AN66271 - CY8C21x34/B

CapSense® Design Guide

Document No. 001-66271 Rev. *H

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Copyrights

Contents

1. Introduction

1.1 Abstract

This document gives design guidance for using capacitive touch sensing (CapSense®) functionality with the CY8C21x34/B family of CapSense Plus™ Controllers. The following topics are covered in this guide:

- Features of the CY8C21x34/B family of CapSense Plus Controllers
- CapSense principles of operation
- Introduction to CapSense design tools
- Detailed guide to tuning the CapSense touch sensing system for optimal performance
- Advanced features such as water tolerance and proximity detection
- Electrical and mechanical system design considerations
- Additional resources and support for designing CapSense into your system

1.2 Cypress's CapSense Documentation Ecosystem

Figure 1-1 and Table 1-1 summarize the Cypress CapSense documentation ecosystem. These resources allow implementers to quickly access the information needed to complete a CapSense product design successfully.

Figure 1-1 shows the typical flow of a product design cycle with capacitive sensing; the information in this guide is most pertinent to the topics highlighted in green. Table 1-1 provides links to the supporting documents for each of the numbered tasks in Figure 1-1.

Figure 1-1. Typical CapSense Product Design Flow

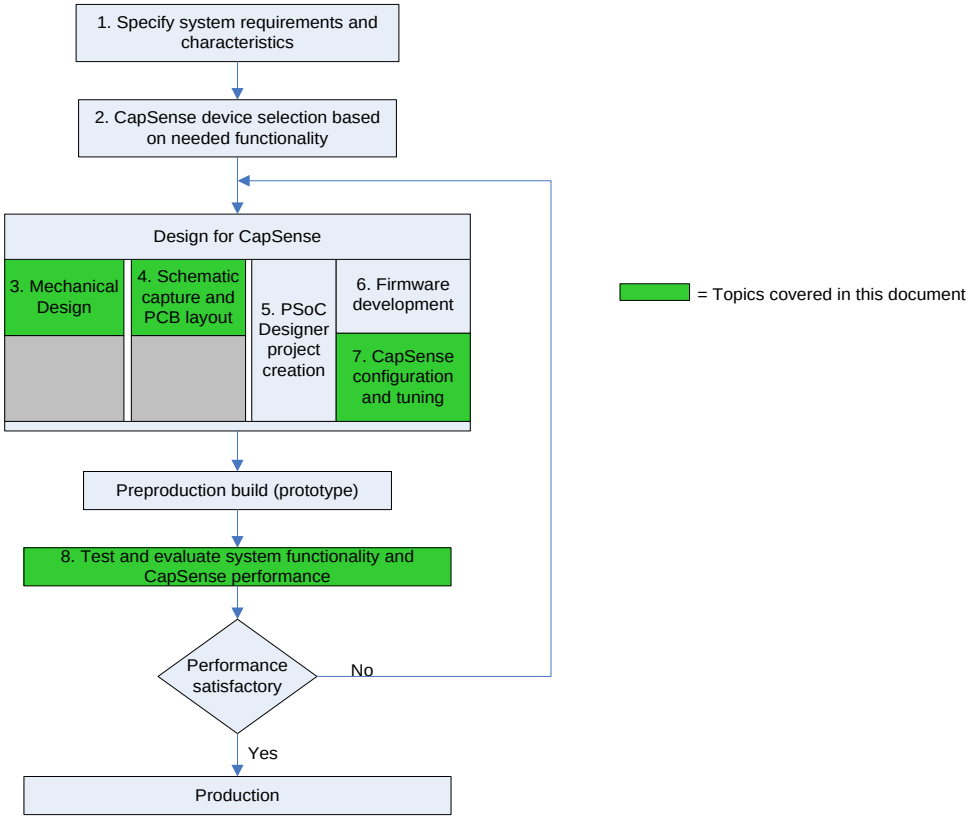


Table 1-1. Cypress Documents Supporting Numbered Design Tasks of Figure 1-1

Numbered Design Task of Figure 1-1	Supporting Cypress CapSense Documentation
1	Getting Started with CapSense Getting Started with PSoC 1
2	Getting Started with CapSense CY8C21x34/B CapSense Device Datasheets
3	Getting Started with CapSense PSoC family-specific CapSense Design Guide (this document) CapSense Application Notes
4	Getting Started with CapSense PSoC family-specific CapSense® Design Guide (this document) CapSense Application Notes
5	PSoC Designer™ User Guides
6	CapSense Application Notes CapSense Code Examples PSoC family-specific Technical Reference Manual (for CY8C21x34/B) AN2014 - Basics of PSoC® 1 Programming AN2015 - PSoC® 1 - Getting Started with Flash & E2PROM AN44168 - PSoC® 1 Device Programming using External Microcontroller (HSSP) PSoC®1 ISSP Programming Specifications

Numbered Design Task of Figure 1-1	Supporting Cypress CapSense Documentation
7	PSoC family-specific CapSense Design Guide (this document) PSoC family-specific CapSense User Module Datasheets (CSD, CSDADC, and SmartSense) PSoC family-specific Technical Reference Manual (for CY8C21x34/B) CapSense Controller Code Examples Design Guide
8	PSoC family-specific CapSense Design Guide (this document) CapSense Code Examples

1.3 CY8C21x34/B CapSense Plus Family Features

Cypress's CY8C21x34/B is a low-power, high-performance, programmable CapSense controller family that features:

1.3.1 Advanced Touch Sensing Features

- Programmable capacitive sensing elements
  - Supports a combination of CapSense buttons, sliders, and proximity sensors with CSD and CSDADC capacitive sensing technology
  - Supports SmartSense™ Auto-tuning (CY8C21x34B)
  - Contains an integrated API to implement buttons and sliders
  - Supports up to 24 capacitive buttons and four sliders
  - Supports proximity sensing up to 5 cm (with on-board PCB trace)
  - Provides water-tolerant performance with shield electrode

1.3.2 Device Features

- High-performance, low-power M8C Harvard-architecture processor
  - M8C processor speeds up to 24 MHz
- Flexible on-chip memory
  - Up to 8 KB of flash and 512 B of SRAM
  - Emulated EEPROM
- Precision, programmable clocking
  - Internal main oscillator (IMO): 24/48 MHz ± 2.5%
  - Internal low-speed oscillator (ILO) at 32 kHz for watchdog and sleep
- Enhanced GPIO features
  - Up to 28 GPIOs with programmable pin configuration
  - 25-mA sink current on all GPIOs
  - Internal resistive pull-up, Hi-Z, open-drain, and strong drive modes on all GPIOs
- Peripheral features
  - Counter, timer, pulse width modulator (PWM), pulse width discriminator (PWD), and hardware 7-segment drive
  - I^2C communication with Master, Slave, and Multimaster configurations
  - SPI, UART, IRDA, and one-wire communication protocols
- Operating conditions
  - Operating voltage: 2.4 V to 5.25 V with operation down to 1.0 V by using on-chip switch mode pump
  - Temperature range: –40 °C to +85 °C
- Packaging
  - 16-pin SOIC (150-MIL) to 32-pin QFN (5×5 mm)
  - AEC -qualified automotive grade parts – CY8C21334 and CY8C21534

1.4 Document Conventions

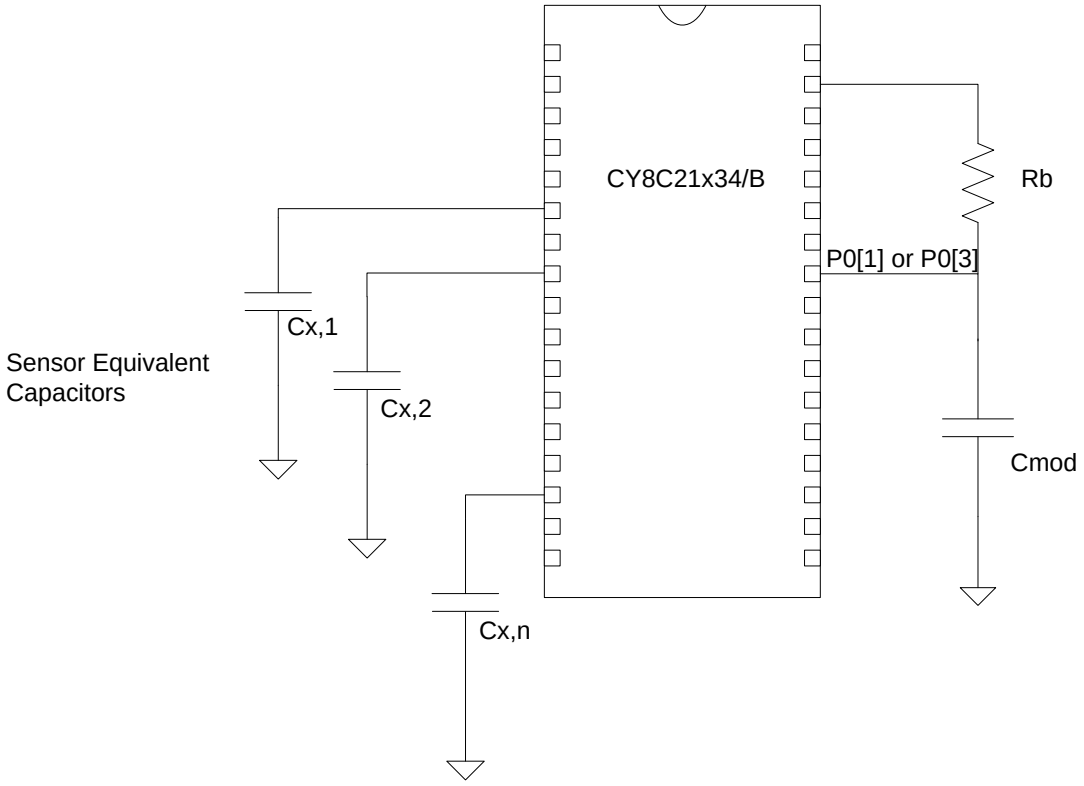
Convention	Usage
Courier New	Displays file locations, user entered text, and source code: C:\ ...cd\icc\
Italics	Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
[Bracketed, Bold]	Displays keyboard commands in procedures: [**Enter**]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

2. CapSense Technology

2.1 CapSense Fundamentals

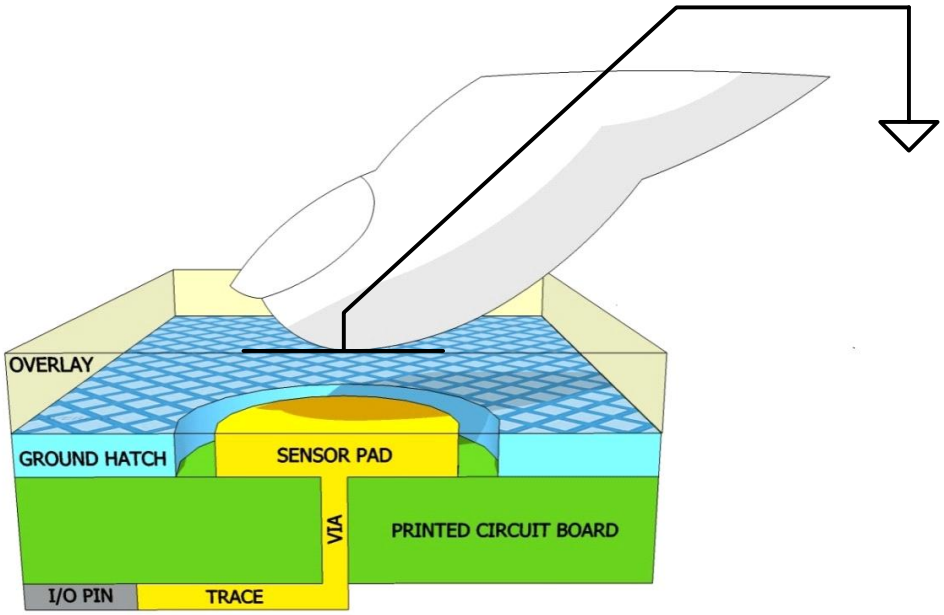
CapSense is a touch-sensing technology that works by measuring the capacitance of each I/O pin on the CapSense controller that is designated as a sensor. As shown in Figure 2-1, the total capacitance on each of the sensor pins can be modeled as equivalent lumped capacitors with values of $C_{x,1}$ through $C_{x,n}$ for a design with n sensors. Circuitry internal to the CY8C21x34/B device converts the magnitude of each C_x into a digital code that is stored for post processing. The other components, namely R_b and C_{MOD}, are used by the CapSense controller's internal circuitry.

Figure 2-1. CapSense Implementation in a CY8C21x34/B PSoC Device



As shown in Figure 2-2, each sensor I/O pin is connected to a sensor pad by traces, vias, or both as necessary. The overlay is a nonconductive cover over the sensor pad that constitutes the product'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 is represented in Figure 2-2. This arrangement constitutes a parallel plate capacitor; its capacitance is given by Equation 1.

Figure 2-2. Cross Section of Typical CapSense PCB with the Sensor being Activated by a Finger



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

where:

C_F = Capacitance affected by a finger in contact with the overlay over a sensor

ε_0 = Free space permittivity

ε_r = Dielectric constant (relative permittivity) of overlay

A = Area of finger and sensor pad overlap

D = Overlay thickness

In addition to the parallel plate capacitance, a finger in contact with the overlay causes electric field fringing between itself and other conductors in the immediate vicinity. The effect of these fringing fields is typically minor compared to that of the parallel plate capacitor and can usually be ignored.

Even without a finger touching the overlay, the sensor I/O pin has some parasitic capacitance (C_P). C_P results from the combination of the CapSense controller internal parasitic and the electric field coupling between the sensor pad, traces, vias, and other conductors in the system. These conductors can be the ground plane, other traces, any metal in the product's chassis or enclosure, and so on. The CapSense controller measures the total capacitance (C_X) connected to a sensor pin.

When a finger is not touching a sensor:

$$C_X = C_P$$ Equation 2

When a finger is on the sensor pad:

$$C_X = C_P + C_F$$ Equation 3

In general, C_P is an order of magnitude greater than C_F. C_P usually ranges from 6 pF to 15 pF, but in extreme cases can be as high as 50 pF. C_F usually ranges from 0.1 pF to 0.4 pF. The magnitude of C_P is of critical importance when tuning a CapSense system and is discussed in CapSense Performance Tuning with User Modules.

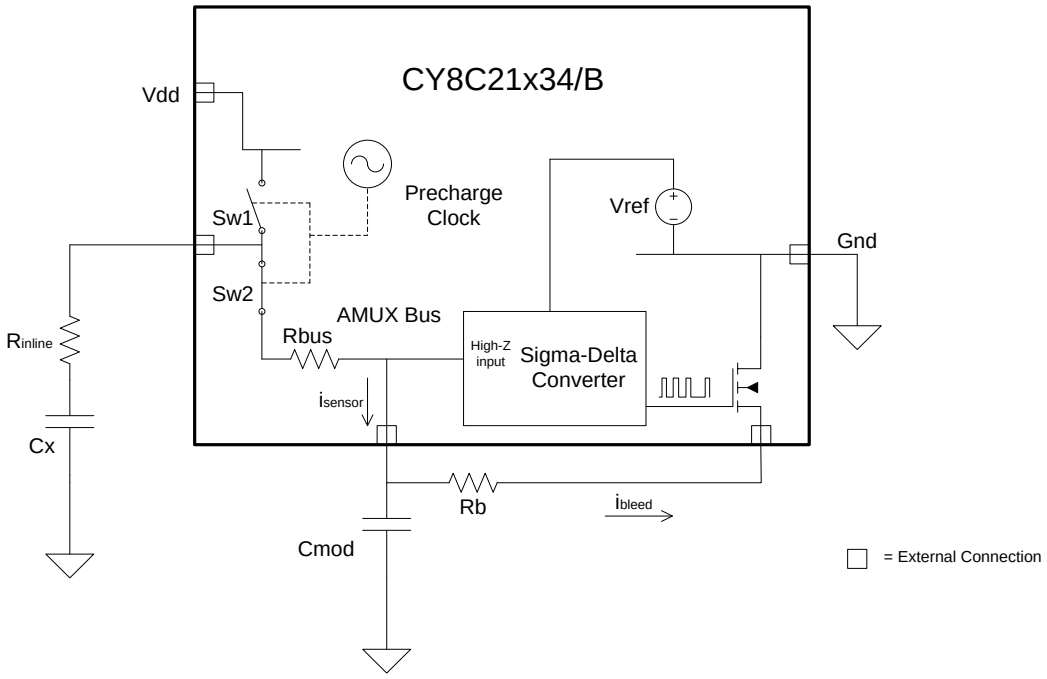
2.2 CapSense Methods in CY8C21x34/B

CY8C21x34/B devices support several CapSense methods for converting sensor capacitance (C_x) into digital counts. These are CapSense Sigma-Delta (CSD), CSD with ADC (CSDADC), and SmartSense. These methods are implemented in the form of PSoC Designer User Modules and are described in Section 3 of this design guide.

