# AN75400 - PSoC® 3 and PSoC® 5LP

# CapSense® Design Guide

Doc. No 001-75400 Rev. *C

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

## Copyrights



### Trademarks



### Source Code



### Disclaimer

# Contents

# 1. Introduction

## 1.1 Abstract

This document provides guidance for designing CapSense® applications with the PSoC® 3 and PSoC® 5LP family of devices. It is intended for design engineers who are familiar with capacitive sensing technology and have chosen the PSoC 3 and PSoC 5LP family of devices for their applications. For a thorough understanding of CapSense technology see Getting Started with CapSense.

## 1.2 Cypress CapSense Documentation Ecosystem

Figure 1-1 and Table 1-1 summarize the Cypress CapSense documentation ecosystem. These resources enable you to quickly access the information you need to complete a CapSense product design. Figure 1-1 shows a typical product design cycle with capacitive sensing. This guide covers the topics highlighted in green. Table 1-1 provides links to supporting documents for each of the numbered tasks in Figure 1-1.

Figure 1-1. Typical CapSense Product Design Flow

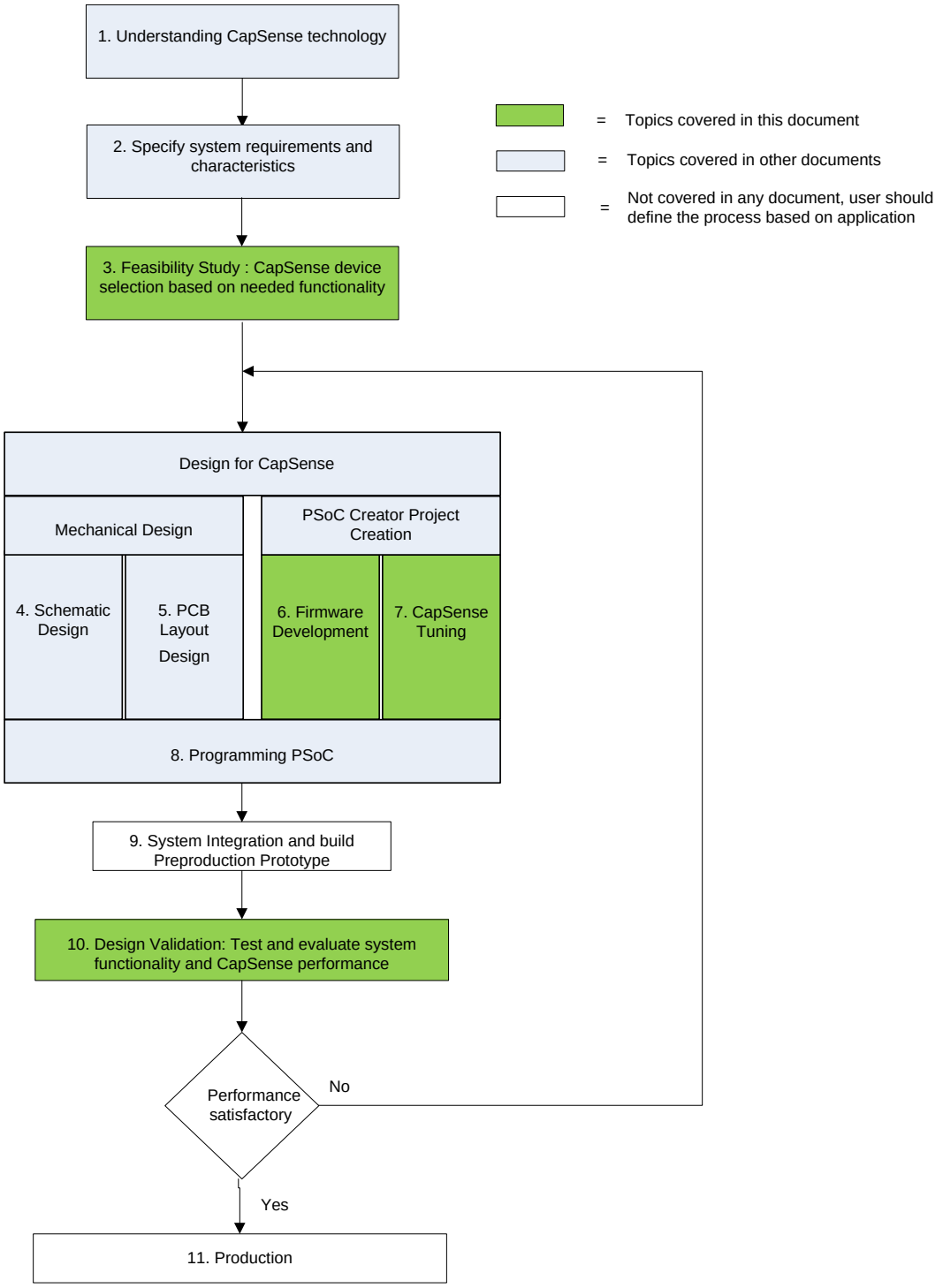


Table 1-1. Cypress Documents Supporting Numbered Design Tasks of Figure 1-1

| Numbered Design Task | Supporting Cypress CapSense Documentation | | |
|---|---|---|---|
| | Name | Section | Description |
| 1 | Getting Started with CapSense | 2 | In-depth theory of CapSense operation |
| 2 | Getting Started with CapSense | 4 and 5 | Describe device features and assist in selecting your device |
| | PSoC 3 Device Datasheets<br>PSoC 5LP Device Datasheets | - | |
| 3 | Getting Started with CapSense | 4 and 5 | Describe device features and assist in selecting your device |
| | This document | 3 | |
| 4 | Getting Started with CapSense | 3 | Provides information such as pin assignments and value for $C_{MOD}$ |
| | This document | 7 | |
| 5 | Getting Started with CapSense | 3 | Provides PCB layout guidelines |
| 6 | PSoC Creator Help Topics (available in the the PSoC Creator IDE under the Help tab) | - | The PSoC Creator Help Topics provide guidelines to use PSoC Creator IDE<br><br>This document and CapSense_CSD component datasheet provide firmware guidelines to develop CapSense applications |
| | This document | 4, 5, 6, and 7 | |
| | CapSense_CSD component datasheet | "Application Programming Interface" | |
| 7 | This document | 6 | Describes how to tune the CapSense system |
| 8 | MiniProg3 User Guide | - | Provide programming details for PSoC 3 and PSoC 5LP |
| | AN61290 - PSoC 3 / PSoC 5LP Hardware Design Considerations | "Programming and Debugging" | |
| 9 | N/A | - | - |
| 10 | Getting Started with CapSense | 3 | Provide important design considerations |
| | This document | 6 and 7 | |
| 11 | N/A | - | - |

## 1.3 PSoC 3 and PSoC 5LP Device Features

PSoC 3 and PSoC 5LP are programmable embedded system-on-chips that integrate configurable analog and digital peripheral functions, memory, and a microcontroller. These devices are highly flexible and can implement many functions in addition to CapSense. Their major features include:

**Device Features**

- 67 MHz 8051 CPU for PSoC 3
- 67 MHz ARM Cortex-M3 CPU for PSoC 5LP
- Up to 64 KB flash, 8 KB SRAM, and 2 KB EEPROM for PSoC 3
- Up to 256 KB Flash, 64 KB SRAM, and 2 KB EEPROM for PSoC 5LP
- 24 channel DMA
- Up to 72 I/O pins
- Low Power Modes
  - 1 µA sleep mode current for PSoC 3
  - 2 µA sleep mode current for PSoC 5LP
  - 200 nA hibernate mode current for PSoC 3
  - 300 nA hibernate mode current for PSoC 5LP
- Analog Functions
  - Configurable delta-sigma ADC with 8 to 20-bit resolution
  - Up to four comparators, opamps, DACs, and multi-function analog blocks
  - 0.1% internal bandgap voltage reference
- Digital Functions
  - Up to four 16-bit configurable timers, counters, and PWM blocks
  - Up to 24 programmable logic device (PLD) based universal digital blocks (UDB)
- Wide variety of packages: QFN, SSOP, and TQFP
- LCD direct drive from any GPIO, up to 46 × 16 segments
- $I^2C$, UART, full speed USB, SPI, and CAN communication interfaces

**CapSense Features**

- Supports a combination of CapSense buttons, sliders, matrix buttons, and proximity sensors
- Supports up to 61 capacitive sensors, CapSense support on all GPIO pins
- Integrated APIs to develop firmware
- Supports water proofing design
- Two channel design: Enough resources to scan two sensors at the same time
- SmartSense™ Auto-Tuning
  - Sets and monitors tuning parameters automatically at power-up and during run time
  - Adapts to changes in user interface design for design portability
  - Compensates for environmental changes during run time
  - Detects touches as low as 0.1 pF

## 1.4 Document Conventions

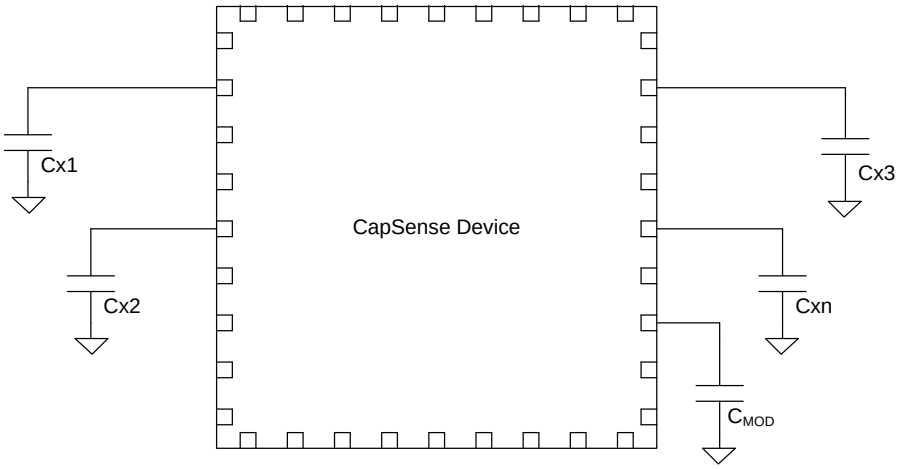
| Convention | Usage |
|---|---|
| Courier New | Displays file locations, user entered text, and source code:<br>C:\ ...cd\icc\ |
| Italics | Displays file names and reference documentation:<br>Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*. |
| [Bracketed, Bold] | Displays keyboard commands in procedures:<br>**[Enter]** or **[Ctrl] [C]** |
| File > Open | Represents menu paths:<br>File > Open > New Project |
| Bold | Displays commands, menu paths, and icon names in procedures:<br>Click the **File** icon and then click **Open**. |
| Times New Roman | Displays an equation:<br>2 + 2 = 4 |
| Text in gray boxes | Describes Cautions or unique functionality of the product. |

# 2. CapSense Technology

## 2.1 CapSense Fundamentals

CapSense is a touch sensing technology that works by measuring the capacitance of each I/O pin that has been designated as a sensor. The total capacitance on each of the sensor pins can be modeled as equivalent lumped capacitors with values of Cx1 through Cxn as shown in Figure 2-1. CapSense technology requires an external modulating capacitor, $C_{MOD}$. $C_{MOD}$ will be discussed in more detail in Capacitance Conversion.

Figure 2-1. CapSense Device Scanning Sensors Cx1 through Cxn




Each sensor I/O pin is connected to a sensor pad by traces, vias, or both, as necessary. A nonconductive overlay is required to cover each sensor pad and constitutes the system's touch interface. When a finger comes into contact with the overlay, the conductivity and mass of the body effectively introduces a grounded conductive plane parallel to the sensor pad. This action is represented in Figure 2-2. This arrangement constitutes a parallel plate capacitor, whose capacitance is given by the following equation:

$$C_F = \frac{\varepsilon_0 \varepsilon_r A}{D}$$ Equation 1

Where:

$C_F$ = The capacitance affected by a finger in contact with the overlay over a sensor

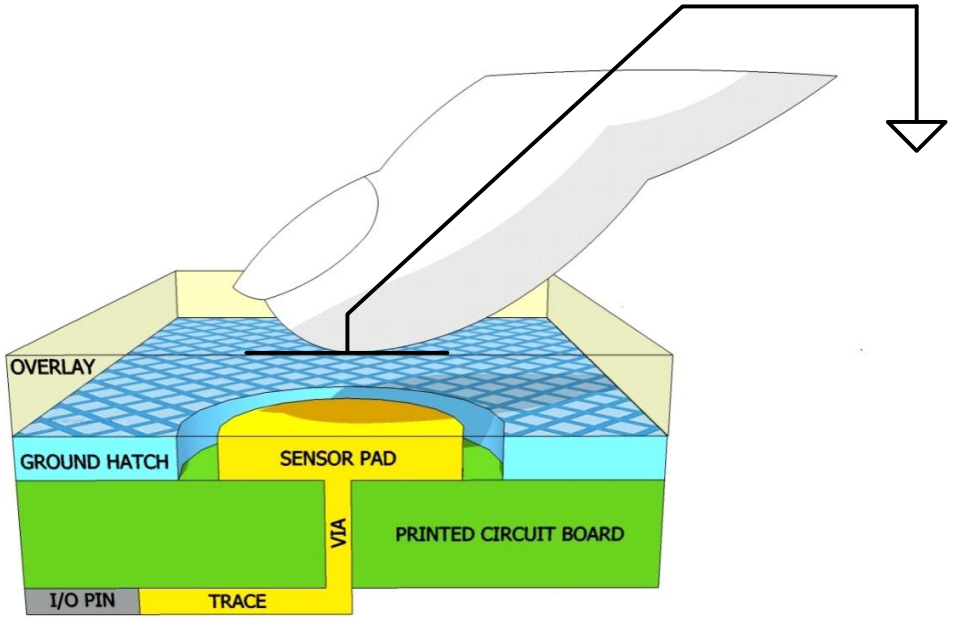
$\varepsilon_0$ = Free space permittivity

$\varepsilon_r$ = Dielectric constant of overlay

A = Area of finger and sensor pad overlap

D = Overlay thickness

Figure 2-2. CapSense System Equivalent Model




Even without a finger touching the overlay, the sensor input pin has some parasitic capacitance ($C_P$). $C_P$ results from the combination of the CapSense controller internal parasitic and electric field coupling among the sensor pad, traces, and vias, and other conductors in the system, such as ground plane, other traces, any metal in the product's chassis or enclosure, and so on. $C_P$ and $C_F$ are parallel to each other because they are both connected between the sensor pin and ground.

When a finger is not touching the sensor:

$$C_X = C_P \quad \text{Equation 2}$$

When a finger is touching the sensor:

$$C_X = C_P + C_F \quad \text{Equation 3}$$

In general, $C_P$ is an order of magnitude greater than $C_F$. $C_P$ usually ranges from 6—15 pF, but in extreme cases it can be as high as 45 pF. $C_F$ usually ranges from 0.1—0.4 pF. The magnitude of $C_P$ is critical when tuning a CapSense system. For optimal performance, $C_P$ should be kept as low as possible.

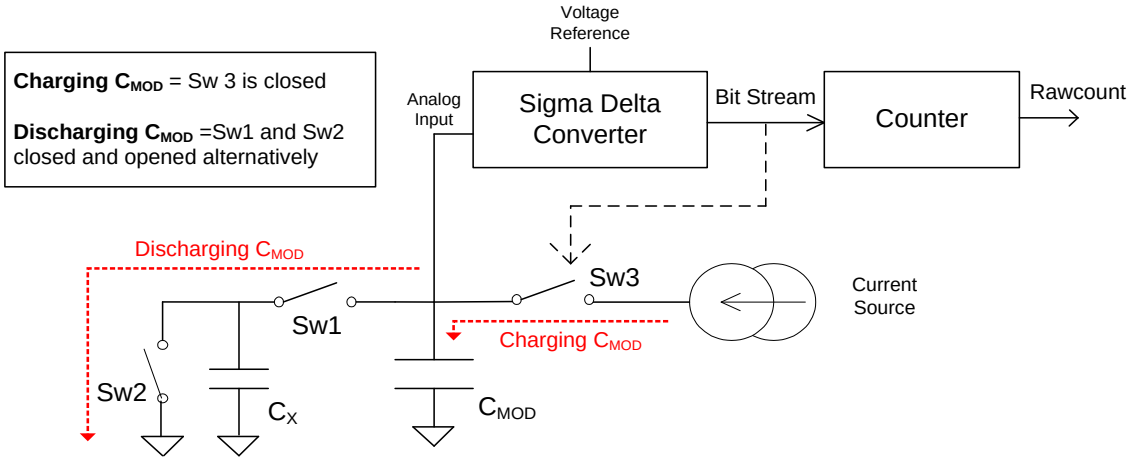
## 2.2 Capacitance Conversion

CapSense devices convert the magnitude of each Cx into digital counts which are stored and processed in order to detect the presence of a finger on or near a sensor pad.

### 2.2.1 CapSense Sigma Delta (CSD)

CapSense Sigma Delta (CSD) is one method for converting sensor capacitance into digital counts. The CSD method outperforms other sensing methods. Figure 2-3 shows a block diagram of the CSD method.

Figure 2-3. CapSense CSD Block Diagram



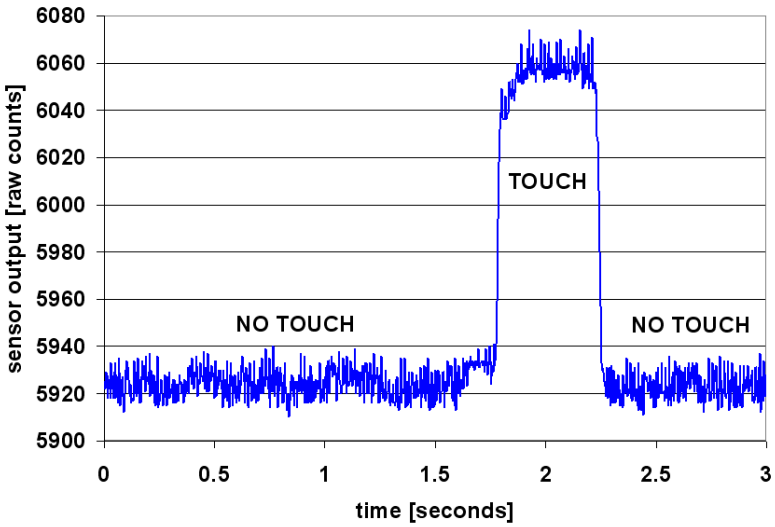

The CSD method requires a large integrating capacitor called a modulating capacitor ($C_{MOD}$). The sensor capacitor $C_X$ is connected to switches Sw1 and Sw2 to form a switched capacitor block. Sw1 connects $C_X$ to $C_{MOD}$, while Sw2 connects $C_X$ to ground. Sw1 and Sw2 open and close alternately without overlapping and provide a discharge path for $C_{MOD}$.

A sigma delta converter is used to convert the capacitance of $C_X$ into digital counts. A constant current source is connected to $C_{MOD}$ through a switch Sw3. The current source charges $C_{MOD}$ when Sw3 is closed. $C_{MOD}$ is connected to the sigma delta converter as an input. Based on its input, the sigma delta converter controls Sw3 such that it maintains the average voltage of $C_{MOD}$ at a reference voltage. The output of the sigma delta controller is a bit stream that represents the duty cycle of Sw3.

The duty cycle of Sw3 is directly proportional to the capacitance of $C_X$. For example, a higher value of $C_X$ increases the $C_{MOD}$ discharge current. To maintain $C_{MOD}$ voltage at the reference voltage, the sigma delta converter increases the duty cycle of Sw3.

the bit stream into a digital value known as raw count. The raw count is interpreted by a high-level algorithm to resolve the sensor's state. When a finger touches the sensor, $C_X$ increases by $C_F$, and raw counts increase proportionally. By comparing the shift in steady state raw counts to a predetermined threshold, the high-level algorithms can determine whether the sensor is in an ON (Touch) or OFF (No Touch) state. Figure 2-4 shows a plot of the raw counts from a number of consecutive scans during which the sensor is touched and then released by a finger.

Figure 2-4. Raw Count versus Time



For an in-depth discussion of Cypress's CSD sensing method, see PSoC 3, PSoC 5LP Architecture TRM.

### 2.2.2 CSD Implementations

There are various ways to implement the CSD method.

**IDAC Sourcing Method:** An IDAC charges $C_{MOD}$ and the sensor discharges $C_{MOD}$ as shown in Figure 2-3.

**IDAC Sinking Method:** An IDAC discharges $C_{MOD}$ and the sensor charges $C_{MOD}$.

**External Resistor Method:** An external resistor discharges $C_{MOD}$ and the sensor charges $C_{MOD}$.

All three implementations are discussed in Current Source Methods.

## 2.3 SmartSense Auto-Tuning

The hardware and firmware that make up a CapSense system have several parameters that determine how the system performs. Tuning these parameters is critical for proper system operation and a pleasant user experience. Unfortunately, tuning is time-consuming because it is an iterative process. In a typical development cycle, the interface is tuned in the initial design phase, during system integration, and before production ramp.

SmartSense Auto-Tuning is an advanced technology from Cypress. SmartSense is a sophisticated algorithm, which automatically optimizes system performance in a wide range of applications. It is easy to use and reduces design cycle time by eliminating manual tuning during the prototype and manufacturing stages. SmartSense Auto-Tuning tunes each CapSense button automatically at power up and maintains optimum button performance during runtime. It adapts for manufacturing variation in PCBs and overlays and automatically tunes out noise from sources such as LCD inverters, AC lines, and switch-mode power supplies.

SmartSense Auto-tuning is discussed in detail in Auto-Tuning.

# 3. CapSense in PSoC 3 and PSoC 5LP

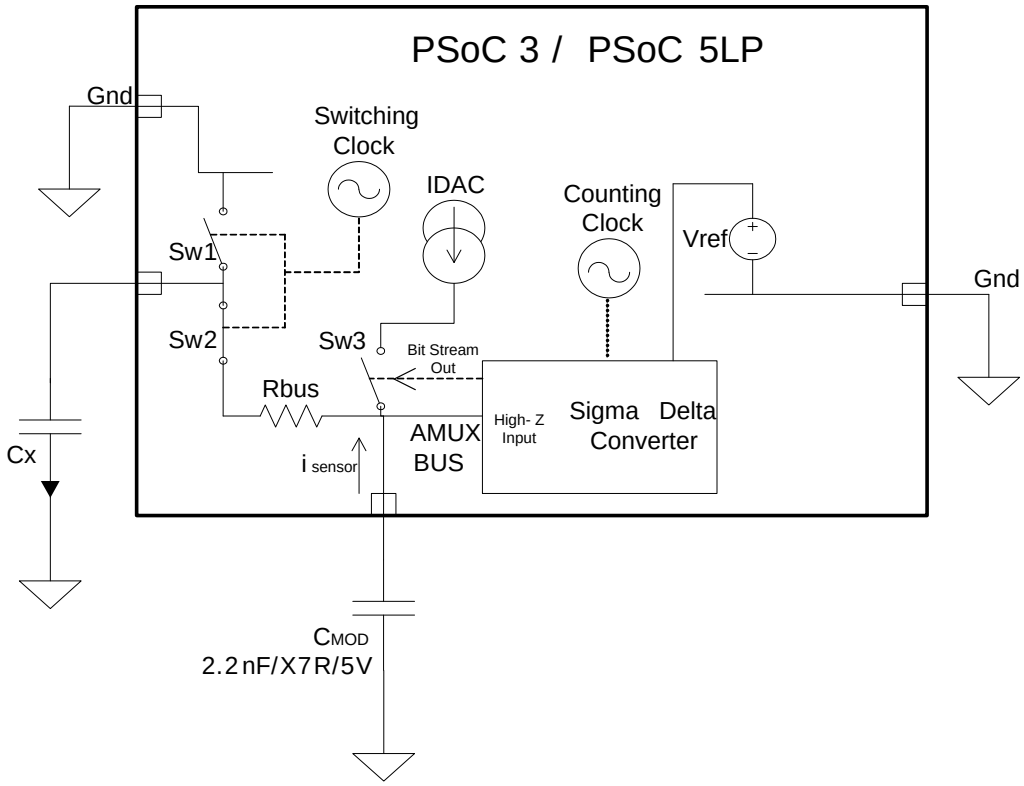
PSoC 3 and PSoC 5LP are programmable devices with a rich set of analog and digital resources. These devices implement CapSense using the CSD method with many unique features. Because these devices are so flexible and resourceful, they can perform many other system functions in addition to CapSense.

## 3.1 CSD Implementation

Refer to Capacitance Conversion for a detailed discussion of the CSD method. Figure 3-1 shows how the CSD method is implemented in the PSoC 3 and PSoC 5LP devices. These devices have up to four DAC resources. CapSense uses one of these to implement the current source (IDAC). The switches are controlled by a non-overlapping clock (Switching Clock). An analog bus (AMUXBUS) connects $C_{MOD}$, $C_{X,}$ IDAC, and the input of the sigma delta converter. The sigma delta converter controls the IDAC such that $C_{MOD}$ voltage remains close to the reference voltage (Vref). The sigma delta converter outputs a bit stream to a counter. The counter outputs the raw count. The CPU processes the raw count and determines the sensor's status.

Figure 3-1 shows the IDAC Sourcing method of implementing CSD. Other methods are explained in Current Source Methods.

Figure 3-1. CSD Implementation (IDAC Sourcing Method)



## 3.2 Unique CapSense Features

### 3.2.1 Two Channel Design

The PSoC 3 and PSoC 5LP devices are capable of scanning two sensors at a time. This cuts the scan time almost in half. Reducing scan time improves the response time for button ON/OFF detection and reduces average power consumption.

Using two channels requires twice the resources of a single channel, therefore, it is only recommended for designs with more than 20 sensors. Number of Channels explains how to select two channel design. Table 3-1 compares the resource requirements for one and two channel designs.

Table 3-1. Resource Comparison for One Channel and Two Channel Design

| Resource Type | One Channel | Two Channel |
|---|---|---|
| Analog Resources[1] | 1 or 2 AMUXBUS<br>1 Comparator<br>1 DAC | 2 AMUXBUS<br>2 Comparators<br>2 DACs |
| Digital Resources[2] | 4 Datapaths<br>19 Macrocells | 6 Datapaths<br>31 Microcells |
| External Components[3] | 1 $C_{MOD}$ capacitor<br>1 set of bleed resistors | 2 $C_{MOD}$ capacitors<br>2 sets of bleed resistors |

### 3.2.2 Water Tolerant Design

Some CapSense capacitive touch sensing systems require reliable operation in the presence of water. White goods, automotive applications, and industrial applications are examples of systems that must perform in environments that include water, ice, and humidity changes. For such applications shield electrodes and guard sensors can provide robust touch sensing.

If your application requires tolerance to water droplets and moisture, a shield electrode should be used. A shield electrode surrounds the sensor as shown in Figure 3-2. The shield electrode is connected to an I/O pin and is driven using the same switching signal as the sensor pin. Using the same signal to drive both the shield electrode and the sensor nullifies the capacitance between them. This means that any water film or droplets that partially cover the sensor and shield are effectively removed. The shield electrode also reduces the $C_P$ of the sensor. The number of I/O pins required depends on the board size and shield area. If the area of the shield electrode is large then it should be driven with multiple I/O pins.

If your application requires tolerance to water flow on the touch surface, a guard sensor along with a shield electrode should be used. The guard sensor should cover the entire touch sensing area. Typically the guard sensor surrounds the perimeter of the touch sensing area as shown in Figure 3-2. The guard sensor is scanned by the CapSense device in the same way as the other sensors. When water is present on the guard sensor, it becomes active and disables scanning of other sensors to avoid detecting a false finger touch.

Shield Electrode and Guard Sensor explains how to enable the shield electrode and guard sensors.

