Application Report
SLAA550–September 2012

Capacitive Touch Sensing, SYS/BIOS

Chris Sterzik

ABSTRACT

SYS/BIOS is designed for use in embedded applications that need real-time scheduling, synchronization, and instrumentation. It provides preemptive multitasking, hardware abstraction, and memory management. This application report uses an adaptation of the capacitive touch library to interface with SYS/BIOS. This interface abstracts the capacitive touch interface, promoting use of the graphical configuration tool within SYS/BIOS to define the scheduling of capacitive touch sensing as another task in the system. This document is written for the MSP430F5529 experimenter's board but can be expanded to other devices with the COMPB peripheral.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/slaa550.

Contents

List of Figures

List of Tables

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

Copyright © 2012, Texas Instruments Incorporated

1 Introduction

This application report implements five capacitive touch buttons with the COMPB, TIMERA1, and WDTA peripherals. The detection of a touch is indicated by the LEDs found in the center of the touch electrodes, while raw measurement data and baseline tracking information are displayed upon the LCD screen.

The application (and firmware) can be divided into three pieces. The first piece is taken from the MSP430F5529 Experimenter's Board User Experience code [1]. The MSP430F5529 Experimenter's Board is the hardware and the firmware from the user experience is reused in this application. This includes the drivers for the LCD. The second piece is an adaptation of the capacitive touch library [2]. The capacitive touch portion includes additional features not found in the traditional library but is also very specific in the capacitive touch implementation using the COMPB, TIMERA1, and WDTA peripherals. The capacitive touch portion is also adapted to interface with the third portion of this application, SYS/BIOS.

SYS/BIOS provides a handy graphical user interface (GUI) for defining real-time scheduling. The purpose of this application report is to use SYS/BIOS to integrate capacitive touch sensing with another simple function that updates the LED indicators and the display.

2 Capacitive Touch Sensing

Capacitive touch sensing is achieved on the MSP430F5529 experimenter's board (and more generally with 5xx family devices) by creating a relaxation oscillator with the COMPB peripheral. The oscillation frequency changes as the electrode capacitance changes. The oscillation is fed into TIMERA1 and the number of oscillations within a WDTA interval is compared with previous measurements. A decrease in the number of oscillations indicates an increase in capacitance while an increase indicates a decrease in capacitance. The firmware is designed to treat gradual changes as changes in the environment while instantaneous increases in capacitance are reported as detections.

2.1 Changes in Capacitance

All copper structures on a PCB, either traces or planes, have a capacitance. This capacitance is a function of the coupling between neighboring structures (other planes and traces) and these structures coupling back to earth ground. As shown in Figure 1, the introduction of a human finger is simply another structure that increases the capacitance from its original untouched value.

Copyright © 2012, Texas Instruments Incorporated

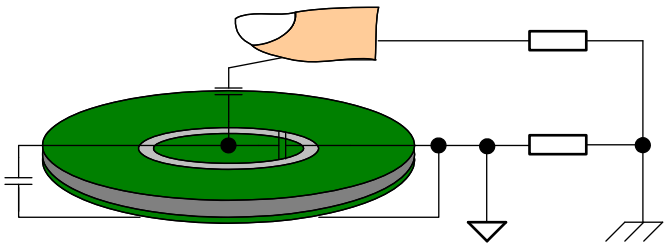

Figure 1. Increase in Capacitance Introduced by a Finger

2.2 Measuring Changes in Capacitance

The capacitive touch implementation is a relaxation oscillator created with the COMPB peripheral. The frequency of the oscillator changes with the capacitance of the electrode. In the MSP430F552x, the CBOUT and TA1CLK signals are internally tied together, so TACCR1 is used to capture the number of relaxation oscillator cycles within a measurement period. The measurement period is defined by the WDTA peripheral used in interval mode.

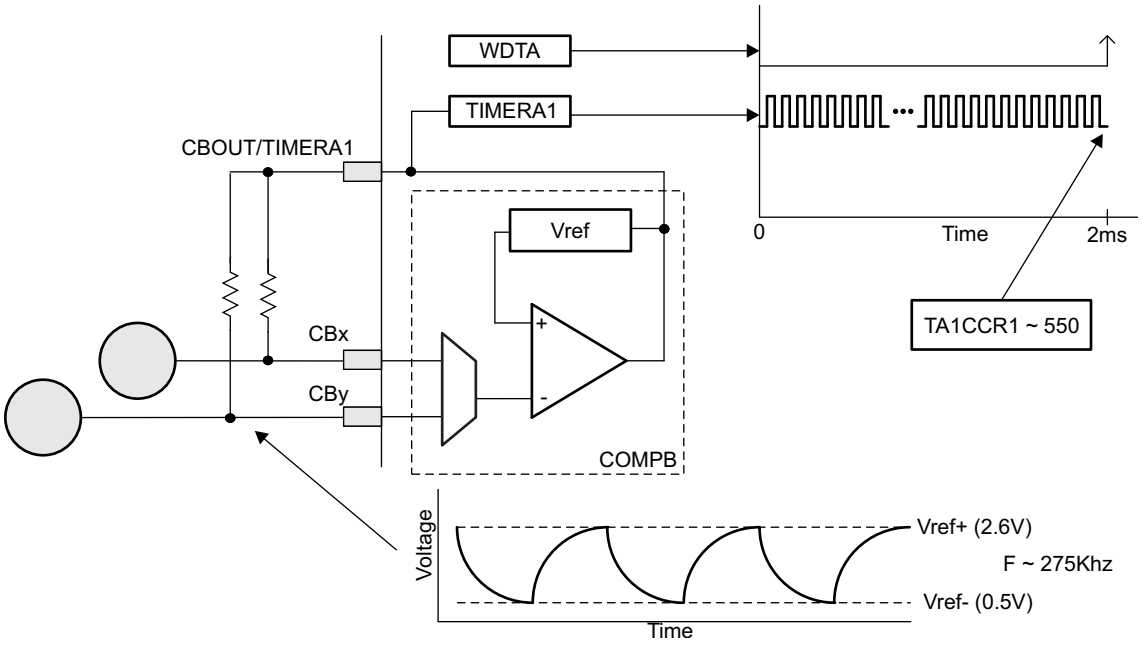

Figure 2. Measuring Capacitance

Figure 2 shows the peripheral implementation of the relaxation oscillator and typical values for the MSP430F5529 Experimenter's board. Without a touch, the oscillation is approximately 275 KHz resulting in 550 oscillations (timer counts) during the measurement period of 2 ms. A touch causes the oscillation frequency to decrease to about 100 KHz resulting in 200 counts.

2.3 Interpreting Changes in Capacitance

The example in the previous section is a straight forward interpretation of a change in capacitance. When the counts decrease by more than 100 counts, the threshold for this application, this change is treated as a possible touch. The change in counts is always relative to a baseline value. The baseline value represents the current untouched capacitance of the electrode. Initially this baseline value can be determined by sampling the electrode and assuming there is no touch or by loading a calibrated value from Flash.