2.2.1 CapSense Sigma-Delta (CSD)

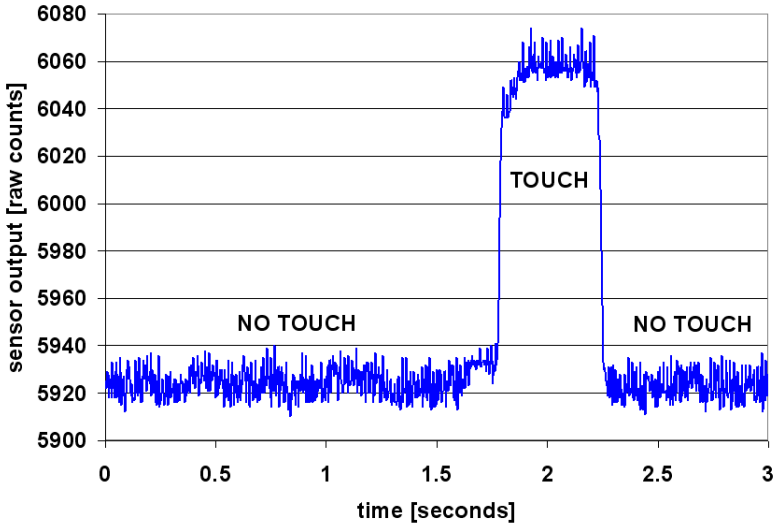
The CSD method in the CY8C21x34/B devices incorporates C_x into a switched-capacitor circuit, as shown in Figure 2-3. The sensor (C_x) is alternatively connected to V_{DD} and the analog mux (AMUX) bus by the underlapped switches Sw1 and Sw2, respectively. Sw1 and Sw2 are driven by a Precharge clock to generate a current (I_{SENSOR}) into the AMUX bus. The magnitude of I_{SENSOR} is directly proportional to the magnitude of C_x. The sigma-delta converter samples AMUX bus voltage and generates a modulating bit stream that controls the constant current source (I_{DAC}), which charges AMUX such that the average AMUX bus voltage is maintained at V_{REF}. The sensor bleeds off the charge I_{SENSOR} from the modulating capacitor (C_{MOD}). $C_{MOD,}$ in combination with Rbus, forms a low-pass filter that attenuates precharge switching transients at the sigma-delta converter input.

Figure 2-3. CSD Block Diagram



In maintaining the average AMUX voltage at a steady state value (V_{REF}), the Sigma-Delta converter matches the average bleed current (I_{BLEED}) to I_{SENSOR} by controlling the bit stream duty cycle. The Sigma-Delta converter stores the bit stream over the duration of a sensor scan and the accumulated result is a digital output value, known as raw count, which is directly proportional to C_x. This raw count is interpreted by high-level algorithms to resolve the sensor state. Figure 2-4 plots the CSD raw counts from a number of consecutive scans during which the finger touches and then releases the sensor. As explained in CapSense Fundamentals, the finger touch causes C_x to increase by $C_{F,}$ which in turn causes raw counts to increase proportionally. By comparing the shift in steady state raw count level to a predetermined threshold, the high-level algorithms can determine whether the sensor is in the ON (Touch) or OFF (No Touch) state.

Figure 2-4. CSD Raw Counts during a Finger Touch



2.2.2 CSD with ADC Functionality (CSDADC)

The CSDADC CapSense method functions in an identical manner as CSD except that it includes the ADC functionality in addition to capacitance measurement.

2.2.2.1 ADC Features

- Absolute and ratiometric ADC input modes with run-time mode switching, which allows you to easily scan different sensor types
- Single-slope ADC for absolute voltage input mode
- Built-in calibration mechanism for the single-slope ADC
- Integrating incremental ADC for ratiometric input mode

2.2.3 SmartSense Auto-Tuning

Tuning the touch-sensing user interface is a critical step in ensuring proper system operation and a pleasant user experience. The typical design flow involves tuning the sensor interface in the initial design phase, during system integration, and finally production fine-tuning before the production ramp. Tuning is an iterative process and can be time consuming. SmartSense Auto-tuning is developed to simplify the user interface development cycle. Further, it is easy to use and significantly reduces the design cycle time by eliminating the tuning process throughout the entire product development cycle, from prototype to mass production. SmartSense tunes each CapSense sensor automatically at power-up and then monitors and maintains optimum sensor performance during runtime. SmartSense Auto-tuning technology adapts for manufacturing variation in PCBs, overlays, and noise generators such as LCD inverters, AC line noise, and switch-mode power supplies, and automatically tunes them out. SmartSense Auto-tuning requires between 2.1 KB to 3.7 KB of flash depending on configuration and 29B of RAM per sensor.

2.2.3.1 Process Variation

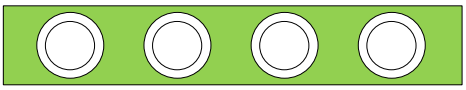
The SmartSense User Module (UM) for the CY8C21x34B is designed to work with sensor parasitic capacitance in the range of 5 pF to 45 pF (typical sensor C_P values are in the range of 10 pF to 20 pF). The sensitivity parameter for each sensor is set automatically, based on the characteristics of that particular sensor. This improves the yield in mass production, because consistent response is maintained from every sensor regardless of C_P variation between sensors within the specified range of 5 pF to 41 pF. Parasitic capacitance of the individual sensors can vary due to PCB layout, PCB manufacturing process variation, or with vendor-to-vendor PCB variation within a multisourced supply chain. The sensor sensitivity depends on its parasitic capacitance; higher C_P values decrease the sensor sensitivity and result in decreased finger touch signal amplitude. In some cases, the change in C_P value detunes the system, resulting in less than optimum sensor performance (either too sensitive or not sensitive enough) or, worst case, a nonoperational

sensor. In either situation, you must retune the system, and in some cases requalify the UI subsystem. SmartSense Auto-tuning solves these issues.

SmartSense Auto-tuning makes platform designs possible. Imagine the capacitive touch sensing multimedia keys in a laptop computer; the spacing between the buttons depends on the size of the laptop and keyboard layout. In this example, the wide-screen machine has larger spaces between the buttons compared to a standard-screen model. More space between buttons means increased trace length between the sensor and the CapSense controller, which leads to higher parasitic capacitance of the sensor. This means that the parasitic capacitance of the CapSense buttons can be different in different models of the same platform design. Though the functionality of these buttons is the same for all laptop models, the sensors must be tuned for each model. SmartSense enables you to do platform designs using the recommended best practices shown in the PCB Layout in Getting Started with CapSense, knowing the tuning will be done efficiently and automatically.

Figure 2-5. Design of Laptop Multimedia Keys for a 21-inch Model

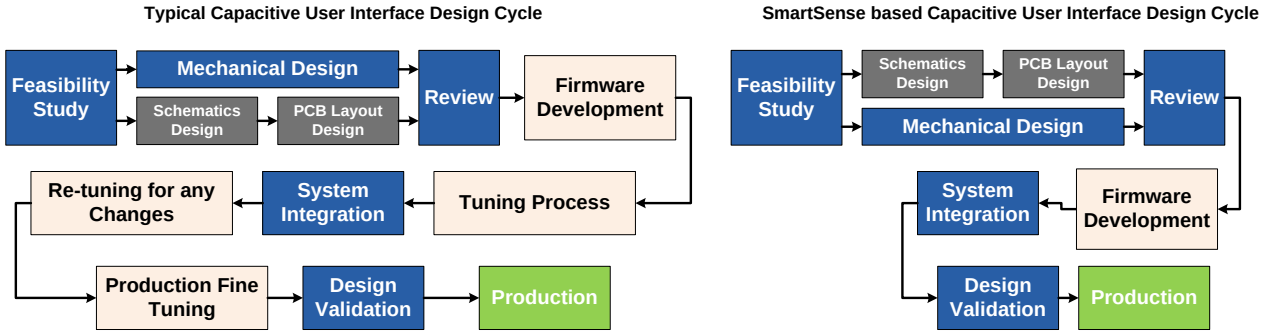
Figure 2-6. Design of Laptop Multimedia Keys for a 15-inch Model with Identical Functionality and Button Size



2.2.3.2 Reduced Design Cycle Time

Usually, the most time-consuming task for a capacitive sensor interface design is firmware development and sensor tuning. With a typical touch-sensing controller, the sensor must be retuned when the same design is ported to different models or when there are changes in the mechanical dimensions of the PCB or the sensor PCB layout. A design with SmartSense solves these challenges because it needs less firmware development effort, no tuning, and no retuning. This makes a typical design cycle much faster. Figure 2-7 compares the design cycles of a typical touch-sensing controller and a SmartSense-based design.

Figure 2-7. Typical Capacitive Interface Design Cycle Comparison



3. CapSense Design Tools

3.1 Overview

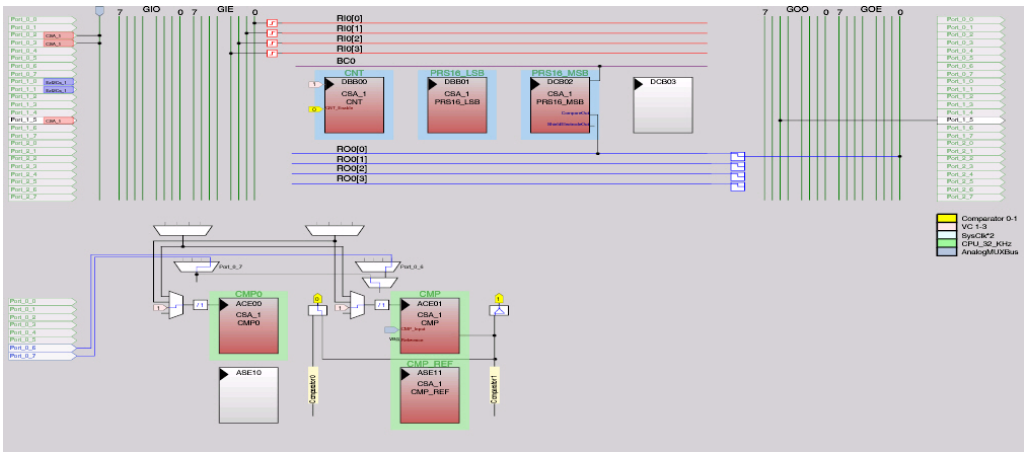
Cypress offers a full line of hardware and software tools for developing your CapSense capacitive touch-sense application. A basic development system for the CY8C21x34/B family includes the components discussed in this chapter. See Resources for ordering information.

3.1.1 PSoC Designer and User Modules

Cypress's exclusive integrated design environment, PSoC Designer, allows you to configure analog and digital blocks, develop firmware, and tune and debug your design. Applications are developed in a drag-and-drop design environment using a library of user modules. User modules are configured either through the Device Editor GUI or by writing into specific registers with firmware. PSoC Designer comes with a built-in C compiler and an embedded programmer. A pro compiler is available for complex designs.

CSD, CSDADC, and SmartSense UMs implement capacitive touch sensors using switched-capacitor circuitry, an analog multiplexer, a comparator, digital counting functions, and high-level software routines (APIs). User modules for other analog and digital peripherals are available to implement additional functionality such as I^2C, SPI, TX8, timers, and PWMs.

Figure 3-1. PSoC Designer Device Editor



3.1.1.1 Getting Started with CapSense User Modules

To create a new CY8C21x34/B project in PSoC Designer:

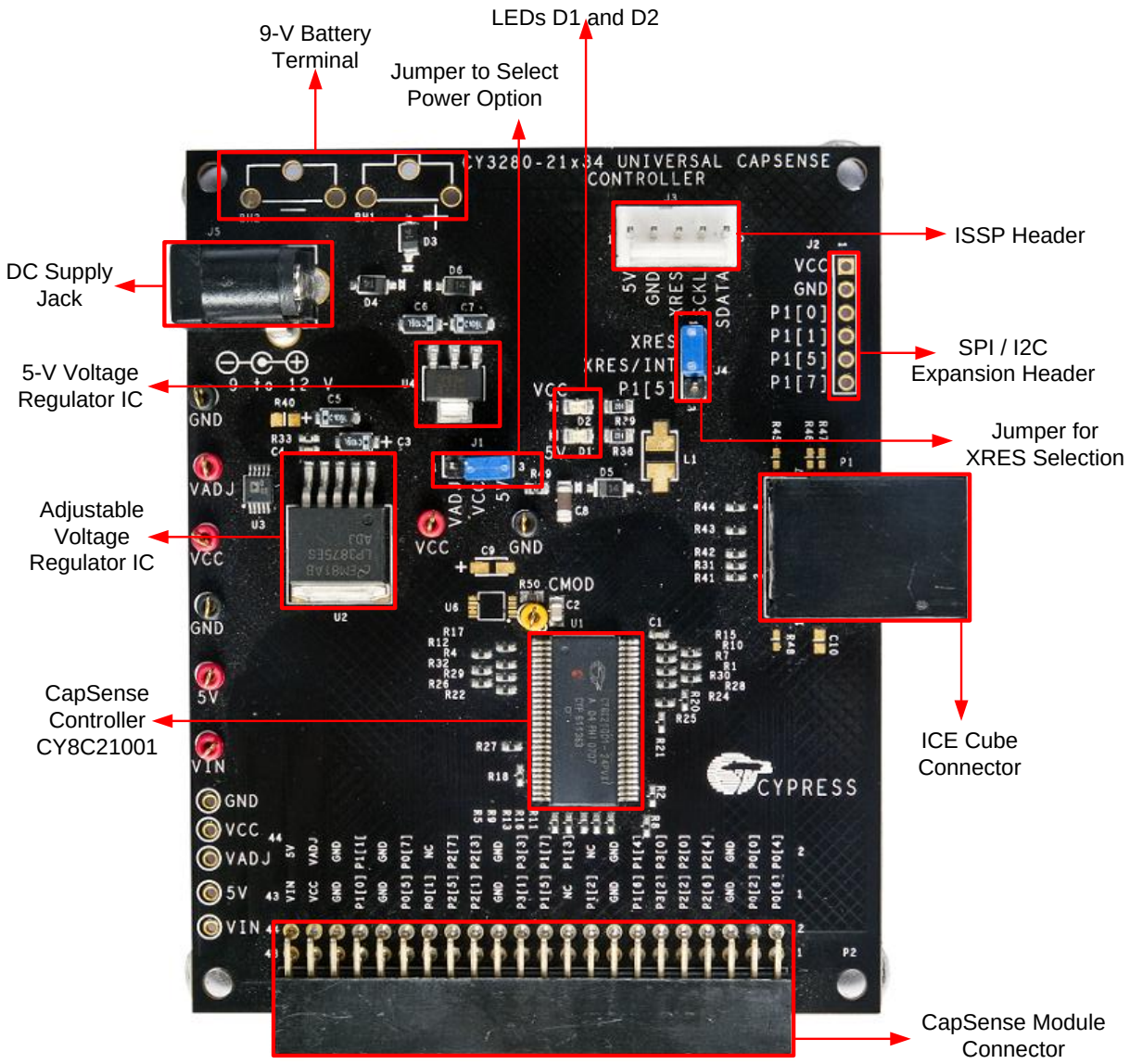
1. Create a new CY8C21x34/B PSoC Designer project.
2. Select and place any user modules requiring dedicated pins (such as I^2C or LCD) and assign ports and pins.
3. Select and place the CSD, CSDADC, or SmartSense UM.
4. Right-click the user module to access the User Module wizard.
5. Set button sensor count, slider configuration, pin assignments, and associations.

6. Set pins and global user module parameters.
7. Generate the application and switch to the Application Editor.
8. Adapt sample code from User Module Datasheet: (CY8C21x34) to implement buttons or sliders.

3.1.2 Universal CapSense Controller Kit

The Universal CY3280-BK1 CapSense Controller Kit features predefined control circuitry and plug-in hardware to make prototyping and debugging easy. Programming and I²C-to-USB Bridge hardware are included for tuning and data acquisition.

Figure 3-2. CY3280-21x34 CapSense Controller Kit



3.1.3 Universal CapSense Controller Module Board

Cypress's module boards feature a variety of sensors, LEDs, and interfaces to meet your application's needs.