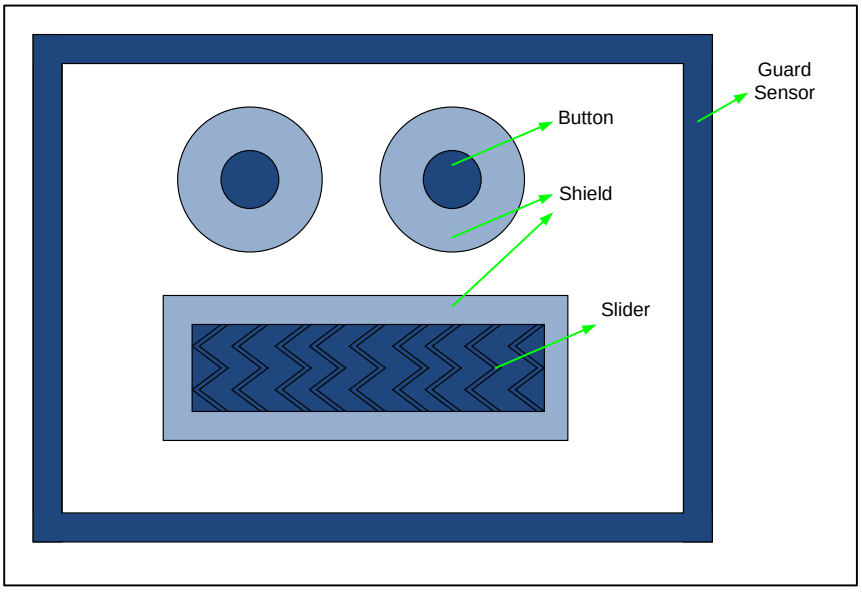
---

[1] There is a Left AMUXBUS and a Right AMUXBUS. One channel design requires only one AMUXBUS, and all sensors are placed on one side of the chip. Two channel design places sensors on both sides of the chip and shorts Left AMUXBUS and Right AMUXBUS together.
At least one DAC is required when is IDAC Sourcing or IDAC Sinking methods are selected. See Current Source Methods.

[2] Datapaths and macrocells are component blocks of universal digital blocks (UDB). There are up to 24 UDB blocks in PSoC 3 and PSoC 5LP devices.

[3] Bleed resistors are required when External Resistor method is selected. See Current Source Methods.

Figure 3-2. Shield Electrode and Guard Sensor




### *3.2.2.1 SIO Pins*

The PSoC 3 and PSoC 5LP devices have special input/output (SIO) pins, which feature a programmable “logic-high” level. This means that the logic-high level of an SIO is not fixed to $V_{DD}$ but can be programmed to a user defined voltage.

This feature is important when SIO pins are used as shield electrodes. A shield electrode is most effective when its signal matches the signal on the CapSense sensor. Using an SIO pin allows you to select a logic-high level that matches the CapSense sensor.

## 3.2.3 Current Source Methods

The PSoC 3 and PSoC 5LP devices allow you to select one of three different CSD implementations. Several factors influence which method will perform best in your design including resource requirements, noise susceptibility, and shield electrode requirements. For all of these methods the default value of Vref is equal to the bandgap voltage, 1.024 V.

Table 3-2 compares the three implementation methods. Note that there are up to four DACs in the PSoC 3 and PSoC 5LP devices depending on the part number. If other functionalities in your design are using all the DACs, the External Resistor method can be used for CapSense. Current Source describes how to select the current source method.

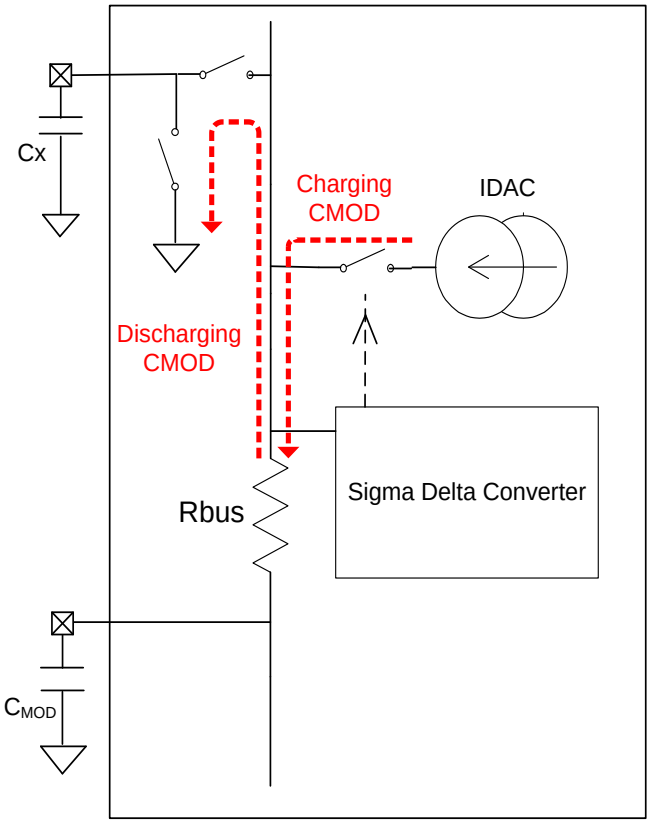
### *3.2.3.1 IDAC Sourcing Method*

In this configuration, the CapSense sensor switches between $C_{MOD}$ and GND continuously, discharging $C_{MOD}$. The IDAC is configured in source mode and is switched on when $C_{MOD}$ voltage drops below Vref. Figure 3-3 shows the block diagram for the IDAC Sourcing method.

This method is susceptible to finger conducted noise because the voltage swing on the sensor is small, between Vref and GND. Increasing Vref with a VDAC will improve noise immunity, but requires an extra DAC resource.

When using the IDAC Sourcing method SIO pins should be used for shield electrodes. This allows you to switch the shield electrode between Vref and GND, ensuring it gets the same signal as the sensor.

Figure 3-3. IDAC Sourcing Method

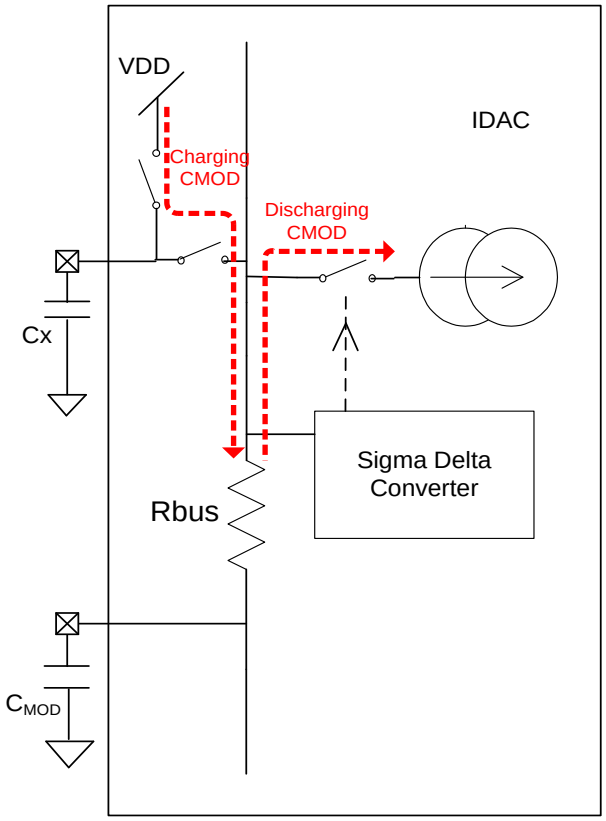


### *3.2.3.2 IDAC Sinking Method*

In this configuration, the CapSense sensor switches between $V_{DD}$ and $C_{MOD}$ continuously, charging $C_{MOD}$. The IDAC is configured in sink mode and switched on when $C_{MOD}$ voltage exceeds Vref. Figure 3-4 shows the block diagram for the IDAC Sinking method.

This method is susceptible to power supply noise because the sensor switches between $V_{DD}$ and Vref.

Figure 3-4. IDAC Sinking Method



### 3.2.3.3 External Resistor Method

This configuration is similar to the IDAC Sinking method except it uses an external resistor (bleed resistor) to discharge $C_{MOD}$ in place of the IDAC. The CapSense sensor switches between $V_{DD}$ and $C_{MOD}$ continuously, charging $C_{MOD}$. The bleed resistor is connected to ground when $C_{MOD}$ voltage exceeds Vref. Figure 3-5 shows the block diagram for the External Resistor method.

This method is susceptible to power supply noise because the sensor switches between $V_{DD}$ and Vref.

Figure 3-5. External Resistor Method

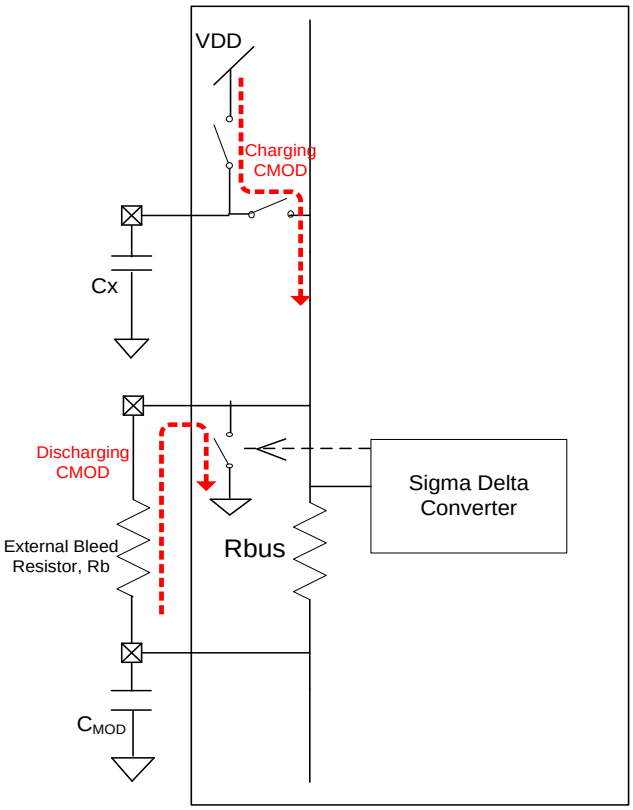



Table 3-2. Comparison of Current Source Methods

| | IDAC Sourcing | IDAC Sinking | External Resistor |
|---|---|---|---|
| DAC Resources | Required | Required | Not Required |
| External Bleed Resistor | Not Required | Not Required | Required |
| Power Supply Noise | Not susceptible | Susceptible | Susceptible |
| Finger Conducted Noise | Susceptible<br>Vref should be increased using VDAC | Less susceptible | Less susceptible |
| Shield Electrode | SIO pin | Any GPIO pin | Any GPIO pin |

## 3.2.4 Tuning

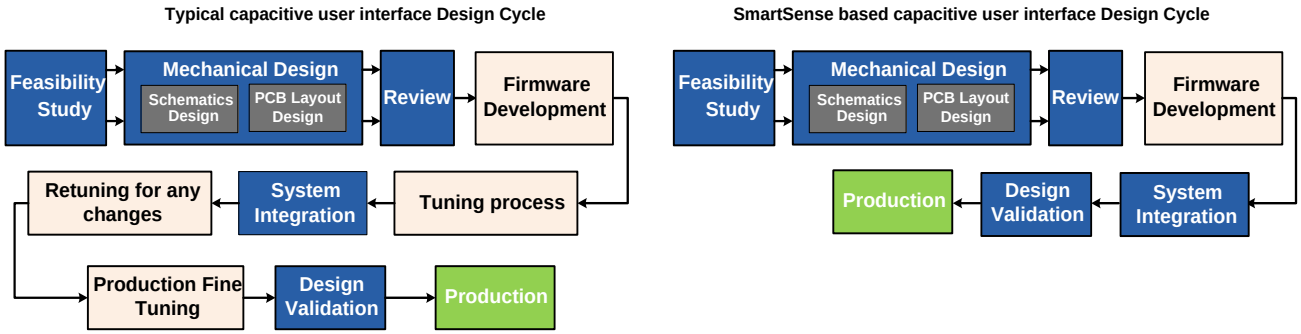
SmartSense Auto-Tuning is an advanced technology that tunes each CapSense button automatically at power up and maintains optimum button performance during runtime. It reduces design cycle time and takes care of process variations. You can also manually tune the CapSense parameters if you require more control over the parameters or if $C_P$ is high. Tuning Method explains how to select the tuning method. CapSense Performance Tuning describes the process for both auto and manual tuning.

### *3.2.4.1 Auto-Tuning Features*

**Reduced Design Cycle Time**

Figure 3-6 illustrates how SmartSense Auto-Tuning can significantly reduce design cycle time.

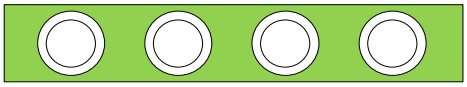
Figure 3-6. Design Cycle Comparison




The following example shows how SmartSense Auto-Tuning saves significant time and makes platform designs possible. Figure 3-7 shows multimedia keys for a 21-inch laptop model and Figure 3-8 shows multimedia keys for a 15-inch model. The keys are CapSense buttons with the same functionality and size. However, the design cannot be directly ported across models because the buttons on the 21-inch model have wider spacing and longer traces between the buttons and CapSense controller. The design would need to be retuned. SmartSense Auto-Tuning makes it possible for the developers to port the same design to other models saving significant time.

Figure 3-7. Laptop Multimedia Keys for a 21-inch Model

Figure 3-8. Laptop Multimedia Keys for a 15-inch Model (Identical Functionality and Button Size)



**Immunity to Process Variations**

$C_P$ can vary due to PCB layout and trace length, PCB manufacturing process variation, or vendor-to-vendor PCB variation within a multi-sourced supply chain. The sensitivity of a button depends on $C_P$; higher $C_P$ values decrease sensitivity, resulting in decreased finger touch signal amplitude. A change in $C_P$ can result in a button becoming too sensitive, not sensitive enough, or non-operational. When this happens, you must retune the system and, in some cases, re-qualify the user interface subsystem. SmartSense Auto-Tuning resolves these issues.

**Ease of Use**

SmartSense Auto-Tuning allows you to quickly and easily design a CapSense application without an in-depth knowledge of all of the parameters.

### *3.2.4.2 Manual Tuning Features*

**More Control**

Manual tuning gives you the ability to select each of the CapSense parameters. This is important for systems with extraordinary operating conditions, such as systems with high noise. Using auto-tuning in noisy systems can lead to false button touches. Manually increasing the finger threshold makes the system more immune to noise.

**High Parasitic Capacitance**

Auto-tuning is designed to work with $C_P$ values in the range of 5—45 pF. If $C_P$ is higher than 45 pF due to long traces or large button sizes, manual tuning should be used.

**Failure Detect Algorithms**

Auto-tuning can change the parameters at every start up and during run time. This can make it difficult to write a failure detect algorithm.

Consider a failure detect algorithm that checks whether the baseline counts are in the range of a value stored in flash at the factory. Assume the stored value is 3000 counts measured at a scan resolution of 12-bits. Baseline counts are the average counts for $C_P$, and scan resolution is the resolution of the capacitance measurement (see Raw Count and Scan Resolution for a more detailed explanation of these terms). When auto-tuning is used, it may set the scan resolution parameter to 13-bits due to a change in physical or environmental conditions. When the scan resolution is set to 13-bits, the baseline becomes 6000 counts, and the system will fail.

**Less RAM and Flash Usage**

The auto-tuning algorithm requires more flash and RAM. Table 3-3 shows the memory requirements for a CapSense design with four sensors.

Table 3-3. Comparison of Memory Requirement for Manual Tuning and Auto-Tuning

| Tuning Method | PSoC 3 | | PSoC 5LP | |
|---|---|---|---|---|
| | RAM (bytes) | Flash (bytes) | RAM (bytes) | Flash (bytes) |
| Manual tuning | 207 | 6345 | 384 | 5104 |
| Auto-tuning | 292 | 8282 | 488 | 6152 |

## 3.2.5 Non-Blocking Architecture

CapSense scanning has three phases:

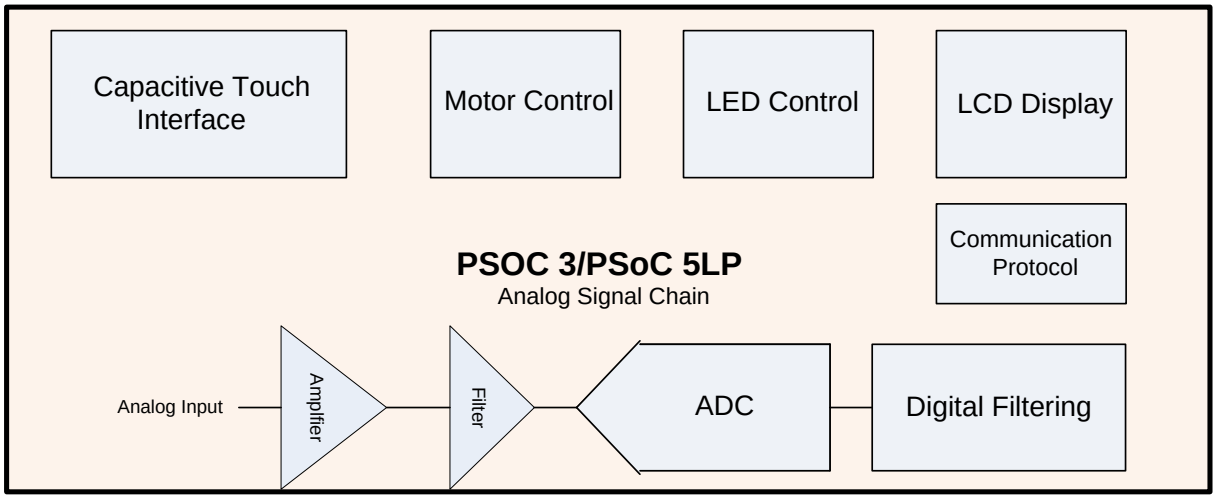
1. **Prescan**: Preparing hardware
2. **Hardware Scan**: Actual hardware scanning
3. **Postscan**: Processing and storing the result

Only phases one and three require the CPU to execute code. The code architecture does not wait until the second phase completes. Instead, the CPU executes the next line of code and as soon as the hardware completes the scan and generates an interrupt, it executes the post-scan code in an ISR. This non-blocking architecture is useful in designs where the CPU serves multiple applications.

## 3.3 CapSense PLUS

CapSense PLUS refers to PSoC devices that perform functions in addition to CapSense. The PSoC 3 and PSoC 5LP devices fall into this category. The wide variety of features offered by these devices allow you to integrate numerous system functions in a single chip as shown in Figure 3—9. This reduces board size, BOM cost, and power consumption.

Figure 3-9. CapSense PLUS Other Functionalities in the System




Visit the following links to learn more:

- Analog Functions
- Communication Modems
- LCD Drive
- Low Power
- USB Connectivity

Visit the following links to see successful designs using PSoC 3 and PSoC 5LP devices:

- Blood Glucose Meter
- Blood Pressure Monitor
- Fertility Monitor
- Infusion Pump
- iPod, iPhone and iPad Accessories
- LED Projector
- Magnetic Card Reader
- Pulse Oximeter
- 3D Active Shutter Glasses

# 4. CapSense Design Tools

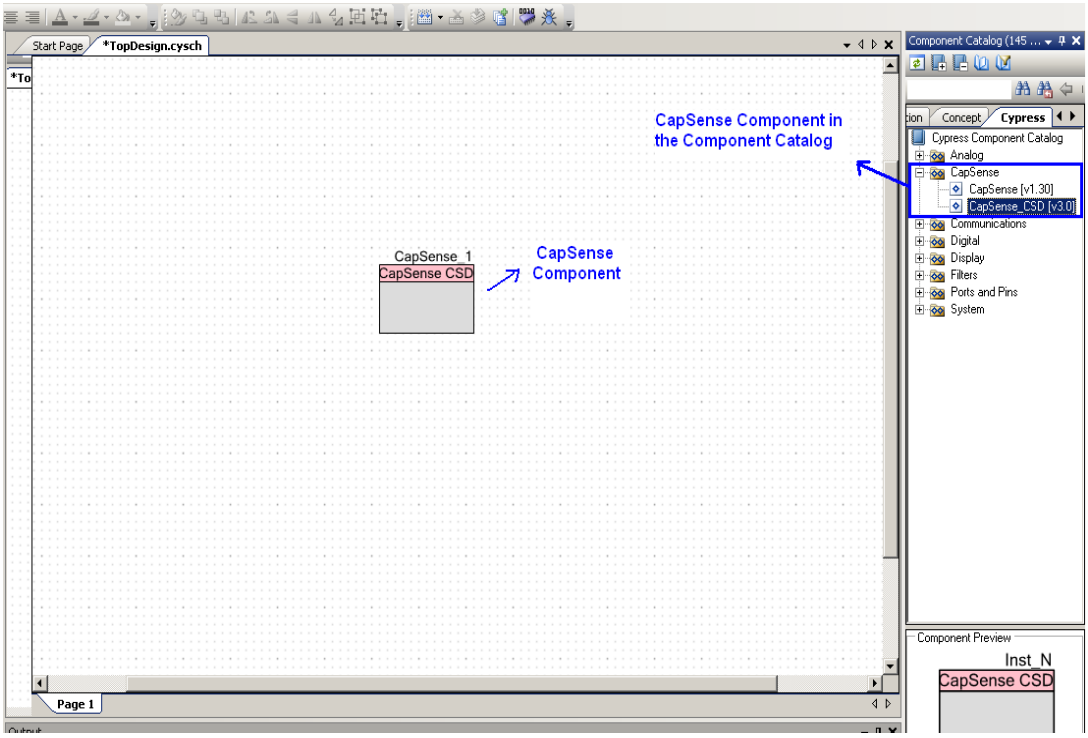
Cypress offers a full line of hardware and software tools for developing your CapSense capacitive touch sensing application. See Resources for ordering information.

## 4.1 PSoC Creator

Cypress's PSoC Creator provides an integrated design environment. PSoC Creator offers a unique combination of hardware configuration and software development in a single, unified tool. Applications are developed in a drag-and-drop design environment using a library of components.

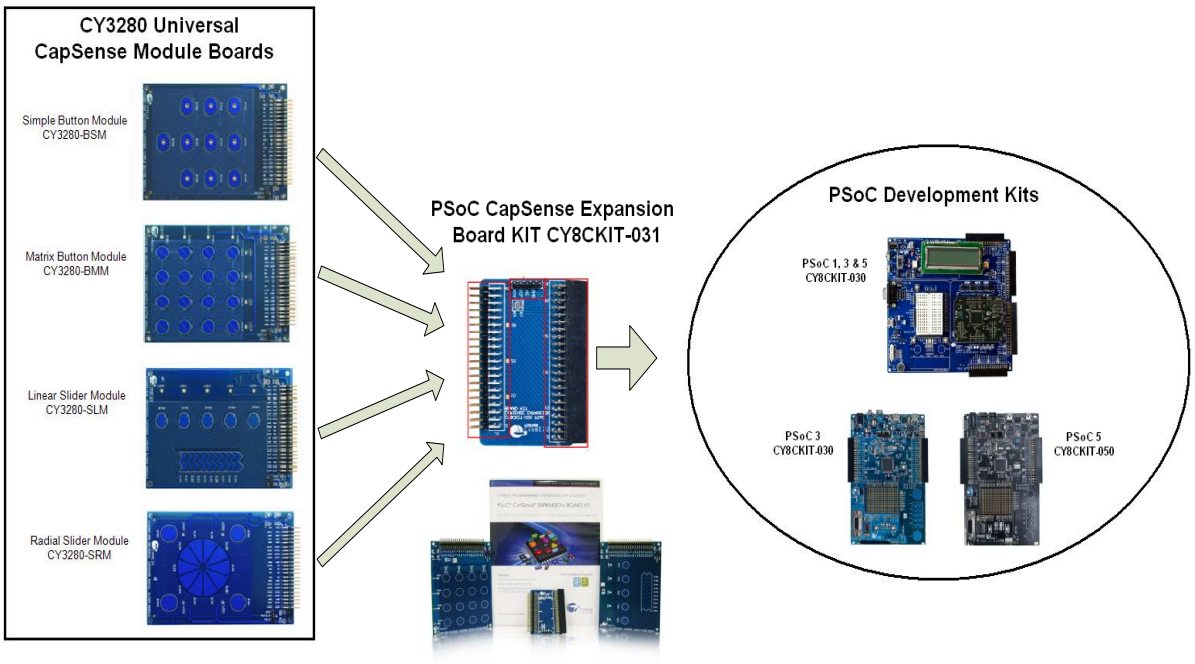
PSoC Creator provides a CapSense_CSD component. This component implements an array of capacitive touch sensors using switched-capacitor circuitry, an analog bus, a comparator, digital counting functions, and high-level software routines (APIs). There are other analog and digital components available to implement additional functionalities such as $I^2C$, SPI, UART, timers, PWMs, amplifiers, ADCs, and LCDs. Figure 4-1 shows a CapSense_CSD component being dragged from the component catalog and placed on TopDesign.

Figure 4-1. PSoC Creator TopDesign



## 4.2 Hardware Kits

Figure 4-2. CapSense Hardware




### 4.2.1 PSoC 3 and PSoC 5LP Development Kits

The PSoC 3 and PSoC 5LP development kits support CapSense functionality.

- CY8CKIT-001 PSoC® Development Kit
- CY8CKIT-030 PSoC® 3 Development Kit
- CY8CKIT-050B PSoC® 5LP Development Kit

### 4.2.2 Universal CapSense Module Boards

Cypress provides Universal CapSense module boards which feature a variety of CapSense sensors, LEDs, and interfaces to meet your needs.

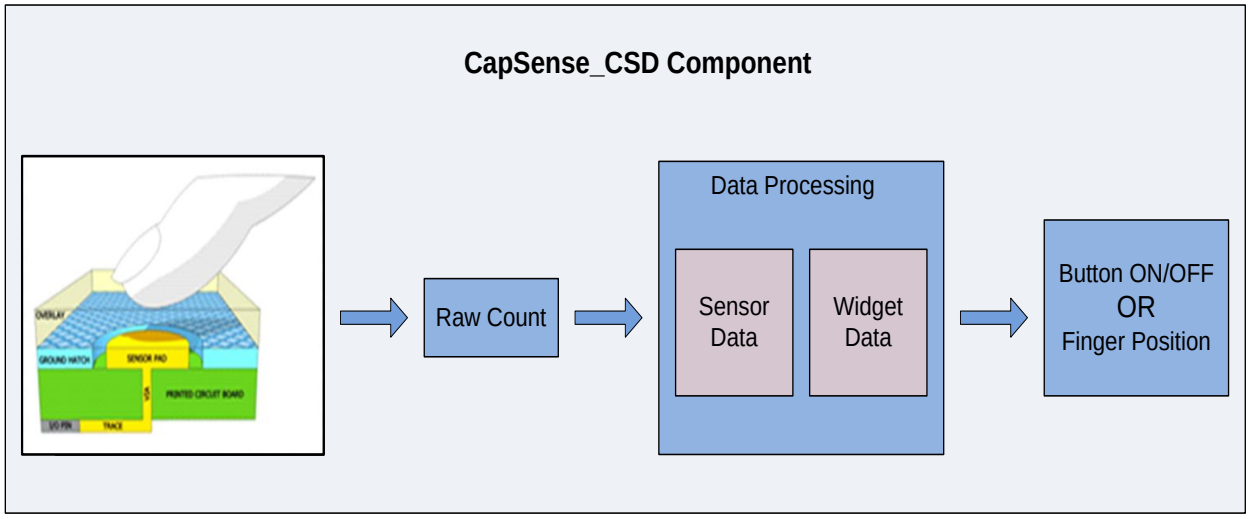
- CY3280-BSM Simple Button Module
- CY3280-BMM Matrix Button Module
- CY3280-SLM Linear Slider Module
- CY3280-SRM Radial Slider Module
- CY3280-BBM Universal CapSense Prototyping Module

### 4.2.3 PSoC CapSense Expansion Board Kit

The CY8CKIT-031 PSoC CapSense Expansion Board Kit connects any of the PSoC 3 and PSoC 5LP Development Kits to any of the Universal CapSense Module Boards. The CY8CKIT-031 provides and interface board and two module boards, CY3280-SLM and CY3280-BMM.

