Introduction

Texas Instruments ultra-low power MSP430 is a fantastic MCU to implement Capacitive Touch techniques. It’s not just for BSW (Buttons, Sliders and Wheels) but can be used for other capacitive sensing applications, such as: Proximity, Grip, and Immersion.

The latest MSP430 devices provide direct capacitive touch I/O pins, which lowers the cost and power when implementing capacitive touch/sensing, while making them easier than ever to use in a system.

Adding to the ease-of-use is the CAPT software library for buttons, sliders and wheels – in fact, any type of capacitive sensing electrode. TI also provides GUI tools for tuning your system, estimating the Power requirements, and configuring your device.

Enough of the marketing, we begin this chapter with the What / Why / How of Capacitive Sensing. Armed with a little background on the subject, we’ll discuss the basic steps to implementing Cap Touch software. Finally, we provide a number of hands-on exercises to experiment and learn how to use the device, library, and tools.

Chapter Outline

- What is Capacitive Touch
- Why Use TI’s MSP430
- How Does Cap Touch Work?
- Capacitive Sensing Details
- Implementing Cap Touch
- Additional Topics
- Lab Exercise

http://www.ti.com/touch
Workshop Agenda

This learning module is part of a 1-day MSP430 Hands-On Workshop. There will be times in the lab exercises where we refer to other chapters for extended learning about something being implemented. For example, we use Hardware Interrupts and Interrupt Service Routines (ISR) in the lab exercise for this chapter, but we leave the gory details of how they work to the chapter on Interrupts.

This said, the chapter still works well as a stand-alone topic, if all you care about learning is the information on Capacitive Touch & Sensing.

http://www.ti.com/touch
Prerequisites, Tools and Objectives

This training is aimed at microcontroller programmers with (at least) a basic understanding of using the C Language.

No Electrical Engineering degree is required, though, a basic knowledge of the physics behind Cap Touch can help when implementing it – even when ‘just doing’ the software. To this end, we spend a few pages providing a light background on the technology.

We also wanted to list the hardware and software requirements for this chapter. Links to these items are provided at the beginning of the Lab10 exercise, as well as the workshop’s installation guide.

At the end of this chapter, we hope you will be able to …

Objectives

The user should be able to…

- Describe how capacitive touch sensing interfaces work
- List three MSP430 features that make it ideal for Capacitive Touch applications
- Define the following terms:
  - RO, fRO
  - Gate Time
  - Threshold
- List the steps needed to implement a capacitive sensing system using the TI Cap Touch Sensing library
- Given a capacitive touch sensor (hardware), write the software to enable the sensor using the MSP430
Chapter Topics

Capacitive Touch.......................................................................................................................... 10-1
   Workshop Agenda....................................................................................................................... 10-2
   Prerequisites, Tools and Objectives ........................................................................................ 10-3
   What is Capacitive Touch ........................................................................................................ 10-5
   Why Use TI’s MSP430 ............................................................................................................ 10-8
   How Does Cap Touch Work? ................................................................................................. 10-10
       A Short Physics Lesson ........................................................................................................ 10-10
       Applied Physics ............................................................................................................... 10-14
   Capacitive Sensing Details .................................................................................................... 10-16
       Capacitive Sensing I/O ....................................................................................................... 10-16
       Gate Time ......................................................................................................................... 10-17
       Power Requirements (and Scan Rate) ............................................................................... 10-18
       Measurement Methods (RO vs fRO) ................................................................................ 10-20
       Sensor Threshold .............................................................................................................. 10-21
   Implementing Cap Touch ....................................................................................................... 10-23
       2. Write Code ................................................................................................................. 10-28
       3. Tuning Sensors (Threshold, Gate time) ...................................................................... 10-33
       Summary: TI’s Cap Touch Library (TI_CAPT) ............................................................... 10-34
   Additional Topics ..................................................................................................................... 10-35
       Hardware Sensor Design (For Reference Only) ............................................................ 10-35
       For More Info .................................................................................................................... 10-36
   Lab Exercise ........................................................................................................................... 10-38
What is Capacitive Touch

From Wikipedia:

In electrical engineering, capacitive sensing is a technology, based on capacitive coupling, that takes human body capacitance as input. Capacitive sensors detect anything that is conductive or has a dielectric different from that of air.

Many types of sensors use capacitive sensing, including sensors to detect and measure proximity, position or displacement, humidity, fluid level, and acceleration. … Capacitive sensors can also replace mechanical buttons.

And, that is where we start. Replacing mechanical switches with Capacitive Touch/Sensing (we’ll abbreviate this with CapTouch) buttons is one of the largest use cases for CapTouch. As the following slide indicates, there are quite a few advantages to using this technology – not the least of which is lower cost.

Why Capacitive Touch Sensing?

Replace mechanical scroll wheel / buttons with Capacitive touch

Advantages
 Higher Reliability
  - No moving parts
  - Longer life
  - Better ESD & environmental protection
 Innovation
  - Flexible layout/design
  - not restricted to buttons
  - Design slimmer products
  - Controls hidden until lit
  - Easy to clean
 Lower Cost - Create directly on PCB (printed circuit board)

If you really miss that good ‘ol click of a mechanical switch, there’s an app for that, so to speak. The technology, called Haptics, can provide a responsive feedback like we’ve grown to appreciate from mechanical implementations. TI has a number of products that can help you to implement Haptics solutions (even a new MSP430 device that does CapTouch along with Haptics). To learn more about these solutions, visit the www.ti.com/touch webpage and click the Haptics tab.

As a final note, the above slide shows one of the more famous CapTouch mechanical replacements – and it may be the place where most of us really learned about Scroll Wheels. Of course, we’re alluding to the iPod scroll wheel. My first generation iPod had a true mechanical scroll wheel. Sure, it worked beautifully. But can you imagine implementing it on today’s small, thin devices? CapTouch has enabled us to miniaturize a number of uniquely helpful (and fun) products … while making them cheaper in the process.
What is Capacitive Touch

Buttons, Sliders, and Wheels (nicknamed BSW) are the most obvious use of CapTouch.

Capacitive Touch Sensors (BSW)

- Common sensors include buttons, sliders, or wheels (BSW)
- A button sensor requires a single element on the PCB
- A slider or wheel is comprised of multiple sense elements

But a whole new generation of CapTouch enabled products are finding innovative new ways to use the technology. Proximity detection in one of the leading ways this is being done.

Proximity Detection Sensors

TI Demo auto-wakes via proximity sensor...
In fact, TI’s CapTouch BoosterPack – which we use throughout the upcoming lab exercise, ships with a demo that stays asleep … until you wave your hand over the board.

### Cap Touch - Pre-Programmed Demo!

Wave your hand over the sensor to wake up a system!

Although, in most of our demos we encourage you to touch

It’s hard to summarize all the places where you can use CapTouch, but here’s a few of them. For example, medical – and other sanitary-minded applications – are making heavy use of proximity detection. Have you used one of those proximity detecting hand-sanitizer dispensers, yet?

#### Summary: Where to Use Capacitive Touch?

<table>
<thead>
<tr>
<th>White Goods &amp; Appliances</th>
<th>Buttons/Sliders/Wheel (BSW) Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>▪ Home Appliances &amp; White Goods</td>
</tr>
<tr>
<td></td>
<td>▪ Electronics: TV’s, Monitors, Blu-ray</td>
</tr>
<tr>
<td></td>
<td>▪ PC and Cellphones Accessories</td>
</tr>
<tr>
<td></td>
<td>▪ Smart Cards</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proximity Sensing Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Tablet, Handsets, Remotes</td>
</tr>
<tr>
<td>▪ “Always on” … wake up the processor</td>
</tr>
<tr>
<td>▪ Meet FCC SAR Compliance</td>
</tr>
<tr>
<td>▪ Sanitary applications</td>
</tr>
<tr>
<td>▪ Inventory Management</td>
</tr>
<tr>
<td>▪ Headsets, 3D glasses, Mice, &amp; Illuminated Keyboards</td>
</tr>
</tbody>
</table>

Hopefully, in the future we’ll be writing about your innovative new way to utilize the technology.
Why Use TI’s MSP430

I’m sure the marketing folks can do a better job of this than us application engineers, but convenience (ease-of-implementation), cost, and low-power requirements are many great reasons to use the MSP430 as your CapTouch controller.

### Capacitive Touch Optimized Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Flash</th>
<th>RAM</th>
<th>Capacitive Touch I/O</th>
<th>Touch Buttons Supported</th>
<th>ADC</th>
<th>Serial (UART, I2C, SPI)</th>
<th>Additional 'Touch' Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP430G2x02</td>
<td>1-8 KB</td>
<td>256 B</td>
<td>✓</td>
<td>≤ 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSP430G2x32</td>
<td>1-8 KB</td>
<td>256 B</td>
<td>✓</td>
<td>≤ 16</td>
<td>10-bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSP430G2x03</td>
<td>2-16 KB</td>
<td>512 B</td>
<td>✓</td>
<td>≤ 24</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSP430G2x33</td>
<td>2-16 KB</td>
<td>512 B</td>
<td>✓</td>
<td>≤ 24</td>
<td>10-bit</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>MSP430F5x11</td>
<td>8-32 KB</td>
<td>1-2 KB</td>
<td>✓</td>
<td>≤ 29</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSP430F5x12</td>
<td>8-32 KB</td>
<td>1-2 KB</td>
<td>✓</td>
<td>≤ 29</td>
<td>10-bit</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>MSP430FR58XX</td>
<td>FRAM</td>
<td></td>
<td>✓</td>
<td>≤ 32</td>
<td>12-bit</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>MSP430FR59XX</td>
<td>RAM</td>
<td></td>
<td>✓</td>
<td>≤ 32</td>
<td>12-bit</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

- MSP430s feature unique peripherals for lower power, higher precision or lower cost Capacitive Sensing implementations
  - MSP430s with Capacitive Touch I/Os provides glue less sensor interface
  - In fact, any MSP430 device with a comparator supports capacitive touch sensing
- More devices with Touch Sense I/O are on the way…

The MSP430 is widely known as the most power-efficient microcontroller. But we don’t stop there; by adding CapTouch sensing GPIO pins to our controllers, we take them to another level.

### Ultra Low Power with Capacitive Touch I/O

- Very little CPU effort required
- Processing Time (Active)
- Lots of sleep time
- Only timers need to be running

### Ultra Low Power with Capacitive Touch I/O Table

<table>
<thead>
<tr>
<th>Mode</th>
<th>Current *</th>
<th>MSP430 State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gate Time</td>
<td>70uA</td>
<td>LPM0 + Pin Osc</td>
</tr>
<tr>
<td>Processing Time</td>
<td>180uA</td>
<td>Active</td>
</tr>
<tr>
<td>Sleep Time</td>
<td>0.5uA</td>
<td>LPM3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1/Scan Rate</th>
<th>Average Current *</th>
</tr>
</thead>
<tbody>
<tr>
<td>50ms (20Hz)</td>
<td>3.7 uA</td>
</tr>
<tr>
<td>100ms (10Hz)</td>
<td>2 uA</td>
</tr>
<tr>
<td>500ms (2Hz)</td>
<td>0.7 uA</td>
</tr>
</tbody>
</table>

* MSP430 Value Line using Capacitive Touch I/O running at 1MHz, 1.8V, Gate Time = 1ms
Great devices notwithstanding, if you can’t figure out how to use them, what good are they? That’s why TI provides a great number of demos, libraries, tools and application notes.