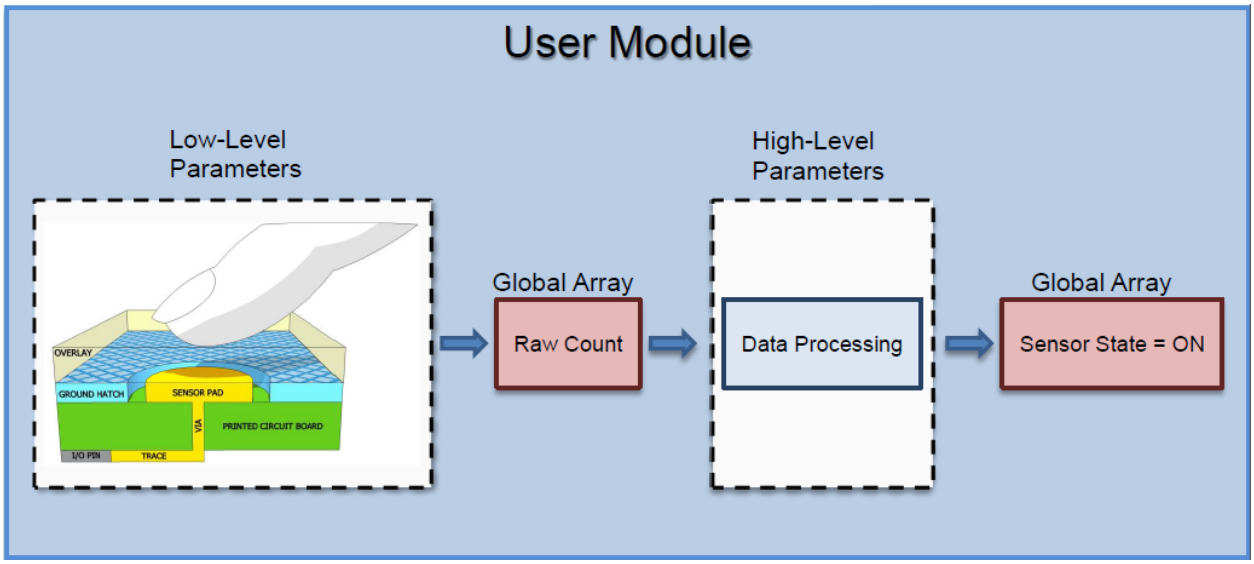
- CY3280-BSM Simple Button Module
- CY3280-BMM Matrix Button Module
- CY3280-SLM Linear Slider Module
- CY3280-SRM Radial Slider Module
- CY3280-BBM Universal CapSense Prototyping Module

3.1.4 CapSense Data Viewing Tools

Often during the CapSense design process, you will want to monitor CapSense data (raw counts, baseline, difference counts, and so on) for tuning and debugging purposes. This can be done with two CapSense data viewing tools, MultiChart and Bridge Control Panel. The application note, AN2397 – CapSense Data Viewing Tools, discusses these tools in detail.

3.2 User Module Overview

Figure 3-3. User Module Block Diagram



User modules contain an entire CapSense system from physical sensing to data processing. The behavior of the user module is defined using several parameters. These parameters affect different parts of the sensing system and can be separated into low-level and high-level parameters that communicate with one another using global arrays.

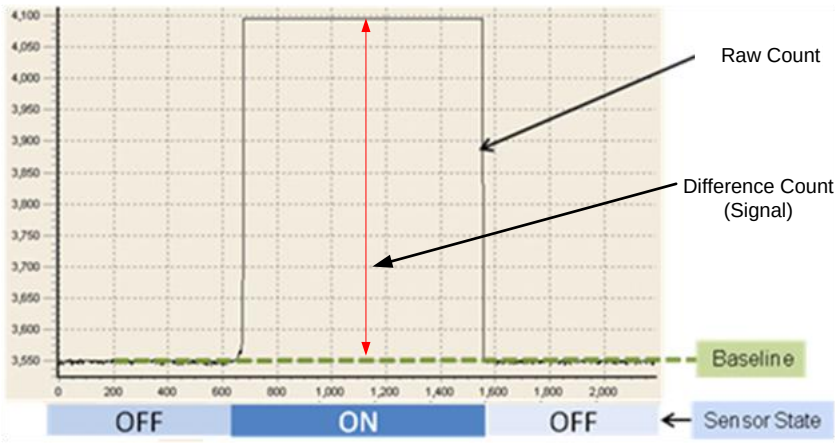
Low-level parameters, such as the speed and resolution for scanning sensors, define the behavior of the sensing method at the physical layer and relate to the conversion from capacitance to raw count. Low-level parameters are unique to each type of sensing method and are described in CSDADC User Module Configurations, CSDADC User Module Parameters, and SmartSense User Module Parameters.

High-level parameters, such as debounce counts and noise thresholds, define how the raw counts are processed to produce information such as the sensor ON/OFF state and the estimated finger position on a slider. These parameters are the same for all sensing methods and are described in CSD User Module High-Level Parameters.

3.3 CapSense User Module Global Arrays

Before learning CapSense User Module parameters, you must be familiar with certain global arrays used by the CapSense system. These arrays should not be altered manually, but may be inspected for debugging purposes.

Figure 3-4. Raw Count, Baseline, Difference Count, and Sensor State



3.3.1 Raw Count

The hardware circuit in the CapSense controller measures the sensor capacitance. It stores the result in a digital form called raw count upon calling the user module API *UMname*_ScanSensor(), where *UMname* can be CSD, CSDADC, or SmartSense.

The raw count of a sensor is proportional to its sensor capacitance. Raw count increases as the sensor capacitance value increases. The raw count values of sensors are stored in the *UMname*_waSnsResult[] integer array. This array is defined in the header file *UMname.h*.

3.3.2 Baseline

Gradual environmental changes such as temperature and humidity affect the sensor raw count, which results in variations in the counts.

The user module uses a complex baselining algorithm to compensate for these variations. The algorithm uses baseline variables to accomplish this. The baseline variables track any gradual variations in raw count values. Essentially, the baseline variables hold the output of a digital low-pass filter to which input raw count values are fed. The baselining algorithm is executed by the user module API *UMname*_UpdateSensorBaseline, where *UMname* can be CSD, CSDADC, or SmartSense.

The baseline values of sensors are stored in *UMname*_waSnsBaseline[] integer array. This array is defined in the header file *UMname.h*.

3.3.3 Difference Count (Signal)

The Difference Count, also known as the signal of a sensor, is defined as the difference in counts between a sensor's raw count and baseline values. When the sensor is inactive, the Difference Count is zero. Activating sensors (by touching) results in a positive Difference Count value.

The Difference Count values of sensors are stored in the *UMname*_waSnsDiff[] integer array, where *UMname* can be CSD, SmartSense, or CSA_EMC. This array is defined in the header file *UMname*.h. Difference count variables are updated by the user module API *UMname*_UpdateSensorBaseline().

3.3.4 Sensor State

Sensor state represents the active/inactive status of the physical sensors. The state of the sensor changes from 0 to 1 upon finger touch and returns to 0 upon finger release.

Sensor states are stored in a byte array named *UMname*_baSnsOnMask[]array, where *UMname* can be CSD, CSDADC, or SmartSense. Each array element stores the sensor state of eight consecutive sensors. Sensor states are updated by the user module API *UMname*_bIsAnySensorActive().

3.3.5 Signal

This array is applicable only to the SmartSense User Module. The difference count is normalized to get the signal. Usually the signal is zero when the senor is inactive. When the sensor is touched, it causes the raw count to increase, and results in a positive signal value. This array is used for tuning the SmartSense User Module when threshold mode parameter is set to manual mode.

Signal values are stored in an integer array named SmartSense_baSnsSignal[].

3.4 CSD User Module Configurations

The CSD User Module has two selectable clock configurations. These configurations use different signal sources for the switched-capacitor circuit on the front end of the CSD, as shown in Figure 2-3 on page 13.

Select the clock configuration when you first place the CSD User Module in your PSoC Designer project. You can change this selection later by right-clicking the CSD User Module and selecting **User Module Selection Options**.

3.4.1 CSD without Clock Prescaler

In this configuration, a 16-bit pseudo-random sequence (PRS) is used as the signal source for the switched-capacitor circuit on the front end of the CSD. System clock (IMO) is the clock source for the PRS. The PRS spreads the clock's frequency spectrum and provides good noise immunity. This configuration requires three digital blocks.

3.4.2 CSD with Clock Prescaler

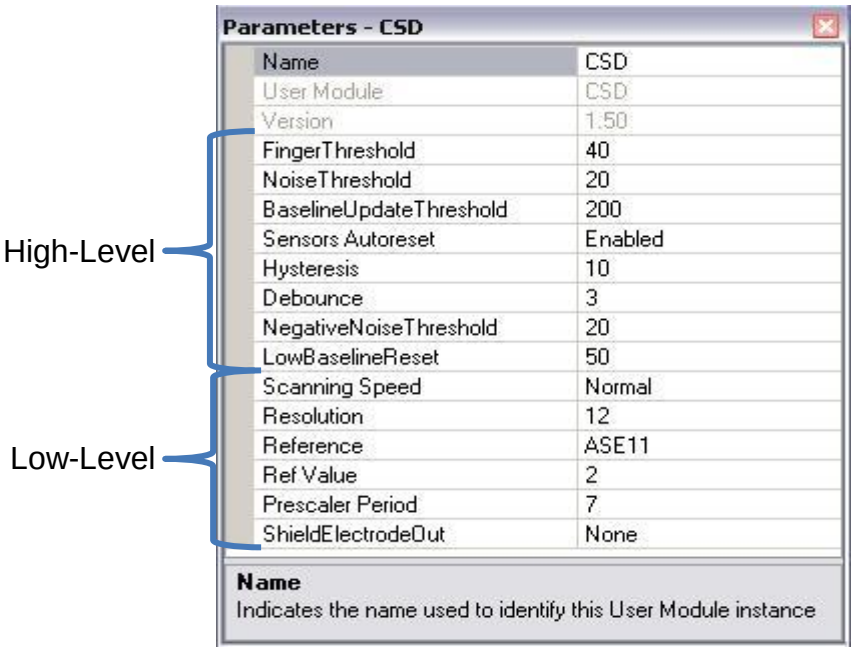
In this configuration, an 8-bit PRS is used as the signal source for the switched capacitor circuit on the front end of the CSD. IMO is divided by an adjustable prescaler divider and is used as the clock source for the PRS. The PRS spreads the clock's frequency spectrum to provide noise immunity. Note that the 16-bit PRS used in the CSD without the Clock Prescaler configuration provides better noise immunity. This configuration requires three digital blocks.

Because it needs a lower switching frequency, use this configuration when operating in an environment with high C_P. The relationship between C_P and switching frequency is discussed in Select Prescaler.

3.5 CSD User Module Parameters

The CSD User Module parameters are classified into high-level and low-level parameters. See Figure 3-5 for a list of CSD User Module parameters and how they are classified.

Figure 3-5. PSoC Designer - CSD Parameters Window



3.6 CSD User Module High-Level Parameters

3.6.1 Finger Threshold

The user module uses the Finger Threshold parameter to judge the active/inactive state of a sensor. If the Difference Count value of a sensor is greater than the Finger Threshold value, the sensor is judged as active. This definition assumes that the hysteresis level is set to zero and Debounce is set to 1.

Possible values are 5 to 255.

3.6.2 Noise Threshold

The user module uses the Noise Threshold value to interpret the upper limit of noise counts in the raw count. For individual sensors, the baselining update algorithm is paused when the raw count is greater than the baseline and the difference between them is greater than this threshold.

For slider sensors, the centroid calculation is paused when the difference count is greater than the Noise Threshold value.

Possible values are 3 to 255. For proper user module operation, the Noise Threshold value should never be set higher than Finger Threshold value minus the Hysteresis setting.

3.6.3 Baseline Update Threshold

When the raw count value is above the current baseline and the difference count is below the Noise Threshold, the difference between the current baseline and the raw count is accumulated into a "bucket." When the bucket fills completely, the baseline increments and the bucket is emptied. The Baseline Update Threshold parameter sets the threshold that the bucket must reach for the baseline to increment.

Possible values are 0 to 255.

3.6.4 Sensors Autoreset

The Sensors Autoreset parameter determines whether the baseline is updated at all times, or only when the difference counts are less than the Noise Threshold value.

When Sensors Autoreset is enabled, the baseline is updated constantly. This limits the maximum duration of the sensor 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 voltage fluctuation in the power supply, a high-energy RF noise source, or a very quick temperature change.

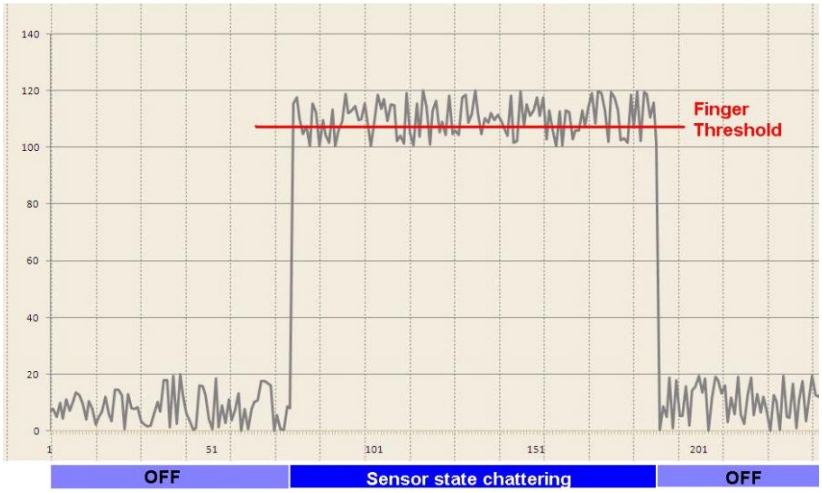
When Sensors Autoreset is disabled, the baseline is updated only when the difference counts are less than the Noise Threshold parameter.

Possible values are Enabled and Disabled.

3.6.5 Hysteresis

The Hysteresis setting prevents the sensor 'ON' state from chattering because of system noise. The function of hysteresis is given in Equation 4. The equation assumes that Debounce is set to 1.

Figure 3-6. Sensor State versus Difference Count with Hysteresis Set to Zero



$$if DifferenceCount \geq FingerThreshold + Hysteresis, SensorState = ON$$
$$if DifferenceCount \leq FingerThreshold - Hysteresis, \ SensorState = OFF$$ Equation 4

3.6.6 Debounce

The Debounce parameter prevents spikes in raw counts from changing the sensor state from OFF to ON. For the sensor state to transition from OFF to ON, the Difference Count value must remain greater than the Finger Threshold value plus the hysteresis level for the number of samples specified.

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

3.6.7 Negative Noise Threshold

The Negative Noise Threshold parameter acts as a negative difference count threshold. If the raw count is less than 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.

3.6.8 Low Baseline Reset

The Low Baseline Reset parameter works in conjunction 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.

3.6.9 High-Level Parameter Recommendations

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

- **Finger Threshold**: Set to 75 percent of Difference Counts with sensor ON
- **Noise Threshold**: Set to 40 percent of Difference Counts with sensor OFF
- **Baseline Update Threshold**: Set to twice the Noise Threshold
- **Sensors Autoreset**: Based on design requirements
- **Hysteresis**: Set to 15 percent of Difference Counts with sensor ON
- **Debounce**: Based on design requirements
- **Negative Noise Threshold**: Set equal to Noise Threshold
- **Low Baseline Reset**: Set to 10

3.7 CSD User Module Low-Level Parameters

The CSD User Module has several low-level parameters in addition to the high-level parameters. These parameters are specific to the CSD sensing method and determine how raw count data is acquired from the sensor.

3.7.1 Scanning Speed

This parameter sets the sensor scanning speed. Although a faster scanning speed provides a good response time, slower scanning speeds give the following advantages:

- Improved SNR
- Better immunity to power supply and temperature changes
- Less demand for system interrupt latency; you can handle longer interrupts

See Table 3-1 and Table 3-2 for scanning speed and resolution performance. Possible values are Ultra Fast, Fast, Normal, and Slow

3.7.2 Resolution

This parameter determines the scanning resolution in bits. The maximum raw count for scanning resolution of N bits is 2^N–1. Increasing the resolution improves sensitivity and SNR, but reduces scan time. See Table 3-1 and Table 3-2 for scanning speed and resolution performance. Possible values are 9 to 16 bits.

Table 3-1. Scanning Speed and Resolution for 24-MHz IMO Operation, without Clock Prescaler (PRS16 Configuration)

Resolution (bits)	Scanning Speed (µs)			
	Ultra Fast	Fast	Normal	Slow
9	75	110	170	300
10	110	170	300	510
11	170	300	510	1010
12	300	510	1010	2030
13	510	1010	2030	4060
14	850	1690	3380	6760
15	1520	3040	6080	12200
16	2880	5720	11500	23200

Table 3-2. Scanning Speed and Resolution for 24-MHz IMO Operation, with Prescaler (PRS8 Configuration)

Resolution (bits)	Scanning Speed (µs)			
	Ultra Fast	Fast	Normal	Slow
9	60	85	150	255
10	85	150	255	510
11	150	255	510	1020
12	255	510	1020	2040
13	510	1020	2010	4080
14	845	1700	3380	6760
15	1530	3060	6120	12100
16	2880	5800	11500	23000

3.7.3 Reference

A voltage reference to the input of the comparator is required for the proper operation of CSD.

Possible values are:

- VBG: Internal voltage reference of 1.3 V derived from a fixed bandgap reference.
- ASE11: Internal variable voltage reference derived from a PWM.
- AnalogColumn_InputSelect_1: External voltage reference such as a resistor divider network or externally filtered PWM/PRSPWM signal. This reference can be connected to a CapSense controller pin and used as the comparator reference source.

3.7.4 Ref Value

The effect of this parameter depends on the Reference parameter selection. This value has no effect when the reference comes from the VBG or from AnalogColumn_InputSelect_1. When the reference comes from ASE11, this parameter sets the reference value.

Possible values are 0 (minimum ¼ V_{DD}) to 8 (maximum ¾ V_{DD}).

3.7.5 Prescaler Period

The sensor capacitance is switched continuously between two potentials while scanning. This parameter sets the prescaler period register and determines the precharge switch output frequency. This parameter is available only for CSD with the clock prescaler configuration.

Possible values are 1 to 255.

