Sensor keypad =
{
    .halDefinition = RO_COMPB_TA1_TA0,
    .numElements = 5,
    .baseOffset = 0,
    .cbpdBits = 0x001F,        // CB0,CB1,CB2,CB3,CB4
    .arrayPtr[0] = &element0,  // point to first element
    .arrayPtr[1] = &element1,
    .arrayPtr[2] = &element2,
    .arrayPtr[3] = &element3,
    .arrayPtr[4] = &element4,
    .cboutTAxDirRegister = (uint8_t *)&P1DIR,  // PxDIR
    .cboutTAxSelRegister = (uint8_t *)&P1SEL,  // PxSEL
    .cboutTAxBits = BIT6, // P1.6
    // Timer Information
    .measGateSource = TIMER_ACLK,      //  ACLK
    .sourceScale = TIMER_SOURCE_DIV_0, // ACLK/1
    /*  50 ACLK/1 cycles or 50*1/32Khz = 1.5ms */
    .accumulationCycles = 50
};
```

3.2.2.3 RO_PinOsc_TA0_zzz

3.2.2.3.1 RO_PinOsc_TA0_WDTp

The common timing parameters, *measGateSource* and *accumulationCycles*, are the same as the COMP_A+ implementation (see Section 3.2.2.1).

The following sensor definition describes a sensor made up of one element that is measured with the RO method for a period of 8192/SMCLK.

Copyright © 2011–2012, Texas Instruments Incorporated

```
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_WDTp_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
               };
```

3.2.2.3.2 RO_PinOsc_TA0_TA1

The two timing parameters define the timer interval that is the gate time for the implementation. The timer module provides four different source settings, a divider to scale the input source, and the interval, which is a 16 bit integer loaded into the timer compare register.

measureGateSource indicates the timer source: TxCLK, ACLK, SMCLK, or INCLK. This parameter is equivalent to the timer source select bits (TxSSEL) in the timer control register (TxCTL).

Table 13. TimerA Source Select Definitions

Definition	Value	Source
TIMER_TxCLK	0x0000	TxCLK
TIMER_ACLK	0x0100	ACLK
TIMER_SMCLK	0x0200	SMCLK
TIMER_INCLK	0x0300	INCLK or inverted TxCLK

sourceScale is used to divide the timer source. This parameter is equivalent to the input divider select bits (ID) found in the timer control register (TxCTL).

accumulationCycles is used to define the timer interval. This value is loaded into the capture/compare register (TxCCR0).

The following example describes a sensor made up of four elements and each element is measured with the RO method for a period of 50/ACLK.

```
const struct Sensor proximity =
{
    /* Gate source is INCLK by default */
    .halDefinition = RO_PINOSC_TA0_TA1,
    .numElements = 1,
    .baseOffset = 0,
    // Pointer to elements
    .arrayPtr[0] = &eProx,
    /*
     *  Timer Information
     *  Gate Source is SMCLK
     */
    .measGateSource = TIMER_SMCLK,
    /* Gate Source divided by 1 */
    .sourceScale = TIMER_SOURCE_DIV_0,
    .accumulationCycles = 11000  // Number of TimerA0 (RO) cycles
};
```


Copyright © 2011–2012, Texas Instruments Incorporated

3.2.2.4 RO_PinOsc_TA0

An alternative implementation of the RO_PinOsc, with select MSP430 devices [1], is to use the internal ACLK connection to the timer capture input. The gate time is the number of capture events (equivalent to ACLK cycles), while the frequency counter is still the peripheral TimerA0 sourced from the relaxation oscillator. Because the capture interrupt represents a single oscillation several interrupts are counted with a software loop to create the equivalent gate time. Unlike the WDT method where the measurement is done in low power mode, this software loop method consumes more power because the CPU stays in Active Mode.

As shown in Figure 20, the only timing parameter definition for the RO_PinOsc_TA0 implementation is the number of ACLK cycles: sensor->*accumulationCycles*.

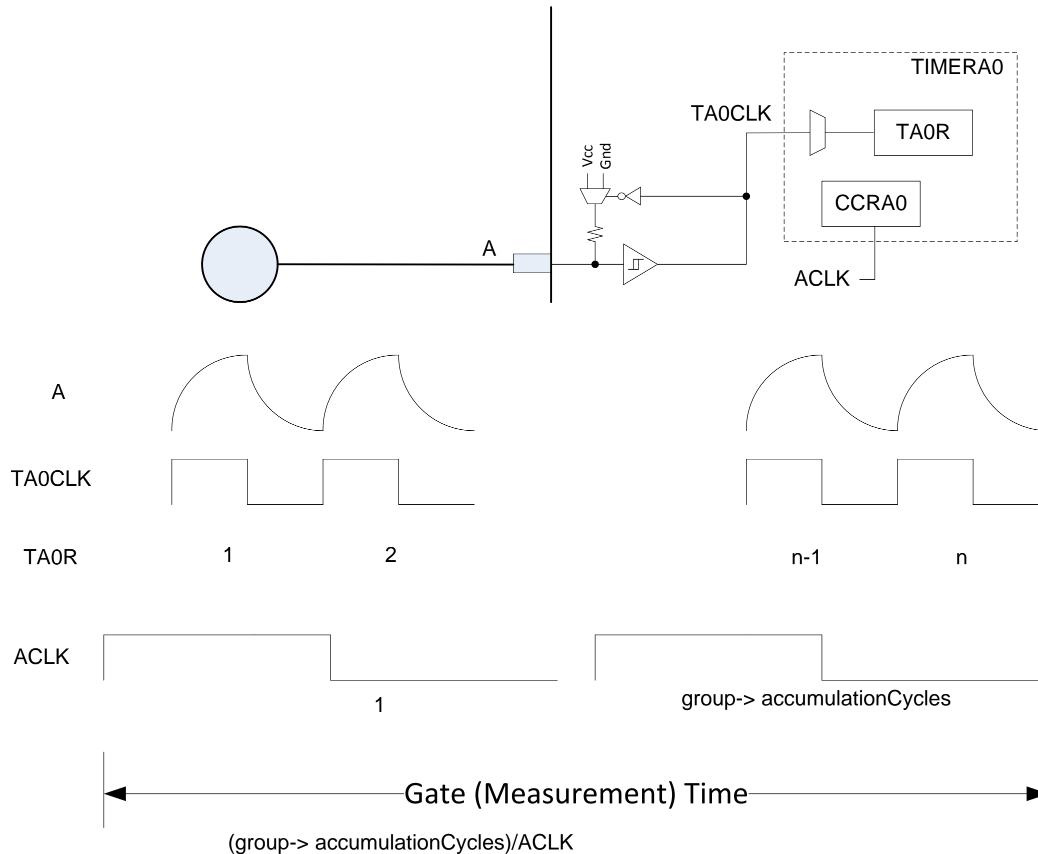


Figure 20. RO_PinOsc_TA0 Timing Parameter

The following sensor definition describes a sensor made up of one element that is measured with the RO method for a period of 100/ACLK.

```
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] See the data sheet to determine if the Timer capture input supports this connection to ACLK.

Copyright © 2011–2012, Texas Instruments Incorporated

3.2.2.5 RO_CSIO_yyy_zzz

3.2.2.5.1 RO_CSIO_TA2_WDTA

The common timing parameters, measGateSource and accumulationCycles, are the same as the COMP_B implementation (see Section 3.2.2.2).

The following sensor definition describes a sensor made up of 6 elements, where each element is measured with the RO method for a period of 64/SMCLK.

```
const struct Sensor keyPad =
{
    .halDefinition = RO_CSIO_TA2_WDTA,
    .inputCapsioctlRegister = (uint16_t *)&CAPSIO0CTL,
    .numElements = 6,
    .baseOffset = 0,
    // Pointer to elements
    .arrayPtr[0] = &key0,
    .arrayPtr[1] = &key1,
    .arrayPtr[2] = &key2,
    .arrayPtr[3] = &key3,
    .arrayPtr[4] = &key4,
    .arrayPtr[5] = &key5,
    .measGateSource = GATE_WDT_SMCLK,
    .accumulationCycles= WDTA_GATE_64
};
```

3.2.2.5.2 RO_CSIO_TA2_TA3

The timing parameters define the timer interval that is the gate time for the implementation. The timer module provides four different source settings, a divider to scale the input source, and the interval, which is a 16-bit integer loaded into the timer compare register.

measureGateSource indicates the timer source: TxCLK, ACLK, SMCLK, or INCLK. This parameter is equivalent to the timer source select bits (TxSSEL) in the timer control register (TxCTL).

Table 14. TimerA Source Select Definitions

Definition	Value	Source
TIMER_TxCLK	0x0000	TxCLK
TIMER_ACLK	0x0100	ACLK
TIMER_SMCLK	0x0200	SMCLK
TIMER_INCLK	0x0300	INCLK or inverted TxCLK

sourceScale is used to divide the timer source. This parameter is equivalent to the input divider select bits (ID) found in the timer control register (TxCTL).

accumulationCycles is used to define the timer interval. This value is loaded into the capture/compare register (TxCCR0).

The following example describes a sensor made up of 6 elements and each element is measured with the RO method for a period of 400/SMCLK.

Copyright © 2011–2012, Texas Instruments Incorporated

```
const struct Sensor keyPad =
{
    .halDefinition = RO_CSIO_TA2_TA3,
    .inputCapsioctlRegister = (uint16_t *)&CAPSIO0CTL,
    .numElements = 6,
    .baseOffset = 0,
    // Pointer to elements
    .arrayPtr[0] = &key0,
    .arrayPtr[1] = &key1,
    .arrayPtr[2] = &key2,
    .arrayPtr[3] = &key3,
    .arrayPtr[4] = &key4,
    .arrayPtr[5] = &key5,
    .measGateSource = TIMER_SMCLK,
    .sourceScale = TIMER_SOURCE_DIV_0,
    .accumulationCycles = 400
};
```

3.2.2.6 RC_PAIR_TA0

The RC method uses the timer peripheral to measure the charge and discharge time of the RC circuit. This measurement can be increased (in time and in counts) by accumulating several charge and discharge cycles as shown in Figure 21. The number of cycles is defined in the parameter: sensor->accumulation cycles.

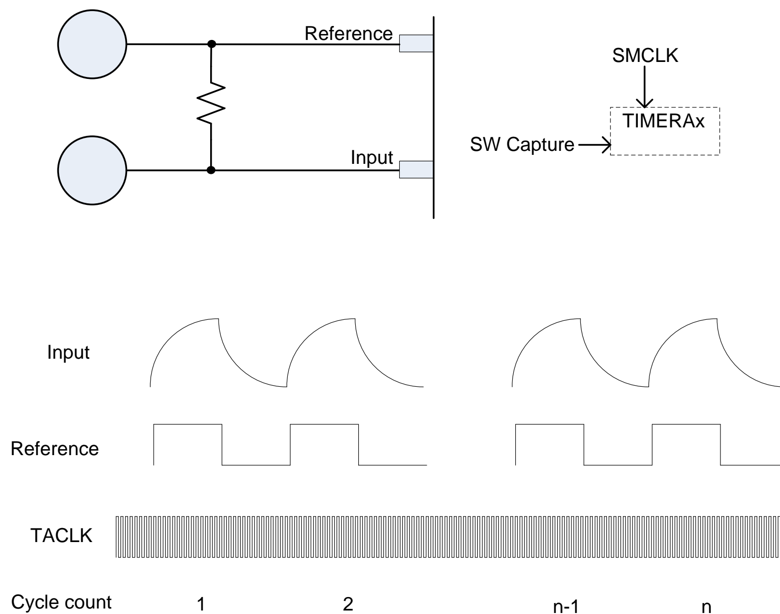


Figure 21. RC Timing Parameters

The following sensor definition describes a sensor made up of two elements that are measured with the RC method. The gate time for each element is four charge and discharge cycles.

```
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
               };
```


Copyright © 2011–2012, Texas Instruments Incorporated

3.2.2.7 fRO_CSIO_TA2_TA3

The timing source for gate timer, TIMERA2, is part of the HAL definition. The source for the gate timer can be scaled with the *sourceScale* parameter to increase the gate time, but it is recommended to use the /1 setting. The gate time is a function of the scale and the number of oscillations shown as 'n' in Figure 22. The number of oscillations is defined by the parameter *accumulationCycles*.

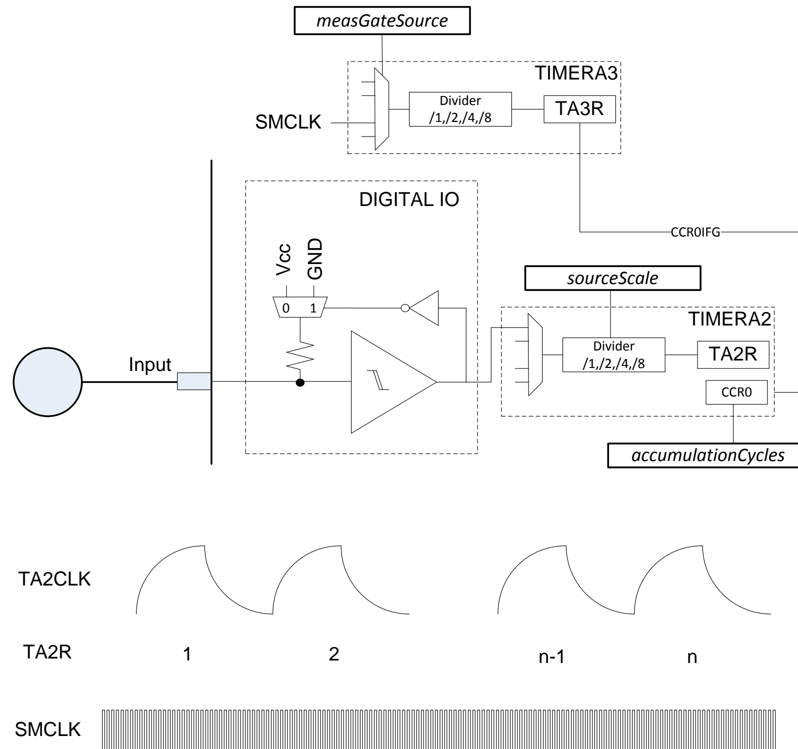


Figure 22. fRO_CSIO_TA2_TA3 Timing Parameters

The timing parameter *measGateSource* is used to select the clock source for the frequency counter, TIMERA3. Typically this source is SMCLK, because it represents the highest resolution clock. In the following code snippet the gate time for each element is 50 relaxation cycles. The number of SMCLK cycles within that gate time represents the capacitance.

```
const struct Sensor keyPad =
{
    .halDefinition = fRO_CSIO_TA2_TA3,
    .inputCapsioctlRegister = (uint16_t *)&CAPSIO0CTL,
    .numElements = 6,
    .baseOffset = 0,
    // Pointer to elements
    .arrayPtr[0] = &key0,
    .arrayPtr[1] = &key1,
    .arrayPtr[2] = &key2,
    .arrayPtr[3] = &key3,
    .arrayPtr[4] = &key4,
    .arrayPtr[5] = &key5,
    .measGateSource = TIMER_SMCLK,
    .sourceScale = TIMER_SOURCE_DIV_0,
    .accumulationCycles = 50
};
```


Copyright © 2011–2012, Texas Instruments Incorporated

3.2.2.8 fRO_PinOsc_TA0_TA1

The timing source for gate timer, TIMERA0, is part of the HAL definition. The source for the gate timer can be scaled with the *sourceScale* parameter to increase the gate time, but it is recommended to use the /1 setting. The gate time is a function of the scale and the number of oscillations shown as 'n' in Figure 23. The number of oscillations is defined by the parameter *accumulationCycles*.