# 5. CapSense_CSD Component

Figure 5-1. CapSense_CSD Component Block Diagram



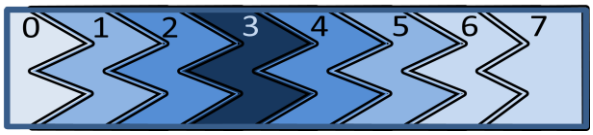

The CapSense_CSD component provides a complete CapSense system. The component has several parameters classified as high-level or low-level. The parameters communicate with each other in firmware using global arrays.

High-level parameters control how the CapSense system processes the raw counts into useful information such as sensor ON/OFF state. Examples of high-level parameters include finger threshold, noise threshold, and debounce count. See High-Level Parameters.

Low-level parameters control how the CapSense system operates at the physical layer. At the physical layer, the capacitance is converted into raw counts. Examples of low-level parameters include IDAC range, IDAC value, and scan clock. See Low-Level Parameters.

The CapSense_CSD component provides several different types of touch interfaces called widgets. Widgets are comprised of one or more sensors. They represent an interface object such as a slider or a touch pad. The widget types include button, slider, radial slider, matrix button, touchpad, and proximity sensor. Figure 5-2 shows an example of a slider. To form this slider widget you would place seven sensors in a straight line with a small gap between them.

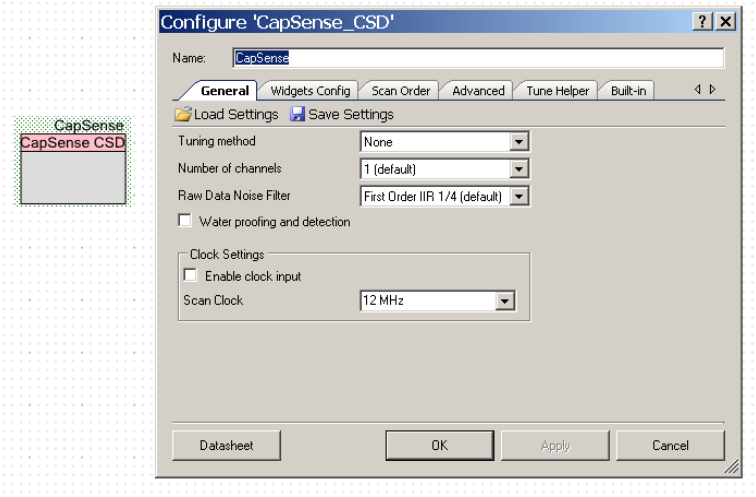
Figure 5-2. A Slider with 7-Segments



## 5.1 Parameter Summary

Figure 5-3 shows a screen shot of the CapSense_CSD component from PSoC Creator as well as the configuration window. You can open the configuration window by double clicking on the component or by right clicking and selecting “Configure”.

Figure 5-3. PSoC Creator CapSense Component




The configuration window has several parameters arranged under different tabs. Table 5-1 summarizes the parameters available in this window and provides relevant links.

Table 5-1. CapSense_CSD Component Configuration Window Parameters

| General | Widget Config | Scan Order | Advanced | | Tune Helper |
|---|---|---|---|---|---|
| Tuning Method | Add Widget | Analog Switch Divider | Analog Switch Drive Source | Shield | Enable Tune Helper |
| Number of Channels | Finger Threshold | IDAC Value | Multiple Analog Switch Divider | Inactive Sensor Connection | Instance Name for EZI2C Component |
| Raw Data Noise Filter | Noise Threshold | Move to Channel 1/ Move to Channel 0 | Analog Switch Divider | Guard Sensor | |
| Water Proofing and Detection | Hysteresis | Sensitivity | Scan Speed | Current Source | |
| Enable Clock Input | Debounce | | PRS EMI Reduction | IDAC Range | |
| Scan Clock | Scan Resolution | | Sensor Auto Reset | Number of Bleed Resistors, channel 0/ channel 1 | |
| | Number of Sensor Elements | | Widget Resolution | Digital Resource Implementation, channel 0/ channel 1 | |
| | API Resolution | | Negative Noise Threshold | Voltage Reference Source | |
| | Position Noise Filter | | Low Baseline Reset | | |
| | No of Dedicated Sensor Elements | | | | |

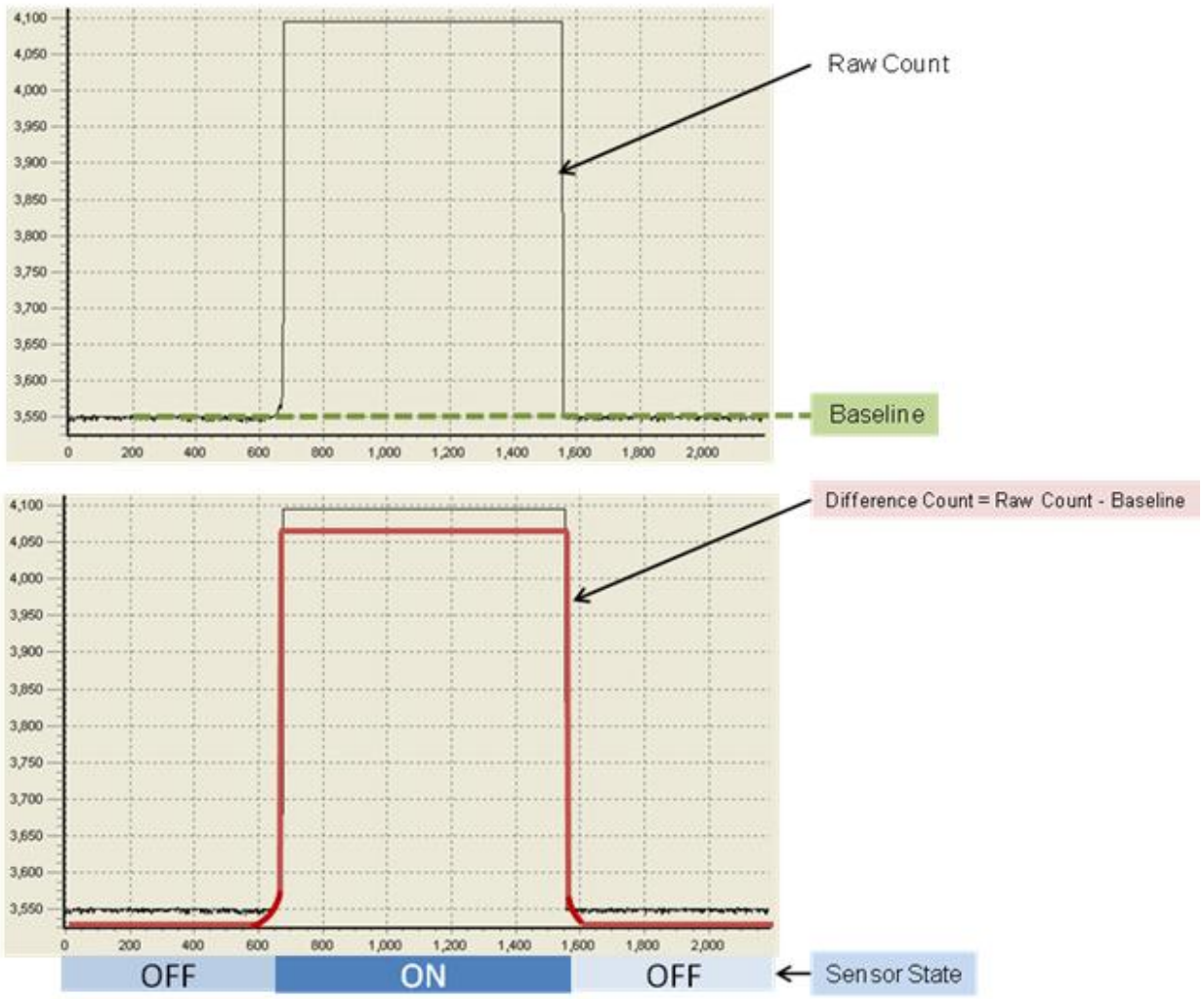
| Low-Level Parameters |
|---|

| High-Level Parameters |
| --- |
| Widget Parameters |
| Other System Parameters |

## 5.2 Global Arrays

The CapSense system uses several global arrays to ensure proper detection and operation in presence of environmental changes. These arrays should not be altered manually, but may be inspected for debugging purposes.

Figure 5-4. Global Parameters




### 5.2.1 Raw Count

The CapSense controller hardware measures the capacitance and provides the result in a digital form called raw count. The value of raw count increases as sensor capacitance increases.

### 5.2.2 Baseline

The raw count value of a sensor varies gradually due to changes in the environment such as temperature and humidity. These gradual variations are compensated for with the baseline values. The baseline keeps track of gradual changes in raw count

using a software algorithm. It is a low-pass filter that is less sensitive to sudden changes in the raw count. The baseline values provide the reference level for computing the difference counts.

### 5.2.3 Difference Count

The difference count is the difference between the raw count and the baseline of the sensor. Usually, the difference count is zero when the sensor is inactive. When the sensor is touched, it causes the raw count to increase, and results in a positive difference count value.

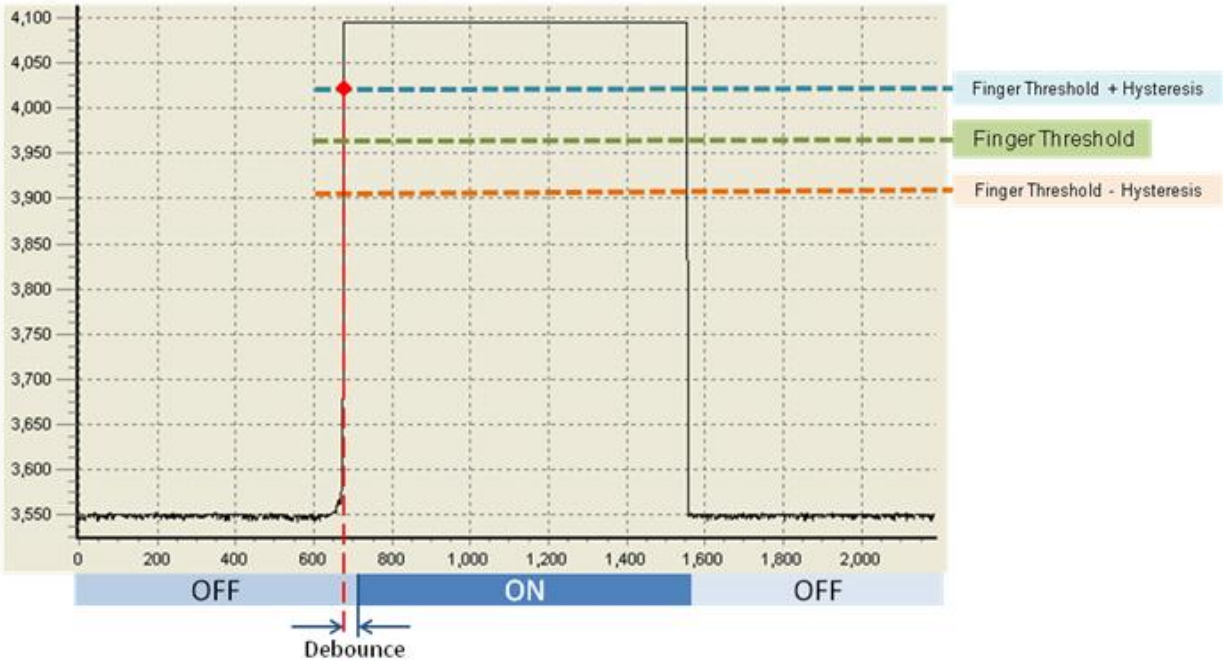
### 5.2.4 Sensor State

The state of each sensor is represented as 1 if the button is ON and 0 if the button is OFF. The ON/OFF states for all of the sensors are stored in a byte array. Each array element can hold the ON/OFF state of eight sensors.

## 5.3 High-Level Parameters

High-level parameters define how the raw counts are processed to produce information such as sensor ON/OFF state, and estimated finger position on a slider. Figure 5-5 and Equation 4 give an overview of the high-level parameters.

Figure 5-5. High-Level Parameters




Equation 4

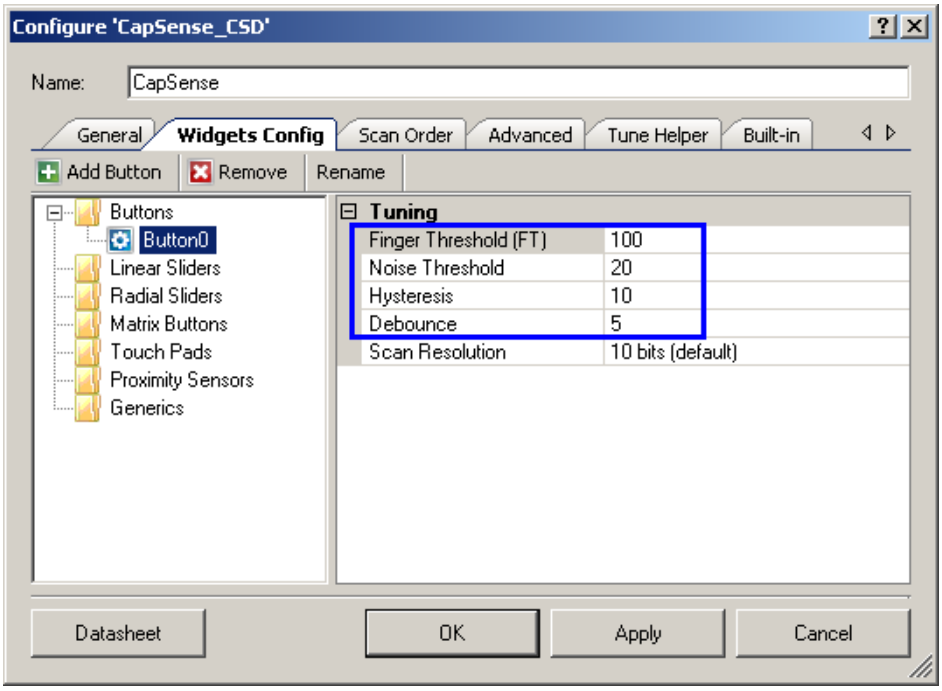
$$if\ (Difference\ Count \geq Finger\ Threshold\ + Hysteresis)\ \&\ (Sample\ Count\ \geq Debounce), \quad Sensor\ State = ON$$

$$if\ (Difference\ Count\ \leq Finger\ Threshold - Hysteresis), \quad Sensor\ State = OFF$$

Where:

Sample Count = the number of samples measured above Finger Threshold + Hysteresis

Figure 5-6. High-Level Parameters




### 5.3.1 Finger Threshold

The finger threshold parameter defines the sensitivity of the sensor to finger touches. It is used in conjunction with the Hysteresis parameter to determine the sensor state, as defined in Equation 4. Possible values are 0 to 255.

### 5.3.2 Hysteresis

The Hysteresis parameter is used in conjunction with the finger threshold to determine sensor state, as defined in Equation 4. Hysteresis adds immunity to noisy transitions. This is a debounce feature of a button. The touch state stays off until the difference counts are a little higher than the finger threshold. The touch state stays on until the difference counts are a little lower than the noise threshold. This prevents the touch/no touch state machine from triggering if the difference counts are noisy and are halfway between noise and finger thresholds.

Possible values are 0 to 255. The Hysteresis parameter setting must be lower than the Finger Threshold parameter setting.

### 5.3.3 Debounce

This Debounce parameter adds a counter to the sensor transition from OFF to ON. For the sensor to transition from OFF to ON, the difference count value must stay above the finger threshold plus hysteresis level for the number of samples specified.

Possible values are 1 to 255. A setting of 1 provides no debouncing.

### 5.3.4 Noise Threshold

For individual sensors, the Noise Threshold parameter sets the upper raw count limit for updating the baseline value. For slider sensors, it sets the lower raw count limit for counting results in the centroid calculation.

Possible values are 3 to 255. The noise threshold value should never be set to higher than Finger Threshold minus Hysteresis value for proper operation of the user module.

### 5.3.5 Negative Noise Threshold and Low Baseline Reset

Figure 5-7. Noise Threshold and Baseline Update Parameters

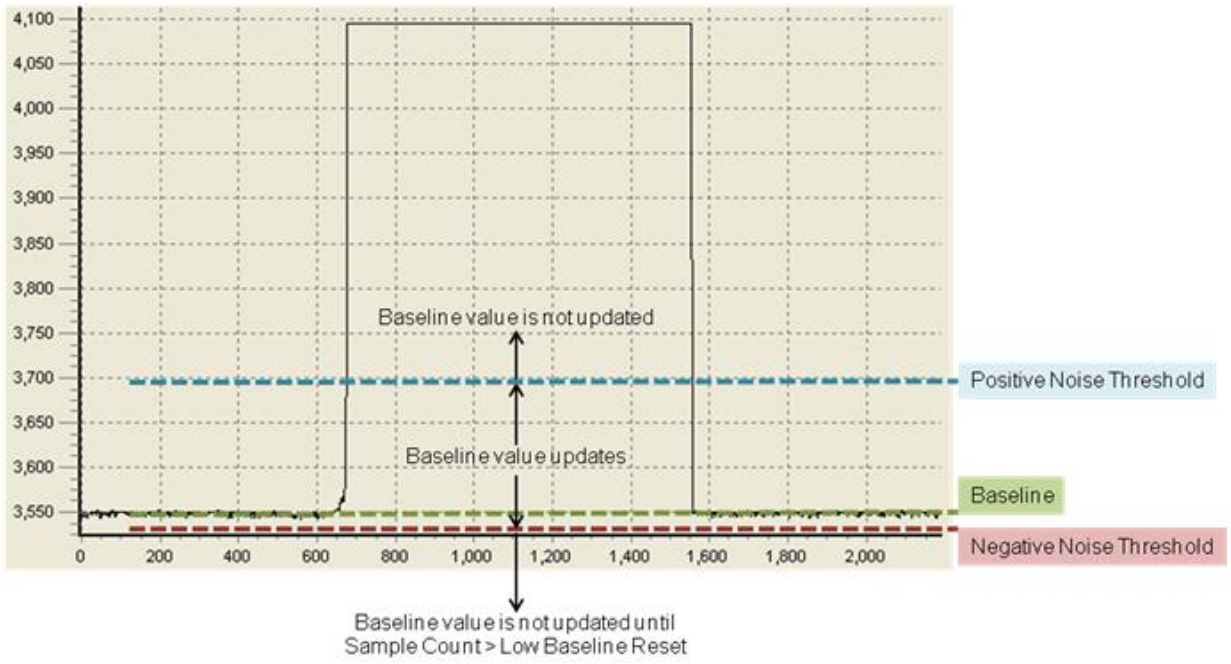
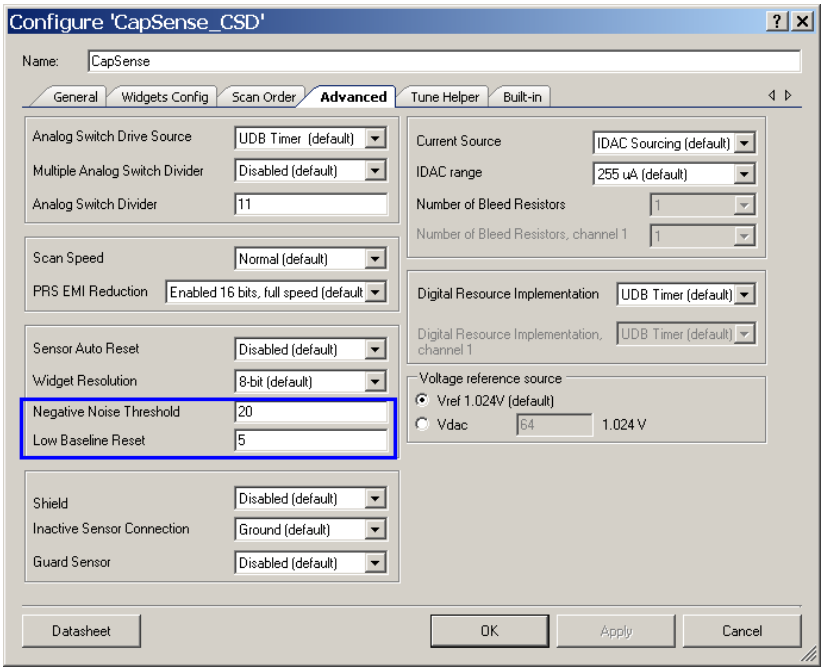



Figure 5-8. Negative Noise Threshold and Low Baseline Reset Parameters

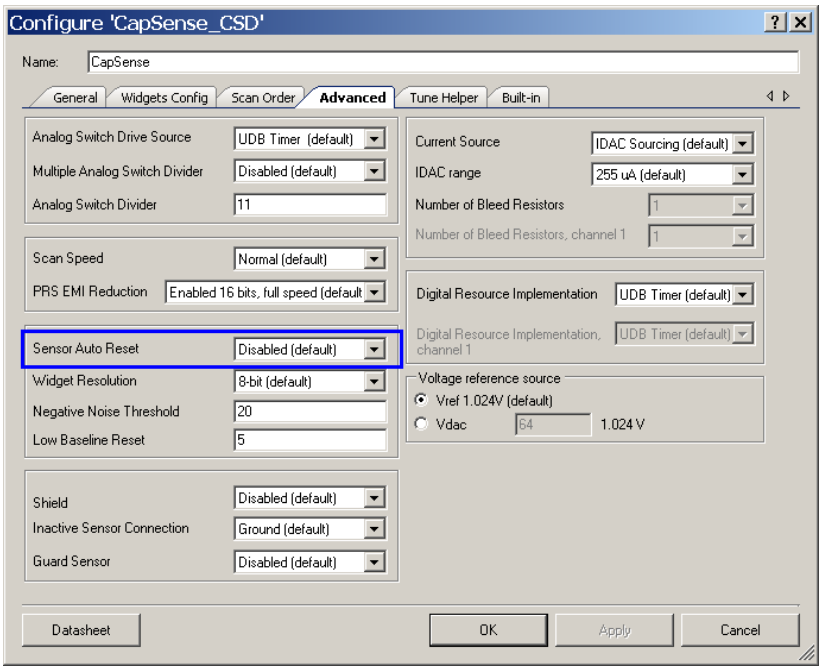



The Negative Noise Threshold parameter acts as a negative difference count threshold. If the raw count is below the baseline minus the negative noise threshold for the number of samples specified by the Low Baseline Reset parameter, the baseline is set to the new raw count value. Possible values are 0 to 255.

The Low Baseline Reset parameter works together with the Negative Noise Threshold parameter. It counts the number of abnormally low samples required to reset the baseline. It is used to correct the finger-on-at-startup condition. Possible values are 0 to 255.

### 5.3.6 Sensor Auto Reset

Figure 5-9. Sensor Auto Reset Parameter



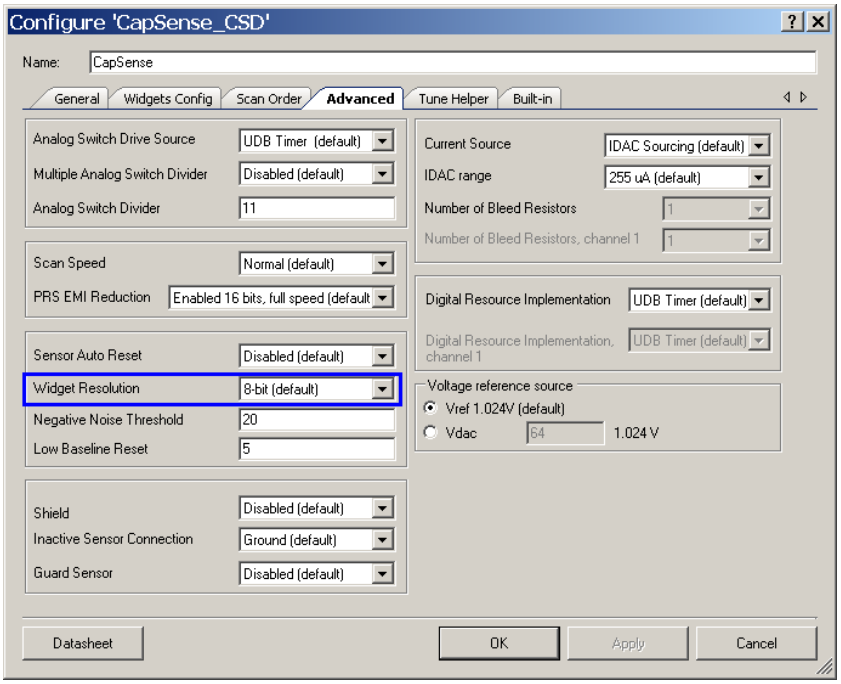

This parameter determines whether the baseline is updated at all times, or only when the difference counts are below the noise threshold.

**Enabled** - The baseline is updated constantly. This limits the maximum time duration of the sensor (typical values are 5 to 10 seconds), but prevents the sensors from permanently turning on when the raw count accidentally rises without anything touching the sensor. This sudden rise can be caused by a large power supply voltage fluctuation, a high-energy RF noise source, or a quick temperature change.

**Disabled** - The baseline is updated only when the difference counts are below the Noise Threshold parameter.

### 5.3.7 Widget Resolution

Figure 5-10. Widget Resolution Parameter



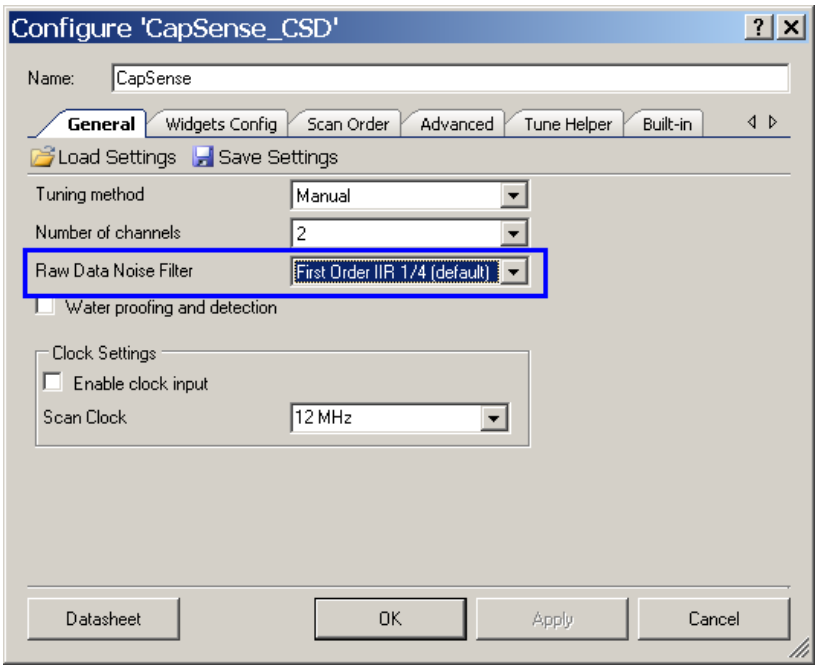

This parameter decides whether to use 8-bit or 16-bit variables for signal count. Signal count is the difference between raw counts and baseline as shown in Figure 5-5. This parameter is set at the widget level. All of the sensors in a widget will have the same resolution.

In most cases the default value of 8-bit is appropriate. However, in designs with high signal levels due to thin overlays or large sensor areas 16–bit resolution should be selected. Because sliders calculate finger position using relative signal levels of adjacent sensors, saturation causes non-smooth operation and 16-bit resolution may be required.

