



# AN65973 - CY8C20xx6A/H/AS

## CapSense® Design Guide

Doc. No. 001-65973 Rev. \*J

Cypress Semiconductor  
198 Champion Court  
San Jose, CA 95134-1709  
[www.cypress.com](http://www.cypress.com)

## Copyrights

© Cypress Semiconductor Corporation, 2010-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit [cypress.com](http://cypress.com). Other names and brands may be claimed as property of their respective owners.

# Contents



<b>1. Introduction</b>	<b>6</b>
1.1 Abstract	6
1.2 Cypress's CapSense Documentation Ecosystem	6
1.3 CY8C20xx6 A/H/AS CapSense Family Features	8
1.3.1 Advanced Touch Sensing Features	8
1.3.2 Device Features	9
1.4 Document Conventions	10
<b>2. CapSense Technology</b>	<b>11</b>
2.1 CapSense Fundamentals	11
2.2 Capacitive Sensing Methods in CY8C20xx6A/AS/H	12
2.2.1 CapSense Sigma-Delta (CSD)	13
2.2.2 CapSense Successive Approximation Electromagnetic Compatibility (CSA_EMC)	14
2.3 SmartSense Auto-Tuning	15
<b>3. CapSense Design Tools</b>	<b>17</b>
3.1 Overview	17
3.1.1 PSoC Designer and User Modules	17
3.1.2 Universal CapSense Controller Kit	18
3.1.3 Universal CapSense Controller Module Board	18
3.1.4 CapSense Data Viewing Tools	19
3.2 User Module Overview	19
3.3 CapSense User Module Global Arrays	20
3.3.1 Raw Count	20
3.3.2 Baseline	20
3.3.3 Difference Count (Signal)	20
3.3.4 Sensor State	21
3.4 CSD User Module Parameters	21
3.4.1 User Module High-Level Parameters	22
3.4.2 CSD User Module Low-Level Parameters	24
3.4.3 CSA_EMC User Module Low-Level Parameters	25
3.4.4 SmartSense User Module Parameters	27
3.4.5 SmartSense_EMC User Module Parameters	28
<b>4. CapSense Performance Tuning with User Modules</b>	<b>30</b>

4.1	General Considerations .....	30
4.1.1	Signal, Noise, and SNR .....	30
4.1.2	Charge/Discharge Rate.....	31
4.1.3	Importance of Baseline Update Threshold Verification.....	32
4.2	Tuning the CSA_EMC User Module.....	32
4.3	Recommended C <sub>INT</sub> Value for CSA_EMC .....	33
4.4	Measuring Sensor C <sub>P</sub> .....	33
4.5	Estimating CSA_EMC Clock.....	34
4.6	Setting Settling Time.....	34
4.7	Monitoring CapSense Data.....	35
4.8	Methods to Increase SNR .....	35
4.8.1	Reduce Noise .....	35
4.8.2	Increase Signal.....	35
4.9	Tuning the CSD User Module.....	35
4.9.1	Recommended C <sub>MOD</sub> Value for CSD .....	36
4.9.2	ShieldElectrodeOut.....	37
4.9.3	I <sub>DAC</sub> Range.....	37
4.9.4	Autocalibration.....	37
4.9.5	I <sub>DAC</sub> Value .....	37
4.9.6	Precharge Source.....	37
4.9.7	Prescaler.....	37
4.9.8	Resolution.....	38
4.9.9	Scanning Speed.....	38
4.9.10	High-Level API Parameters .....	39
4.9.11	Set High-Level Parameters .....	40
4.10	Using the SmartSense User Module.....	40
4.10.1	Guidelines for SmartSense.....	40
4.10.2	Understanding the Difference .....	40
4.10.3	Recommended C <sub>CMOD</sub> Value for SmartSense.....	41
4.10.4	SmartSense User Module Parameters.....	41
4.10.5	SmartSense_EMC User Module Specific Guidelines .....	41
4.10.6	Scan Time of a CapSense Sensor.....	42
4.10.7	SmartSense Response Time .....	43
4.10.8	Method to Ensure Minimum SNR Using the SmartSense_EMC User Module.....	44
4.10.9	Firmware Design Guidelines .....	45
<b>5.</b>	<b>Design Considerations .....</b>	<b>47</b>
5.1	Overlay Selection.....	47
5.2	ESD Protection.....	48
5.2.1	Prevent.....	48
5.2.2	Redirect .....	48
5.2.3	Clamp.....	48
5.3	Electromagnetic Compatibility (EMC) Considerations.....	48
5.3.1	Radiated Interference.....	48
5.3.2	Radiated Emissions .....	49
5.3.3	Conducted Immunity and Emissions .....	49
5.4	Software Filtering.....	49
5.5	Power Consumption.....	50
5.5.1	System Design Recommendations .....	50

5.5.2	Sleep-Scan Method .....	50
5.5.3	Response Time versus Power Consumption .....	50
5.5.4	Measuring Average Power Consumption .....	51
5.6	Pin Assignments .....	51
5.7	GPIO Load Transient.....	52
5.7.1	Hardware Guidelines to Reduce GPIO Load Transient Noise .....	53
5.7.2	Firmware Guidelines to Compensate GPIO Load Transient Noise.....	53
5.8	PCB Layout Guidelines.....	55
<b>6.</b>	<b>Low-Power Design Considerations .....</b>	<b>56</b>
6.1	Additional Power Saving Techniques.....	56
6.1.1	Set Drive Modes to Analog HI-Z.....	56
6.1.2	Putting it All Together .....	57
6.1.3	Sleep Mode Complications.....	57
6.1.4	Pending Interrupts.....	57
6.1.5	Global Interrupt Enable .....	57
6.2	Post Wakeup Execution Sequence .....	58
6.2.1	PLL Mode Enabled .....	58
6.2.2	Execution of Global Interrupt Enable.....	58
6.2.3	I <sup>2</sup> C Slave with Sleep Mode .....	58
6.2.4	Sleep Timer.....	58
<b>7.</b>	<b>Resources.....</b>	<b>59</b>
7.1	Website.....	59
7.2	Datasheet.....	59
7.3	Technical Reference Manual .....	59
7.4	Development Kits.....	59
7.4.1	Universal CapSense Controller Kit.....	59
7.4.2	Universal CapSense Module Boards.....	59
7.4.3	In-Circuit Emulation (ICE) Kits .....	60
7.5	Sample Board Files .....	60
7.6	PSoC Programmer.....	62
7.7	CapSense Data Viewing Tools.....	62
7.8	PSoC Designer .....	62
7.9	Code Examples.....	63
7.10	Design Support .....	63
	<b>Glossary.....</b>	<b>64</b>
	<b>Revision History .....</b>	<b>70</b>
	Document Revision History.....	70

# 1. Introduction



## 1.1 Abstract

This document provides design guidance for using the capacitive sensing (CapSense®) functionality with the CY8C20xx6A/AS/H family of CapSense controllers. The following topics are covered in this guide:

- [Features of the CY8C20xx6A/AS/H family of CapSense controllers](#)
- [CapSense principles of operation](#)
- [Introduction to CapSense design tools](#)
- [Detailed guide to tuning the CapSense system for optimal performance](#)
- [System electrical and mechanical design considerations for CapSense](#)
- [Low-power design considerations for CapSense](#)
- [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