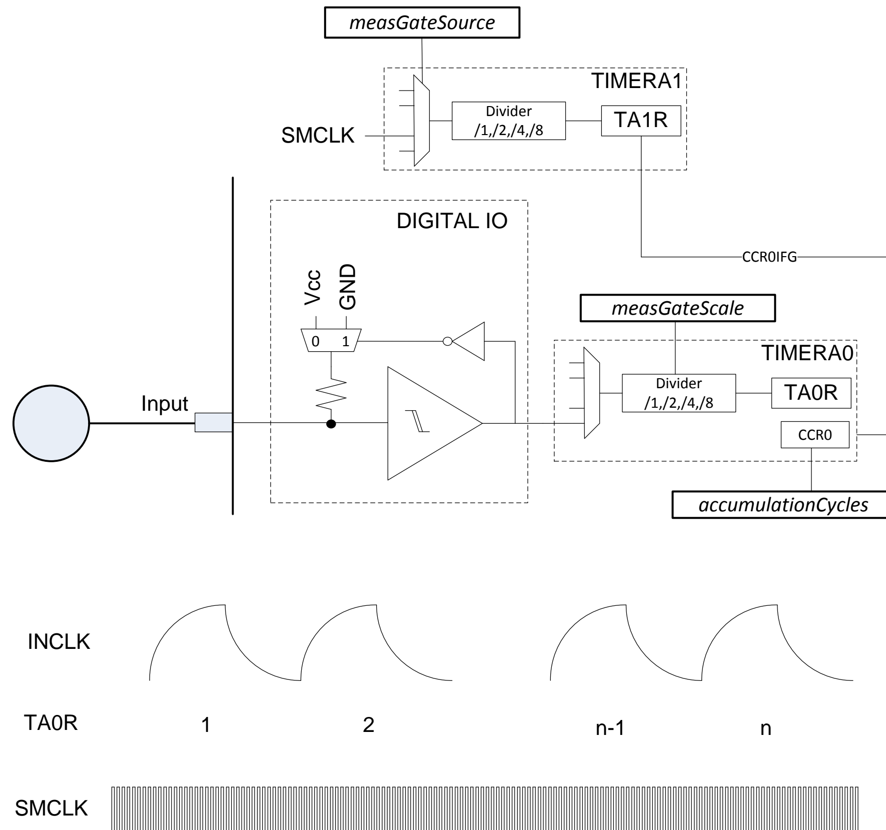


Figure 23. fRO_PinOsc_TA0_TA1 Timing Parameters

The timing parameter *measGateSource* is used to select the clock source for the frequency counter, TIMERA1. Typically this source is SMCLK, because it represents the highest resolution clock. In the following code snippet the number of SMCLK cycles counted during the gate time of 11K relaxation cycles represents the capacitance.

```
const struct Sensor proximity =
               {
                  /* Gate source is INCLK by default */
                  .halDefinition = fRO_PINOSC_TA0_TA1,
                  .numElements = 1,
                  .baseOffset = 0,
                  // Pointer to elements
                  .arrayPtr[0] = &eProx,
                  // Timer Information
                  /* Measurement Source is SMCLK */
                  .measGateSource = TIMER_SMCLK,
                  /* Gate Source divided by 1 */
                  .sourceScale = TIMER_SOURCE_DIV_0,
                  .accumulationCycles = 11000  // Number of RO cycles
               };
```


Copyright © 2011–2012, Texas Instruments Incorporated

3.2.2.9 fRO_PinOsc_TA0_SW

In the case of the fRO_PinOsc_TA0_SW implementation, the variable gate timer is created with the relaxation oscillator and the peripheral TimerAx. The frequency counter is a software loop with a frequency of MCLK/10.

The timing sources are part of the HAL definition therefore the only parameter to define is the number of oscillations for the gate time. This number, shown as 'n' in Figure 24, is defined by the variable accumulationCycles at the sensor level: sensor->accumulationCycles.

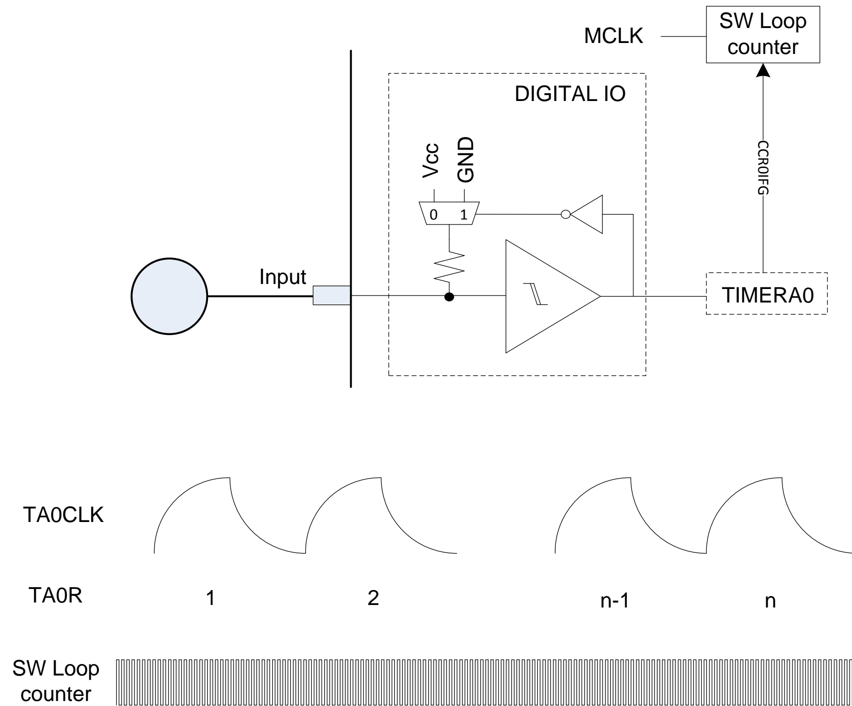


Figure 24. fRO_PinOsc_TA0 Timing Parameters

3.2.2.10 fRO_COMPB_TA1_TA0

The COMP_B implementation is found in the 5xx and 6xx MSP430 families of devices. These families can support clocks speeds up to 25 MHz, further improving the sensitivity demonstrated in Table 3.

The timing source for gate timer, TIMERA1, is part of the HAL definition. The source for the gate timer can be scaled with the *sourceScale* parameter to increase the gate time, but it is recommended to use the /1 setting. The gate time is a function of the scale and the number of oscillations shown as 'n' in Figure 25. The number of oscillations is defined by the parameter *accumulationCycles*.

Copyright © 2011–2012, Texas Instruments Incorporated

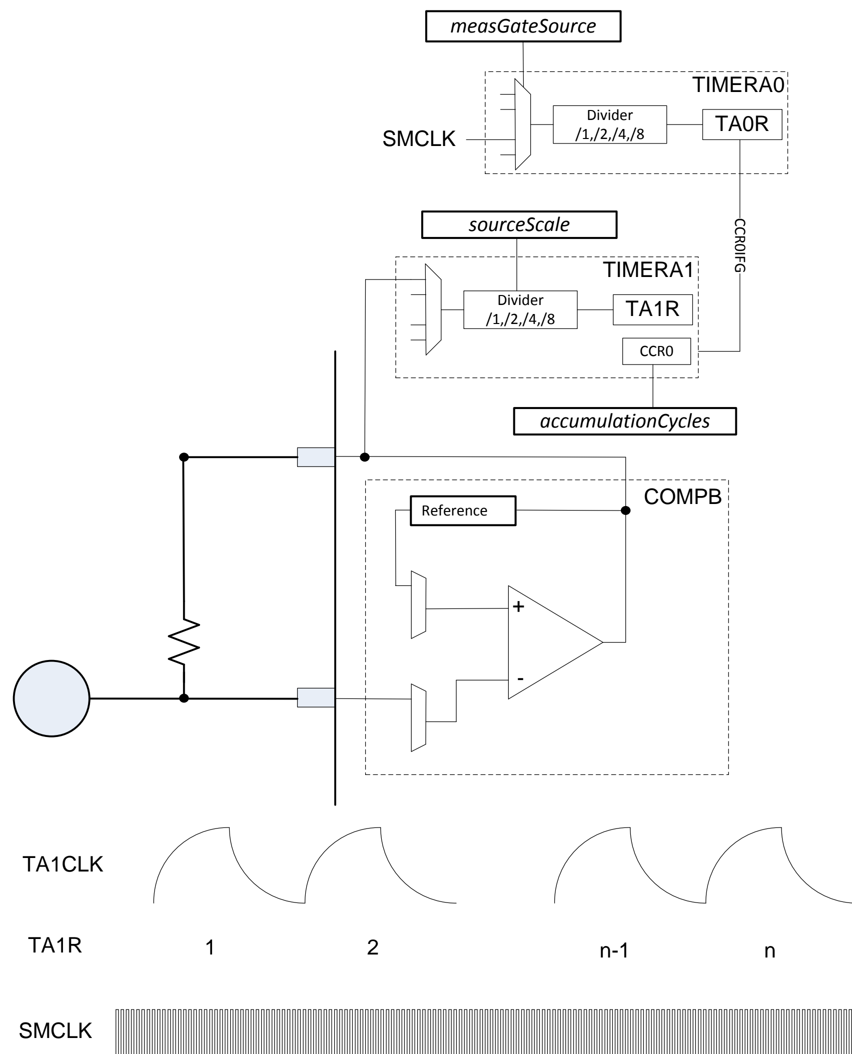


Figure 25. fRO_COMPB_TA1_TA0 Timing Parameters

The timing parameter *measGateSource* is used to select the clock source for the frequency counter, TIMERA0. Typically this source is SMCLK, because it represents the highest resolution clock. In the following code snippet the number of SMCLK cycles counted during the gate time of 10 relaxation cycles (per element) represents the capacitance.

Copyright © 2011–2012, Texas Instruments Incorporated

```
const struct Sensor keypad =
{
    .halDefinition = fRO_COMPB_TA1_TA0,
    .numElements = 5,
    .baseOffset = 0,
    .cbpdBits = 0x001F, //CB0,CB1,CB2,CB3,CB4
    .arrayPtr[0] = &element0,  // point to first element
    .arrayPtr[1] = &element1,
    .arrayPtr[2] = &element2,
    .arrayPtr[3] = &element3,
    .arrayPtr[4] = &element4,
    .cboutTAxDirRegister = (uint8_t *)&P1DIR,  // PxDIR
    .cboutTAxSelRegister = (uint8_t *)&P1SEL,  // PxSEL
    .cboutTAxBits = BIT6, // P1.6
    /*
     *  Measurement source is SMCLK.  The gate source, TACLK, is fed by
     *  relaxation oscillator and is defined by HAL.
     */
    .measGateSource = TIMER_SMCLK,       //  SMCLK
    .sourceScale = TIMER_SOURCE_DIV_0, // RO/1
    /*
     *  The measurement time is variable and a function of the relaxation
     *  oscillator.  Assuming an untouched frequency of 500Khz, 10 cycles would
     *  be 20us.
     */
    .accumulationCycles = 10
};
```

3.2.2.11 fRO_COMPB_TAx_SW

The fRO_COMPB_TAx_SW implementation has the same hardware description as the RO_COMPB_TAx_WDTA implementation (see Section 3.2.2.2). As already mentioned the key difference between the RO and fRO methods is that the frequency counter and gate timer inputs are switched. The gate timer now is a function of the capacitance being measured and the frequency counter is fed by a fixed frequency (a system clock). In the case of the fRO_COMPB_TAx_SW implementation, the variable gate timer is created with the relaxation oscillator and the peripheral TimerAx. The gate time is a software loop with a frequency of MCLK/10.

The fast RO method is typically used in devices that have multiple timers available, so that the frequency counter function is performed with another hardware timer instead of with a software loop. This not only decreases power consumption (running in LPM0 instead of in active mode), but is also removes the 10x factor associated with the software.

There is only one timing parameter, accumulationCycles. Figure 26 shows how the parameter accumulationCycles is used to accumulate multiple relaxation oscillator cycles in the fRO_COMPB_TAx_SW method.