### Lot’s of TI Touch Design Collateral

#### Hardware & Demo’s
- Getting Started with the MSP430 LaunchPad Workshop
- MSP430 Capacitive Button, Sliders and Wheels
- MSP430F2xx/6xx Family User’s Guide (Timer D and TEC sections)
- Wireless Remote Ctrl With Capacitive Touch Pad Using MSP430F2xx
- Download MSP430™ Capacitive Touch Sense Software Library
- MSP430 Low Cost PinOsc Capacitive Touch Overview
- MSP430 Low Cost PinOsc Capacitive Touch Keypad
- Capacitive Touch Sensing, SYS/BIOS
- MSP430G2xx3 Cap Touch Matrix Remote

#### Capacitive Sensing Software Library

#### Application Notes
- [Getting Started with the MSP430 LaunchPad Workshop](#)
- [MSP430 Capacitive Button, Sliders and Wheels](#)
- [MSP430F2xx/6xx Family User’s Guide (Timer D and TEC sections)](#)
- [Wireless Remote Ctrl With Capacitive Touch Pad Using MSP430F2xx](#)
- [Download MSP430™ Capacitive Touch Sense Software Library](#)
- [MSP430 Low Cost PinOsc Capacitive Touch Overview](#)
- [MSP430 Low Cost PinOsc Capacitive Touch Keypad](#)
- [Capacitive Touch Sensing, SYS/BIOS](#)
- [MSP430G2xx3 Cap Touch Matrix Remote](#)

#### Graphical Touch Tools

To summarize the benefits…

### Summary of MSP430 Touch Benefits

<table>
<thead>
<tr>
<th>Touch optimized peripherals</th>
<th>Capacitive Touch I/O module requires no external parts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 nsec Timer D resolution offers faster response time, higher</td>
</tr>
<tr>
<td></td>
<td>sensitivity and support for more buttons</td>
</tr>
<tr>
<td>Ultra Low Power</td>
<td>1.8V operation =&gt; Battery Powered Applications</td>
</tr>
<tr>
<td></td>
<td>&lt;1uA Average current</td>
</tr>
<tr>
<td>Flexible Design &amp; Low Cost</td>
<td>Capacitive touch I/O available on <a href="#">MSP430 Valueline</a></td>
</tr>
<tr>
<td></td>
<td>&lt;2KB of Flash, 14 bytes of RAM</td>
</tr>
<tr>
<td></td>
<td>Sensors can be integrated into PCB</td>
</tr>
<tr>
<td></td>
<td>Available on all MSP430 with wide range of peripherals</td>
</tr>
<tr>
<td>Design collateral</td>
<td>Collateral for proximity/grip detection, matrix keypad etc.</td>
</tr>
<tr>
<td></td>
<td>Greater than <a href="#">10 cm robust proximity detection</a></td>
</tr>
<tr>
<td></td>
<td>1uA capacitive grip detection technology</td>
</tr>
<tr>
<td></td>
<td>Application note for SYS/BIOS integration</td>
</tr>
<tr>
<td>Touch Sense Software</td>
<td><a href="#">Free, Open source</a> software to develop Button, Sliders,</td>
</tr>
<tr>
<td></td>
<td>Wheels, Proximity. API abstractions for max flexibility</td>
</tr>
<tr>
<td></td>
<td>Baseline tracking improves environmental robustness</td>
</tr>
</tbody>
</table>
How Does Cap Touch Work?

Hopefully we won’t scare any of you away with a short lesson on the physics behind CapTouch. Our goal isn’t to make you an expert, but rather to provide a little background to those of us who are not Electrical Engineers (EE). Knowing a little bit about how CapTouch works can make it easier to implement our system/software designs … and make it seem a little less intimidating, in the process.

A Short Physics Lesson

We start with VR … and we don’t mean virtual reality

One of the first rules that an EE learns is: V = IR. That is, Voltage is related to Current and Resistance. See the graphic below for the definitions of Voltage and Resistance.

Our circuit with a battery and a light-bulb is a good example of a simple electrical circuit. The graphic also contains the schematic diagram for this simple circuit. We use schematic diagrams since their notation is much easier to draw than light-bulbs and batteries.
Add in a little tender loving C

Circuits get a little more complex when we add in an element called Capacitance. Of course, being called Capacitive Touch, this is something we really cannot avoid.

![Capacitance Diagram]

- Capacitor is a passive component that stores energy in the form of an electrostatic field
- Field is created by two plates, separated by non-conductive material (called dielectric)
- Current flows when voltage is applied to ‘empty’ capacitor; this ‘charges’ it
- When charged, a capacitor acts like an open circuit (no current flows thru it)

- Take voltage source away, capacitor holds charge like a rechargeable battery
- Some MSP430 development kits use a big capacitor ('Super Cap') as a power source

- Create a circuit without the battery voltage source and the capacitor discharges
- Current will flow from C while enough charge remains in it

 Hopefully this concept of storing and releasing energy isn’t too difficult. It’s actually the crux of what makes our CapTouch systems work.

The key to making a good capacitor is to provide the greatest amount of surface area between the two very-close ‘plates’; and, using a good dielectric (non-conductive substance) between them. The better the dielectric, the better the capacitor – plain and simple.

We rate dielectrics by giving the dielectric substance a value. We call this the substance’s dielectric constant. (For example, air is “1” – good for capacitance; water is “80” – not so good.)

By the way, Super Caps are super cool. With today’s advances in producing capacitors, we can get powerful capacitors (that store quite a bit of energy) into a relatively small space. And for many applications they are a lot more convenient than using batteries, such as in development boards. (Keep an eye out for this in an upcoming new TI MSP430 Launchpad.)
How long does it take to charge up a capacitor?

Well, we’re glad you asked. This is an important aspect to our CapTouch technology.

Bottom line, the time depends upon the quality of the capacitor; or better put, how much energy the capacitor can hold. So, the amount of time varies depending upon the rating of the capacitor. We document the time it takes, though, by defining a value that represents how long it takes to charge the capacitor up to 63% of its capacity; this value is called the capacitor’s **Time Constant** – which is abbreviated using the Greek letter \( \tau \) (tau).

But, what’s really significant is that the Time Constant is … well, constant. So much so, that many designers use it to establish a time base; for example, as when creating a table clock.

---

**RC Time Constant – Relaxation Oscillator**

<table>
<thead>
<tr>
<th>Consistent Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>• For given values of R and C: Charging time is very precise and is described by constant ( \tau )</td>
</tr>
<tr>
<td>• Discharge time is also precise and follows a similar curve</td>
</tr>
<tr>
<td>• These characteristics have often been used to establish time-bases, such as a clock</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relaxation Oscillator (RO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Create oscillation by continual charge/discharge of C</td>
</tr>
<tr>
<td>• Tell time by counting # of oscillations</td>
</tr>
</tbody>
</table>

---

What if we were to continually charge, and then discharge an RC circuit. This is easily done in a microcontroller:

- We turn on a GPIO pin, which starts filling a capacitor.
- We could use an analog comparator (we have them on our MSP430) to tell us when the capacitor is (close to) full.
- The comparator triggers an interrupt where we turn off the GPIO pin and let the capacitor discharge
- When the capacitor is (almost) empty, the comparator interrupts the CPU again … and on and on

This constant oscillation of charging/discharging is called a Relaxation Oscillator (RO for short). All we really need to do is count the number of up/down cycles and we can tell time.

By the way, our latest MSP430 devices don’t need to use anything as prosaic as a comparator and interrupt to implement a RO, as we’ll see in a couple pages. We are now building many of our GPIO pins to generate a very power-efficient oscillator (RO) on command; no extra pins, comparators, interrupts or power are required.
Times Are Changing

As you might already suspect, if we change the value of C (capacitance) in our circuit, we change the frequency of our oscillator. More C … begets a slower oscillator.

<table>
<thead>
<tr>
<th>Change C … Changes the Time Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>✷ Changing the capacitance affects the oscillations</td>
</tr>
<tr>
<td>✷ A larger value for C means it takes longer to charge/discharge</td>
</tr>
</tbody>
</table>

![Graph showing oscillations with lower and higher capacitance]

Hopefully you will agree that this isn’t really that difficult to figure out. A bigger value for C means that it takes longer to charge the capacitor. Longer to charge, longer to discharge, means that one complete oscillation just takes longer. Hence, the oscillator slows down.

Hmm, what if we could find a way to force C to change?

If we knew how to count oscillations, and could change the value of C, then couldn’t we really make something cool … that’s where we’re going next.
### Applied Physics

OK, so we actually gave away the whole idea for this in the last section.

Applying our physics lesson to CapTouch, we can effectively build an RC circuit right into a printed circuit board (PCB). As seen below, we can build our “button” out of metal traces (or in this case, we could call them pads) in the circuit board; these traces almost touch, but don’t quite.

Can see the schematic symbols drawn over the top of the diagram below? Where the traces are “almost touching” these metal pads effectively create a capacitor.

- Cap Touch elements are often created directly in the PCB (printed circuit board)
- They have an inherent capacitance & resistance – which creates a specific oscillation frequency when charged/discharged
- When a conductive element (e.g Finger) is present, it affects the capacitance of the system (“free space coupling”)
- How is C affected? C is directly proportional to the dielectric value
  - Dielectric is the 3D space between the conductors – not only on the PCB, but the air around it
  - While air has a value ~1, a finger has a much greater dielectric constant (>1)

But here’s the cool thing, electrical circuits are not 2D, they are 3D. (Well, at least this is ‘cool’ if you’re not at university taking a Fields & Waves course.)

What this means is that the dielectric isn’t just the stuff between the two metal traces on the board, it’s actually everything in the 3D space around the metal traces. Most likely, the ‘dielectric’ in 3D space is just air – which, as you may remember is a good dielectric, with dielectric constant of 1.

So, how might we change the value of C in our button’s circuit? How about introducing another, lower quality, dielectric near our circuit, this would replace some of the air (which acts as a good dielectric). In fact, human tissue has a lot of water in it, so that might work well in changing the total dielectric value in the circuit – hence, it should change the value of C.

And there you go, if we change C, we change the frequency of oscillation. In in the next section we cover how the oscillations can be counted, which means we’ll finally have a way to detect when a finger ‘presses” a CapTouch “button”.

A Few Additional Notes on our Application of Physics

• With a CapTouch button, how can it sense if we’re pushing the button really hard? Nothing is moving, so how will it know?
   It all comes down to replacing more air with finger. Take a look at this diagram:

   ![Diagram showing how pushing harder covers more of the button](image)

   See how pushing harder covers more of the button. Therefore, more “air” dielectric is displaced ... and yes, we can detect these small changes in the resulting capacitance.

• If I push hard on the board, won’t I end up connecting the metal traces that make up my button?
   Actually ... no, it won’t. The key here is that we need to put an insulating overlay over the metal, otherwise you would be right and we might short-circuit the button. For best performance, the key is to find a very thin, insulating overlay.

• Finally, let’s (loosely) define a few terms. Throughout this document, and most others on the subject, a few terms are used interchangeably. We just want to speak to that here.
   It boils down to this, what do we call the metal traces that make up our “button”?

   Following are some of the terms you will see used:
   - **Electrode**: this is actually a pretty decent term. Wikipedia defines this as:
     
     “An electrode is an electrical conductor used to make contact with a nonmetallic part of a circuit”

     So this is term describes quite well what we’re doing with CapTouch.
   - **Touchpad**: is another widely used term. Wiki loosely defines touchpad as:
     
     “A tactile sensor”

     Most of us probably understand this term because it’s a ‘pad’ on the PCB that we ‘touch’.
   - **Sensor**: Again from Wiki:
     
     “A sensor (also called detector) is a converter that measures a physical quantity and converts it into a signal which can be read by an observer”

     Again, this works pretty well. This is just what we’re trying to do. With regards to the TI CapTouch library, though, we really use this term as an abstract object. We define Sensor to mean: Button, Slider, Wheel, and Proximity. For example, a Slider could actually be made up of many electrodes that we end up swiping across.
   - **Element**: If we have a sensor that could be made up of many electrodes – what does the TI library call the individual electrodes? We call them Elements.

   Later in the chapter, you’ll see that the TI Library allows us to define the Elements (i.e. electrodes) for our applications. Additionally, we define Sensors that are made up from 1 or more Elements.