Copyright © 2012, Texas Instruments Incorporated

The detection of a touch is a straight forward comparison between the current measurement and the baseline value. The magnitude of change must exceed the threshold and the change must be a decrease in counts (increase in capacitance). Increasing counts (decreasing capacitance) or decreasing counts that do not meet the threshold criteria are treated as environmental changes. These changes are typically associated with temperature but can also be due to changes in the power supply or noise. The drift compensation algorithm tracks these changes and updates the baseline value appropriately. In the configuration section more detail will be provided on how to adjust how the baseline is updated.

3 Firmware Design

The capacitive touch sensing firmware has been written specifically to interface with SYS/BIOS. In addition to interfacing with SYS/BIOS the capacitive touch sensing uses a custom register structure to describe the physical and performance parameters of a capacitive touch sensor. As shown in Figure 3, the capacitive touch portion is intended to be a black box to the application. The interface with the CTS_SYSBIOS black box consists of three parts that are described in the following three sections: the capacitive touch configuration, the SYS/BIOS configuration, and the result information that the application uses. For example, in this application the result information is used to illuminate an LED and update a display. The SYS/BIOS configuration describes the steps to setup the application as well as the parameter in the configuration that dictates the sensor scan rate. The capacitive touch configuration section describes how both the physical orientation and performance parameters are represented in various software structures.

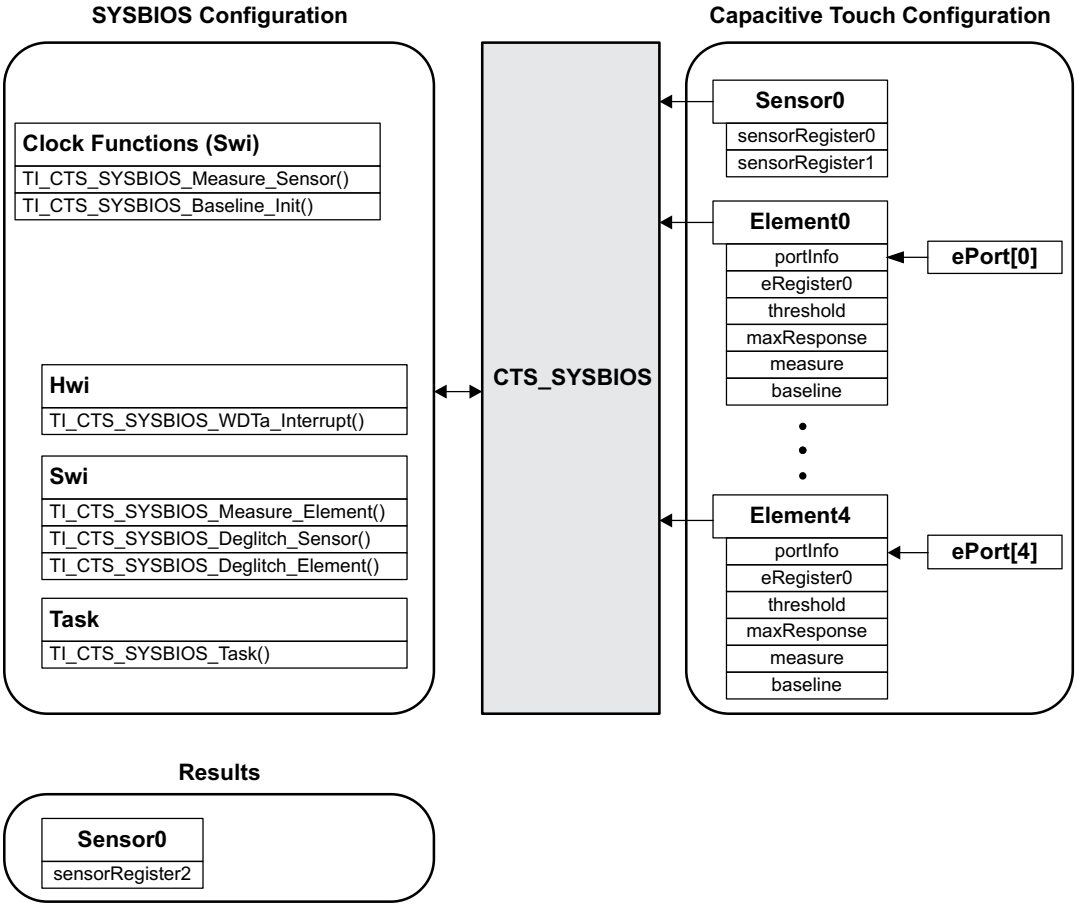

Figure 3. Firmware, Interface, and Configuration

Copyright © 2012, Texas Instruments Incorporated

3.1 Capacitive Touch Configuration

3.1.1 Physical Definition

The physical definition is divided into element and sensor definitions. The element definition defines which COMPB input is being used to measure the electrode capacitance. The sensor definition defines the feedback path for the relaxation oscillator as well as the elements the sensor is comprised of.

Elements are defined within the PortInfo structure eport found in the file structure.c. The definition is simply the inverting input channel selection, CBIMSEL, for control register CBCTL0. For a more detailed description of the control register and its function, see the *MSP430x5xx and MSP430x6xx Family User's Guide* (SLAU208).

The sensor definition is also found in structure.c. The structure combTaxConfig identifies the COMPB inputs that are applied to the COMPB register CBCTL3. This structure also identifies CBOUT pin for the feedback path to the electrode via a resistor. As shown in Figure 2, the CBOUT pin has a shared functionality and is also the input to a Timer. For this example with the MSP430F5529, the CBOUT/TA1CLK signal on P1.6 is used. The port mapping feature of this device can be used to move the signal to another pin, but that is beyond the scope of this document. Another structure, in structure.c, is sConfigFlash. In this structure, the number of elements is explicitly defined as well as which elements from the element definition make up the sensor.

Figure 4 shows the definitions found in structure.c for this five button application.

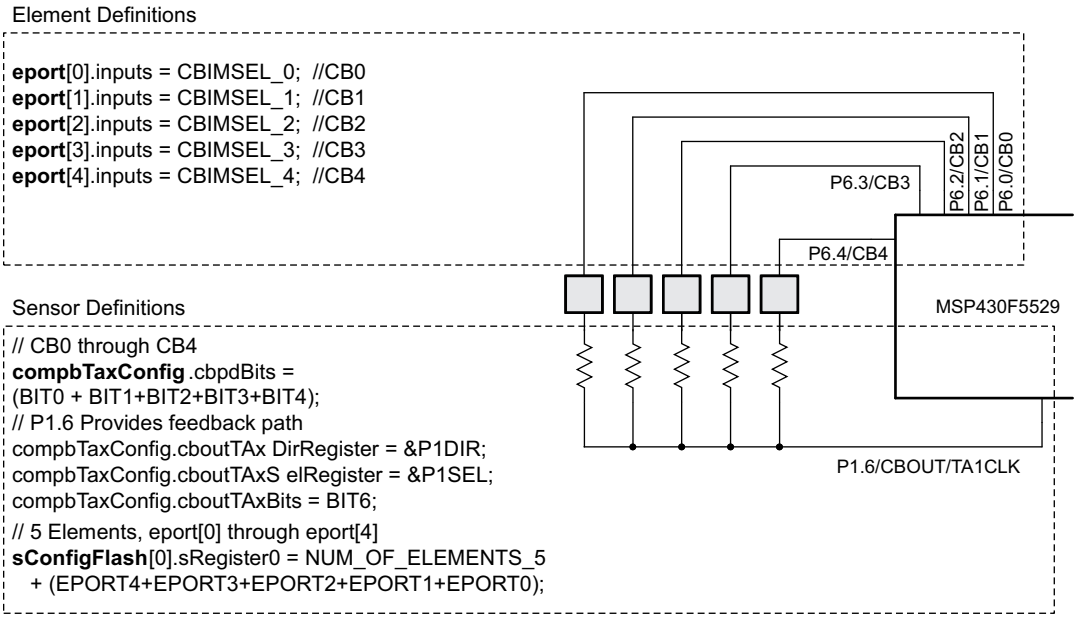

Figure 4. Physical Definition

3.1.2 Performance Settings

The performance settings are divided into three categories: sensitivity, baseline tracking, and noise countermeasures.