Copyright © 2011–2012, Texas Instruments Incorporated

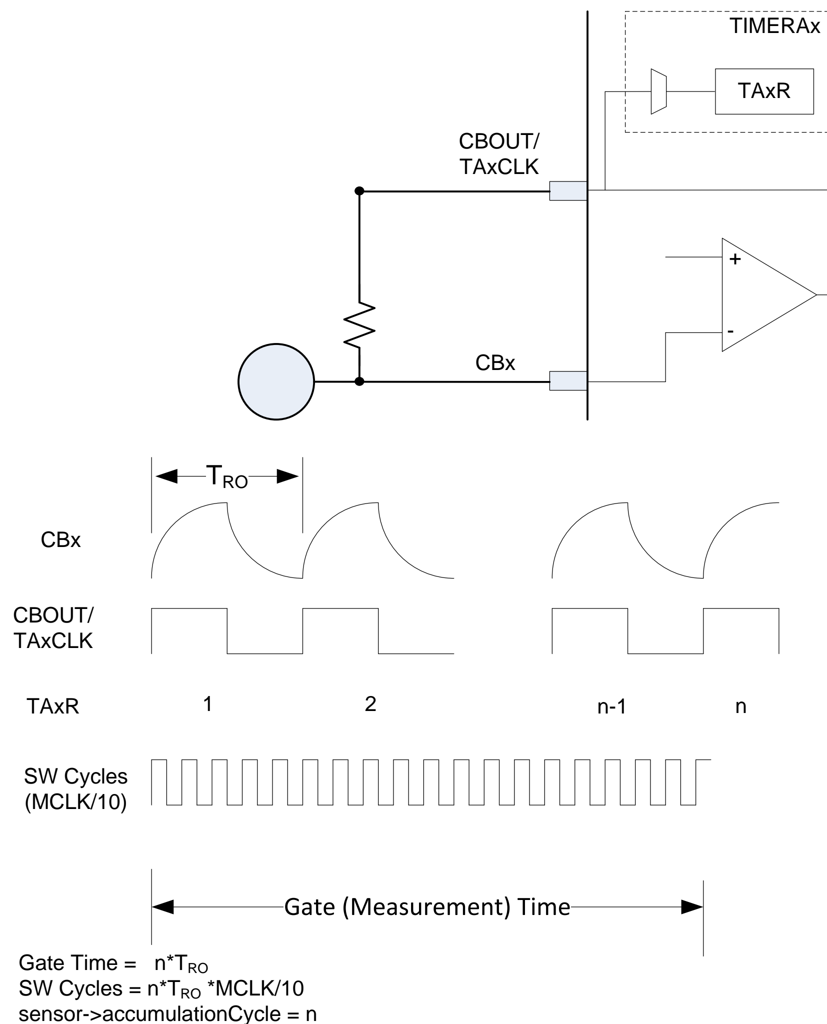


Figure 26. fRO_COMP_B_TAx_SW Timing Parameters

3.2.2.12 fRO_COMPAp_TAx_SW

The fRO_COMPAp_TAx_SW implementations have the same hardware description as the RO_COMPAp_TAx_WDTp implementations (see Figure 17). As already mentioned the key difference between the RO and fRO methods is that the frequency counter and gate timer inputs are switched. The gate timer now is a function of the capacitance being measured and the frequency counter is fed by a fixed frequency (a system clock). In the case of the fRO_COMPAp_Tax_SW implementation, the variable gate timer is created with the relaxation oscillator and the peripheral TimerAx. The gate time is a software loop with a frequency of MCLK/10.

The fast RO method is typically used in devices that have multiple timers available, so that the frequency counter function is performed with another timer peripheral instead of with a software loop. This not only decreases power consumption (running in LPM0 instead of in active mode), but is also removes the 10x factor associated with the software.

There is only one timing parameter; accumulationCycles. Figure 27 shows how the parameter accumulationCycles is used to accumulate multiple relaxation oscillator cycles in the fRO_COMPAp_TA0_SW method.

Copyright © 2011–2012, Texas Instruments Incorporated

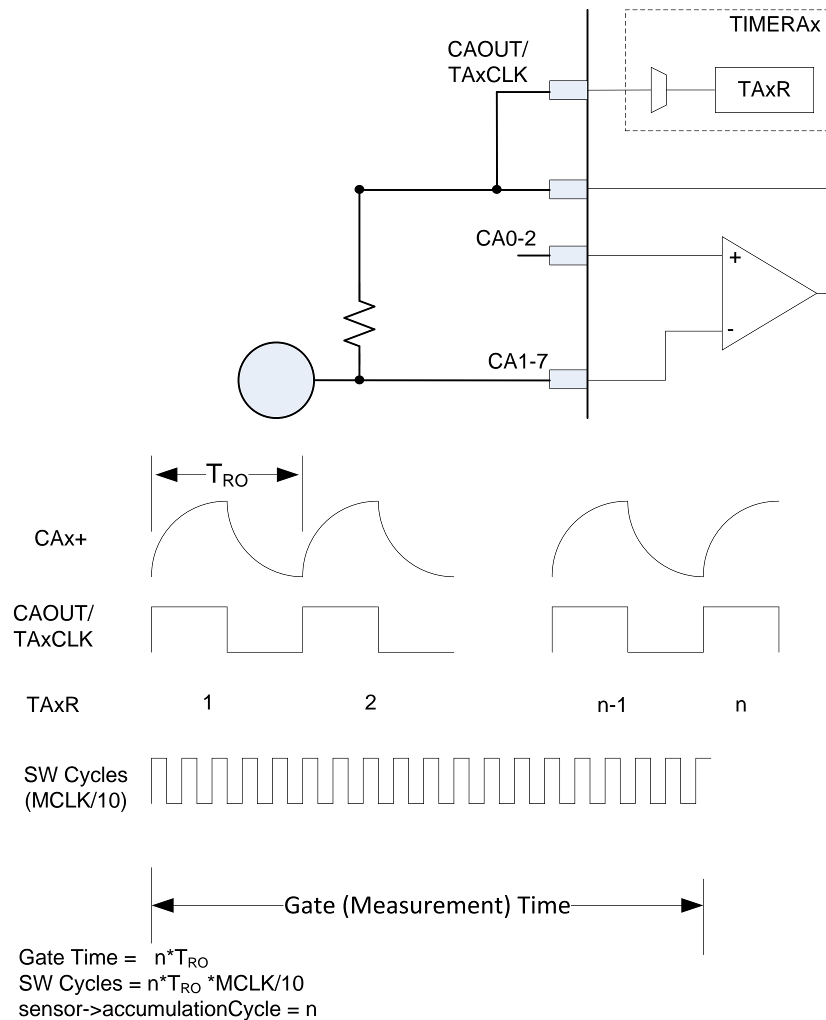


Figure 27. fRO_COMPAp_TA0_SW Timing Parameters

3.2.2.13 *fRO_COMPAp_SW_TAx*

There are three timing parameters; *measureGateSource*, *sourceScale*, and *accumulationCycles*. The gate timer which defines the measurement period is defined by accumulationCycles. The software loop counts the relaxation oscillator cycles until *accumulationCycles* is reached and at this time reads the timer, TAx. The *measureGateSource* and *sourceScale* configure the TAx peripheral. Specifically these parameters define the source (typically SMCLK) and the timer divider (typically divide by 1).

measureGateSource identifies the clock source for TAx. This parameter is the equivalent to the TASSELx bits found in the TimerA control register TACTL.

Table 15. measureGateSource Definitions for fRO_xxxx_SW_Txx

Name	Definition	Clock Source
TIMER_TxCLK	0x0000	TxCLK
TIMER_ACLK	0x0100	ACLK
TIMER_SMCLK	0x0200	SMCLK
TIMER_INCLK	0x0300	INCLK

Copyright © 2011–2012, Texas Instruments Incorporated

sourceScale is used to divide the timer source. This is equivalent to the input divider bits (IDx) found in the TimerA control register TACTL.

Table 16. sourceScale Definitions for fRO_xxxx_SW_Txx

Name	Definition	Clock Source
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 defines the number of relaxation oscillator cycles per gate period. In the fRO_COMPAp_SW_TAx method the counting of relaxation oscillator cycles is done with a software polling loop that looks for a comparator interrupt flag to indicate that an oscillation has occurred.

3.2.3 Slider and Wheel Specific Sensor Variables

The following definitions are only required with the use of API functions TI_CAPT_Wheel and TI_CAPT_Slider.

To include the wheel or slider API within the library the following definitions need to be made in the structure.h file:

```
/ 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
```

In the structure.c file the sensor definitions for *points* and *sensorThreshold* need to be added.

The variable *points* is used to define the number of points along a slider or wheel.

sensorThreshold defines the cumulative response required by the sensor to declare a valid touch. The intent of this variable is to distinguish a genuine interaction with the sensor from an unintentional interaction that may activate only one element.

The wheel or slider *sensorThreshold* is compared with the response of the dominant element and its neighbors (summation of x-1, x, and x+1). The endpoints of a slider are a special case which requires a comparison of only the end element (the dominant element) and the one neighbor. If the response exceeds the sensorThreshold then a valid use case of the sensor has been validated and the position is calculated. If no valid use case is detected, then the ILLEGAL_SLIDER_WHEEL_POSITION, defined in the structure.h file is returned.

4 Resources

Depending upon the configuration this library can consume several different resources making them completely unavailable to the main application or only unavailable during actual measurement cycles. Resources that are completely unavailable are typically the digital IO and allocated memory resources.

The library does perform a simple context save of all the registers used to minimize the need for resetting peripherals. It should be noted that the context save is not extensive and a good practice is to clear IFG flags before enabling interrupts. [1]

[1] An explicit example of this is with TimerA3 where the Library does not use all three capture and control registers however, the CCIFG may be set when the timer is used.

Copyright © 2011–2012, Texas Instruments Incorporated

4.1 Time

The API calls found in the library are 'blocking' calls and do not return the CPU to the application until after the measurement is complete. The dominant factor on how long the CPU is unavailable is the gate time. This time can either be a number of cycles from a fixed (system clock) or variable (relaxation oscillator) clock source. shows some example gate times for various capacitance measurement methods and settings. It is important to note that sensitivity is directly related to the gate time. Shortening the gate time results in a decrease in sensitivity. In the fRO method the sensitivity can be increased (while keeping the shorter gate time) by increasing the fixed system clock frequency as described in Section 2.3.

Table 17. Gate Time Examples

Method	Gate Time Source (.measGateSource)	Interval Definition (.accumulationCycles)	Time (ms)
RO_XXX_YYY_WDTp RO_XXX_YYY_WDTA	ACLK = VLO ~ 12 kHz	64 (WDTp_GATE_64)	5.33
	SMCLK = 2 MHz	8192 (WDTp_GATE_8192)	4.1
	SMCLK = 1 MHz	512 (WDTp_GATE_512)	0.512
RO_PINOSC_TA0	ACLK = VLO ~12 kHz	100	8.3
RC_PAIR_XXX	Rise+Fall (untouched) ~1.4 µs Rise+Fall (touched) ~ 1.6 µs	20 20	0.028 + TBD[(1)] 0.032 + TBD
fRO_COMPx_YYY_SW	RO(untouched) ~ 700 kHz RO(touched) ~ 600 kHz	4000 4000	5.71 6.67
fRO_PINOSC_YYY_SW	RO(untouched) ~ 1.2 MHz RO(touched) ~ 1 MHz	4000 4000	3.334
fRO_COMPx_SW_YYY	RO(untouched) ~ 700 kHz RO(touched) ~ 600 kHz	500 500	0.714 0.833

(1) This additional time is the overhead associated with using software to setup the charge and discharge over several cycles.

4.2 Memory: Flash and RAM

The amount of code space consumed by the library is directly a function of the number of elements, the number of sensors, the measurement method, and the level of abstraction. shows an example of how the code size increases with higher levels of abstraction.

Table 18. Example Flash Resource Allocation

API Calls	Library (bytes)	Configuration Structure: Six Elements, Three Sensors (RO_PinOsc_TA0_WDTp)	Comments
TI_CAPT_Raw	376 (0x178)	114 (0x72)	Optimization level 0 (CCSv5, TI v4.1.2)
TI_CAPT_Init_Baseline TI_CAPT_Update_Baseline TI_CAPT_Custom	1814 (0x716)	114 (0x72)	Optimization level 0 (CCSv5, TI v4.1.2)
TI_CAPT_Init_Baseline TI_CAPT_Update_Baseline TI_CAPT_Custom TI_CAPT_Button TI_CAPT_Wheel	2478 (0x9AE)	120 (0x78)	Optimization level 0 (CCSv5, TI v4.1.2)

RAM can be allocated statically to maintain the baseline tracking feature. The amount of RAM needed is a function of the total number of elements: 2 byte per element. The library uses the TOTAL_NUMBER_OF_ELEMENTS definition to indicate that RAM needs to be allocated for the baseline tracking and how much. When using only the TI_CAPT_RAW API TOTAL_NUMBER_OF_ELEMENTS should not be defined and therefore no RAM resources are consumed.

RAM can also be allocated statically or dynamically to perform the measurements to determine a change in capacitance (TI_CAPT_Custom and sensor abstractions). In the event that the RAM is allocated statically, the definition RAM_FOR_FLASH must be made in the structure.h file. The amount of RAM space allocated is dependent upon the largest number of elements per sensor, 2 bytes per element.

Copyright © 2011–2012, Texas Instruments Incorporated

At the cost of additional FLASH space, this RAM can be allocated dynamically from a HEAP. To allocate the RAM dynamically the RAM_FOR_FLASH definition must be omitted. The HEAP size needs to be set (in the IDE) to 2 bytes plus 2 bytes per number of elements in the largest sensor.

4.3 System Clocks

The library does not make any adjustments to the system clocks (MCLK, SMCLK, or ACLK) and uses them as defined in the application layer for capacitance measurements. It is important to understand the clock usage of the library in the context of the application. For example if the capacitance measurement time is set with watch dog timer interval to 8192/SMCLK, then changing the frequency of SMCLK in the application also changes the measurement time during the capacitance measurement. If the clock source for the capacitance measurement is changed in an application, then it is important to re-initialize the baseline tracking accordingly.

4.4 Peripherals

Different combinations of peripherals can used to measure changes in capacitance. While these peripherals are not available to the application during a capacitance measurement, most of the peripherals can be shared or multiplexed in time with other applications or functions.

4.4.1 TimerA and TimerB

The timer peripheral is reconfigured and initialized with every measurement and therefore can be used for other functions when a capacitance measurement is not taking place.