   The Library also provides ‘abstracted’ functions which provide straightforward answers to whether a Button sensor is being touched ... or maybe, where a finger is positioned around the Elements that make up a Wheel sensor.
Capacitive Sensing Details

**Capacitive Sensing I/O**

As stated earlier in the chapter, many MSP430 devices implement a RO feature as part of the GPIO on the device. Think of it as having an RO peripheral built right into the pin.

Using this feature, these devices can gluelessly implement CapTouch sensing.

**Specialized GPIO Pins**

- Many MSP430 devices have I/O pins that contain a PinOsc feature.
- PinOsc can implement a pin oscillator (i.e. RO on a pin) without any additional hardware.
- Devices without this feature may require an external resistor, extra interrupt, and/or a comparator.

On the MSP430 Value Line (‘G series) devices this feature is called PinOsc (short for Pin Oscillator). The new Wolverine (‘FR58/59xx) devices have a similar feature, although it’s simply called “Cap Sense I/O”.

---

**GPIO pins with PinOsc Feature**
Gate Time

Gate Time is an important value in our measurement of capacitance. But before we delve into it, let’s talk about how we measure capacitance.

As we discussed earlier, the oscillations on a pin are directly proportional to the amount of capacitance. So this means we can get a relative measurement of capacitance by counting the number of oscillations in a fixed amount of time. This can be accomplished by using two timers, as we demonstrate below:

Effectively, we use the oscillations as the “clock” for the “Timer/Counter”; in this way, it’s really acting as a counter, counting the number of times the waveform oscillates up/down.

The other “Timer” in our diagram – we used the Watchdog (WDT) as an example – defines the amount of time in which we count RO oscillations. If we keep the measurement window time consistent, we can detect changes in capacitance by the varying number of oscillation counts.

So, back to the term Gate Time. Well, that’s just the name we give to the length of the measurement window.

Other than using a consistent Gate Time – to make sure our results are consistently accurate – Gate Time factors into a design in other important ways.

Longer gate times provide better results. The longer sampling time minimizes the effect of small perturbations, effectively averaging them out.

Then again, longer measurement periods mean more power is used. It also means that you can scan fewer electrodes – if quantity is one of your important careabouts.

Also, since the Timers are only used during the actual measurement window, they could be reused for other tasks at other times. Longer Gate Times means the timers are less available to other parts of your program.

In a few pages, when we discuss how to implement CapTouch, you’ll see the term Gate Time used quite a bit.
Power Requirements (and Scan Rate)

Talking of power, how little power is required by the MSP430 to perform CapTouch sensing?

There are many factors, but it basically comes down to time – how long do we have to expend energy scanning electrodes. The top three factors include:

1. How many Elements need to be scanned?
2. How long will the scan take? Basically, this is the idea of Gate Time, which we just discussed.
3. How often do you need scan your Elements?

Thinking about this last one, you may have seen a simple example that shows a Sensor being continuously scanned as part of the while loop at the end of main(). Conceptually, this is fine; but in a real-world system, this is wasteful. Scanning too frequently does not provide any benefit to the end-user; it just uses more power than is necessary.

This leads us to another important characteristic of CapTouch implementation, called Scan Rate. This is the rate – that is, how often – our Sensor’s should be scanned. The more often we scan, the more “responsive” the system will appear. But too often and we just waste power.

We have seen effective implementations, when power is absolutely critical, with Scan Rates as low as 2Hz (twice a second). While it’s nice to know we can go that low, it’s more common to see rates similar to that shown in the graphic below (20Hz).

This graphic is actually a screen capture from the Capacitive Touch Power Designer tool. It does an effective job at letting you specify all the system variables that will affect your power dissipation – it then calculates an estimated power budget for your CapTouch implementation. It even shows, in bar graph fashion, the power “states” of the processor. (We’ll see a larger drawing of these states on the next page.)
What information does the *Power Designer* tool need to know?

**Number of elements, Gate Times** (notice, you can list different Gate Times for different types of Sensors), **Scan Time**, as well as the **MSP430 device, voltage**, and what **clock frequency** you want to run system at.

Speaking of the ‘bar graph’ diagram from the Power Design tool, the next diagram is from one of the MSP430 marketing presentations; it does a good job of showing the power “states” of the MSP430 when running a CapTouch measurement.

![Ultra Low Power with Capacitive Touch I/O](image)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Current *</th>
<th>MSP430 State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gate Time</td>
<td>70 uA</td>
<td>LPM0 + Pin Osc</td>
</tr>
<tr>
<td>Processing Time</td>
<td>180 uA</td>
<td>Active</td>
</tr>
<tr>
<td>Sleep Time</td>
<td>0.5 uA</td>
<td>LPM3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1/Scan Rate</th>
<th>Average Current *</th>
</tr>
</thead>
<tbody>
<tr>
<td>50ms (20Hz)</td>
<td>3.7 uA</td>
</tr>
<tr>
<td>100ms (10Hz)</td>
<td>2 uA</td>
</tr>
<tr>
<td>500ms (2Hz)</td>
<td>0.7 uA</td>
</tr>
</tbody>
</table>

* MSP430 Value Line using Capacitive Touch I/O running at 1MHz, 1.8V, Gate Time = 1ms

Basically, when measuring 1 electrode, the CPU will go thru four CPU “power” states:

4. **CPU Active** – the CPU is required to setup the Timers (Gate Time and Oscillation counter).

5. **CPU in Low Power Mode** (LPMx) – unless you have something else for the CPU to do, it can be put into a Low Power Mode. The power usage here is threefold. Both timers will consume some power as they perform their tasks. Also, the pin oscillator – while very power efficient – does consume a small amount of power.

6. **CPU Active** – the Gate Time’r wakes up the CPU when the measurement is complete. The CPU then disables the timers and pin oscillators, thus freeing them up for other uses. The CPU may also have to do some calculations, depending upon what library function you used to perform the scan.

7. **CPU in LMP3** – once again, unless you have other work for the CPU to perform, it can go into Low Power Mode. In common use, your processor will spend more time asleep than awake. (Sounds kind of like a cat’s life, doesn’t it.)

The power numbers shown above are great, with average current, in one use-case, down below 1 µA. Some of the upcoming MSP430 devices will drive these power numbers even lower!
Measurement Methods (RO vs fRO)

Yet another implementation detail is the Measurement Method. There are a few different ways to measure capacitance, but the two most common methods are RO and fRO.

We know RO means Relaxation Oscillator. The other, fRO, just means Fast RO. The following graphic outlines the differences between these two methods.

Do you want to use a fixed gate time and count how many cycles occur? This method is called RO – which is what we’ve discussed up until this point in the chapter. This is more widely used than fRO.

With fRO, you measure the length of Gate Time while counting a fixed number of oscillations.

This excerpt (from Capacitive Touch Sensing, MSP430™ Button Gate Time Optimization and Tuning Guide (SLAA574.PDF)) provides some good reasons you might pick one over the other.

2.3 Selecting a Measurement Method

Determining whether the RO method or fRO method is right for a given system requires an analysis of the goals for the system as well as the system’s environment. For example, if the goal for a system is scanning a large array of keys while maintaining an HMI specification for response time, then fRO is more than likely the correct path for the design. (Conversely, if the goal for the system is to be a highly robust and highly noise-immune system for an automotive application, the RO method is a better choice.)

Because the RO method requires only one Timer_A peripheral, it also presents a value proposition as it can be used with devices that only have one Timer_A (MSP430G2xx2 devices, for example).
Sensor Threshold

*When is a “touch” really a touch?*

**Tuning** your system, to determine the answer to this question, is done empirically. Press the button and look at the resulting waveforms. Before we talk about how to see these waveforms, though, let’s discuss what **waveform** we are actually talking about.

By waveform, we are just talking about a plot of the count values from the timer. (Because, as we said before, the timer count values are directly proportional to the capacitance.) Another way to say this is that for every Gate Time measurement, we will get one timer count value that we can plot, as in the following graphic.

It appears that the above waveform consists of around 500 count times. That means we scanned this element 500 times.

When we draw these waveforms, we often normalize the data so that the background noise is minimized. Normalization also ‘fixes’ the waveform so that “touches” always go upward. If you looked at the Raw timer data, you would actually see the count values go down when there is a touch (since there are fewer oscillations).

With that stated, we can now define **Threshold** as a value you pick to represent a “positive” touch action. By examining your test data, you need to pick a count value that’s above the noise – but not so high that a ‘light’ finger press would be missed.

Our earlier diagram helps to demonstrate this idea.
MSP30 TouchPro Tool

Until recently, tuning your CapTouch system was more difficult. You would have to capture the timer count values, and then plot them with graphing software. (At least CCS had decent graphing/plotting capabilities, which helped to make this a little easier.)

With the release of the new TouchPro Tool (in May 2013), things have gotten a lot easier. You can send data to this tool (running on your PC) over the serial port directly from your application.

This means we can now view our ‘count’ value waveforms in near real-time. This makes experimentation much easier. You can try different types of contact (or proximity) and watch how they affect the waveforms in real-time. While the tool may be pretty simple to use, it’s very powerful in use.

![MSP430 Touch Pro Tool (GUI)](image)

- Using this tool, you can visualize your sensor count values in real-time
- Take the appropriate count threshold values from this display and plug them into your sensor definition data structures.

A few more comments about the tool:

- Multiple channels are supported. This is critical for multi-element sensors, like wheels and sliders.
- The tool expects data to be sent in a specific format. You can either use the function we created for our hands-on lab exercise – or write your own. The documentation provides all the details.
- The tool runs on Java, so you'll need to have at least the JRE (Java Runtime) installed for the tool to work.
- Save and restore waveforms, right within the tool – or export them for use in another program, like Excel.
- For better accuracy, you can select a point on the graph and look at the Y value to see its count value.
"He who fails to plan is planning to fail" Winston Churchill

Winston, we couldn't agree more. With the proper planning, CapTouch becomes much easier to implement.

To aid you in the Planning phase, we created a Planning Worksheet. Filling out this worksheet should step you through all the items you need to implement a CapTouch design. Over the next couple pages we will introduce you to most of the items on the worksheet. Later you will get hands-on experience with it in the lab exercises.

From a Coding perspective, using the TI CAPT library makes implementation pretty easy. The key is getting all the necessary data structures, timers and clocking implemented correctly. For a microcontroller programmer, these should be familiar issues to tackle.

In effect, we already introduced you to the Tuning phase of the design. We'll use the TouchPro GUI to assist us with Tuning our designs for just the right “feel”. The tool can also help you with tuning your gate and scan times; again, by providing real-time feedback.
If you like charting out your development using block diagrams, you might appreciate this, more detailed, planning sequence.

Steps to Implement Capacitive Touch/Sensing

**Design Phase**
- Determine # of:
  - Buttons
  - Sliders/Wheels
  - Proximity
  - Sensors

**Select MSP430 Device**

**Select a HAL:**
- Measurement Method (RO, fRO)
- GPIO pins
- Timers

**Estimate Timing:**
- Scan rate
- Gate times
  (Estimate power)

**Write Code**
- structure.h
- structure.c
- User’s code

**Responsive enough?**

**Tuning Phase**
- Gate Time (resolution)
  - ↑ time for more
  - ↓ time for less
- Threshold
  - Adjust up/down until desired feel

**Done**

1. Planning – What Do You Need to Detect?

The first section of our Planning Worksheet asks you what sensors/elements you need in your application.

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Quantity</th>
<th># of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sliders</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proximity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total # Elements</td>
</tr>
</tbody>
</table>

How would you fill out this Worksheet table, if we asked you to write a program to utilize all the features of the Capacitive Touch BoosterPack?
Here’s how we might implement a program for the CapTouch BoosterPack. One key item is remembering that Wheels (and Sliders) are made up of multiple elements. (Note: If you are unsure as to how many elements are in a Sensor – like the wheel – you should examine the board’s schematic diagram.)