3.7.6 Shield Electrode Out

Shield Electrode Out should be enabled for applications requiring water-tolerant performance. The shield electrode signal source can be selected from one of the free digital row buses (Row_0_Output_1 to Row_0_Output_3). Each row output can be routed to one of three pins. Set the Row LUT Function to A.

Possible values are Enabled and Disabled.

3.8 CSDADC User Module Configurations

The CSDADC combines capacitance sensing with ADC functionality. It saves code space by reusing modules common to both the CSD and ADC. You should use the CSDADC when your design requires both capacitance sensing and ADC functionality. Applications that need one or the other should use the CSD User Module or the ADC8 or ADC10 User Modules.

The CSDADC User Module has four selectable clock configurations: PRS16, PWM8, PRS8, and VC2. These configurations use different signal sources for the switched capacitor circuit on the front end of the CSD.

You should select the clock configuration when you first place the CSDADC User Module into your PSoC Designer project. You can change this selection later by right-clicking the CSDADC User Module and selecting **User Module Selection Options**.

3.8.1 CSDADC with PRS16 Clock Source

In this configuration, a 16-bit pseudo-random sequence (PRS) is used as the signal source for the switched-capacitor circuit on the front end of the CSD. System clock (IMO) is the clock source for the PRS. The PRS spreads the clock's frequency spectrum and provides good noise immunity. This configuration requires three digital blocks.

3.8.2 CSDADC with PRS8 Clock Source

In this configuration, an 8-bit PRS is used as the signal source for the switched-capacitor circuit on the front end of the CSD. System clock (IMO) is the clock source for the PRS. The PRS spreads the clock's frequency spectrum to give noise immunity. Note that the 16-bit PRS used in the CSD without the Clock Prescaler configuration provides better noise immunity. This configuration requires three digital blocks.

Because it needs a lower switching frequency, you should use configuration when operating in an environment with high C_P, The relationship between C_P and switching frequency is discussed in CapSense Performance Tuning with User Modules.

3.8.3 CSDADC with PWM8 Clock Source

In this configuration, IMO is divided by an adjustable prescaler divider and used for the switched capacitor circuit on the front end of the CSD. This configuration is more sensitive to noise at the frequency of operation and its harmonics. This configuration requires two digital blocks.

This configuration is suitable for sensing through high-resistance materials (such as ITO) or when you want low operation frequency because of emission problems.

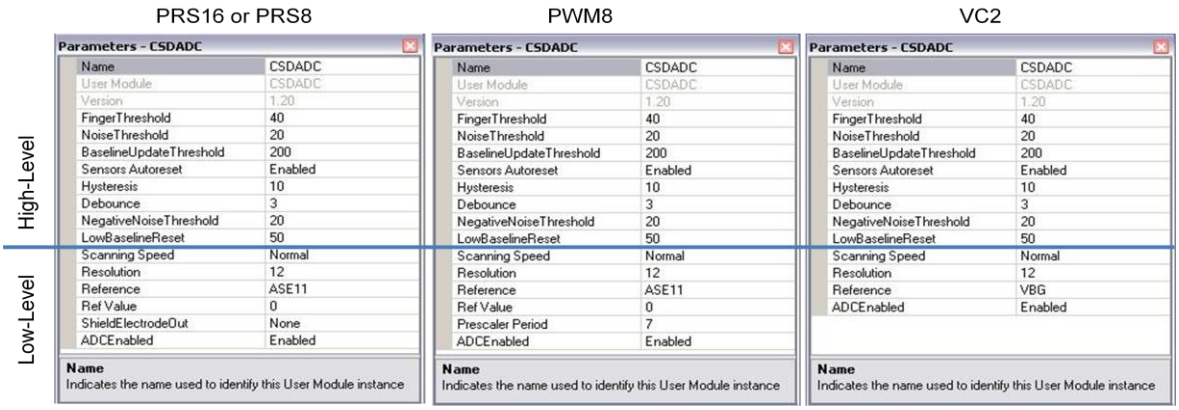
3.8.4 CSDADC with VC2 Clock Source

In this configuration, the system clock is divided using an adjustable divider, VC2, and used for the switched-capacitor circuit on the front end of the CSD. This configuration is more sensitive to noise signals at the frequency of operation and its harmonics. This configuration requires one digital block.

This configuration is suitable for sensing through high-resistance materials (such as ITO) or when you want low operation frequency because of emission problems.

3.9 CSDADC User Module Parameters

Figure 3-7. PSoC Designer - CSDADC Parameters Windows



3.10 Low-Level Parameters

3.10.1 Scanning Speed

This parameter sets sensor scanning speed. Although faster scanning speed provides good response time, slower scanning speeds provide the following advantages:

- Improved SNR
- Better immunity to power supply and temperature changes
- Less demand for system interrupt latency; you can handle longer interrupts

See Table 3-1 and Table 3-2 for scanning speed and resolution performance. Possible values are Ultra Fast, Fast, Normal, and Slow.

3.10.2 Resolution

This parameter determines the scanning resolution in bits. The maximum raw count for scanning resolution of N bits is 2^N–1. Increasing the resolution improves sensitivity and SNR, but reduces scan time. See Table 3-1 and Table 3-2 for scanning speed and resolution performance.

Possible values are 9 to 16 bits.

3.10.3 Reference

A voltage reference to the input of the comparator is required for the proper operation of CSD.

Possible values are:

- VBG: Internal voltage reference of 1.3 V derived from a fixed band-gap reference.
- ASE11: Internal variable voltage reference derived from a PWM.
- AnalogColumn_InputSelect_1: External voltage reference such as a resistor divider network or externally filtered PWM/PRSPWM signal. This reference can be connected to a CapSense controller pin and used as the comparator reference source.

3.10.4 Ref Value

The effect of Ref Value depends on the Reference parameter. When Reference is set to ASE11, Ref Value sets the reference value.

Possible values are 0 (minimum ¼ V_{DD}) to 8 (maximum ¾ V_{DD}).

Increasing Ref Value decreases sensitivity, but increases the influence on the shield electrode. If the design has sensors with noticeable capacitance differences (sensor pads of different sizes), you can use an API function to balance raw counts by setting a higher Ref Value for the sensors with larger capacitance.

Ref Value has no effect when Reference is set to VBG or AnalogColumn_InputSelect_1. Ref Value is not available in CSDADC with the VC2 clock source configuration.

3.10.5 Prescaler Period

Prescaler Period determines the precharge switch output frequency.

Possible values are 1 to 255. The recommended values are $2^N - 1$ to obtain the maximum signal-to-noise ratio (1, 3, 7, 15, 31, 63, 127, or 255). Other values can result in more noise, especially at low resolution and high scan speed.

Prescaler Period is available only for configurations with prescaler.

3.10.6 Shield Electrode Out

The shield electrode signal source can be selected from one of the free digital row buses (Row_0_Output_1 to Row_0_Output_3). Each row output can be routed to one of three pins. Set the Row LUT Function to A.

Possible values are Enabled and Disabled.

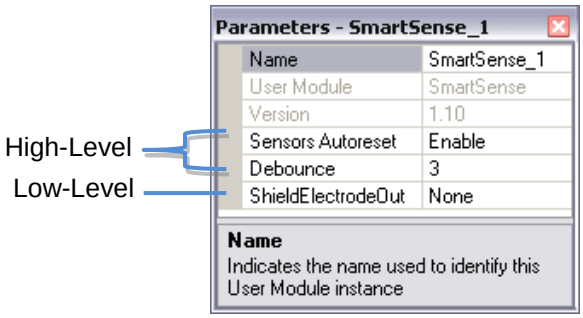
This parameter is available only in CSDADC with the PRS8/PRS16 clock source configurations. In CSDADC with the PWM8 clock source configuration, the shield electrode signal is permanently connected to Row_0_Output_0.

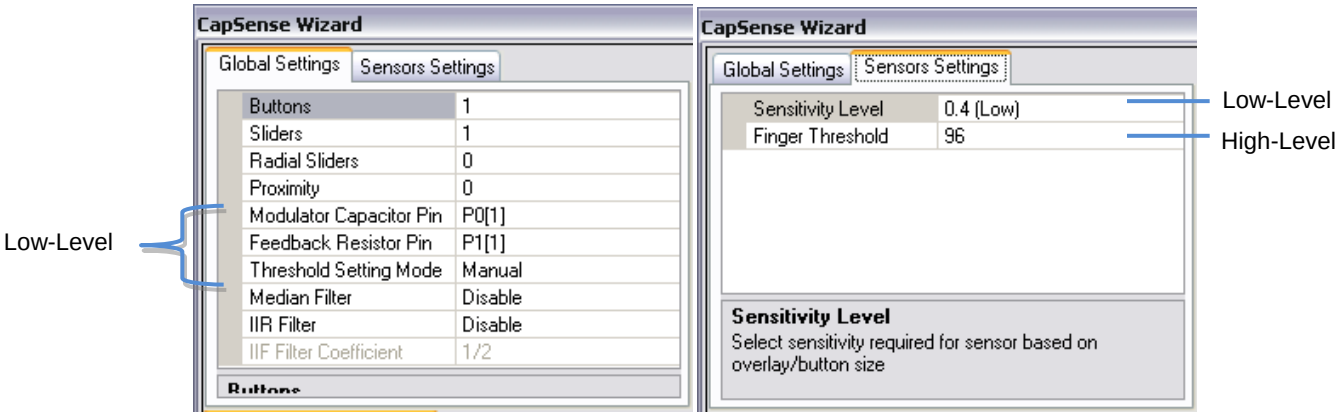
3.10.7 ADC Enabled

When this parameter is set to Enabled, the ADC routines are included in the compliable code and can be called from user code. When this parameter is set to Disabled, the ADC routines are not included. Setting this parameter to Disabled can be useful in saving the flash memory if the ADC is no longer required by your design. When this parameter is set to Disabled, the CSDADC User Module behaves the same as the CSD User Module.

3.11 SmartSense User Module Parameters

Figure 3-8. PSoC Designer SmartSense Parameters





3.12 Low-Level Parameters

3.12.1 Shield Electrode Out

A shield electrode is used to reduce parasitic capacitance. The shield electrode signal can be routed to one of the free digital row buses (Row_0_Output_1 – Row_0_Output_3). Each row output can then be routed to one of the allowed pins. Set the Row LUT Function to A.

3.12.2 Modulator Capacitor Pin

This parameter sets the pin to connect the external modulator capacitor (C_{MOD}). Choose from the available pins P0[1], P0[3].

3.12.3 Feedback Resistor Pin

This parameter sets the pin to connect the external feedback resistor (R_b). Choose from the available pins: P1[1], P1[5], P3[1]. Some pins are not available on some device packages. Use pins P1[5] or P3[1] to avoid programming problems.

3.12.4 Threshold Setting Mode

You can select between automatic and manual threshold settings using this parameter. The automatic threshold mode uses an algorithm that dynamically adjusts the finger threshold based on the noise seen in the system. In manual threshold mode, you can provide a fixed finger threshold value. The details on which circumstances to use manual threshold setting mode and how to determine the finger threshold value is provided in section Configuring SmartSense User Module.

3.12.5 Sensitivity Level

Sensitivity is used to increase or decrease the strength of a signal from a sensor. Lower value for sensitivity (0.1 pF) leads to a stronger signal from the sensor. Designs with thicker overlays require stronger signal from sensor for proper implementation. The available options for sensitivity selection are High (0.1 pF), Medium (0.2 pF), and Low (0.4 pF).

To produce a stronger signal from the sensor (High) sensitivity, SmartSense UM uses more time for sensor scanning. This means setting 0.1 (High) sensitivity for a sensor will consume more scan time compared to the sensor that has sensitivity level set to 0.2 pF (Medium).

3.12.6 Finger Threshold

This parameter is applicable only when threshold setting mode is set to Manual mode. It is recommended that you set this value to 80 percent of the sensor signal stored in the SmartSense_baSnSSignal[] array. This array can be easily monitored using I^2C or UART communication protocols.

4. CapSense Performance Tuning with User Modules

Optimal user module parameter settings depend on board layout, button dimensions, overlay material, and application requirements. These factors are discussed in Design Considerations. Tuning is the process of identifying the optimal parameter settings for robust and reliable sensor operation.

4.1 General Considerations

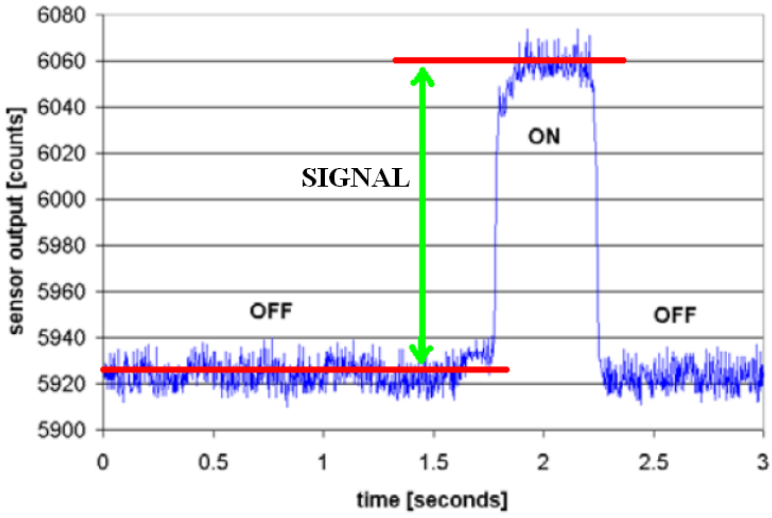
4.1.1 Signal, Noise, and SNR

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 signal and noise in the context of touch sensing.

4.1.1.1 CapSense Signal

The CapSense signal is the change in the sensor response when a finger is placed on the sensor, as demonstrated in Figure 4-1. 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 you place a finger on the sensor, the average output increases to 6060 counts. Because the CapSense signal tracks the change in counts due to the finger, Signal = 6060 – 5925 = 135 counts.

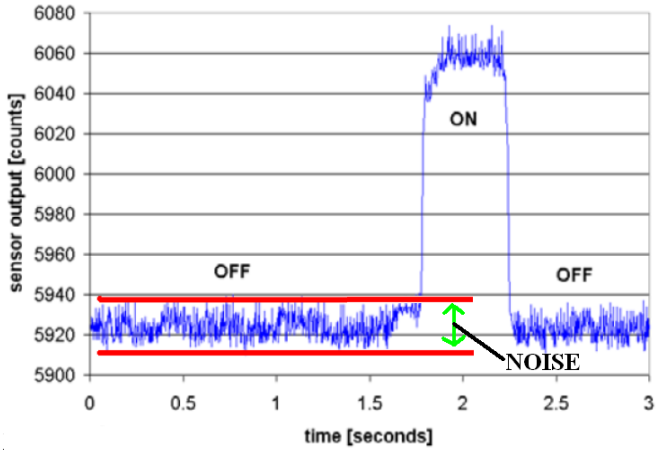
Figure 4-1. Example of CapSense Signal



4.1.1.2 CapSense Noise

CapSense noise is the peak-to-peak variation in sensor response when a finger is not present, as shown in Figure 4-2. In this example, the output waveform without a finger is bound by a minimum of 5912 counts and a maximum of 5938 counts. Because the noise is the difference between the minimum and maximum values of this waveform, Noise = 5938 – 5912 = 26 counts.

Figure 4-2. Example of CapSense Noise



4.1.1.3 CapSense SNR

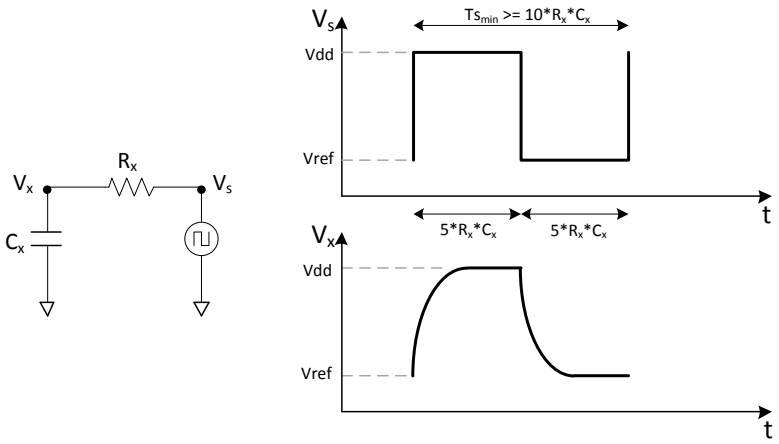
CapSense SNR is the simple ratio of signal and noise. Continuing with the example, if the signal is 135 counts and noise is 26 counts, and then SNR is 135:26, which reduces to an SNR of 5.2:1. The minimum recommended SNR for CapSense is 5:1, which means the signal is five times larger than the noise.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

4.1.2 Charge/Discharge Rate

To achieve maximum sensitivity in the tuning process, the sensor capacitor must be charged and discharged fully during each cycle. The charge/discharge path switches between two states at a rate set by the Clock parameter in the user module.

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 4-3.