## 5.3.8 Filter Selection

### *5.3.8.1 Raw Data Noise Filter*

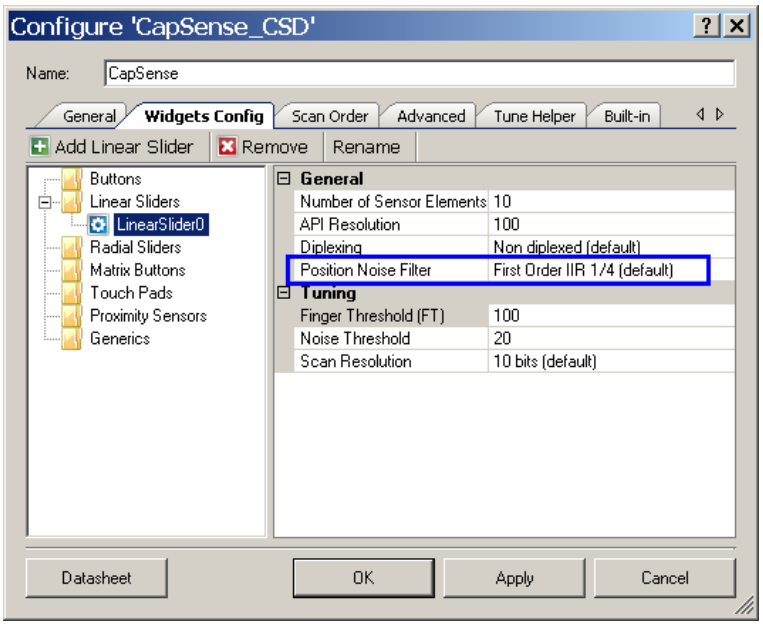
Figure 5-11. Raw Data Noise Filter Parameter




Raw count is the raw output of the CapSense system and has inherent noise. The noise on raw counts can be minimized by using software filters. Raw Data Noise Filter parameter allows you to select from several software filters to clean up the raw counts and finger-position data. See Software Filtering to understand the use of each filter type.

#### *5.3.8.2 Position Noise Filter*

Figure 5-12. Position Noise Filter Parameter




Some widget types such as linear slider and radial slider have finger position output. The Position Noise Filter parameter allows you to select from several software filters to clean up the finger position output.

### 5.3.9 High-Level Parameter Recommendations

The following recommendations are a starting place for selecting the optimal parameter settings

- **Finger Threshold**: Set to 75 percent of Raw Counts with sensor ON
- **Noise Threshold**: Set to 40 percent of Raw Counts with sensor OFF
- **Negative Noise Threshold**: Set equal to Noise Threshold
- **Hysteresis**: Set to 15 percent of Raw Counts with sensor ON
- **Low Baseline Reset**: Set to 10
- **Sensors Autoreset**: Based on design requirements
- **Debounce**: Based on design requirements
- **Widget Resolution**: Select 8-bits unless you are using a slider and the signal is saturating due to thin overlay or large sensor area, then select 16-bits
- **Filter Selection:** See Software Filtering to choose an appropriate filter

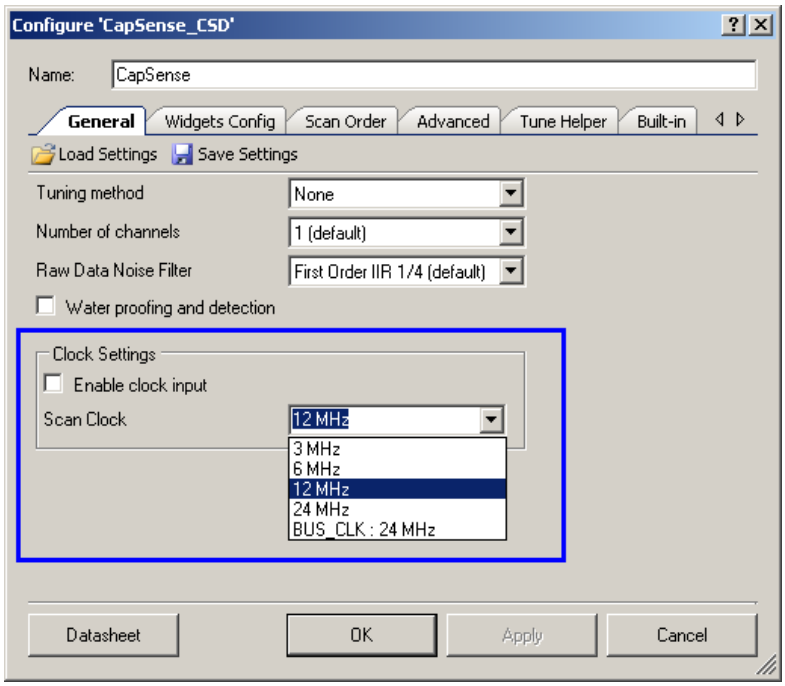
## 5.4 Low-Level Parameters

Low level parameters define the behavior of the CapSense system at the physical layer and relate to the conversion from capacitance to raw count.

## 5.4.1 Clock Settings

### *5.4.1.1 Scan Clock*

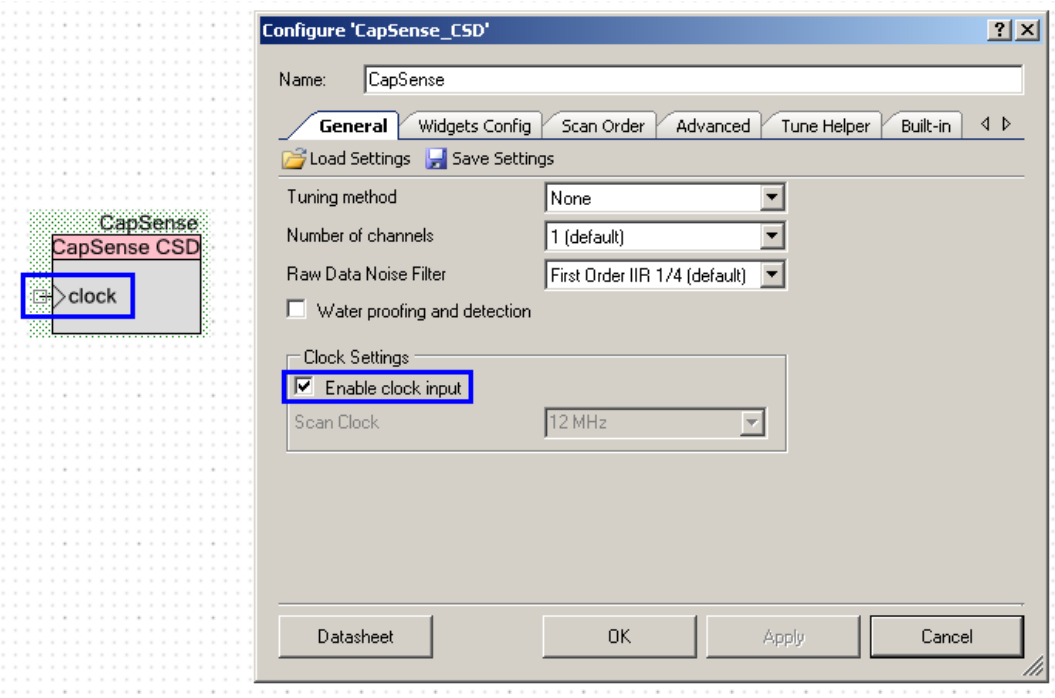
Figure 5-13. Scan Clock Parameter




The Scan Clock parameter selects the source clock for the CapSense_CSD block. The Scan Clock does not depend on the CPU frequency. It is derived from the master clock, therefore the master clock frequency must be equal to or greater than the Scan Clock parameter setting.

### *5.4.1.2 Enable Clock Input*

Figure 5-14. External Clock Input Parameter

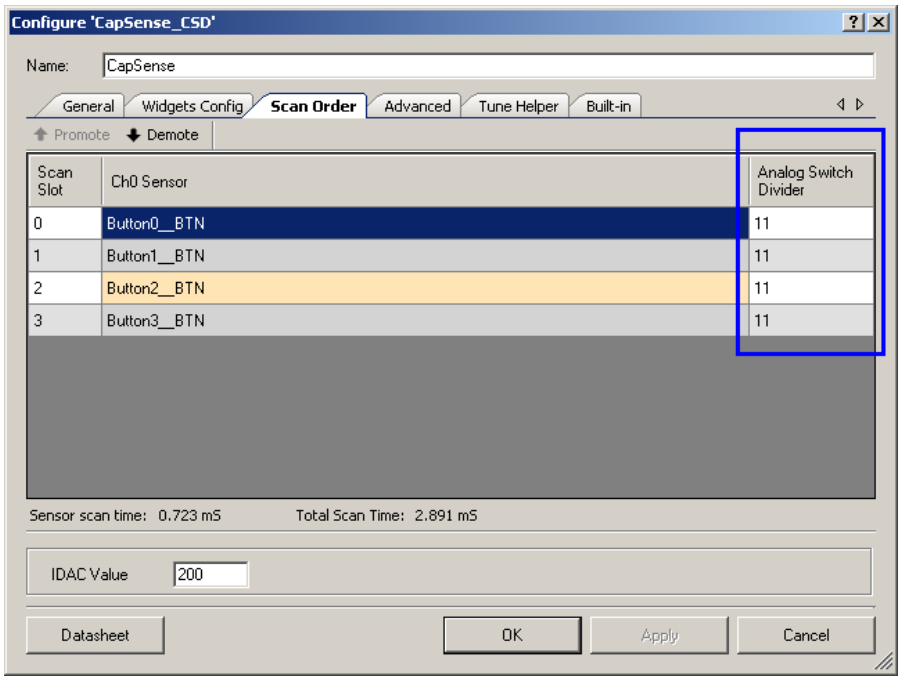



Selecting the Enable Clock Input parameter means that the source clock originates outside of the CapSense_CSD block. When this option is enabled, the Scan Clock frequency is disabled and a terminal appears on the CapSense_CSD component as shown in Figure 5—14. Digital signals in the system and external signal routed through GPIO pins can be connected to this terminal.

## 5.4.2 Analog Switch Divider

### *5.4.2.1 Analog Switch Divider (Prescaler)*

Figure 5-15. Analog Switch Divider Parameter




The Analog Switch Divider parameter sets the switching frequency for the CapSense sensors. The CapSense sensors are continuously switched between $C_{MOD}$ and GND in IDAC Sourcing mode. They switch between $C_{MOD}$ and $V_{DD}$ in case of IDAC sinking and External Resistor mode. The Switching Clock frequency is defined as:

$$Switching\ Clock = \frac{Scan\ Clock}{Analog\ Switch\ Divider}$$ Equation 5

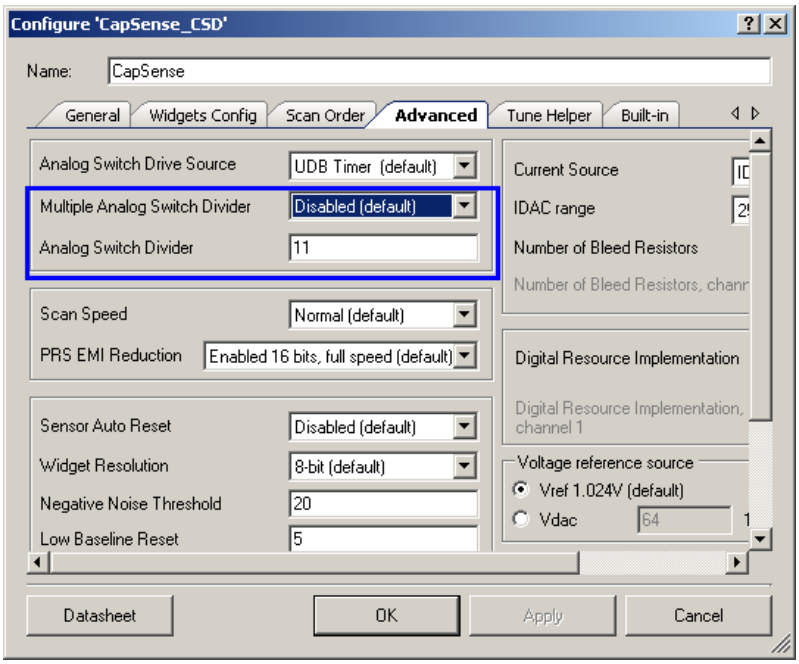
Where,

Scan Clock = Source clock to the CapSense_CSD block as explained in the Clock Settings

Switching Clock = The frequency with which CapSense sensors are switched

### *5.4.2.2 Multiple Analog Switch Divider*

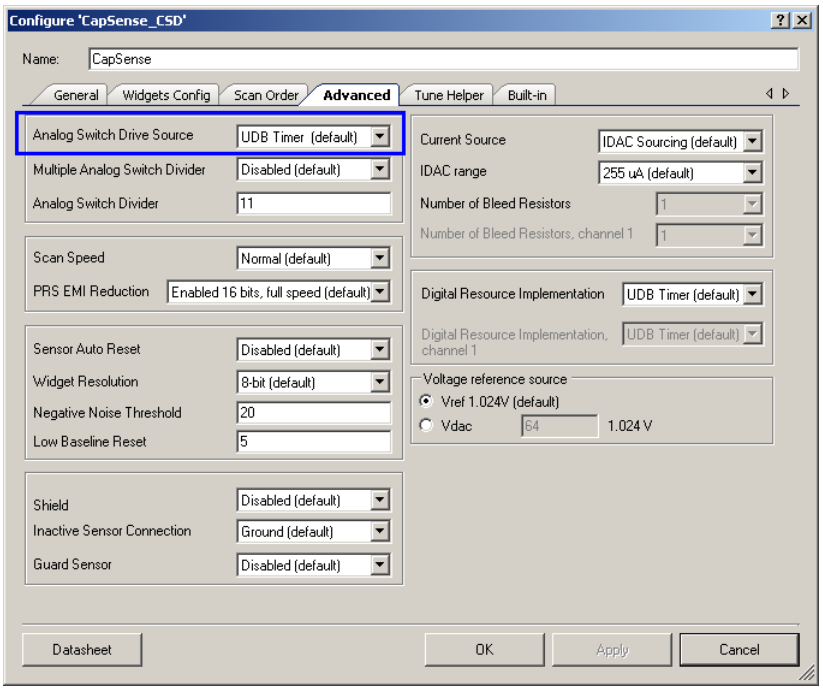
Figure 5-16. Multiple Analog Switch Divider Parameter




The CapSense_CSD allows you to select an individual Analog Switch Divider value for each CapSense sensor or to have one common Analog Switch Divider value for all of the sensors. A common Analog Switch Divider value is suitable when $C_P$ is nearly the same for all of the sensors. If there is large variation in $C_P$ between sensors, the Multiple Analog Switch Divider parameter should be enabled. When the Multiple Analog Switch Divider parameter is enabled, the values are set in the scan order tab shown in Figure 5—15.

### *5.4.2.3 Analog Switch Drive Source*

Figure 5-17. Analog Switch Drive Source Parameter




This parameter selects how the Analog Switch Divider is implemented. The Analog Switch Divider is a timer based divider. The resource utilized to implement this timer can be selected from the following three options:

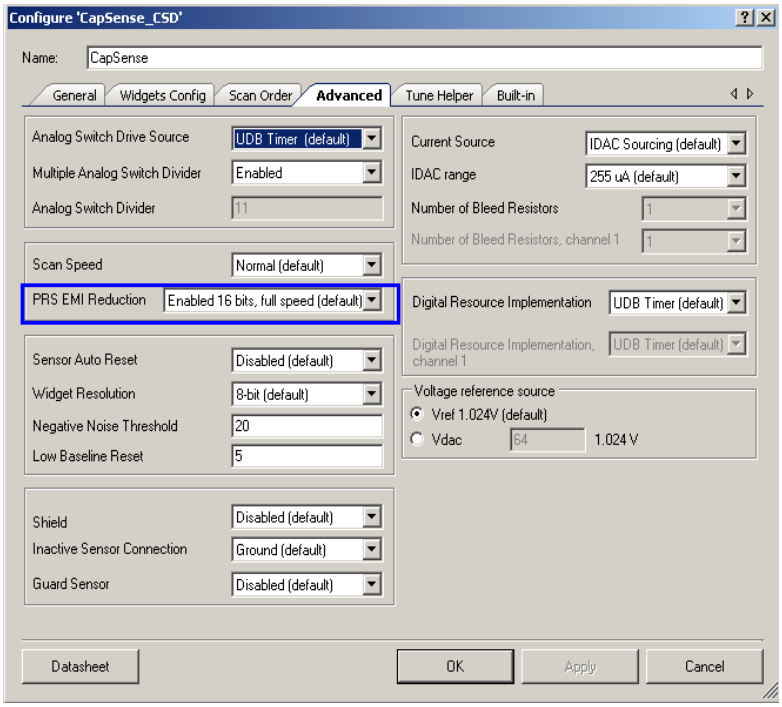
**Direct** – When this option is selected no Analog Switch Divider is used, the Scan Clock itself becomes the Switching Clock. This option is used when there is critical resource usage requirement, but you must ensure that the design is working properly.

**UDB Timer** – When this option is selected, the timer for the Analog Switch Divider is implemented using a UDB. This is default option because there are multiple UDB block.

**FF Timer** - When this option is selected, the timer for the Analog Switch Divider is implemented using a fixed function digital block.

### 5.4.3 Pseudo Random Sequence (PRS)

Figure 5-18. PRS EMI Reduction Parameter




This parameter gives you the ability to use a pseudo random sequence (PRS) on the Switching Clock. The PRS varies the Switching Clock frequency around its central frequency to spread its spectrum and reduce EMI. You can select from the following options:

**Disabled** – This option provides best SNR, hence recommended if electromagnetic interference reduction is not required for the application.

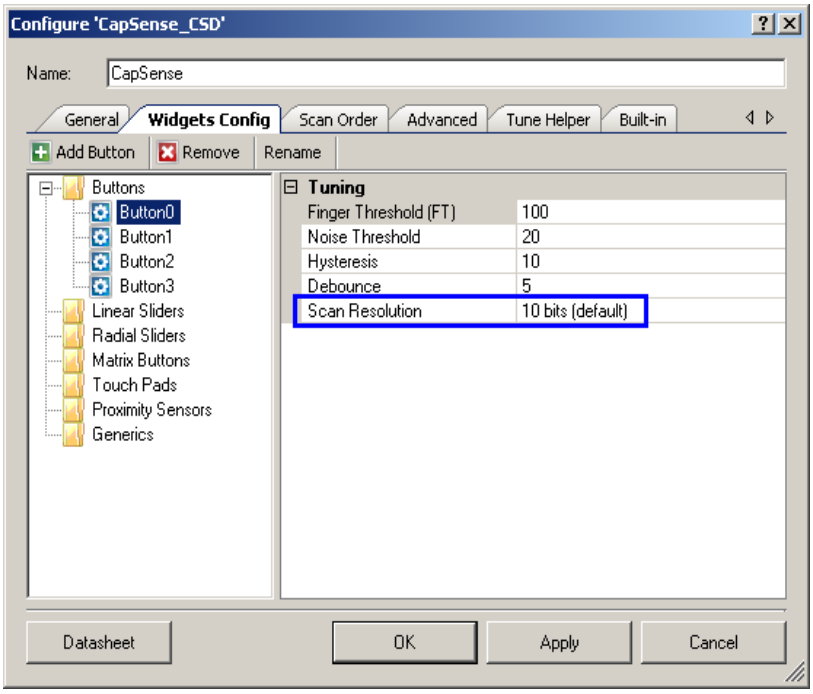
**Enabled 8 bits** – 8-bit provides better SNR compared to 16 bit, but the shorter repeat period causes increased EMI.

**Enabled 16 bits, full speed (default)** – 16 bit provides a lower SNR but superior EMI reduction. This is the recommended option to reduce EMI.

**Enabled 16 bits, 1/4 speed** – Requires a 4 times faster clock to obtain the same PRS clock output as Enable 16 bits, full speed

### 5.4.4 Scan Resolution

Figure 5-19. Scan Resolution Parameter

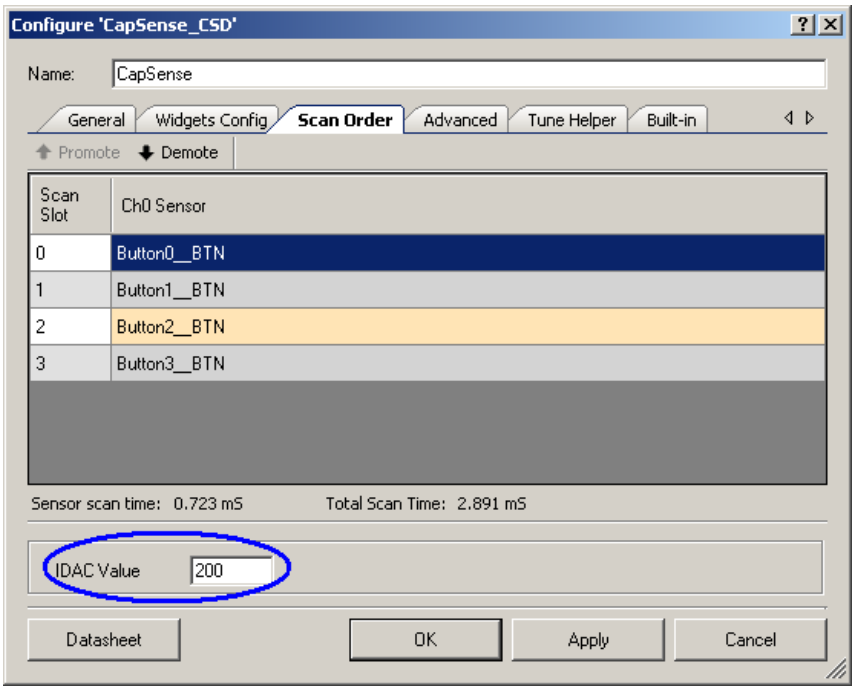



This parameter determines the scanning resolution in bits. The maximum raw count value for a Scan Resolution of N bits is $2^{N-1}$. Increasing the resolution improves sensitivity, but also increases the time required to scan a sensor. Possible values are 8 to 16 bits.

## 5.4.5 IDAC Current

### *5.4.5.1 IDAC Value*

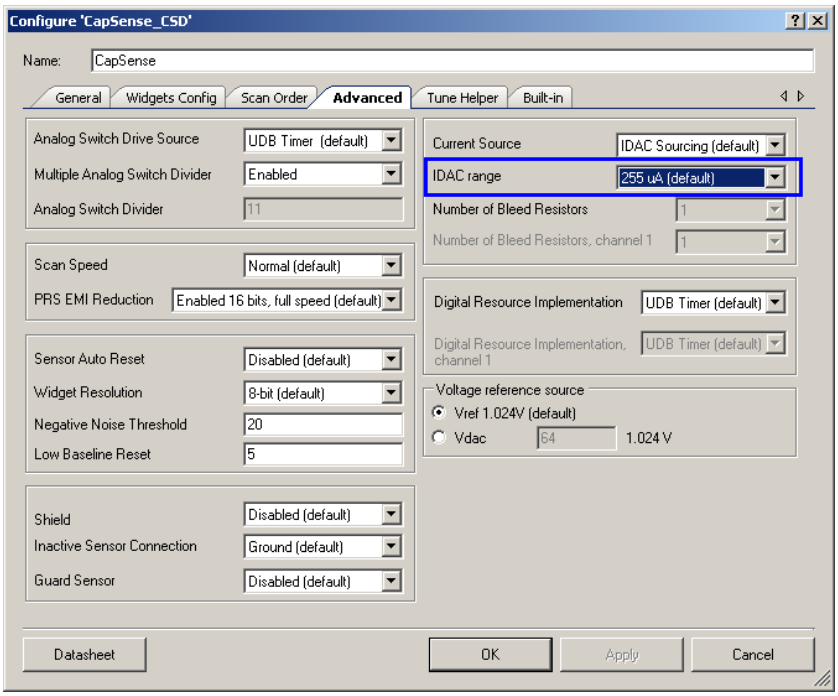
Figure 5-20. IDAC Value Parameter




This parameter sets the IDAC current. When the IDAC current decreases, raw counts increase along with sensitivity. The maximum raw count value for a Scan Resolution of N bits is $2^{N-1}$. The IDAC current should be set such that raw counts are 50 percent to 80 percent of maximum value.

### *5.4.5.2 IDAC Range*

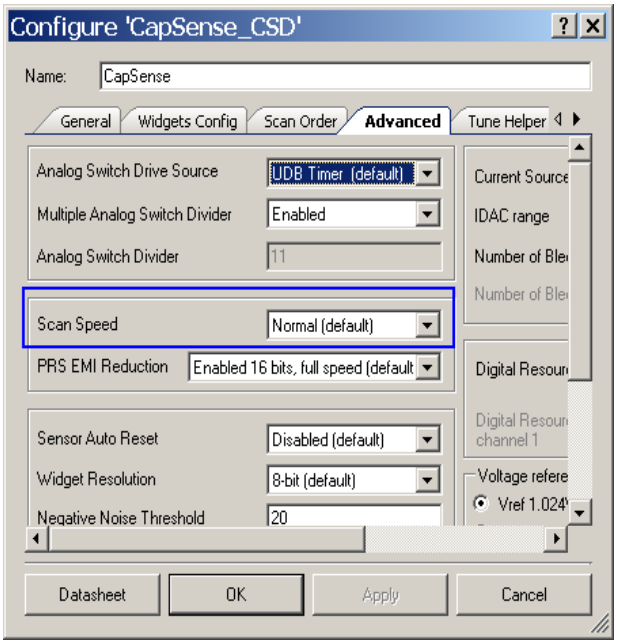
Figure 5-21. IDAC Range Parameter




This parameter selects the range of the IDAC current. The IDAC current required varies directly with $C_P$. For high $C_P$ a higher range should be chosen and vice versa. Possible values are 32 µA, 255 µA, and 2048 µA.

## 5.4.6 Scan Speed

Figure 5-22. Scan Speed Parameter



This parameter sets sensor scanning speed. Faster scanning speeds provide good response time. Slower scanning speeds provide improved SNR and better immunity to power supply and temperature changes. Possible values are Very Fast, Fast, Normal, and Slow.

The Scan Speed clock sets the Counting Clock using the following equation:

$$Counting\ Clock = \frac{Scan\ Clock}{Scan\ Speed\ Divider}$$ Equation 6

Where,

Scan Clock = source clock for the CapSense_CSD block

Scan Speed Divider = 1 (Very Fast), 2 (Fast), 3 (Normal), 4 (Slow)

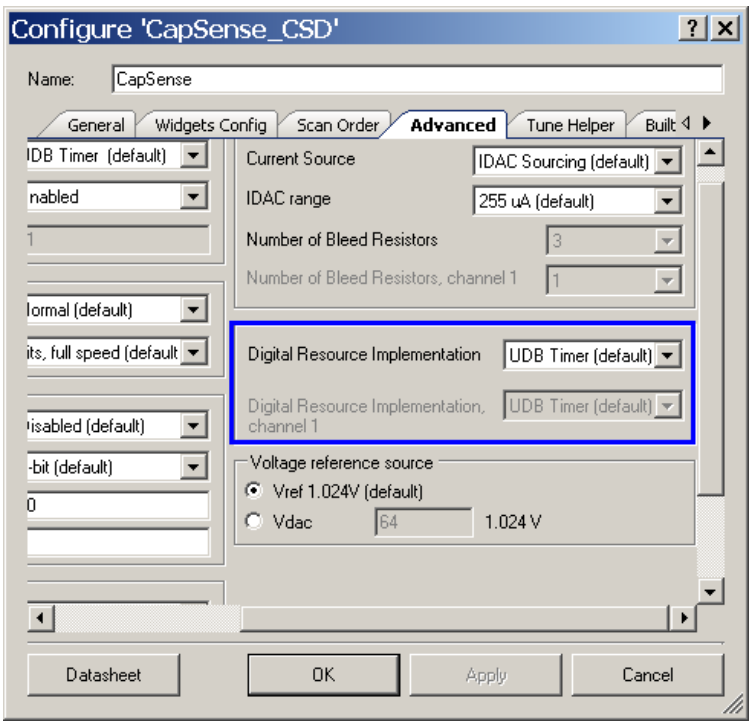
Scan time depends on the Scan Speed, Scan Resolution, Scan Clock, and CPU clock. Table 5-2 shows the scan time for scanning a single sensor with Scan Clock = 24 MHz and CPU clock = 48 MHz.