3.1.2.1 Sensitivity

The sensitivity of each element is a function of the measurement time and the threshold setting. As previously mentioned, the measurement time is established with the WDTA peripheral in interval mode. The WDTA can be sourced from ACLK or SMCLK and then divided by one of the pre-configured values found in the WDTA control register, WDTCTL [3]. Table 1 shows some example measurement times for various configurations of SMCLK and ACLK.

Copyright © 2012, Texas Instruments Incorporated

Table 1. Example measureTime Settings

measureTime	Value	Source	Frequency	Measurement Time
SOURCE_ACLK + SOURCE_DIVIDE_64	0x0027	ACLK (REFO)	REFO = 32 Khz	2 ms
SOURCE_SMCLK + SOURCE_DIVIDE_512	0x0005	SMCLK(DCO/8)	DCO = 8 Mhz, SMCLK = 1 Mhz	512 µs
SOURCE_SMCLK + SOURCE_DIVIDE_512	0x0005	SMCLK(DCO/2)	DCO = 8 Mhz, SMCLK = 4 Mhz	128 µs

Increasing the measurement time increases sensitivity. The sensitivity should be increased to provide a signal to noise ratio in the range of 10:1 to 15:1. Both the signal and the noise are the deviation from the normal or average capacitance of the system with and without a touch, respectively.

The threshold setting should be set below signal levels used to determine the measurement time. The threshold setting to noise ratio should be on the order of 5:1. In this application, the typical deviation due to a touch is more than 200 counts and the noise is ~2 counts (100:1). The threshold is set at 100 (50:1).

The WDTA settings for the measurement time and the threshold values are found in the FlashElementConfig structure, eConfigFlash. As shown in the code snippet below, the WDTA settings are bit settings within eRegister0 and the threshold is a value.

```
// P6.0, CB0
FlashElementConfig eConfigFlash[TOTAL_NUMBER_OF_ELEMENTS] = {
{
     .eRegister0 = (EREGISTER0_BASEUP_7+EREGISTER0_FILTER_3
+EREGISTER0_DRIFT_4+SOURCE_ACLK+SOURCE_DIVIDE_64),
     .threshold = 100,
     .baseline = 350,
},
```

This structure definition is also found in the structure.c file along with the physical definition.

3.1.2.2 Baseline Tracking Settings

The basis of detection is the magnitude of the current measurement exceeding a reference by a certain magnitude (threshold). This reference is called the baseline and there is a separate baseline value associated with each element.

The firmware provides two mechanisms for initializing the baseline value for each element. The first is the TI_CTS_SYSBIOS_Baseline_Init API. This function measures each element within the sensor and the measured value is used as the baseline. The second is to use the TI_CTS_Copy_Flash_to_RAM API, which copies the baseline value found in Flash to the RAM variables. The first mechanism is useful in applications where the environment changes significantly (supply voltage or temperature) during sleep modes or unpowered periods. The drawback of the first mechanism is that if a finger or other object is present during the initialization these objects (and there associated capacitance) is considered part of the baseline and will not register detection. Once the finger or object is removed, the system recognizes the decrease in capacitance and adjusts the baseline appropriately, and normal operation will resume. The second mechanism does not have this startup limitation since a preprogrammed intial value is used as the baseline. The drawback of the second mechanism is that if there is a large deviation in the environment from the preprogrammed initial baseline the element may report a false detection (1) or appear insensitive until the baseline adjusts to the new environmental conditions.

The baseline needs to accurately represent the ‘untouched’ environment including any changes over time associated with the environment. How the changes are applied to the baseline is managed through the sensor sampling rate and the control bits found in the element definition.

(1) If a false detection occurs, then this halts the baseline update mechanism and the sensor will be ‘stuck’. It is important to select a sufficiently large threshold value to prevent such a condition. To account for variations in oscillation frequencies over temperature and V_{CC}, and set the thresholds accordingly, see the device-specific data sheet tolerances.

Copyright © 2012, Texas Instruments Incorporated

The sensor scan rate and the base update control bits define how often the baseline is updated. A fast baseline update has the benefit of tracking changes in the environment quickly and therefore preserving sensitivity. The cost of tracking changes quickly is the inadvertent tracking of slowly approaching objects or objects that 'hover' around the threshold region. In these cases the baseline tracking can cause the element to become insensitive and only respond to fast approaching objects.

A typical sensor scan rate is on the order of 10 to 20Hz. This application measures the sensor at a 10Hz rate [(2)], which is discussed in the SYSBIOS section. The baseline is updated every 8th consecutive measurement without a sensor detection.

The other two bit settings that control the baseline update are the filter and drift compensation settings. As the name implies, the filter setting applies a filter to the measured data. For this application, the filter limits the current measurement to a range of half the threshold above or below the current baseline and then averages the current measurement with the previous three measurements. After the filtering, the drift compensation is applied.

The drift compensation is a weighting of the filtered response and the current baseline to establish a new baseline. The baseline update rate and filter settings are intended to heavily dampen the response of the baseline tracking. The drift compensation is different in that the response is only heavily damped if the capacitance is increasing. In the case of an increasing capacitance the baseline is given the larger weighting (7/8) while in the case of a decreasing capacitance the filtered response is given the larger weighting.

The following code snippet shows where the bit settings are found in structure.c. These settings are recommended for most human interaction applications. There are applications that require the baseline tracking to be more agile and, therefore, the baseline update cannot be so heavily damped. Section B.1 further describes the element definition settings.

```
// P6.0, CB0
FlashElementConfig eConfigFlash[TOTAL_NUMBER_OF_ELEMENTS] = {
{
     .eRegister0 = (EREGISTER0_BASEUP_7+EREGISTER0_FILTER_3
                    +EREGISTER0_DRIFT_4+SOURCE_ACLK+SOURCE_DIVIDE_64),
     .threshold = 100,
     .baseline = 350,
},
```

3.1.2.3 Noise Countermeasures

The heavy damping provided by the baseline tracking also provides protection from noise corrupting the baseline value. This protection is only applied to measurements that do not meet the detection criteria. Once a threshold crossing is detected, the sensor can perform repeated measurements to confirm the detection and prevent false triggers due to noise. The number of repeated measurements performed to confirm a detection is found in the structure sConfigFlash and are the deglitch bits of senor register1 (sConfigFlash.sRegister1). As shown in the code snippet, the deglitch setting is two. This requires three consecutive measurements (the initial measurement plus two verification measurements) that meet the threshold criteria before detection is indicated. Section B.2 describes the sensor register and the settings.

```
const FlashSensorConfig sConfigFlash[TOTAL_NUMBER_OF_SENSORS] = {
   {
     .sRegister0 = ((NUM_OF_ELEMENTS_5)
     +(EPORT4+EPORT3+EPORT2+EPORT1+EPORT0)),
     .sRegister1 = ((0)+REPRESENTATION_0+DEGLITCH_2)
   }
};
```

(2) Each element takes 2 ms to measure. Five elements result in a 10 ms typical measurement time. The remaining 90 ms can be used to perform other tasks or enter a low-power mode.