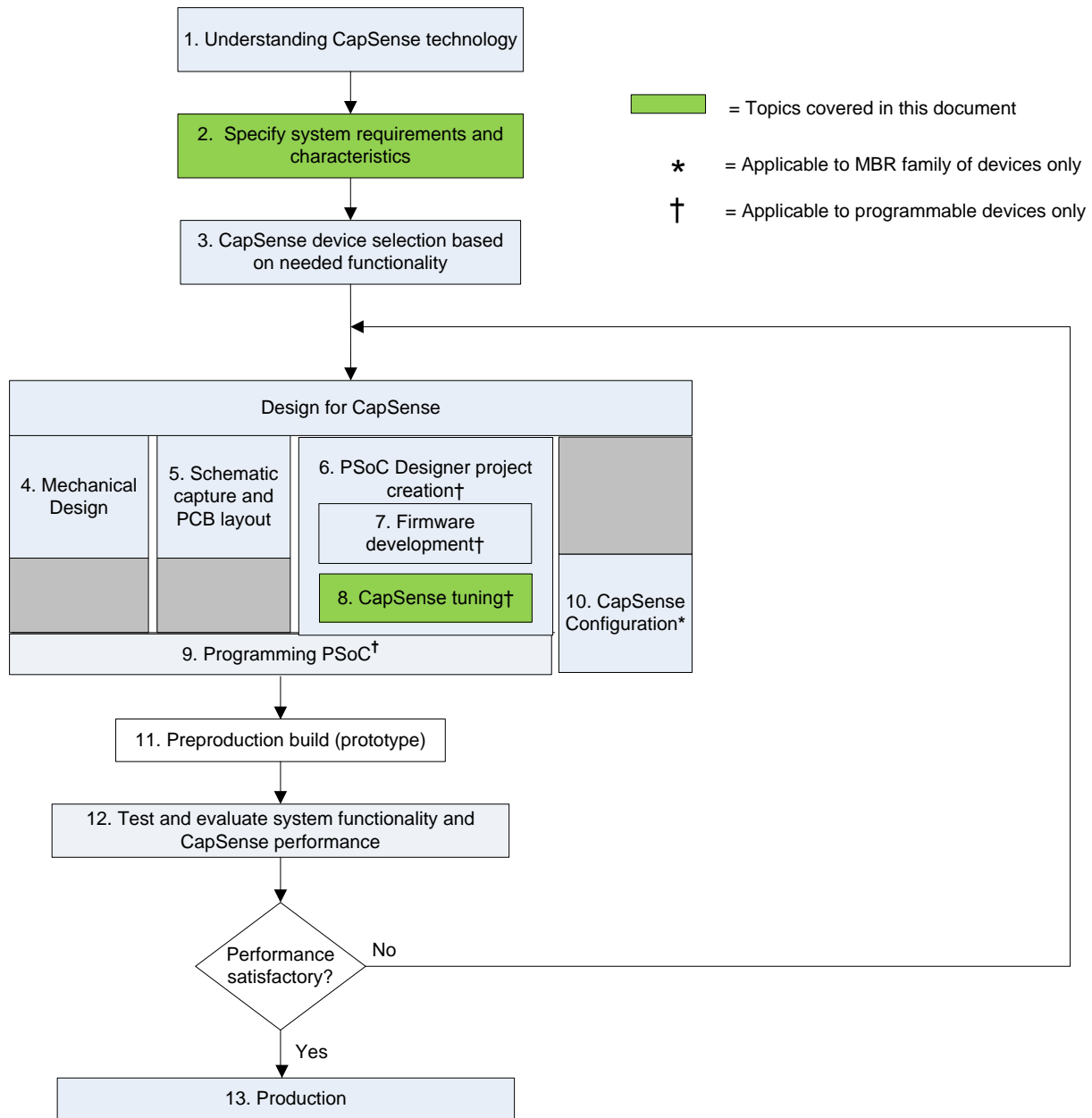


Table 1-1. Cypress Documents Supporting Numbered Design Tasks of Figure 1-1

Numbered Design Task of Figure 1-1	Supporting Cypress CapSense Documentation
1	<ul style="list-style-type: none"> <li>• <a href="#">Getting Started with CapSense</a></li> </ul>
2	<ul style="list-style-type: none"> <li>• <a href="#">Getting Started with CapSense</a></li> <li>• <a href="#">CY8C20xx6A/AS/H CapSense Device Datasheets</a></li> </ul>
3	<ul style="list-style-type: none"> <li>• <a href="#">Getting Started with CapSense</a></li> <li>• PSoC Family-Specific CapSense Design Guide (this document)</li> </ul>
4	<ul style="list-style-type: none"> <li>• <a href="#">Getting Started with CapSense</a></li> </ul>
5	<ul style="list-style-type: none"> <li>• <a href="#">Getting Started with CapSense</a></li> <li>• <a href="#">PSoC Designer™ User Guides</a></li> </ul>
6	<ul style="list-style-type: none"> <li>• <a href="#">PSoC Designer User Guides</a></li> </ul>
7	<ul style="list-style-type: none"> <li>• <a href="#">Assembly Language User Guide</a></li> <li>• <a href="#">C Language Compiler User Guide</a></li> <li>• <a href="#">CapSense Code Examples</a></li> <li>• PSoC Family-Specific Technical Reference Manual (for <a href="#">CY8C20xx6A/AS/H</a>)</li> </ul>
8	<ul style="list-style-type: none"> <li>• PSoC Family-Specific CapSense Design Guide (this document)</li> <li>• PSoC Family-Specific CapSense User Module Datasheets (CSD and <a href="#">SmartSense™</a>)</li> <li>• PSoC Family-Specific Technical Reference Manual (for <a href="#">CY8C20xx6A/AS/H</a>)</li> <li>• <a href="#">CapSense Controller Code Examples Design Guide</a></li> <li>• <a href="#">AN2397 -CapSense Data Viewing Tools</a></li> </ul>
9	<ul style="list-style-type: none"> <li>• <a href="#">Programmer User Guide</a></li> <li>• <a href="#">MiniProg3 User Guide</a></li> <li>• <a href="#">AN2026c - In-System Serial Programming (ISSP) Protocol for CY8C20xx6, CY8C20xx6A, CY8CTMG2xx, and CY8CTST2xx, CY7C643xx, and CY7C604xx</a></li> <li>• <a href="#">AN44168 - PSoC 1 Device Programming using External Microcontroller (HSSP)</a></li> <li>• <a href="#">AN59389 - Host-Sourced Serial Programming for CY8C20xx6, CY8CTMG2xx, and CY8CTST2xx</a></li> </ul>
11	<ul style="list-style-type: none"> <li>• PSoC Family-Specific CapSense Design Guide (this document)</li> <li>• <a href="#">CapSense Code Examples</a></li> </ul>

## 1.3 CY8C20xx6A/H/AS CapSense Family Features

Cypress's CY8C20xx6A/H/AS is a low-power, high-performance, programmable CapSense controller family that includes the following features.

### 1.3.1 Advanced Touch Sensing Features

- Programmable capacitive sensing elements
  - ☐ Supports a combination of CapSense buttons, sliders, and proximity sensors
  - ☐ Integrated API to implement buttons and sliders
  - ☐ Supports up to 36 capacitive sensors or 36 GPIO or sliders
  - ☐ Supports parasitic sensor capacitance range of 5 pF to 45 pF
- SmartSense™ Auto-tuning enables fast time to market
  - ☐ Sets and monitors tuning parameters automatically at power-up and at run time
  - ☐ Design portability - self tunes for changes in user interface design
  - ☐ Environmental compensation during run time
  - ☐ Detects touches as low as 0.1 pF



- Enhanced noise immunity and robustness
  - ☐ SmartSense compensates for environment and noise variations automatically
  - ☐ SmartSense\_EMC offers superior noise immunity for applications with challenging conducted and radiated noise conditions
  - ☐ Internal regulator provides stability against power supply noise and ripple up to 500 mV of supply  $V_{DD}$  ripple acceptable
  - ☐ Integrated API of software filters for SNR improvement
- Ultra low-power consumption
  - ☐ Three power modes for optimized power consumption
  - ☐ Active, sleep, and deep-sleep modes (deep-sleep current: 100 nA)
  - ☐ 28  $\mu$ A per sensor at 125 ms scan rate