Table 5-2. Scan Time for a Single Sensor in µs versus Scan Speed and Resolution.

| Resolution (bits) | Scanning Speed | | | |
|---|---|---|---|---|
| | Very Fast | Fast | Normal | Slow |
| 8 | 58 | 80 | 122 | 208 |
| 9 | 80 | 122 | 208 | 377 |
| 10 | 122 | 208 | 377 | 718 |
| 11 | 208 | 377 | 718 | 1400 |
| 12 | 377 | 718 | 1400 | 2770 |
| 13 | 718 | 1400 | 2770 | 5500 |
| 14 | 1400 | 2770 | 5500 | 10950 |
| 15 | 2770 | 5500 | 10950 | 21880 |
| 16 | 5500 | 10950 | 21880 | 43720 |

### 5.4.7 Digital Resource Implementation

Figure 5-23. Digital Resource Implementation Parameter



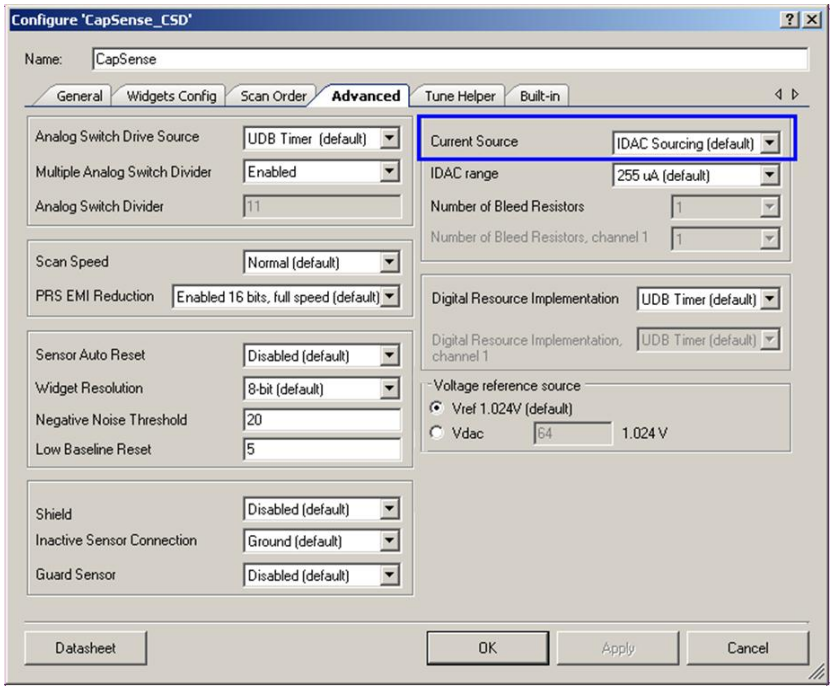

The Scan Speed setting uses a timer based divider to create the Counting Clock. This parameter allows you to select the resource for implementing this divider. The Digital Resource Implementation can be chosen individually for each channel. The options are:

**UDB Timer** – When this option is selected, the timer is implemented using a UDB. This is default option because there are multiple UDB block.

**FF Timer** - When this option is selected, the timer is implemented using a fixed function digital block.

### 5.4.8 Current Source

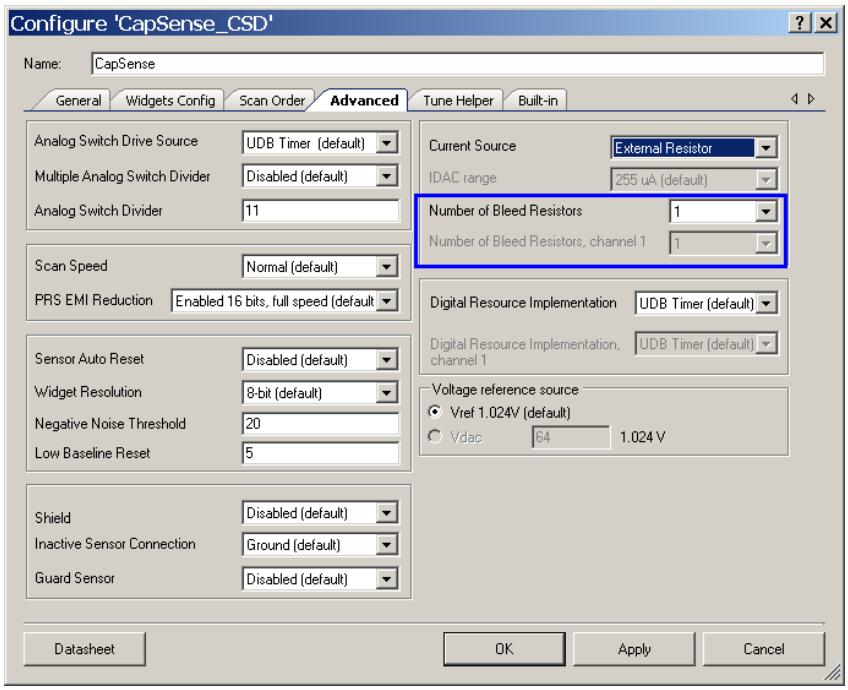
Figure 5-24. Current Source Parameter




This parameter selects the current source method. Possible values are IDAC Sourcing, IDAC Sinking, and External Resistor. See Current Source Methods for a detailed description of these current source methods.

### *5.4.8.1 Number of Bleed Resistors*

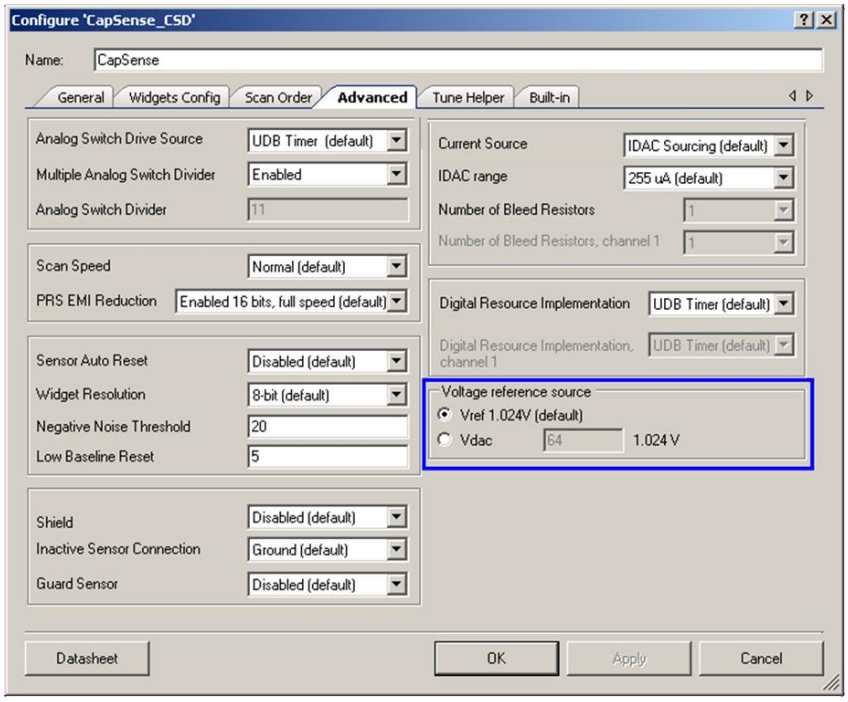
Figure 5-25. Number of Bleed Resistors Parameter




The External Resistor method requires bleed resistors external to PSoC device. This parameter allows you to specify the number of external bleed resistors. Up to three bleed resistors can be connected. For two-channel capacitive sensing, each channel needs a separate set of bleed resistors.

### 5.4.9 Voltage Reference Source

Figure 5-26. Voltage Reference Source Parameter



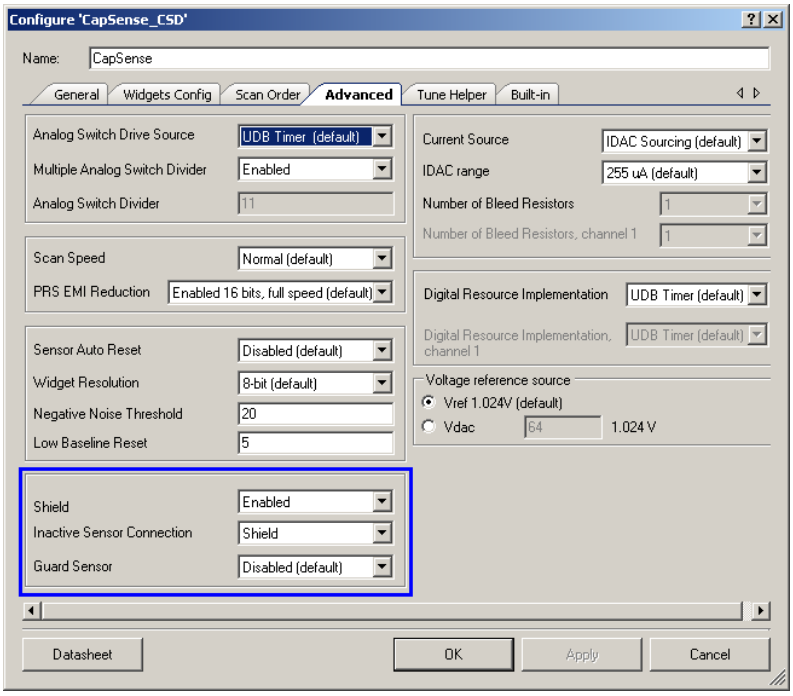

This parameter allows you to specify the voltage reference source. The options are:

**Vref 1.024 V** – This is the bandgap voltage

**Vdac** – This allows you to select any voltage from 0 to 4 V. It is useful when using the IDAC Sourcing method. By increasing the voltage reference you can increase the voltage swing on the sensor. This provides better immunity to finger conducted noise. This setting consumes one DAC resource.

## 5.4.10 Shield Electrode and Guard Sensor

Figure 5-27. Shield Electrode and Guard Sensor Parameters




### *5.4.10.1 Shield*

This parameter should be enabled if your design uses shield electrodes. See Water Tolerant Design.

### *5.4.10.2 Inactive Sensor Connection*

This parameter specifies the connection for sensors when they are not being scanned. Possible values are:

**GND** – This is the recommended setting because it produces less noise.

**Shield** – This setting should be selected ff your design uses shield electrodes and includes widgets with adjacent sensors such as sliders. Connecting the inactive sensors to the shield electrode prevents false triggering when water is present.

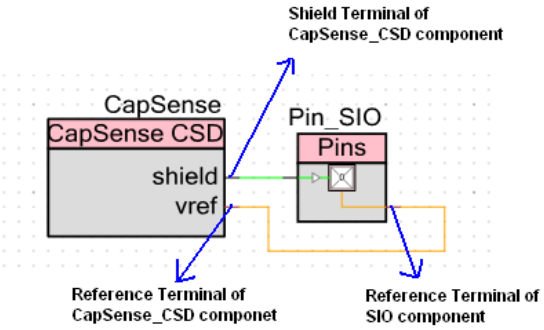
**HI-Z** – Leaves the inactive sensors at high impedance. Use this setting for reducing the total parasitic capacitance caused by the ground connected sensors.

### *5.4.10.3 Guard Sensor*

This parameter should be enabled if your design uses a guard sensor. See Water Tolerant Design.

### *5.4.10.4 Using SIO Pin for Shield Electrode*

Figure 5-28. SIO Pin Used as Shield Electrode

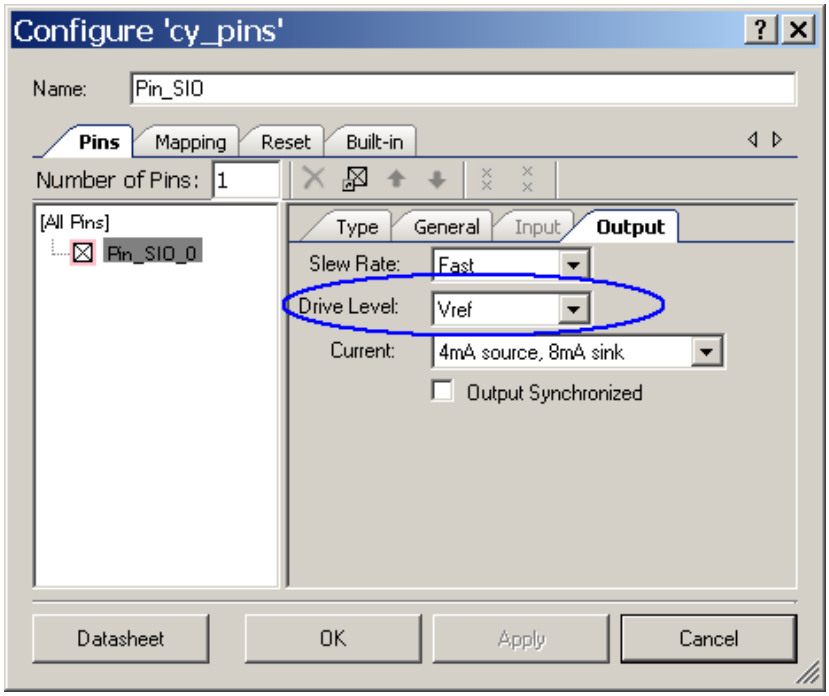



When the Shield is enabled a shield terminal appears on the component as shown in Figure 5—28.

An I/O pin should be connected to this terminal. When you use the IDAC Sinking or External Resistor method as your Current Source, a normal GPIO pin should be used. However, if you are using the IDAC Sourcing method, you should connect the shield an SIO pin. Using an SIO pin allows you to match the shield signal to the sensor signal.

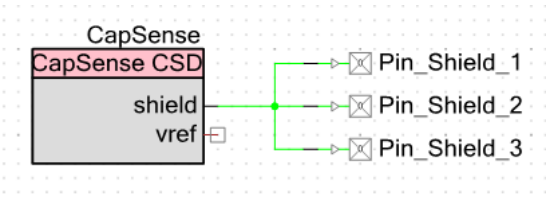
To use an SIO pin as shield electrode, drag a Digital Output Pin from the component catalog. Open the configuration window and set the Drive Level option to “Vref” as shown below.

Figure 5-29. Configuring the Drive Level to Make the Pin Component SIO



### 5.4.10.5 Multiple Shield Electrode Pins

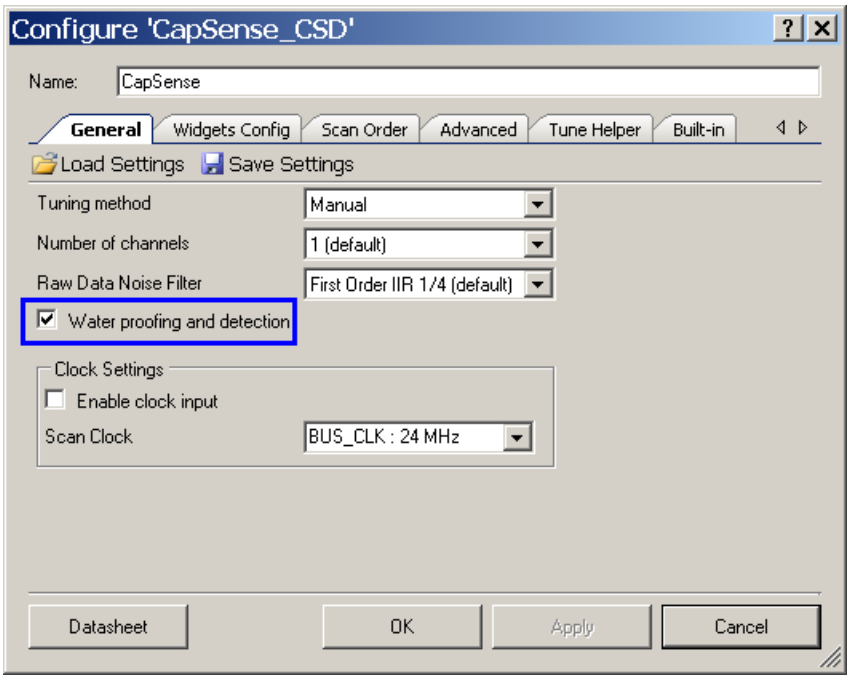
Figure 5-30. Multiple Shield Electrode Pins




When the shield area is large, a single pin may not be sufficient to drive the shield electrode. You need to verify that the shield electrode completely charges and discharges. To check this, probe the shield electrode pin and monitor it on an oscilloscope. Because normal probes may add significant capacitance to the pin they should be set to 10x mode. FET probes are recommended. If the shield electrode is not charging and discharging completely then use multiple I/O pins to drive the shield electrode. I/O pins (both GPIO and SIO) have drive strength of 4 mA.

### 5.4.10.6 Water Proofing and Detection

Figure 5-31. Water Proofing and Detection Parameter



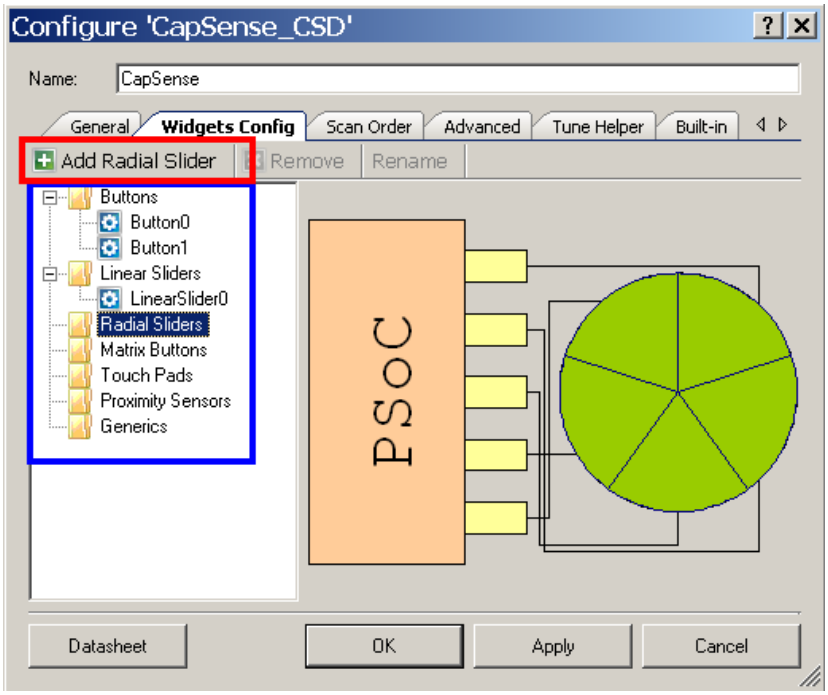

Another way to enable the shield electrode and guard sensor is to select the Water Proofing and Detection parameter on the General tab. Checking this box ensures both shield electrode and guard sensor are enabled on the Advanced tab.

## 5.5 Widget Configuration

The Widgets Config tab allows you to add and configure widgets.

### 5.5.1 Adding a Widget

Figure 5-32. Adding Widgets




To add a widget select the type of widget and click Add.

### 5.5.2 Number of Sensor Elements

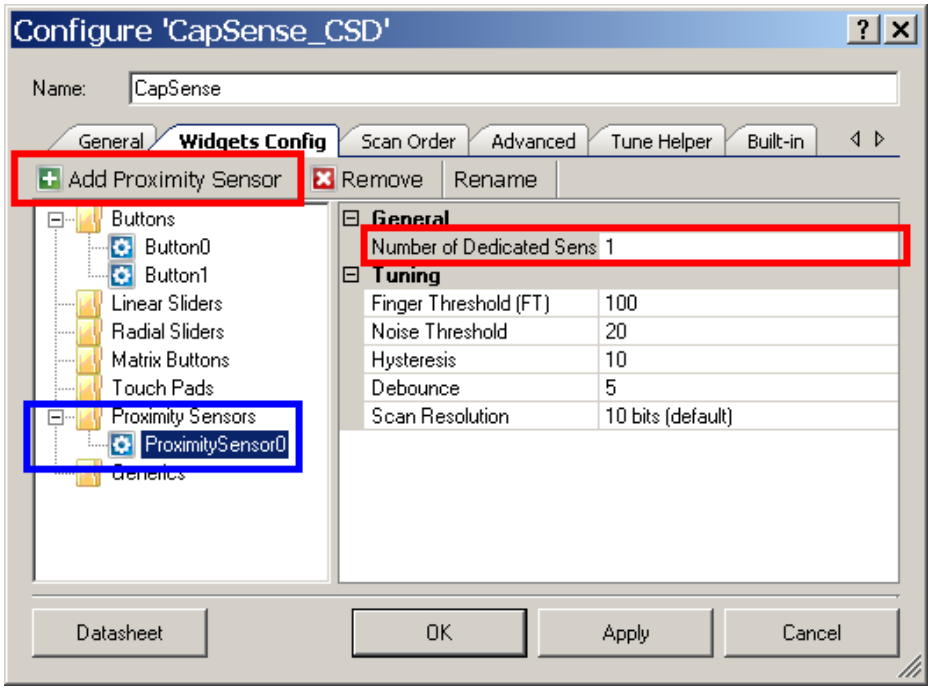
The number of sensors for each widget is set in this tab.

### 5.5.3 API Resolution

Some widget types such as slider and touch pad use finger position data instead of sensor ON/OFF. The resolution of the finger position data is called the API resolution.

### 5.5.4 Proximity Sensor

Figure 5-33. Proximity Sensor Parameters



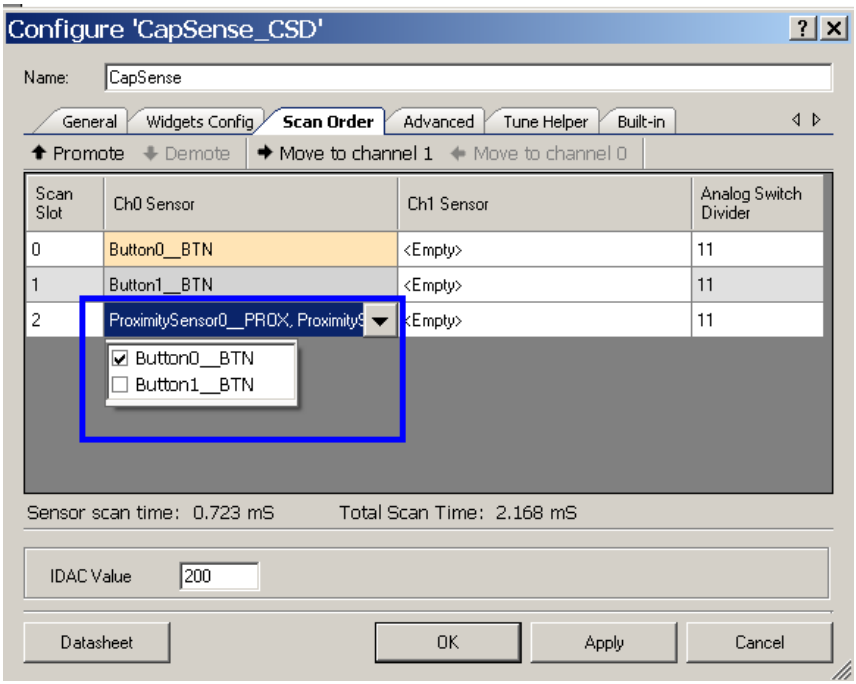

A proximity sensor detects the presence of hand near the sensor without actually touching it. A proximity sensor can consist of a long trace around the perimeter of the user interface on the PCB. Alternately, you can short the sensors together and scan them as a single sensor. You can combine these two approaches and short the proximity sensor trace that surrounds the user interface with the other sensors and scan them all as a single sensor.

**Note** In PSoC Creator the proximity sensor is disabled while all other sensors are enabled by default. It must be enabled explicitly before scanning as shown in the following code snippet.

```
17
18      CyGlobalIntEnable;  /* Uncomment this line to enable global interrupts. */
19      CapSense_Start();
20      CapSense_EnableWidget(CapSense_PROXIMITYSENSOR0__PROX); /*Enabling proximity sensor explicitly*/
21      CapSense_InitializeAllBaselines();
22
23      for(;;)
24      {
25          if ( CapSense_IsBusy()==0 )
26          {
27              CapSense_UpdateEnabledBaselines();
28              CapSense_CheckIsAnyWidgetActive();
29              CapSense_ScanEnabledWidgets();
30          }
31      }
32  }
```

### *5.5.4.1 Number of Dedicated Sensor Elements*

Figure 5-34. Combing Existing Sensors with Proximity Sensor

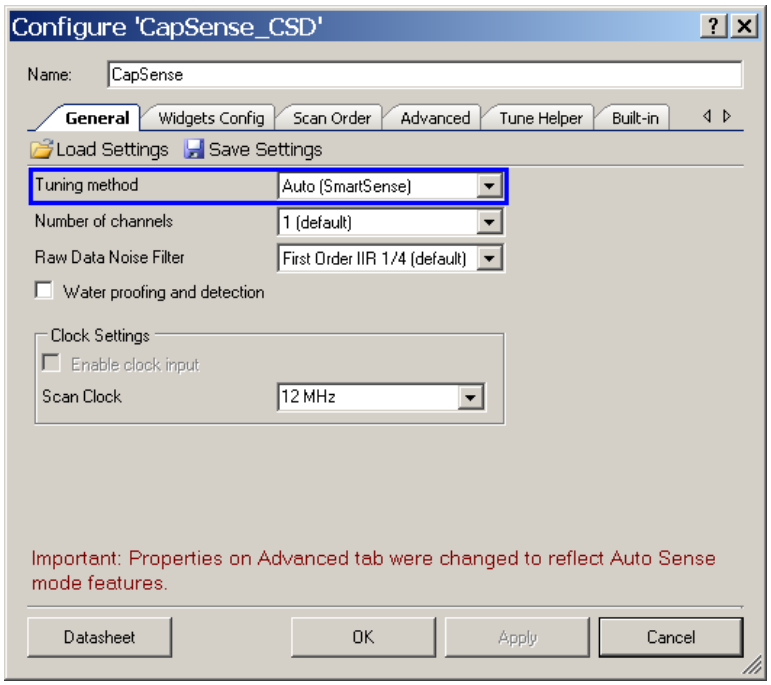



To combine existing sensors with a dedicated proximity sensor set the Number of Dedicated Sens on the Widgets Config tab as shown in Figure 5-33. Then select which of the existing sensors you want to combine with the proximity sensor using the drop-down menu on the Scan Order tab.

Combining existing sensors with the proximity sensor will not affect their operation, however, the resulting sensor will have a high $C_P$ leading to a long scan time. Therefore, proximity should be scanned at a lower rate than the other sensors in order to avoid long scanning intervals.

## 5.6 Tuning Method

Figure 5-35. Tuning Method Parameter




This parameter allows you to select how the CapSense system will be tuned. For all tuning methods the Tuner GUI can be used to monitor the CapSense signals. Possible values are:

**Auto** – The SmartSense Auto-Tune algorithm sets all the parameters to optimum values. The following parameters cannot be changed when Auto Is selected:

Table 5-3. Fixed Parameters when Auto is Selected

| Parameter | Setting |
|---|---|
| Current Source | IDAC Sourcing / IDAC Sinking<br>External Resistor is not supported |
| IDAC Range | 255 µA |
| Multiple Analog Switch Divider | Enabled |
| Scan Speed | Normal |
| PRS | Enabled 16 bits, full speed |

**Manual** – You must tune all the parameters manually.

**None** – The parameters are fixed and stored in flash. The fixed parameters should have been tuned using either manual tuning or auto-tuning method.