### 1a. What Do You Need To Detect?

**a. What types of items do you need to detect in your application?**

For example, let’s look at the Cap Touch Boosterpack demo:

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Quantity</th>
<th># of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sliders</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wheels</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Proximity</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1+4+1 = 6</td>
<td></td>
</tr>
</tbody>
</table>

Which device to select? Picking a device with CapTouch I/O’s provides an obvious advantage. Beyond that, the choice really has more to do with whatever other tasks your system needs to solve. From a CapTouch perspective, the Value Line series (with PinOsc) provides great value.

### 1b. Device Selection

**b. Pick your device**

- Obviously, choosing a device with CapTouch I/O’s will minimize the hardware requirements of your system

<table>
<thead>
<tr>
<th>Device</th>
<th>Flash</th>
<th>RAM</th>
<th>Capacitive Touch I/O</th>
<th>Touch Buttons Supported</th>
<th>ADC</th>
<th>Serial (UART, I2C, SPI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP430G2x02</td>
<td>1-8 KB</td>
<td>256 B</td>
<td>✔</td>
<td>✔</td>
<td>≤ 16</td>
<td>10-bit</td>
</tr>
<tr>
<td>MSP430G2x32</td>
<td>1-8 KB</td>
<td>256 B</td>
<td>✔</td>
<td>✔</td>
<td>≤ 16</td>
<td>10-bit</td>
</tr>
<tr>
<td>MSP430G2x03</td>
<td>2-16 KB</td>
<td>512 B</td>
<td>✔</td>
<td>✔</td>
<td>≤ 24</td>
<td>✔</td>
</tr>
<tr>
<td>MSP430G2x33</td>
<td>2-16 KB</td>
<td>512 B</td>
<td>✔</td>
<td>✔</td>
<td>≤ 24</td>
<td>10-bit</td>
</tr>
<tr>
<td>MSP430F51x1</td>
<td>8-32 KB</td>
<td>1-2 KB</td>
<td>✔</td>
<td>✔</td>
<td>≤ 29</td>
<td>✔</td>
</tr>
<tr>
<td>MSP430F51x2</td>
<td>8-32 KB</td>
<td>1-2 KB</td>
<td>✔</td>
<td>✔</td>
<td>≤ 29</td>
<td>10-bit</td>
</tr>
<tr>
<td>MSP430FR58XX/59xx</td>
<td>FRAM</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>≤ 32</td>
<td>12-bit</td>
</tr>
</tbody>
</table>

- Your choice may also depend greatly upon what other device features – or clock frequencies - are needed in your complete application

- Is cost a consideration? The MSP430 Value Line devices (‘G2xxx) provide Cap Sensing I/O’s … while being very competitively priced
**Implementing Cap Touch**

**HAL** (hardware abstraction layer) is a common term used by libraries when their software directly touches hardware. This is exactly the case for the CapTouch library. They have defined over 20 different HAL implementations – this makes it easy to select the hardware that you want to use when implementing CapTouch.

Below, see how the HAL names are defined – each part of the name maps to different hardware functionality. The Library Users Guide provides HAL recommendations for most devices.

**1c. Select HAL (Measurement Method & H/W)**

- **Pick an appropriate HAL**
  - Cap Sense library defines various HAL (hardware abstraction layer) definitions
  - They make it easy to pick your method and h/w used by the library

**Considerations**

- Table reflects recommendations from the Cap Sense library authors
- HAL name reflects h/w choices
  - Choose between RO and fRO
  - How the RO is created
  - Which timers are used
The final part of the Planning process is to choose the frequencies and rates for your CapTouch implementation. Below, we show a screen capture from the TI CapTouch Power Designer tool, as it effectively encapsulates the details we need to specify.

1d. Gate & Scan Times

Fill-in the Power Designer GUI from our earlier values, then...

- **d. Estimate initial values for Gate and Scan times**
  - The default values are probably a good starting place.
  - What response time is needed – is 50ms appropriate in your application?
  - Will your application be battery powered?
  - What environmental conditions might need to be covered?
    - How to bond the overlay to the electrode?
    - What other electric/magnetic fields might be present in the system?
    - How thick is the overlay insulating layer over your sensor elements?

Note: During the ‘Tuning’ step, you may end up tweaking these time values.

We can enter the device we just selected. Then add your voltage and preferred clock frequency.

Next, enter the numbers of buttons, sliders, wheels and proximity sensors from the top of your Planning Worksheet. Finally, we need to choose the Gate and Scan timings. Notice that they allow you to specify different Gate Times for each type of sensor. This is appropriate, as it takes quite a bit more time to effectively read a Proximity sensor versus a Button.

What timing values should you choose? The default values, suggested by the Power Design Tool, are a good starting point. Based on the needs of your end application – extreme power sensitivity, lots of electromagnetic noise in your system’s environment, thick overlay material, etc. – you may want to alter these values.

If you’re experienced you may already know you need to alter your Gate or Scan times. For the rest of us, we may just want to choose the defaults and – based on feedback during the Tuning phase of our development flow – alter our timings later, if necessary.
2. Write Code

The planning is done, how do we translate these choices into the code that will drive our designs?

Using the TI CapTouch Library involves adding 3 things to your project, as well as adding a bit of code to your main program.

2. Writing Code

To begin, let’s examine what code you must either include or write to implement Capacitive Touch Sensing with the TI library

First thing is to add the Library files. It’s usually easiest to just add the entire folder (which contains 4 source files) directly to your project. We show you how to do this in the lab exercise. No changes are usually required to any of these files.

Next, add the two files: structure.c / structure.h to your project. The TI library provides many different code examples, one for each of the HAL definitions. We recommend that you copy the ‘structure’ files from the appropriate example. (It just means less editing later on.) You will use these files to describe the Elements and Sensors you have chosen for your application – basically, this comes down to allocating data struct’s for each “Element” and “Sensor”.

Finally, you will need to add at least three items to your mainline code:

- Call into the Library to establish the background capacitance of your system.
- Setup your own timer, configured to provide the Scan Rate you’ve chosen.
- Finally, whenever your Scan Rate Timer interrupts your program, you call a Library function to scan each of your sensors.

We quickly look at each of these here – but the lab gives you a chance to code it yourself.

---

1 The only reason to alter one of these files is when you plan to re-use one of the timers utilized by your CapTouch HAL selection. In this case, both ‘uses’ must share the interrupt service routine (ISR) which is defined in the Library source code (in the file, CTS_Layer.c).
structure.h

This file creates #definitions for many of the items you will use in your code. Oddly enough, this file also contains #defines for Library code. This being the case, you could say this file is divided into two halves – the top part you will need to edit to match your application choices, the bottom part must be left unaltered.

Looking at the graphic below, we can see the four changes you need to make to this file.

Except for adding extern definitions for each of your Element and Sensor structure variables, everything else deals with modifying #defines for your application. Some of these involve entering a value (from your Planning Worksheet) for a #define. Others only require you to un-comment the #defines that are required for your implementation choices.

For example, if you plan to use Slider sensors, you need to un-comment out two #defines towards the bottom of the user-configuration section of this file.
structure.c

The `structure.c` file also breaks into two halves. In this case, though, you will need to edit both halves.

The first part of this file involves defining each `Element` (i.e. electrode) in your system. You need to allocate – and complete – an `Element` data structure for each of the `Elements` you specified on your Planning Worksheet.

We have already discussed most of the concepts that ended up being fields of the `Element` data structure. At this point, we will leave it to the following graphic – as well as the upcoming lab exercises – for you to learn more about these data structures.

```
#include "structure.h"

// Define struct for each element in your system
const struct Element middle = {
    Specify GPIO pin (port & bit),
    .threshold = 121,
    .maxResponse = 121+655,
};
const struct Element down = {
    Specify GPIO pin (port & bit),
    .threshold = 113,
    .maxResponse = 113+655,
};
```

As a side note, here’s a little additional information about `.maxResponse`.

![Diagram of Element Measurement Parameters: Button Example](image)
The second part of the *structure.c* file involves creating data structures for each of the Sensors in your system (which should all be on your Planning Worksheet).

You might notice below that this is where we actually specify in our code which HAL we have selected. It’s also where we define the Gate Times in our code.

```c
const struct Sensor wheel = {
    .halDefinition = RO_PINOSC_TA0_WDTp,
    .numElements = 4,
    .points = 64,
    .sensorThreshold = 75,
    .baseOffset = 0,

    .arrayPtr[0] = &up,
    .arrayPtr[1] = &right,
    .arrayPtr[2] = &down,
    .arrayPtr[3] = &left,

    .measGateSource = GATE_WDT_SMCLK,
    .accumulationCycles = WDTp_GATE_8192
};

const struct Sensor myButton = {
    .halDefinition = RO_PINOSC_TA0_WDTp,
    .numElements = 1,
    .baseOffset = 4,
    .arrayPtr[0] = &middle,

    .measGateSource = GATE_WDT_SMCLK,
    .accumulationCycles = WDTp_GATE_8192
};
```

**structure.c (Sensors)**

- **Continuing structure.c**
  - The 2nd part of file allocates a structure for each Sensor
  - Sensors are made up of 1 or more Elements (defined at the top of the file)

- **Sensor structures contain:**
  - Which HAL you’ve chosen
  - How many elements in the sensor
    - A button has 1
    - Wheels/sliders have more than 1
  - # of points: How many virtual points do you want the library to track (for wheels/sliders only)
  - sensorThreshold: cumulative response required by the sensor (all elements) to declare a valid touch
  - baseOffset : think of this as a running count of all elements (starting with 0)
  - arrayPtr[i]: address of each Element used for this sensor (in order)
  - Timer info: Gate Time clock rate

You may have also noticed that Sensor data structures reference the structures for each of their Elements. In fact they do this in three ways:

- How many Elements are there for this Sensor?
- The Sensor structure has an array that contains pointers to each of the associated Elements. It’s important to note, that the Elements should be specified in this array in the order that a user would “swipe” over their associated electrodes. Doing so means that the Library Wheel function – TI_CAPT_Wheel() – will be able to provide you with positional information (i.e. where along the wheel the user’s finger is positioned).
- Finally, the .baseOffset field is also references the list of Elements.

![Baseline Array of Elements](image)

- The Library maintains a 'baseline' array of all the elements you define.
- The .baseOffset field in the Sensor data structure should reference the index of it’s first element (i.e. arrayPtr[0] )
- For example, in the Slider shown here, we would want to reference “element3”: `.baseOffset = 3;`
### main.c

As we mentioned earlier, there aren't many things we need to add to our mainline code to enable CapTouch. One of these items is hidden in the function below, called `ConfigTimer1()`. The code in this function is not CapTouch specific; rather, this is the timer we use to generate our Scan Rate time-base. Whenever this timer interrupts the CPU, we go run the `TI_CAPT_Button()` function in the Timer1 interrupt service routine.

```c
#include "CTS_Layer.h"
#include "structure.h"

// Main Function
void main(void)
{
    WDT_halt(); // Turn off watchdog timer
    ConfigClocks();
    ConfigPins(); // Initialize pins for ULP
    ConfigTimer1(); // Set Timer for Scan rate

    // Establish capacitance baseline
    TI_CAPT_Init_Baseline( &myButton );
    TI_CAPT_Update_Baseline( &myButton, 5 );

    // Background loop starts here
    while (1) {
        __bis_SR_register( LPM1_bits + GIE );
    }
}
```

The preceding graphic shows how we can use the Library functions to manage and access the Sensors in our system. First, for each Sensor, we needed to establish an initial capacitance baseline value.

With that done, we can access the Sensor's "value" using the appropriate function:

- `TI_CAPT_Wheel()`
- `TI_CAPT_Slider()`
- `TI_CAPT_Button()`

See the input/output descriptions for each of these functions in the graphic to the right →
3. Tuning Sensors (Threshold, Gate time)

Do you remember where we specified the \textit{threshold} value in our code?