Figure 4-3. Charge/Discharge Waveforms



Make sure to 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 period and maximum frequency are:

$$Ts_{min} = 10 \times R_X C_X$$ Equation 5

$$fs_{max} = \frac{1}{10 \times R_X C_X}$$ Equation 6

For example, assume the series resistor includes a 560-Ω external resistor and up to 800 Ω of internal resistance, and the sensor capacitance is typical:

R_X = 1.4 kΩ

C_X= 24 pF

The value of the time constant and maximum front-end switching frequency in this example is:

Ts_{min} = 0.34 µs

fs_{max} = 3 MHz

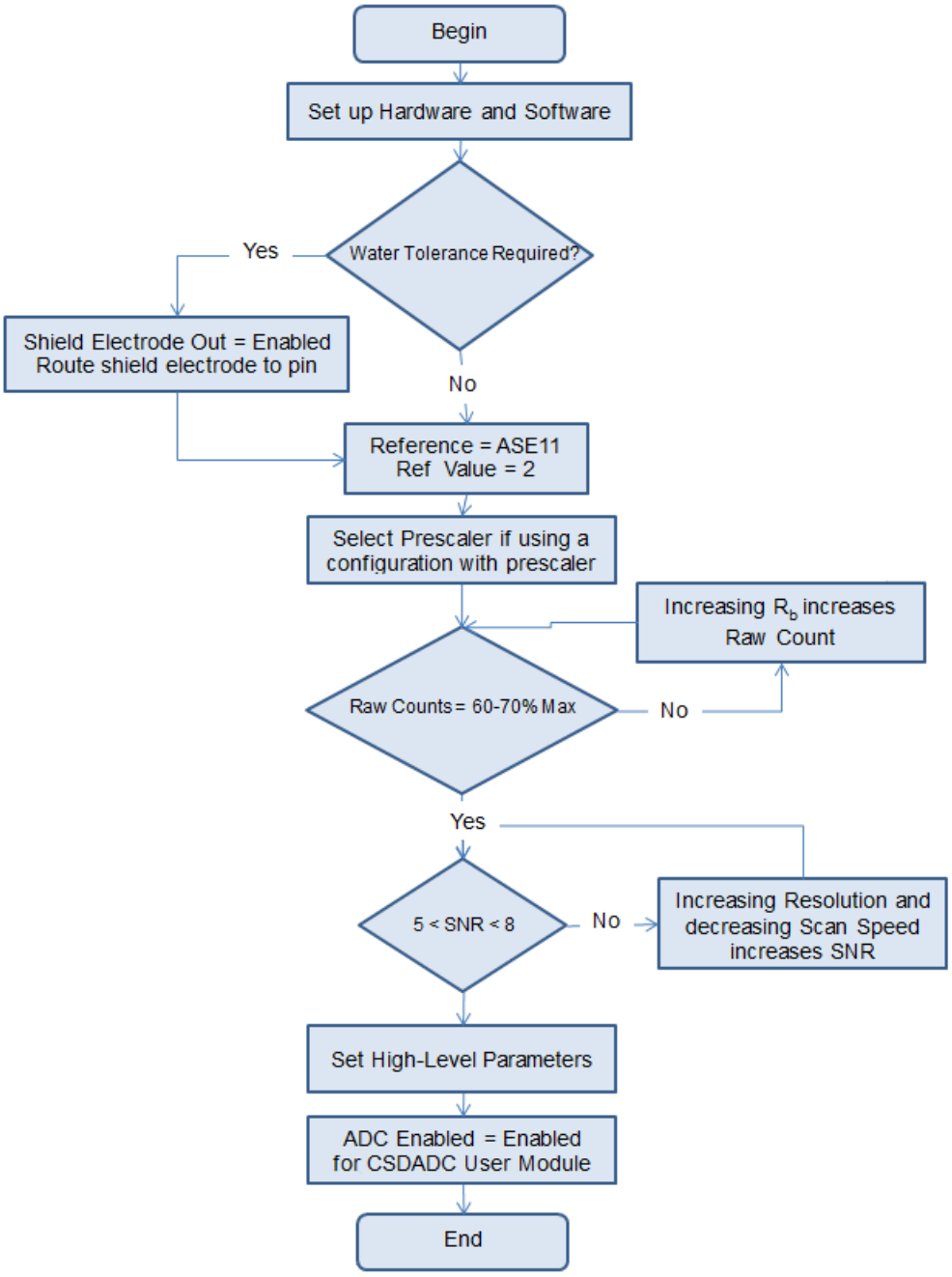
4.1.3 Importance of Baseline Update Threshold Verification

Temperature and humidity both cause the average number of counts to drift over time. The baseline is a reference count level for CapSense measurements that is an important part of compensating for environmental effects. High-level decisions, such as Finger Present and Finger Absent states, are based on the reference level established by the baseline. Because each sensor has unique parasitic capacitance associated with it, each capacitive sensor has its own baseline.

Baseline tracks the change in counts at a rate set by the Baseline Update Threshold parameter. Make sure to match the update rate to the intended application. If the update rate is too fast, the baseline will compensate for any changes introduced by a finger, and the moving finger will not be detected. If the update rate is too slow, relatively slow environmental changes may be mistaken for fingers. During development, you should verify the Baseline Update Threshold settings.

4.2 Tuning the CSD and CSDADC User Modules

Figure 4-4. Tuning CSD and CSDADC User Modules



4.2.1 Setup Hardware and Software for Tuning

Before starting the tuning process, certain hardware and software tools are required, as listed in the Overview. Tuning requires monitoring the raw counts, baseline, and difference counts of the sensors through a communication interface. A code example that uses an I^2C communication protocol to communicate with the PC can be downloaded here. After downloading the example:

1. Open the code example using PSoC Designer. In the Workspace Explorer, right-click **CSD User Module** and select **CSD Wizard**. You are shown four CapSense buttons assigned to the pins. Change the number of buttons and pin assignments according to your application requirements.
2. Change Modulator Capacitor Pin and Feedback Resistor Pin, if required.
3. Generate and build the project.
4. Program the CapSense controller with the hex file.
5. See "Instructions to use USB-I2C Tool to Monitoring CapSense Data" located in the code example's documentation for instructions on how to monitor raw count, baseline, and difference count.

4.2.2 Select Prescaler

To select the Prescaler value, use the following process:

1. Measure the maximum sensor parasitic capacitance using an LCR meter or the CapSense controller. If you are using an LCR meter, skip to Step 5.
2. Use the CSD User Module with clock prescaler and the following parameters:
   a. Scan Speed = Normal
   b. Resolution = 16
   c. Reference = VBG
   d. Prescaler period = 15 (SysClk = 24 MHz), 7 (SysClk = 12 MHz), or 3 (SysClk = 6 MHz)
   e. ShieldElectrodeOut = Enable or disable depending on the application
3. Build the project and download it to the CapSense controller.
4. Monitor only the raw counts from all the sensors (including slider segments) and note the highest raw count value.
5. Use Equation 7 to calculate maximum parasitic capacitance.

$$C_X = \frac{\text{Highest Raw Coun}}{88.85\text{e}12\times(V_{DD}-1.3)} \quad \text{Equation 7}$$

6. Use Equation 8 to calculate fs_{max}.

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

7. Calculate the prescaler value based on the clock configuration.
8. For CSD with the clock prescaler configuration:

$$\text{Prescaler} \geq \frac{\text{SysClk}}{\text{fs}_{\max}\times 2} - \quad \text{Equation 9}$$

9. For CSDADC with the PWM8 clock source configuration:

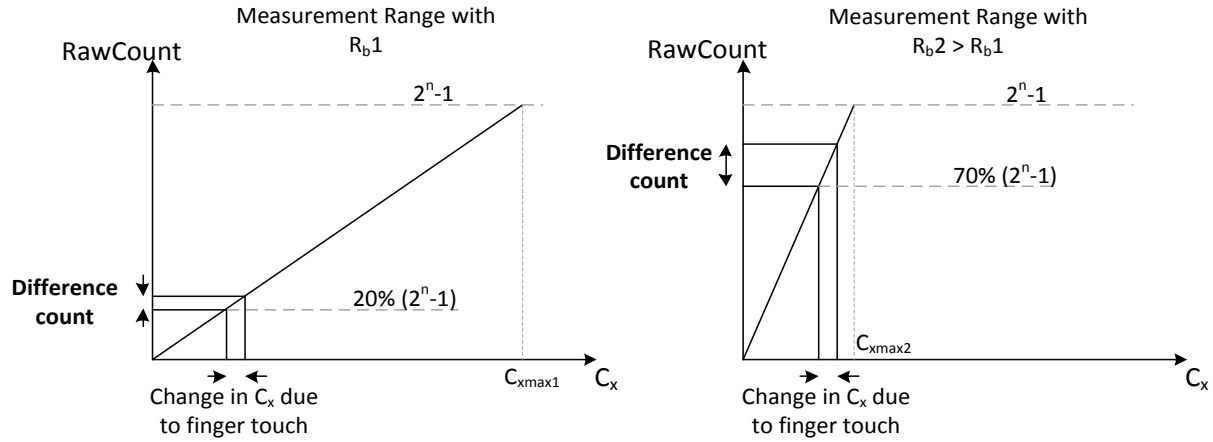
$$\text{Prescaler} \geq \frac{\text{SysClk}}{\text{fs}_{\max}} - \quad \text{Equation 10}$$

10. For better SNR performance, round the calculated prescaler to the nearest $2^n - 1$ value (1, 3, 7, 15, 31, and so on.)

4.2.3 Set Raw Count Range with R_b

The value of the bleed resistor, R_b, determines the sensitivity of the CapSense sensor. Sensitivity is the difference count measured between a touch and no touch condition. Increasing R_b improves both sensitivity and the raw count value in a no touch condition. Increasing R_b also decreases the maximum capacitance value that can be measured by the CapSense sensor.

Figure 4-5. CSD Measurement Range



Consider the two raw count measurement range graphs in Figure 4-5. In the left graph, $R_b = R_b1$, the system can measure capacitance up to C_{xmax1}. The raw count is 20 percent of its maximum value. The difference count produced due to the finger touch is very small, low sensitivity. In the right graph, the bleed resistor is increased, $R_b2 > R_b1$. The raw count is now 70 percent of its maximum. The difference count in this case is much larger than in the first case, which increases the sensitivity.

Typical values for R_b are between 500 Ω and 10 kΩ. To find the optimal resistance:

1. Start with a 2-kΩ resistor for R_b.
2. Monitor the raw count measurements for all sensors.
3. Adjust R_b until the raw counts are 70 percent of the full-scale range at the selected scanning resolution. For example, if 9-bit resolution is selected, the full-scale range is 511 counts. R_b should be adjusted until the raw data is 358 counts.

In designs with multiple sensors, it can be difficult to set all of the raw counts exactly to 70 percent. In this case, set all of the raw counts between 60 and 70 percent of the maximum.

4.2.4 Set High-Level Parameters

Follow the recommendations given in High-Level Parameter Recommendations.

4.3 Configuring SmartSense User Module

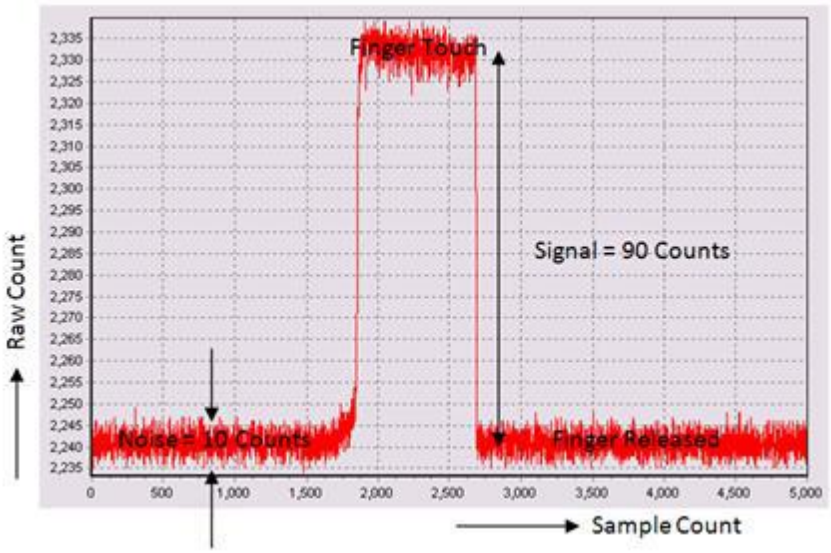
Here are the basic guidelines to configure the SmartSense User Module in a typical CapSense application:

1. Prepare the target board. Assemble the target application PCB and fix the overlay on it. Use nonconductive glue or special adhesive tape to stick the overlay. Avoid air gaps between the PCB and the overlay, as it can reduce the sensitivity substantially and cause multiple false button triggers because of the air gap shifting under the touch.
2. Set up a real-time data charting tool such as Multi-Chart (section CapSense Data Viewing Tools) to monitor one or more data series in real-time. The sensor rawcount (waSnsResult), baseline (waSnsBaseline), difference counts (waSnsDiff), signal (baSnsSignal), and button finger threshold (baBtnFThreshold) must be observed during the tuning process. You can use the UART-USB Bridge or the I2C-USB Bridge hardware to build an interface between the CapSense controller and PC via the USB port. Do not use the LCD or any other numerical displays to monitor data, because they are slow and do not allow visualizing the data dynamics.

   Note When using SmartSense and EzI2Cs, P1[1] is not available for R_b, as P1[1] is used for SCL.

3. The default configurations to start with are:
   - Set IMO = 24 MHz, CPU_Clock = SysClk/1
   - Place the SmartSense User Module configuration for IMO = 24 MHz
   - Disable **Sensor Autoreset**. Set **Debounce** to 3 and **Shield Electrode Out** to None.
   - Under Global Settings in the CapSense wizard, assign one sensor under the category "Button". Select the modulation Capacitor and Feedback resistor pin as well. Set the **Threshold Setting Mode** as Manual, and disable the Median and IIR filters. Ensure that in the board, the R_b value is 15 K and the C_{MOD} is 10 nF.
   - In the Sensor Settings tab in the CapSense wizard, set **Sensitivity Level** to Low and **Finger Threshold** to 96.
4. Generate/build the project with the sample firmware code provided in the user module datasheet.
5. Monitor the sensor raw count and calculate the SNR. Figure 4-6 shows the monitored sensor RawCount. According to CapSense best practices, the SNR for a robust design must be greater than 5. The monitored SNR in this case is 9. Read SNR improvement techniques in step 7 if the SNR is less than 5.

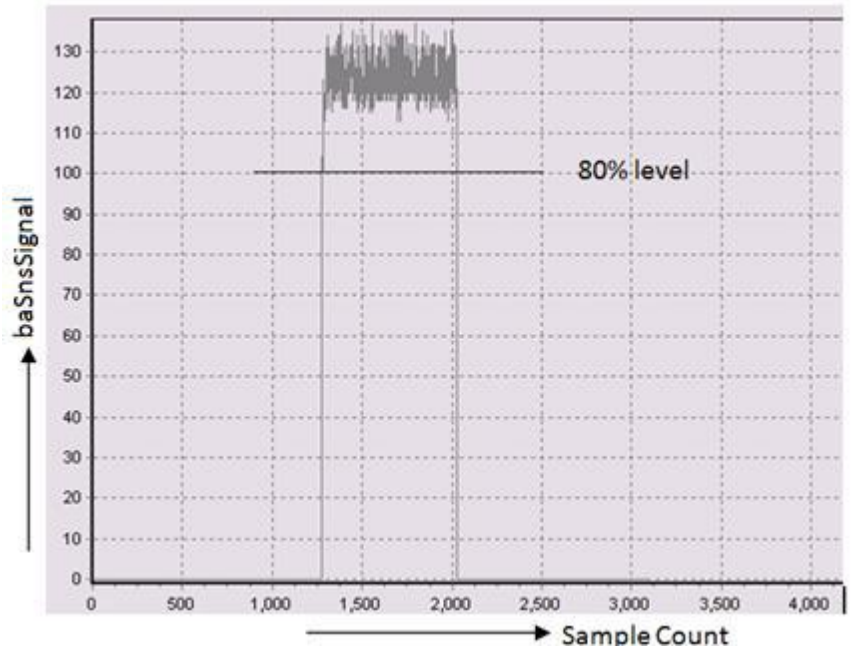
Figure 4-6. Monitored Sensor RawCount



6. You can skip this step if the application does not require precise controlling of finger threshold or if the automatic threshold setting mode is selected. The automatic threshold mode uses an algorithm that dynamically adjusts the finger threshold based on the noise seen in the system. You should choose the manual mode over the automatic threshold mode in the following scenarios:
   - If the RAM/ROM footprint is smaller than the number of sensors it must support.
   - Designs that require precisely adjusted sensor finger thresholds.
   - Designs that need to be operated under noisy conditions.

If the manual threshold setting mode is used, then to complete the tuning process you must set the finger threshold as well. Monitor the sensor signal baSnsSignal. Determine 80 percent of its maximum value and this value is the finger threshold. In the example shown in Figure 4-7, it is nearly 100 counts.

Figure 4-7. Monitoring the Sensor Signal baSnsSignal



Now open the CapSense wizard, and under the Sensor Settings tab, set the finger threshold to the calculated value (it is 100 for the example shown in Figure 4-7).