Copyright © 2012, Texas Instruments Incorporated

3.2 SYS/BIOS Configuration

A great description of SYS/BIOS can be found in the *TI SYS/BIOS v6.33 Real-time Operating System User's Guide* (SPRUEX3). The threading model provides for a variety of situations. Hardware interrupts (Hwis), software interrupts (Swis), tasks, idle functions, and periodic functions are all supported. Again, the intent is to provide a capacitive touch solution that integrates with SYS/BIOS and ultimately other application tasks.

The Hwi, Swi, and task instances are static and configured using the XGCONFIG tool in Code Composer Studio™. The configuration is broken into two sections, the general setup and the application specific setup. The general setup is presented as a set of instructions to be followed to provide the correct measurement functionality for any capacitive touch applicaiton. The application specific setup will address the timing of the system and specifically the scan rate of the capacitive touch sensing which can vary from application to application.

3.2.1 General Setup

There are three Swis, one Hwi, one semaphore, and one task that need to be setup within SYS/BIOS to provide the capacitive touch functionality. The main goal of this setup is to enable the real time operating system control of the CPU during the measurement phase. Traditionally the MSP430 is held in a low power state during the measurement. With this application the CPU can still be held in a low power state or used for other tasks. The instructions for setup can be found in Appendix A.

Figure 5 shows the Hwi, Swis, and tasks associated with the measurement of all five keys. In Figure 5, the following sequence is shown: the keys are measured, detection was made on the first key, the first key is re-measured twice, and the sensor result fields are updated. The Idle task time between the Swis and Hwis represents the measurement time, which for this application is 2 ms. The Hwis and Swis are significantly shorter but are artificially lengthened in this diagram to improve visibility.

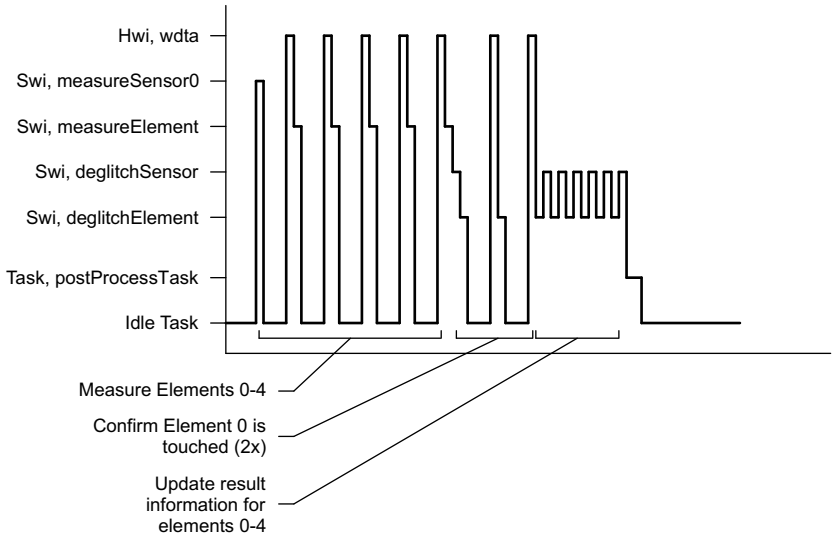

Figure 5. Example Description of Measurement Sequence

The sequence found in Figure 5 is handled by SYS/BIOS and 'hidden' from the application. The start of the sequence is found in the application and discussed in the next section.

3.2.2 Application Specific Setup

The application specific settings in SYS/BIOS are related to timing. Two clock instances are needed to perform the initialization and sensor measurements. The initialization is a 'one-shot' event that initializes the RAM variables for baseline tracking. The measurements are performed periodically after initialization to achieve a scan rate of 10Hz. The following sections describe the SYS/BIOS clock settings and the settings for the initialization and measurement clock instances.

Copyright © 2012, Texas Instruments Incorporated

3.2.2.1 Clock Settings

For most SYS/BIOS Clock settings the default values can be used. The exceptions are the settings for tickPeriod and tickMode. Figure 6 shows a tickPeriod of 10 ms. Therefore the period of 10 ticks associated with the measurement clock instance (Section 3.2.2.3) represents a 10Hz sampling rate. Adjusting the sampling rate can be done through the SYS/BIOS configuration tool by editing either the Tick period or the period of the measurement.

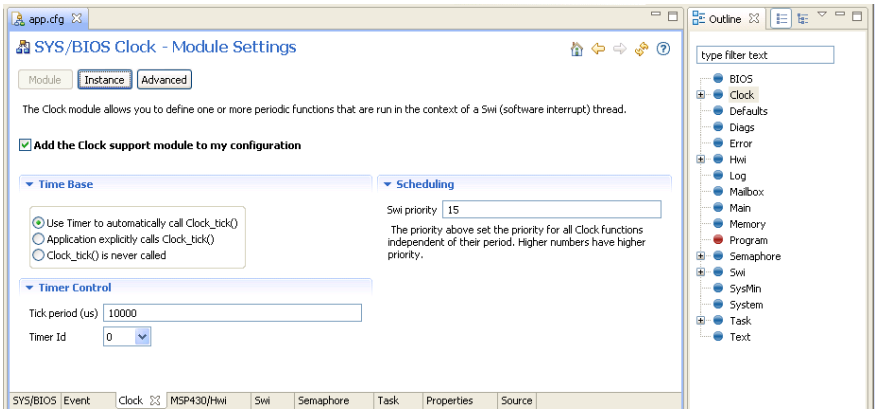

Figure 6. SYS/BIOS Clock Setting for tickPeriod

As shown in Figure 7, the default value of TickMode_DYNAMIC must be updated with TickMode_PERIODIC.