If you said, “The Element's data structure”, then you got it correct.

In practice, we set the \textit{Threshold} to “0” when we first write our code. We then run the code, connecting it to the \textit{TouchPro GUI} tool and measure the count values for various degrees of touch. We then select an appropriate \textit{Threshold} value for each \textit{Element}, then go back and add that value to our code (in the \texttt{structure.c} file).

What if there is so much noise that you cannot pick an effective \textit{Threshold} value?

You might remember that we can increase a \textit{Sensor}'s sensitivity by increasing the Gate Time. Here’s a case where you might want to do that. Luckily, we won’t see this in our upcoming lab exercise – there will be some noise, but it won’t be significant enough to cause a problem.

\textbf{Note:} There is a short discussion on increasing \textit{Gate Times} and sensitivity in the \textit{Hardware Sensor Design} part of this chapter (page 10-35).

Finally, how do we connect our program to the GUI tool?

We need to “instrument” our code (i.e. add a little bit of code) to connect our program to the tool. For each \textit{Sensor}, we add two function calls to our code.

- The first call, \texttt{TI\_CAPT\_Custom()} returns an array of normalized timer “count” values. This array consists of one ‘count’ value for each \textit{Element} in the \textit{Sensor}.
- We then use the UART to send these values to the host computer running the \textit{TouchPro GUI}. The GUI recognizes and plots the data in real-time.
Summary: TI's Cap Touch Library (TI_CAPT)

Here is a block diagram summary of TI's CapTouch Library. We can see the HAL definitions we discussed earlier sitting adjacent to the MSP430 device hardware.

The next layer provides some internal (to the Library) functions to filter and extract the important data for us.

The final layer provides user-callable functions. Some of these are more abstracted than others. For example, as we saw before, the Button function returns a binary value of whether it's being touched or not. At the other end of the spectrum, the Custom and Raw functions provide little to no abstraction. We use Custom for Tuning our system; Raw is rarely used in straightforward CapTouch applications, but could be very useful in unique circumstances.
Additional Topics

Hardware Sensor Design (For Reference Only)

We state “For Reference Only” since hardware design is really outside the scope of this workshop. Our main purpose here is to bring awareness to a few h/w considerations … but more importantly, to refer you to the excellent designers guide:

**Capacitive Touch Sensing, Sensor Design Guide**
(http://www.ti.com/lit/pdf/slaa576)

### Hardware Design of Capacitive Sensors

**Hardware sensor design is outside the scope of this workshop**

- We hope, in this section, to provide an awareness of some design aspects and how they will affect your software
- Check the “For More Info” topic for App Notes references

**General hardware design considerations**

- Connecting to sensor electrodes (i.e. touchpads)
  - Keep the wire as short as possible as it adds to the base capacitance (C)
  - Similarly, try to avoid sharp bends as this also affects C
  - Avoid putting high-speed/current drive wires next to the touchpad wires
- Touchpad shape & size
  - Normal solid filled circular or square pads work well
  - The pad can have a hole drilled through it to provide backlighting without influencing the capacitive performance
  - For scrollbar/slider, don’t make the pads too big, a normal finger should be able to cover 1½ touchpads
- PCB considerations
  - Since Cap Sensing is often placed on top of other electronics, it is often helpful to put grounding on the underside of the PCB
  - Overlay materials should be a good dialectric (non-conductive); and eliminate any air holes between overlay and sensor

### Sensor Element – Size Makes a Difference

**Common Sense Rules**

- Larger sized electrodes are easier to detect
- Law of diminishing returns occurs when the pad is larger than the finger touching it
- Thicker overlay materials reduces capacitive touch effectiveness
- Try counteracting smaller signals with longer gate times … but increases power

**Bottom Line**
Getting “more” interaction between a sensor and finger provides a better signal
TI provides quite a few options when implementing Capacitive Touch and Haptics solutions. You find the MSP430 class solutions under the “Capacitive Touch” tab at http://www.ti.com/touch.

Here’s a picture of this page, but since it’s so small, we’ve repeated references to all the applications notes, user guides, and documents on the next page of this book.
To learn more…

◆ Training
  • Getting Started with the MSP430 LaunchPad Workshop

◆ Tools
  • MSP430 Touch Pro Tool
  • MSP430 Capacitive Touch Power Designer GUI

◆ Design Guides
  • Capacitive Touch Sensing, MSP430™ Button Gate Time Optimization and Tuning Guide
  • Capacitive Touch Sensing, MSP430™ Slider/Wheel Tuning Guide
  • Capacitive Touch Sensing, Sensor Design Guide
  • Capacitive Touch Sensing, SYS/BIOS
  • Getting Started with the MSP430 LaunchPad Workshop
  • MSP430 Low Cost PinOsc Capacitive Touch Overview
  • MSP430x5xx and MSP430x6xx Family User’s Guide (Rev. M)

◆ Software Library
  • MSP430 Capacitive Touch Sense Software Library

◆ Reference Designs
  • 1-uA Capacitive Grip Detection Based on MSP430 Microcontrollers Appnote
  • MSP430 Low Cost PinOsc Capacitive Touch Keypad
  • 10cm Proximity detection Appnote
  • Wireless Remote Controller With Capacitive Touch Pad Using MSP430F51x2

Go … Read … Learn more about implementing Capacitive Touch/Sensing with the MSP430!
Lab 10: Capacitive Touch

Objective

The objective of this lab is to learn the hardware and software utilized by the capacitive touch technique on the MSP430 LaunchPad and Capacitive Touch BoosterPack.

Lab10a: Explore the Boosterpack Demo
Lab10b: View wheel counts using the TI TouchPro GUI Tuning Tool
Lab10c: Implement a simple touch key application
Lab10d: (Optional) Use Grace to configure clocks/timers

Requirements

- Capacitive Touch BoosterPack (430BOOST-CAPTOUCH1) available here for US$10.
- During Workshop Installation, you should have downloaded and installed the following:
  - Getting Started with Capacitive Touch - http://www.ti.com/lit/scala491c

- The Capacitive Touch BoosterPack includes an MSP430G2452 that is pre-programmed with a capacitive touch demo. To eliminate the potential to break the pins of your devices while extracting them, if you have version 1.5 of the LaunchPad board, we will simply reprogram the ‘G2553 already on your board. In other words, we recommend that you DO NOT INSTALL the MSP430 device that comes with the Cap Touch BoosterPack.

- This lab exercise was written for the MSP430 Value-Line LaunchPad kit - version 1.5. This version has been shipping for 1½ years, as of this writing.

If you are using the older version 1.4 of the Value-Line Launchpad, we recommend: downloading the older version of this workshop (v2.10) as it contains directions and lab files for using the ‘G2452 device that comes with the Cap Touch BoosterPack.
Lab Topics

Lab 10: Capacitive Touch

Objective .......................................................... 10-38
Requirements ...................................................... 10-38
Lab 10a – Capacitive Touch Boosterpack Demo .......................................................... 10-40
Lab 10b – Touch Pro GUI Tool ‘Wheel’ Demo .............................................................. 10-41
Lab 10c – Create a Capacitive Sense Project From a Blank Page .......................... 10-45
Planning .............................................................. 10-45
  Complete the Cap Touch Planning Worksheet ......................................................... 10-45
Project and File Management .................................................. 10-48
  Create (and Explore) New Project ........................................................................ 10-48
  Add Cap Touch Sensing Library to Your Project .................................................. 10-53
  Add Starter Files from Library’s Examples ............................................................. 10-53
Writing/Editing Code .......................................................... 10-56
  structure.c .............................................................................................................. 10-56
  structure.h ............................................................................................................. 10-58
  main.c .................................................................................................................. 10-60
  Build, Load and Run ............................................................................................. 10-63
Tuning the Button .................................................................................. 10-64
  Threshold ............................................................................................................ 10-65
Ideas for Further Exploration .......................................................... 10-65
(Optional) Lab 10d Using Grace to Configure Clocks & Timer .................................. 10-66
  What Do We Want? .............................................................................................. 10-66
  Create Project and Use Grace ............................................................................... 10-66
  Accessing the Code From Grace ........................................................................... 10-72
Lab 10a – Capacitive Touch Boosterpack Demo

Install Hardware and Run the Boosterpack Demo Program

1. Verify that you have all the necessary requirements as listed on page 10-38.

2. Plug the BoosterPack PCB onto the top of the LaunchPad’s Molex BoosterPack connectors.
   
   Make sure the Texas Instruments logo is nearest the buttons on the LaunchPad board. Then plug the LaunchPad board into your computer’s USB port.

3. Open Code Composer in your usual workspace and import the Cap Touch demo.
   
   Project → Import Existing CCS Eclipse Project

   Import the project:
   
   C:\MSP430_LaunchPad\Labs\Lab10a

   Make sure that the single discovered project is selected: that you are NOT copying the files; then click Finish.

4. Build the project and load it into the MSP430 on your Launchpad.

   Click on the project in the Project Explorer pane to make it active, and then click the Debug button on the menu bar to build and program the code into your ‘G2553 device.

5. Terminate Debug and close the project.

   Click the Terminate button on the CCS menu bar to return to the debug perspective. Close Lab Lab10a.

6. Cycle the power on the LaunchPad board by removing and re-inserting the USB connection.

7. Bring the demo out of sleep mode and try out the buttons and scroll wheel.

   Pass your hand close over the Capacitive Touch surface – the demo is set to remain in low power mode until it detects a proximity event. You should see the LEDs illuminate in sequence.

   Touch your fingertip to the rocket button in the center circle and note the LED under it and the red LED on the LaunchPad PCB light. Touch again to turn them off. Also, try touching between the inner and outer circle to momentarily illuminate LEDs on the outside ring.

8. Run the program: CapTouch_BoosterPack_UserExperience_GUI.exe

   Find the program in the SLAC490 folder that you downloaded (and unzipped)

   <SLAC490>\Software\CapTouch_BoosterPack_UserExperience_GUI\CapTouch_BoosterPack_UserExperience_GUI.exe

   Give the tool a few moments to link with your LaunchPad, and then touch any of the Capacitive Touch buttons. Note that gestures are also recognized.

9. Exit the Demo’s GUI tool when you are done. Also, close the Lab10a project in CCS.
Lab 10: Capacitive Touch

Lab10b – Touch Pro GUI Tool ‘Wheel’ Demo

The Texas Instruments Touch Pro GUI provides an example application that only utilizes the scroll wheel feature of the Boosterpack. Additionally, it contains the serial port code required to send data to the Touch Pro GUI.

1. Verify which Windows COM port your MSP430 Launchpad is using:

   If you haven’t already done this info earlier in the workshop, please open the Windows Device Manager and write down the UART COM port number.

   MSP430 Application UART (COM __________ )

   **Hint:** In Windows 7 or 8, just click on the Start Menu, then type in “Device Manger”.

2. Open Code Composer in your usual workspace and import the Cap Touch demo.

   **Project → Import Existing CCS Eclipse Project**

   Import the project:

   C:\TI\msp430\TouchPro_1_01_00_00\32bit\TouchProTool_project_files\examples\TouchProTool_Demo_Bit_Banging\CCS

3. Build / Load / and Run the project on your MSP430 Launchpad.

   Click on the project in the Project Explorer pane to make it active, and then click the Debug button on the menu bar to build the program and download it into your ‘G2553 device.

   Click Run … unfortunately, the current version of this demo does not utilize any LED’s, so you cannot see that it’s running … until we get the TouchPro GUI running.
4. Run the MSP430 TouchPro GUI tool from the Windows Start menu:

![MSP430 TouchPro GUI](image)

5. Select your COM port in the TouchPro tool.

![Select COM port in TouchPro](image)

**Troubleshooting Suggestions**

If your COM port doesn’t show up:

1. Halt the program, unplug the Launchpad, wait 10-15 seconds and then plug the board it back into a different USB port … once Windows says it’s found the board, try launching the TouchPro GUI and select your COM port.