7. The SmartSense sensor tuning process is completed with the previous step. However, there can be scenarios where the SNR achieved in Step 5 is below 5:1, for example, when using a thicker overlay. In such cases, follow these recommendations:

   - Change the sensitivity level from ‘Low’ to ‘Medium’. If the SNR of 5:1 is still not achieved, change the level to ‘High’.
   - Sometimes, the requirement of 5:1 cannot be achieved due to excess noise observed in the system. Under these circumstances, software filters should be used. Start with an IIR filter of coefficient 1/2. If this does not reduce the noise, change the IIR coefficient to 1/4.

5. Design Considerations

When designing capacitive touch sense technology into your application, it is important to remember 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. For more in-depth information, see Getting Started with CapSense.

5.1 Overlay Selection

In CapSense Fundamentals, Equation 1 is presented for finger capacitance as follows:

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

Where:

ε_0 = Free space permittivity

ε_r = Dielectric constant of overlay

A = Area of finger and sensor pad overlap

D = Overlay thickness

To increase the CapSense signal strength, choose an overlay material with higher dielectric constant, decrease the overlay thickness, and increase the button diameter.

Table 5-1. Overlay Material Dielectric Strength

Material	Breakdown Voltage (V/mm)	Min. Overlay Thickness at 12 kV (mm)
Air	1200–2800	10
Wood – dry	3900	3
Glass – common	7900	1.5
Glass – Borosilicate (Pyrex®)	13,000	0.9
PMMA Plastic (Plexiglas®)	13,000	0.9
ABS	16,000	0.8
Polycarbonate (Lexan®)	16,000	0.8
Formica	18,000	0.7
FR-4	28,000	0.4
PET Film (Mylar®)	280,000	0.04
Polymide film (Kapton®)	290,000	0.04

Conductive material cannot be used as an overlay because it interferes with the electric field pattern. For this reason, do not use paints containing metal particles in the overlay.

An adhesive is used to bond the overlay to the CapSense PCB. A transparent acrylic adhesive film from 3M™ called 200MP is qualified for use in CapSense applications. This special adhesive is dispensed from paper-backed tape rolls (3M™ product numbers 467MP and 468MP).

5.2 ESD Protection

Robust ESD tolerance is a natural byproduct of careful system design. By considering how contact discharge will occur in your product, particularly in your user interface, it is possible to withstand an 18-kV discharge event without incurring any damage to the CapSense controller.

CapSense controller pins can withstand a direct 2-kV event. In most cases, the overlay material provides sufficient ESD protection for the controller pins. Table 5-1 lists the thickness of various overlay materials required to protect the CapSense sensors from a 12-kV discharge, as specified in IEC 61000-4-2. If the overlay material does not provide sufficient protection, apply ESD countermeasures in the following order: Prevent, Redirect, Clamp.

5.2.1 Prevent

Make sure that all paths on the touch surface have a breakdown voltage greater than potential high-voltage contacts. Also, design your system to maintain an appropriate distance between the CapSense controller and possible sources of ESD. If it is not possible to maintain adequate distance, place a protective layer of a high breakdown voltage material between the ESD source and CapSense controller. One layer of 5 mil-thick Kapton® tape will withstand 18 kV.

5.2.2 Redirect

If your product is densely packed, it may not be possible to prevent the discharge event. In this case, you can protect the CapSense controller by controlling where the discharge occurs. A standard practice is to place a guard ring on the perimeter of the circuit board that is connected to the chassis ground. As recommended in PCB Layout Guidelines, providing a hatched ground plane around the button or slider sensor can redirect the ESD event away from the sensor and CapSense controller.

5.2.3 Clamp

Because CapSense sensors are purposely placed in close proximity to the touch surface, it may not be practical to redirect the discharge path. In this case, including series resistors or special-purpose ESD protection devices may be appropriate.

The recommended series resistance value is 560 Ω.

A more effective method is to provide special-purpose ESD protection devices on the vulnerable traces. ESD protection devices for CapSense need to be low capacitance. Table 5-2 lists devices recommended for use with CapSense controllers.

Table 5-2. Low-Capacitance ESD Protection Devices Recommended for CapSense

ESD Protection device		Input Capacitance	Leakage Current	Contact Discharge Maximum Limit	Air Discharge Maximum Limit
Manufacturer	Part Number				
Littelfuse	SP723	5 pF	2 nA	8 kV	15 kV
Vishay	VBUS05L1-DD1	0.3 pF	0.1 µA <	±15 kV	±16 kV
NXP	NUP1301	0.75 pF	30 nA	8 kV	15 kV

5.3 Electromagnetic Compatibility (EMC) Considerations

5.3.1 Radiated Interference

Radiated electrical energy can influence system measurements and potentially influence the operation of the processor core. Interference may enter the CapSense Controller at the PCB level, through CapSense sensor traces, and through any other digital or analog inputs. Layout guidelines for minimizing the effects of RF interference include:

- **Ground Plane**: Provides a ground plane on the PCB.
- **Series Resistor**: Place series resistors within 10 mm of the CapSense controller pins.
  - The recommended series resistance for CapSense input lines is 560 Ω.
  - The recommended series resistance for communication lines, such as I^2C, and SPI is 330 Ω.

- **Trace Length**: Minimize trace length whenever possible.
- **Current Loop Area**: Minimize the return path for current. Provide hatched ground instead of solid fill within 1 cm of the sensors and traces to reduce the impact of parasitic capacitance.
- **RF Source Location**: Partition systems with noise sources such as LCD inverters and switched-mode power supplies (SMPS) to keep them separated from CapSense inputs. Shielding the power supply is another common technique for preventing interference.

5.3.2 Radiated Emissions

To reduce radiated emissions from the CapSense sensor, select a low frequency for the switched capacitor clock. This clock is controlled in firmware using the Prescaler option. Increasing the Prescaler value decreases the frequency of the switching clock

5.3.3 Conducted Immunity and Emissions

Noise entering a system through interconnections with other systems is referred to as conducted noise. These interconnections include power and communication lines. Because CapSense controllers are low-power devices, conducted emissions must be avoided. The following guidelines will help reduce conducted emission and immunity:

- Use decoupling capacitors as recommended by the datasheet.
- Add a bidirectional filter on the input to the system power supply. This is effective for both conducted emissions and immunity. A pi-filter can prevent power-supply noise from effecting sensitive parts, while also preventing the switching noise of the part itself from coupling back onto the power planes.
- If the CapSense controller PCB is connected to the power supply by a cable, minimize the cable length and consider using a shielded cable.
- You can place a ferrite bead around the power supply or communication lines to filter out high frequency noise.

5.4 Software Filtering

Software filters are one of the techniques for dealing with high levels of system noise. Table 5-3 lists the types of filters that are useful for CapSense.

Table 5-3. Table of CapSense Filters

Type	Description	Application
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 filter	High-frequency white noise (1/f noise)
Median	Nonlinear filter that computes median input value from a buffer of size N	Noise spikes from motors and switching power supplies
Jitter	Nonlinear filter that limits current input based on previous input	Noise from thick overlay (SNR < 5:1), especially useful for slider centroid data
Event-Based	Nonlinear filter that causes a predefined response to a pattern observed in the sensor data	Commonly used during nontouch events to block CapSense data transmission
Rule-Based	Nonlinear filter that causes a predefined response to a pattern observed in the sensor data	Commonly used during normal operation of the touch surface to respond to special scenarios such as accidental multibutton selection

A code example that uses all these filters can be downloaded here.

Table 5-4 shows the RAM and flash requirements for different software filters. The amount of flash required for each filter type depends on the performance of the compiler. The requirements listed here are for both the ImageCraft compiler and the ImageCraft Pro compiler

Table 5-4. RAM and Flash Requirements

Filter Type	Filter Order	RAM (Bytes per sensor)	Flash (Bytes) ImageCraft Compiler	Flash (Bytes) ImageCraft Pro Compiler
Average	2-8	6	675	665
IIR	1	2	429	412
	2	6	767	622
Median	3	6	516	450
	5	10	516	450
Jitter filter on raw counts	N/A	2	277	250
Jitter filter on slider centroid	N/A	2	131	109

A code example that uses all these filters can be downloaded here.

5.5 Power Consumption

5.5.1 System Design Recommendations

For many designs, minimizing power consumption is an important goal. There are several ways to reduce the power consumption of your CapSense capacitive touch-sensing system.

- Set GPIO drive mode for low power
- Turn off high power blocks
- Optimize CPU speed for low power
- Operate at a lower V_{DD}

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

5.5.2 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.

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

$$I_{AVE} = \frac{(I_{Act} \times t_{Act}) + (I_{Slp} \times t_{Slp})}{T}$$ Equation 11

Where:

I_{ACT} = active current

T_{ACT} = active time

I_{Slp} = sleep current

t_{Slp} = seep time

T = total time period

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

$$P_{AVE} = V_{DD} \times I_{AVE}$$ Equation 12

Where:

V_{DD} = supply voltage

I_{AVE} = average current

An Excel-based average power consumption calculator is available. Click here to download the calculator.

5.5.3 Response Time versus Power Consumption

As illustrated in Equation 12, the average power consumption can be reduced by decreasing I_{AVE} or V_{DD}. I_{AVE} may be decreased by increasing sleep time. Increasing sleep time to a very high value will lead to poor response time of the CapSense button. As a result, the sleep time must be based on system requirements.

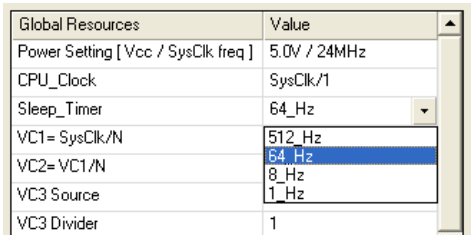
In any application, if both power consumption and response time are important parameters to be considered, then you can use an optimized method that incorporates both continuous-scan and sleep-scan modes. In this method, the device spends most of its time in sleep-scan mode, where it scans the sensors and goes to sleep periodically, as explained in the previous section, thereby consuming less power. When a user touches a sensor to operate the system, the device jumps to continuous-scan mode, where the sensors are scanned continuously without invoking sleep, giving good response time. The device remains in continuous-scan mode for a specified timeout period. If the user does not operate any sensor within this timeout period, the device jumps back to the sleep-scan mode.

5.5.4 Measuring Average Power Consumption

The following instructions describe how to determine average power consumption when using the sleep-scan method:

1. Build a project that scans all of the sensors without going to sleep (continuous-scan mode). Include a pin-toggle feature in the code before scanning the sensors. Toggling the state of the output pin serves as a time marker that can be tracked with an oscilloscope.
2. Download the project to the CapSense device and measure the current consumption. Assign the measured current to I_{ACT}.
3. Get the sleep current information from the datasheet and assign it to I_{Slp}.
4. Monitor the toggling output pin with an oscilloscope and measure the period between two toggles. This gives the active time. Assign this value to t_{Act}.
5. Apply sleep-scan to the project. The period of the sleep-scan cycle, T, is set by selecting the sleep timer frequency in the global resources window, as shown in Figure 5-1.
6. Subtract active time from the sleep-scan cycle time to get the sleep time. $t_{Slp} = T - t_{Act}$.
7. Calculate the average current using Equation 11.
8. Calculate average power consumption using Equation 12.

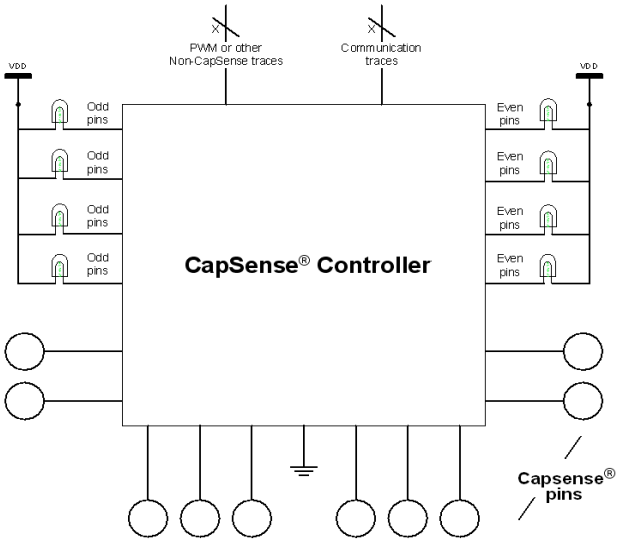
Figure 5-1. Global Resources Window



5.6 Pin Assignments

An effective method to reduce interaction between CapSense sensor traces and communication and non-CapSense traces is to isolate each by port assignment. Figure 5-2 shows a basic version of this isolation for a 32-pin QFN package. Because each function is isolated, the CapSense controller is oriented such that there is no crossing of communication, LED, and sensing traces.

Figure 5-2. Recommended: Port Isolation for Communication, CapSense, and LEDs



The CapSense controller architecture requires a current budget for even and odd port pin numbers. If the current budget for odd port pin numbers is 100 mA, the total current drawn though all odd port pins should not exceed 100 mA. In addition to the current budget limitation, there is also maximum current limitation for each port pin defined in the CapSense controller datasheet.

All CapSense controllers provide high current sink and source capable port pins. When using high current sink or source from port pins, you should use the ports that are closest to device ground pin to minimize the noise.

5.7 PCB Layout Guidelines

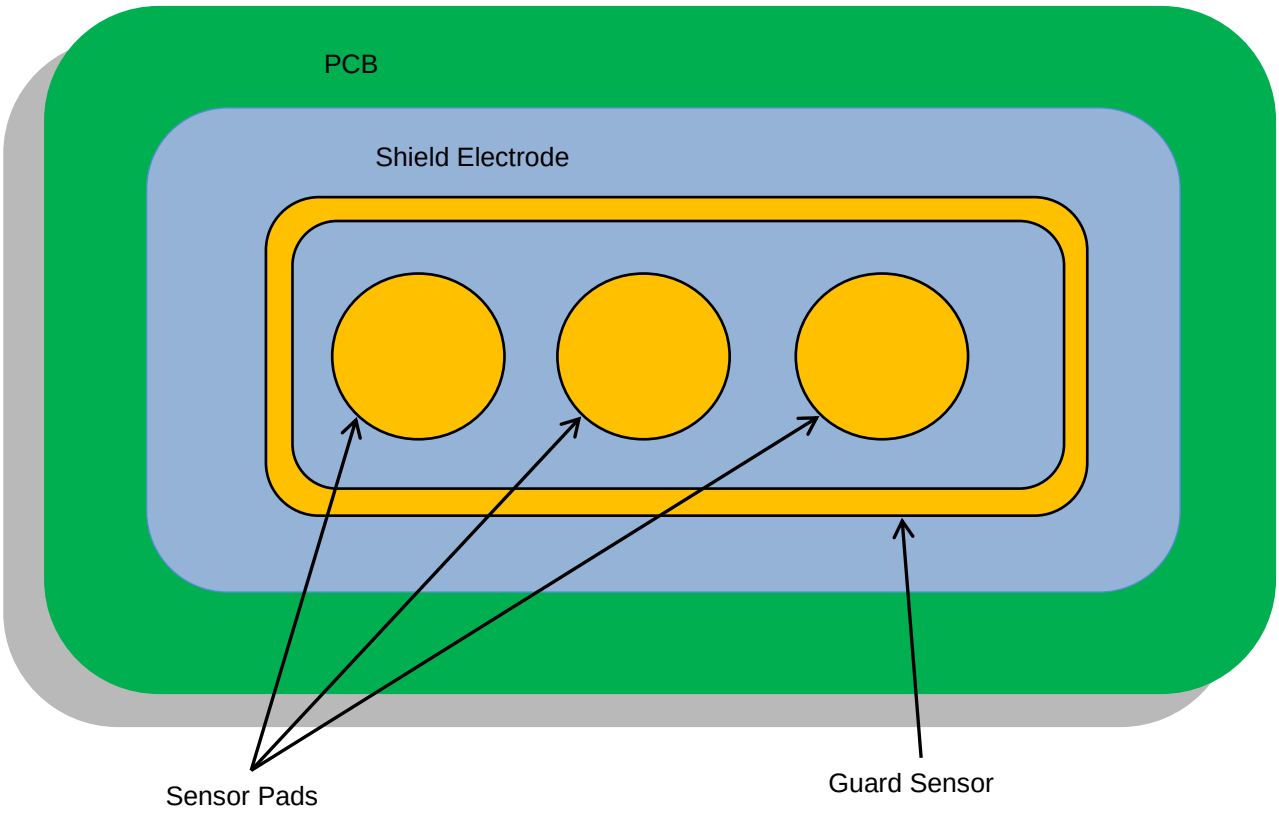
Detailed PCB layout guidelines are available in Getting Started with CapSense.

6. Water Tolerance

Some CapSense capacitive touch-sensing applications 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.