### 5.6.1 Sensitivity

The Sensitivity parameter sets the value of $C_F$ for the SmartSense Auto-Tune algorithm. The Sensivity setting is multiplied by 0.1 pF to determine $C_F$. A setting of 1 represents $C_F$ = 0.1 pF a setting of 4 represents $C_F$ = 0.4 pF.

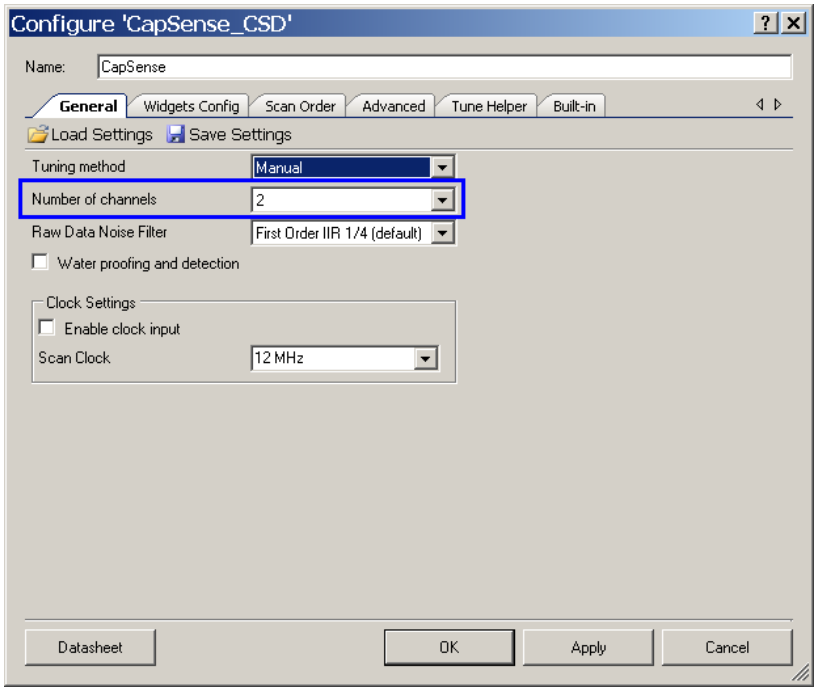
SmartSense Auto-Tune uses the Sensitivity parameter to calculate the parameters listed in Table 5-4.

Table 5-4. Parameters Set by Auto-Tune Using Sensitivity

| Parameter | When it is calculated |
|---|---|
| Analog Switch Divider | Once at CapSense_CSD start up |
| IDAC Value | Once at CapSense_CSD start up |
| Scan Resolution | Once at CapSense_CSD start up |
| Finger Threshold | Continuously during sensor scanning |
| Noise Threshold | Continuously during sensor scanning |
| Hysteresis | Continuously during sensor scanning |

## 5.7 Number of Channels

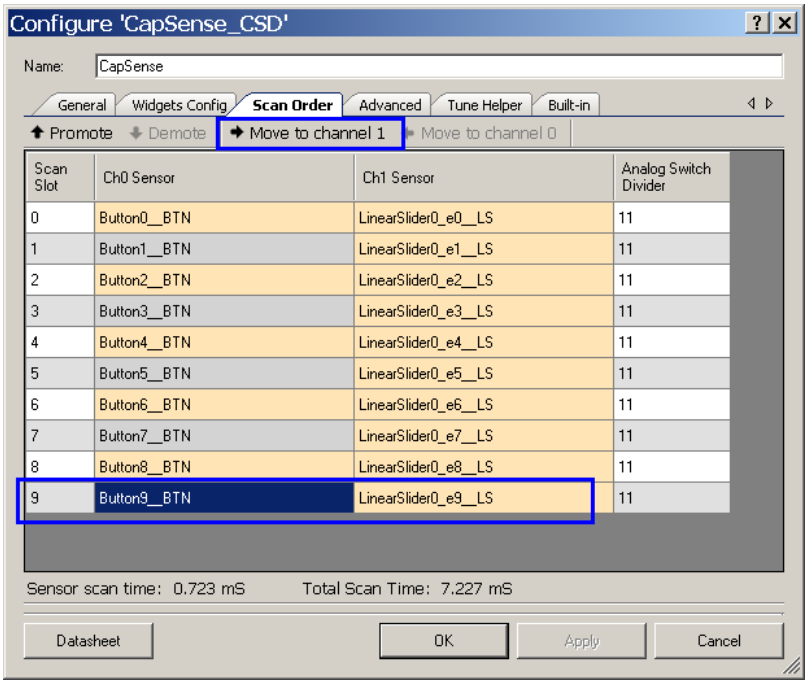
Figure 5-36. Number of Channels




This parameter allows you to implement a two channel design.

### 5.7.1 Move to Channel 1 / Channel 0

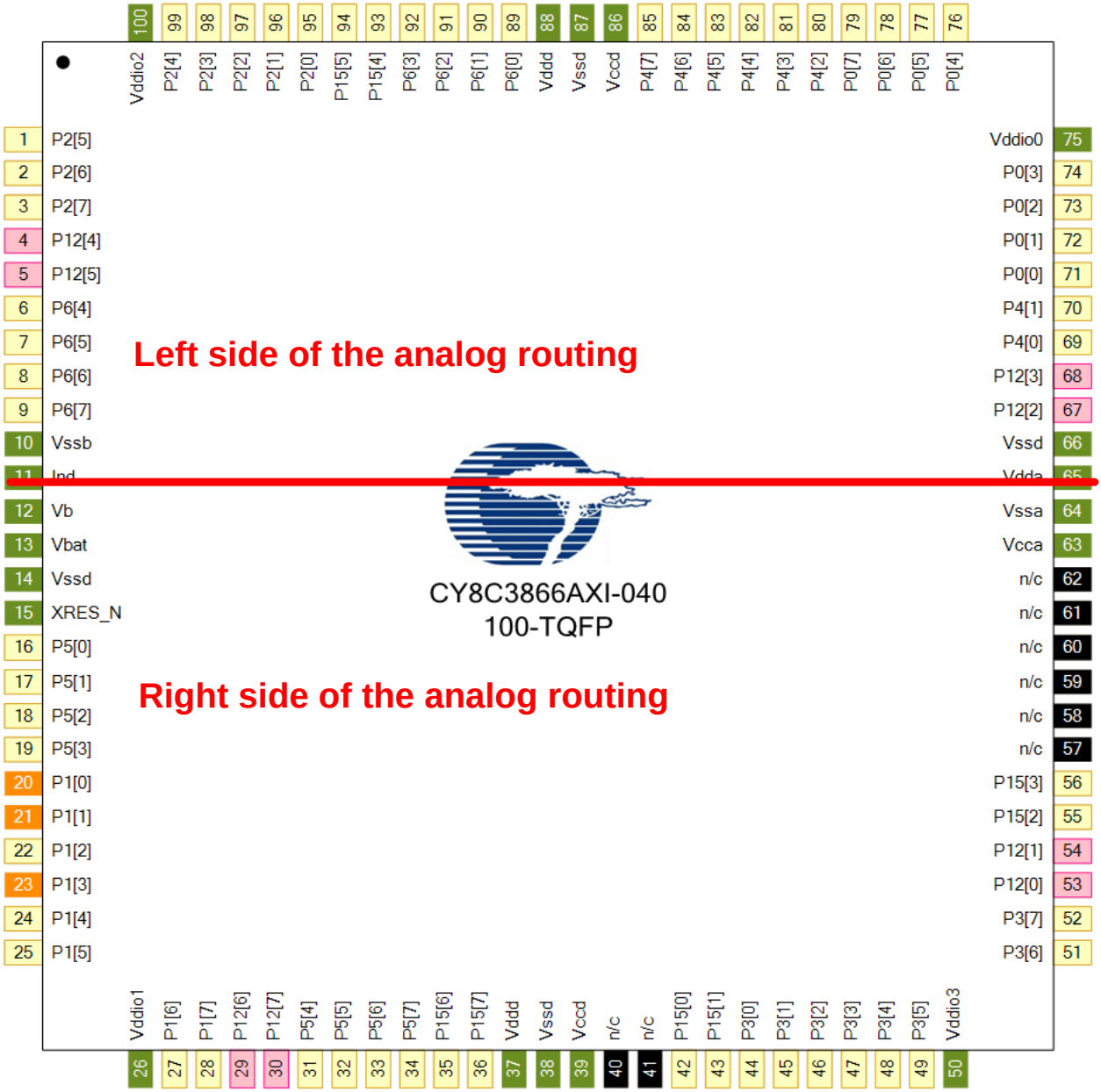
Figure 5-37. Setting Channel Assignment for Sensor




This parameters allows you to specify the channel assignment for each sensor. On the Scan Order tab, select the desired sensor, and click on Move to Channel 1 or Move to Channel 0.

## 5.7.2 Pin Assignment for Two Channel Design

Figure 5-38. Pin Assignment for Two Channel Design




Each GPIO of PSoC is a part of either the left side or the right side of analog routing, depending on its connection to the internal routing buses. See the section “Analog Routing” in the device datasheet for more details. Figure 5-38 is a pin diagram of PSoC that shows the GPIOs and their respective side in the analog routing.

In a two-channel design, all of the sensors assigned to the same channel should reside on one side of the routing and all of the sensors assigned to the other channel should reside on the other side of the routing. If you are only using one channel in your design, the $C_{MOD}$ and the Bleed Resistor pins should be on the same side of the routing as all of the sensors.

## 5.8 Tuner GUI

One of the useful features of CapSense_CSD component is an embedded GUI that lets you monitor the CapSense signals. The Tuner GUI is available for both Auto and Manual tuning. CapSense Performance Tuning explains the tuning process. The Tuner GUI connects to your PC's USB through the MiniProg3 and uses $I^2C$ to communicate with the device.

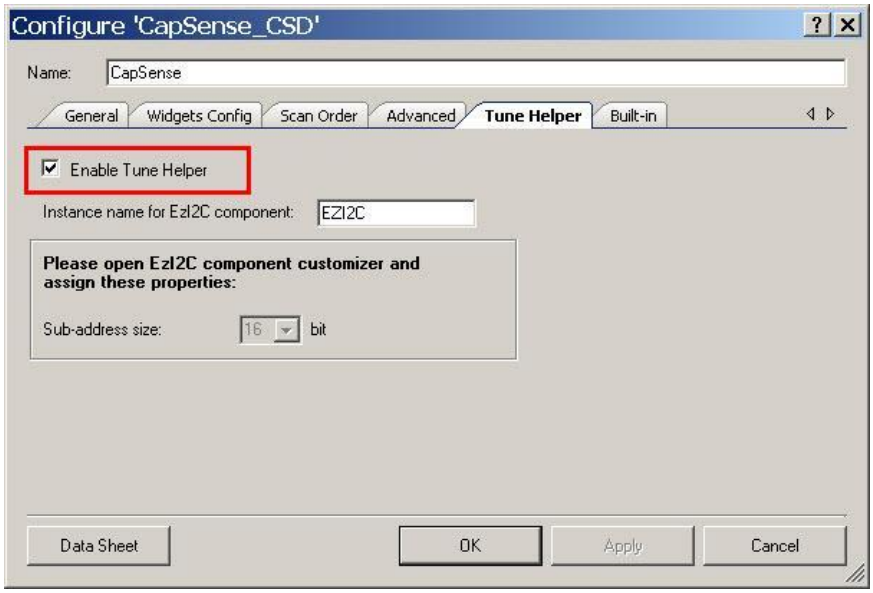
Figure 5-39. MiniProg3 as $I^2C$-USB Bridge


### 5.8.1 Setup

**Note** Complete Steps 1-5 prior to building the PSoC Creator project.

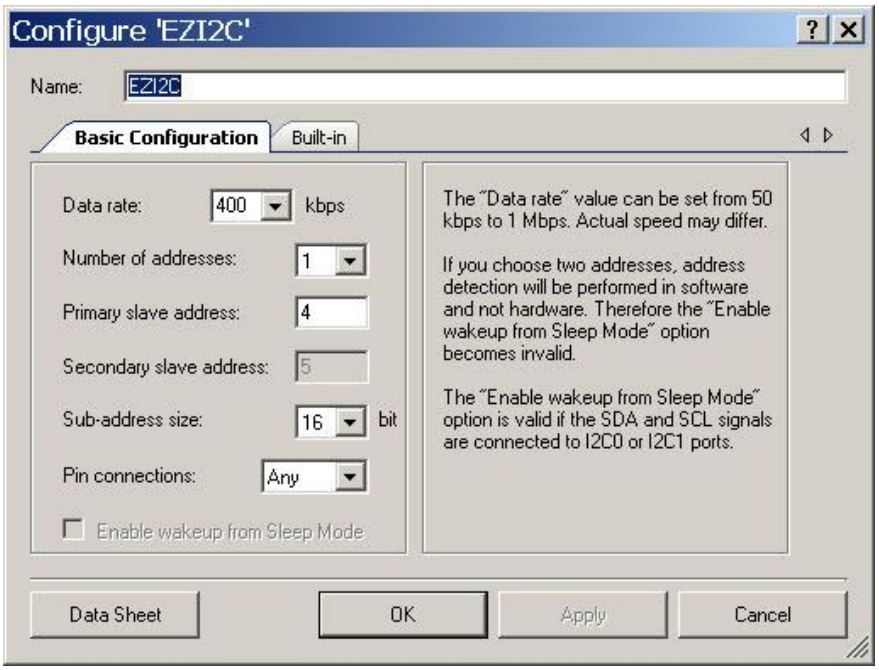
1. Set the Tuning Method to Auto and Manual. If None is selected the Tuner GUI will not work.
2. Add a Widget.
3. Select Enable Tuner Helper on the Tune Helper tab.

Figure 5-40. Enabling Tuner



4. Place and configure an EZI2C component.

Figure 5-41. Rename the EZI2C Component




5. Add the following section of code to your project.

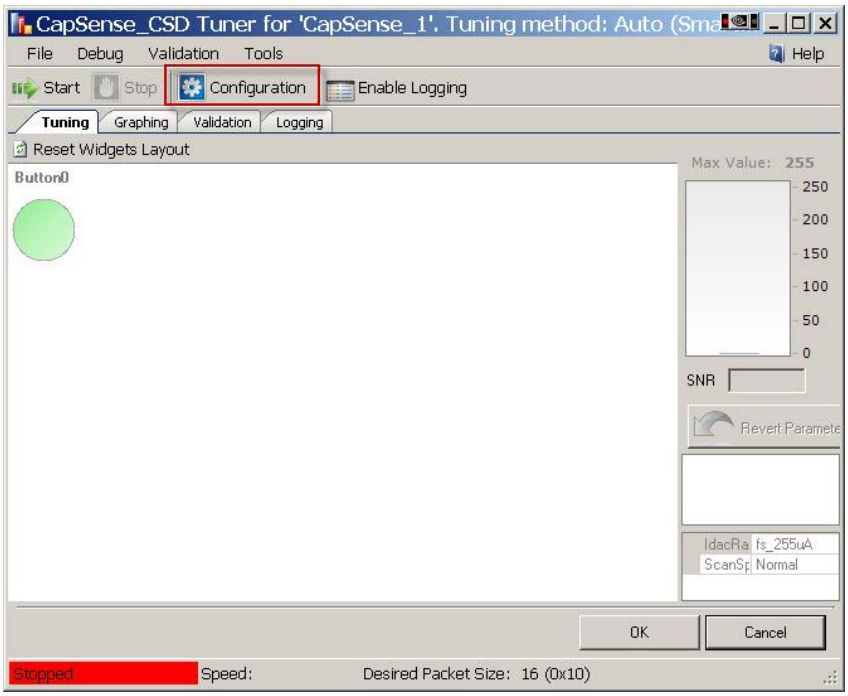
```
     #include <device.h>
void main()
{
CyGlobalIntEnable; /* Global Interrupt enable */

CapSense_TunerStart(); /* CapSense Block and Tuner Communication power up and
Intialization */

While(1)
{
    CapSense_TunerComm(); /*Scan all the sensors and send the result to PC */
}
}
```

**Note** Complete the following steps *after* you have programmed the device and correctly connected the MiniProg3.

1. Right click on the CapSense_CSD component and click Launch Tuner.

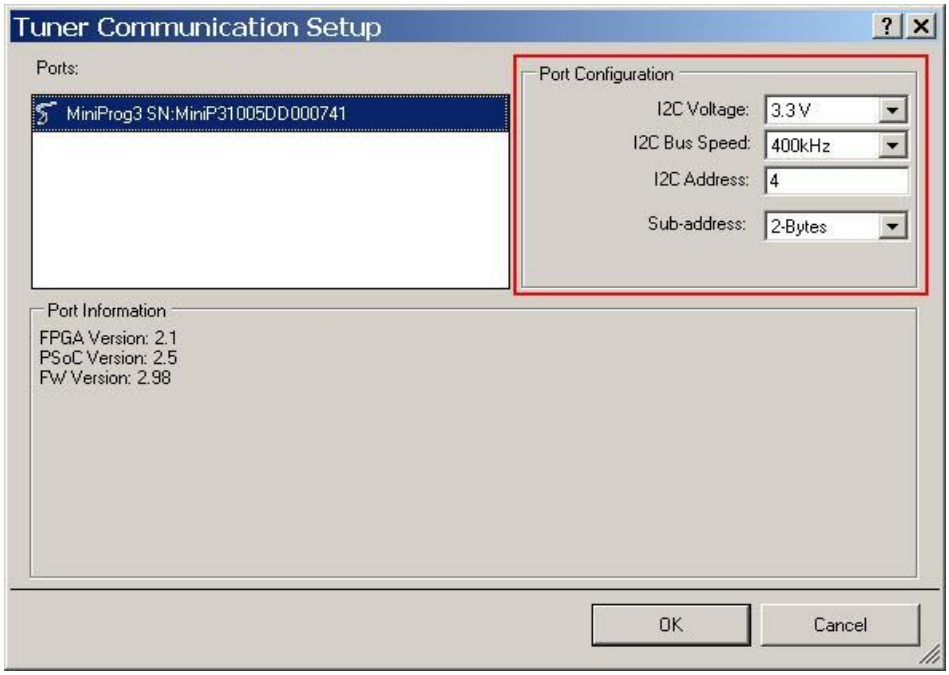
2. Click on Configuration in the Tuner window.

Figure 5-42. Tuner GUI




3. Click on the MiniProg3 in the Tuner Communication Setup window and set the parameters as shown here:

Figure 5-43. Tuner Communication Setup

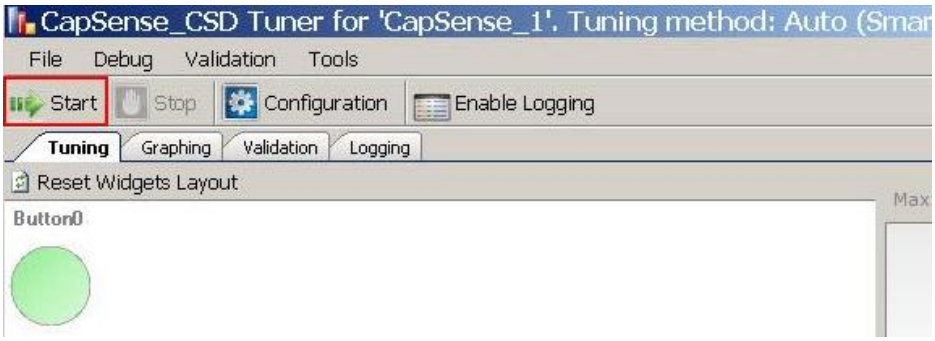



**Note** The voltage settings for the MiniProg3 must match the voltage settings on the board.

4. Click OK, and the Tuner Communication Setup window will close.

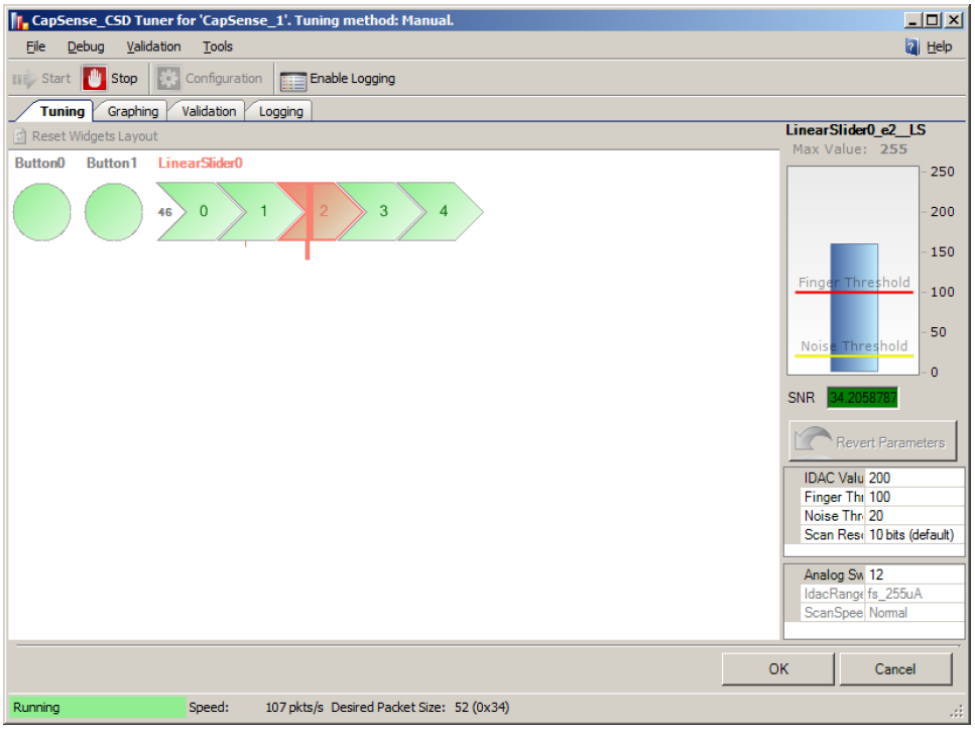
5. Click on Start in the Tuner window to initiate I²C communication and begin monitoring the CapSense parameters.

Figure 5-44. Starting Tuner Communication



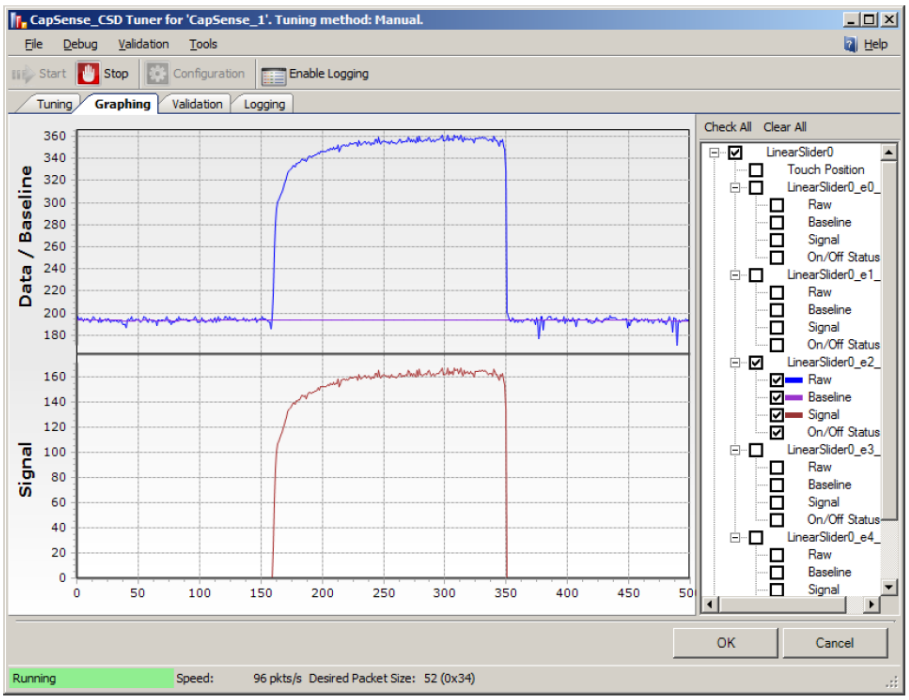

6. Set the parameters for each sensor. Clicking OK at the lower right of the Tuner window sets the parameters in the component once tuning is complete.

Figure 5-45. Setting the Parameters for a Sensor



7. The Graphing tab allows you to monitor signals. You can log data using the Logging tab.

Figure 5-46. Monitoring the CapSense Results




**Note** The raw count signal displayed in the Graphing tab is the unfiltered raw count. If a filter is used on raw count, the SNR should be calculated using the filtered raw count. See Raw Data Noise Filter

# 6. CapSense Performance Tuning

Tuning is the process of determining the optimum values for the CapSense parameters. Tuning is required to maintain high sensitivity to touch and to compensate for process variations in the sensor board, overlay material, and environmental conditions.

## 6.1 Fundamentals

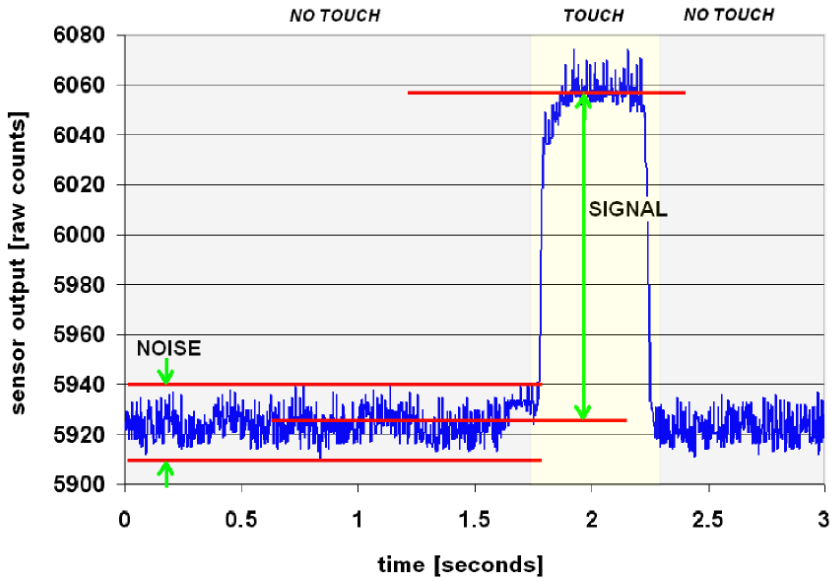
### 6.1.1 Signal, Noise, and SNR

A well-tuned CapSense system reliably discriminates between ON and OFF sensor states. To achieve this level of performance, the CapSense signal must be significantly larger than the CapSense noise. The measure that compares signal to noise is the signal-to-noise ratio (SNR). Before discussing the meaning of SNR for CapSense, it is first necessary to define what signal and noise are in the context of touch sensing.

#### *6.1.1.1 CapSense Signal*

The CapSense signal is the change in the sensor response when a finger is placed on the sensor, as shown in Figure 6-1. The output of the sensor is a digital counter with a value that tracks the sensor capacitance. In this example, the average level without a finger on the sensor is 5925 counts. When a finger is placed on the sensor, the average output increases to 6060 counts. The CapSense signal tracks the change in counts due to the finger, so Signal = 6060 – 5925 = 135 counts.

Figure 6-1. Example of CapSense Signal and Noise




#### *6.1.1.2 CapSense Noise*

CapSense noise is the peak-to-peak variation in sensor response when a finger is not present, as shown in Figure 6-1. In this example, the output waveform without a finger is bounded by a minimum of 5912 counts and a maximum of 5938 counts. The noise is the difference between the minimum and the maximum values of this waveform, so Noise = 5938 – 5912 = 26 counts.