2. Try doing the same thing as tip #1, but this time close CCS before (or after) you unplug the Launchpad.

3. Try troubleshooting tip # 2 (closing CCS and unplugging the board) a couple more times. Note, since the program should now be in flash, you should not have to re-open CCS in order to use the TouchPro GUI tool.

4. Occasionally, we have had to reset our computer for the TouchPro tool to gain access to the Launchpad’s COM port.
6. **Visualize Capacitance Touch/Sensing in the TouchPro GUI.**

   Once the tool connects, you will see a lot of noise displayed in the interface:

   ![CapTouch Display](image)

   This is just background noise. As soon as you touch the wheel touchpad, you should see the display automatically re-scale the display. This will minimize the noise and you’ll see the much larger count values pertaining to your touch.

   ![CapTouch Display](image)

   If you scroll around the wheel you should see something like this (notice the scale on the left has changed):

   Notice the 4 channels of data. These are coming from the four elements that make up the scroll wheel.

   ![CapTouch Display](image)

   Try touching on the buttons (i.e. arrows) on the wheel. You should see each channel peak with a single impulse.

   Later, we will utilize this tool to help us ascertain the best Threshold value for a button.

7. **Disconnect the Launchpad from the display by clicking the USB connector icon.**

   ![USB Connector Icon](image)
8. **Save the data collected by the MSP430 TouchPro Tool.**

   Select *Save* or *Save As* from the *File* menu to save the data. Later, you could open this data and view it again … or you could open the data file in another program, such as Excel.

   **Note:** The GUI does not keep an indefinite amount of data. After about 500 samples it starts discarding the oldest data.

9. **Terminate your Debug session.**

10. **Close the project.**

    Before closing the project, you are welcome to examine the project’s source code, although in the next lab we will create our own example.

11. **You can also exit the GUI tool, as we won’t be using this for the next exercise.**
Lab10c – Create a Capacitive Sense Project From a Blank Page

In this section, we’ll learn how to build a simple Capacitive Touch project. The goal of this is to use the middle button on the BoosterPack board to light the middle LED. When we push the button, the LED should go on; when we release the button, the LED should go off.

Planning

As discussed in the presentation, in the Planning phase we’ll make many of our design decisions. This should enable us to quickly create code during the Design phase of the lab.

As we said, our goal in this lab is to get the Middle Button working on the Capacitive Touch Boosterpack. ‘Clicking’ this button should turn on the middle LED. (Note: The middle LED on the Boosterpack is connected to the same pin as the RED LED on the ‘G2553 Value-Line Launchpad.)

Here is a diagram of the Port/Pin connections to the BoosterPack Touchpad electrodes. From this diagram (and the schematic, if you go examine it), we want to scan I/O Port 2, Bit 5.

![Diagram of Port/Pin connections](image)

Complete the CapTouch Planning Worksheet

1. Fill out the Capacitive Touch Planning Worksheet.

   A couple copies of this worksheet should be in this workbook. (In fact, it should be the next 2 pages in the book.) You might find it easier if you remove the page so that you can refer to it throughout the exercise. (You’ll also find a PDF copy of the worksheet in the Lab10c folder.)

   a) Sensor / Element Table  (The "What do you want to create" table…)

      The worksheet begins with the table where you list the number of Sensors and Elements you will have in your design.

      Fill in the number of Sensors & Elements portion of the worksheet

      **Hint:** Remember, Buttons and Proximity sensors only have 1 Element, but Wheels and Sliders usually have more than 1.
b) Variable Names and Port/Pin#’s

On the bottom half of the first page, you need to create a list of variable names for each of the Sensors and Elements. We will use these names when we create data structures for each of them.

The Port/Pin#’s are important since we need to specify this information in the data structures of each of Element. (This is required as the Library function call needs to know which GPIO pin to scan when sensing capacitance.)

For each Sensor/Element, Fill-in the Variable names & Port/Pins #’s

Hint: We recommend that you list the multiple Elements of Sliders and Wheels “in order”. On your board, the elements of multi-element sensors are laid out in a specific order, following the way a user would slide their finger across the elements in a continuous motion. For example, if a slider had 4 elements, as the user swiped from K0 to K1, the library would pick up less-and-less of K0, and more of K1.

For the library to calculate this correctly, we need to specify the Elements of our Sensor “in order”. Listing them on the worksheet in the correct order will make it easier when we create the Sensor data structure in the Design/Coding phase.

All this discussion … and we’re not even using a slider or wheel in this part of the lab.

c) Select a MSP430 device and HAL

This is an easy decision for this lab exercise, since we are using the Boosterpack with the MSP430 Value-Line Launchpad. (If you still need a hint, check out what is highlighted on the worksheet.)

Of course, in your own application you may decide to choose a different device based on your system’s needs. For example, the MSP430G2553 does not have a USB port; if that’s something you need, you might choose another device, such as the MSP430F5529.

Write in your selections

d) Selecting Clock, Gate, and Scan Rates

Fill in the remaining portion of the worksheet

This has you re-enter the #’s of Elements from the beginning of the worksheet, as well as your selected device.

You will also pick the speed and voltage of the device. Finally, you need to enter the necessary Gate Time(s) and Scan Rate. We suggest using the default values from the MSP430 CapTouch Power Designer. (In fact, we will be using the Power Design Tool in the next step of the lab.)
# Capacitive Touch Planning Worksheet

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Quantity</th>
<th># of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sliders</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proximity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total # Elements**

## Variable Names (for Sensors & Elements)

List the variable Names for Each Sensor and Element you will be using. For example, if creating a “wheel” sensor that contains 4 elements, the variables might look like:

```
myWheel_Sensor      left_element  Port 2  Bit 1
                    up_element    Port 2  Bit 4
                    right_element Port 2  Bit 3
                    down_element  Port 2  Bit 2
```

<table>
<thead>
<tr>
<th>Sensor Name</th>
<th>Element Name</th>
<th>GPIO Port</th>
<th>PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Planning Worksheet Page 1
Selection Device & HAL

Please select the MSP430 device that you plan to use in your system. Remember, there are devices, such as the MSP430G2553 (used on the Value-Line Launchpad) that have hardware dedicated to Capacitive Sensing.

You should also select a HAL (hardware abstraction layer) from the TI Capacitive Touch Library. The library documentation recommends specific HAL choices for various devices (as shown to the right).

<table>
<thead>
<tr>
<th>MSP430 Device</th>
<th>HAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Capacitive Touch Library</td>
</tr>
<tr>
<td></td>
<td>Hardware Abstraction Layer</td>
</tr>
</tbody>
</table>

Select Initial Gate/Scan Rates … Run the Power Design Tool

Enter the following values. The timing values under the gate times (and next to the scan rate) are good initial suggestions. We recommend using a ‘pencil’ for the timing values, since you may want to change them during the Tuning phase of development.
### Capacitive Touch Planning Worksheet

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Quantity</th>
<th># of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sliders</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proximity</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total # Elements</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Variable Names (for Sensors & Elements)

List the variable Names for Each Sensor and Element you will be using. For example, if creating a “wheel” sensor that contains 4 elements, the variables might look like:

- myWheel_Sensor
- left_element
- up_element
- right_element
- down_element

<table>
<thead>
<tr>
<th>Sensor Name</th>
<th>Element Name</th>
<th>GPIO Port</th>
<th>PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Device & HAL

Please select the MSP430 device that you plan to use in your system. Remember, there are devices, such as the MSP430G2553 (used on the Value-Line Launchpad) that have hardware dedicated to Capacitive Sensing.

You should also select a HAL (hardware abstraction layer) from the TI Capacitive Touch Library. The library documentation recommends specific HAL choices for various devices (as shown to the right).

<table>
<thead>
<tr>
<th>MSP430 Device</th>
<th>HAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Capacitive Touch Library</td>
</tr>
<tr>
<td></td>
<td>Hardware Abstraction Layer</td>
</tr>
</tbody>
</table>

Select Initial Gate/Scan Rates … Run the Power Design Tool

Enter the following values. The timing values under the gate times (and next to the scan rate) are good initial suggestions. We recommend using a ‘pencil’ for the timing values, since you may want to change them during the Tuning phase of development.
2. Run the MSP430 CapTouch Power Designer to estimate the power requirements.

From the start menu choose:

Running the Power Designer should give you a result similar to this:

Notice, the CPU should be in Low Power Mode most of the time.

3. Finish the worksheet and close Power Designer.

If you haven’t already done so, finish entering your selections in the Planning Worksheet. When finished, go ahead and close the Power Design tool.

Hint: If you make changes to any of the items specified in the Power Designer tool, you may want to re-open the tool and apply the new choices. This will give you an updated estimate of your power usage.
**Lab 10: Capacitive Touch**

### Project and File Management

#### Create (and Explore) New Project

4. Create a new CCS Lab10c project.

   ![New CCS Project](image)

   - **Project** → **New CCS Project**
   - Locate the project:
     
     C:\MSP430_LaunchPad\Labs\Lab10c

   ![Screenshot of CCS Project settings]

   - **Project name**: Lab10c
   - **Output type**: Executable
   - **Location**: C:\MSP430_LaunchPad\Labs\Lab10c

**Note:** You should get a **Warning** if you choose the template: *Empty Project (with main.c)*

This is because the Lab10c folder already has a `main.c` file in it. The warning notifies you of the conflict; as well as telling you it could not create `main.c`. (In other words, it leaves the original `main.c` untouched.)
5. **Expand the Lab10c project in the Project Explorer pane.**

Even though we chose the CCS "Empty Project" template, you should notice there are six source files already in your program.

*Note, if you have completed the preceding labs in this workshop, the code we have provided should look familiar.*

Let’s examine the code that is already in the project:

- **Grace folder:** These files came from the Grace tool. We “cheated” by using this tool to create the functions needed to setup our Clocks and Scan Rate Timer. If you want see how we created these files, please refer to *(Optional) Lab10d Using Grace to Configure Clocks & Timer* (on page 10-66).

- **BCSplus_init.c:** This file sets up our clocks per the Planning Worksheet;
  
  | MCLK  | DCO = 1 MHz | (Note, we discuss clocks in Chapter 3) |
  | SMCLK | DCO / 2 = 500 Khz |
  | ACLK  | VLO = 12 KHz |

- **Timer1_A3_init.c:** Configures Timer1_A3 to create interrupts at 20 Hz rate.
  
  - This matches the 20 Hz (50ms) Scan rate described on the Planning Worksheet.
  - We use this rate for two reasons: (1) We will flash an LED at 1Hz to indicate the system is alive; and (2) We want to scan our CapTouch sensors every 50ms.
  - Notice in `main()` that we call `Timer1_A3_graceInit();` this configures the timer. Looking further down in `main.c`, you’ll find the interrupt service routine (ISR). Currently, this only “blinks” the LED at 1Hz. Later, you will add code to scan the CapTouch button. (Note that we discussed interrupts and ISR’s in Chapter 5.)

- **UART folder:** This code was extracted from the *Bit-Banging* example that ships with the TouchPro Tuning Tool.
  
  - Folder contains the files: `uart.c`, `uart.h`
  - The primary function, `UART_sendDataFrame()`, sends formatted count data across the UART’s serial port to the TouchPro GUI.
  - Learn more about UART serial ports in Chapter 7.
• **unusedInterrupts.c**: The MSP430 compiler complains (i.e. warns you) if there are interrupt vectors that have not been configured, yet.
  - This file simply defines all the interrupt vectors and maps them to a `UNUSED_HWI_ISR()` function.
  - Notice that two vectors in this file are commented out. This is because these two interrupts are used (and defined) in this lab. *Timer1* vector is defined in `main.c`, while the *Watchdog* vector will be utilized by the *CapTouch Library*.