### 1.3.2 Device Features

- High-performance, low-power M8C Harvard-architecture processor
  - ☐ Up to 4 MIPS with 24-MHz internal clock, external crystal resonator, or clock signal
- Flexible on-chip memory
  - ☐ Up to 32 KB of flash and 2 KB of SRAM
  - ☐ Emulated EEPROM
- Precision, programmable clocking
  - ☐ Internal main oscillator (IMO): 6/12/24 MHz  $\pm$  5%
  - ☐ Option for precision 32-kHz external crystal oscillator
- Enhanced general-purpose input output (GPIO) features
  - ☐ Up to 36 GPIOs with programmable pin configuration
  - ☐ 25-mA sink current / GPIO and 120-mA total sink current / device
  - ☐ Internal resistive pull-up, high-z, open-drain, and strong drive modes on all GPIOs
- Peripheral features
  - ☐ Three 16-bit timers
  - ☐ Full-speed USB - 12 Mbps USB 2.0 compliant
  - ☐ I<sup>2</sup>C - Master (100 kHz) and Slave (up to 400 kHz)
  - ☐ SPI - Master and Slave - configurable range of 46.9 kHz to 12 MHz
  - ☐ Up to 10-bit ADC - 0 to 1.2 V input range
- Operating conditions
  - ☐ Wide operating voltage: 1.71 V to 5.5 V
  - ☐ Temperature range: -40 °C to +85 °C

## 1.4 Document Conventions

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

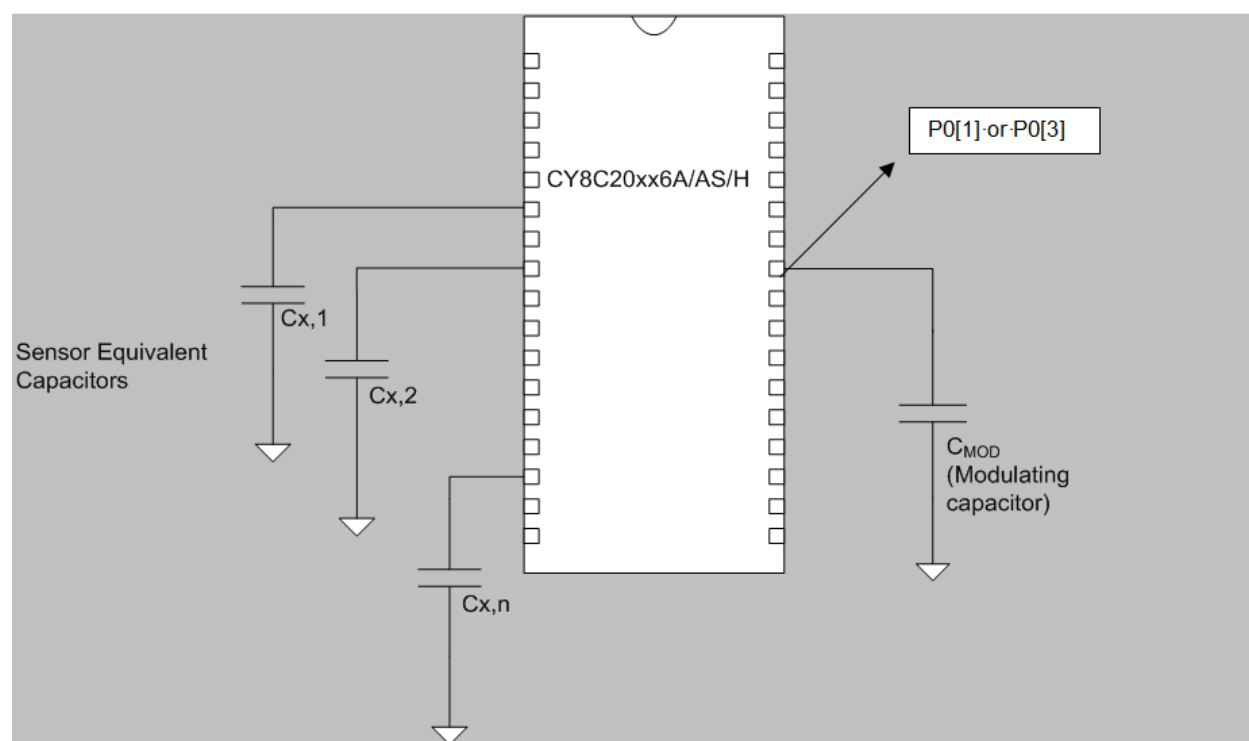
## 2. CapSense Technology



### 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 CY8C20xx6A/AS/H device converts the magnitude of each  $C_x$  into a digital code that is stored for post processing. The other component,  $C_{MOD}$ , is used by the CapSense controller's internal circuitry and is discussed in detail in [Capacitive Sensing Methods in CY8C20xx6A/AS/H](#).

Figure 2-1. CapSense Implementation in a CY8C20xx6A/AS/H PSoC Device



As shown in [Figure 2-1](#), 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 touches the overlay, the conductivity and mass of the body effectively introduce 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.

$$C_F = \frac{\epsilon_0 \epsilon_r A}{D}$$

Equation 1

Where:

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

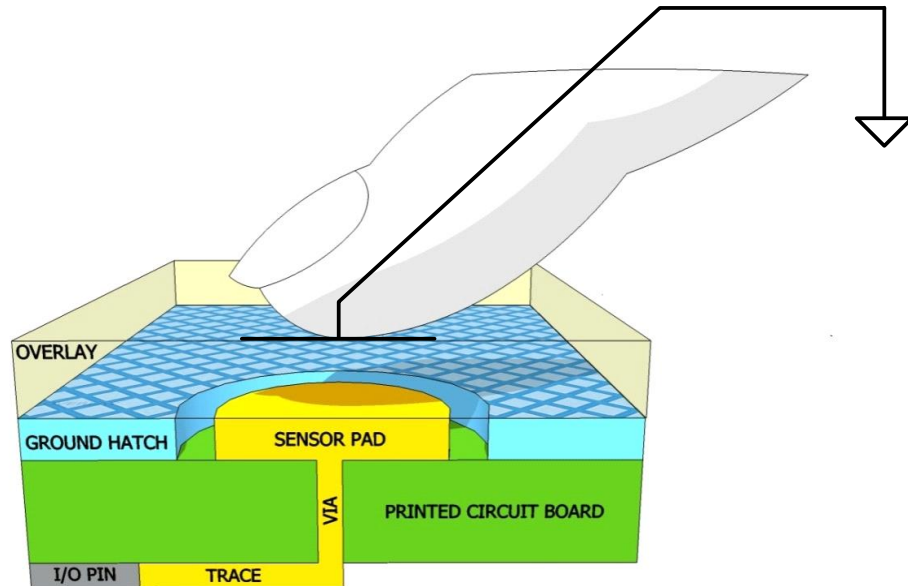
$\epsilon_0$  = Free space permittivity

$\epsilon_r$  = Dielectric constant (relative permittivity) of overlay

A = Area of finger and sensor pad overlap

D = Overlay thickness

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



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 electric field coupling between the sensor pad, traces, vias, and other conductors in the system such as ground plane, other traces, any metal in the product's chassis or enclosure, and so on. The CapSense controller measures the total capacitance ( $C_X$ ) connected to a sensor pin.

When a finger is not touching a sensor:

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

When a finger is on the sensor pad,  $C_X$  equals the sum of  $C_P$  and  $C_F$ :

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

In general,  $C_P$  is an order of magnitude greater than  $C_F$ .  $C_P$  usually ranges from 10 pF to 20 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](#).