4.4.2 Watchdog Timer

The Watchdog timer ISR is defined within the library and included with a precompiler directive. This definition prohibits using the WDT peripheral as a true watch dog. However, the source code is provided and the ISR can be modified to support the scheduling of the WDT for different tasks along with capacitive touch at different times within an application.

4.4.3 Comparator_A and Comparator_B

The comparator peripheral is reconfigured and initialized with every measurement and therefore can be used for other tasks within an application when a capacitance measurement is not taking place. It is not recommended to connect other inputs to the capacitive sensor element because this might interfere with the capacitance measurement.

4.4.4 Digital IO

It is not recommended to share or multiplex functions on IO pins that are used for capacitance measurements.

5 API Calls

The library provides three different layers of abstraction. The lowest level of abstraction is the TI_CAPT_Raw API function call. This function call measures the appropriate sensor and provides the raw capacitance measurement to the application layer. The TI_CAPT_RAW function is the most powerful in that it allows the most flexibility in interpretation and application of the capacitance measurement.

The next level of abstraction is the TI_CAPT_Custom API function call. This API calls the TI_CAPT_Raw function and also includes a baseline tracking algorithm. The TI_CAPT_Custom API provides, to the application layer, the magnitude of change of the measured capacitance from the baseline capacitance.

Changes are only provided to the application layer if the change is in the direction of interest. The direction of interest can be either an increase in capacitance or a decrease in capacitance. The direction of interest can be set with the TI_CAPT_Update_DOI API, and the default setting is an increase in capacitance. If the change in capacitance is against the direction of interest this information is used by the baseline tracking but not provided to the application layer.

Copyright © 2011–2012, Texas Instruments Incorporated

The TI_CAPT_Custom API is intended for use with custom sensor designs and applications that are not supported by the other APIs. In these custom applications the baseline tracking can still be used but the interpretation of the change in capacitance must be handled in the application code.

The level of abstraction above the TI_CAPT_Custom API, includes the sensor representation of a button, group of buttons, a wheel, and a slider. These APIs are TI_CAPT_Button, TI_CAPT_Buttons, TI_CAPT_Wheel, and TI_CAPT_Slider, respectively. These APIs include the interpretation and application of the TI_CAPT_Custom function.

lists the APIs of the capacitive touch library. In addition to the APIs just described, there are APIs to adjust the baseline tracking.

Table 19. API Functions

Category	API Function
Capacitance Measurement	uint8_t TI_CAPT_Button(const struct Sensor *);
Capacitance Measurement	const struct Element * TI_CAPT_Buttons(const struct Sensor *);
Capacitance Measurement	uint16_t TI_CAPT_Slider(const struct Sensor*);
Capacitance Measurement	uint16_t TI_CAPT_Wheel(const struct Sensor*);
Capacitance Measurement	void TI_CAPT_Custom(const struct Sensor *, uint16_t*);
Capacitance Measurement	void TI_CAPT_Raw(const struct Sensor*, uint16_t*);
Baseline Tracking	void TI_CAPT_Init_Baseline(const struct Sensor*)
Baseline Tracking	void TI_CAPT_Update_Baseline(const struct Sensor*,uint8_t)
Baseline Tracking	void TI_CAPT_Reset_Baseline_Tracking(void);
Baseline Tracking	void TI_CAPT_Update_Tracking_DOI(uint8);
Baseline Tracking	void TI_CAPT_Update_Tracking_Rate(uint8_t,uint8_t);

5.1 *uint8_t TI_CAPT_Button(Sensor *);*

Inputs: pointer to Sensor that defines a button

Outputs: 0/1,

Function: Measure the button. A 0 means that the change in capacitance is less than or equal to the threshold set in the element and 1 means that the change in capacitance has exceeded the threshold.

5.2 *element * TI_CAPT_Buttons(Sensor *);*

Inputs: pointer to Sensor that defines group of elements where each element represents a button

Outputs: pointer to an element or 0

Function: Measure the sensor (buttons) and determine which button, if any, is being touched. This function returns the pointer to the element that exceeds its threshold by the largest margin (% of the maxResponse - threshold). If no button exceeds its threshold, then this function returns a 0.

5.3 *uint16_t TI_CAPT_Slider(Sensor *);*

Inputs: pointer to Sensor that defines group of elements which makeup slider

Outputs: location on the slider; ILLEGAL_SLIDER_WHEEL_POSITION-> No touch; 0-max -> touch at location where max is defined in the Sensor as the variable points.

Function: Measures the elements within the sensor. This function returns either an invalid number to indicate that no touch was measured or a valid number representing the touch position along the slider.

Copyright © 2011–2012, Texas Instruments Incorporated

```
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
                 };
```

As shown in Figure 28, the order of the elements within the Sensor should represent the order of the elements along the slider. The first element identified within the sensor position represents the lowest value: the outer edge of the first element in the Sensor array is position 0. The last element represents the largest value: the outer edge of the last element in the array represents the resolution number found in the Sensor (points).

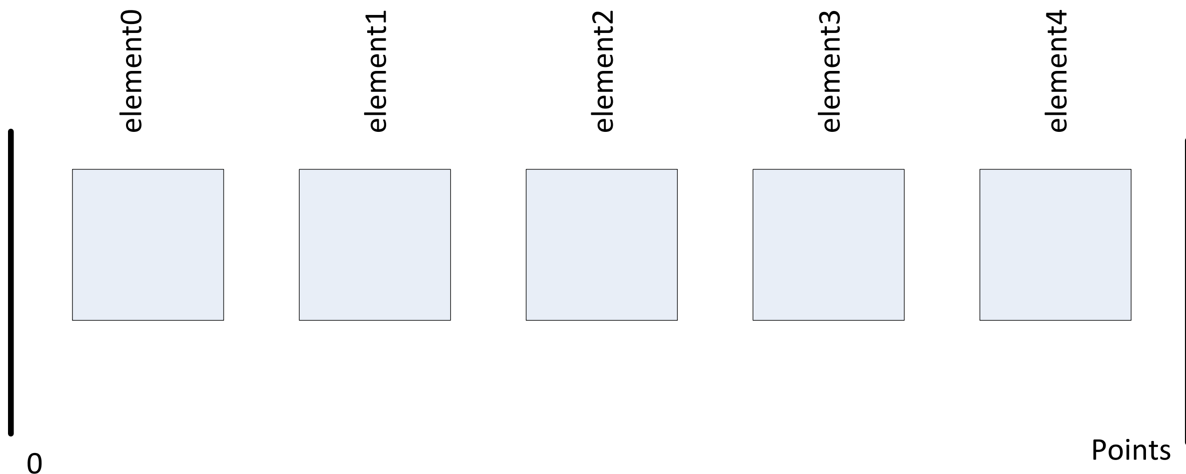


Figure 28. Slider Example

If no element exceeds its threshold (found in the element definition), then this function returns the value ILLEGAL_SLIDER_WHEEL_POSITION, which is defined in the structure.h file.

5.4 *uint16_t TI_CAPT_Wheel(Sensor *);*

Inputs: pointer to Sensor that defines group of elements which makeup wheel

Outputs: location on the slider, ILLEGAL_SLIDER_WHEEL_POSITION-> No touch

0-max -> touch at location where max is defined by Sensor structure definition 'points'

Function: Measure the elements within the sensor. This function returns either an invalid number to indicate that no touch was measured or a valid number representing the position along the wheel.

The order of the elements within the Sensor structure should represent the order of the elements around the wheel. The first element identified within the sensor position represents the lowest value: the outer edge of the first element in the Sensor array is position 0. The last element represents the largest value: the outer edge of the last element in the array represents the point where the value wraps around.

Copyright © 2011–2012, Texas Instruments Incorporated

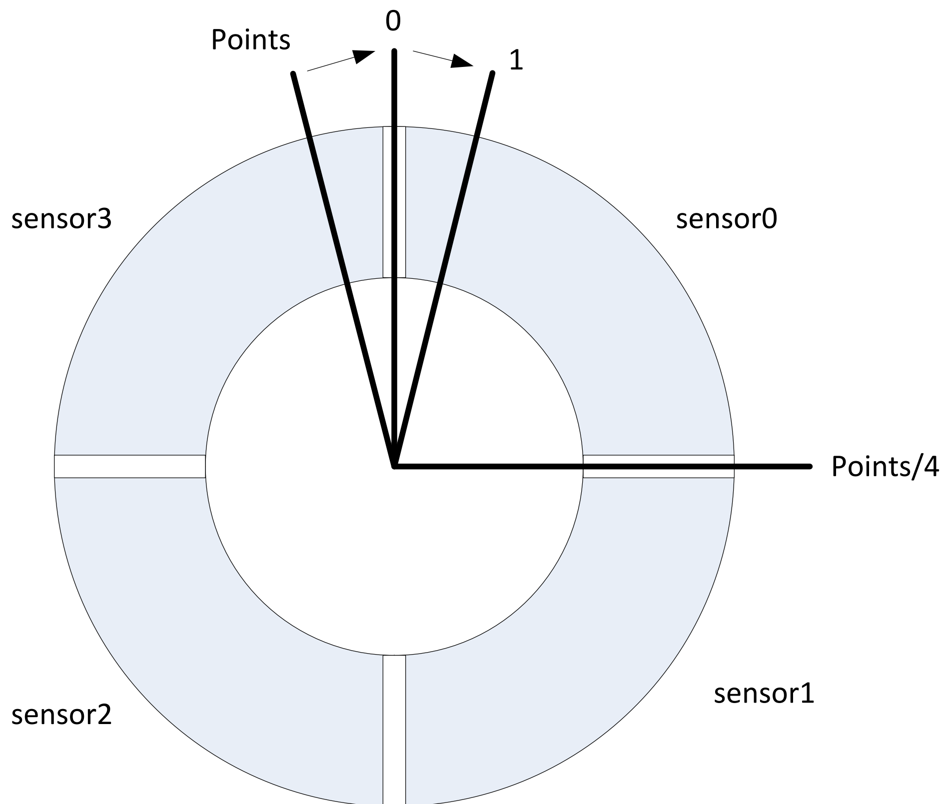


Figure 29. Wheel Example

If no element exceeds its threshold (set in the element structure), then this function returns the value ILLEGAL_SLIDER_WHEEL_POSITION, which is defined in the structure.h file.

5.5 void TI_CAPT_Custom(Sensor *, uint16_t *);

Inputs: pointer to Sensor that defines group of elements which makeup a custom interface and the pointer to the array that is updated with the results of the measurement.

Outputs: None

Function: Measure the change in capacitance relative to the baseline (capacitance history) for each element within the sensor.

The order of the elements within the Sensor structure can be arbitrary but must be understood between the application and configuration. The first element in the array corresponds to the first element within the Sensor structure.

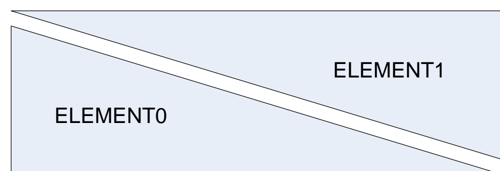


Figure 30. Custom Slider Example

This function requires the application to allocate an array which this function can fill when called. This type of API is useful when the function of the sensor needs to be controlled within the application layer but the measurement and baseline tracking can still be managed by the library.

Copyright © 2011–2012, Texas Instruments Incorporated

5.6 void TI_CAPT_Raw(Sensor*, uint16_t*);

Inputs: pointer to Sensor that defines group of elements which makeup a custom interface and the pointer to the array that is updated with the results of the measurement.

Outputs: None

Function: Measure the capacitance of each element within the Sensor. This function updates the input array with the timer representation of capacitance.

The order of the elements within the Sensor structure is arbitrary and must be managed by the application and configuration. The first element in the array being passed corresponds to the first element within the Sensor structure.

This function requires that the application allocate an array which this function can fill when called. This type of API is useful when the function of the sensor and baseline tracking needs to be controlled within the application layer but the measurement can still be managed by the library.

5.7 void TI_CAPT_Init_Baseline(Sensor *);

Inputs: pointer to Sensor

Outputs: None

Function: Measure the sensor and directly place measured values into the associated baseline variables. Using this function loads the measurements into the RAM space for each element within the sensor. Various functions automatically average the current measurement with the existing baseline function and may cause erroneous performance until the tracking algorithm reaches a steady-state value representative of the environment. It is recommended to use this API when the state of the RAM space is unknown, such as on power up.

5.8 void TI_CAPT_Update_Baseline(Sensor *, uint8_t);

Inputs: pointer to Sensor and the number of measurements to average with baseline.

Outputs: None

Function: Average baseline with number of measurements defined in input. The purpose of this function is to take measurements solely for updating baseline value for each element within the sensor.

5.9 void TI_CAPT_Reset_Tracking (void);

Inputs: None.

Outputs: None

Function: Reset the baseline tracking so that the direction of interest is an increase in capacitance. Also reset the tracking rates so that the baseline tracks changes in the direction of interest at the slow setting (01) and changes in capacitance against the direction of interest at the fast setting (00). I.e. track decreases in capacitance at the fast setting and increases in capacitance at the slow setting.

5.10 void TI_CAPT_Update_Tracking_DOI (uint8_t);

Inputs: The direction of interest.

Outputs: None

Function: If the input is true (non-zero), then the direction of interest is an increase in capacitance. If the input is 0x00, then the direction of interest is a decrease in capacitance. In most applications the direction of interest is an increase in capacitance, because the introduction of an object within a field causes an increase in capacitance. In some situations it is beneficial to identify when an object is present and then change the direction of interest to detect when the object is removed. This is typically useful in applications where the object is stationary for long periods of time.

Copyright © 2011–2012, Texas Instruments Incorporated

5.11 void TI_CAPT_Update_Tracking_Rate (uint8_t);

Inputs: The rate of how quickly the baseline adjusts to changes in capacitance that are in the direction of interest and against the direction of interest.

Table 20. Update Tracking Rate Format

Input Value	Tracking Rate in Direction of Interest	Tracking Rate Against Direction of Interest
0000 0000b	Very Slow	Fast
0001 0000b	Slow (Default)	Fast (Default)
0010 0000b	Medium	Fast
0011 0000b	Fast	Fast
0000 0000b	Very Slow	Fast
0100 0000b	Very Slow	Medium
1000 0000b	Very Slow	Slow
1100 0000b	Very Slow	Very Slow

Outputs: None

Function: Update the tracking rates per Table 20.

