

## Techniques for Robust Touch Sensing Design

*Author: Burke Davison  
Microchip Technology Inc.*

### INTRODUCTION

The purpose of this application note is to describe the best design practices when developing capacitive touch applications in noisy environments. This application note will begin by defining the problems caused by noise, and explain how that noise typically affects systems. Hardware guidelines will then be provided to help maximize the natural signal-to-noise ratio (SNR) of the application. Software techniques are then covered to describe some of the common methods used to filter a sensor's signal to increase the SNR further, and then to make a decoding decision based on the behavior of the capacitive sensor.

The hardware design topics that will be covered are:

1. Selecting a sensor size
2. Determining the sensors' separation
3. Covering material thickness
4. Using ground planes to your advantage
5. Designing the sensors' layout
6. Selecting an adhesive
7. Using series resistance on sensors
8. Choosing VDD to maximize noise immunity

mTouch™ sensing solution systems have passed industry test standards in conducted noise, radiated noise, and radiated susceptibility. This application note describes the important aspects of capacitive touch design which, when coupled with good PCB techniques, will allow these systems to continue performing in these extreme testing conditions.

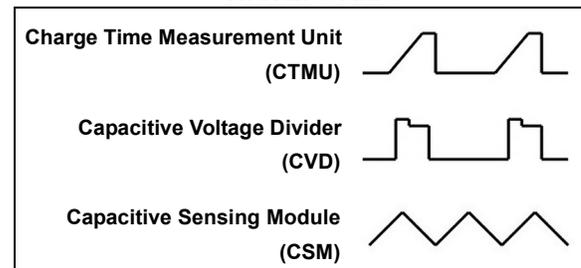
For information on the basics of capacitive touch sensing and other more advanced topics, visit the Microchip web site at <http://www.microchip.com/mTouch>.

### Basic Capacitive Touch Review

Capacitive sensors are areas on a PCB that have been filled with copper and then connected back to the PIC® device using a trace. The PIC device will then measure the sensor in some manner that allows it to notice small shifts in capacitance. The capacitance is continuously read in software and when a change occurs, the system will register a press on that sensor.

There are two main methods for detecting a shift in capacitance using a PIC device. The first is to use a voltage measurement where the system places a specific known voltage onto the sensor and looks for a shift in the amount of voltage on the sensor. This includes methods such as Microchip's Charge Time Measurement Unit (CTMU) and the Capacitive Voltage Divider (CVD). The alternative is to measure the sensor using a frequency approach such as the Capacitive Sensing Module (CSM), which uses a fixed current source and a comparator to create a circuit that changes its frequency based on the capacitance seen at the sensor. The waveforms for all three scanning methods can be found below in Figure 1.

**FIGURE 1: mTOUCH™ SENSING ACQUISITION METHODS' WAVEFORMS**



This application note will focus on the hardware design of the system and the sections of firmware not involved in signal acquisition. With only a few exceptions, the signal processing portion of the firmware will be identical regardless of the sensing method chosen.

### Noise Immunity vs. Low Power

When developing a capacitive touch system, it is important to know what your main goal should be from the very start of product development. For the majority of applications, how the system is powered will answer this question.

For line-powered systems, conducted noise immunity is the main concern. Voltage-based acquisition methods should be used.

For battery-powered systems, low power is the main concern. Both voltage-based and frequency-based methods can be implemented in these applications.

# AN1334

It is also possible that some systems may overlap between these two regions. A cell phone that has the option of being powered through a USB cable is one example. The majority of the time, it would be concerned with low power; however, it needs to be careful of conducted noise when being powered through the main line. For this reason, only voltage-based acquisition methods should be used in these systems. While noise immunity and low power are not mutually exclusive, focusing on one will require that design trade-offs be made to the other. For example, implementing a slew rate limiter filter to reduce susceptibility to conducted noise will require increasing the sample rate of the system which will increase the overall power consumption. Lowering VDD is an excellent idea in low-power applications, but doing so will also decrease your noise immunity (see **Section “Hardware Design Consideration #8”**). This application note focuses on decreasing noise susceptibility and treats low power as a secondary goal. If low power is the main goal of the application, visit [www.microchip.com/XLP](http://www.microchip.com/XLP) for more technical details.

## EFFECTS OF NOISE ON CAPACITIVE TOUCH SENSORS

### Mechanical Buttons vs. Capacitive Sensors

Before considering how to develop a robust capacitive touch application, it is important to understand the fundamental reason why noise is a concern. When using a mechanical button, the microcontroller's port circuitry decides whether the switch's pin is being pulled high or low and provides a single-bit digital result to the user. This result is then debounced to adjust for ringing, and the state of the button is based on the state of the debounce variable.

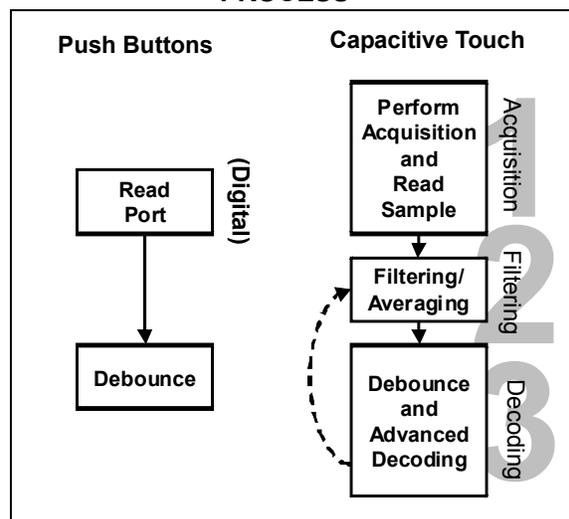
Capacitive touch sensor applications, however, are analog. The first clear difference is the need to manually perform the reading process. When using a mechanical switch, the microcontroller is able to read the pin using its internal hardware logic. For capacitive touch applications, separate hardware modules will need to be used to manipulate the sensor line. Whether it is using a voltage-based measurement or a frequency-based measurement, the analog result will be provided in the form of an integer value. This value is then typically filtered using different digital signal processing techniques to amplify the signal and attenuate the noise. The filter value is then sent through some form of debounce algorithm and a more complex decoding process. An extra layer of complexity is also added when the system is designed to perform in a closed loop manner, adjusting its behavior based on the sensor's current state.

The capacitive touch software process can be simplified into three distinct phases:

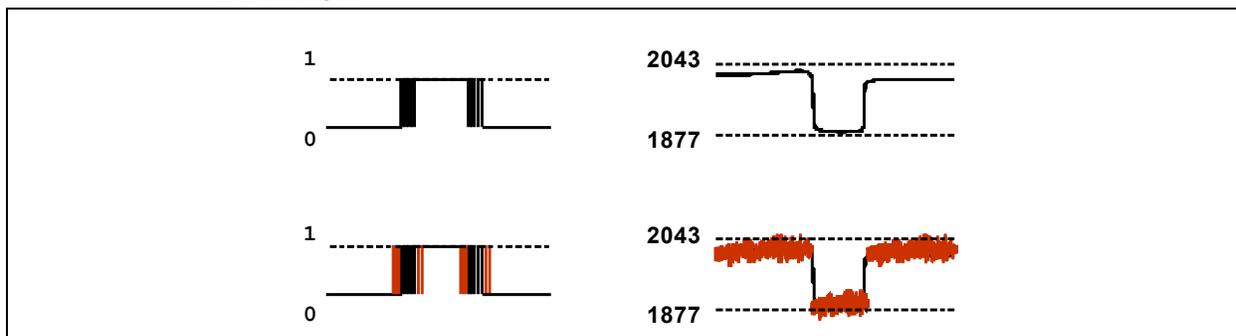
1. **Acquisition**  
Using a voltage-based or frequency-based measurement technique to obtain a sample from the capacitive touch sensor.
2. **Filtering**  
Manipulating the incoming sensor samples to increase the effective SNR of the system by attenuating the noise.
3. **Decoding**  
Determining whether a sensor is pressed or released based on the current value of the sensor samples and the sensor's previous behavior.

Figure 2(a) illustrates the difference between push buttons and capacitive touch sensors, and labels the three main software stages of a capacitive touch system.

**FIGURE 2(A): MECHANICAL BUTTON VS. CAPACITIVE TOUCH SENSOR SOFTWARE PROCESS**



The difference between digital and analog results can also be seen in Figure 2(b). The push button is always in one of three states: high, low, or ringing from a recent transition. The capacitive touch sensor does not perform in the same way due to its analog result. Instead, it is able to drift and move. When noise is injected on the sensor, it affects the quality of the readings, not just the time required to make a state transition.

**FIGURE 2(B): MECHANICAL BUTTON VS. CAPACITIVE TOUCH SENSOR SIGNAL NOISE BEHAVIOR**

### Conducted and Radiated Noise

Conducted and radiated noise are the two main classifications of injected noise that can create instability in capacitive touch systems.

Conducted noise is caused in systems that are powered externally from the device. This can include systems powered off the main-line power, desktop-powered USB devices, or any other situation that may mean the user is not sharing a ground with the device.

Radiated noise is a common challenge across all capacitive touch systems. Since the capacitive touch sensor is a high-impedance input when being scanned, it essentially performs as a high-frequency antenna. Thus, electronic devices radiating electro-magnetic fields near the capacitive touch system will cause the readings to be affected. This can include cell phones, high-power communication lines, and fluorescent lights to name a few.

There are two main reasons why these two types of noise show up:

1. When a user presses on a capacitive touch sensor, he/she is becoming part of the system, so, if the user and the system are on different ground references, the system will interpret the user as an injected AC signal on the sensor.
2. Analog readings are susceptible to outside forces pushing them slightly in one direction or the other. The digital result of a mechanical switch is either high or low.

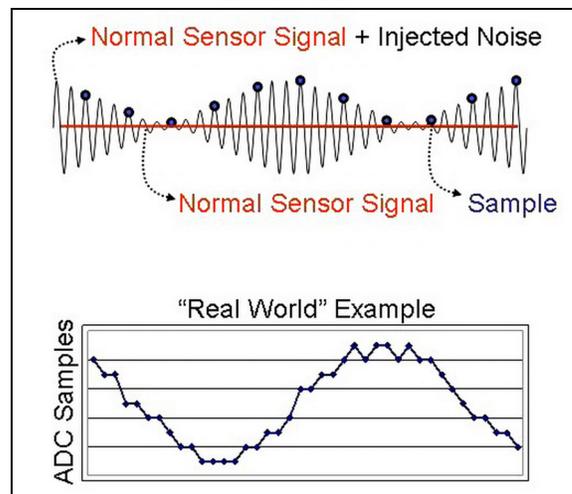
This application note will describe the different system design techniques that are recommended to overcome these two noise types. In addition to these guidelines, designers should be aware of the future working environment of the application and ensure there are no excessively noisy electronics nearby that may interfere with the system.

### Capacitive Touch Sensor Noise Behavior

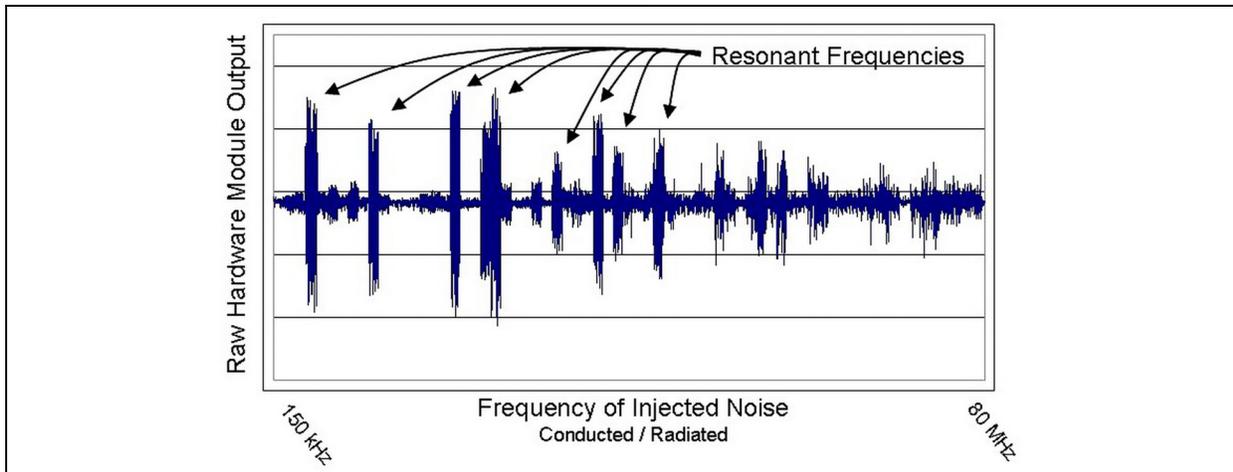
Injecting noise on a capacitive touch sensor will cause the system to become more unstable. Voltage-based mTouch sensing solution reading methods such as

CTMU and CVD will be affected differently than frequency-based reading methods such as CSM. In voltage-based systems, the voltage of the sensor at a specific point in time is what determines the integer value of the reading. In frequency-based systems, the effect on the readings will vary based on the frequency of the injected noise. For this reason, voltage-based acquisition systems should be used on any system concerned with conducted or radiated noise immunity.

In voltage-based systems, injected noise can cause a positive or negative offset away from the natural sample value. If the sampling rate falls on a harmonic of the injected noise, resonance can occur. When this happens, the samples are falling on the peaks or valleys of the injected noise. An example of this behavior can be seen in Figure 3. When sampling at one of these harmonics, the readings may all fall on the peaks of the noise or somewhere in the middle based on the starting time of the acquisition. Because of this, multiple readings at the same frequency will show a large amount of noise. This can be seen in Figure 4 where some of the noise frequencies are harmonics of the sampling rate and others are not.