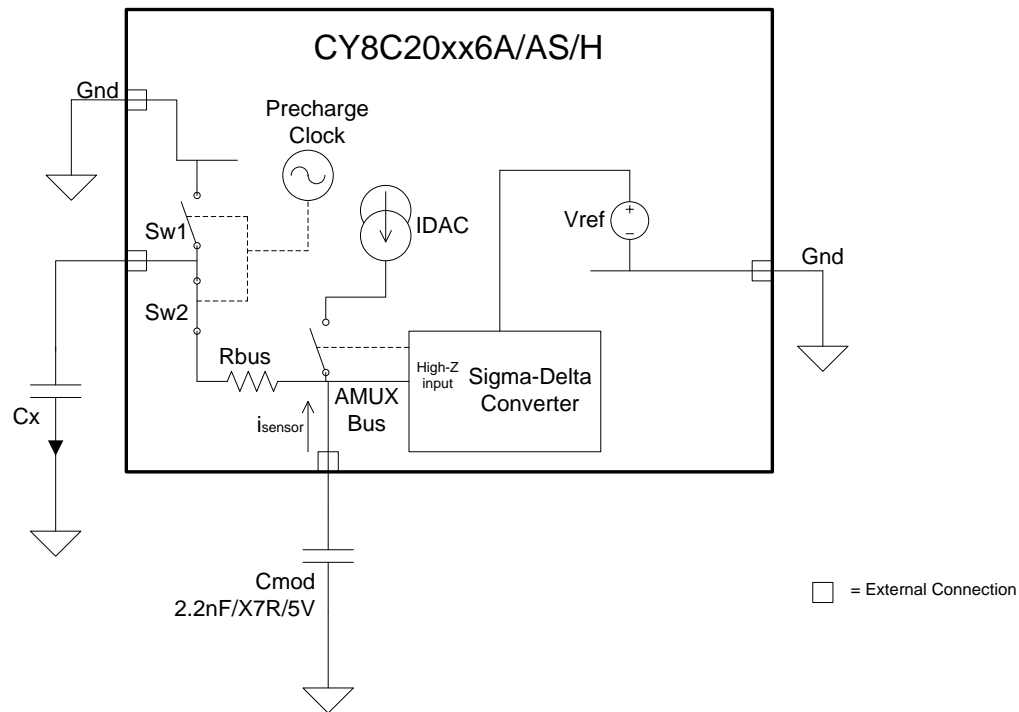
## 2.2 Capacitive Sensing Methods in CY8C20xx6A/AS/H

CY8C20xx6A/AS/H devices support several CapSense methods for converting sensor capacitance ( $C_X$ ) into digital counts. These are CapSense Sigma Delta (CSD), CapSense Successive Approximation Electro-Magnetic Compatibility (CSA\_EMC), SmartSense, and SmartSense\_EMC. These methods are implemented in the form of a PSoC Designer User Module and are described in the following sections.

### 2.2.1 CapSense Sigma-Delta (CSD)

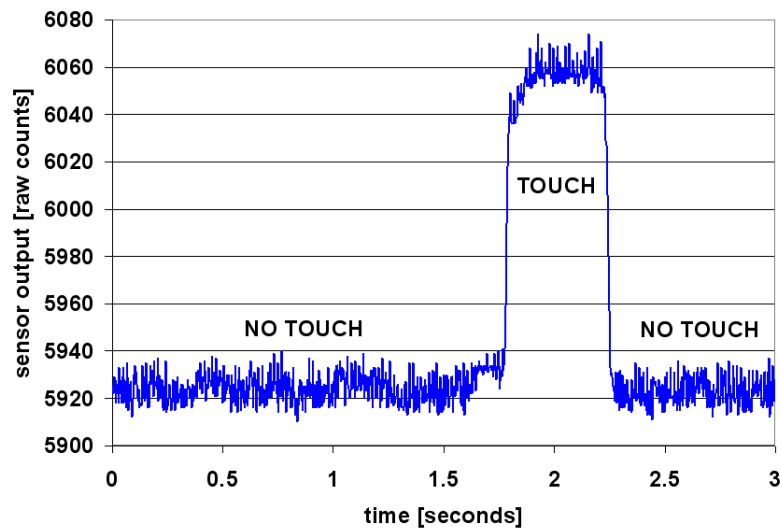
The CapSense Sigma-Delta method in CY8C20xx6A/AS/H devices incorporates  $C_X$  into a switched capacitor circuit as shown in Figure 2-3. The sensor ( $C_X$ ) is alternatively connected to GND and the analog mux (AMUX) bus by the underlapped switches Sw1 and Sw2, respectively. Sw1 and Sw2 are driven by a Precharge clock to bleed current ( $I_{\text{SENSOR}}$ ) from the AMUX bus. The magnitude of  $I_{\text{SENSOR}}$  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 ( $I_{\text{DAC}}$ ), which charges AMUX such that the average AMUX bus voltage is maintained at  $V_{\text{REF}}$ . The sensor bleeds off the charge  $I_{\text{SENSOR}}$  from the modulating capacitor ( $C_{\text{MOD}}$ ).  $C_{\text{MOD}}$  in combination with  $R_{\text{BUS}}$  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 ( $V_{\text{REF}}$ ), the Sigma-Delta converter matches the average charge current ( $I_{\text{DAC}}$ ) to  $I_{\text{SENSOR}}$  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.

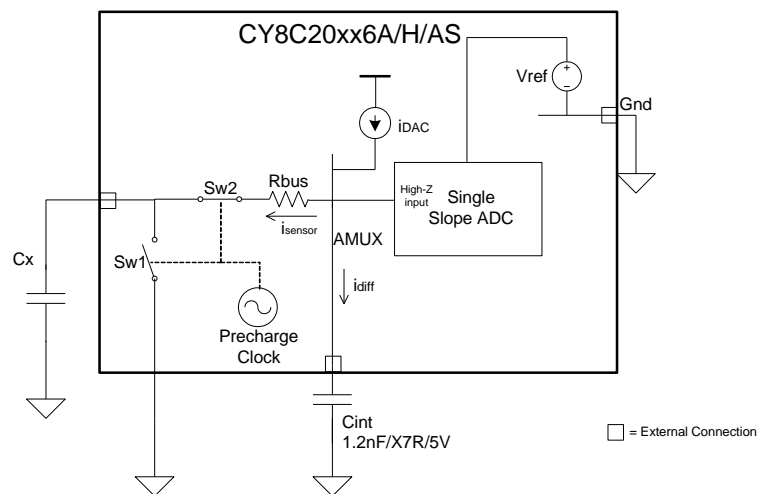
Figure 2-4. CSD Raw Counts during a Finger Touch



## 2.2.2 CapSense Successive Approximation Electromagnetic Compatibility (CSA\_EMC)

The CapSense Successive Approximation Electromagnetic Compatibility (CSA\_EMC) method used in CY8C20xx6A devices incorporates  $C_X$  into a switched capacitor circuit, as shown in Figure 2-5.

Figure 2-5. CSA\_EMC Block Diagram



The constant current source ( $I_{DAC}$ ) provides  $I_{DAC}$  amount of current into the AMUX. The sensor ( $C_X$ ) which is alternatively connected between AMUX bus and GND by the switches  $Sw1$  and  $Sw2$ , respectively, drains away  $I_{SENSOR}$  amount of current from the AMUX bus. The magnitude of  $I_{SENSOR}$  is directly proportional to the magnitude of  $C_X$ . The switches  $Sw1$  and  $Sw2$  are clocked by a non-overlapping clock known as precharge clock.

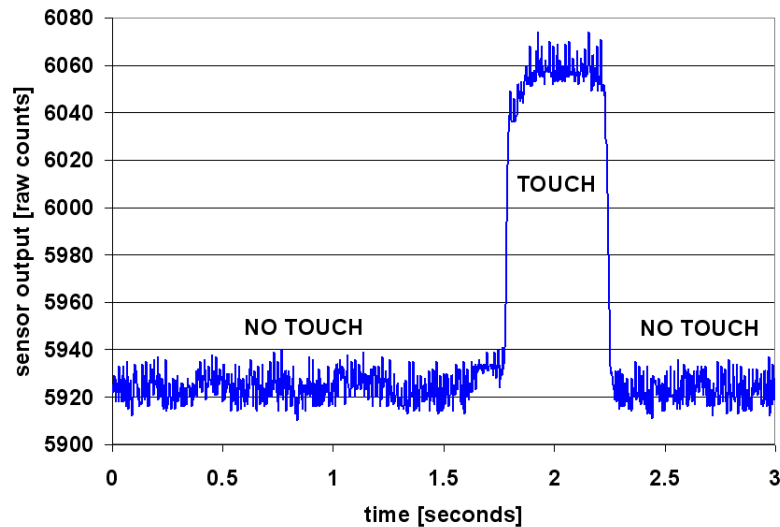
The integration capacitor  $C_{INT}$  integrates the difference current  $i_{Diff}$  (difference of  $I_{DAC}$  and  $I_{SENSOR}$ ) and increases its potential. This charge integration continues until the potential developed across  $C_{INT}$  reaches an equilibrium level at which  $I_{SENSOR}$  becomes equal to  $I_{DAC}$ . This integration time is referred to as settling time.