6 Establishing Measurement Parameters

The measurement parameters, maxResponse, threshold, and sensorThreshold are impacted by timing parameters selected within the sensor definition. Calibration is an iterative process where the sensor parameters are changed to provide the appropriate response before the measurement parameters are selected.

6.1 Measurement Functions

The TI_CAPT_Raw function does not use any of the measurement parameters and can be used to establish a threshold for the TI_CAPT_Custom function. The TI_CAPT_Custom requires a threshold parameter to disable the baseline tracking when one or more elements within a sensor exceed the threshold. It is important to note that with the raw function, an increase in capacitance is represented by an increase in counts with the RC and fRO methods and by a decrease in counts with the RO method.

```
#include "CTS_Layer.h"

unsigned int raw_data[4];

void main(void)
{
 ...
  while(1)
  {
    TI_CAPT_Raw(&group,raw_data);
    __no_operation();      // set breakpoint here
  }
}
```

Table 21. Example Raw Results With RO Method

Active Element [(1)]	raw_data[0]	raw_data[1]	raw_data[2]	raw_data[3]
None	394	435	426	367
0 (light)	257	424	427	369
0 (normal)	137	410	428	371
0 (heavy)	110	304	420	371
None	390	435	426	367

(1) The difference between a light, normal, and heavy press is the surface area. In applications with a finger, as more pressure is applied the end of the finger flattens creating a larger surface area.

Copyright © 2011–2012, Texas Instruments Incorporated

Table 21. Example Raw Results With RO Method (continued)

Active Element [(1)]	raw_data[0]	raw_data[1]	raw_data[2]	raw_data[3]
1 (light)	367	223	408	367
1 (normal)	361	165	401	366
1 (heavy)	226	117	332	365
None	389	435	425	368
2 (normal)	382	349	146	341
None	390	435	426	267
3 (normal)	388	421	255	111

From , the threshold for elements 0 and 1 can be established from the difference between the interaction and no interaction. A good rule of thumb is half the difference. For example, in this configuration to ensure detection of a light touch on sensors 0 and 1, the thresholds are (394-257)/2 and (435-223)/2, respectively.

6.2 Button and Buttons

Defining the threshold value for the TI_CAPT_Button and TI_CAPT_Buttons abstractions can be done with the either the TI_CAPT_Raw or TI_CAPT_Custom functions. The TI_CAPT_Custom function is preferred simply because the information is provided as a magnitude of change instead of a raw value. The TI_CAPT_Custom function measures the magnitude of change from the baseline that is being tracked by the library. The magnitude of change is only returned for the direction of interest. Changes in the opposite direction are represented by a 0. In the following code example the direction of interest is an increasing capacitance. See the TI_CAPT_Update_Tracking_DOI API for a description on changing the direction of interest.

```
#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
  }
}
```

Table 22. Example Change in Capacitance Results With RO method

Active Element	delta_data[0]	delta_data[1]	delta_data[2]	delta_data[3]
None	0	0	0	0
0 (light)	130	11	0	0
0 (normal)	188	16	0	0
0 (heavy)	287	71	0	0
None	0	0	0	0
1 (light)	14	205	13	1
1 (normal)	30	288	35	2
1 (heavy)	222	328	91	2
None	0	0	0	0

Copyright © 2011–2012, Texas Instruments Incorporated

Table 22. Example Change in Capacitance Results With RO method (continued)

Active Element	delta_data[0]	delta_data[1]	delta_data[2]	delta_data[3]
2 (normal)	3	49	292	24
None	0	0	0	0
3 (normal)	0	5	52	243

The threshold calculation from is similar to that shown with , 130/2 and 205/2.

APIs that use an array of elements, like the TI_CAPT_Buttons API, require the definition of the maxResponse parameter in addition to the threshold. With multiple elements within a sensor the maxResponse is used to normalize the response of each element and identify which element has the dominant response. The purpose of the normalization is to account for possible differences in element performance. As an example, the maxResponse is simply the result from the heavy interaction. Keep in mind the relationship between threshold and maxResponse as described in Section 3.1.1.

6.3 Sensor Arrays: Wheels and Sliders

With the wheel and slider APIs the threshold and maxResponse measurement parameters take on slightly different meanings. The threshold represents the minimum response expected as the interaction first 'slides' into the elements area. The maxResponse represents the maximum return as the interaction slides across the element. This is typically found to be the center of the element which has the largest area overlap between the element and interaction. The following example illustrates how to use the custom function to measure the performance of a slider and determine the values for the threshold and maxResponse variables.

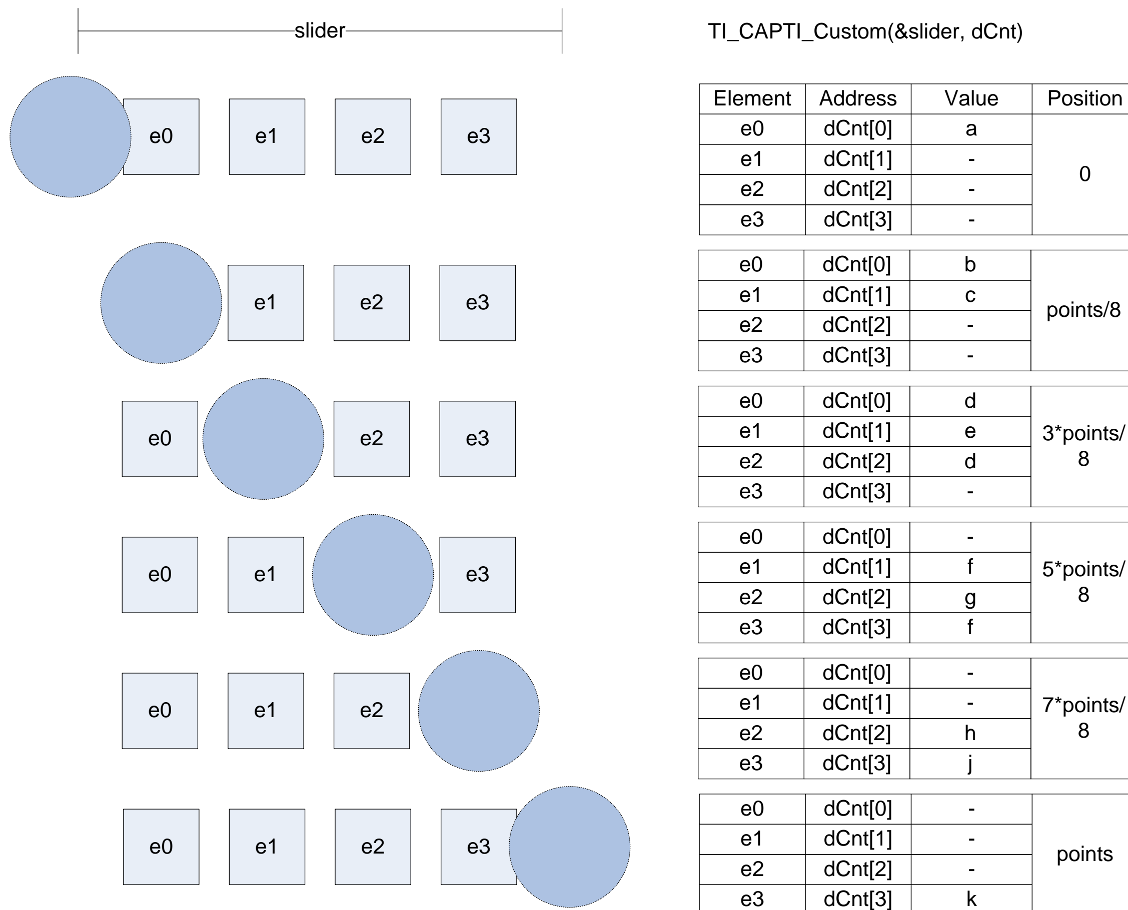


Figure 31. Measurement Example of a Four-Element Sensor

Copyright © 2011–2012, Texas Instruments Incorporated

Ideally the geometry of the electrodes results in equivalent non-zero responses for a, c, d, f, h, and k. More importantly, the response should be greater than 10% of the corresponding maximum return, b, e, g, or j.

Table 23. Measurement Example of a Four-Element Sensor

Element	threshold	maxResponse [(1)]
e0	(a+d)/2	b, or (threshold +655) whichever is smaller
e1	(c+f)/2	e, or (threshold +655) whichever is smaller
e2	(d+h)/2	g, or (threshold +655) whichever is smaller
e3	(f+k)/2	j, or (threshold +655) whichever is smaller

(1) In some geometry the value within maxResponse is not truly the largest return from the electrode but the return recorded at the center of the electrode. The important criteria are that the neighbors (for a slider or wheel) have equal returns.

If the design prohibits meeting these criteria, then one should consider using the TI_CAPT_Custom function and performing the position calculations within the application layer. If the TI_CAPT_Custom function is used then only the threshold value is required as mentioned earlier.

Wheels and sliders also require a third measurement parameter that is part of the sensor structure, sensorThreshold. As described in Figure 32, the sensorThreshold defines the valid area of the wheel or slider. A good starting value for a wheel is 15. Decreasing this value increases the area, but the accuracy is a function of how closely the interaction is to the center line. Conversely, increasing this value reduces the area of interaction more closely to the center line of the wheel. The slider is a special case of the wheel in that the endpoints present a discontinuity in the sensorThreshold calculation. For sliders it is recommended to set the sensorThreshold to 0, otherwise the ends of the slider can be truncated.

Copyright © 2011–2012, Texas Instruments Incorporated

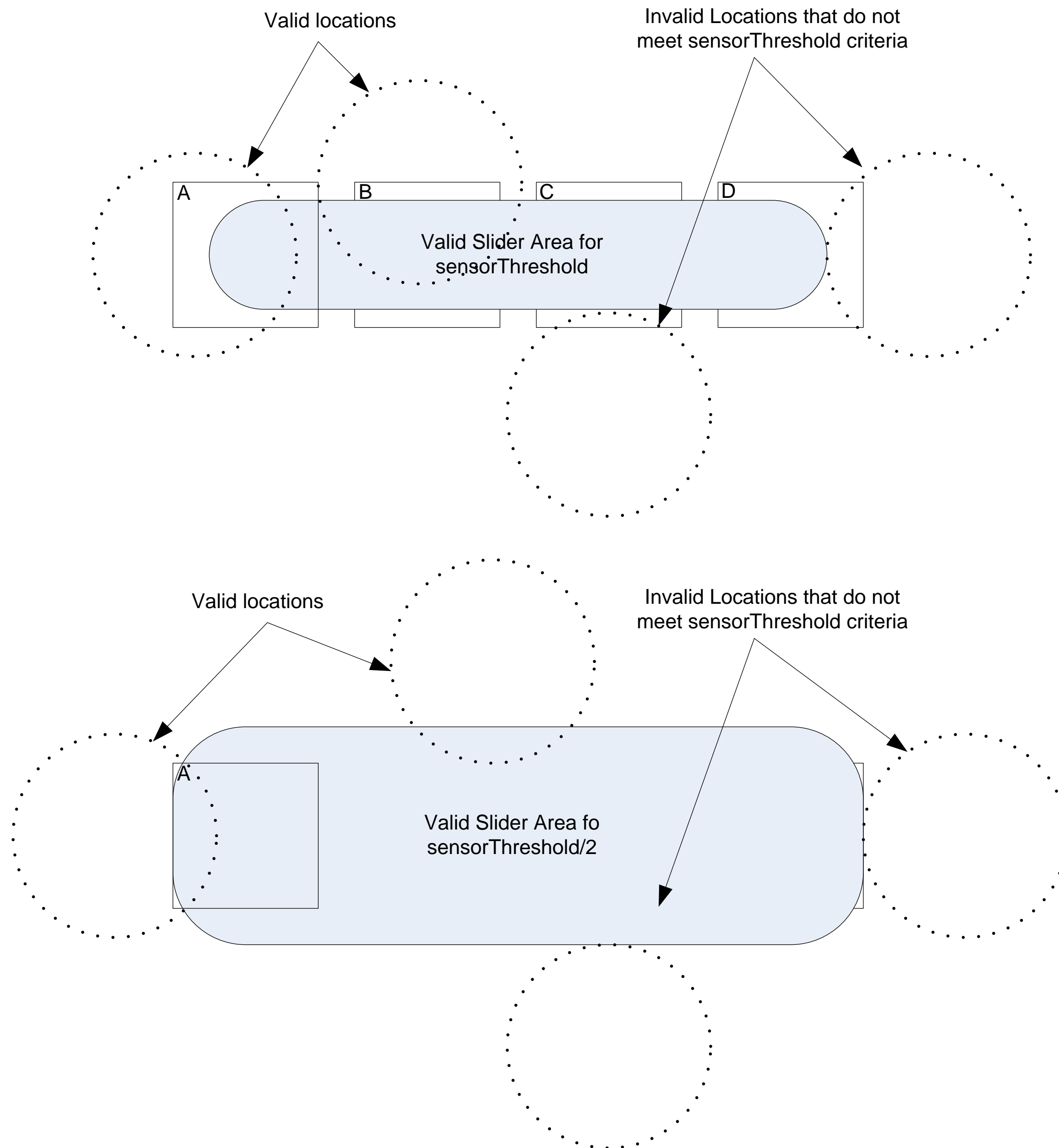


Figure 32. Valid Slider Locations as a Function of the Sensor Threshold

Copyright © 2011–2012, Texas Instruments Incorporated

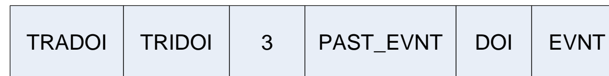
7 CTS_Layer.c and CTS_Layer.h Detailed Description

The functions found in CTS_Layer.c and CTS_Layer.h can be divided up into four groups: Status/Baseline Control Register, baseline tracking, measurement functions, and sensor abstraction.