**FIGURE 3: VOLTAGE-BASED HARMONIC ACQUISITION EXAMPLE**

**FIGURE 4: EXAMPLE VOLTAGE-BASED-ACQUISITION NOISE BEHAVIOR**



This can be seen in Figure 4, where some of the noise frequencies are harmonics of the sampling rate and others are not.

Given the knowledge of how the system will behave, some important hardware considerations that make development a faster and simpler process can now be defined.

## Signal-to-Noise Ratio

First, in order to understand how hardware and software changes are affecting the system, it is necessary to have a way of measuring the signal's current performance. Sensitivity, or the amount that the system shifts, is not a good enough measure alone to define if a system is stable. For example, in a system where the average sensor output value is 20000 and a shift of 2000 is reached, you could simply subtract 18000 from each reading and claim a 100%, 2000 count shift was achieved. In reality, however, the shift or "signal" must be compared with the amount of noise. If noise was causing the sensor to drift by 1000 counts at any point in time, the system is in trouble.

One of the easiest ways to determine how stable a system is, or how much the system is affected by noise, is to look at its Signal-to-Noise Ratio (SNR). Just as it sounds, this is a way of measuring how strong the signal is when compared to unwanted disturbances of noise.

For the purpose of this application note, the SNR formula being used is:

### EQUATION 1: SIGNAL-TO-NOISE RATIO

$$SNR = \frac{\mu_U - \mu_P}{\sigma_U}$$

Where:

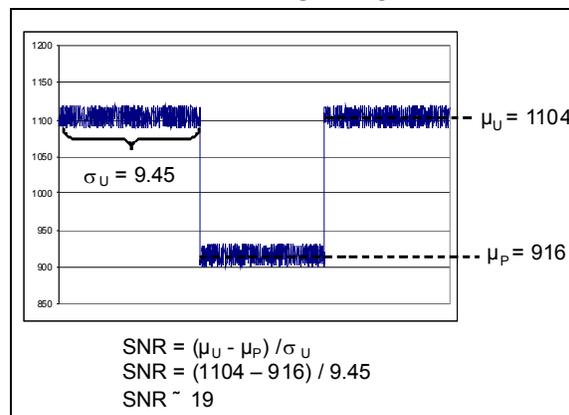
$\mu_U$  is the unpressed average

$\mu_P$  is the pressed average

$\sigma$  is the unpressed standard deviation

The numerator of the equation is the amount that the system will shift when pressed. The denominator is a measure of how much the noise is able to affect the readings. Using these as a ratio, a single number can be used to describe the quality of the sensor's signal by answering the question: How much shift are you looking for compared to the amount of noise you are trying to avoid? An example SNR calculation is shown in Figure 5.

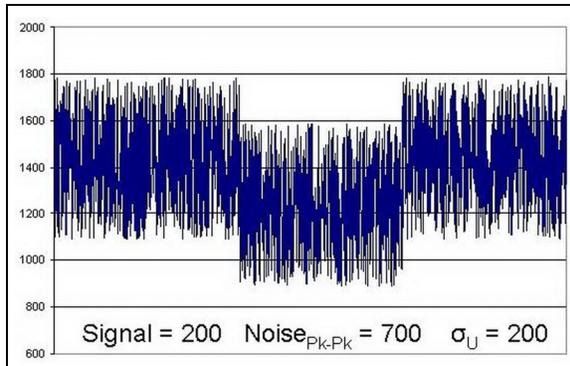
**FIGURE 5: EXAMPLE SNR CALCULATION**



There are many other ways to calculate the SNR of a system. The important thing is to choose a method that provides consistent numbers across multiple measurements so informed decisions can be made about which of the changes made are good and which are bad.

For reference, Figure 6 is provided to show an example of what a signal-to-noise ratio of 1 would look like using Equation 1. Note that since the standard deviation of the noise and not the peak-to-peak value is being used, an SNR of 1 leaves no safe place to put a threshold. To be able to place a fixed threshold on the system so that the pressed section plus its noise is completely separated from the unpressed section plus its noise, a system with an SNR of at least 3.5 is needed.

**FIGURE 6: EXAMPLE: SIGNAL-TO-NOISE RATIO = 1.0**



## HARDWARE DESIGN

The hardware design of a capacitive touch application is crucial to the system's overall success. The decisions made in this step of the process will determine how difficult it is to get a working, robust application. If the hardware design guidelines are followed, it will be significantly easier and faster to pass industry noise standards. Likewise, not following the guidelines will make success much more difficult and in some cases impossible. Keep this in mind while deciding which of these guidelines to follow in your future applications.

### The Basic Capacitance Equation

The most important thing about hardware design is to remember that the basic capacitance equation, shown in Equation 2, defines the relationship between hardware design decisions and the resulting sensitivity of the system.

### EQUATION 2: CAPACITANCE / SENSITIVITY

$$C = \epsilon_r \epsilon_0 \frac{A}{d}$$

Where:

C is the capacitance, or sensitivity

$\epsilon_r$  is the relative permittivity of the cover

$\epsilon_0$  is the permittivity of free space

A is the overlapping area

d is the distance

For example, if the distance between the finger and the sensor is decreased by half, the sensitivity will double. If the area of the sensor is doubled (assuming it is still smaller than the area of a finger's press) then the sensitivity will also double.

Another important characteristic of capacitive touch sensors is the existence of parasitic capacitance,  $C_P$ . Equation 3 explains how  $C_P$  can affect a system's sensitivity. You are only able to take a measurement of the total capacitance on the sensor,  $C_{TOT}$ , so the stronger the effect of  $C_P$  the less you may be able to see  $C_F$ , the change in capacitance due to a finger. This relationship is shown in Figure 7.

### EQUATION 3: TOTAL CAPACITANCE

$$C_{TOT} = C_P + C_F$$

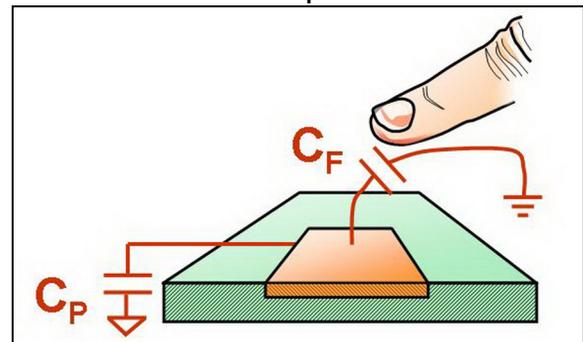
Where:

$C_{TOT}$  is the total capacitance

$C_P$  is the parasitic capacitance

$C_F$  is the finger's capacitance

**FIGURE 7: DIAGRAM SHOWING  $C_F$  AND  $C_P$**



Equation 2 and Equation 3 will be the basis of the hardware design guidelines for capacitive touch. The equations are simply physics. The guidelines are recommendations that will attempt to maximize your system's base SNR and should be followed whenever possible. In some cases, an application may require that some of the guidelines not be followed. For example, a system might have size constraints or may require a thick covering material to protect it from damage. If this is the case, extra care should be taken to ensure a quality signal-to-noise ratio.

## Hardware Design Consideration #1

### SELECTING A SENSOR SIZE

**Best Option:** The sensor size should be the same as an average user's finger press (15x15 mm or 0.6x0.6 inch).

**Option 2:** Design sensors to be smaller than optimal.

Effects:

- Overlapping area, 'A' in Equation 2, is limited which reduces the maximum sensitivity.
- Adequate sensor separation will become more important to minimize the amount of sensor crosstalk.
- Use a thin cover to gain some extra sensitivity.

**Option 3:** Design sensors to be larger than optimal.

Effects:

- Parasitic capacitance, 'C<sub>P</sub>' in Equation 3, can increase because of the increased proximity to ground which reduces sensitivity.
- Conducted noise disturbance is increased.
- Press shifts will vary by larger degrees because small fingers will cause less of a shift than large fingers due to less overlapping area, 'A' in Equation 2.
- Proximity sensing capability is increased.

In Equation 2, 'A' is defined as the overlapping area. For capacitive touch applications, this means that you are limited by the smallest capacitive plate. If the sensor is smaller than a finger's press, the sensor's area is the limiting factor. If the sensor is larger than a finger's press, the finger is now the limiting factor.

You cannot change the user's finger size, but you can adjust the sensor size to maximize the sensitivity. The larger the sensor, the more parasitic capacitance will be able to lower the sensitivity and the more conducted noise will be injected into the system when a user presses. The smaller the sensor, the greater the chance that it is the limiting factor on sensitivity instead of the user's finger size. For this reason, the ideal sensor size is about the area of a finger press.

Some exceptions to this rule do exist. Thick covers will lower the sensitivity of a sensor drastically, so slightly larger sensors may be warranted. There are also situations where the user's finger is not what is trying to be detected. For instance, if a proximity sensor is being made and you are only interested in knowing when a user's hand is near the sensor, then a much larger sensor could be used.

## Hardware Design Consideration #2

### DETERMINING THE SENSORS' SEPARATION

**Best Option:** Separate sensors as much as possible. Ideal minimum separation is 2-3 times the cover's thickness.

Effects:

- The distance, 'd' in Equation 2, between the sensors is kept high compared to the distance between the finger and the sensor, which results in reduced sensor crosstalk.
- Parasitic capacitance, 'C<sub>P</sub>' in Equation 3, is kept low compared to the finger's capacitance, 'C<sub>F</sub>', which results in increased sensitivity.

**Option 2:** Create slotted air gaps in the cover.

Effects:

- The relative permittivity, 'ε<sub>r</sub>' in Equation 2, between the sensors is lowered to "1", which results in decreased coupling between the sensors which decreases sensor crosstalk.

**Option 3:** Use ground traces between the sensors.

Effects:

- The distance, 'd' in Equation 2, between the sensor and ground is lowered, which increases the amount of coupling between sensor and ground, which results in a slight shielding effect and reduces the crosstalk between sensors separated by the ground.
- However, this increases the parasitic capacitance, 'C<sub>P</sub>' in Equation 3, which will reduce sensitivity.

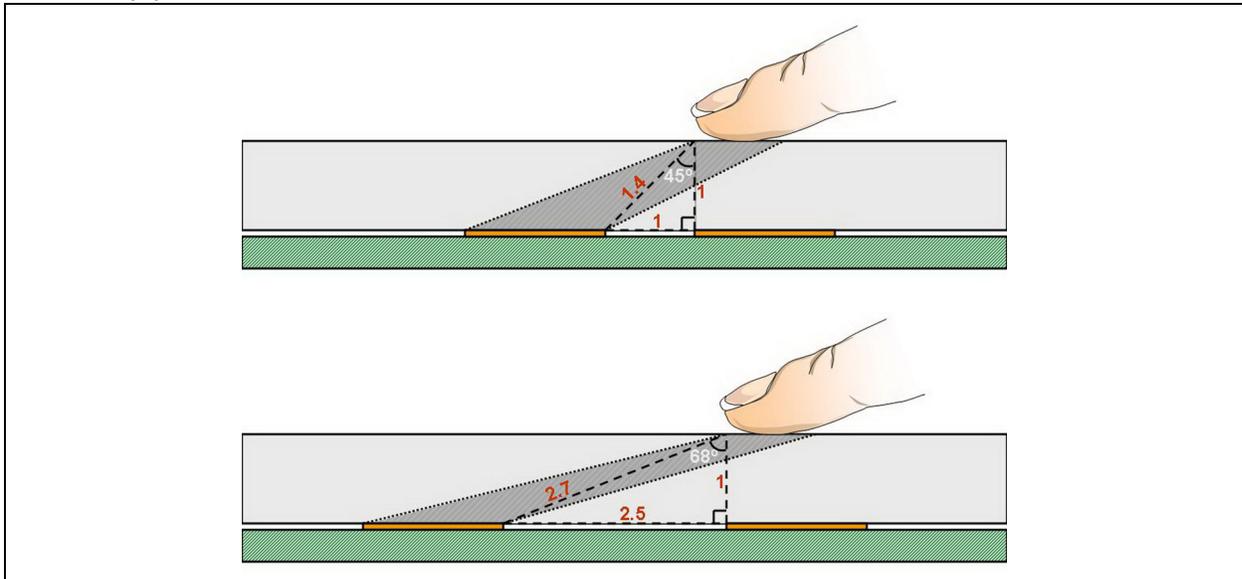
Crosstalk can become a challenge in capacitive touch applications if covers are too thick or sensors are too closely placed together. Crosstalk is the unwanted shift of a different sensor from the one you are intending to press. If crosstalk is a problem in an application, the software must compare the two sensors and determine which sensor is "more pressed". This adds an extra step to the decoding process, increases the likelihood of error and, (depending on how it is implemented) can limit your system to one touch at a time. To include multi-touch in a "Most Pressed" system, the algorithm can be changed to be a "two most pressed" configuration or one of the sensors that will be pressed can be physically separated from the others similar to a

computer's shift key. Following this guideline will allow you to avoid implementing a system that must compare each sensor with every other sensor.

Figure 8(a) shows how a finger's press can affect the sensors located around the target sensor. By separating the sensors by 2-3 times the cover's thickness, the strength of the finger-to-sensor coupling is limited to a low and manageable amount. An alternative way of thinking about this relationship is to focus on the distance variable, 'd', in Equation 2. If the

sensors are separated by the same amount that the cover is thick, a press on one sensor will be like pressing the other sensor through a cover that is 1.4 times thicker. By separating the sensors as shown in Figure 8(a) by 2.5 times the covers thickness, that crosstalk press is now equivalent to pressing through a cover that is 2.7 times the cover's thickness. This results in a much more decreased crosstalk response from the system which, in turn, increases your effective signal.