• **main.c**:  
  - The `main()` function sets up the clocks, GPIO, and Timer1; the function ends in a while loop that enables GIE (global interrupt enable) and puts the CPU to sleep.
  - The CPU stays asleep waiting for the Timer1 interrupt to wake it up.
  - `GPIO_init()` and the Timer1 ISR functions are both defined in this file.

6. **Build the project.**

Before we add the Capacitive Touch library and code, let’s get the base project built and working.

Build the project by clicking on the toolbar – *Hammer icon*.

During the build, you should have noticed an error. Apparently, the compiler cannot find a header file.
7. **Add (-I) include search paths to your project.**

   Add search paths to the project’s Build properties. You need to add these 2 locations:
   - Project directory
   - Uart directory (inside the project folder)

   Open the Project Properties:
   
   Right-click on the project → Properties
   Select: Build → MSP430 Compiler → Include Options
   Then click on the: Add path button

CCS makes it easy to add a folder from the workspace, just click on the *Workspace* button and choose the appropriate folder.

Do this 2 times, each time selecting a different required folder.

The `#include search path` should end up looking like this:
8. **Trying building your file again. Upon a successful build, go ahead and run the program.**

   When running, you should see the Green LED (as well as LED6 on the Boosterpack) flashing on/off about once per second. If this is the case, then we know that our clocks and 20Hz scan rate timer are working properly.

9. **Terminate debug and go back to editing your program.**

   Now we can get on with adding the actual Cap Touch code.
Lab 10: Capacitive Touch

Add Cap Touch Sensing Library to Your Project

10. Import the Capacitive Touch Library.

This library has not been pre-built, rather, it is provided as a folder of source (.c/.h) files. You should have installed these files when you downloaded and unzipped SLAC489.ZIP. Import the Library source files folder as shown.

File → Import → General → File System

![Import Screen]

Note the “From Directory” in the graphic above. If you installed your files per our directions, you should have this folder. If you chose a different path for the CapTouch Library, your ‘From’ directory location will be different.

11. Add another (-I) search path to the new Library folder.

Once again, this is easy since we can just add it using the Workspace button.

**Hint:** Every time you add source files to a project, stop and think if you need to include a directory to the compiler’s search path.

**Note:** If you build the program right after step 11, you will get an error. We will “fix” this on the next page.

Add Starter Files from Library’s Examples

12. What HAL version did we choose in our Planning for this application?
13. **Copy** `structure.h` and `structure.c` from the CapTouch Library’s Code Examples folder.

Rather than create these files from scratch, we’re going to copy then modify them – which is what we recommend you do when you implement your own design.

We suggest choosing the files that match your HAL selection. The HAL we chose was RO_PINOSC_TA0_WDTp; therefore, we recommend copying the ‘structure’ files from that directory path. (Doing so will mean fewer modifications required later on.)

Right-click on your project → Add Files…

Navigate to your CAPT library Code Examples folder and copy the two files to your project:

<your cap touch library folder>/Code_Examples/RO_PINOSC_TA0_WDTp

---

14. **Build your project.**

Now that we have added all the necessary files – though we still need to edit a few of them – let’s build the project to make sure all the project references are correct.

Oops, did you get a bunch of errors like we did?

Oh, that’s right. If we look through the documentation for the library (SLAA490b.PDF; Section 3.1; Page 9), it tells us that the library requires that the “GCC language extensions” should be enabled for CCS.
15. Enable support for GCC extensions.

Right-click on Project → Properties → Build → MSP430 Compiler → Advanced Options → Language Options → Enable support for GCC ext’s

You may remember that one of the files we just added (structure.c) sets up a number of data structures, which we’ll be editing in a few steps.

We want this option, since it enables our program to access uninitialized structures; e.g. allowing element three to be accessed without having to access elements one and two.

For more information, see:


16. Try building your program again.

Your program should build without errors once the GCC switch is enabled.

**Hint:** Please remember that if you try to building your program with the Release (i.e. optimized) build configuration, you will end up running into the same path and gcc errors.

In other words, you must add these options to every build configuration that you plan to use.
Writing/Editing Code

Get your Planning Worksheet ready. We’ll be using it as we write our code and fill-in the blanks.

structure.c

Per our earlier planning, we want our structure.c file to contain 1 Sensor (middle button) which contains 1 Element. The structure.c file we copied gets us close to what we need; it just requires a little editing.

17. Open structure.c and remove the code that isn’t required.

- Delete the superfluous comments at the beginning of the file (tedious to scroll thru)
- Eliminate all Element struct’s except the middle_element.
- Finally, we’ll keep the middle_button sensor, but you can get rid of the other two

At this point, the file should look like this:

```c
#include "structure.h"

//PinOsc Wheel: middle button P2.5
const struct Element middle_element = {
    .inputPxselRegister = (unsigned char *)&P2SEL,
    .inputPxsel2Register = (unsigned char *)&P2SEL2,
    .inputBits = BIT5,
    // When using an abstracted function to measure the element
    // the 100*(maxResponse - threshold) < 0xFFFF
    // ie maxResponse - threshold < 655
    .maxResponse = 350+655,
    .threshold = 350
};

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

Note:
Don’t accidentally delete the #include. You’ll need this!
18. Update the following items – most of them are from the Planning Worksheet.
Many items are fine, just as they are:
- Port and bit definitions
- HAL selection
- Number of elements
- Gate clock source
But other items need to be updated...

**Hint:** Try changing the “middle_element” name using the CCS refactoring feature. Highlight the variable, then: Right-click → Refactor → Rename You should see the name change in both places where it exists.

Set initial Threshold to 0. We choose to use a #define to make updating easier.

Change variable names to those on your Planning Worksheet.

All elements are stored by the library in an array. Since we now only have 1 Element, its base offset (i.e. index) is zero.

Our planning worksheet says we have a gate time of 1.024ms
SMCLK = DCO (1MHz) ÷ 2 = 512 K
Gate Time = SMCLK ÷ 512 = 1.024ms
structure.h

19. Open structure.h for editing.

20. Update the Public Globals area.

Replace all the declarations listed here with two that match your needs. You should have one Element and one Sensor declaration named according to your Planning Worksheet. (These should match the structures we just declared in structure.c.)

21. Update the values in the Ram Allocation area of structure.h.

a) The #define for TOTAL_NUMBER_OF_ELEMENTS should match the value on the first page of your Planning Worksheet.

b) Also, make sure that the definition for RAM_FOR_FLASH is uncommented (since, in this exercise, we don’t want to use dynamic memory).

22. Set the maximum number of elements per sensor definition.

Calculate the maximum number of elements per sensors ... in our case this is easy. It’s 1. So, in the Structure Array Definition area, update the definition for:

```
#define MAXIMUM_NUMBER_OF_ELEMENTS_PER_SENSOR 1
```

23. Verify/update the remaining #define’s in the “User Configuration Section” of the file.

You should specifically check on two items in the remaining part of the User Configuration Section (the top part of this file):

- Make sure the HAL you chose (on your Planning Worksheet) is uncommented. If you imported this file from the correct code example, this should already be set correctly.

- Since we’re not using a slider or wheel in this exercise, you can comment out their #definitions.
24. Save your changes. The top portion of your code should look like our code below:

Note, we removed some of the commented-out #defines to minimize space in this listing:

```c
//****************************************************************************
// The following elements need to be configured by the user.
//****************************************************************************
#ifndef CTS_STRUCTURE
#define CTS_STRUCTURE

#include "msp430.h"
#include <stdint.h>

/* Public Globals */
extern const struct Element myButton_element;  // myButton_element is defined in structure.c
extern const struct Sensor  myButton;          // myButton is defined in structure.c

//****************************************************************************
// TOTAL_NUMBER_OF_ELEMENTS represents the total number of elements used, even if
// they are going to be segmented into separate groups. This defines the
// RAM allocation for the baseline tracking. If only the TI_CAPT.Raw function
// is used, then this definition should be removed to conserve RAM space.
#define TOTAL_NUMBER_OF_ELEMENTS 1

// If the RAM_FOR_FLASH definition is removed, then the appropriate HEAP size
// must be allocated. 2 bytes * MAXIMUM_NUMBER_OF_ELEMENTS_PER_SENSOR + 2 bytes
// of overhead.
#define RAM_FOR_FLASH

//****************************************************************************
// This defines the array size in the sensor structure. In the event that
// RAM_FOR_FLASH is defined, then this also defines the amount of RAM space
// allocated (global variable) for computations.
#define MAXIMUM_NUMBER_OF_ELEMENTS_PER_SENSOR 1

//****************************************************************************
// These variables are references to the definitions found in structure.c and
// must be generated per the application.

#ifndef RO_COMPAp_TA0_WDTp
#define RO_COMPAp_TA0_WDTp 64
#endif
#ifndef RO_PINOSC_TA0_WDTp
#define RO_PINOSC_TA0_WDTp 65
#endif

//****************************************************************************
// Are wheel or slider representations used?
#ifndef SLIDER
#define SLIDER
#endif
#ifndef ILLEGAL_SLIDER_WHEEL_POSITION
#define ILLEGAL_SLIDER_WHEEL_POSITION 0xFFFF
#endif

//****************************************************************************
// End of user configuration section.
//****************************************************************************
```
main.c

We’re now going to adapt the current program so that it lights the LED when the middle button is touched. We already have a program that initializes our system and puts the CPU to sleep; then, Timer1 wakes the system every 50ms and toggles the green LED.

Further, we have already defined our capacitive touch sensor (and element) in the structure.c./h files. What remains is adding a few items to main.c. These include:

- A couple header files and global variables
- Code to read the sensor’s baseline (background) capacitance
- An if statement that lights the LED when a button press is sensed
- UART function to send data to the TouchPro tool. This will aid with ‘tuning’ your button’s responsiveness

#include Section

25. Open main.c and add the header files required to use the CapTouch library.

The CTS_Layer.h file includes prototypes for all the CapTouch library functions, so we will need to include it. Also, we need to reference structure.h, since it provides visibility to the Sensor variable we defined in structure.c.

```c
#include "CTS_Layer.h" //Access the CapTouch Library
#include "structure.h" //Ref to Max # channels and Sensor variable
```

#define Section

26. Add two items to the #define section of main.c.

We use the TUNE definition to determine whether or not to send data to the TouchPro GUI.

We’ve seen earlier that the TouchPro tool can handle multiple data channels; in our case, this means sending count data from multiple Elements. How many channels will we send? It’s common to set this equal to the total number of Elements in your system – which happens to be defined in structure.h (as TOTAL_NUMBER_OF_ELEMENTS).

```c
#define TUNE 1   //Turn off "tuning" by setting to "0"
#define NUMBER_OF_CHANNELS_TO_TUNE TOTAL_NUMBER_OF_ELEMENTS
```

In our example, what will be the value for total number of elements? ___________________

Global Variables

27. No global variables are needed.

Though, later we will add a local variable to the Timer1 ISR.
main() function

28. Add the following function calls to main(), after the call to UART_init().

These two functions call into the Cap Touch library to calculate/update the baseline capacitance for our Button sensor. The first call makes an initial measurement; the second makes two more measurements to ensure accuracy.

```c
TI_CAPT_Init_Baseline(&myButton);     //Calculate button's baseline capacitance
TI_CAPT_Update_Baseline(&myButton,2); //Update the baseline
```

That’s all the code required for the main() function. The rest of the code will be added to the Timer1 ISR.

Timer1 ISR

Three items need to be added to the Timer1 ISR.

29. First, declare an array to hold the timer count values.

These values will be passed, via the UART, to the TouchPro GUI. As such, we need the length of the array to equal the number of Elements (i.e. electrodes) we want to tune, which just so happens to be one of the #defines we created earlier in the file.