A single slope ADC is used to convert the equilibrium potential on  $C_{INT}$  to digital output counts, known as raw count, which is proportional to  $C_X$ . This raw count is interpreted by high-level algorithms to resolve sensor state.

The  $I_{DAC}$  current is set using successive approximation method to make sure the equilibrium voltage on  $C_{INT}$  is in the linear conversion region of ADC.

Figure 2-6 plots the CSA\_EMC raw counts from a number of consecutive scans during which the sensor is touched and then released by a finger. 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 the steady state raw count level to a predetermined threshold, the high-level algorithms can determine whether the sensor is in an ON (Touch) or OFF (No Touch) state.

Figure 2-6. CSA\_EMC Raw Counts during a Finger Touch



The CSA\_EMC CapSense algorithm is enhanced to work well in the presence of RF interference. CSA\_EMC is used in applications where CapSense is exposed to conducted interference, AC noise, and other noise sources such as inverters, transformers, and power supplies. [CSA\\_EMC User Module Low-Level Parameters](#) discusses this topic in detail.

## 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. 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 run time. This 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.

### 2.3.1.1 Process Variation

The SmartSense User Module (UM) for the CY8C20xx6A/H/AS 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 45 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 2-7. Design of Laptop Multimedia Keys for a 21-inch Model



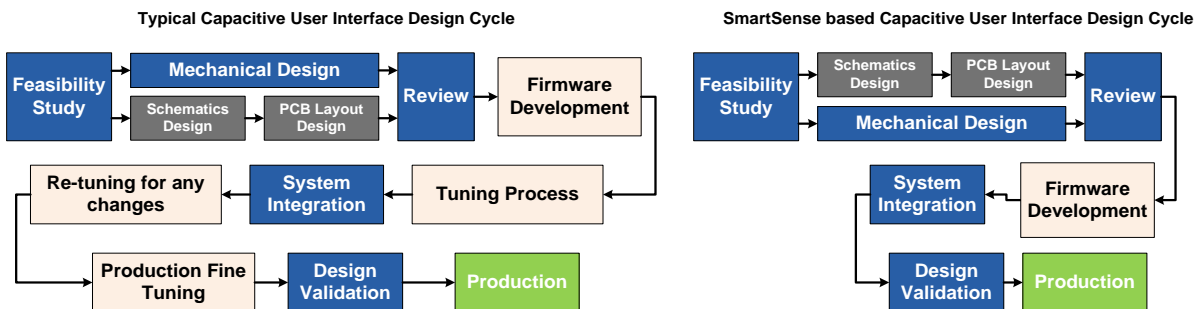
Figure 2-8. Design of Laptop Multimedia Keys for a 15-inch Model with Identical Functionality and Button Size



### 2.3.1.2 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 2-9](#) compares the design cycles of a typical touch-sensing controller and a SmartSense-based design.

Figure 2-9. Typical Capacitive Interface Design Cycle Comparison





## 3. CapSense Design Tools



### 3.1 Overview

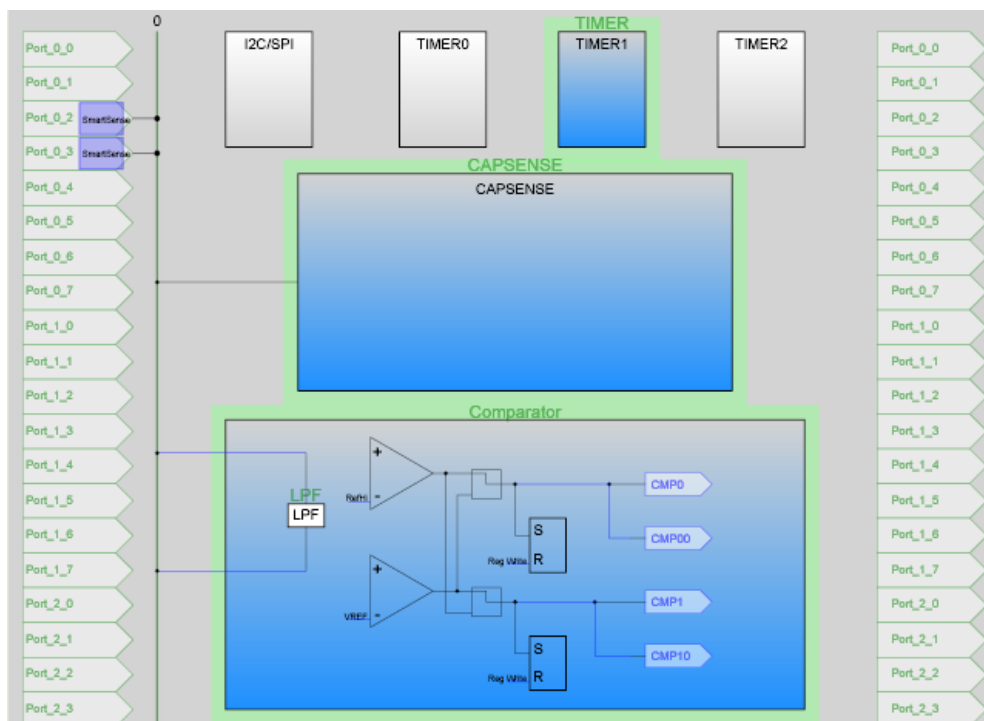
Cypress offers a full line of hardware and software tools for developing your CapSense capacitive touch-sense application. A basic development system for the CY8C20xx6A/H/AS family includes the following components. See [Resources](#) for ordering information.

#### 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.

The CSA\_EMC User Module implements 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<sup>2</sup>C, SPI, TX8, and timers.

Figure 3-1. PSoC Designer Device Editor



### 3.1.1.1 Getting Started with CapSense User Modules

To create a new CY8C20xx6A/H/AS project in PSoC Designer:

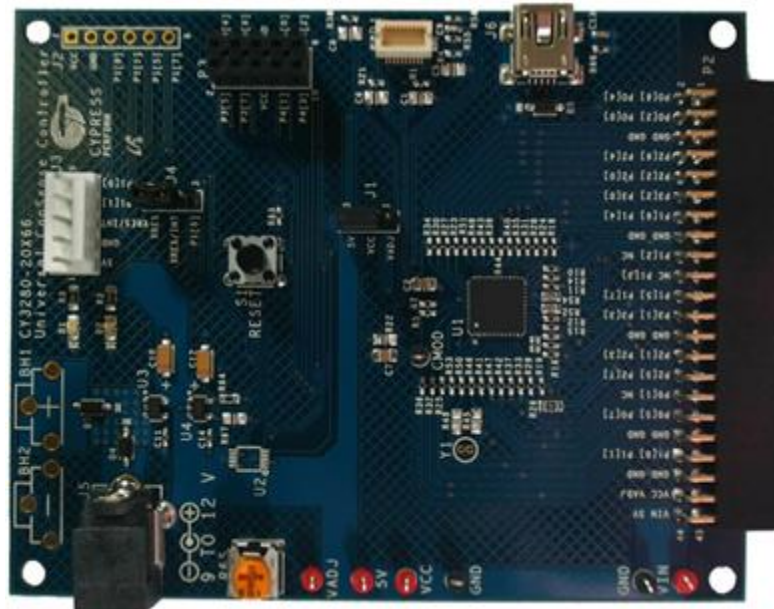
1. Create a new PSoC Designer project with CY8C20xx6 as the target device.
2. Select and place the CSD/CSA\_EMC/SmartSense User Module.
3. Right-click the user module to access the User Module wizard.
4. Set button sensor count, slider configuration, pin assignments, and associations.
5. Set pins and global user module parameters.
6. Generate the application and switch to the Application Editor.
7. Adapt sample code from the user module datasheet to implement buttons or sliders.

For a detailed step-by-step procedure to create a PSoC Designer project and to configure the User Module wizard, see the datasheet of the specific user module. For code examples on CapSense user modules, see [Code Examples](#).

### 3.1.2 Universal CapSense Controller Kit

The Universal [CY3280-20xx6](#) CapSense Controller Kit features predefined control circuitry and plug-in hardware to make prototyping and debugging easy. The MiniProg hardware is included with the kit for programming, and the I<sup>2</sup>C-to-USB Bridge hardware is included for tuning and data acquisition.