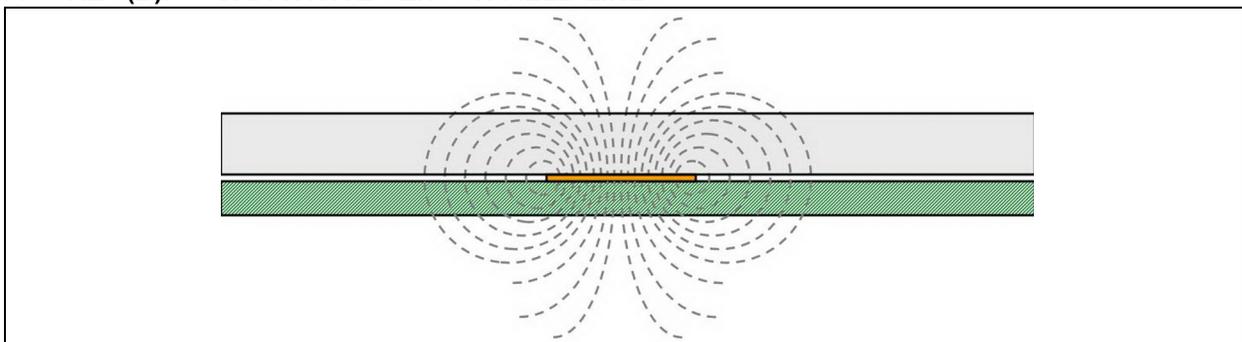
**FIGURE 8(A): DIAGRAM OF FINGER-TO-SENSOR COUPLING**



Sensor-to-sensor coupling is the other form of crosstalk that can negatively impact a design. Figure 8(b) shows how field lines will radiate from a capacitive sensor. The

ability of those field lines to affect a neighboring sensor is based on the distance it travels and the material it is travelling through.

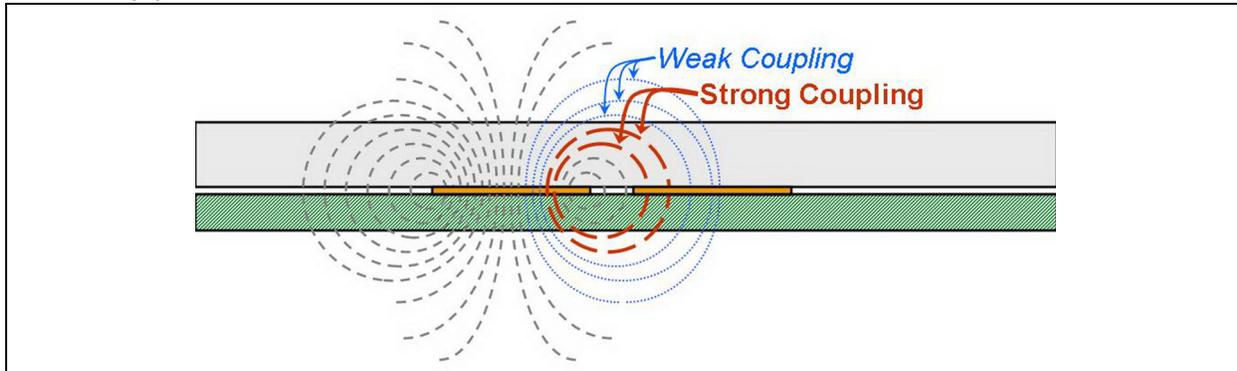
**FIGURE 8(B): CAPACITIVE SENSOR FIELD LINES**



If the field lines are able to propagate only through the cover as shown in Figure 8(c), the effect will be strong. If the field lines must go through the cover, exit into free space, and then return through the cover in order to affect a neighboring sensor. The amount of crosstalk will be significantly reduced. This is shown in the figure as the difference between the strong and weak coupling field lines. By following the first hardware design guideline, the field lines will be forced to travel

through free space to reach a neighboring sensor and so the crosstalk caused by sensor-to-sensor coupling will be insignificant.

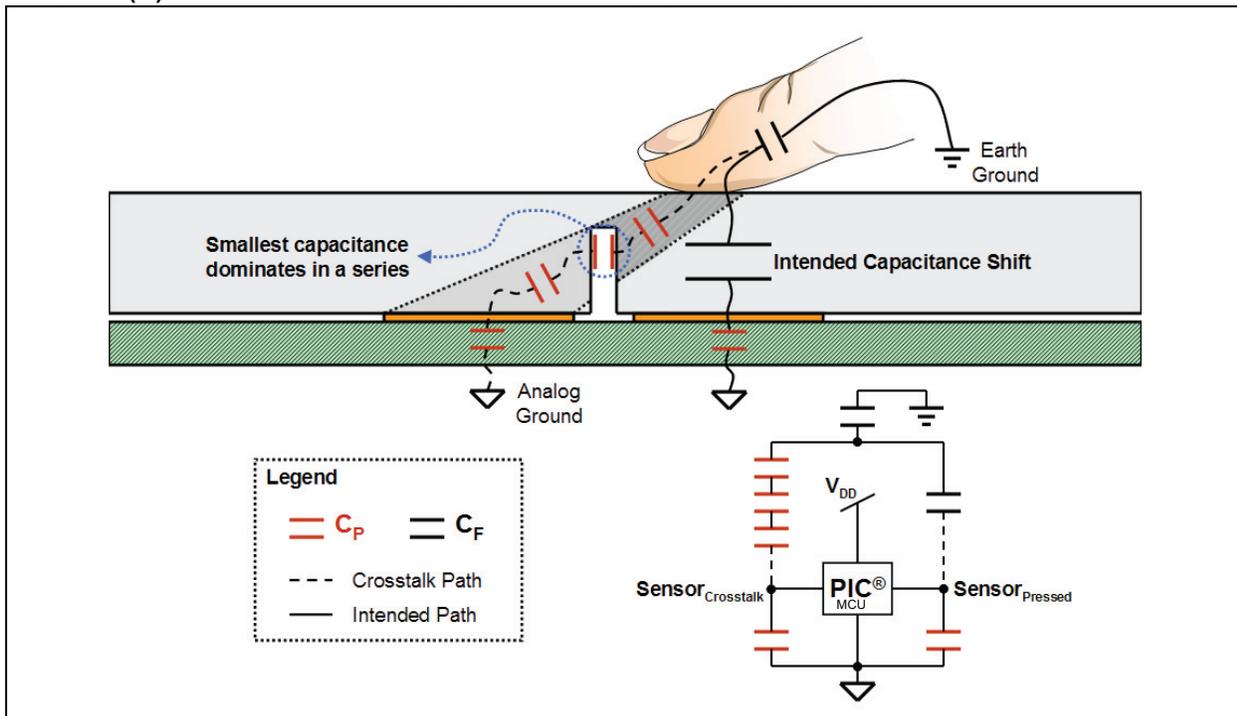
**FIGURE 8(C): DIAGRAM OF FIELD LINES SHOWING SENSOR-TO-SENSOR CROSSTALK**



Alternatively, air gaps could be placed in the cover or PCB that will require the field lines to travel through free space. Figure 8(d) illustrates this possibility. Notice how the crosstalk path now has a very small capacitor in series with the normal parasitic capacitances. This

small capacitor will dominate the others and will result in an overall crosstalk shift that is very low. The larger the air gap, the smaller the capacitor, and the better this method will perform.

**FIGURE 8(D): DIAGRAM OF CROSSTALK IN A SLOTTED COVER SYSTEM**



Finally, another option is to limit the sensitivity of the sensor by using nearby ground traces to block the field lines. Before using this technique, review **Section “Hardware Design Consideration #5”** to understand the recommended use of ground near sensors. Reducing the sensitivity of a system should not be a design decision that is made lightly and should only be used when the other possibilities have been exhausted.

By following this hardware design guideline, your designs will have reduced finger-to-sensor coupling and sensor-to-sensor coupling. This will result in a system that sees very little crosstalk which will allow the response time to speed up due to decreased processing overhead and the reliability of the system will increase as the sensors' signals become more immune to these negative effects.

## Hardware Design Consideration #3

### COVERING MATERIAL THICKNESS

**Best Option:** Keep the cover as thin as possible. Ideally, given 15x15 mm sensors, the cover thickness should not exceed 3 mm to maximize sensitivity. See Figure 8(a)

**Option 2:** Cover is thicker than optimal, but the sensors' areas are increased to provide additional sensitivity.

See Figure 8(b).

**Option 3:** Cover is thicker than optimal, but slots in the covering material are created to allow the sensor closer to the surface.

See Figure 8(b).

**Option 4:** Cover is thicker than optimal, but slots in the covering material are created to allow EMI gaskets or springs to bridge the gap between the PCB and the finger.

See Figure 8(b).

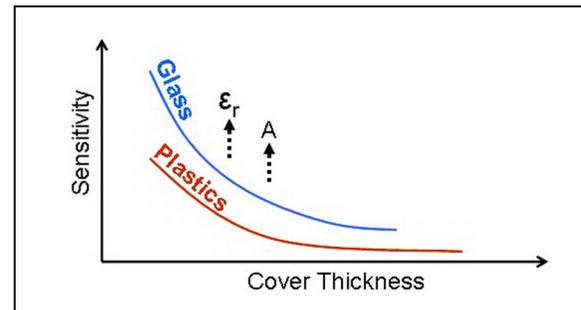
When covers are thicker than optimal:

- The distance, 'd' in Equation 2, between the sensor and the user's finger increases, which causes the capacitance between the finger and the sensor, ' $C_F$ ' in Equation 3, to decrease which results in decreased sensitivity.
- Sensor-to-sensor crosstalk increases as shown in Figure 8(c) due to additional sensor field lines being able to travel through the high-permittivity cover compared to the low-permittivity air.

The thickness of the covering material is very important in affecting the sensitivity of capacitive touch systems. Product designers will usually try to make the covering material as thick as they can to increase the durability of the end product, but thick covers will decrease the system's sensitivity extremely fast. Equation 2 helps to explain why thick covers are such a concern for capacitive touch applications. As the distance between the PCB and the finger is increased, the expected capacitance shift is decreased.

The relationship between cover thickness and sensitivity can be seen in Figure 9. One thing to note about the graph is the importance that the permittivity of the material plays in defining the curves. A high permittivity material will allow for a larger sensitivity shift than an equally thick, but lower permittivity material. High permittivity has the negative effect of increasing the amount of crosstalk, however. Figure 8(a) shows the effect that a finger can have on a neighboring sensor. As the cover's permittivity increases, so does this coupling.

**FIGURE 9: RELATIONSHIP BETWEEN COVER THICKNESS AND SENSITIVITY**



If your application absolutely requires a thick covering material for some reason, consider creating a slot in the covering material where the sensor will be so the sensor can be placed closer to the user's finger. Conductive foam products are also available that can be used to fill the gap if the whole PCB cannot fit in the slot. For more information about avoiding air gaps between the PCB and the covering material, see Section "Selecting an Adhesive".

## Hardware Design Consideration #4

### USING GROUND PLANES TO YOUR ADVANTAGE

#### Ground planes on the front of the PCB:

- Increases human-to-electrical-ground coupling, reducing the effect of conducted noise.
- Increases parasitic capacitance, ' $C_P$ ' in Equation 3, when placed near sensors which results in decreased sensitivity.

To minimize  $C_P$  and maximize conducted noise reduction, a sensor-to-ground-plane separation of 1-2 times the cover thickness is suggested. Solid ground planes are recommended to maximize the human-to-electrical-ground coupling.

#### Ground planes on the back of the PCB:

- Shields sensors from radiated emissions coming from behind the system.
- Increases parasitic capacitance, ' $C_P$ ' in Equation 3, when placed near or behind sensors which results in decreased sensitivity.

Unless shielding from a radiating source behind the system is required, not placing ground planes directly behind sensors is recommended to keep sensitivities high. If shielding is required but sensitivity is a concern, consider using a gridded ground plane.

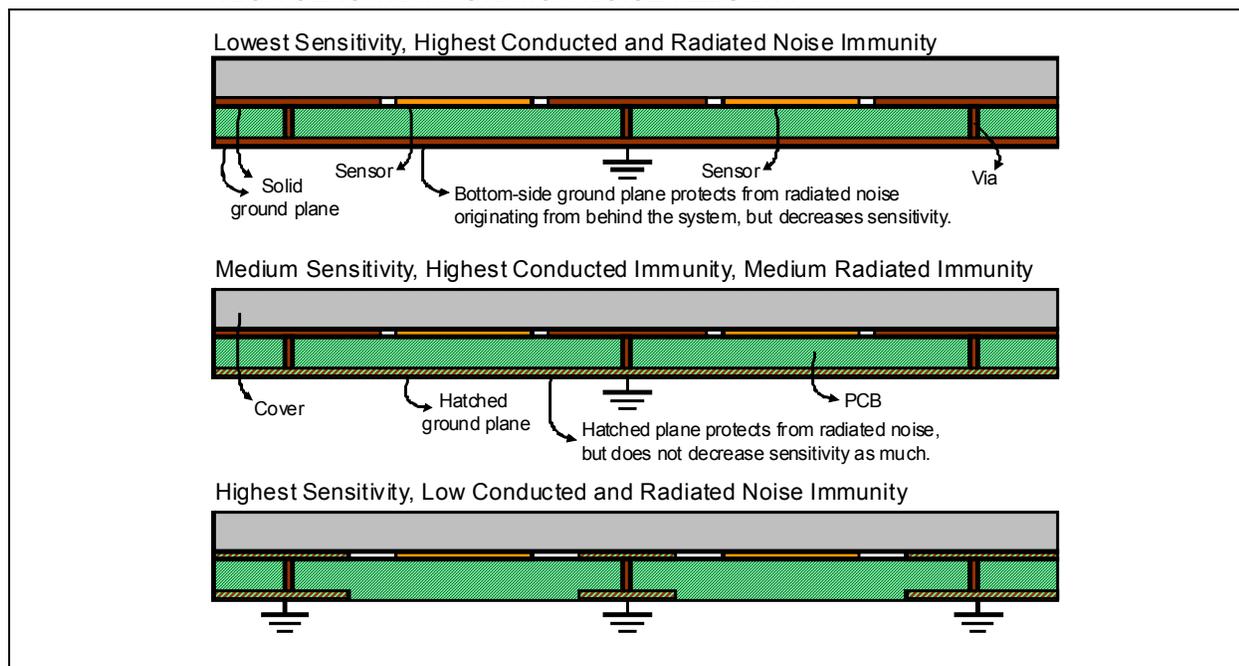
# AN1334

You can adjust  $C_P$  in Equation 3 using these techniques if you know that an increased amount of noise will be present near a sensor. The negative effects could become a challenge if the sensitivity of the system is suffering, but many times this allows the designer to anticipate problems and proactively correct them. Solid and gridded ground planes can be combined behind and to the sides of sensors. By using these tools effectively,  $C_P$  can be designed to find a balance between sensitivity and noise immunity.