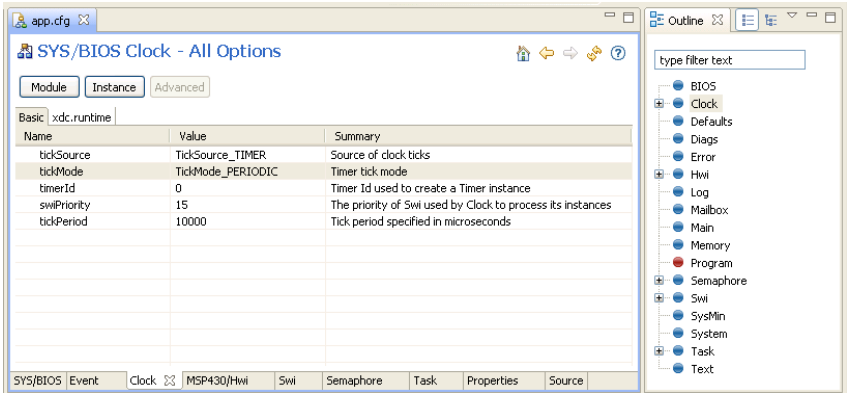

Figure 7. SYS/BIOS Clock setting for TickMode

3.2.2.2 Initialization

There are two capacitive touch clock instances, initSensorBaseline and measureSensor. The initSensorBaseline calls the appropriate function which performs a single measurement and loads this value into the baseline tracking algorithm. This is the 'starting' point for the baseline tracking and all successive measurements are compared to the baseline. As an alternative to performing a measurement to update the baseline, the baseline value can be preprogrammed in Flash and loaded at power up. This is done in the TI_CTS_Copy_Flash_to_Ram() function which is not part of the SYS/BIOS setup but found in the main application.

Figure 8 shows that for this example the initSensorBaseline is called one time, 5 clock ticks (50ms) after the initial start. Argument passed is '0' to indicate which sensor is to be initialized. The argument passed corresponds to the index in the array sConfig.

Copyright © 2012, Texas Instruments Incorporated

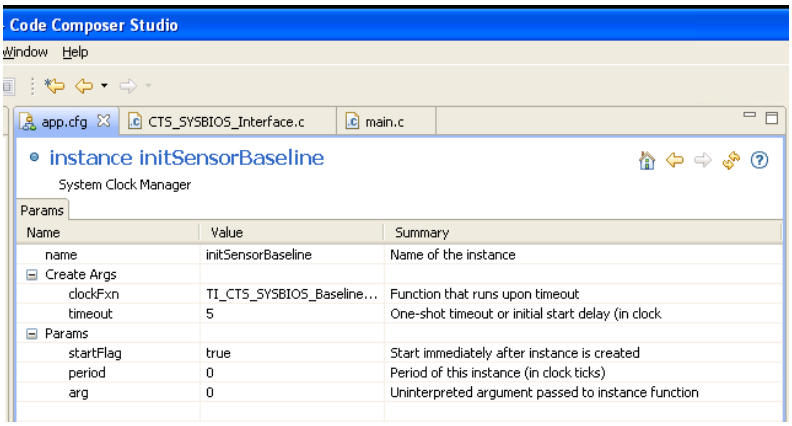

Figure 8. Clock Instance of initSensorBaseline

3.2.2.3 Scan Rate

The measureSensor clock instance settings will be influenced by the measurement time and the number of elements. In this application the measurement time is 2ms and the number of elements is 5. Additionally if any element is touched it is re-measured 2 times to validate the touch or 'debounce' the key. This could lead to a total measurement time of about 30ms (5*2ms + 5*2*2ms). This provides a maximum sampling rate of about 30Hz. Most applications require 10-20Hz, so this does leave a fair amount of margin. This margin can decrease as thick laminates are added and the measurement time needs to be increased for sensitivity.

The measureSensor clock instance has a number of implications. First, this clock instance determines the scan rate of the sensor. Second, the measureSensor instance calls the functions which sets up and uses HW peripherals to perform the capacitive touch measurement. It is important to ensure that there are no conflicts with peripheral allocation. An example of a conflict would be a clock instance of measureSensor occurring before the system is finished measuring the capacitive touch elements from a previous initSensorBaseline instance.

Table 2 shows the setup for the measurement instance. Like the baseline update instance the argument passed represents the index in the sensor array sConfig. In this application there is only one sensor and the index is 0. The startup delay is 11 clock ticks (110ms) and then the instance occurs periodically every 10 clock ticks (100ms). The initSensorBaseline instance has a delay of 5 clock ticks (50ms) while the measureSensor instance has a delay of 11 ticks. The 6 tick delay (60ms) provides sufficient time to complete the necessary measurements associated with initSensorBaseline.

Table 2. Clock Instance of measureSensor

Name	Value	Summary
name	measureSensor0	Name of the Instance
Create Args		
clockFxn	TI_CTS_SYSBIOS_Measure_Sensor	Function that runs upon timeout
timeout	11	One-shot timeout or initial start delay (in clock ticks)
Params		
startFlag	True	Start Immediately after instance is created
period	10	Period of this instance (in clock ticks)
Arg	0	Uninterpreted argument passed to instance function

3.3 MSP430F5529 Experimenter's Board Application

The final portion of this application is the visual representation shown in Figure 9: a touch indicator and a display showing the current baseline value and how the current measurement has deviated from the baseline.

Copyright © 2012, Texas Instruments Incorporated

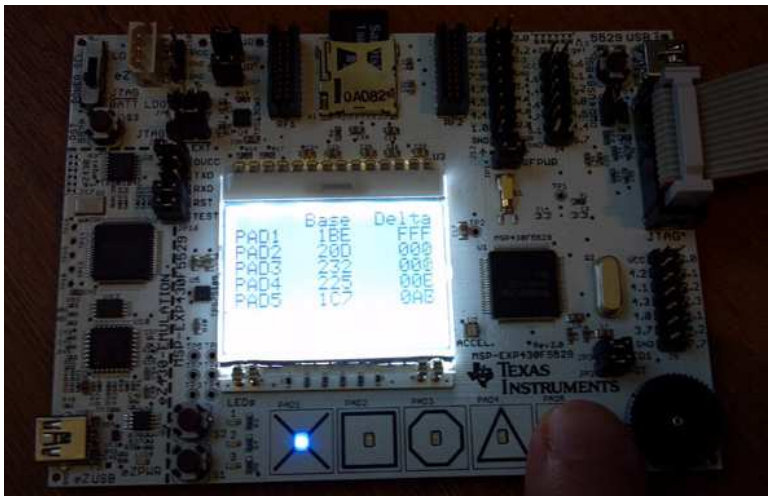

Figure 9. Visual Representation of Touch

3.3.1 LED and Display Update Clock Instance

Another clock instance is added to update the LEDs as well as the display. This clock instance is offset from the measurement and given the same period of 100ms; see Table 3.

Table 3. Clock Instance of Display Update

Name	Value	Summary
name	ledUpdate	Name of the Instance
Create Args		
clockFxn	F5529_LED_Display_Update	Function that runs upon timeout
timeout	16	One-shot timeout or initial start delay (in clock ticks)
Params		
startFlag	True	Start Immediately after instance is created
period	10	Period of this instance (in clock ticks)
Arg	Null	Uninterrupted argument passed to instance function

The function F5529_LED_Display_Update (found in main.c) pulls the baseline and measurement data directly from the element configuration structure (eConfig[i]). This information is used to calculate the difference between the baseline and the measurement and displayed as the delta in two's complement.

The LEDs are updated based upon the detail field found in register two of the sensor configuration.

Copyright © 2012, Texas Instruments Incorporated

3.3.2 Sensor Register 2: Detail and Detection

The LED_Update function found in led.c uses the detail information in sensor register 2. The least significant bits in the detail field represent which key in the sensor has been pressed while the detect field simply indicates that at least one of the elements in the sensor has registered a detection.