Figure 3-2. CY3280-20xx6 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.

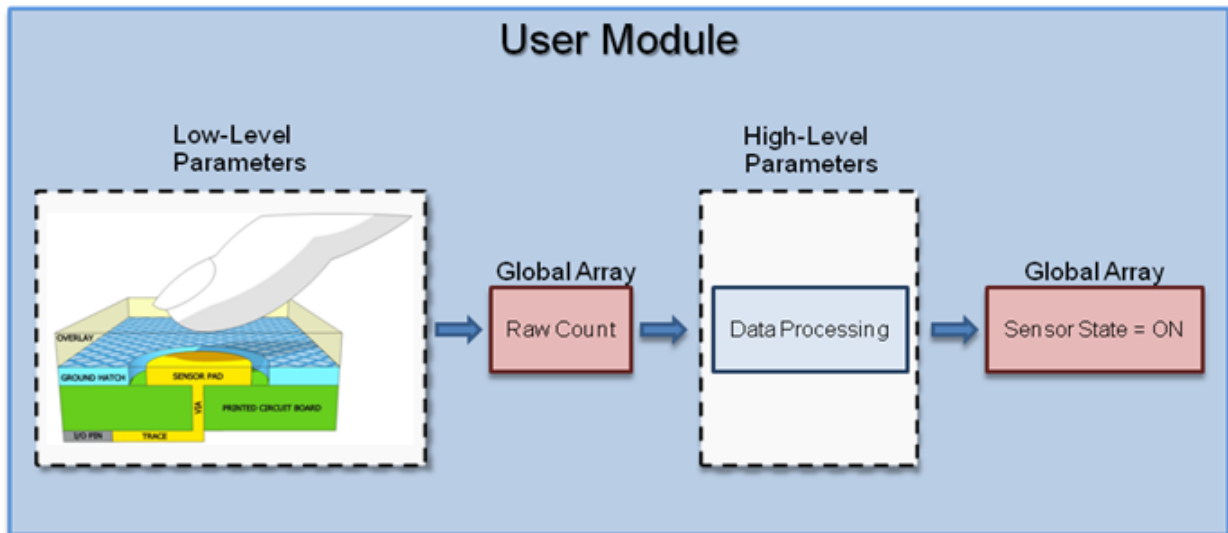
- [CY3280-BSM](#) Simple Button Module
- [CY3280-BMM](#) Matrix Button Module
- [CY3280-SLM](#) Linear Slider Module
- [CY3280-SRM](#) Radial Slider Module
- [CY3280-BBM](#) Universal CapSense Prototyping Module

### 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. 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.

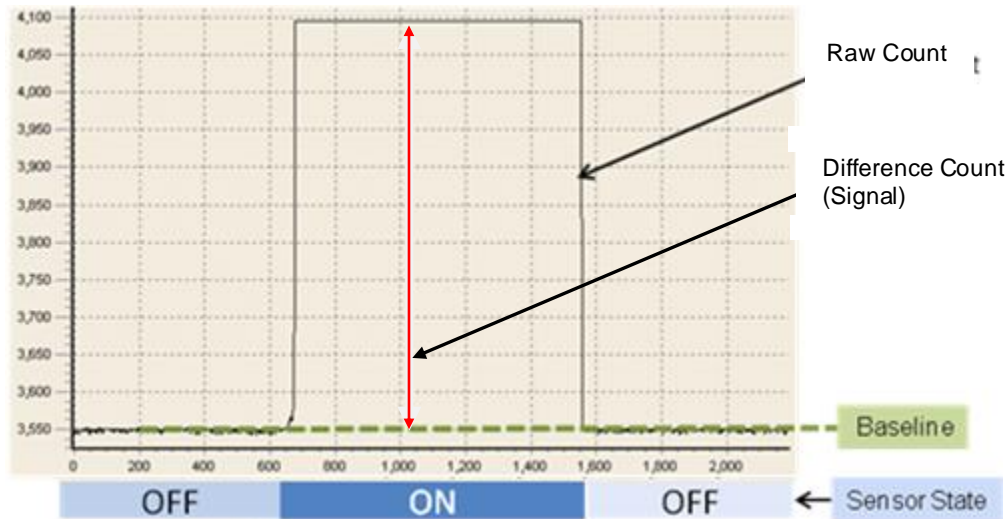
Low-level parameters, such as the speed and resolutions for scanning sensors, define the behavior of the sensing method at the physical layer and relate to the conversion from capacitance to raw count. Low-level parameters are unique to each type of sensing method and are described in [CSD User Module Low-Level Parameters](#), [CSA\\_EMC User Module Low-Level 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 [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`, `SmartSense`, or `CSA_EMC`.

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`, `SmartSense`, or `CSA_EMC`.

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.

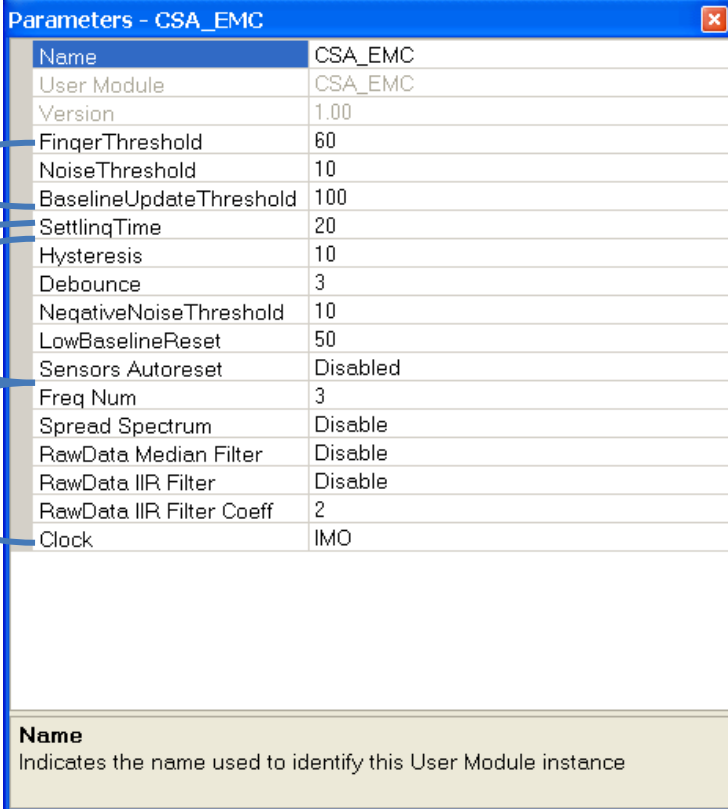
Sensor states are stored in a byte array named `UMname_baSnsOnMask[]` array, where `UMname` can be CSD, SmartSense, or CSA\_EMC. This array is defined in the header file `UMname.h`. Each array element stores the sensor state of eight consecutive sensors.

Sensor states are updated by the user module API `UMname_bIsAnySensorActive()`.

## 3.4 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 CSA\_EMC User Module parameters and how they are classified. See [Figure 3-6](#) for a list of CSD User Module parameters and how they are classified.