7.1 Status/Baseline Control Register

A status control register, shown in Figure 33, is provided within the capacitive touch layer and serves as an interface between the application and the library for controlling the baseline tracking. The EVNT and PAST_EVNT fields are not intended to be used by the application but are used within the capacitive touch layer to perform various functions. The DOI, TRADOI, and TRIDOI fields are intended to be accessed by the application and the following APIs are provided for doing so: TI_CAPT_Update_Tracking_DOI () and TI_CAPT_Update_Tracking_Rate ().

```
uint16 ctsStatusReg = 0;
```



Name	Bit(s)	Description
EVNT	0	Event: One of the elements in the group has detected a threshold crossing 0 – No Event has occurred 1 – An event has occurred
DOI	1	Direction of Interest 0 – Increasing Capacitance 1 – Decreasing Capacitance
PAST_EVNT	2	Past Event: An event occurred on the previous scan 0- No event occurred on previous scan 1- An event occurred on the previous scan
	3	Unused
TRIDOI	4-5	Tracking Rate in direction of interest 00 – Very Slow 01 – Slow 10 – Medium 11 – Fast
TRADOI	6-7	Tracking Rate against direction of interest 00 – Very Slow 01 – Slow 10 – Medium 11 – Fast
	8-15	Unused

Figure 33. Status/Baseline Control Register (RAM)

Copyright © 2011–2012, Texas Instruments Incorporated

7.2 Baseline Tracking

Baseline tracking is one of the main features of the capacitive touch layer. The baseline capacitance is followed or tracked to account for any environmental changes that impact the mechanism used to make the capacitance measurement. This includes but is not limited to V_{CC}, temperature, and humidity.

7.2.1 Direction of Interest

The representation of an increase in capacitance is an increase in counts for the RC and FastRO methods while a decrease in counts in the RO method. The purpose of identifying a direction of interest is to establish if the application is looking for an increase or decrease in capacitance. In most human interface applications the direction of interest is an increase in capacitance. The presence of a finger or touch increases the capacitance of an element. Increases in capacitance can also be caused by environmental factors but the assumption is that these changes are relatively slow in comparison to the interaction with a person. Changes in capacitance that are in the direction of interest but are not large enough in magnitude to exceed the threshold may be changes due to the environment. This requires an update in the base capacitance. To insure that these changes are not due to a slow moving object, it is recommended to make adjustments in the direction of interest very slowly. The tradeoff in choosing the adjustment rate is accounting for slow moving objects and rapid environmental changes.

Capacitance changes that are against the direction of interest typically represent only a change in the environment. Because the change can be attributed to the environment without any ambiguity the baseline can be adjusted more dramatically to account for the shift.

7.2.2 Examples of Direction of Interest

An application needs to detect when a block of wood is in place and then removed. The block is typically left in place for several days. The direction of interest is an increase in capacitance to identify when the block is in place and then the direction of interest is changed to a decrease in capacitance to identify when the block has been removed. Once the block is in place any additional increase in capacitance is treated as a change against the direction of interest and the baseline is updated accordingly. In the same way, after the block is removed, if there is a decrease in capacitance this is treated as a change against the direction of interest.

7.2.3 Updating the Baseline Capacitance

Figure 34 shows how baseline updates occur when the change in capacitance is against the direction of interest or under certain conditions when the change of capacitance is in the direction of interest.

A change in capacitance that is in the direction of interest must meet two criteria before it is applied to the baseline tracking. First it must be less than the threshold and second no event within the sensor can occur. A change in capacitance that exceeds the threshold indicates an event. When an event occurs it is possible that the other elements within the sensor are excited even if only by a very small amount. The past event flag, PAST_EVNT, indicates that one of the elements within a sensor has experienced a threshold crossing. Therefore if the PAST_EVNT flag is true within a sensor it is important to suspend baseline updates in the direction of interest.

When the change is against the direction of interest it is limited in magnitude to the threshold/2 before being applied to the baseline.

Table 24 and Table 25 show how the current capacitance measurement and the baseline are weighted to determine the new baseline. The default setting of the library are the 'fast' setting for changes against the direction of interest and 'slow' for changes in the direction of interest.

Copyright © 2011–2012, Texas Instruments Incorporated

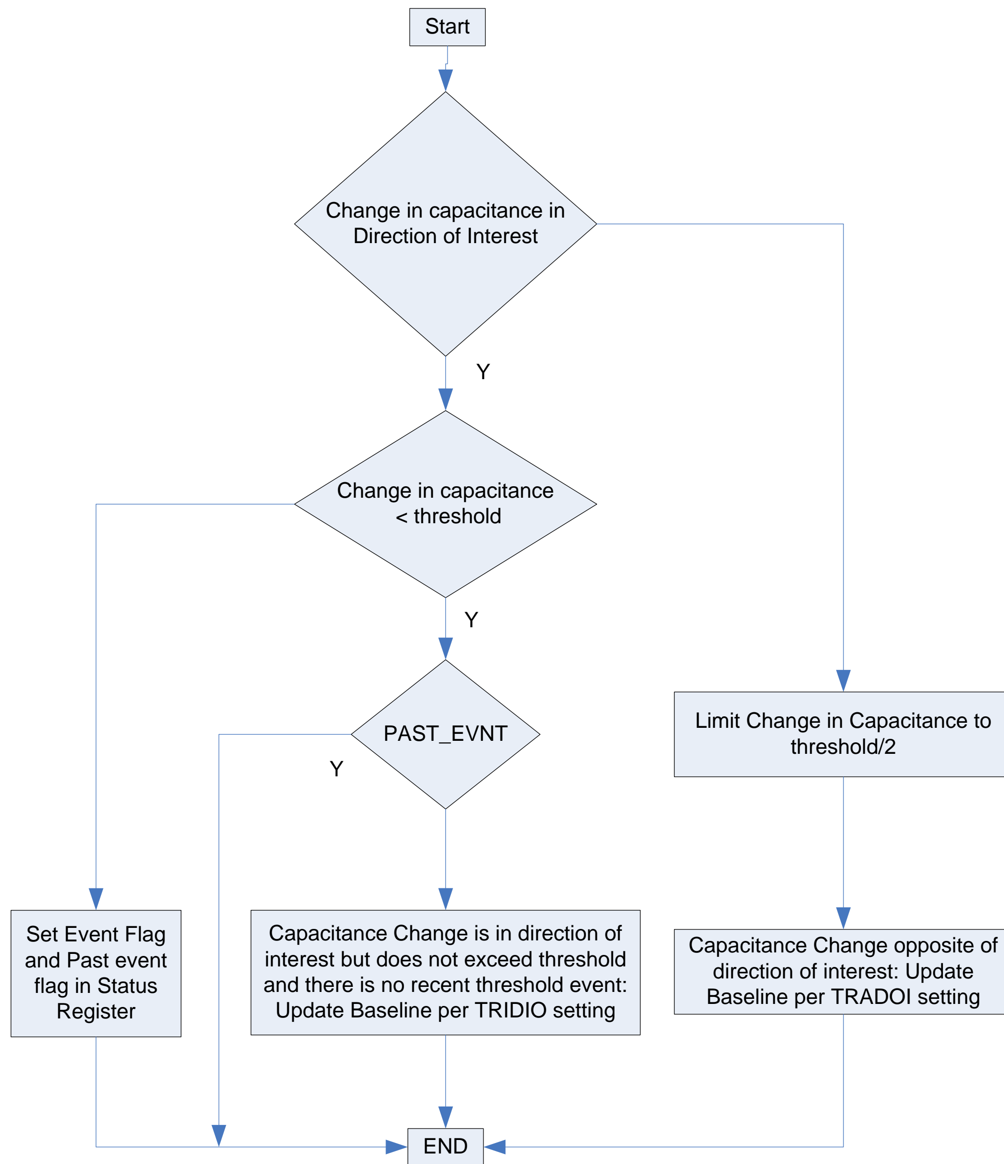


Figure 34. Base Capacitance Update

Table 24. Tracking Settings Against Direction of Interest

Setting	Description	RO (meas < base)	RC, Fast RO (meas > base)
3	Fast	base = meas/2 +base/2	base = meas/2 +base/2
2	Medium	base = meas/4 + 3*base/4	base = meas/4 + 3*base/4
1	Slow	base = meas/64 + 63*base/64	base = meas/64 + 63*base/64
0	Very Slow (Default)	base = meas/128 + 127*(base/128)	base = meas/128 + 127*(base/128)

Table 25. Tracking Settings in Direction of Interest

Setting	Description	RO (meas < base)	RC, Fast RO (meas > base)
3	Fast	base = meas/2 +base/2	base = meas/2 +base/2
2	Medium	base = meas/4 + 3*base/4	base = meas/4 + 3*base/4
1	Slow (Default)	base = base – 2	base = base + 2
0	Very Slow	base = base – 1	base = base + 1

Copyright © 2011–2012, Texas Instruments Incorporated

7.3 Measurement Functions

7.3.1 Delta Measurement + Base Capacitance Tracking: Custom API Call

The 'custom' API measures the change in capacitance for the elements of a given structure. The inputs for the custom API function are the pointer to the sensor and a pointer to the first element of the array in which the capacitance change is recorded. The custom API call measures the capacitance of each element with the 'raw' function.

7.3.1.1 Delta Calculation

The HAL definition (hal_definition) and the direction of interest determine the delta calculation. The hal_definitions are arranged so that all values less than 64 are methods who's count values directly relate to the change in capacitance (that is, an increase in counts means an increase in capacitance) when the hal_definition greater than 64 relate inversely (that is, an increase in capacitance results in a decrease in counts). With the RC and Fast Scan RO methods an increase in capacitance is indicated by an increase in counts. Conversely with the RO method, an increase in capacitance is indicated by a decrease in counts.

The delta calculation performed within the custom API results in either a 0 or non-zero value. The non-zero value is simply the difference between the measured capacitance and the baseline capacitance of a given element. A 0 value indicates that the change in capacitance is opposite (against) the direction of interest.

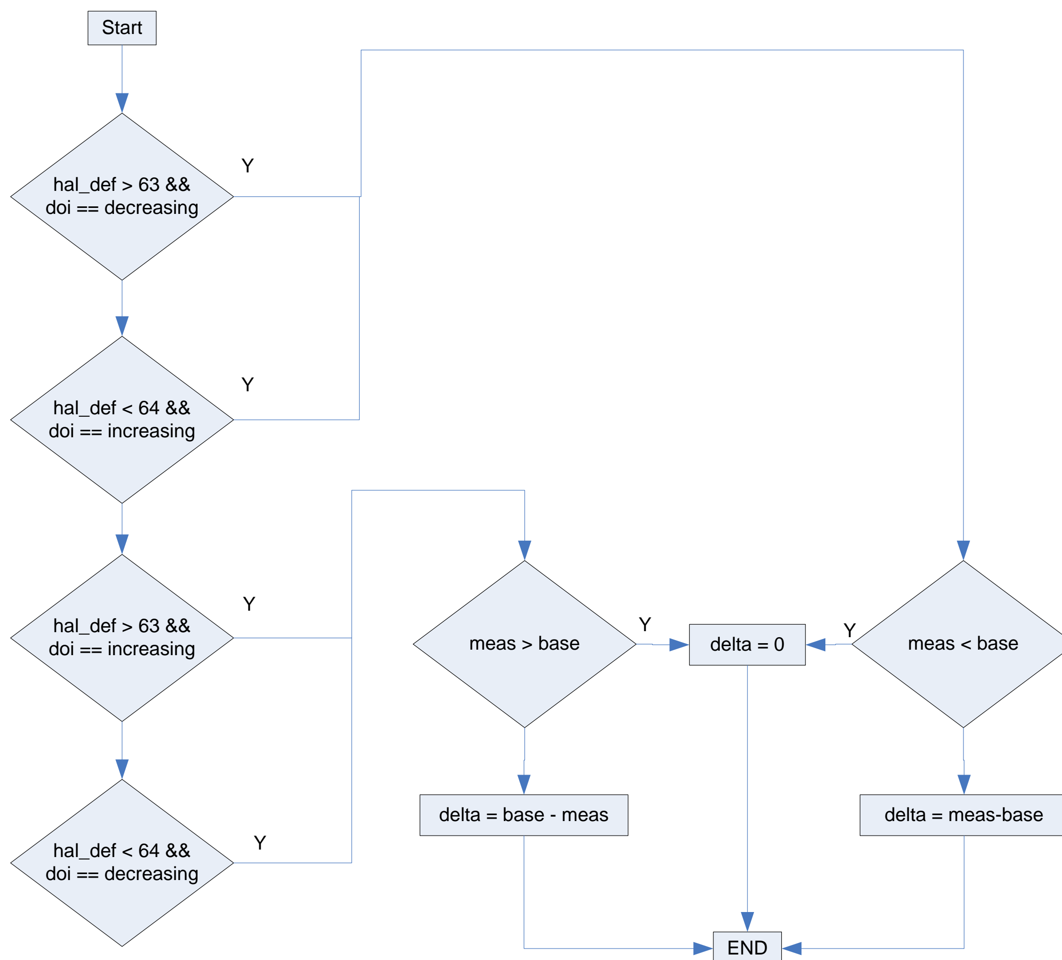


Figure 35. Delta Measurement Block Diagram

Copyright © 2011–2012, Texas Instruments Incorporated

7.3.2 Raw Capacitance Measurement: Raw API Call

The single purpose of the RAW measurement function is to call the appropriate HAL function based upon the user configuration. This function updates the RAM variables provided within the function call, which is used by the higher level calling function.

The 'Raw' feature calls the appropriate HAL function. The HAL definition for a group of elements is found in the sensor structure.

```
sensor0.halDefinition
```

The hal_definition represents a combination of MSP430 peripherals to accomplish the cap touch function.

Table 26. Example HAL Definitions

Name	Type	Number	Measurement Hardware	Measurement Timer	Gate Timer
RC_PAIR_TA0	RC	1	Digital IO	TimerA0	SW
fRO_PINOSC_TA0_SW	fRO	25	Digital IO	SW	TimerA0
fRO_COMPB_TA1_TA0	fRO	31	COMP_B	TimerA0	TimerA1
RO_COMPAp_TA0_WDTp	RO	64	COMP_A+	TimerA0	WDTp
RO_PINOSC_TA0_WDTp	RO	65	Digital IO	TimerA0	WDTp

7.4 Sensor Abstractions

7.4.1 Button, Buttons

uint8_t TI_CAPT_Button(Sensor *);

Inputs: pointer to Sensor that defines button

Outputs: 0/1,

Function: Measure the button. A 0 means that the change in capacitance is less than or equal to the threshold set in the Sensor and 1 means that the change in capacitance has exceeded the threshold.