Table 4. Sensor Register 2, sRegister2

Field	Description
Detail	Bit representation of which button in the sensor (group of buttons) is being touched. Location along slider or wheel, (0-points)
Detect	Indicates that the sensor has detected a valid touch

4 Summary

Integration of capacitive touch into an existing solution can be made easier with the help of SYS/BIOS. The capacitive touch example code is specifically written to interface with SYS/BIOS so that SYS/BIOS can be used to control the sensor sampling rate. While the code is written specifically for the MSP430F5529 experimenter's board, the code is reusable and can be ported to other 5xx/6xx MSP430 product families and board configurations.

5 References

1. *TI SYS/BIOS v6.33 Real-time Operating System User's Guide* (SPRUEX3)
2. *Capacitive Touch Library* (SLAA490)
3. *MSP430x5xx and MSP430x6xx Family User's Guide* (SLAU208)
4. *MSP430F551x, MSP430F552x Mixed Signal Microcontroller Data Manual* (SLAS590)
5. *MSP-EXP430F5529 Experimenter Board User's Guide* (SLAU330)

Copyright © 2012, Texas Instruments Incorporated

Appendix A Interface Setup Instructions Instructions

Instructions on how to create a SYS/BIOS project can be found in the *Creating a SYS/BIOS Project* section in the *TI SYS/BIOS v6.33 Real-time Operating System User's Guide* (SPRUEX3). The folder CTS (and all of its contents) need to be copied into the workspace folder. The files found in this folder are listed in Table 5.

Table 5. Capacitive Touch SYS/BIOS File List

File Name	Description
structure.c/.h	Configuration files describing elements and sensors
CTS_SYSBIOS_HAL.c/.h	Hardware abstraction layer specific to single implementation with the MSP430F552x
CTS_SYSBIOS_Layer.c/.h	Post processing of measured data. Representation of buttons, sliders, and wheels. Update of baseline tracking algorithm.
CTS_SYSBIOS_Interface.c/.h	Function definitions to be pointed to by Hwis, Swis, and Tasks in SYS/BIOS.

The project settings need to be updated to enable support for GCC extensions. The include settings should also include the CTS folder. Figure 10 and Figure 11 shows the properties dialog box and the appropriate additions for CCSv5.1.

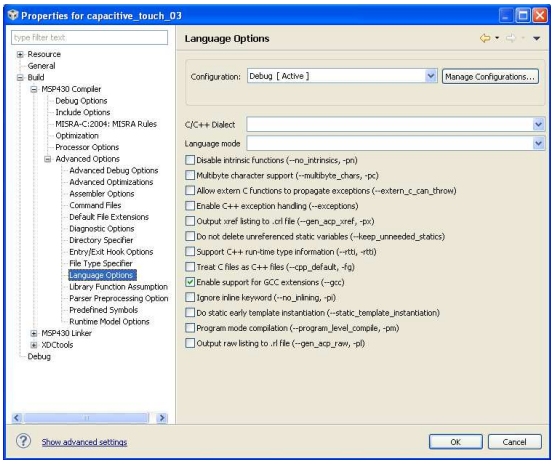

Figure 10. CCSv5.1; Advanced Options; Language Options

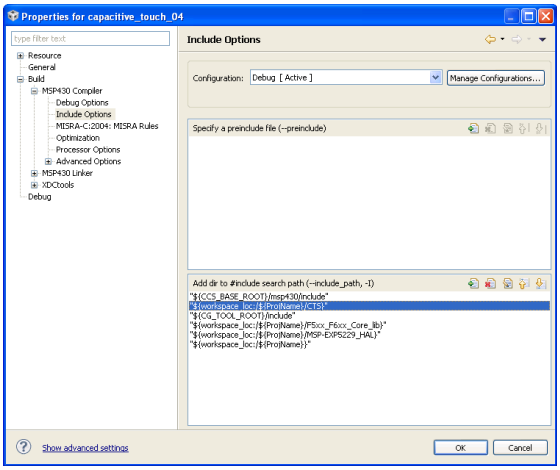

Figure 11. CCSv5.1, Include Options

Copyright © 2012, Texas Instruments Incorporated

Once the files have been included and the appropriate project settings are made, then the SYS/BIOS configuration tool can be used to setup the capacitive touch functions.

The other Swis in the capacitive touch solution are deglitchSensor, deglitchElement, and measureElement. The deglitchSensor helps manage the transition from initial measurement to debouncing keys that may have detected a possible touch.

Table 6. Swi Instance, deglitchSensor

Name	Value	Summary
name	deglitchSensor	Name of the Instance
Create Args		
fxn	TI_CTS_SYSBIOS_Deglitch_Sensor	Swi Function
Params		
arg0	0	Swi function argument 0
arg1	0	Swi function argument 1
priority	13	Swi priority
trigger	0	Initial Swi trigger value

The deglitchElement Swi calls the function which determines if the threshold criterion has been met by all of the repeated measurements of an element and sets or clears the appropriate bits in the sensor register to indicate touch or no-touch.

Table 7. Swi instance, setupCTSdbnce

Name	Value	Summary
name	deglichElement	Name of the Instance
Create Args		
fxn	TI_CTS_SYSBIOS_Deglitch_Element	Swi Function
Params		
arg0	0	Swi function argument 0
arg1	0	Swi function argument 1
priority	12	Swi priority
trigger	0	Initial Swi trigger value

The setupmeasureSensor Swi calls the function that sets up the measurement cycle for both a regular measurement and a baseline initialization.

Table 8. Swi instance, measureElement

Name	Value	Summary
name	measureElement	Name of the Instance
Create Args		
fxn	TI_CTS_SYSBIOS_Measure_Element	Swi Function
Params		
arg0	0	Swi function argument 0
arg1	0	Swi function argument 1
priority	14	Swi priority
trigger	0	Initial Swi trigger value

There is one Hwi for the capacitive touch sensing. This Hwi is the WDTA peripheral. The parameters are all left at the default setting. In the MSP430F5529 the interrupt number for the watch dog timer is 57. For more information, see the *MSP430F551x, MSP430F552x Mixed Signal Microcontroller Data Manual* (SLAS590).

Copyright © 2012, Texas Instruments Incorporated

Table 9. Hwi instance, wdta

Name	Value	Summary
name	wdta	Name of the Instance
Create Args		
intNum	57	Interrupt number
hwiFxn	wdta_interrupt	Pointer to ISR function
Params		(default)

There is one task and one semaphore for the capacitive touch sensing. The semaphore is ctsSem and the task is ctsTask. The task is posted once all the measurements have been completed. This serves as a transition point between the critical and non-critical portions of the capacitive touch sensing. The task performs post measurement algorithms for baseline tracking as well as representation algorithms for wheels, sliders, and groups of buttons.