Figure 10 demonstrates three levels of noise immunity that are possible using different grounding techniques around the sensors.

It is hard to stress enough the importance of providing your application with a sufficient forward-facing ground plane if conducted noise will be a concern. For systems that do not have a direct connection to earth ground, a finger can appear to be a noise source injected directly on top of a capacitive touch sensor. The extra ground planes will allow the finger to couple better to the floating ground, decreasing the amount of disturbance.

**FIGURE 10: CROSS-SECTIONAL DIAGRAM OF GROUNDING TECHNIQUES TO DESIGN FOR HIGH SENSITIVITY OR HIGH NOISE IMMUNITY**



## Hardware Design Consideration #5

Keep sensitivities high and noise low: Keep sensor trace lengths short.

### DESIGNING THE SENSORS' LAYOUT

*Best Option: Keep sensor traces thin and short.*

There are two main reasons to follow this advice. First, keeping trace lengths short will minimize  $C_P$  which will increase the sensitivity of the system. Long traces are also more susceptible to behaving like antennas which will increase the noise floor of the application. Communication lines should be kept away from sensor traces if at all possible. If not, run them perpendicular to the sensor traces to minimize their disturbance. You can also guard them with ground traces to couple the communication lines to ground instead of the more sensitive capacitive sensor traces. Avoid running capacitive sensor traces parallel with any noise-causing lines and keep them separated from ground and other capacitive sensor lines to reduce parasitic capacitance.

## Hardware Design Consideration #6

### SELECTING AN ADHESIVE

**Best Option:** Always use an appropriate adhesive.

Adhesive is used to secure the covering material to the PCB and is another important element to a robust capacitive touch system. Equation 2 will help to explain the necessity of a good connection. The relative permittivity of air is about 1. Plastics are usually between 2 and 3. Glass is about 4. If you have air separating the cover and PCB, your effective  $\epsilon_r$  will be significantly decreased. For example, a 1 mm air gap will decrease your sensitivity to a half or a quarter of what it was. Remember that when three capacitors are in series, the smallest will dominate.

For systems using the metal over capacitive technique, it is especially important that a good adhesive is found for the application. Distances of tens of microns (10 micron  $\approx$  0.4 mil) can make a significant difference in these designs. Talking to a representative from 3M or another adhesives manufacturer is recommended to ensure your choice is the best for your custom application.

There are several other important factors to keep in mind when choosing or working with a commercial adhesive:

1. Keep the adhesive thin in order to keep your sensitivity high. For most regular capacitive touch systems, 2 mil (50 micron) is a good thickness.
2. Always read the bonding instructions for the adhesive. Some data sheets specify a required amount of pressure, temperature, and time to achieve a secure, lasting grip.
3. Check the temperature limitations of your adhesive. In some environmental conditions, the glue can fail which will lead to unpredictable behavior from your capacitive touch application.
4. Be careful of air bubbles when applying the adhesive. If there are bubbles in the glue, your sensitivity will suffer the same as if you had an air gap between the cover and the PCB.
5. Make sure the adhesive type matches well with the covering material. Different adhesives are made for low surface energy and high surface energy plastics. Most adhesives will adhere to glass and PCB with few problems.

Some example adhesives that may perform well are:

**High Surface Energy Plastics:**

**Example:** ABS or Polycarbonate (PC)  
3M's Adhesive Transfer Tape 467MP

**Low Surface Energy Plastics:**

**Example:** Polypropylene (PP)

3M's Adhesive Transfer Tape 9626

3M's Adhesive Transfer Tape F-9752PC

3M's Adhesive Transfer Tape 9122

**3M's Optically Clear Adhesive (OCA):**

8211, 8212, 8213, 8214, 8215

All of these will adhere to PCB and glass.

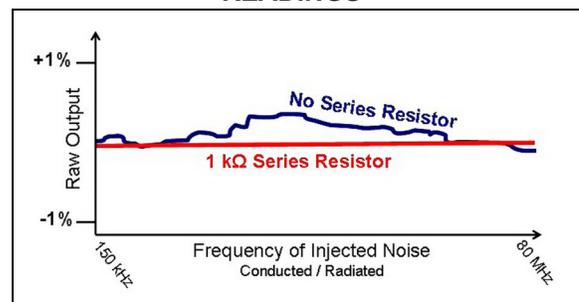
## Hardware Design Consideration #7

### USING SERIES RESISTANCE ON SENSORS

While not as mandatory as the other provided guidelines, this simple step will stabilize the sensor readings in both voltage-based and frequency-based systems. A typical resistor value is 1 k $\Omega$ , but the value can range from 100 $\Omega$  to 10 k $\Omega$ . Figure 11 demonstrates the relative difference a series resistor can make during an industry noise test.

**Note:** If using the CTMU acquisition method, do not exceed the module's maximum input impedance of 2.5 k $\Omega$  or the scan rate will decrease.

**FIGURE 11: EFFECT OF A SERIES RESISTOR ON THE STABILITY OF A SENSOR'S READINGS**



## Hardware Design Consideration #8

### CHOOSING $V_{DD}$ TO MAXIMIZE NOISE IMMUNITY

**Best Option:** Keep  $V_{DD}$  as high as possible to maximize noise immunity.

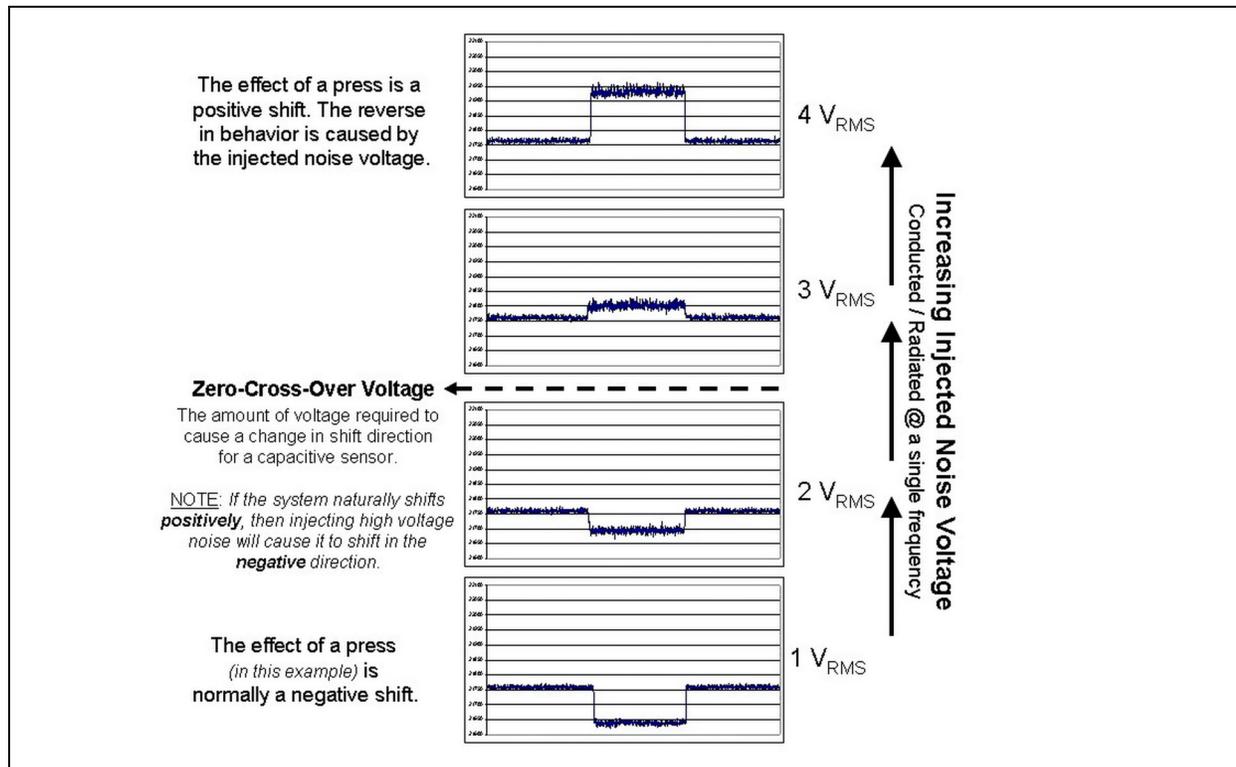
Although not ideal for low-power applications, high  $V_{DD}$  systems will perform better in conducted noise environments than low  $V_{DD}$  systems. This is due to the fact that all capacitive touch systems will eventually be overpowered by the injected noise as the voltage level of the noise is increased. Making  $V_{DD}$  a higher value will require a higher voltage level of injected noise before this happens.

# AN1334

In voltage-based acquisition systems, the behavior you are attempting to delay is the reverse press phenomenon. Figure 12 shows a regular sensor being injected with noise of increasing voltage levels. The

noise begins adding voltage to the reading which eventually overpowers the normal capacitive sensing behavior and causes a positive shift when pressed.

**FIGURE 12: REVERSE SHIFT BEHAVIOR WHEN INJECTING CONDUCTED NOISE ON A VOLTAGE-BASED ACQUISITION SYSTEM**



## SOFTWARE TECHNIQUES

### Consider Your System Requirements

Your system's limitations are based on two main factors. First, your hardware design decisions will result in a base SNR for the application. If the base SNR is high, the software will not need to filter the signal as much and the detection process will be fast and easy. If the base SNR is low, the software will need to heavily filter the signal and the detection process will need to go through more steps to make sure no false triggers are registered. The second factor to determining your limitations is the application's performance requirements. If the product must have a specific response time or if there is a limited amount of memory available, then some software techniques may not be applicable.

For example, in gaming systems speed is the most important requirement. Care should be taken when choosing a software filtering technique that the response time does not suffer significantly. The code

size should be kept small to allow for fast execution. And the sampling rate of our system may need to be increased.

When considering any of the techniques described in this section, remember that all of them have a cost in time, power, and memory usage. The benefits should always be weighed against the costs.

### Sampling Rates

One of the first decisions when creating capacitive touch firmware is whether to trigger a new scan from the main loop or from the Interrupt Service Routine. For applications concerned with noise, the second approach is recommended. Fixed-time sampling rates are important to the correct operation of filters and detection decisions should be based off a concept of how long in real-world time a new behavior has been measured. If the sample rate of the system is based on a non-fixed interval like a function call from the main loop, other applications in the system could change the sampling rate. For example, a system that is actively controlling a power supply's output voltage will have priority over mTouch sensing due to its special timing

requirements. If the power supply control application does not allow mTouch sensing to regularly scan, the system could miss a press or release.

The second decision that must be made is: What will the fixed sampling rate be? This is largely dependant on the acquisition method that has been chosen as well as the specific requirements of the system. Voltage-based acquisition methods scan often and very quickly, while frequency-based acquisition methods scan over a longer period of time at a slower rate. Gaming systems that require a very fast response time may scan over 100 times a second, while a battery-powered proximity sensor may only scan three times a second until it notices a user nearby. For many systems concerned about noise, the sampling rate and the decoding rate may be different. For example, a system could scan a sensor once every 50  $\mu$ s, continuously updating the filter, but only run the decoding sequence every 10 ms. This can reduce the amount of processing overhead while still allowing the system to adjust constantly to the changing environment.

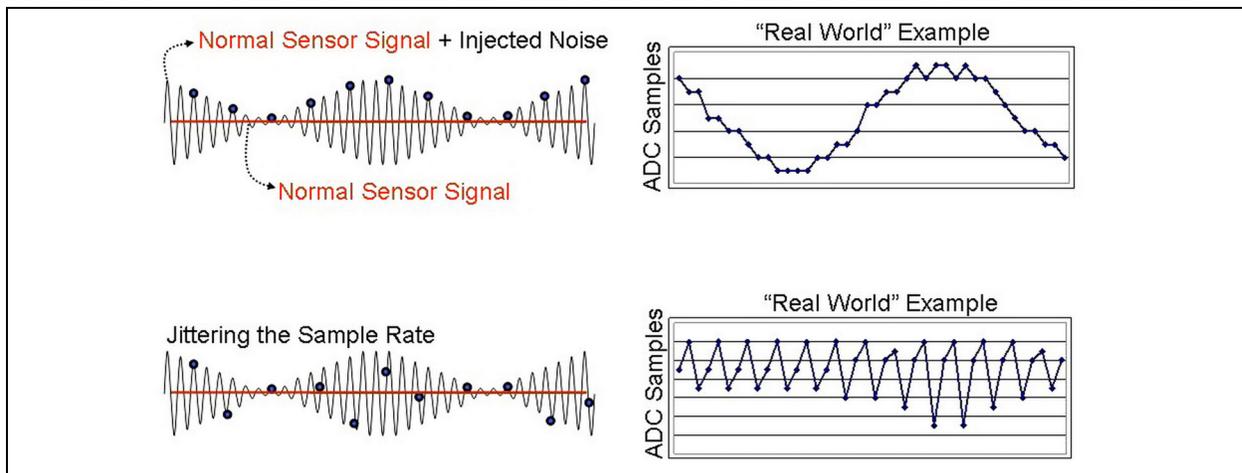
Software filters and the decoding algorithm will need to be designed with the sampling rate of the sensors in mind. If not, the filters will become too fast and suffer from too little noise reduction or they will become too slow and may cause the signal to be dampened or slowed. Decoding algorithms that do not consider the sampling rate of the system may have problems achieving a required response time or may change states (on/off) too quickly.