```c
uint16_t counts[NUMBER_OF_CHANNELS_TO_TUNE];  //Count values for TouchPro GUI
```

30. Use the CapTouch library to determine if the button is being pushed. If detected, light the LED.

We’ve already seen the code to turn the LED on/off. In fact, it’s in almost every lab in the workshop.

The TI_CAPT_Button() function call determines whether the button was pushed. By passing a pointer to the sensor, the Button function will count the oscillations over the button’s Gate Time. If this count exceeds the Threshold, it will return 1, otherwise it returns 0.

```c
if (TI_CAPT_Button(&myButton))       //If button press is detected
{
    P1OUT |= BIT0;                   //Light the LED
}
else                                  //or else
{
    P1OUT &= ~BIT0;                 //Turn off the LED
}
31. Add code to “Tune” the system.

If tuning is enabled, we want to send ‘count’ data to the TouchPro GUI. This requires two functions: (1) to get the count values; (2) send the data serially.

```c
if (TUNE)
{
  // Library call to measure timer counts due to capacitance
  TI_CAPT_Custom(&myButton, counts);

  // Send count value to TouchPro GUI
  UART_sendDataFrame(counts, NUMBER_OF_CHANNELS_TO_TUNE);
}
```

TI_CAPT_Custom() does not return a value, but rather updates the ‘counts’ array with timer count values obtained during the Gate Time.

**Note:** This is why the sensor data structure has the .baseOffset field; in essence, this field is an index into the ‘counts’ array. In other words, it lets the TI_CAPT_Custom() function know which position(s) to store count data within the ‘counts’ array.

The UART_sendDataFrame() function formats and sends the ‘counts’ array to the TouchPro GUI. Its arguments are the ‘counts’ array, as well as the length of that array.

**Hint:** If you had many different sensors that you were tuning, you could call the TI_CAPT_Custom() function for each sensor; after all the counts are collected, call the UART_sendDataFrame() function.
Build, Load and Run

32. **Click the Hammer button to build the program.**
    Fix any errors that pop up. Once you’ve had a clean build …

33. **Click the Debug button to start the debugger and load the program to your MSP430.**

34. **Run your program and observe the results.**

   Is the Green LED still flashing on/off about once per second? ________________________

   Describe how the BoosterPack’s middle LED. Does it light up? ______________________

   Does the middle LED ever turn off? _____________________________________________

   Why does the LED behave this way? ____________________________________________

   __________________________________________________________________________

   In fact, can you see the effects of your finger pressing the middle CapTouch button? ______

   __________________________________________________________________________
Tuning the Button

35. With the program still running under CCS, launch the TouchPro GUI tool. Then, select the COM port associated with your MSP430 Launchpad.

If you need a hint for how to do this, refer back to Lab10b – Touch Pro GUI Tool ‘Wheel’ Demo, starting on step 4 (page 10-42). Hopefully you will not need to refer back to the Troubleshooting Suggestions (on page 10-42).

Once your program is running and connected to the TouchPro Tool, it should start displaying count values. Here’s a screen capture of our system, pressing the button three times:

![CapTouch Display](image)

OK, so what can we learn from visualizing the data in this way?
- First, the button appears to be working, even if the LED is always turned on.
- The LED is always on since we have the Threshold set at “0”. As you can see, the count value is always above 0; hence, the Button function always returns “1”.
- There is only one channel of data … which is what we expected, since we are only using a single sensor, which has just one element.
- In our example, there appears to be some periodic noise (maybe the flashing LED?).
- Try different types of button presses to get a feel for how they show up in the GUI.
- Remember, if you want to re-view this data later, use the File menu to save it; then you can re-load the count data later on.
Threshold

36. Pick a Threshold value.

You want to pick a threshold that is high enough above the noise so that it doesn’t trigger erroneously, but low enough not to miss any actual touches.

Based on our results above, we’re going to pick 100. (If we wanted to allow lighter touches, maybe 80 would be better. But, for now we’ll stick with 100.)

What’s your touch number? ___________________________________________________

In other words, what Threshold value are you going to choose?

37. Now we can finalize the code and set the threshold.

Terminate the Debug session and return to editing.

In structure.c, change the value for Threshold, replacing 0 with your chosen value.

#define THRESHOLD 100

38. Build, Load and Run your code again.

Does the LED light when you push the button? ________________________________

Do you like the responsiveness and feel of your button? ___________________________

If you’re not happy with how your button works, repeat the steps of opening up the TouchPro Tool (if you don’t still have it open). Try experimenting further with how hard you press and what count value is displayed.

Hint: You can use the mouse to select nodes on the graph to get see its exact count value. This is especially helpful if your tool is displaying multiple channels of data.

Make sure you are looking at the Y value at the bottom of the graph. That’s the value which displays the count data.

39. Terminate the active debug session using the Terminate button, then close the Lab10c project.

Ideas for Further Exploration

Feel free to experiment with some other choices that can affect your capacitive sensing designs.

- Try different Threshold values.
- Slow down and/or speed up the Scan rate (by changing the Timer1 configuration).
- Trying increasing and decreasing the Gate Time. (BTW, what happens to the count values when you increase the Gate Time?)
- Give another HAL a try – different timers, pin oscillator, and/or measurement methods.
- Try adding more sensors/elements to your program.
- Try modifying the DCO and VLO system clocks. (Note that this also affects both of your timers.)
- Checking the power is a little problematic unless you have an oscilloscope since the code spends the majority of its time in LPM3. But this is something else to experiment with. It is interesting to see how Gate and Scan times affect power usage.
(Optional) Lab10d Using Grace to Configure Clocks & Timer

In lab 10c, we cheated and used the Grace tool. It conveniently wrote the code required to configure our clocks and our Scan Rate timer. Of course, this isn’t really cheating – in fact, this is one of the reasons TI created Grace – to make MSP430 programming easier.

While Grace is covered in quite a bit more detail in Chapter 8, we wanted to walk thru the steps we used to create ‘our’ code.

What Is the Goal?

The goal is to implement our application as we described in our Planning Worksheet.

In our case, the Planning Worksheet stated:

- Run the high-speed clocks (MCLK, SMCLK) off of the DCO clock source, running at 1 MHz.
- The planning worksheet also states that we want to scan our buttons at a 20 Hz (50ms) rate. (We end up using the VLO clock for this slow rate.)

Create Project and Use Grace

1. Create a new CCS project using the “Grace” template.
2. Open the “Device Overview” in the Grace configuration file.

When your Grace-enabled project opens, it should default to opening in the Grace GUI. It should look like this. (If not, look for – and open – the file named main.cfg.)

Click on the “Device Overview” button to get a graphical layout of your MSP430 device:

The numbers indicate which modules (and the order) we will configure them in.

To begin with, on this screen, go ahead and set the voltage correctly.
3. Configure the clocks – BCS+ (Basic Clock System+).

Click on the blue box – Oscillators Basic Clock System+, which opens its configuration panel.

When using any of these modules, first of all, make sure the **Enable** is selected. Then click on **Basic User** to view:
4. **BCS+ Power User.**

At first glance, the default clock options looked like they would match our needs, but to obtain a 1ms gate time, we need to get SMCLK running at half of its current speed.

To do this, click the **Power User** button.

As you can see, by changing the SMCLK Divider, we can now get it down to 500 kHz.

![Configure Clock Source](image1)

5. **Move to the Watchdog+ Timer module.**

To do this, click on the **Grace** tab (lower-left of editing window), which will get you back to the **Device Overview**, where you can easily pick another module.

![External Digital Source](image2)

Once you’re in the **Watchdog+** module, click on the **Basic User** view.
6. Configure the Watchdog Timer as an interval timer running with a 1.024 ms period.

![Grace (MSP430) - WDT+ - Basic User Mode](image)

From our Planning Worksheet, our goal was to get the WDT+ timer down to a 1ms time period. So working back from there, we chose the 500 kHz SMCLK, along with the 512 Divider.

**Hint:** Funny, though, but we are not actually going to use this code – or even program the WDT+ timer ourselves. Because we're using the RO_PINOSC_T0_WDTp version of the HAL, the library functions will handle programming the Gate Timer for us.

So, why did we go through this exercise?

We used to scratch our heads and use calculators or spreadsheets to figure out how to set the clocks and timers to the rates we needed. Nowadays, Grace makes this much easier for the devices it supports.

When we began this exercise, we didn’t know which clock to use, or how to set its dividers. In just a couple minutes, though, Grace helped us figure that out … and, it even writes the code for us.

7. Double-check your Gate Time settings in `structure.c`.

The clock information above, together with our selections in our `structure.c`, will provide us with the correct gate timing.

```c
// Timer Information
.measGateSource = GATE_WDT_SMCLK,
//.accumulationCycles = WDTp_GATE_32768
//.accumulationCycles = WDTp_GATE_3192
.accumulationCycles = WDTp_GATE_512
//.accumulationCycles = WDTp_GATE_64
```

8. Finally, navigate your way over to the Timer1_A3 module.

9. Enable Timer1_A3 and go to the Basic User view.

   Once again, taking our cue from the Planning Worksheet, we wanted a 20 Hz Scan Rate.

   Selecting Interval Mode and plugging in the 50ms desired Time Period easily provides us our
   chosen 20 Hz clock rate.

   **Hint:** Why did we choose Timer1?

   Timer1 was a convenient choice as it wasn’t already being used in the application.
   Due to our HAL selection, the CapTouch library was going to be using WDT+ and
   Timer0. They only use these peripherals while running one of the TI_CAPT_ function
   calls – which means we could have still used one of them to do our Scan Rate timing.

   Even though it could be done, it was much easier utilizing another, available Timer.
Accessing the Code From Grace

10. Build your Grace project.

To gain access to the code created by Grace, it first needs to be generated; this is done by building the project. Go ahead and click the:

Hammer toolbar icon

When the build finishes, notice that a new “src” folder was added to the project.

As you can see above, here are the two files we used in our previous lab exercise.

11. Grab these files and try them in your previous project, if you want to test them out.

You could even use Grace to regenerate these files using different clock rates and such. This would be an easy way to experiment with the previous lab exercise.
# Capacitive Touch/Sensing Planning Worksheet

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Quantity</th>
<th># of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sliders</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wheels</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Proximity</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Total # Elements**: 1

## Variable Names (for Sensors & Elements)

List the variable Names for Each Sensor and Element you will be using. For example, if creating a “wheel” sensor that contains 4 elements, the variables might look like:

```
myWheel_Sensor ........................ left_element Port 2 Bit 1
........................ up_element Port 2 Bit 4
........................ right_element Port 2 Bit 3
........................ down_element Port 2 Bit 2
```

<table>
<thead>
<tr>
<th>Sensor Name</th>
<th>Element Name</th>
<th>GPIO Port</th>
<th>PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>myButton</td>
<td>myButton_element</td>
<td>Port 2</td>
<td>5</td>
</tr>
</tbody>
</table>

---

Planning Worksheet

Page 1 of 2
Selection Device & HAL

Please select the MSP430 device that you plan to use in your system. Remember, there are devices, such as the MSP430G2553 (used on the Value-Line Launchpad) that have hardware dedicated to Capacitive Sensing.

You should also select a HAL (hardware abstraction layer) from the TI Capacitive Touch Library. The library documentation recommends specific HAL choices for various devices (as shown to the right).

<table>
<thead>
<tr>
<th>MSP430 Device</th>
<th>MS430G2553</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>RO_PINOSC_TA0_WDTp</td>
</tr>
<tr>
<td>Capacitive Touch Library</td>
<td>Hardware Abstraction Layer</td>
</tr>
</tbody>
</table>

Select Initial Gate/Scan Rates ... Run the Power Design Tool

Enter the following values. The timing values under the gate times (and next to the scan rate) are good initial suggestions. We recommend using a 'pencil' for the timing values, since you may want to change them during the Tuning phase of development.

![System Parameters Table]