Figure 3-5. PSoC Designer – CSA\_EMC Parameter Window

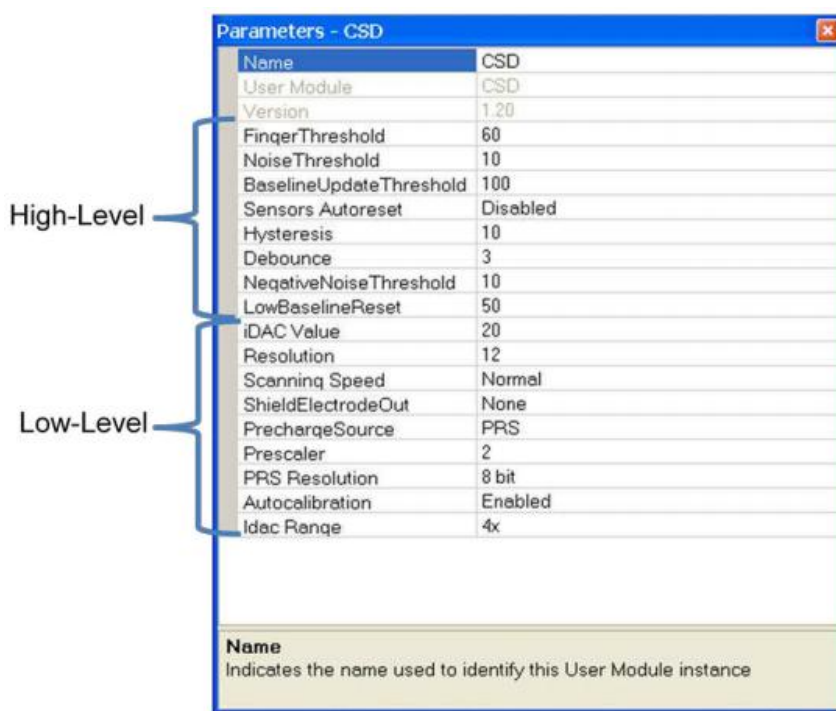


Name	Value
User Module	CSA_EMC
Version	1.00
FingerThreshold	60
NoiseThreshold	10
BaselineUpdateThreshold	100
SettlingTime	20
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	10
LowBaselineReset	50
Sensors Autoreset	Disabled
Freq Num	3
Spread Spectrum	Disable
RawData Median Filter	Disable
RawData IIR Filter	Disable
RawData IIR Filter Coeff	2
Clock	IMO

Name
Indicates the name used to identify this User Module instance

Figure 3-6. PSoC Designer – CSD Parameter Window



Name	CSD
User Module	CSD
Version	1.20
FingerThreshold	60
NoiseThreshold	10
BaselineUpdateThreshold	100
Sensors Autoreset	Disabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	10
LowBaselineReset	50
iDAC Value	20
Resolution	12
Scanning Speed	Normal
ShieldElectrodeOut	None
PrechargeSource	PRS
Prescaler	2
PRS Resolution	8 bit
Autocalibration	Enabled
Idac Range	4x

**Name**  
Indicates the name used to identify this User Module instance

## 3.4.1 User Module High-Level Parameters

### 3.4.1.1 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 3 to 255.

For the recommended value, see [Set High-Level Parameters](#).

### 3.4.1.2 Hysteresis

The Hysteresis setting prevents the sensor state from random toggling because of system noise. The Hysteresis parameter is used in conjunction with the Finger Threshold to determine the sensor state. If the Difference Count exceeds the Finger Threshold + Hysteresis level for Debounce number of samples, the sensor state changes from OFF to ON. If the Difference Count drops below the Finger Threshold – Hysteresis level, the sensor state changes from ON to OFF. Equation 4 illustrates the Hysteresis function.

$$\text{if } \text{DifferenceCount} \geq \text{FingerThreshold} + \text{Hysteresis}, \text{SensorState} = \text{ON}$$

$$\text{if } \text{DifferenceCount} \leq \text{FingerThreshold} - \text{Hysteresis}, \text{SensorState} = \text{OFF}$$

Equation 4

For the recommended value, see [Set High-Level Parameters](#).

### 3.4.1.3 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.

For the recommended value, see [Set High-Level Parameters](#).

#### 3.4.1.4 Baseline Update Threshold

As previously explained, the baseline variables keep track of any gradual variations in raw count values. In other words, baseline variables hold the output of a digital low-pass filter to which the input raw count values are fed. The Baseline Update Threshold parameter is used to adjust the time constant of this low-pass filter.

Baseline update threshold is directly proportional to the time constant of this filter. The higher the baseline update threshold value, the higher the time constant.

Possible values are 0 to 255.

For the recommended value, see [Set High-Level Parameters](#).

#### 3.4.1.5 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 minus Hysteresis.

For the recommended value, see [Set High-Level Parameters](#).

#### 3.4.1.6 Negative Noise Threshold

The Negative Noise Threshold helps the user module to understand the lower limit of noise counts in the raw count. The baselining update algorithm is paused when the raw count is below the baseline and the difference between them is greater than this threshold.

Possible values are 0 to 255.

For the recommended value, see [Set High-Level Parameters](#).

#### 3.4.1.7 Low Baseline Reset

The Low Baseline Reset parameter works in conjunction with the Negative Noise Threshold parameter. If the sample count values are less than the baseline minus the negative noise threshold for the specified number of samples, the baseline is set to the new raw count value. 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.

For the recommended value, see [Set High-Level Parameters](#).

#### 3.4.1.8 Sensors Autoreset

Enabling Sensor Autoreset feature prevents sensor from being in ON state for indefinite period of time. This parameter determines whether the baseline is updated always, or only when the difference counts are below the Noise Threshold parameter.

When Sensors Autoreset is enabled, the baseline is always updated even if the Difference Count is greater than Noise Threshold. This limits the maximum duration of a sensor in the ON state when the sensor is touched continuously (typically for more than 5 to 10 seconds), but prevents the sensor from permanently being ON when the raw counts accidentally increase without finger touch on the sensor. This sudden increase can be caused by an electrical damage in the system or by a metal object placed close to the sensor.

When Sensors Autoreset is disabled, the baseline is updated only when the Difference Count is below the Noise Threshold parameter. Hence, as long as the sensor is touched, the sensor is in ON state.

Possible values are 'Enabled' and 'Disabled'. For the recommended setting, see [Set High-Level Parameters](#).



### 3.4.2 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.

#### 3.4.2.1 $I_{DAC}$ Value

The  $I_{DAC}$  parameter sets the capacitance measurement range. A higher value corresponds to a wider range. Adjust the  $I_{DAC}$  value such that raw counts are at about 50 to 70 percent of full range. This parameter can be changed at run time using the user module API `CSD_SetIdacValue()`.

Possible values are 1 to 255.

#### 3.4.2.2 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, but reduces scan time.

Possible values are 9 to 16 bits.

Table 3-1. Resolution and Scan Speed

Resolution	Scan Speed for Individual Buttons ( $\mu$ s)			
	Ultra Fast	Fast	Normal	Slow
9	57	78	125	205
10	78	125	205	380
11	125	205	380	720
12	205	380	720	1400
13	380	720	1400	2800
14	720	1400	2800	5600
15	1400	2800	5600	11000
16	2800	5600	11000	22000

#### 3.4.2.3 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

Possible values are Ultra Fast, Fast, Normal, and Slow.

#### 3.4.2.4 Shield Electrode Out

A shield electrode is used to reduce parasitic capacitance. This parameter selects where to route the output of the shield electrode.

Possible values are P0[7] or P1[2].

#### 3.4.2.5 Precharge Source

This parameter selects the clock source for precharge switches.

Possible values are PRS and Prescaler. Use the PRS source in most cases to get better EMI immunity and lower emission.



### 3.4.2.6 Prescaler

This parameter sets the prescaler ratio and determines the precharge switch output frequency. This parameter also affects the PRS output frequency.

Possible values are 1, 2, 4, 8, 16, 32, 64, 128, and 256.

### 3.4.2.7 PRS Resolution

This parameter changes the PRS sequence length.

Possible values are 8-bit and 12-bit. Corresponding sequence lengths are 511 and 2047 input clock periods. Use an 8-bit setting if the 12-bit setting does not provide good SNR.

### 3.4.2.8 Autocalibration

When Autocalibration is enabled, the raw count value is normalized as a percentage of the max count ( $2^N - 1$ ) where N is the resolution. Autocalibration overrides the device editor settings.

When Autocalibration is disabled, the raw count value depends on  $I_{DAC}$  range,  $I_{DAC}$  value, resolution, sensor capacitance, IMO frequency, prescaler, precharge source, and  $V_{REF}$  parameters set in the device editor.

Autocalibration consumes ROM and RAM resources and increases start time. Autocalibration does not automatically select the  $I_{DAC}$  range value. If the raw count value after calibration is less than half of the resolution range, you should increase the  $I_{DAC}$  range or reduce the precharge frequency. Autocalibration works to improve marginally functional configurations.

### 3.4.2.9 $I_{DAC}$ Range

The  $I_{DAC}$  Range parameter scales the  $I_{DAC}$  current output. For example, selecting 2x will scale the  $I_{DAC}$  output to twice the range.