## Jittering the Sample Rate

### *Voltage-Based Acquisition Methods Only*

One of the problems that can occur in systems that use the Interrupt Service Routine to trigger a new scan is that the scans are then vulnerable to noise being injected at a harmonic of the sampling rate. Jittering will help to dampen high frequency noise being injected on to the system either through radiated or conducted noise. Figure 13 shows an example of what this can look like.

**FIGURE 13: SAMPLING EXAMPLE – BEFORE AND AFTER JITTERING THE SAMPLE RATE**



The simplest solution is to change the sampling rate by a very small amount each time a sample is taken using the “jittering” technique. For example, if you are scanning a sensor once every 400  $\mu$ s, you may decide to delay the reading by an extra 0-10  $\mu$ s (an amount that changes each time a sample is taken) to make sure you are not hitting any harmonics. Although this will technically change our sample rate, it will not change the average sampling rate and the change you are making is insignificant compared to the total sampling interval.

In the example jittering implementation below (Example 1), the Least Significant bits from our last ADC sample are used to create the random, short delay. The value is masked with 0x0F to limit the maximum amount of delay that is possible.

### EXAMPLE 1: JITTERING

```
#define UINT8    (unsigned char)
#define UINT16  (unsigned int)

void interrupt ISR()
{
    if (TOIE & TOIF)
    {
        // Short Delay
        jitter = ADRESL & 0x0F;
        while(jitter--);

        CVD_Service();
    }
}
```

## Oversampling

This is the process of using more than one acquisition sample per “reading”. For example, in most systems, the Interrupt Service Routine determines when a new reading should take place. When this occurs, the system acquires a value and saves it as the new reading. To increase the stability of your readings, you could have the system acquire two samples off the same sensor by scanning it twice and then adding the two samples together to create one sensor “reading”.

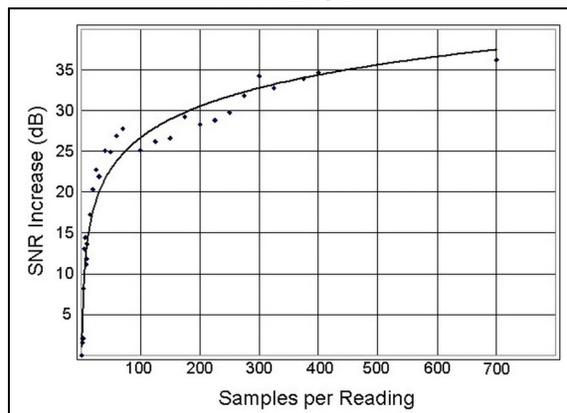
**Note:** ‘Sample’ is used to refer to a single scan of a sensor using a hardware module (e.g. ADC/CSM/CTMU). ‘Reading’ is used to refer to a group of samples that have been added together which are then sent as-is to the filtering and decoding routines.

There are several reasons why this technique is helpful to a system:

1. Sampling errors caused by impulse noise will not affect the system as much since each sample is only part of a full reading value. The system effectively averages out some of the errors during the acquisition process.
2. Samples do not have decimal values. By allowing the system to scan multiple times per reading, it is able to gain additional resolution on the signal.

The benefit of this method to the effective SNR of the system is shown in Figure 14, but it does have clear diminishing returns that should be considered against the requirement of time and power consumption

**FIGURE 14: OVERSAMPLING TRADE-OFF: TIME VS. SNR INCREASE**

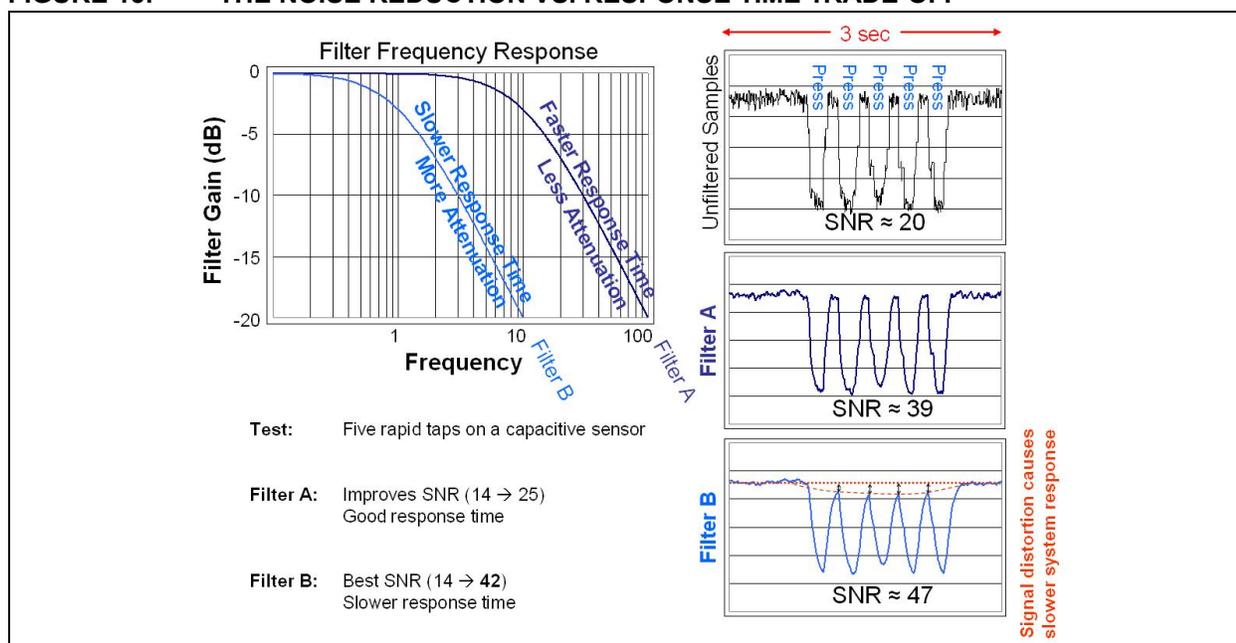


## Software Filters

Filters are algorithms that take an input signal and output a modified version of the signal. The function it performs is based on the type of filter it is. The bandwidth of a filter also plays a large role in how it performs. From a firmware perspective, the function of the filter is defined in the code structure and operations that are performed. The bandwidth is usually set by constants in the implementation that determine what number a value should be divided by, how many times to bit-shift left or right, and what coefficient should be used to multiply the result by.

With filters, there is a constant trade off between noise reduction and response time delay. This trade-off can be visualized in the bandwidth of the filter as shown in Figure 15.

**FIGURE 15: THE NOISE REDUCTION VS. RESPONSE TIME TRADE-OFF**



If the filter's bandwidth is narrow, less noise will be able to pass through, but it may take a long time for the filter to follow the signal. On the other hand, if the filter's bandwidth is wide, more noise will be able to pass through, but it will follow the signal more closely. Combining multiple filters can allow the designer to get the benefits of each while limiting the negative impact.

There are three types of filters that are commonly used in mTouch sensing solution applications. More filters could easily be added to this list and may be more appropriate for a specific application, but these have been chosen because most designs will be able to use one or more of them.

The three filter types are:

1. Slew Rate Limiter

*Used as the first input filter on new incoming samples to reject impulse noise and smooth the signal.*

*Implemented in the acquisition routine.*

2. L-Point Running Average

*Used to create a slow-updating (high time constant) baseline ("average") for each sensor as a reference point during decoding. Allows the system to track environmental changes such as temperature and humidity.*

*Implemented in the filtering routine.*

3. Low Pass Butterworth

*Used to reject white noise on the sensor readings while still maintaining a fast response time (low time constant).*

*Implemented on the 'reading' variable in the filtering routine before sending it to the decoding algorithm.*

## FIR Filters vs. IIR Filters

Finite-Impulse Response (FIR) filters take a fixed number of previous inputs and use them to create the next output.

Finite-Impulse Response Filter Benefits:

- Simple implementations
- Better filter stability
- Fewer concerns about integer precision

Infinite-Impulse Response (IIR) filters take the input and use it in combination with the previous output of the filter to determine the next filter output.

Infinite-Impulse Response Filter Benefits:

- Low memory requirements
- Low processing requirements

Ultimately, both filter types can be useful to an application. For capacitive touch systems, IIR filters can be used for slowly-updating environmental baselines. The filter should be designed so that it can handle impulse noise without becoming unstable. FIR filters can be used for quickly-updating sensor variables like the reading.

## Filter: Slew Rate Limiter

The Slew Rate Limiter (SRL) filter's main design goal is to reject impulse noise from sensors' readings. Unlike the other two filters described in this section, implementing the SRL filter requires a specific scanning technique that will possibly change the sample rate of your design.

The concept of the SRL filter is simple. The PIC device is maintaining a "current reading" variable for each sensor. In most systems, when a new sensor reading is created, the "current reading" variable is replaced by the new value. In an SRL filtered system, when a new reading value is generated, the "current reading" value is then either decremented or incremented by 1 based on whether the latest reading is higher or lower than the "current reading" variable. For example, if a sensor's current reading is 200 and the next acquisition results in a value of 300, the system will update the "current reading" to 201. In order for the system to reach a current reading of 300, the next 99 scans must be higher than the current reading.

This behavior is very beneficial because it limits the influence of each sample. If impulse noise is affecting the system, a single impulse-noise-affected reading will only cause 1 bit of noise on the reading variable. On the other hand, because you are updating the current reading variable so slowly, you need to update the rate of the samples. When a user presses on a sensor, the current reading variable needs to be able to move with the finger's capacitance at a fast rate. There is also no need to access the decoding function of the system after each individual reading since it can only shift by 1 each time.

There will be several parts to the SRL filter implementation to take care of these special requirements. First, the system will scan based on a timer's interrupt at a fast rate. After each of these scans, it will run the SRL filter to increment/decrement the "current reading" variable. After the N<sup>th</sup> sample, a flag is set that allows the decode function to run. An example code implementation of this filter can be seen below in Example 2.

## EXAMPLE 2: SLEW RATE LIMITER FILTER

```

#define SCANS_PER_DECODE 100

UINT16 reading;
UINT16 counter;

void main(void)
{
    // Main Loop
    while(1)
    {
        if (counter >= SCANS_PER_DECODE)
        {
            mTouch_decode();
            counter = 0;
        }
    }
}

void interrupt ISR(void)
{
    UINT16 newReading;

    if (TMR0IE && TMR0IF)
    {
        // Take a reading and store the value
        newReading = mTouch_getReading();

        // Initialize the reading if starting
        if (reading == 0) reading = newReading;

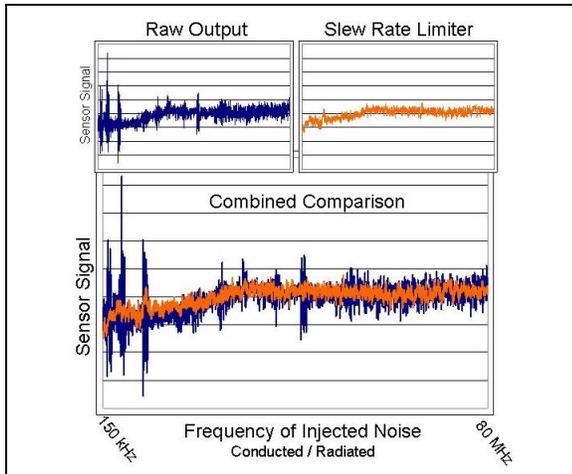
        // Slew Rate Limiter
        if (newReading > reading)    reading++;
        else                          reading--;

        counter++;
    }
}

```

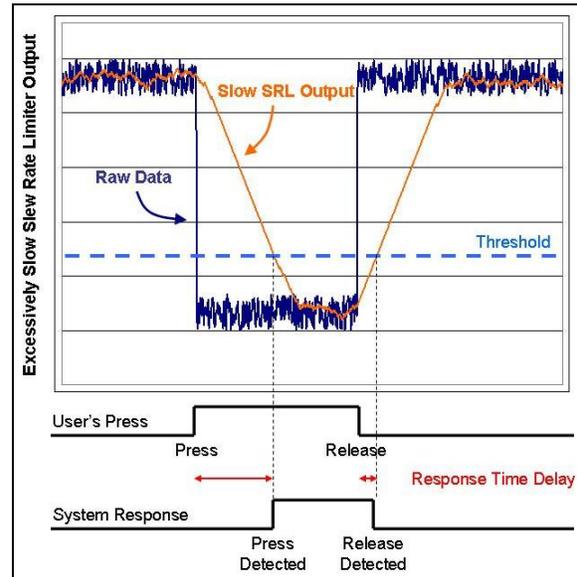
The benefits of this filter can be seen in Figure 16. The impulse noise on the system has been rejected and the signal is now more easily decoded.

## FIGURE 16: SLEW RATE LIMITER FILTER BEHAVIOR



Care should be taken to make sure the SRL filter is not moving too slowly. If the sampling rate is too low, the current reading value will not update fast enough and can cause problems with response times. An example of what this too-slow behavior will look like is shown in Figure 17.

## FIGURE 17: EXCESSIVELY SLOW SLEW RATE LIMITER FILTER EXAMPLE



To solve this issue, either:

1. Reduce the amount of time between timer interrupts.
2. Change the increment/decrement amount to a value larger than 1. However, the larger this value is, the less the filter will be able to reject impulse noise.