Table 10. Semaphore, ctsSem

Name	Value	Summary
name	ctsSem	Name of the instance
Create Args		
count	0	Initial semaphore count
Params		
event	null	Event instance to use it non-NULL
eventId	1	eventide if using events
mode	Mode_BINARY	Semaphore mode

Table 11. Task, postProcessTask

Name	Value	Summary
name	postProcessTask	Name of the Instance
Create Args		
fxn	0	Task function
Params		
arg0	0	Task function argument. Default is 0.
arg1	0	Task function argument. Default is 0.
priority	1	Task priority (0 to numPriorities-1 or -1). Default is 1.
stack	null	Task stack pointer. Default is null.
stackSize	512	Task stack size in MAUs
stackSection	.bss:taskStackSection	Mem section used for statically created task stacks
stackHeap	null	Mem heap used for dynamically created task stack
Env	null	Environment data struct
vitalTaskFlag	true	Exit system immediately when the last task with this flag is set to TRUE has terminated

Copyright © 2012, Texas Instruments Incorporated

Appendix B Register Definitions

B.1 Element Registers

There are two sets of data associated with each element. One set is found in Flash and is used as the starting point at power up. The other set is found in RAM and is used during run time. The TI_CTS_Copy_Flash_to_RAM () function loads the Flash values into the corresponding RAM locations. Table 12 shows the different fields found in the element configuration.

Table 12. Element Configuration

Field	Description
portInfo	Pointer to the port definition, in this case the definition is the comparator input mux setting
eRegister0	Contains bit fields defining settings for the drift compensation, filtering and measurement time
Threshold	The value the measured data must exceed (relative to the baseline) inorder to declare a possible touch.
maxResponse	The maximum expected response from a touch
Measure	The current measurement value
Baseline	The current baseline value

The port information, portInfo, points to port configuration for the element. This port information is used to configure the COMPB multiplexer appropriately to measure the element.

eRegister0 contains four fields: measureTime, driftComp, filter, and baseUpdate. The measureTime defines the WDTA interval and is a function of the source selected and the divider. The filter and driftComp fields define how the measured data is used to update the baseline. The filter simply represents different filters that are applied to the measured data and the driftComp determines the weight or significance of that filtered data when updating the baseline.

Table 13. Filter Settings

Filter	ValueF	Description
EREGISTER0_FILTER_0	0x0000	The filtered response equals the measurement
EREGISTER0_FILTER_1	0x0400	The filtered response is an average of the current and previous measurements
EREGISTER0_FILTER_2	0x0800	The filtered response is an average of the current and previous 4 measurements
EREGISTER0_FILTER_3	0x0C00	The current measurement is limited so that the difference between the baseline and measured value does not exceed threshold/2 in magnitude. The filtered response is an average of the current (limited) measurement and previous 4 measurements.

Once the measured data has been filtered, a weighted average of the current baseline and the filtered response are used to update the baseline.

Table 14. driftComp Settings

Filter	Value	Description
EREGISTER0_DRIFT_0	0x0000	No drift compensation. The new baseline equals the filtered response.
EREGISTER0_DRIFT_1	0x0080	The new baseline is an average of the current baseline and the filtered response.
EREGISTER0_DRIFT_2	0x0100	The new baseline is a weighted average: 3/4 current baseline, 1/4 filtered response
EREGISTER0_DRIFT_3	0x0180	The new baseline is a weighted average: 7/8 current baseline, 1/8 filtered response
EREGISTER0_DRIFT_4	0x0200	If the filtered response is greater than the current baseline then the weighted average is 1/8 current baseline, 7/8 filtered response. Else the weighted average is 7/8 current baseline, 1/8 filtered response.

Copyright © 2012, Texas Instruments Incorporated

Table 14. driftComp Settings (continued)

Filter	Value	Description
EREGISTER0_DRIFT_5	0x0280	If the filtered response is greater than the current baseline then the weighted average is 1/4 current baseline, 3/4 filtered response. Else the weighted average is 7/8 current baseline, 1/8 filtered response.

For the relaxation oscillator implementation used in this application an increase in counts represents a decrease in capacitance. The 'direction of interest' or direction associated with an increase in capacitance (and possible touch) is a decrease in counts. If the filtered response is greater than the current baseline then this represents a decrease in capacitance. In the case of settings 4 and 5 the filtered response is given a larger weighting allowing the tracking algorithm to more quickly 'track' or follow the decrease in capacitance. In the event the filtered response is less than the baseline then this represents an increase in capacitance. This increase may be the result of environmental changes or an approaching finger. In this case the heavier weighting is applied to the current baseline. This 'slows' down the update to help differentiate between slow moving objects and environmental changes.

The update rate to the baseline can also be reduced by adjusting the updateBase field in eRegister0. This value represents the number of measurements in between baseline updates (when no threshold detection has occurred). Setting updateBase to 0, the baseline will update with each measurement (that does not meet the threshold criteria). The maximum value of updateBase is 7; therefore, the minimum baseline update rate is 1/8 the scan rate.

Table 15. baseUpdate Settings

Filter	Value	Description
EREGISTER0_BASEUP_0	0x0000	Baseline update rate equals element scan rate
EREGISTER0_ BASEUP _1	0x2000	Baseline update rate equals (element scan rate) /2
EREGISTER0_ BASEUP _2	0x4000	Baseline update rate equals (element scan rate) /3
EREGISTER0_ BASEUP _3	0x6000	Baseline update rate equals (element scan rate) /4
EREGISTER0_ BASEUP _5	0xA000	Baseline update rate equals (element scan rate) /6
EREGISTER0_ BASEUP _7	0xE000	Baseline update rate equals (element scan rate) /8

B.2 Sensor Registers

There are three sensor registers created to describe each sensor. Sensor registers 0 and 1 define the sensor configuration while register 2 contains the result information.

Table 16. Sensor Register 0, sRegister0

Field	Description
elementLocation	This identifies at the bit level which elements are part of the sensor.
numberOfElements	Indicates the number of elements within the sensor
Sensitivity	For wheels and sliders, this value determines how 'centered' a touch needs to be on the wheel or slider to be considered a valid touch.

Table 17. Sensor Register 1, sRegister1

Field	Description
Points	When '0', this indicates a button or group of buttons; any non-zero value will be the number of points represented along a slider or wheel.
Representation	When points = 0, this value determines how a group of buttons is represented. When points is > 0; 0 = wheel, 1 = slider
Deglitch	Number of successful retries before a touch is declared

Copyright © 2012, Texas Instruments Incorporated

Table 18. Sensor Register 2, sRegister2

Field	Description
Detail	Bit representation of which button in sensor (group of buttons) is being touched Location along slider or wheel, (0-points)
Detect	Indicates that the sensor has detected a valid touch

Copyright © 2012, Texas Instruments Incorporated

Appendix C Modifying the Number of Elements

C.1 Modifying structure.c

eport

The eport array defines which inputs to the COMPB multiplexer are being used. The order is not significant but the relationship with the variable elementLocation found in the sensor definition (sRegister0) must be understood.