**Note** If you use a Raw Data Noise Filter in your system, calculate noise using the filtered raw count. The Tuner GUI reports the unfiltered raw count on Graphing tab, therefore, you should log the data to calculate noise.

#### *6.1.1.3 CapSense SNR*

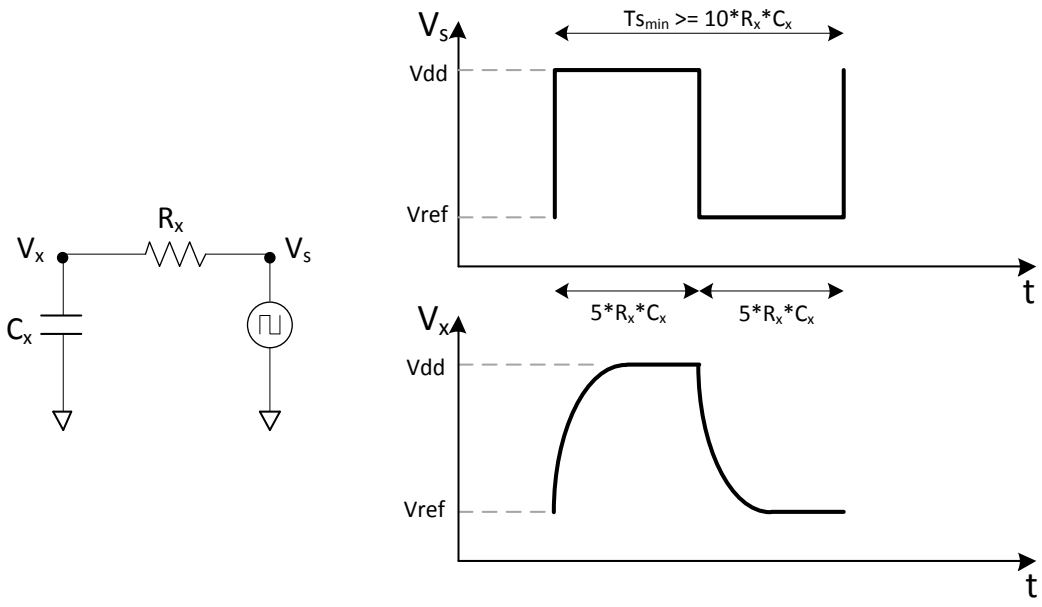
CapSense SNR is the simple ratio of signal and noise. Continuing with the example, if the signal is 135 counts and noise is 26 counts, and then SNR is 135:26, which reduces to an SNR of 5.2:1. The minimum recommended SNR for CapSense is 5:1, which means the signal is five times larger than the noise. Filters are commonly implemented in firmware to reduce noise. See Software Filtering for more information.

### 6.1.2 Charge/Discharge Rate

To achieve maximum sensitivity in the tuning process, the sensor capacitor must be fully charged and discharged during each cycle. The charge/discharge path switches between two states at a rate equal to switching clock defined by Equation 5.

The charge/discharge path includes series resistance that slows down the transfer of charge. The rate of change for this charge transfer is characterized by an RC time constant involving the sensor capacitor and series resistance, as shown in Figure 6-2.

Figure 6-2. Charge/Discharge Waveforms




Set the charge/discharge rate to a level that is compatible with this RC time constant. The rule of thumb is to allow a period of 5RC for each transition, with two transitions per period (one charge, one discharge).The equations for minimum time period and maximum frequency are:

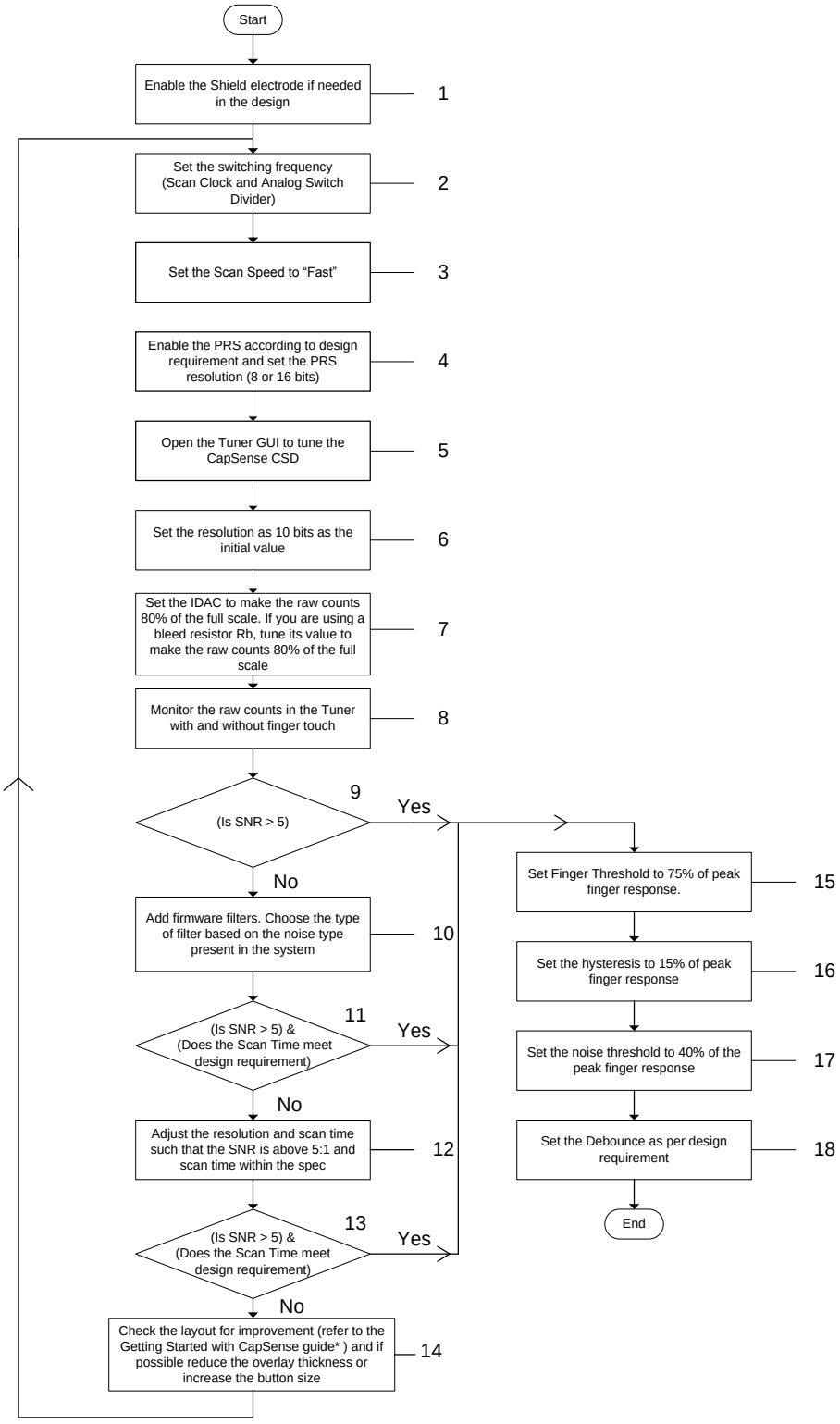
$$Ts_{min} = 10 \times R_X C_X$$ Equation 7

$$fs_{max} = \frac{1}{10 \times R_X C_X}$$ Equation 8

## 6.2 Manual Tuning

The flow chart in Figure 6-3 provides step-by-step instructions for manually tuning a CapSense_CSD component.

Figure 6-3. Tuning the CapSense_CSD Component



1. See Water Tolerant Design for information on when you should use a shield electrode.
2. The switching frequency of each capacitive sensor should be set such that the sensor completely charges and discharges as described in Analog Switch Divider. If not, increase the Analog Switch Divider value in the CapSense_CSD Configuration window. Once you have changed this value you must rebuild the project and reprogram the device.
3. The Scan Speed can be changed later if the SNR or scan time are unacceptable.
4. See Pseudo Random Sequence (PRS) for information on how and when you should use this feature.
5. See Tuner GUI for step-by-step instructions on how to open the GUI.
6. Lower resolutions (8 or 9) can also be used as initial values if the design has an overlay of less than 1 mm.
7. Adjust the IDAC Value in the Tuner GUI until the raw counts reach 80 percent of full scale value. If you cannot 80 percent using the Tuner GUI, adjust the IDAC Range in the CapSense_CSD component configuration window. Once you have changed this value you must rebuild the project and reprogram the device.

   If you are using a Bleed Resistor Rb instead of the IDAC sinking/IDAC sourcing method, change the resistance value to make the rawcounts equal to 80 percent of the full-scale value.
8. Record the peak-to-peak noise and peak finger response.
9. SNR is simply the ratio of signal to noise. For good CapSense design, the SNR should be above 5:1. If you have met this requirement, skip to step 15.
10. See Filter Selection to select the right filter type for the noise present in your system. The First Order IIR 1/4 filter is a good initial selection as it requires minimal SRAM and gives fast response.
11. Check if the SNR is now above 5:1. If so, confirm the scan time meets your design requirements. To check the scan time:
    - Click OK on the Tuner GUI to update the CapSense_CSD Component's parameters
    - Select the Scan Order tab to see the scan time

    If both the SNR and scan speed requirements are met, skip to step 15.
12. You can increase the resolution and lower the scan speed to further improve the SNR. However, both of these changes will increase the scan time.
13. Refer to step 11.
14. If the SNR is still below 5:1 you may need to look for improvements in the PCB layout or overlay design. Reducing overlay thickness and increasing button diameter increase sensitivity. See the Chapter 3 of Getting Started with CapSense guide for PCB design guidelines.
15. Use 75 percent of the peak finger response you measured while checking the SNR.
16. Use 15 percent of the peak finger response you measured while checking the SNR.
17. Use 40 percent of the peak finger response you measured while checking the SNR.
18. See Debounce. In general, the debouncing value should be set to 2. In fast scanning designs, the debounce value should be set to a higher value like 5.

**Note** You can manually tune parameters even when Tuning Method is set to Auto. If you alter the parameters using the Tuner GUI and click OK, the manually entered parameters will be applied to the component.

## 6.3 SmartSense Auto-Tuning

When you select Auto as the Tuning Method the SmartSense algorithm selects the CapSense_CSD component parameters. SmartSense maintains the SNR for each sensor between 5:1 and 11:1 to ensure robust CapSense operation while maximizing performance. You only need to set the four low-level parameters discussed in Parameter Settings. SmartSense is only available for IDAC sinking/IDAC sourcing methods. It is not available if you are using the Bleed Resistor method.

### 6.3.1 Guidelines

In order to use SmartSense Auto-Tuning:

- $C_P$ for all sensors must be 5—45 pF
- $C_F$ must be at least 0.1-pF
- $C_{MOD}$ capacitor X7R, 2.2 nF, voltage rating more than 5 V
- SmartSense supports the same APIs as manual tuning. However, you should not use APIs that modify the CapSense_CSD component parameters in firmware unless you understand exactly what effect it will have on the performance of your design.

### 6.3.2 Parameter Settings

#### *6.3.2.1 Sensor Auto Reset*

This parameter determines whether the baseline is updated at all times or only when the signal difference is below the noise threshold. When set to Enabled the baseline is updated constantly. This setting limits the maximum time that a sensor may remain on (typically it is 5 to 10 seconds), but it prevents the sensors from permanently turning on when the raw count suddenly rises without anything touching the sensor because of any failure condition of the system. See Sensor Auto Reset.

#### *6.3.2.2 Debounce*

The Debounce parameter adds a debounce counter to the sensor's active transition. For the sensor to be declared as active from inactive state, a finger touch signal should be present on the sensor for debounce number of consecutive scans. This parameter affects all of the sensors similarly. See Debounce.

#### *6.3.2.3 Modulator Capacitor Pin*

This parameter selects the pin to which the $C_{MOD}$ capacitor is connected. The available pins are P0[1] and P0[3].

**Note** An external 2.2 nF capacitor is mandatory for SmartSense to work correctly.

#### *6.3.2.4 Sensitivity Level*

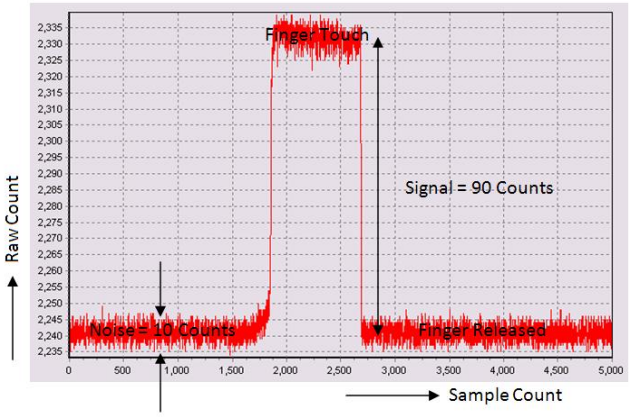
Sensitivity is used to increase or decrease sensor signal strength. Possible settings are High (0.1 pF), Medium High (0.2 pF), Medium Low (0.3 pF), and Low (0.4 pF).

Designs with thicker overlays require stronger signals for proper implementation .To produce a stronger signal from a sensor, SmartSense must use more time to scan the sensor. This means that a High sensitivity sensor takes longer to scan time than a Medium High sensitivity sensor.

When setting this parameter, use the lowest sensitivity value possible. To ensure robust performance:

1. Open the Tuner GUI.
2. Set the sensitivity level to Low(0.4 pF), and calculate the SNR. Figure 6-4 shows a typical raw count graph with a finger touch. Adjust the sensitivity level until SNR is more than 5:1 and less than 10:1.

Figure 6-4. Raw Count Graph for a Typical Sensor with a Finger Touch



# 7. Design Considerations

When designing capacitive touch-sense technology into your application, it is crucial to keep in mind that the CapSense device exists within a larger framework. Careful attention to every level of detail from PCB layout to user interface to end-use operating environment will lead to robust and reliable system performance. There is in-depth information given in the document Getting Started with CapSense for the Overlay Selection, ESD Protection and electromagnetic compatibility (EMC) Considerations, and so on. In this section, PSoC 3 and PSoC 5LP specific design considerations are explained.

## 7.1 Software Filtering

The CapSense_CSD component provides some readymade firmware filters to remove the noise readily from the CapSense scanning results. The following table describes these filters and when they should be used.

Table 7-1. Table of CapSense Filters Provided with Component

| Filter | Description | Application |
|---|---|---|
| **Median** | Nonlinear filter that computes median input value from a buffer of size 3 | Noise spikes from motors and switching power supplies |
| **Average** | Finite impulse response filter (no feedback) with equally weighted coefficients | Periodic noise from power supplies |
| **IIR** | Infinite impulse response filter (feedback) with a step response similar to an RC low pass filter | High frequency noise |
| **Jitter** | Nonlinear filter that limits current input based on previous input | Noise from thick overlay (SNR < 5:1), especially useful for slider centroid data |

**Notes**

1. Median Filter in the component uses a buffer of size two. The buffer contains two previous samples. The current sample and two previous samples stored in buffer are sorted and the middle value is taken as the median value.
2. Average Filter in the component uses a buffer of size two. The buffer contains two previous samples. Filter output is the average value of the two previous samples and current sample.

Figure 7-1. Filtering of Raw Counts
Black = Unfiltered Raw Counts
Green = Filtered Raw Counts

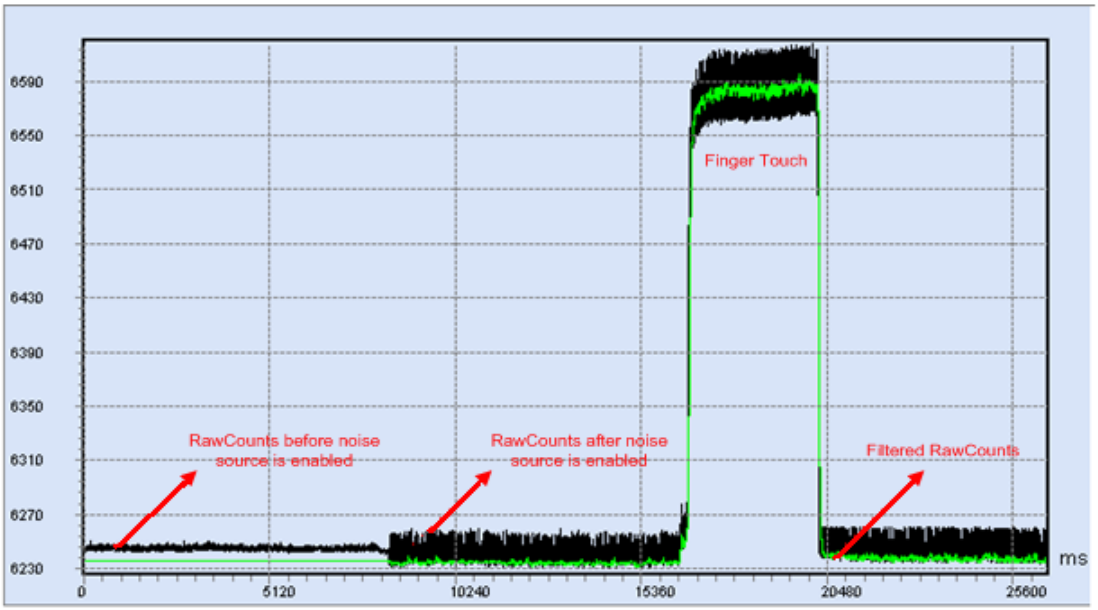



Figure 7-1 shows the effect of noise and filtering on raw counts. The noise was created by switching neighboring sensors at 500 kHz in order to simulate a noise due to high switching lines near to sensor. In this example:

- Noise without noise source = N = 8 counts (peak to peak)
- Noise with noise source= N1 = 40 counts (peak-to-peak)
- Noise after applying filter = N2 = 12 counts
- Signal (peak finger response) = S= 320 Counts
- SNR before applying filter = SNR1 = S / N1=320 / 40 = 8
- SNR after applying filter = SNR2 = S / N2= 320 /12 =26.7

## 7.2 Power Consumption

In battery-powered applications, lower current levels translate into longer battery life. With this in mind, a system designer should minimize the average current consumption of the system.

Table 7-2. PSoC3 Current Consumption for Scanning One Sensor

| CPU (MHz) | mA ($V_{DD}$=3.3 V) | mA ($V_{DD}$=5 V) |
|---|---|---|
| 3 | 3.7 | 4.7 |
| 6 | 4.4 | 5.5 |
| 12 | 5.9 | 6.9 |
| 24 | 8.4 | 9.5 |
| 48 | 14.5 | 16.3 |

There are several ways to reduce the power consumption of your CapSense capacitive touch-sensing system.

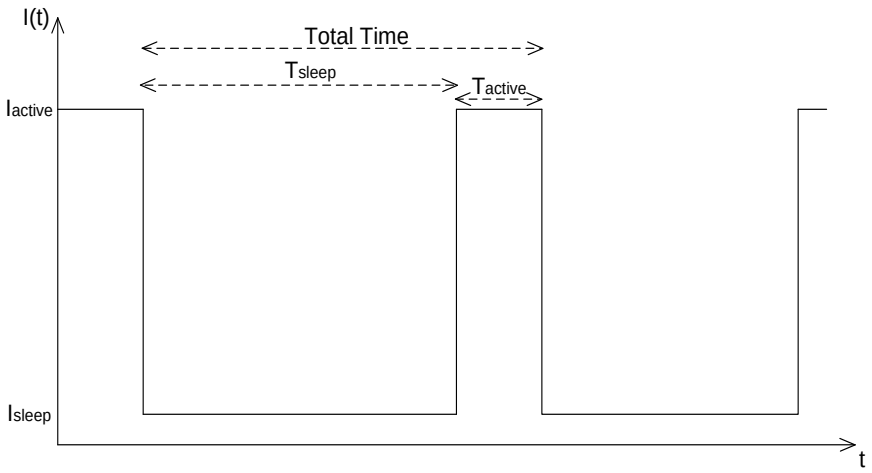
- **Set GPIO drive mode to HI-Z:** Be default all the GPIO pins are configured as HI-Z. But if the project has changed some pin drive mode configurations then they should be made HI-Z when not used.
- **Optimize CPU speed for low power:** The CPU does not need to be fast for CapSense operation, because the dedicated hardware scans the sensor. However, the CPU configures the hardware before it starts the scanning and after hardware completes the scanning the CPU processes the results. Therefore slowing down the CPU increases the scan time. See Non-Blocking Architecture to understand how the total scan time is divided into prescan, hardware scan and post-scan.
- **Operate at a lower $V_{DD}$.**

In addition to these suggestions, applying the sleep-scan method can be effective.

### 7.2.1 Sleep-Scan Method

In typical applications, the CapSense controller does not always need to be in the active state. The device can be put into the sleep state to stop the CPU and the major blocks of the device. Current consumed by the device in sleep state is much lower than the active current.

Figure 7-2. Sleep-Scan Method




The average current consumed by the device over a long time period can be calculated by using the following equation.

$$Iaverage = \frac{(Iactive \times Tactive) + (Isleep \times Tsleep)}{Total\ Time}$$ Equation 9

Where,

$I_{average}$ = Device average current

$I_{active}$ = Device active current

$T_{active}$ = Device active time

$I_{sleep}$ = Device sleep current

$T_{sleep}$ = Device sleep time

Total Time = $T_{active}$ + $T_{sleep}$

The average power consumed by the device can be calculated as follows:

$$Paverage = V_{DD} \times Iaverage$$ Equation 10

Where,

$P_{average}$ = Device average power

$V_{DD}$ = Device supply voltage

$I_{average}$ = Device average current

## 7.2.2 Measuring Average Power Consumption

### *7.2.2.1 Active Current*

To measure active current, program the device and measure the current for all the supply pins i.e. $V_{DDD}$, $V_{DDA}$, and $V_{DDIO}$. The best way to do this is to short the supply pins and measure the current at a common supply point.

### *7.2.2.2 Active Time*

You can measure active time by toggling a GPIO pin before and after the scan loop. The following code snippet shows a typical CapSense scan loop with pin set before scanning starts and reset after scanning is completed.

```
22      for(;;)
23      {
24          if (CapSense_IsBusy() ==0 )
25          {
26              Pin_Write(0);
27              CapSense_UpdateEnabledBaselines();
28              if (CapSense_CheckIsWidgetActive())
29              {
30              /* Do required */
31              }
32
33              /*Check other CapSense sensros active and do the required */
34
35              Pin_Write(1);
36              CapSense_ScanEnabledWidgets();
37          }
38          /* Other application code than CapSense */
39      }
40
41
42
43  }
44
```

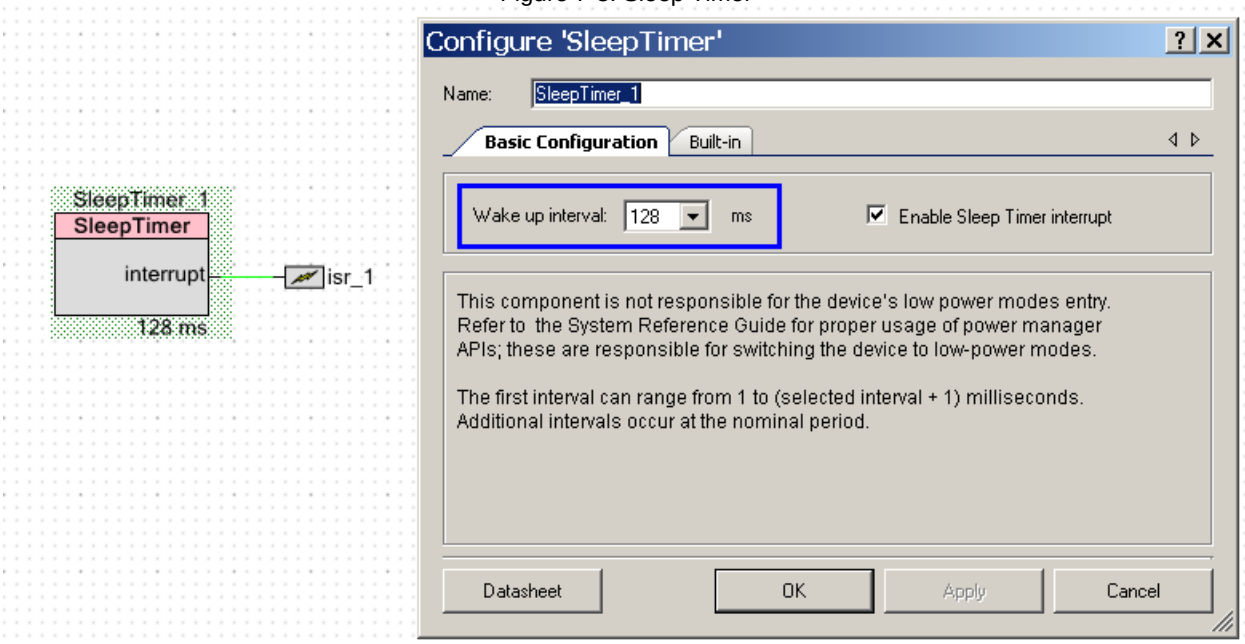
### *7.2.2.3 Sleep Current*

The sleep current value for the PSoC 3 is 1 µA and for PSoC 5LP is 2 µA. You can measure sleep current by creating a project that keeps the device in sleep mode forever and use the method described in Active Current.

### 7.2.2.4 Sleep Time

The API “CyPmSleep()” put the device into sleep mode. A Sleep Timer generates interrupt after the specified interval to wake up the device as shown in Figure 7-3.

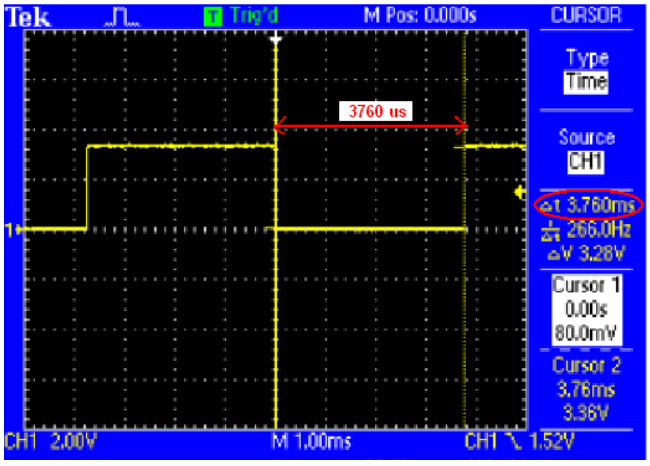
Figure 7-3. Sleep Timer




### 7.2.2.5 Calculating the Average Power

**Active Current** From Table 7-1, CPU = 24 MHz, operating at 3 V
$I_{active}$ = 8.4 mA.

Figure 7-4. Active Time Measurement




**Active Time** for PSoC 3, CPU = 24 MHz, Resolution 12-Bits and Scan Speed Normal
$T_{active}$ = 3760 µs.

**Sleep Current** from the datasheet
$I_{sleep}$ = 1 µA

**Sleep Time**, assume sleep period is set to 128 ms in sleep timer
Sleep Time = Total Time – Active Time
Total Time = Scan Interval = 128 ms
$T_{sleep}$ = 128,000 µs – 3760 µs= 124240 µs

**Average Current**, using Equation 9
$I_{average}$ = 247.7 µA

**Average Power**
$P_{average} = V_{DD} \times I_{average}$
$P_{average}$ = 3.3 V x 247.7 µA = 817.4 µW

The average power of the system using sleep-scan mode is 817.4 µW. While the average power of the system without using sleep mode is 27.7 mW.