## Calculating the Minimum Sampling Rate in a Slew Rate Limited System

The system needs to be able to transition from its average, unpressed sensor value to the sensor's threshold in the required response time. That leads to the creation of Equation 4(a) below, with the example behavior shown in Figure 18(a).

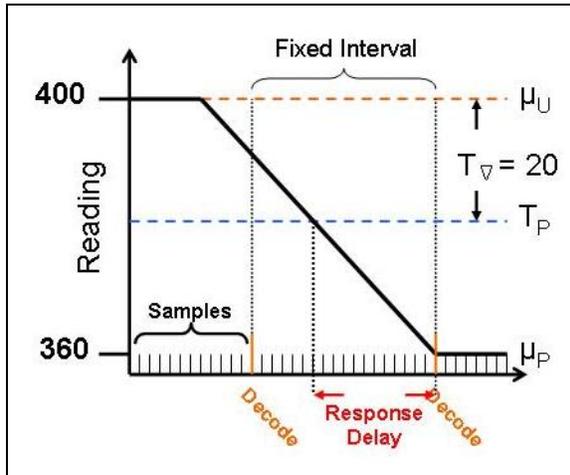
### EQUATION 4(A): MINIMUM SAMPLING FREQUENCY (BASIC)

$$f_{s \min} = \frac{T_{\nabla}}{t_r n_{\max}}$$

Where:

- $f_{s \min}$  is the minimum sampling frequency
- $T_{\nabla}$  is the absolute value of the relative (to the baseline) press threshold
- $t_r$  is the required response time
- $n_{\max}$  is the maximum step size of the SRL filter after one sample

**FIGURE 18(A): FIXED INTERVAL DECODE FUNCTION CALLS**



Calculating the minimum sample frequency with a system that implements debouncing is also simple. The previous result can be multiplied by the debounce requirement to determine how fast the system must sample to make sure a press can be detected in the given response time, as shown in Equation 4(b).

**EQUATION 4(B): MINIMUM SAMPLING FREQUENCY (DECODE AFTER FIXED NUMBER OF SAMPLES)**

$$f_{s \min} = \frac{T_{\nabla}}{t_r n_{\max}} * \text{debounce}_{\max}$$

Where:

|                          |   |
|--------------------------|---|
| $f_{s \min}$             | is the minimum sampling frequency                                       |
| $T_{\nabla}$             | is the absolute value of the relative (to the baseline) press threshold |
| $t_r$                    | is the required response time   |
| $n_{\max}$               | is the maximum step size of the SRL filter after one sample             |
| $\text{debounce}_{\max}$ | is the maximum required debounce count to change states                 |

Because the speed of the SRL filter is defined to be  $T_{\nabla} / n_{\max}$ , if the press occurs asynchronously with the decode function calls, it will not be past the threshold by the time the next decode function call is performed. So there will be a system delay while waiting for the next call.

However, we know that we are, at minimum, going to call the decode function once while the shift is occurring or we will call it right as the threshold is reached. Instead of calling the decode function every  $T_{\nabla} / n_{\max}$  samples, the counter value can be calculated each

time to determine the minimum number of steps that would be required to cross the threshold given the current reading value. This decode counter calculation is shown in Equation 5 and an example of its behavior is shown in Figure 18(b).

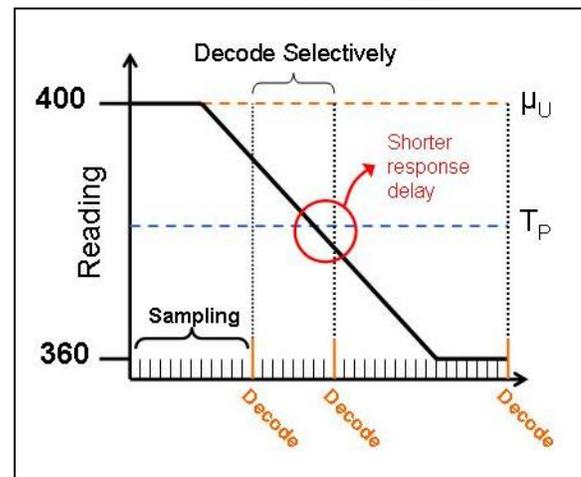
**EQUATION 4(C): MINIMUM SAMPLING FREQUENCY (DECODE SELECTIVELY)**

$$f_{s \min} = \frac{1}{t_r} \left( \frac{T_{\nabla}}{n_{\max}} + N_{\min} \right)$$

Where:

|              |   |
|--------------|---|
| $f_{s \min}$ | is the minimum sampling frequency                                       |
| $T_{\nabla}$ | is the absolute value of the relative (to the baseline) press threshold |
| $t_r$        | is the required response time   |
| $n_{\max}$   | is the maximum step size of the SRL filter after one sample             |
| $N_{\min}$   | is the minimum number of samples per decode function call               |

**FIGURE 18(B): SELECTIVE DECODE FUNCTION CALLS**



It is important to be careful when calling the decode function at non-fixed intervals. If the system is using a secondary filter in later stages for the sake of decoding real-time motion, the amount of time that elapses between decode function calls will change. This means that the secondary filter will need to be updated with care.

Equation 4(c) shows how to calculate the minimum sampling rate for a system that has the same required response time as before, but now is able to call the decode function selectively.

## EQUATION 5: DECODE FUNCTION CALL COUNTER CALCULATION

$$DecodeCounter = \frac{|reading - T_P|}{n_{max}} + 2$$

Where:

reading is the current reading value  
 $T_P$  is absolute press threshold  
 $n_{max}$  is the maximum step size of the SRL filter after one sample

## Filter: L-Point Running Averages

This filtering technique is used in a vast number of applications and has been documented thoroughly. Its behavior is defined in Equation 6. The current value,  $x[n]$ , is averaged with the last  $L-1$  values. When deciding on a value for  $L$ , keep in mind the division operation. If a power of 2 is chosen for the value of  $L$ , the division operation can be simplified to a series of bit-shifts, reducing the complexity of the filter.

## EQUATION 6: L-POINT RUNNING AVERAGE (FIR)

$$y[n] = \frac{1}{L} \sum_{k=0}^{L-1} x[n-k]$$

Where:

$y[n]$  is the output at time 'n'  
 $x[n]$  is the input at time 'n'  
 $L$  is the memory of the filter  
 $k$  is a counter variable

An example code implementation of the above filter can be seen in Example 3.

## EXAMPLE 3: FIR L-POINT AVERAGE FILTER

```
#define HISTORY 8 // 'L'
#define UINT8 (unsigned char)
#define UINT16 (unsigned int)

UINT16 reading[HISTORY];
UINT8 index;

UINT16 FIR_Average(UINT16 newReading)
{
    UINT8 i;
    UINT16 average = 0;

    // Replace oldest reading
    reading[index] = newReading;

    // Sum all reading values
    for (i = 0; i < HISTORY; i++)
    {
        average += reading[i];
    }

    // Divide by the history window size
    // NOTE: This operation is simple and fast
    // as long as HISTORY is a power of 2.
    average = (UINT16)(average/HISTORY);

    // Update index for next function call
    index++;
    if (index >= HISTORY)
    {
        index = 0;
    }

    return average;
}
```

Also notice that, as implemented in Equation 6, this is an FIR filter which means that a single reading can only affect the output for a specific number of samples,  $L$ . After this period, the sample no longer has an influence on the system's behavior. For noise-injected situations, this is a very beneficial characteristic for our filters to have. However, the memory cost of this filter will make any filters with a large window difficult to implement.

To solve this limitation of the filter, its behavior can be changed from FIR to IIR as shown in Equation 7. This creates a fixed, low memory cost for the filter but degrades some of its features. The IIR version of the L-Point Running Average will become more distorted from the original as  $L$  gets larger.

## EQUATION 7: L-POINT RUNNING AVERAGE ESTIMATE (IIR)

$$y[n] = y[n-1] + \frac{x[n] - y[n-1]}{L}$$

An example code implementation of the above filter can be seen below in Example 4.

**EXAMPLE 4: IIR L-POINT AVERAGE FILTER**

```

#define HISTORY 8 // 'L'
#define UINT8 (unsigned char)
#define UINT16 (unsigned int)

UINT16 average;

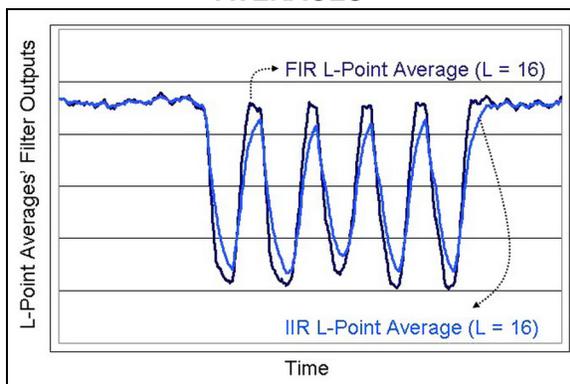
UINT16 IIR_Average(UINT16 newReading)
{
    // Update Average
    average -= (UINT16)(average/HISTORY);
    average += newReading;

    // NOTE:
    // This filter implementation has a gain
    // of 'HISTORY'. For that reason, we must

    // either divide the average by HISTORY
    // or multiply the reading by HISTORY
    // before we are able to compare them.
    return (UINT16)(average/HISTORY);
}

```

Figure 19 shows the difference between the FIR L-point running average and the IIR L-point running average estimation. Notice that the estimation introduces a response time delay in the system. For this reason, it is not recommended to use this filter directly on the reading signal. However, the time delay of this filter is not a problem for the baseline calculation. Since the baseline should be moving very slowly, the time delay will not affect its behavior negatively.

**FIGURE 19: FIR VS. IIR RUNNING AVERAGES****Filter: Low Pass Butterworth**

This filter implementation is an alternative to the L-point running average. While both are low pass filters, the digital low pass filter in this section is based on the digital implementation of a Butterworth filter and can be seen defined in Equation 8. Notice that the only complex operation in this filter is the multiplication between K and the previous output of the filter. If K is chosen wisely, this filter can be easily implemented with the use of bit-shifts.

**EQUATION 8: DIGITAL BUTTERWORTH LOW PASS FILTER**

$$y[n] = x[n] + x[n-1] + Ay[n-1]$$

Where:

$y[n]$  is the output at time 'n'

$x[n]$  is the input at time 'n'

A is the filter's coefficient ( $0 \leq A < 1$ )

Smart values for A include: 0.8125, 0.8750, 0.9375, and 0.9688, to name a few. These can be implemented easily using only bit-shift operations on the  $y[n-1]$  value. An example of this can be seen below:

**EXAMPLE 5: LOW PASS BUTTERWORTH FILTER**

```

#define UINT8 (unsigned char)
#define UINT16 (unsigned int)
#define FILTER_GAIN 3

typedef struct
{
    UINT16 y;
    UINT16 x;
} FILTER;

FILTER filter [NUM_BTNS];

UINT16 LP_Butterworth(UINT16 reading)
{
    // Pointer to the correct filter
    // variables for our sensor...
    FILTER* h = &filter[index];

    // Temporary variables
    UINT16 x1, y1, ay1;
    UINT16 temp0, temp1;

    // Populate temporary variables
    x1 = (h -> x);
    y1 = (h -> y);

    // Calculate: ( a * y[1] )
    // Where, a = 0.8125.
    temp0 = y1 >> 2;
    temp1 = y1 >> 4;
    ay1 = y1 - temp0 + temp1;

    // 1st Order Filter Equation:
    // y1 = x[0] + x[1] + (a * y[1])
    y1 = reading + x1 + ay1;

    // Store values for next time
    (h -> y) = y1;
    (h -> x) = reading;

    // Return the filter's new result
    return y1 >> FILTER_GAIN;
}

```

## EQUATION 9: CALCULATING FOR 'A'

$$Ay[n-1] = y[n-1] - \frac{y[n-1]}{4} + \frac{y[n-1]}{16}$$

$$Ay[n-1] = y[n-1] \left( 1 - \frac{1}{4} + \frac{1}{16} \right)$$

$$Ay[n-1] = 0.8125y[n-1]$$

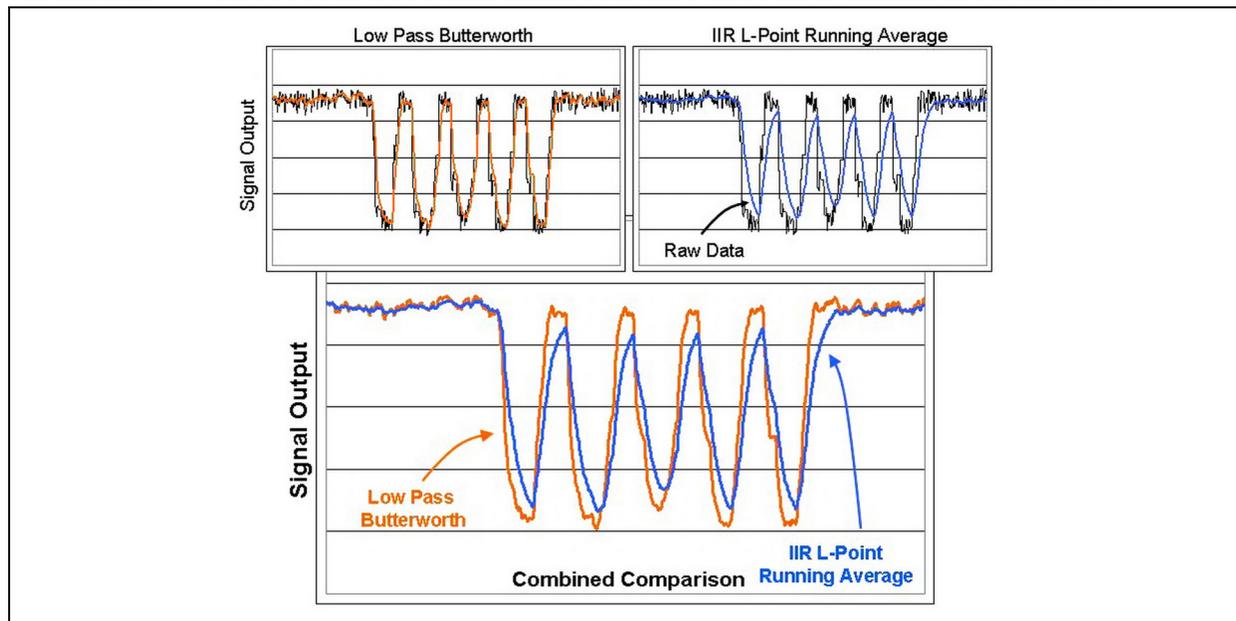
$$A = 0.8125$$

As the value of 'A' gets closer to 1, the cutoff frequency of the Butterworth low pass filter approaches zero. This increases the effectiveness of the filter, but will slightly

increase the filter's settling time. Also, as 'A' gets closer to 1, the integer value of  $y[n]$  will increase quickly. This puts an upper limit on the value of 'A' if you want to continue representing each sensor's signal with the typical 16-bit integer value.