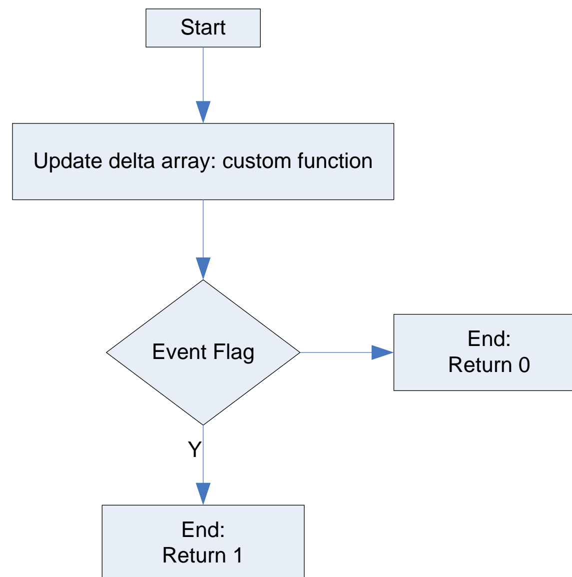


Figure 36. Single Button Algorithm

Copyright © 2011–2012, Texas Instruments Incorporated

Element * TI_CAPT_Buttons(Sensor *);

Inputs: pointer to Sensor that defines group of elements where each element represents a button

Outputs: pointer to an element structure

Function: This function return is the structure pointer to the element that exceeds its threshold by the largest margin: normalized to (maxResponse - threshold value) for each element. If no button exceeds its threshold (set in the element structure), then this function returns a 0 or 'Null Pointer'.

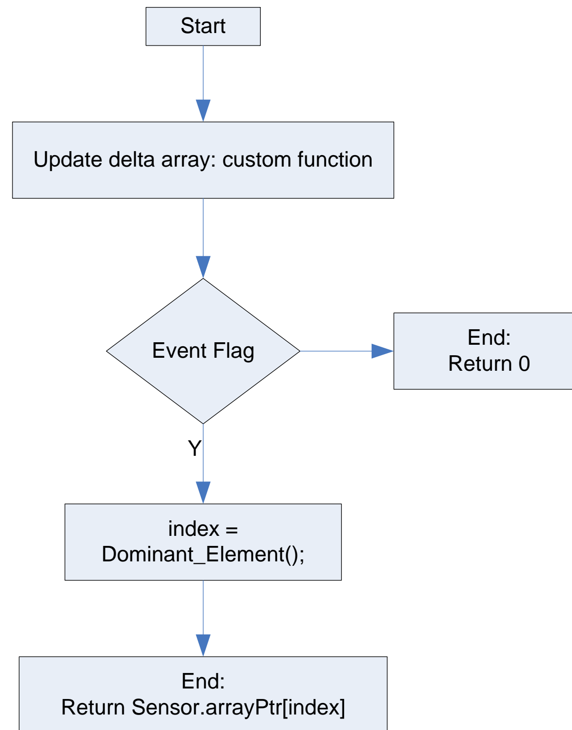


Figure 37. Array of Buttons Algorithm

7.4.2 Slider and Wheel

A wheel or slider is a sensor type consisting of an array of elements. The sensor is divided into a number of points defined by the user. The orientation of the array (first to last) is left to the interpretation of the application. From the perspective of the library the first element within the array definition corresponds to the 0 value on the slider and the last element corresponds to the number of points defined by the user.

The algorithm for the slider and wheel functions is shown in Figure 38. The TI_CAPT_Custom function is used to measure the change in capacitance for each element defined in the sensor. This function also updates the baseline tracking and the event flag status (see Section 7.3.1)

An additional detection mechanism is provided at the sensor level for wheels and sliders. The event flag is the means to determine a threshold crossing at the element level while the *sensorThreshold* variable provides a threshold at the sensor level. The intent of this mechanism is to distinguish a genuine interaction with the sensor from an unintentional interaction that may activate only one element.

Finally the slider and wheel functions calculate the position where the interaction (touch) takes place. The slider and wheel require four types of configuration parameters to present a location. These four parameters are the number of resolvable points, the sensor level threshold, the element level threshold for each element, and the maximum response for each element.

Copyright © 2011–2012, Texas Instruments Incorporated

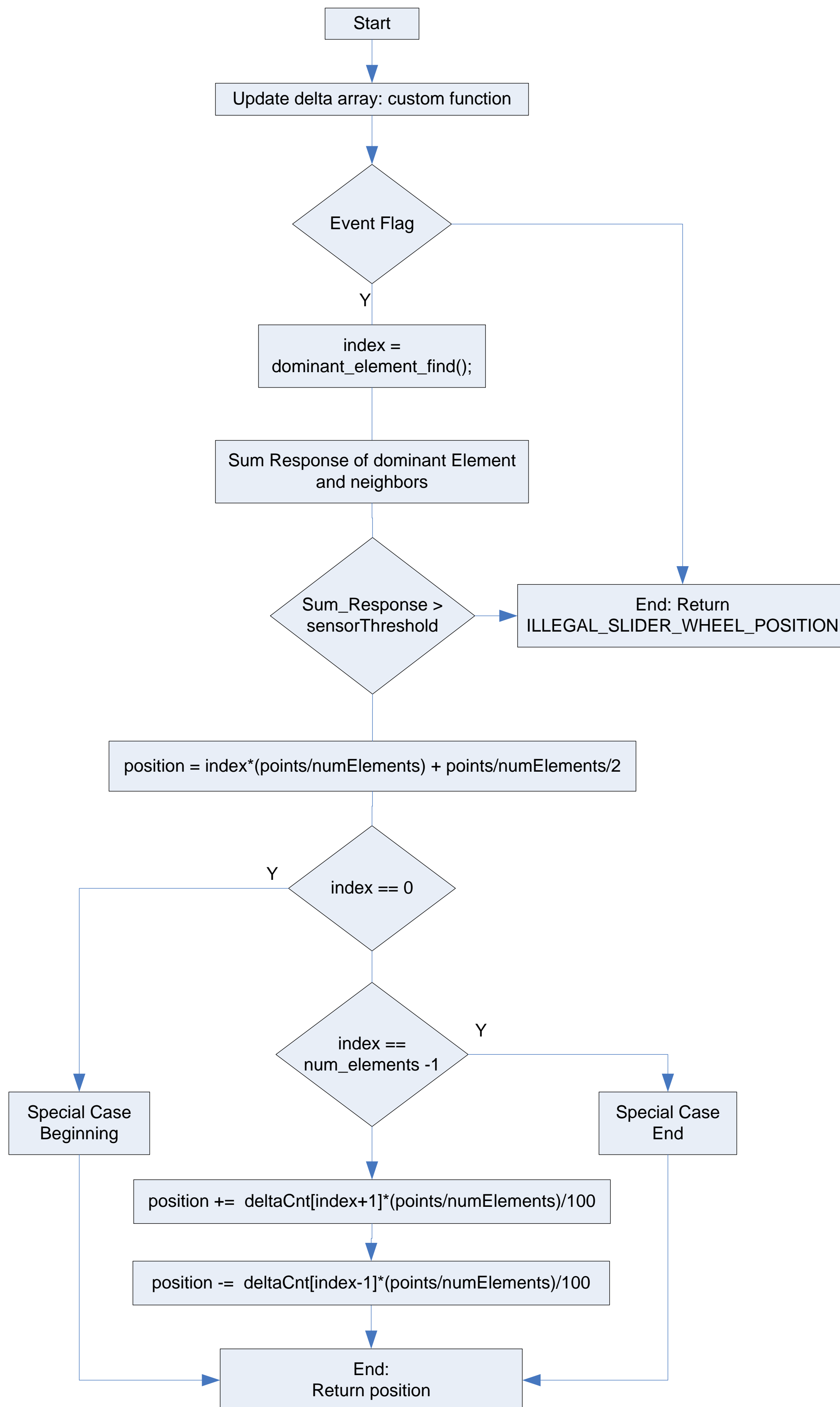


Figure 38. Slider and Wheel Algorithm

Copyright © 2011–2012, Texas Instruments Incorporated

7.4.2.1 Slider Detection

The slider sensorThreshold is a compared with the response of the dominant element and its neighbors. As shown in Figure 39, The endpoints are a special case which requires a comparison of only the end element (the dominant element) and the one neighbor.

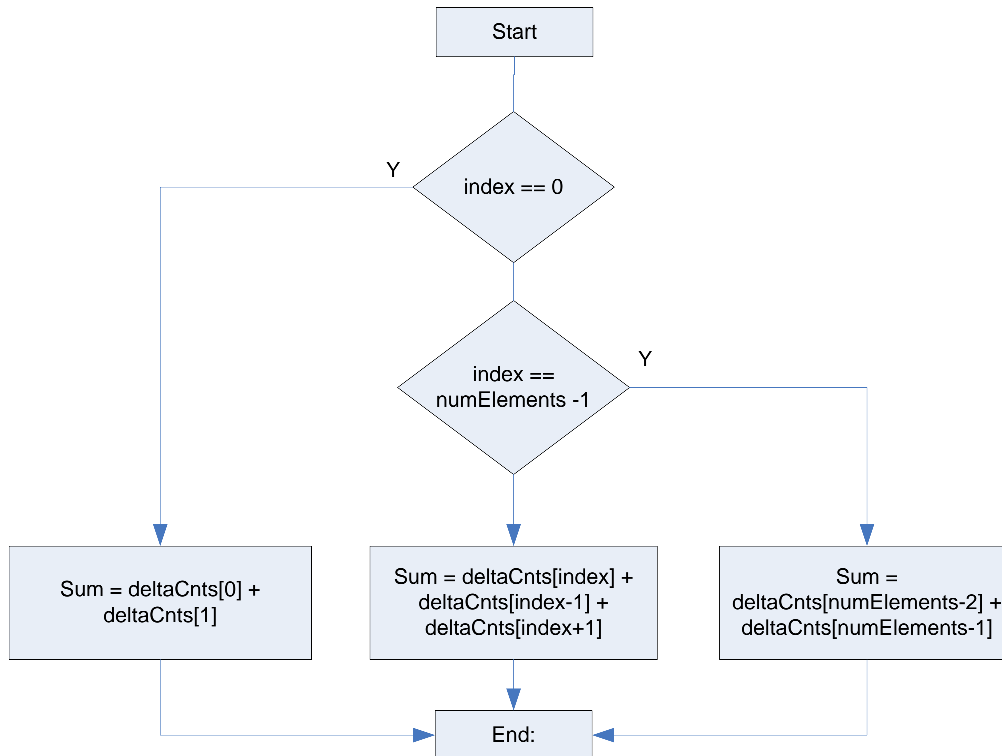


Figure 39. Slider Threshold Detection

As the interaction moves (slides) beyond the center of the last element the contribution from the neighbor goes to 0 and the threshold of the sensor is only a function of the last element. Therefore the sensorThreshold defines how far the finger can deviate from the center position of the ends and still be a part of the slider. As long as the sensor response exceeds the sensorThreshold, the position is calculated.

7.4.2.2 Slider Position

Calculation of the slider position is predicated upon passing the sensor threshold criteria. If the criterion is not met, then the function simply returns a predefined value to indicate no interaction was detected. When there is a valid interaction, the function determines the position from the response of the dominant element and its nearest neighbors. The dominant element function determines the middle element for establishing a 'base' position while one or two neighboring elements are used to pull or weight the final position. In the example in Figure 40, the slider has 64 positions and there are four elements in the slider array.

Copyright © 2011–2012, Texas Instruments Incorporated

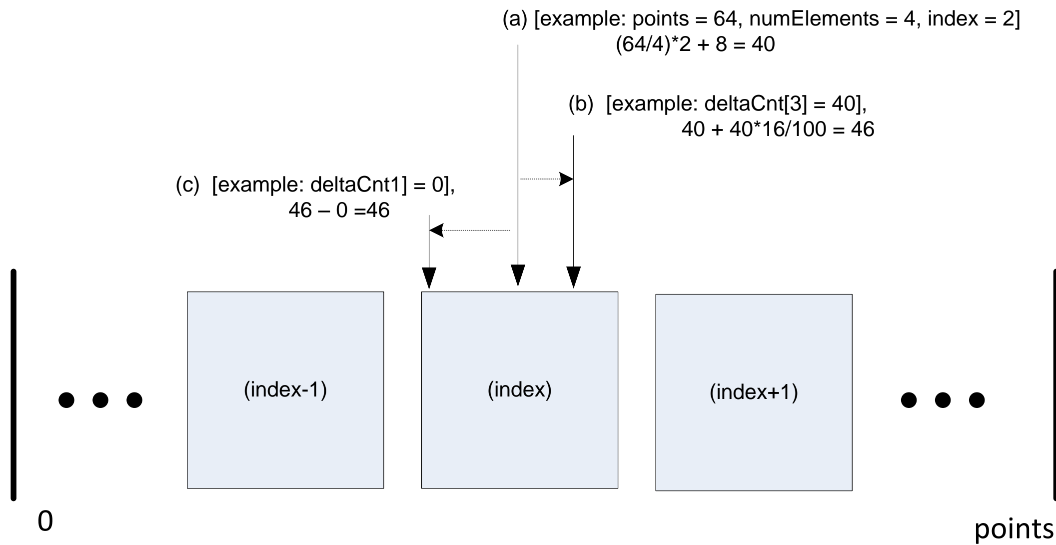


Figure 40. Slider and Wheel Process Middle Algorithm

In the special case where the dominant element is either the beginning or end element of the slider, then only the nearest neighbor is used to weight or influence the position.