6.1 Shield Electrode and Guard Sensor

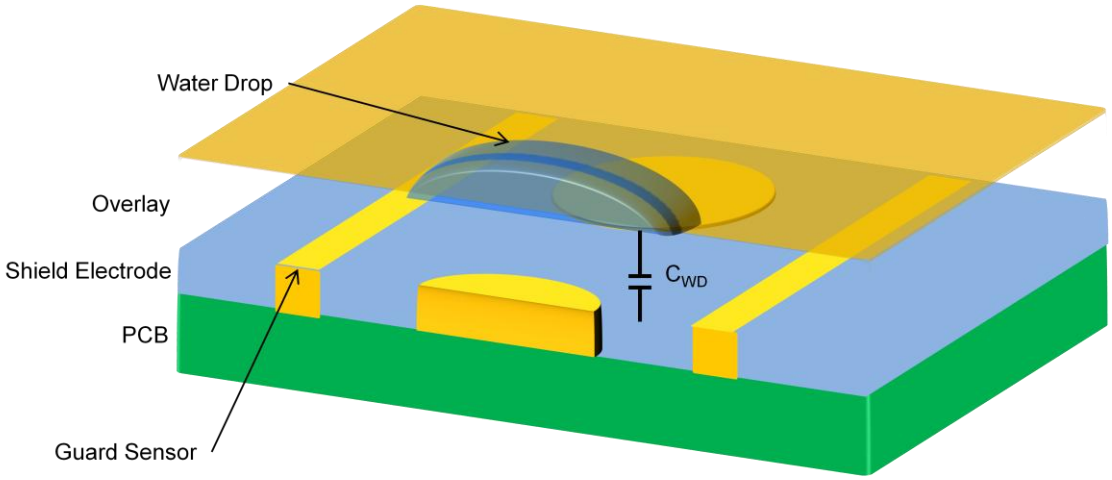
Figure 6-1. PCB Layout with Shield Electrode and Guard Sensor



6.1.1 Shield

Shield electrodes protect CapSense button sensors from detecting false touches caused by water drops. When water drops are present on the overlay surface, the coupling between the shield electrode and sensor pad is increased by C_{WD}, as shown in Figure 6-2.

Figure 6-2. Capacitance Measurement with Water Drop



- C_{WD} – Capacitance between the water drop and shield electrode

The purpose of the shield electrode is to set up an electric field around the touch sensors that helps attenuate the effects of water. The shield electrode works by mirroring the voltage of the touch sensor on the shield.

Follow these guidelines to ensure proper shield operation:

- Schematic
- Layout
- Firmware development

6.1.1.1 Schematic

Select the proper pin to drive the shield electrode out signal. Do not use the following pins to drive the shield electrode out signal.

- Modulator capacitor pin (C_{MOD}): P0[1] or P0[3]
- Feedback resistor pin (R_B): P1[1] or P1[5]
- Other port pins: P0[0], P0[4], P1[0], P1[4], P2[0], P2[4], and P3[0]

6.1.1.2 Layout

Follow the layout guidelines given in Getting Started with CapSense.

6.1.1.3 Firmware Development

Step 1: Select the shield electrode signal source from one of the free digital row buses (Row_0_Output_1 to Row_0_Output_3):

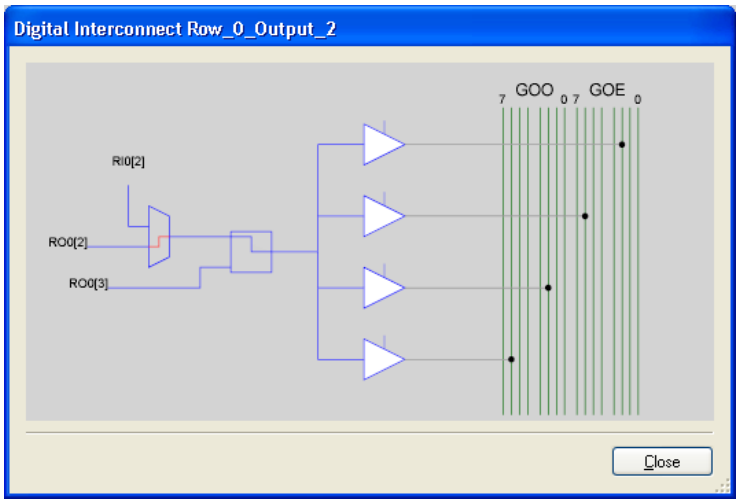
- Select Row_0_Output_1 if shield pin is P0[5], P1[5], P2[1], P2[5]
- Select Row_0_Output_2 if shield pin is P0[2], P0[6],P1[2], P1[6], P2[2], P2[6], P3[2]
- Select Row_0_Output_3 if shield pin is P0[7],P1[3], P1[7], P2[3], P2[7]

Step 2: Route the shield electrode out to the shield pin. Follow these steps:

1. Select the shield electrode out.
2. Route the row output signal to Global out Odd/Even.
3. Connect Global out Odd/Even to the Port pin. For example, if P0[2] is selected as the shield pin: from the user module properties, select the shield electrode out on Select Row_0_Output_2.

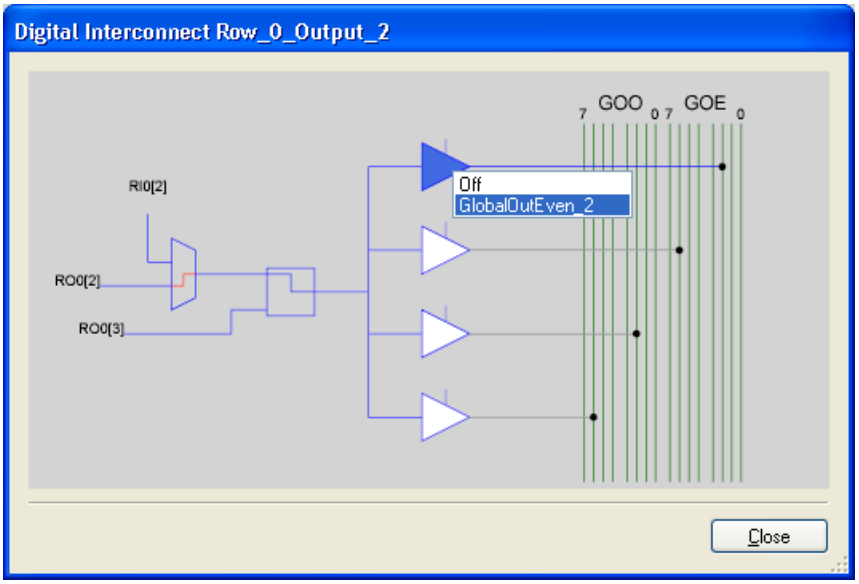
4. Click on **Row_0_Output_2** and open the digital interconnect view, as shown in Figure 6-3.

Figure 6-3. Schematic View of Digital Interconnect



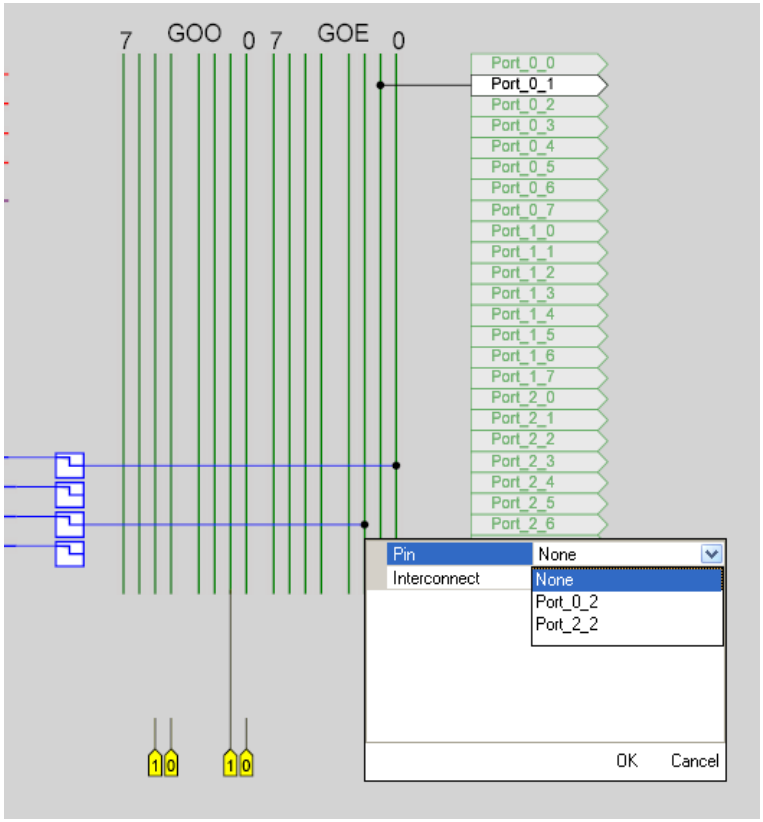
5. Select **Row_0_Output_2_Drive_0** and select **GlobalOutEven_2** as shown in Figure 6-4.

Figure 6-4. Schematic View of Output Select



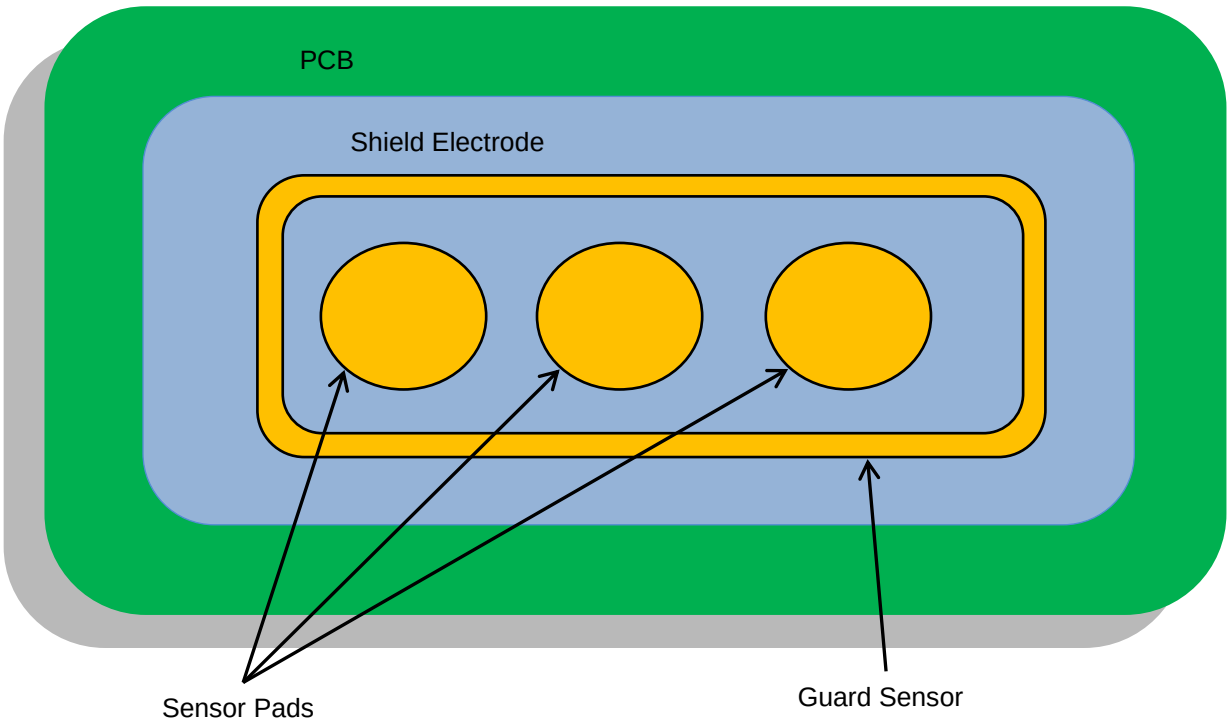
6. Click on **GlobalOutEven_2** and select **P0[2]** in the Pin option, as shown in the following figure.

Figure 6-5. PCB with Shield and Guard Sensor



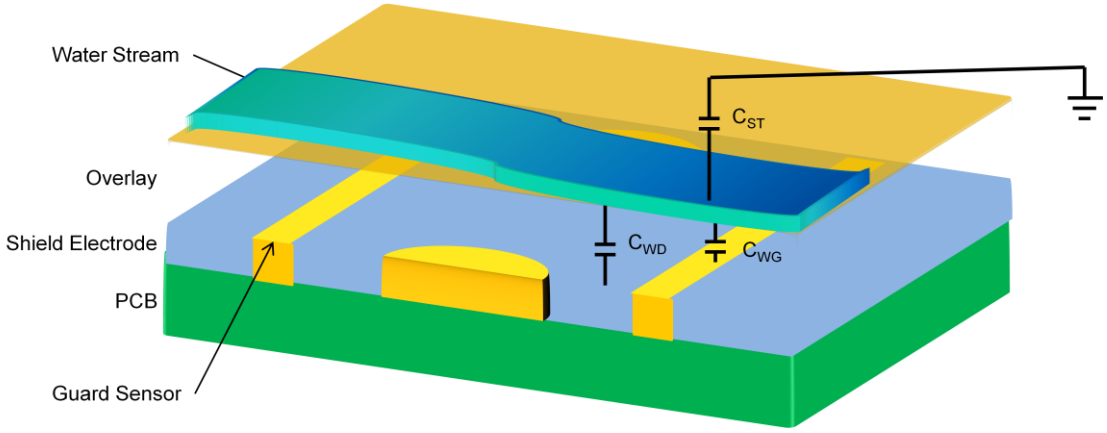
6.1.2 Guard Sensor

Figure 6-6. PCB with Shield and Guard Sensor



A guard sensor is a copper trace, as shown in Figure 6-6, that surrounds all the sensors on the PCB, which is used to detect the presence of a continuous water stream. When a water stream is present on the sensing surface, a large-capacitance C_{ST} is added to the system, as shown in Figure 6-7. This capacitance may be several times larger than C_{WD}. Because of this, the effect of the shield electrode is completely masked and the raw counts measured by the sensor will be the same as or even higher than a finger touch. In this situation, a guard sensor will help; when it detects a water stream, it will block the other sensors from triggering.

Figure 6-7. Capacitance Measurement with Water Stream



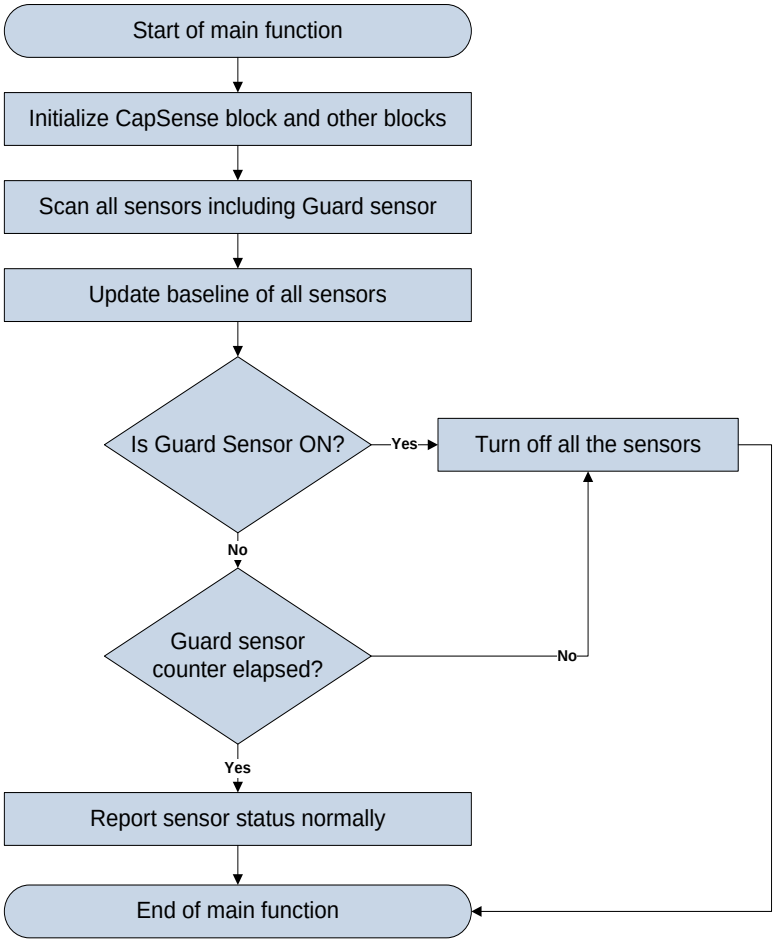
- C_{WD} - Capacitance between the water stream and shield electrode.
- C_{ST} - Capacitance between the water stream and the system ground.
- C_{WG} - Capacitance between the water stream and the guard sensor

The guard sensor should be implemented in firmware. One CapSense pin and a counter (hardware/software) are required to implement the guard sensor.

When a water stream is applied to the board, the guard sensor detects this event and disables the touch sensor processing logic. Additional guard sensor "dead" time prevents unlocking the sensor prematurely. When the water stream is gone, the guard counter suppresses touch processing for a short time (guard sensor counter). This eliminates false touch detection from any water that remains on the board.

Figure 6-8 is a flow chart that shows how to implement the guard sensor in firmware.

Figure 6-8. Flow Chart to Implement the Guard Sensor



6.2 Design Recommendations

The following system-level and PCB layout recommendations apply to CapSense systems that are exposed to water.