The benefit of this type of filter when compared with the L-point running average can be seen in Figure 20. The increase in the effective SNR of the sensor is much higher when implementing the Butterworth than the running average; however, the running average estimate is very inexpensive to implement and performs the job well as a filter for the sensor's baseline.

**FIGURE 20: LOW PASS BUTTERWORTH FILTER VS. L-POINT RUNNING AVERAGE**



## Setting Thresholds

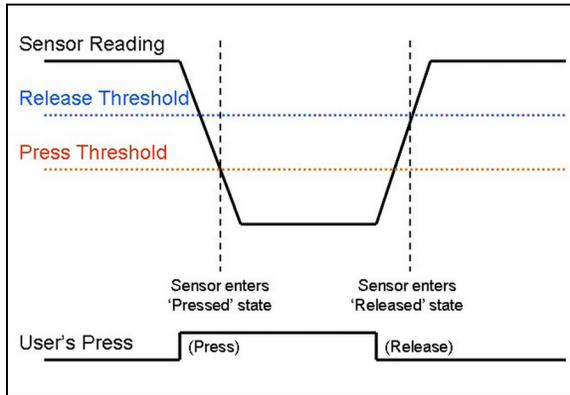
After the sensor signal has been read and filtered, the decoding process should begin. During this step, the sensor's signal will be compared against a pre-determined threshold to decide if there is a press.

When establishing a threshold, there are two main options: setting a fixed threshold or calculating the threshold at run-time. Setting a fixed threshold is the easiest and least time/memory intensive, but it may not work in every situation. For mass production systems, it is possible there will be slight differences in the sensor's default values across boards. If the application is using a fixed threshold, there is a possibility that the threshold may not work for all of them. Calculating the threshold at run-time is the other option. When this is implemented, the system will take several readings on power-up and will determine where the threshold should be set based on the system's

samples. For the majority of applications, a calculated threshold will not significantly affect the system's behavior and is preferred for its flexibility.

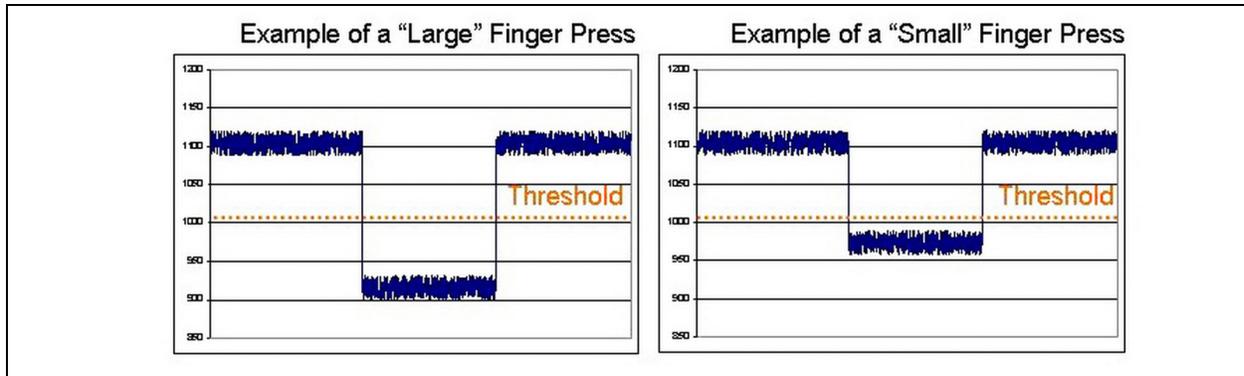
In some cases, two thresholds may be used in the same way as hysteresis – one used to enter the pressed state and one used to enter the released state. A diagram showing this behavior is seen in Figure 21.

**FIGURE 21: THRESHOLD HYSTERESIS BEHAVIOR**



Choosing where to place the thresholds for a system can be one of the most important and difficult tasks of getting a robust solution working. Assuming the sensor's signal moves down when pressed, if the threshold is too high, false triggers can become a risk. If the threshold is too low, the system may be unable to detect a press in all situations. It is important to remember that your system will be used by a wide variety of people. People with big hands will cause more of a shift than those with small hands. Make sure you are setting the thresholds and testing them with a cross-section of individuals so that everyone will be able to activate the sensor.

**FIGURE 22: SENSOR SIGNAL VS. TIME AND FREQUENCY OF RESULT**

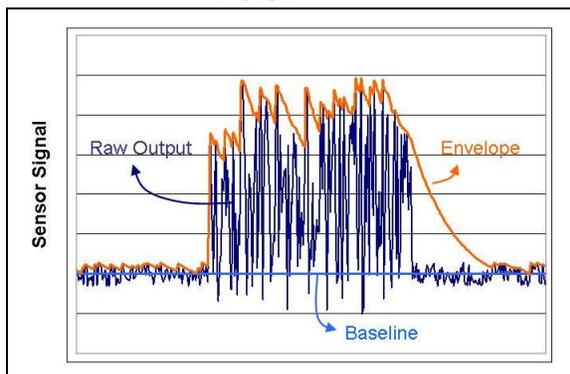


## Envelope Detector

The Envelope Detector is a decoding technique that uses an extra variable to track the average deviation from the sensor's baseline (or "average"). This can be particularly effective in systems that experience a large amount of injected noise. Figure 23 illustrates an example of a system that would benefit from this decoding technique.

When a high level of injected noise is present on capacitive touch sensors, press detection can be difficult if the decoding algorithm is only looking for a shift in sample values. Some frequencies of noise may cause the press behavior shown in Figure 23. When this occurs, an envelope can be created to track the noise level of the system and this can be used with a threshold to make press decisions. An example implementation of this decoding technique has been provided in Example 6.

**FIGURE 23: ENVELOPE EXAMPLE DURING HIGH INJECTED NOISE**



## EXAMPLE 6: ENVELOPE DETECTOR

```
#define INT16 (signed int)
#define UINT16 (unsigned int)

#define UPDATE_SPEED 4
// Fastest-Slowest
// 2, 4, 8, 16, 32

UINT16 envelope;
UINT16 reading;
UINT16 average;

UINT16 Update_Envelope_Detector(void)
{
    INT16 delta;

    // Delta = | Average - Reading |
    delta = average - reading;
    if (delta < 0)
    {
        delta = -delta;
    }

    // Update envelope
    envelope -= (UINT16)(envelope/UPDATE_SPEED);
    envelope += (UINT16)(delta);

    return envelope;
}
```

Keep in mind while implementing this technique that the delta value is the absolute value of the difference between the current sensor reading and the average. This means that shifts in both the positive and negative direction will cause the envelope to increase. If this behavior is not desirable for your system, the absolute value section of the code example can be removed. For more information about the envelope detector decoding algorithm, see application note, AN1317, "mTouch™ Conducted Noise Immunity Techniques for the CTMU".

## COMMON CHALLENGES

There are several behaviors of capacitive touch systems that commonly appear. In this section, the main issues seen in these applications will be discussed and the different possible solutions described that can be used.

The challenges and solutions covered in this section are:

- Crosstalk
- Impulse Noise
- Unresponsive Buttons
- Flickering Buttons
- Reversed Operation

## Common Challenges: Crosstalk

Crosstalk is the undesired shift of a sensor when an adjacent sensor is pressed. It is mainly a side effect of placing sensors too close together, but can be also significantly affected based on the thickness and relative permittivity of the covering material.

**Best Option:** Increase the amount of separation between sensors. See Section "Hardware Design Consideration #1" for all available options.

If possible, reducing the amount of crosstalk using hardware design techniques is preferred to the software adjustments that can be made. Crosstalk will also become more of a problem in systems with thick covers, so reducing the thickness may also be a required step. For more information on the causes of crosstalk and the hardware changes you can make to reduce this effect, see Section "Hardware Design Consideration #1".

### Option 2: Adjust the Thresholds

If these hardware solutions are not adequate or cannot be used by your specific application, adjusting the software may be your only solution. Changing the threshold by increasing the amount of shift required to detect a press might be the best option if the system has more sensitivity than needed. By requiring that the sensor's readings shift more than the maximum crosstalk shift of the system, you can eliminate the ability of crosstalk to activate a sensor. Keep in mind when doing this, however, that noise on the system may increase the maximum crosstalk shift which could cause false triggers in the future.

### Option 3: Implement the "Most Pressed" Algorithm

Alternatively, the system can be changed to compare each sensor's shift with the other sensors in order to determine which one is "most pressed." Implementing this algorithm will limit your system's capabilities. First, the system will only be able to support one press at a time since only the highest shift sensor is always picked. To minimize the effect of this limitation, only compare nearby sensors to each other. The second limitation this algorithm places on a design is on response time. The extra execution time required by the processor and the requirement that all sensors must be scanned before the decoding process can begin will both slow the system down. Depending on the application, this may or may not be an acceptable design trade-off.

## Common Challenges: Impulse Noise

Impulse noise appears as individual or small groups of readings that behave in a significantly different manner than the readings before and after them due to a noise source and not a finger's press. These spikes can quickly destabilize a system if not handled correctly. Additionally, if noise spikes are a common problem for the system, they can also lower the effective SNR.

Software filters are usually the easiest method for removing this problem. The key is to adjust the filter to reduce the affect of the spikes, but not affect the response time of the system. For example, if the decision is made to take the average and just slow it down to minimize the impact a single reading can have, it will reduce the noise spikes in the system. However, it will also slow down the response time since reliance is now on a much slower average.

**Best Solution:** *Implement the Slew Rate Limiter filter.*

Infinite Impulse Response (IIR) filters are not going to be as effective as Finite Impulse Response (FIR) filters will be in this case. Since IIR filters allow one reading to affect the signal for a (theoretically) infinite period of time into the future, the effect of a noise spike could cause the filter to become unstable. The best option in this case is to use the Slew Rate Limiter filter described in the software section of this application note. Because of its design, it will eliminate all impulse noise from the system with a minimal impact on the overall system.

## Common Challenges: Unresponsive Buttons

Unresponsive buttons are sensors that will not change state when a finger is placed on them. The two main software features that will affect your sensors to possibly make them unresponsive are thresholds and debounce values. Sensor sensitivity can change as noise is injected on a system. This change can reduce the sensitivity to the point that the threshold is no longer crossed. Increased noise on the system will increase the probability of a single sensor reading falling outside of the acceptable range to change state. If the maximum debounce value is high and the noise is high, it is possible the sensor will never respond.

There are several things that can be done to fix this problem in a system.

**Best Solution:** *Adjust the Thresholds*

First, the threshold can be made easier to cross by modifying the software. When noise reduces the sensitivity of the system, the easier threshold will decrease the possibility of an unresponsive button. This may not be a solution if making the threshold easier to cross will introduce false triggers into the system. The best way to make sure there is plenty of

sensitivity to work with in this situation is to follow the hardware design guidelines described in this application note.

**Option 2:** *Check Both Directions for Shifts*

Another possibility is that the system is being injected with a specific amount of noise that has caused it to reverse its press behavior. When this happens, pressing on the sensor will cause a shift in the opposite direction as expected. The solution to this problem is to place the threshold on both sides of the readings. This may not be an option, however. In some systems, the direction of a shift indicates a specific event. For example, in some water-resistant systems a shift in one direction is a press while a shift in another direction indicates the presence of water. Implementing a shift-check in both directions would eliminate the water resistance of the application.

**Option 3:** *Use Hardware to Increase Sensitivity*

Finally, if adjusting the thresholds does not solve the problem, hardware changes may need to be made to increase your system's base sensitivity. First, make sure the application is following all of the design guidelines. The next step is to begin increasing sensor area, decreasing crosstalk through sensor separation, decreasing the cover thickness, or changing its material. All of these suggestions can be traced back to their origins in Equation 2 which defines the relationship between capacitance and hardware design.

## Common Challenges: Flickering Buttons

The term "flickering button" refers to a sensor that will rapidly toggle its state while a finger is present. This is distinctly separate from a false trigger, which is a sensor changing state while there is no finger. Flickering buttons are caused in the same way as unresponsive sensors. As noise is injected on a sensor, the sensitivity of the sensor can change. If the sensitivity is reduced to the point that it approaches the threshold, noise can sometime cause the readings to jump above and below it. This will result in the sensor rapidly changing states.