## 7.3 Response Time

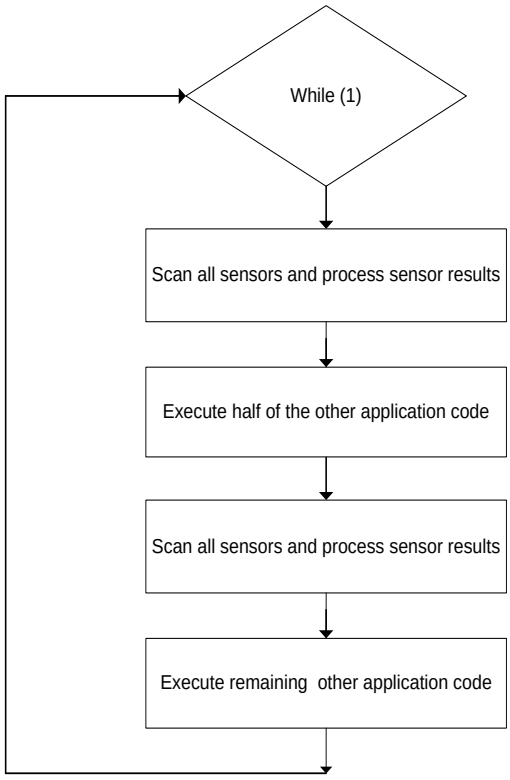
### 7.3.1 Sleep-Scan Mode

Putting the device into sleep mode between scans can save a significant amount of power. However, if you increase sleep time to a very high value it will result in poor response time. If both power consumption and response time are important in your design, you can use a method that incorporates both continuous-scan and sleep-scan modes. Using this approach, the device spends most of its time in sleep-scan mode where it scans the sensors and then goes to sleep. When a user touches a sensor, the device jumps to continuous-scan mode where the sensors are always scanned. The device remains in continuous-scan mode for a specified period. If the user does not touch a sensor within this period, the device returns to sleep-scan mode.

### 7.3.2 Long Background Loop

Response time can also suffer when the background loop spends too long executing code for applications other than CapSense. You can perform CapSense scanning multiple times per loop as shown in Figure 7-5.

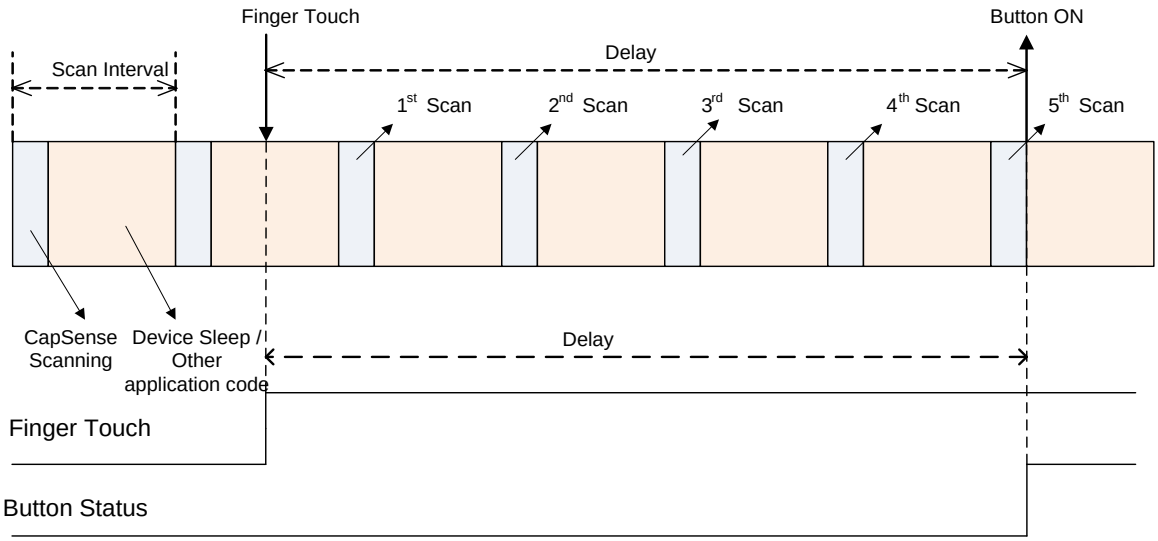
Figure 7-5. Scanning Multiple Time within the Loop to Improve the Response Time




### 7.3.3 Debounce

Debounce protects your system from detecting false finger touches, however, it also delays finger detection as shown in Figure 7-6.

Figure 7-6. Scan Interval and Detection Delay, Debounce Count = 5




The maximum delay due to debounce is:

$$Max\ Delay = Debounce \times Scan\ Interval$$

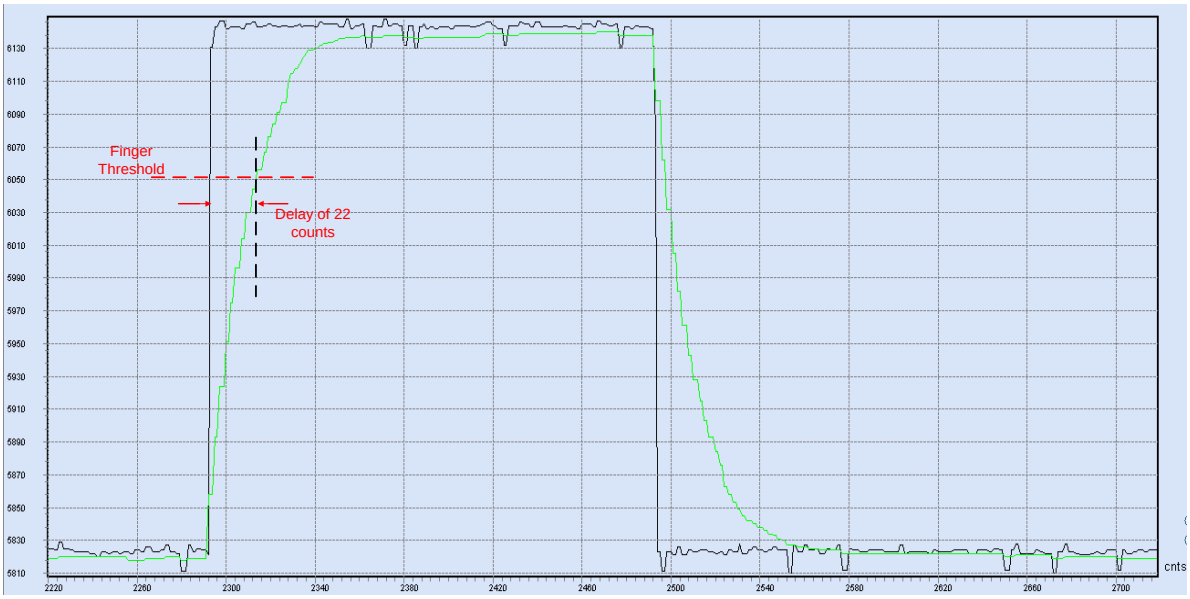
Debounce should set to 1 when you use sleep mode. Sleep mode delays response time, so it should effectively debounce the signal. If this is not adequate, you can scan a sensor multiple times once it detects a finger touch as shown in the following code snippet.

```
if (CapSense_CheckIsWidgetActive(CapSense_BUTTON0__BTN))
{
    /* Scan the sensor multiple times */
    for (i=0; i < DEBOUNCE ; i++)
    {
        if ( (CapSense_IsBusy() == 0 ) && CapSense_CheckIsWidgetActive(CapSense_BUTTON0__BTN) )
        {
            CapSense_ScanEnabledWidgets();
        }
    }
    /* After number of scans equal to debounce count is complete, check for sensor active.
    * If it is still active, then sensor can be considered as ON.
    */
    if (CapSense_CheckIsWidgetActive(CapSense_BUTTON0__BTN))
    {
        /*  Process the code which is based on sensor active */
    }
}
```

### 7.3.4 Filter Delay

Using filters will also delay finger touch detection. The effect of using an IIR16 filter is shown below. Note that IIR16 filter adds 1/16 of new data and 15/16 of old data to calculate the new filtered sample.

Figure 7-7. Filter Delay on the Finger Touch




Step change due to Finger = 320 counts

Finger threshold = 75% of peak finger response = 240 counts.

To calculate the delay caused by the filter:

$T_n = a \times r^{n-1}$

Where,

a = 1, step change or peak finger response

Tn = 0.25, because the 75% of the step change is threshold

r = 1/16, because the filter is IIR16

n = 22

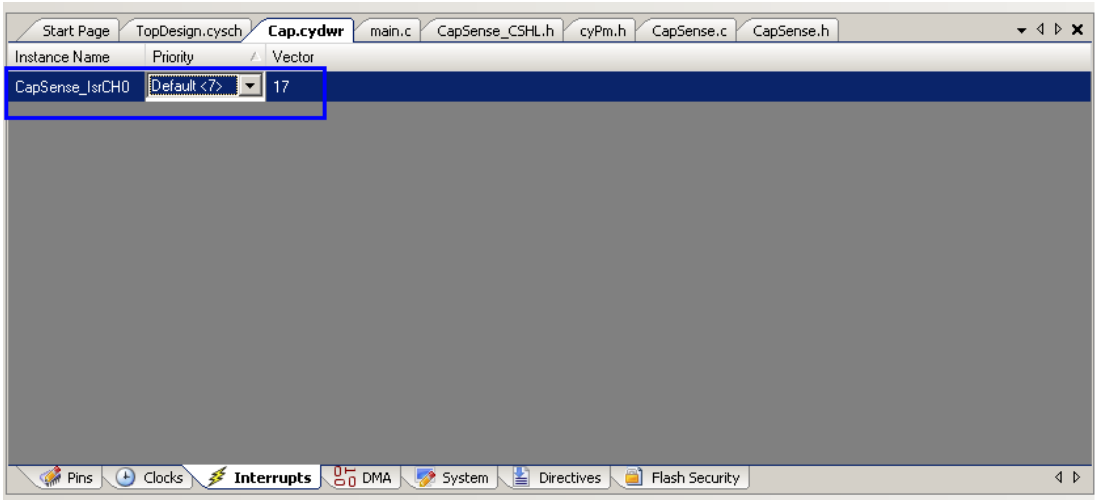
This means that finger touch is only detected after 22 scans. To minimize filter delay, choose the appropriate filter. For example, using an IIR4 filter will only cause a 7 scan delay.

If an IIR16 filter has to be used you should employ multiple scans as described in Debounce. However, the sensors should be scanned multiple times every loop regardless of their ON/OFF state.

### 7.3.5 Interrupt Priority

As discussed in Non-Blocking Architecture, the code architecture of the CapSense_CSD component is non-blocking in nature, which means the CPU can execute other application code when the hardware is scanning CapSense sensors. Once the hardware completes the scanning it generates an interrupt to CPU. To change the priority of the CapSense interrupt open the <project name>*.cydwr* file and click on the Interrupts tab.

Figure 7-8. Changing Interrupt Priority




The priority of the CapSense interrupt can be reduced if there are other higher priority interrupts such as communication protocols. If the CapSense interrupt is asserted when a high priority interrupt is being serviced, the CapSense interrupt will be serviced with delay. While this delay does not cause any data loss, it does delay scanning of the next sensor, and increases overall scanning time.

## 7.4 Pin Assignments

### 7.4.1 Opamp Output Pins

The opamp outputs are directly connected to P0[0], P0[1], P3[6], and P3[7]. If these pins are used for CapSense, the opamps become unusable. These pins have higher $C_P$ because they are directly connected to opamp outputs. Therefore, these four pins should not be used for CapSense. If you must use these pins, they should be used for buttons and should not be used as sliders or touch pads. Also, the traces for these pins should be short to help reduce $C_P$.

### 7.4.2 Pin Assignment for Two Channel Design

When two channel design is used, the pins belonging to particular channel should reside on the same side of the chip. See Number of Channels for more details on two channel design.

### 7.4.3 $C_{MOD}$ Pin assignment

For most efficient analog routing the following pins should be used for $C_{MOD}$:

- Left side: P2 [0], P2 [4], P6 [0], P6 [4], P15 [4]
- Right side: P1 [0], P1[4], P5 [0], P5 [4]

## 7.5 PCB Layout Guidelines

Detailed PCB layout guidelines are available in Getting Started with CapSense.

# 8. Resources

## 8.1 Website

Visit CapSense® PLUS which shows CapSense applications on PSoC 3 and PSoC 5LP.

## 8.2 Datasheet

The datasheets for the PSoC 3 and PSoC 5LP family of devices are available at www.cypress.com.

- PSoC 3 Datasheets
- PSoC 5LP Datasheets

## 8.3 Technical Reference Manual

The following technical reference manual to provide quick and easy access to information on PSoC 3 and PSoC 5LP architecture including top-level architectural diagrams, register summaries, and timing diagrams.

- PSoC 3 TRM
- PSoC 5LP TRM

## 8.4 Development Kits

### 8.4.1 PSoC 3 and PSoC 5LP Development Kits

- CY8CKIT-001 PSoC® Development Kit
- CY8CKIT-030 PSoC® 3 Development Kit
- CY8CKIT-050 PSoC® 5LP Development Kit

### 8.4.2 Interface Board to Attach Module Boards to Development Kit

- CY8CKIT-031 PSoC CapSense Expansion Board Kit

### 8.4.3 Universal CapSense Module Boards

#### *8.4.3.1 Simple Button Module Board*

The CY3280-BSM Simple Button Module consists of ten CapSense buttons and ten LEDs. This module connects to any CY3280 Universal CapSense Controller Board

#### *8.4.3.2 Matrix Button Module Board*

The CY3280-BMM Matrix Button Module consists of eight LEDs and eight CapSense sensors organized in a 4×4 matrix format to form 16 physical buttons. This module connects to any CY3280 Universal CapSense Controller Board.

#### 8.4.3.3 Linear Slider Module Board

The CY3280-SLM Linear Slider Module consists of five CapSense buttons, one linear slider (with ten sensors), and five LEDs. This module connects to any CY3280 Universal CapSense Controller Board.

#### 8.4.3.4 Radial Slider Module Board

The CY3280-SRM Radial Slider Module consists of four CapSense buttons, one radial slider (with ten sensors), and four LEDs. This module connects to any CY3280 Universal CapSense Controller Board

#### 8.4.3.5 Universal CapSense Prototyping Module

The CY3280-BBM Universal CapSense Prototyping Module provides access to every signal routed to the 44-pin connector on the attached controller boards. The Prototyping Module board is used in conjunction with a Universal CapSense Controller board to implement additional functionality that is not part of the other single-purpose Universal CapSense Module boards

### 8.4.4 MiniProg3

MiniProg3 is used to program and debug PSoC 3 and PSoC 5LP parts. It is also used as $I^2C$-USB communication bridge for Tuner GUI.

- CY8CKIT-002 PSoC® MiniProg3 Program and Debug Kit http://www.cypress.com/?rID=39045

## 8.5 PSoC Programmer

PSoC Programmer is a flexible, integrated programming application for programming PSoC devices. It can be used with PSoC Designer and PSoC Creator to program any design onto a PSoC device.

PSoC Programmer provides you a hardware layer with APIs to design specific applications using the programmers and bridge devices. The PSoC Programmer hardware layer is fully detailed in the COM guide documentation as well as example code across the following languages: C#, C, Perl, and Python.

## 8.6 Multi-Chart

Apart from Tuner GUI which uses $I^2C$ communication user can use UART for monitoring the CapSense results. Multi-Chart is a simple PC tool based on UART communication for real-time CapSense data viewing and logging. The application allows you to view data from up to 48 sensors, save and print charts, and save data for later analysis in a spreadsheet.

## 8.7 PSoC Creator

Cypress offers an exclusive Integrated Design Environment, PSoC Creator. With PSoC Designer you can configure the hardware and develop firmware in a single tool.

## 8.8 Code Examples

Cypress offers a large collection of code examples to get your design up and running fast. Open PSoC Creator, and click on ‘find example project’ link in the Start Page. Following two example projects are recommended for familiarizing with CapSense

- CapSense_CSD_Design
- CapSense_CSD_With Tuner

## 8.9 Design Support

Cypress has a variety of design support channels to ensure the success of your CapSense solutions.

- Knowledge Based Articles – Browse technical articles by product family or perform a search on various CapSense topics.
- CapSense Application Notes – Refer to a wide variety of application notes built on information presented in this document.
- White Papers – Learn about advanced capacitive-touch interface topics.
- Cypress Developer Community – Connect with the Cypress technical community and exchange information.
- Video Library – Quickly get up to speed with tutorial videos.
- Quality & Reliability – Cypress is committed to complete customer satisfaction. At our Quality website you can find reliability and product qualification reports.
- Technical Support – World class technical support is available online.

# Glossary

**AMUXBUS**

Analog multiplexer bus available inside PSoC that helps to connect I/O pins with multiple internal analog signals.

**SmartSense™ Auto-Tuning**

A CapSense algorithm that automatically sets sensing parameters for optimal performance after the design phase and continuously compensates for system, manufacturing, and environmental changes.

**Baseline**

A value resulting from a firmware algorithm that estimates a trend in the Raw Count when there is no human finger present on the sensor. The Baseline is less sensitive to sudden changes in the Raw Count and provides a reference point for computing the Difference Count.

**Button or Button Widget**

A widget with an associated sensor that can report the active or inactive state (that is, only two states) of the sensor. For example, it can detect the touch or no-touch state of a finger on the sensor.

**Difference Count**

The difference between Raw Count and Baseline. If the difference is negative, or if it is below Noise Threshold, the Difference Count is always set to zero.

**Capacitive Sensor**

A conductor and substrate, such as a copper button on a printed circuit board (PCB), which reacts to a touch or an approaching object with a change in capacitance.

**CapSense®**

Cypress's touch-sensing user interface solution. The industry's No. 1 solution in sales by 4x over No. 2.

**CapSense Mechanical Button Replacement (MBR)**

Cypress's configurable solution to upgrade mechanical buttons to capacitive buttons, requires minimal engineering effort to configure the sensor parameters and does not require firmware development. These devices include the CY8CMBR3XXX and CY8CMBR2XXX families.

**Centroid or Centroid Position**

A number indicating the finger position on a slider within the range given by the Slider Resolution. This number is calculated by the CapSense centroid calculation algorithm.

**Compensation IDAC**

A programmable constant current source, which is used by CSD to compensate for excess sensor $C_P$. This IDAC is not controlled by the Sigma-Delta Modulator in the CSD block unlike the Modulation IDAC.

**CSD**

CapSense Sigma Delta (CSD) is a Cypress-patented method of performing self-capacitance (also called self-cap) measurements for capacitive sensing applications.

In CSD mode, the sensing system measures the self-capacitance of an electrode, and a change in the self-capacitance is detected to identify the presence or absence of a finger.

**Debounce**

A parameter that defines the number of consecutive scan samples for which the touch should be present for it to become valid. This parameter helps to reject spurious touch signals.

A finger touch is reported only if the Difference Count is greater than Finger Threshold + Hysteresis for a consecutive Debounce number of scan samples.

**Driven-Shield**

A technique used by CSD for enabling liquid tolerance in which the Shield Electrode is driven by a signal that is equal to the sensor switching signal in phase and amplitude.

**Electrode**

A conductive material such as a pad or a layer on PCB, ITO, or FPCB. The electrode is connected to a port pin on a CapSense device and is used as a CapSense sensor or to drive specific signals associated with CapSense functionality.

**Finger Threshold**

A parameter used with Hysteresis to determine the state of the sensor. Sensor state is reported ON if the Difference Count is higher than Finger Threshold + Hysteresis, and it is reported OFF if the Difference Count is below Finger Threshold – Hysteresis.

**Ganged Sensors**

The method of connecting multiple sensors together and scanning them as a single sensor. Used for increasing the sensor area for proximity sensing and to reduce power consumption.

To reduce power when the system is in low-power mode, all the sensors can be ganged together and scanned as a single sensor taking less time instead of scanning all the sensors individually. When the user touches any of the sensors, the system can transition into active mode where it scans all the sensors individually to detect which sensor is activated.

PSoC supports sensor-ganging in firmware, that is, multiple sensors can be connected simultaneously to AMUXBUS for scanning.

**Gesture**

Gesture is an action, such as swiping and pinch-zoom, performed by the user. CapSense has a gesture detection feature that identifies the different gestures based on predefined touch patterns. In the CapSense component, the Gesture feature is supported only by the Touchpad Widget.

**Guard Sensor**

Copper trace that surrounds all the sensors on the PCB, similar to a button sensor and is used to detect a liquid stream. When the Guard Sensor is triggered, firmware can disable scanning of all other sensors to prevent false touches.

**Hatch Fill or Hatch Ground or Hatched Ground**

While designing a PCB for capacitive sensing, a grounded copper plane should be placed surrounding the sensors for good noise immunity. But a solid ground increases the parasitic capacitance of the sensor which is not desired. Therefore, the ground should be filled in a special hatch pattern. A hatch pattern has closely-placed, crisscrossed lines looking like a mesh and the line width and the spacing between two lines determine the fill percentage. In case of liquid tolerance, this hatch fill referred as a shield electrode is driven with a shield signal instead of ground.

**Hysteresis**

A parameter used to prevent the sensor status output from random toggling due to system noise, used in conjunction with the Finger Threshold to determine the sensor state. See Finger Threshold.

**IDAC (Current-Output Digital-to-Analog Converter)**

Programmable constant current source available inside PSoC, used for CapSense and ADC operations.

**Liquid Tolerance**

The ability of a capacitive sensing system to work reliably in the presence of liquid droplets, streaming liquids or mist.

**Linear Slider**

A widget consisting of more than one sensor arranged in a specific linear fashion to detect the physical position (in single axis) of a finger.

**Low Baseline Reset**

A parameter that represents the maximum number of scan samples where the Raw Count is abnormally below the Negative Noise Threshold. If the Low Baseline Reset value is exceeded, the Baseline is reset to the current Raw Count.

**Manual-Tuning**

The manual process of setting (or tuning) the CapSense parameters.

**Matrix Buttons**

A widget consisting of more than two sensors arranged in a matrix fashion, used to detect the presence or absence of a human finger (a touch) on the intersections of vertically and horizontally arranged sensors.

If M is the number of sensors on the horizontal axis and N is the number of sensors on the vertical axis, the Matrix Buttons Widget can monitor a total of M x N intersections using ONLY M + N port pins.

When using the CSD sensing method (self-capacitance), this Widget can detect a valid touch on only one intersection position at a time.

**Modulation Capacitor (CMOD)**

An external capacitor required for the operation of a CSD block in Self-Capacitance sensing mode.

**Modulator Clock**

A clock source that is used to sample the modulator output from a CSD block during a sensor scan. This clock is also fed to the Raw Count counter. The scan time (excluding pre and post processing times) is given by $(2^N – 1)$/Modulator Clock Frequency, where N is the Scan Resolution.

**Modulation IDAC**

Modulation IDAC is a programmable constant current source, whose output is controlled (ON/OFF) by the sigma-delta modulator output in a CSD block to maintain the AMUXBUS voltage at $V_{REF}$. The average current supplied by this IDAC is equal to the average current drawn out by the sensor capacitor.

**Mutual-Capacitance**

Capacitance associated with an electrode (say TX) with respect to another electrode (say RX) is known as mutual capacitance.

**Negative Noise Threshold**

A threshold used to differentiate usual noise from the spurious signals appearing in negative direction. This parameter is used in conjunction with the Low Baseline Reset parameter.

Baseline is updated to track the change in the Raw Count as long as the Raw Count stays within Negative Noise Threshold, that is, the difference between Baseline and Raw count (Baseline – Raw count) is less than Negative Noise Threshold.

Scenarios that may trigger such spurious signals in a negative direction include: a finger on the sensor on power-up, removal of a metal object placed near the sensor, removing a liquid-tolerant CapSense-enabled product from the water; and other sudden environmental changes.

**Noise (CapSense Noise)**

The variation in the Raw Count when a sensor is in the OFF state (no touch), measured as peak-to-peak counts.

**Noise Threshold**

A parameter used to differentiate signal from noise for a sensor. If Raw Count – Baseline is greater than Noise Threshold, it indicates a likely valid signal. If the difference is less than Noise Threshold, Raw Count contains nothing but noise.

**Overlay**

A non-conductive material, such as plastic and glass, which covers the capacitive sensors and acts as a touch-surface. The PCB with the sensors is directly placed under the overlay or is connected through springs. The casing for a product often becomes the overlay.

**Parasitic Capacitance ($C_P$)**

Parasitic capacitance is the intrinsic capacitance of the sensor electrode contributed by PCB trace, sensor pad, vias, and air gap. It is unwanted because it reduces the sensitivity of CSD.

**Proximity Sensor**

A sensor that can detect the presence of nearby objects without any physical contact.

**Radial Slider**

A widget consisting of more than one sensor arranged in a specific circular fashion to detect the physical position of a finger.

**Raw Count**

The unprocessed digital count output of the CapSense hardware block that represents the physical capacitance of the sensor.

**Refresh Interval**

The time between two consecutive scans of a sensor.

**Scan Resolution**

Resolution (in bits) of the Raw Count produced by the CSD block.

**Scan Time**

Time taken for completing the scan of a sensor.

**Self-Capacitance**

The capacitance associated with an electrode with respect to circuit ground.

**Sensitivity**

The change in Raw Count corresponding to the change in sensor capacitance, expressed in counts/pF. Sensitivity of a sensor is dependent on the board layout, overlay properties, sensing method, and tuning parameters.

**Sense Clock**

A clock source used to implement a switched-capacitor front-end for the CSD sensing method.

**Sensor**

See Capacitive Sensor.

**Sensor Auto Reset**

A setting to prevent a sensor from reporting false touch status indefinitely due to system failure, or when a metal object is continuously present near the sensor.

When Sensor Auto Reset is enabled, the Baseline is always updated even if the Difference Count is greater than the Noise Threshold. This prevents the sensor from reporting the ON status for an indefinite period of time. When Sensor Auto Reset is disabled, the Baseline is updated only when the Difference Count is less than the Noise Threshold.

**Sensor Ganging**

See Ganged Sensors.

**Shield Electrode**

Copper fill around sensors to prevent false touches due to the presence of water or other liquids. Shield Electrode is driven by the shield signal output from the CSD block. See Driven-Shield.

**Shield Tank Capacitor ($C_{SH}$)**

An optional external capacitor ($C_{SH}$ Tank Capacitor) used to enhance the drive capability of the CSD shield, when there is a large shield layer with high parasitic capacitance.

**Signal (CapSense Signal)**

Difference Count is also called Signal. See Difference Count.

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

The ratio of the sensor signal, when touched, to the noise signal of an untouched sensor.

**Slider Resolution**

A parameter indicating the total number of finger positions to be resolved on a slider.

**Touchpad**

A Widget consisting of multiple sensors arranged in a specific horizontal and vertical fashion to detect the X and Y position of a touch.

**Trackpad**

See Touchpad.

**Tuning**

The process of finding the optimum values for various hardware and software or threshold parameters required for CapSense operation.

**$V_{REF}$**

Programmable reference voltage block available inside PSoC used for CapSense and ADC operation.

**Widget**

A user-interface element in the CapSense component that consists of one sensor or a group of similar sensors. Button, proximity sensor, linear slider, radial slider, matrix buttons, and touchpad are the supported widgets.

# Revision History

## Document Revision History

| Document Title: AN75400 - PSoC® 3 and PSoC® 5LP CapSense® Design Guide<br>Document Number: 001-75400 | | | |
|---|---|---|---|
| **Revision** | **Issue Date** | **Origin of Change** | **Description of Change** |
| ** | 01/11/2012 | PVKV | New Design Guide |
| *A | 11/23/2012 | NIDH | Updated for PSoC 5LP |
| *B | 08/18/2014 | NIDH | Updated figure 5-7 and the text underneath to match the actual left and right sides of the analog routing.<br>Updated figure 6-3 to add reference to the Getting Started with CapSense guide. |
| *C | 01/21/2016 | VAIR | Added Glossary. |