- Shield-electrode copper-hatch recommendations:
  - Top layer – 7-mil trace and 45-mil grid (15-percent fill)
  - Bottom layer – 7-mil trace and 70-mil grid (10-percent fill)
  - Shield electrode between buttons should be at least 10 mm wide
  - Only areas surrounding sensor pads and the CapSense controller should be grounded
- Place the sensor surface vertically or at an angle to the horizontal so that water drops naturally and move off the sensors and large water drops do not accumulate.
- Use water-repellent and nonabsorbent overlay material. This minimizes water streaks and films on the device panel. This is especially important if the water is highly conductive, such as seawater.
- Use a guard sensor in situations where the application may be subject to continuous water streams. A guard sensor is not required if the device will be subjected only to rain.

7. Proximity Sensing

Proximity sensors detect the presence of a hand or other conductive object before it makes contact with the capacitive touch surface. Imagine a hand stretched out to operate a car audio system in the dark. Proximity detection enables the system to light up with the approach of a finger. For example, the buttons of an audio system will glow with its backlight LEDs when the user's hand is near.

7.1 Types of Proximity Sensors

7.1.1 Button

A button with large C_P and small difference counts can work as a proximity sensor. The sensitivity of a proximity sensor implemented as a button is much higher than a regular capacitive touch-sensing button.

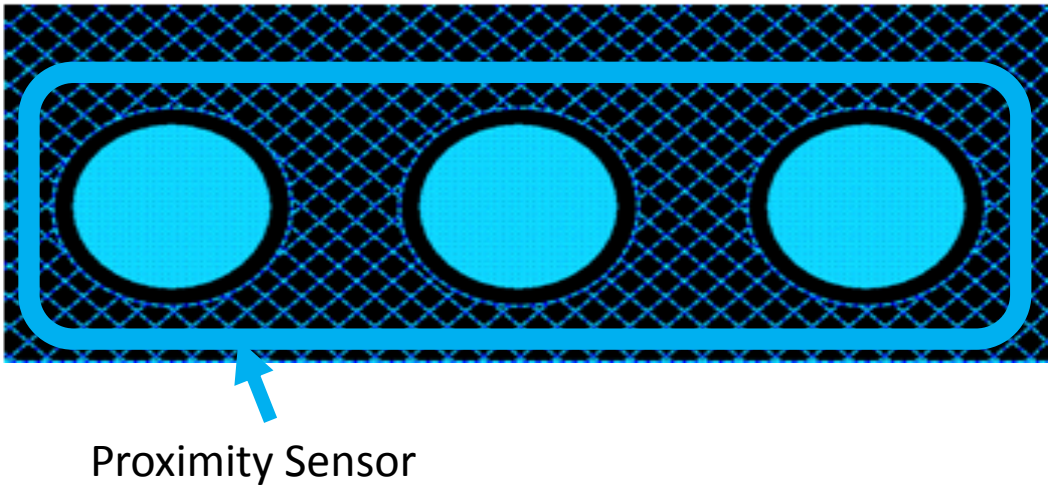
7.1.2 Wire

A single length of wire works well as a proximity sensor. Because detecting a hand relies on the capacitance change from electric field changes, any stray capacitance or objects affecting the electrical field around the wire will affect the range of the proximity sensor. Using a wire sensor is not an optimal solution for mass production because of manufacturing cost and complexity.

7.1.3 PCB Trace

A long PCB trace can form a proximity sensor. The trace can be a straight line or it can surround the perimeter of a system's user interface, as shown in Figure 7-1. This method is appropriate for mass production, but it is not as sensitive as a wire sensor.

Figure 7-1. Proximity Sensor Using PCB Trace



7.1.4 Sensor Ganging

Another way to implement a proximity sensor is to gang sensors together. This is accomplished by combining multiple sensors into one large sensor, using firmware to connect the sensors from the internal analog multiplexer bus in PSoC Designer. Be careful not to exceed the C_P limit of your design when using this method.

7.2 Design Recommendations

Using a shield electrode effectively extends the detection distance of the proximity sensor. This is particularly helpful when the proximity sensor must operate in the presence of metal.

A wire sensor increases the beneficial effect of the shield electrode because it can be located farther from the shield electrode.

Avoid large, solid ground-fill areas inside the proximity sensor, because this decreases the sensitivity.

Slower scan speeds provide increased sensitivity at all distances. For a proximity sensor, the scanning speed should be very slow.

8. Low Power Design Considerations

Power consumption is an important aspect of microcontroller designs. Among the several techniques to reduce the average current used by the CapSense controller, sleep mode is the most popular. The CapSense controller uses sleep mode when it is not required to perform any function, similar to a cell phone backlight dimming after an idle period. This is done to reduce the average current consumed by the device, a necessity of all battery applications. The CapSense controller enters sleep mode by writing a '1' to the SLEEP bit within the CPU_SCR0 register (Bit 3). This is accomplished by calling the M8C_Sleep macro. While in sleep mode, the central CPU is stopped, the internal main oscillator (IMO) is disabled, the Bandgap Voltage reference is powered down, and the Flash Memory Model is disabled. The only circuits left in operation are supply voltage monitor and 32-kHz internal oscillator. Power saving techniques other than sleep mode include:

- Disable CapSense (PSoC) analog block references
- Disable continuous time (CT) and switch capacitor (SC) blocks
- Disable CapSense (PSoC) analog output buffers
- Set drive modes to analog HI-Z

Sleep mode has negative effects for a design. If not used carefully, it can cause unpredictable operation. The PSoC must be awakened from sleep correctly and the user must be aware that the device is sleeping to allow extra processing.

8.1 Additional Power Saving Techniques

All the power saving techniques, with the exception of sleep mode, is application based and some of these produce undesirable results. Each technique is discussed in detail here.

```
ABF_CR0 &= 0xc3; // Buffer Off
```

8.1.1 Set Drive Modes to Analog HI-Z

The state of the CapSense controller drive modes can affect power consumption. You can change the drive modes only on pins that do not cause adverse affects to the system. The change must occur in a sequence that does not produce line glitches. This sequence depends on the current drive mode of the pin and the state of port data register. With the CapSense controller drive mode structure, the pin must temporarily be in either Resistive Pull-up or Resistive Pull-down drive mode when switching between HI-Z or Strong drive modes. The temporary drive mode is the opposite of the previous value on the pin. So, if the pin is driven high, then the temporary drive mode must be Resistive Pull-down. This ensures that the drive mode of the pin is not resistive, which eliminates any possible glitch.

The drive modes are set manually in software, before going to sleep. There are three registers, PRTxDM0, PRTxDM1, and PRTxDM2, which control the drive modes. One bit per register is assigned to a pin. So, to change the drive mode of a single pin, three register writes are needed. However, this is convenient because an entire port is changed by the same three register writes. The correct pit pattern for Analog HI-Z is 110b. Use the following code to set port zero to Analog HI-Z, from Strong, by first going to Resistive Pull-down.

```
PRT0DM0 = 0x00; // low bits
PRT0DM1 = 0xff; // med bits
PRT0DM2 = 0xff; //high bits
```

8.1.2 Putting it All Together

The following code is a sample of a typical sleep preparation sequence for a 28-pin part. In this sequence, interrupts are disabled, the analog circuitry is turned off, all drive modes are set to Analog HI-Z, and interrupts are re-enabled.

```
void PSoC_Sleep(void){
    M8C_DisableGInt;
    ARF_CR &= 0xf8; // analog blocks Off
    ABF_CR0 &= 0xc3; // analog buffer off
    PRT0DM0 = 0x00; // port 0 drives
    PRT0DM1 = 0xff;
    PRT0DM2 = 0xff;
    PRT1DM0 = 0x00; // port 1 drives
    PRT1DM1 = 0xff;
    PRT1DM2 = 0xff;
    PRT2DM0 = 0x00; // port 2 drives
    PRT2DM1 = 0xff;
    PRT2DM2 = 0xff;
    M8C_EnableGInt;
    M8C_Sleep;
}
```

8.1.3 Sleep Mode Complications

The CapSense controller can exit sleep either from a reset or through an interrupt. There are three types of resets within the CapSense controller: External Reset, Watchdog Reset, and Power-On Reset. Any of these resets takes the CapSense controller out of sleep mode; when the reset deasserts, the CapSense controller begins executing code starting at *Boot.asm*. Available interrupts to wake the CapSense controller are Sleep Timer, Low-Voltage Monitor, GPIO, Analog Column, and Asynchronous. Sleep mode complications arise when using interrupts to wake the CapSense controller or attempting digital communication while asleep. These considerations are discussed in detail in the following sections.

8.1.4 Pending Interrupts

If an interrupt is pending, enabled, and scheduled to take after a write to the SLEEP bit in the CPU_SCR0 register, the system will not go to sleep. The instruction still executes, but the CapSense controller does not set the SLEEP bit. Instead, the interrupt is serviced, which effectively causes the CapSense controller to ignore the sleep instruction. To avoid this, interrupts should be globally disabled while sleep preparation occurs and then re-enabled just before writing the SLEEP bit.

8.1.5 Global Interrupt Enable

The Global Interrupt Enable register (CPU_F) need not be enabled to wake the CapSense controller from interrupts. The only requirement to wake up from sleep by an interrupt is to use the correct interrupt mask within the INT_MSKx registers, as in the example below. If global interrupts are disabled, the ISR that wakes the CapSense controller is not executed but the CapSense controller still exits sleep mode.

In this case, you must manually clear the pending interrupt or enable global interrupts to allow the ISR to be serviced. Interrupts are cleared within the INT_CLRx registers.

```
//Set Mask for GPIO Interrupts M8C_EnableIntMask(INT_MSK0, INT_MSK0_GPIO)

// Clear Pending GPIO Interrupt
INT_CLR0 &= 0x20;
```

8.2 Post Wakeup Execution Sequence

If the CapSense controller is awakened through a reset, then execution starts at the beginning of the boot code. If the CapSense controller is awakened by an interrupt service routine, the first instruction to execute is the one immediately following the sleep instruction. This is because the instruction immediately following the sleep instruction is pre-fetched before the CapSense controller is asleep. Therefore, if global interrupts are disabled, the instruction execution will continue where it left off before sleep is initiated.

8.2.1 PLL Mode Enabled

If PLL mode is enabled, the CPU frequency must be reduced to the minimum of 3 MHz before going to sleep. This is because the PLL always overshoots as it attempts to relock after the CapSense controller wakes up and is re-enabled. Additionally, you should wait 10 ms after wakeup before normal CPU operation begins to ensure proper execution. This implies that, to use sleep mode and the PLL, the software must be able to execute at 3 MHz. A simple write to the OSC_CR0 register can reduce CPU speed. However, this register just sets a divider of SYSCLK, which means that the CPU speed will vary between part families with different SYSCLKs. Typically, SYSCLK is 24 MHz.

```
OSC_CR0 &= 0xf8; // CPU = 3 IMO = 24
```

8.2.2 Execution of Global Interrupt Enable

It is not desirable to get an interrupt on the instruction boundary of writing the SLEEP bit. This may cause all firmware preparations for going to sleep to be bypassed, if a sleep command is executed on a return from interrupt (reti) instruction. To prevent this, disable interrupts temporarily before sleep preparations and then re-enable before going to sleep. Because of the timing of the Global Interrupt instruction, an interrupt cannot occur during the next instruction, which in this case is setting the SLEEP bit.

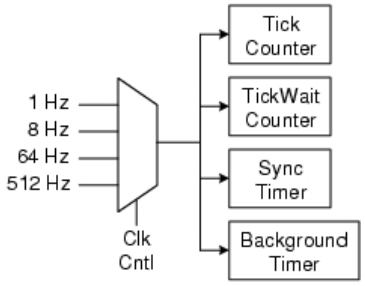
8.2.3 I^2C Slave with Sleep Mode

There are a few complications using an I^2C Slave in sleep mode. Because the IMO and CPU are shut during sleep, there is no processing within the CapSense controller. The problem arises with the I^2C address. When an I^2C START condition is sent to a particular address, the CapSense controller cannot process the address and therefore responds with a NAK. A typical workaround is to set up falling edge interrupts on either the clock or data lines of the I^2C bus. The master can then send a dummy START condition to wake up the CapSense controller. There is a lag time between waking up and being able to process an I^2C address, so the master may need to delay up to 200 µs before the next transmission or continue to send until an ACK is received. This solution has a second problem in that the CapSense controller will wake up on any I^2C falling edge traffic, which causes higher total active time and sleep currents. Another solution is to use a third GPIO pin to wake up the CapSense controller and then send the initial START condition after the appropriate delay time.

8.2.4 Sleep Timer

The CapSense controller offers a sleep timer and a Sleep Timer User Module. These are used while CapSense controller is asleep and both perform similar functions. The actual sleep timer runs off the internal low-speed oscillator, which is never turned off. At selectable intervals of 1 Hz, 8 Hz, 64 Hz, and 512 Hz, the timer generates an interrupt. It is often useful to wake the CapSense controller up periodically, to do some processing or check for activity. An example of this is to periodically wake up to scan a sensor. The Sleep Timer User Module uses the sleep timer to generate some additional functionality. This functionality includes a background tick counter to generate periodic interrupts, a delay function for program loops, a settable down counter, and a loop governor to control loop time. Figure 8-1 shows a simple block diagram for this functionality.

Figure 8-1. Sleep Timer User Module Block Diagram



9. Resources

9.1 Website

The Cypress CapSense Controllers website gives access to all of the reference material discussed in this section.

The CY8C21x34/B web page has a variety of technical resources for the CapSense CY8C21x34/B family of devices.

9.2 Datasheet

The datasheet for the CapSense CY8C21x34/B family of devices is available at http://www.cypress.com.

- CY8C21234, CY8C21334, CY8C21434, CY8C21534, CY8C21634
- CY8C21234B, CY8C21334B, CY8C21434B, CY8C21534B, CY8C21634B

9.3 Technical Reference Manual

Cypress has created the following technical reference manual to provide quick and easy access to information on CapSense controller functionality, including top-level architectural diagrams, registers, and timing diagrams.

- CY8CPLC20, CY8CLED16P01, CY8C29x66, CY8C27x43, CY8C24x94, CY8C24x23, CY8C24x23A, CY8C22x13, CY8C21x34, CY8C21x23, CY7C64215, CY7C603xx, CY8CNP1xx, and CYWUSB6953 PSoC(R) Programmable System-on-Chip Technical Reference Manual (TRM)

9.4 Development Kits

9.4.1 Universal CapSense Controller Kit

Universal CapSense Controller Kits feature predefined control circuitry and plug-in hardware to make prototyping and debugging easy. Programming and I^2C-to-USB Bridge hardware are included for tuning and data acquisition.

- CY3280-BK1 Universal CapSense Controller

9.4.2 Universal CapSense Module Boards

9.4.2.1 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.

9.4.2.2 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.

9.4.2.3 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.

9.4.2.4 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.

9.4.2.5 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.

9.4.3 In-Circuit Emulation (ICE) Kit

The ICE pod provides the interconnection between the CY3215-DK In-Circuit Emulator and the target PSoC device in a prototype system by way of a flex cable or PCB through package-specific pod feet. The following pod is available:

- CY3250-21X34QFN In-Circuit Emulation (ICE) Pod Kit for Debugging QFN CY8C21x34 PSoC Devices

9.5 PSoC Programmer

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

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

9.6 MultiChart

MultiChart is a simple PC tool 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.

9.7 PSoC Designer

Cypress offers an exclusive Integrated Design Environment, PSoC Designer. With PSoC Designer, you can configure analog and digital blocks, develop firmware, and tune your design. Applications are developed in a drag-and-drop design environment using a library of precharacterized analog and digital functions, including CapSense. PSoC Designer comes with a built-in C compiler and an embedded programmer. A pro compiler is available for complex designs.

9.8 Code Examples

Cypress offers a large collection of code examples to get your design up and running fast.

- CapSense Controller Code Examples Design Guide
- CSD Software Filters with EzI2Cs Slave on CY8C21x34
- Interfacing PSoC to an SPI EEPROM
- Pseudo Random Sequence Generator
- CSD with I2CHW Slave on CY8C21x34
- CMPPRG User Module Example
- Receive String using CYRF6936 and CY8C27643
- CSD with SPIS on CY8C21x34

9.9 Design Support

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

- Knowledge Base 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.
- CapSense Product Selector Guide – See the complete product offering of the Cypress CapSense product line.
- 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: AN66271 - CY8C21x34/B CapSense® Design Guide Document Number: 001-66271			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**	12/29/2010	ANBA	New Design Guide
*A	3/3/2011	ARVM	Multiple chapter enhancements for content and reader clarity
*B	12/07/2011	MATT	Added bulleted abstract Added Section 1.2 Moved Document Revision History to end of document Rewrote Section 2 for clarity Updated SmartSense tuning Added Water Tolerance and Proximity Sensing Added Low Power
*C	01/30/2012	VAIR	Corrected maximum rawcounts value from 2^(N-1) to (2^N) - 1.
*D	06/19/2012	ANWA	Added Section 3.3.5 Updated the entire sections 3.12.4 and 3.12.5 with new content.
*E	09/06/2012	ZINE	Updated links to external documents
*F	01/16/2015	DCHE	Added references to '*Getting Started with PSoC 1*' and '*PSoC 1 Device Programming Application Notes*' in Table 1-1.
*G	01/20/2016	VAIR	Added Glossary.
*H	04/27/2017	AESATMP9	Updated logo and copyright.