Figure 12. eport Array Relationship to Sensor Configuration

Adding and removing elements involves adding or removing values from the array eport. Once the modifications are made to the eport array the definition, TOTAL_NUMBER_OF_ELEMENTS, must be updated in structure.h.

```
// P6.0, CB0
PortInfo eport[TOTAL_NUMBER_OF_ELEMENTS] = {
{
              .inputBits = CBIMSEL_0,
},
// P6.1, CB1
{
              .inputBits = CBIMSEL_1,
},
// P6.2, CB2
{
              .inputBits = CBIMSEL_2,
},
// P6.3, CB3
{
              .inputBits = CBIMSEL_3,
},
// P6.4, CB4
{
              .inputBits = CBIMSEL_4,
}
};
```

compbTaxConfig

The compbTaxConfig,cbpdBits field needs to be updated to reflect the all of the inputs used by the comparator. These bits are directly transferred into the port disable register, CBCTL3, to disable the digital logic on those pins. BIT0 corresponds to CB0, BIT1 CB1, and so on.

```
CompbTaxInfo compbTaxConfig =
{
                          // These bits disable the digital IO on the corresponding CBx bits
                          //BIT0->CB0-> P6.0 on the F5529
                          //BIT4->CB4-> P6.4 on the F5529
                          // ...
                          // BIT8->CB8-> P7.0 on the F5529
                          // Please note that CB8,9,10, and 11 are not available on
all
                          // package types.
            .cbpdBits = (uint16_t)(BIT0+BIT1+BIT2+BIT3+BIT4),
            // CBOUT/TA1CLK is found on P1.6
            // CBOUT/TA1CLK can also be port mapped to port4 (P4.0-P4.7) but
            // this is beyond the scope of this application.
```


Copyright © 2012, Texas Instruments Incorporated

```
            .cboutTAxDirRegister = (uint8_t *)&P1DIR, // PxDIR
            .cboutTAxSelRegister = (uint8_t *)&P1SEL, // PxSEL
            .cboutTAxBits = BIT6, // P1.6
};
```

eConfigFlash

The eConfigFlash array contains the configuration data for each element. If an element is added or removed the appropriate indices should be added or removed. Note that eConfigFlash[0] corresponds to eport[0], eConfigFlash[1] to eport[1], and so on.

```
// P6.0, CB0
FlashElementConfig eConfigFlash[TOTAL_NUMBER_OF_ELEMENTS] = {
      {
   .eRegister0 = (EREGISTER0_BASEUP_7+EREGISTER0_FILTER_3+EREGISTER0_DRIFT_4+
                  SOURCE_ACLK+SOURCE_DIVIDE_64),
          .threshold = 100,
          .baseline = 350,
          .maxResponse = 255
},
// P6.1, CB1
{
   .eRegister0 = (EREGISTER0_BASEUP_7+EREGISTER0_FILTER_3+EREGISTER0_DRIFT_4+
                  SOURCE_ACLK+SOURCE_DIVIDE_64),
          .threshold = 100,
          .baseline = 395,
          .maxResponse = 290
},
// P6.2, CB2
{
   .eRegister0 = (EREGISTER0_BASEUP_7+EREGISTER0_FILTER_3+EREGISTER0_DRIFT_4+
                  SOURCE_ACLK+SOURCE_DIVIDE_64),
          .threshold = 100,
          .baseline = 415,
          .maxResponse = 320
},
// P6.3, CB3
{
   .eRegister0 = (EREGISTER0_BASEUP_7+EREGISTER0_FILTER_3+EREGISTER0_DRIFT_4+
                  SOURCE_ACLK+SOURCE_DIVIDE_64),
          .threshold = 100,
          .baseline = 405,
          .maxResponse = 315
},
// P6.4, CB4
{
   .eRegister0 = (EREGISTER0_BASEUP_7+EREGISTER0_FILTER_3+EREGISTER0_DRIFT_4+
                  SOURCE_ACLK+SOURCE_DIVIDE_64),
          .threshold = 110,
          .baseline = 350,
          .maxResponse = 275
}
};
```

sConfigFlash

The fields numberOfElements and elementLocations need to be updated to reflect if the new elements are used in the sensor. The example in Figure 13 shows a sensor comprised of four elements and the elements are connected to eport[0], eport[3], eport[4], and eport[5].

Copyright © 2012, Texas Instruments Incorporated

Figure 13. Example Sensor Configuration and Relationship to eport Array

NOTE: Having non-consecutive elements in a sensor is only allowed for groups of buttons and is not supported with the wheel and slider sensor implementations.

As mentioned at the beginning of this section, the relationship between the eport array and the elementLocation must be noted. Predefined values of eport are provided to help prevent errors when index of the eport array does not match the comparator input (ie eport[0] connected to CB5). The following code snippet shows the code associated with the example found in **Figure D-2 ??????**.

```
const FlashSensorConfig sConfigFlash[TOTAL_NUMBER_OF_SENSORS] = {
   {
     .sRegister0 = ((NUM_OF_ELEMENTS_4)+(EPORT5+EPORT4+EPORT3+EPORT0)),
     .sRegister1 = ((0)+REPRESENTATION_0+DEGLITCH_2)
   }
};
```

C.2 Modifying structure.h

The definition of TOTAL_NUMBER_OF_ELEMENTS needs to be updated to reflect the new total number of elements.

Copyright © 2012, Texas Instruments Incorporated

Appendix D Modifying the number of Sensors

The example in this application report is of a single sensor. Similar to modifying the elements, modifying the sensors requires editing the structure.c and structure.h files. In addition to the structure.c/.h files, clock instances must be added to address new sensors.

D.1 Modifying structure.c, sConfigFlash

The structure sConfigFlash is an array of configurations, where each index represents a sensor. To add a new sensor a new index must be added to sConfigFlash. In the following example code two sensors have been added, each made of one element.

```
const FlashSensorConfig sConfigFlash[TOTAL_NUMBER_OF_SENSORS] = {
{
     // Sensor0
     .sRegister0 = ((NUM_OF_ELEMENTS_4)+(EPORT5+EPORT4+EPORT3+EPORT0)),
     .sRegister1 = ((0) + REPRESENTATION_0 + DEGLITCH_2)
}
,{
     // Sensor1
     .sRegister0 = ((NUM_OF_ELEMENTS_1) + (EPORT1)),
     .sRegister1 = ((0) + REPRESENTATION_0 + DEGLITCH_2)
}
,{
     // Sensor2
     .sRegister0 = ((NUM_OF_ELEMENTS_1) + (EPORT2)),
     .sRegister1 = ((0) + REPRESENTATION_0 + DEGLITCH_2)
}
};
```

D.2 Modifying structure.h

The definition of TOTAL_NUMBER_OF_SENSORS needs to be updated to reflect the new total number of sensors.

D.3 Updating the Clock instances in SYS/BIOS

The instances for the additional sensors need to be added to the System Clock Manager. The clockFxn remains the same; however, the name, timeout, period, and arg variables need to be updated appropriately. The setting of the timeout and period must be made to prevent contention with the measurement of another sensor. Again it is important to remember the timing relationships so that capacitance measurements are not corrupted. The arg variable represents the index of the sensor array, sConfig. The name variable is left to the creativity of the user.

NOTE: In the case of the baseline initialization, the period is still 0 indicating a ‘one-shot’ instance.

Copyright © 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 which meet ISO/TS16949 requirements, mainly for automotive use. Components which have not been so designated are neither designed nor intended for automotive use; and TI will not be responsible for any failure of such components to meet such requirements.

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 © 2012, Texas Instruments Incorporated