**Best Solution:** *Implement Threshold Hysteresis*

The best and most effective solution to this problem is to adjust your algorithm so that it implements a large hysteresis on the thresholds. Separate press and release thresholds will be able to tolerate a much larger amount of system noise. If sensitivity levels are high, the thresholds will be able to be placed further apart, increasing the noise resistance of the system. Similarly, as the thresholds are placed closer together, the system begins to return to its single-threshold behavior and can once again experience flickering buttons. For these reasons, a combination of hysteresis and debouncing should be the default solution to this problem in applications.

## Option 2: Increase the Debounce Requirement

The easiest way to reduce this problem is to increase the maximum debounce value, but this method can have mixed results. Increasing the debounce count will decrease the response time of the system and will only decrease the probability of flickering. The smaller probability of flickering will translate to a slower flicker, but most likely not the complete elimination of it.

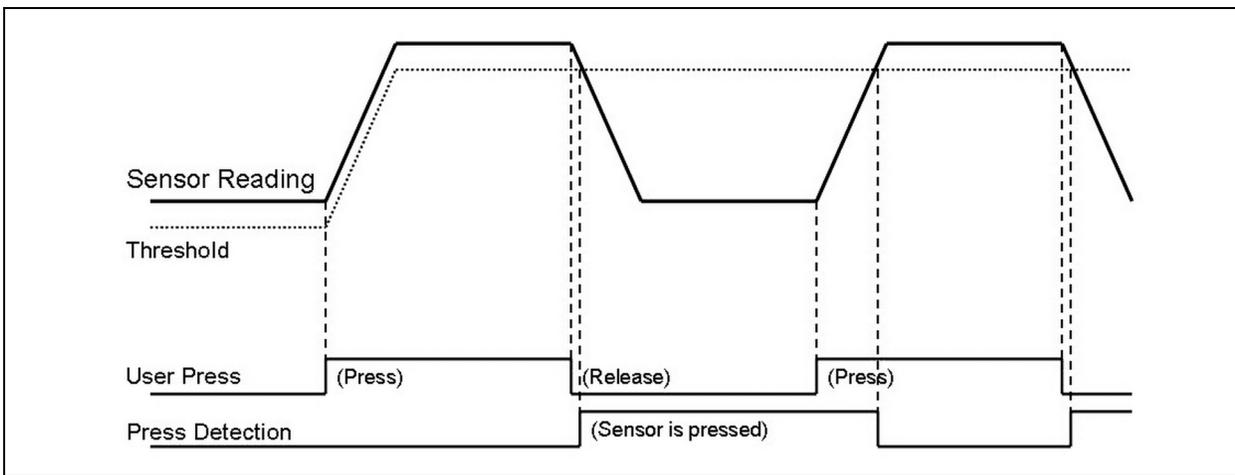
## Option 3: Adjust the Thresholds

A third option is to make the threshold easier to cross or to increase the sensitivity of the system through hardware changes. However, if there is a large amount of noise on the system, this still may not be enough by itself to solve the problem.

## Common Challenges: Reversed Operation

Reversed operation can occur in systems with a high amount of injected noise. The noise can cause the system to shift in the opposite direction as the no-noise behavior. If the system was shifting down on each press, it is now shifting up. This can be a problem in systems that only look for a shift in one direction. Figure 24 shows how a system looking for a “down” shift can enter a reversed operation state if an “up” shift occurs. There are two main ways of solving this problem. The baseline’s speed can be decreased and thresholds can be placed on both sides of the baseline.

**FIGURE 24: ENTERING REVERSED OPERATION DUE TO NOISE**



## Best Solution: Check Both Directions for Shifts

The more robust solution to this problem is to create a threshold on either side of the average so that a shift in either direction will result in a detected press. Not all systems may work well with this solution, however. For example, in some systems designed to work in wet conditions, a shift in one direction means a finger is present while a shift in the other direction corresponds to the effect of water. In a situation such as this, it may be wiser to ignore any shifts in the wrong direction. Ultimately, the decision should be based on the specific application’s requirements.

## Option 1: Adjust the Filters’ Time Constants

Slowing down the baseline’s speed would require that a longer opposite-direction-from-normal shift occur before the system enters the reversed operation state. This would cause unresponsive buttons when specific types of noise are injected onto the application, but may be the best option for a given system. Just slowing down the average will not eliminate the problem, however. If a user were to press on a sensor for a very long period of time, the slow average will eventually follow the finger and when it is finally removed, reversed operation could still occur. This is a performance trade-off that must be considered when

using any baseline in a system. Determining the difference between the environment naturally shifting the readings and a finger artificially shifting the readings can sometimes be difficult.

## Option 2: Increase the VDD of the system

While this will not eliminate the problem, raising VDD will fight against the reversing behavior by increasing the amount of noise that must be injected on the system before it flips. This option is placed last because increasing VDD results in a higher current consumption for the system.

## CONCLUSION

Good hardware design choices are the foundation to a robust capacitive touch design. By following the provided design guidelines, your custom application will have a high base signal-to-noise ratio (SNR) which will decrease your required software overhead, speed up your system's overall response time, and allow your system to perform well even in noisy conditions.

Although there are many opportunities to adjust a design to meet a specific application's needs, the best design choices for a robust system are:

1. The sensor size should be the same as an average user's finger press (15x15 mm)
2. Sensors should be separated as much as possible. Ideal minimum separation is 2-3 times the cover's thickness.
3. Keep the cover as thin as possible. Ideally the cover thickness should not exceed 3 mm to maximize sensitivity.
4. Use ground planes to your advantage to minimize radiated and conducted noise, but be careful of reducing sensitivity.
5. Keep sensor traces thin and short.
6. Always use an appropriate adhesive.
7. Add series resistors to the sensor traces.
8. Keep VDD as high as possible to maximize noise immunity.

Strategic use of acquisition techniques such as jittering the sample rate and implementing a slew rate limiter will reduce the amount of noise that is introduced through the readings. Digital filters such as the L-Point Running Average and the low pass Butterworth filter allow the system to track environmental changes and further increase the SNR of the signal. Finally, specific algorithm implementations such as threshold hysteresis, debouncing, and the "most pressed" algorithm will allow the system to react to even the most rare of noise disturbances.

From start to finish, capacitive touch systems should be developed with the key focus of increasing the SNR of the sensors' signal. Changing from a digital, mechanical switch to an analog, capacitive sensor is not a one-step process, but following the given hardware suggestions and implementing the provided software algorithms will make the system both cost effective and robust.

For information on the basics of mTouch sensing and other more advanced topics, visit the Microchip web site at <http://www.microchip.com/mTouch>.

## GLOSSARY OF TERMS

### **Average**

A value calculated in real time by the system's firmware to estimate what the next reading of the sensor should be assuming no external interference.

### **Base SNR**

The signal-to-noise ratio of the unfiltered sensor signal.

### **Baseline**

See 'Average'. These two terms are used interchangeably.

### **Cover**

A layer of typically plastic or glass that is placed between the application's PCB and the user's finger.

### **Crosstalk**

The undesired shift of a neighboring sensor's readings when the user is pressing on a different sensor.

### **CSM ("Capacitive Sensing Module")**

An mTouch™ sensing solution hardware module used to measure the capacitive shift of a sensor using a frequency-based method. A timer module is used to count the number of oscillations the sensor's signal performs in a fixed amount of time.

### **CTMU ("Charge Time Measurement Unit")**

An mTouch™ sensing solution hardware module available in some PIC18 and PIC24 devices that uses a voltage-based acquisition method to measure the capacitance of a sensor.

### **CVD ("Capacitive Voltage Divider")**

An mTouch™ sensing solution acquisition technique that uses a PIC MCUs ADC module to take a voltage-based capacitive measurement of a sensor.

### **Debounce**

Algorithm process that requires the same answer be independently calculated N times in a row before a state change can occur.

### **Decoding**

The algorithm process of taking an analog integer value and using it to determine the current state of the sensor.

### **False Triggers**

Incorrect sensor state transitions that are not caused by a finger's press. Do not confuse these with 'Flickering Buttons' which occur when a finger is present.

### **Flickering Buttons**

The sensor behavior of quickly toggling between sensor states while a finger remains pressed on the sensor. Do not confuse these with 'False Triggers' which occur when a finger is not present.

### **Hysteresis**

A control theory technique that uses several signal thresholds to eliminate or reduce fast state toggling while the signal is transitioning.

### **Noise**

The unwanted jitter of a signal usually caused by an external source.

### **Noise Immunity**

The ability to remove or ignore the noise on a sensor's signal.

### **Noise Spikes**

Individual or small groups of readings that behave in a significantly different manner than the readings before and after them due to a noise source and not a finger's press.

### **Oversampling**

Taking more than one sample of a sensor's signal and combining them into one final reading that is then processed by the firmware's algorithm.

### **Parasitic Capacitance**

The unwanted capacitance that exists between two elements of a circuit simply because of their proximity to each other.

### **Permittivity**

A measure of how much resistance is encountered when forming an electric field through a material. Higher permittivity values mean less resistance.

### **Reading**

The analog integer value that represents the sensor's current value and that is passed to the filtering or decoding algorithms. Not to be confused with 'Sample'.

### **Reversed Operation**

A phenomenon caused by a large amount of noise on the system which reverses the operation of the sensor. Pressing makes the sensor think it has been released and releasing makes the sensor think it has been pressed.

### **Sample**

A single result from a hardware module that describes the sensor's current value. Multiple samples might be combined using the 'Oversampling' technique to create a 'Reading' which is then used in the algorithms calculations.

### **Sensitivity**

A measure of how much a sensor's value will shift when a finger is pressed on it. The shift is sometimes defined in terms of the percentage of the total value, and sometimes as the absolute shift amount.

### **Signal-to-Noise Ratio (SNR)**

A measure of how much sensitivity a system has compared to the level of noise on the signal. The higher the SNR, the cleaner the signal.

### **Threshold**

A limit used to define at what point a sensor should change states.

---

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

**Trademarks**

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC<sup>32</sup> logo, rPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICKit, PICtail, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2010, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-60932-435-3

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

**QUALITY MANAGEMENT SYSTEM  
CERTIFIED BY DNV  
== ISO/TS 16949:2002 ==**



---

---

## WORLDWIDE SALES AND SERVICE

---

---

### AMERICAS

**Corporate Office**  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
<http://support.microchip.com>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

**Atlanta**  
Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455

**Boston**  
Westborough, MA  
Tel: 774-760-0087  
Fax: 774-760-0088

**Chicago**  
Itasca, IL  
Tel: 630-285-0071  
Fax: 630-285-0075

**Cleveland**  
Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

**Dallas**  
Addison, TX  
Tel: 972-818-7423  
Fax: 972-818-2924

**Detroit**  
Farmington Hills, MI  
Tel: 248-538-2250  
Fax: 248-538-2260

**Kokomo**  
Kokomo, IN  
Tel: 765-864-8360  
Fax: 765-864-8387

**Los Angeles**  
Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608

**Santa Clara**  
Santa Clara, CA  
Tel: 408-961-6444  
Fax: 408-961-6445

**Toronto**  
Mississauga, Ontario,  
Canada  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

**Asia Pacific Office**  
Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2401-1200  
Fax: 852-2401-3431

**Australia - Sydney**  
Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

**China - Beijing**  
Tel: 86-10-8528-2100  
Fax: 86-10-8528-2104

**China - Chengdu**  
Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

**China - Chongqing**  
Tel: 86-23-8980-9588  
Fax: 86-23-8980-9500

**China - Hong Kong SAR**  
Tel: 852-2401-1200  
Fax: 852-2401-3431

**China - Nanjing**  
Tel: 86-25-8473-2460  
Fax: 86-25-8473-2470

**China - Qingdao**  
Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205

**China - Shanghai**  
Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066

**China - Shenyang**  
Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

**China - Shenzhen**  
Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760

**China - Wuhan**  
Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

**China - Xian**  
Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

**China - Xiamen**  
Tel: 86-592-2388138  
Fax: 86-592-2388130

**China - Zhuhai**  
Tel: 86-756-3210040  
Fax: 86-756-3210049

### ASIA/PACIFIC

**India - Bangalore**  
Tel: 91-80-3090-4444  
Fax: 91-80-3090-4123

**India - New Delhi**  
Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

**India - Pune**  
Tel: 91-20-2566-1512  
Fax: 91-20-2566-1513

**Japan - Yokohama**  
Tel: 81-45-471- 6166  
Fax: 81-45-471-6122

**Korea - Daegu**  
Tel: 82-53-744-4301  
Fax: 82-53-744-4302

**Korea - Seoul**  
Tel: 82-2-554-7200  
Fax: 82-2-558-5932 or  
82-2-558-5934

**Malaysia - Kuala Lumpur**  
Tel: 60-3-6201-9857  
Fax: 60-3-6201-9859

**Malaysia - Penang**  
Tel: 60-4-227-8870  
Fax: 60-4-227-4068

**Philippines - Manila**  
Tel: 63-2-634-9065  
Fax: 63-2-634-9069

**Singapore**  
Tel: 65-6334-8870  
Fax: 65-6334-8850

**Taiwan - Hsin Chu**  
Tel: 886-3-6578-300  
Fax: 886-3-6578-370

**Taiwan - Kaohsiung**  
Tel: 886-7-213-7830  
Fax: 886-7-330-9305

**Taiwan - Taipei**  
Tel: 886-2-2500-6610  
Fax: 886-2-2508-0102

**Thailand - Bangkok**  
Tel: 66-2-694-1351  
Fax: 66-2-694-1350

### EUROPE

**Austria - Wels**  
Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393

**Denmark - Copenhagen**  
Tel: 45-4450-2828  
Fax: 45-4485-2829

**France - Paris**  
Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

**Germany - Munich**  
Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

**Italy - Milan**  
Tel: 39-0331-742611  
Fax: 39-0331-466781

**Netherlands - Drunen**  
Tel: 31-416-690399  
Fax: 31-416-690340

**Spain - Madrid**  
Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

**UK - Wokingham**  
Tel: 44-118-921-5869  
Fax: 44-118-921-5820

07/15/10