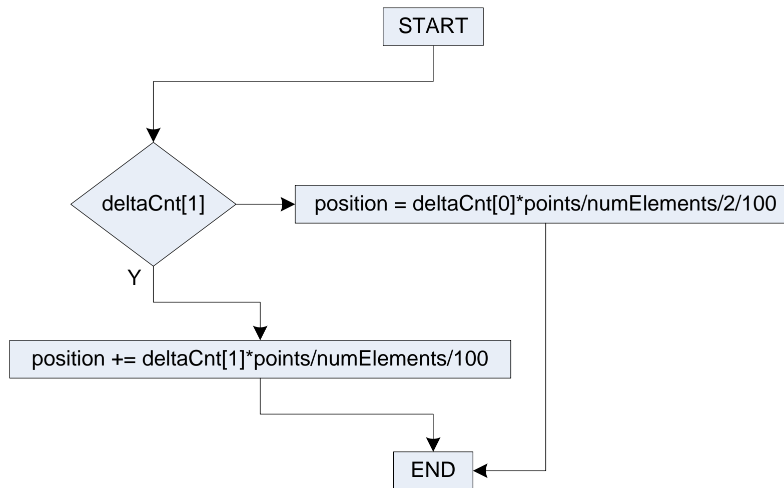


Figure 41. Slider Algorithm: Beginning of Slider

Copyright © 2011–2012, Texas Instruments Incorporated

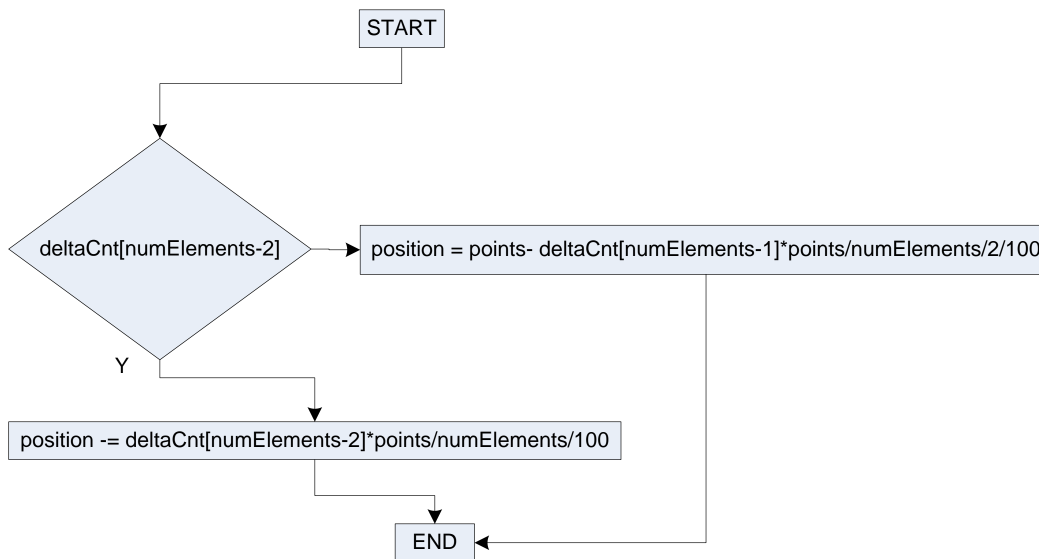


Figure 42. Slider Algorithm: End Of Slider

7.4.2.3 Wheel Detection

The wheel sensorThreshold is a compared with the response of the dominant element and its neighbors (summation of x-1, x, and x+1). The endpoints are a special case which requires the 'wrap around' to be accounted for, see Figure 43. Once the normalized responses of these three elements are added then the value is compared with the sensorThreshold. If the threshold is exceeded, then the function continues on to calculate the position.

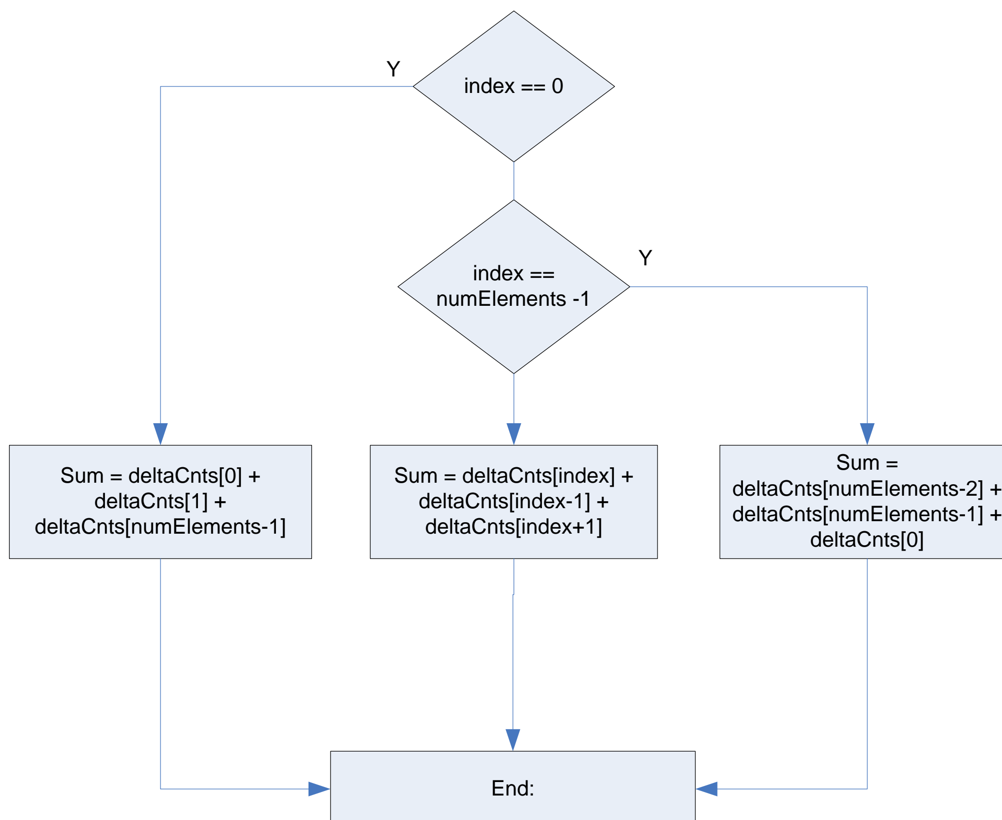


Figure 43. Wheel Threshold Detection

Copyright © 2011–2012, Texas Instruments Incorporated

7.4.2.4 Wheel Position

As previously mentioned the wheel is simply a special case of the slider. Additional handling needs to be put in place to account for the 'wrap around' from the end of the array back to the beginning. Figure 44 and Figure 45 show the algorithm for calculating the position when the dominant element is the beginning and end elements of the array,

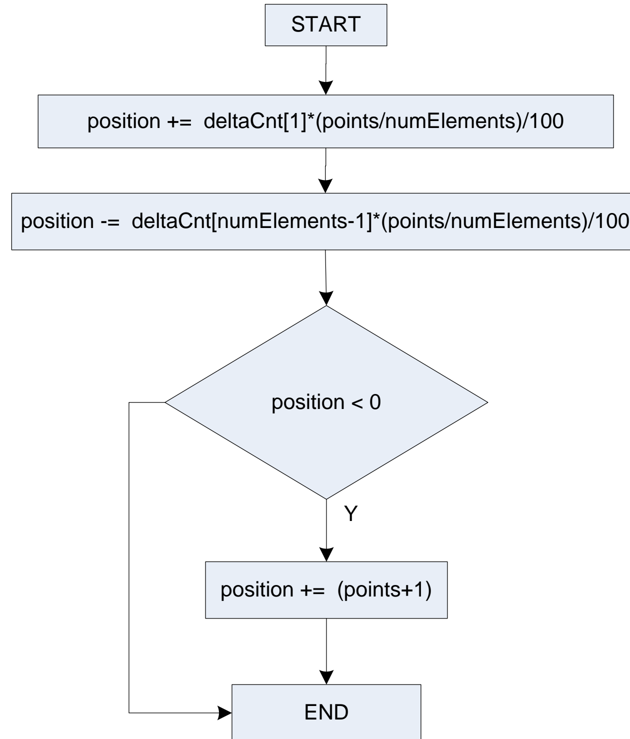


Figure 44. Wheel Algorithm: Beginning

Copyright © 2011–2012, Texas Instruments Incorporated

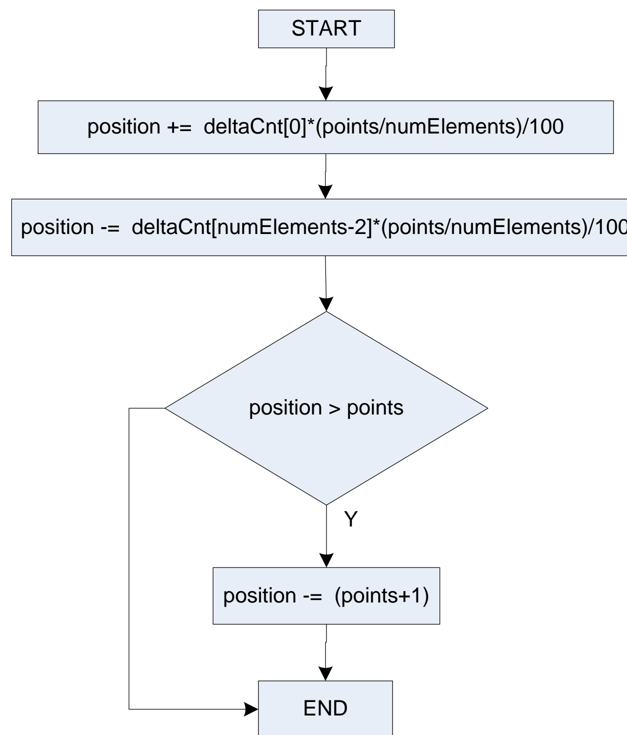


Figure 45. Wheel Algorithm: Ending

7.5 Dominant Element Identification

The identification of a threshold crossing actually takes place in the base capacitance update function (see Section 7.3.1). When a threshold crossing event has occurred then the following is used to determine the dominant element within the sensor structure and scale the response to a range from 0 to 100. A zero indicates that the response is equal to the threshold, and a value of 100 indicates a response equal to the maxResponse.

Copyright © 2011–2012, Texas Instruments Incorporated

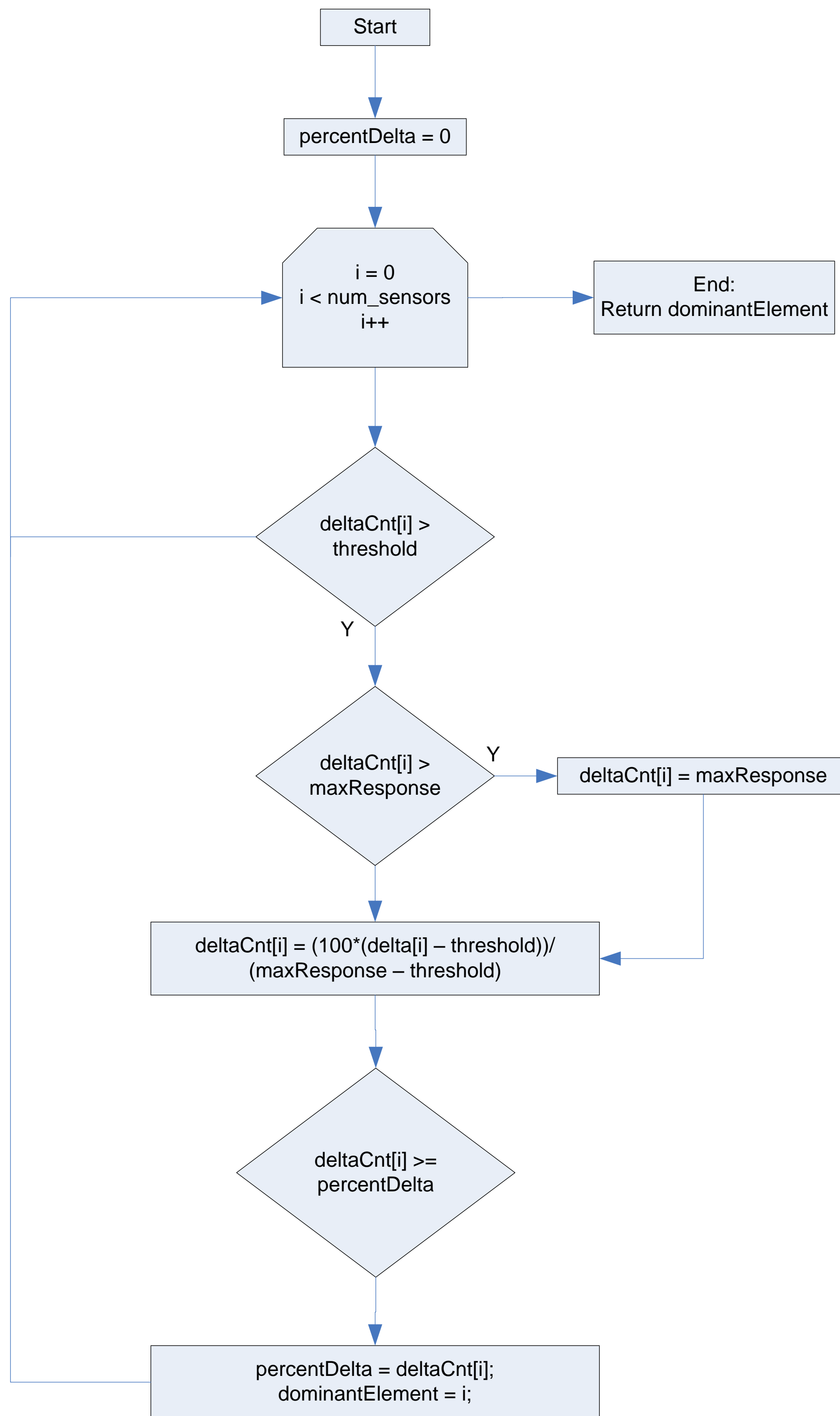


Figure 46. Dominant Element Identification Function

Copyright © 2011–2012, Texas Instruments Incorporated

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2013, Texas Instruments Incorporated