Possible values are 1x, 2x, 4x, and 8x.

## 3.4.3 CSA\_EMC User Module Low-Level Parameters

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

### 3.4.3.1 Settling Time

The Settling Time parameter controls the software delay that allows the voltage on the  $C_{MOD}$  capacitor to stabilize. Each loop has nine CPU cycles per iteration. Select a settling time based on Equation 5.

$$\text{Settling Time} \geq 10 \times R_{SERIES} \times C_P \quad \text{Equation 5}$$

Where:

$R_{SERIES}$  = 400-Ω + series resistor placed between port pin and sensor (typical value 560 Ω)

$C_P$  = sensor base capacitance

Possible values are 2 to 255.

### 3.4.3.2 Freq Num

This parameter improves EMC performance by implementing a patented EMC improvement technology. Freq Num = 1 corresponds to the standard scanning algorithm and Freq Num = 3 turns on the advanced algorithm. Enabling the advanced scanning algorithm increases the scanning time and RAM usage by a factor of three.

Possible values are 1 (standard scanning algorithm) and 3 (advanced algorithm).

### 3.4.3.3 Spread Spectrum

This parameter improves EMC performance by implementing a firmware-based spread-spectrum technique that randomly changes the clock value during scanning. Spread spectrum is enabled when Freq Num is set to 1.

Possible values are 1 (enabled) and 3 (disabled).

#### 3.4.3.4 Raw Data Median Filter

The median filter looks at the three most recent samples from a sensor and reports the median value. It is used to remove short noise spikes. This filter generates a delay of one sample. This filter is generally not recommended because of the delay and RAM usage. Enabling this filter consumes (Number of Sensors × 2 × Freq Num) bytes of RAM and 100 bytes of flash. It is disabled by default.

Possible values are Enabled and Disabled.

#### 3.4.3.5 RawData IIR Filter

This infinite impulse response (IIR) filter reduces noise in the conversion result (raw count). Filtering on the raw counts can be more effective than filtering the XY coordinate, but requires more RAM. Enabling this filter consumes an additional 100 bytes of flash. It is disabled by default. The default IIR coefficient is 0.5.

Possible values are Enabled and Disabled.

#### 3.4.3.6 RawData IIR Filter Coefficient

This is the coefficient for the Raw Count IIR filter.

Possible values are 2 ( $\frac{1}{2}$  previous sample +  $\frac{1}{2}$  current sample) and 4 ( $\frac{3}{4}$  previous sample +  $\frac{1}{4}$  current sample).

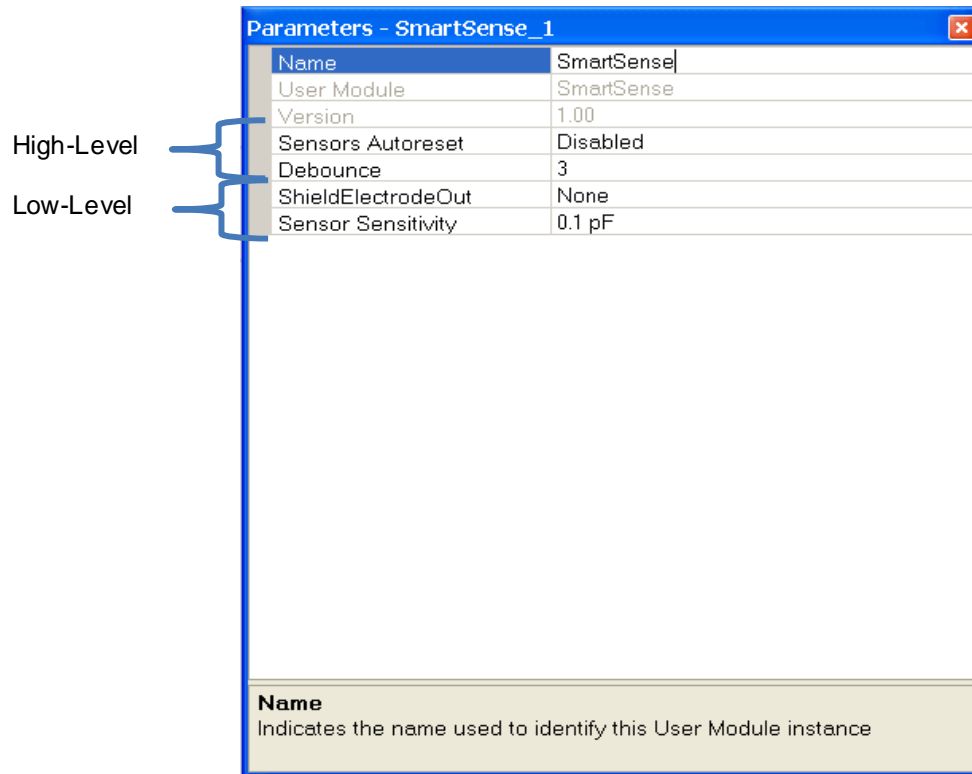
#### 3.4.3.7 Clock

The Clock parameter can be used to increase the effective resistance of the sensor. If the sensor area is large, the effective resistance may be too high for the autocalibration of the switched capacitor circuit. Large proximity sensors may encounter decreased sensitivity. In this case, the settling voltage is too far below the comparator threshold. Setting a larger divider of the internal main oscillator (IMO) increases the effective resistance, which compensates for the high capacitance.

Possible values are IMO, IMO/2, IMO/4, and IMO/8.

### 3.4.4 SmartSense User Module Parameters

Figure 3-7. PSoC Designer SmartSense Parameters



#### 3.4.4.1 Shield Electrode Out

A shield electrode is used to reduce parasitic capacitance. This parameter selects where to route the output of the shield electrode.

Possible values are P0[7] or P1[2].

#### 3.4.4.2 Sensor Sensitivity

This parameter is used to increase and decrease the sensitivity of a sensor.

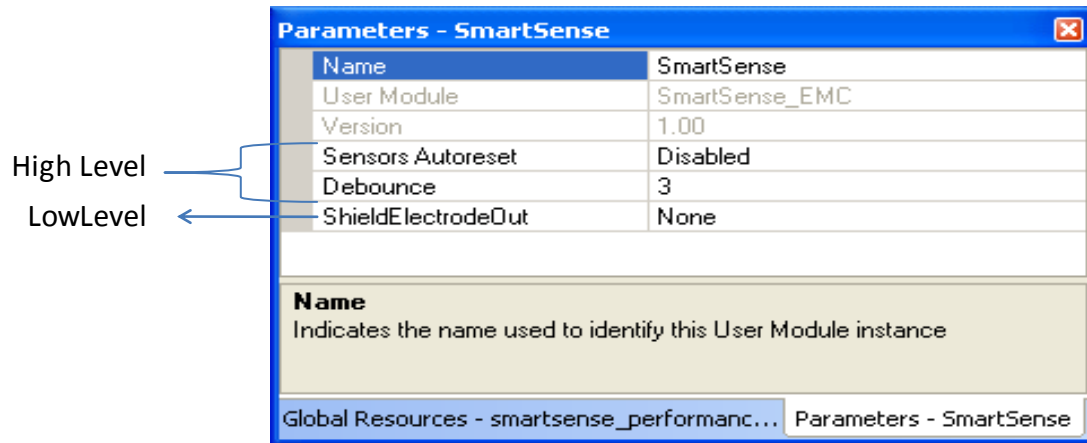
Possible values are 0.1 pF, 0.2 pF, 0.3 pF, and 0.4 pF.

#### 3.4.4.3 MultiChart for Monitoring CapSense User Module Parameters

Tuning the CapSense system requires you to monitor the CapSense User Module global arrays. The MultiChart application helps to monitor this parameter very easily. See the application note [AN2397](#) for more details on the use of MultiChart.

### 3.4.5 SmartSense\_EMC User Module Parameters

Figure 3-8. PSoC Designer SmartSense\_EMC Parameters



#### 3.4.5.1 Shield Electrode Out

A shield electrode is used to reduce parasitic capacitance. This parameter selects where to route the output of the shield electrode.

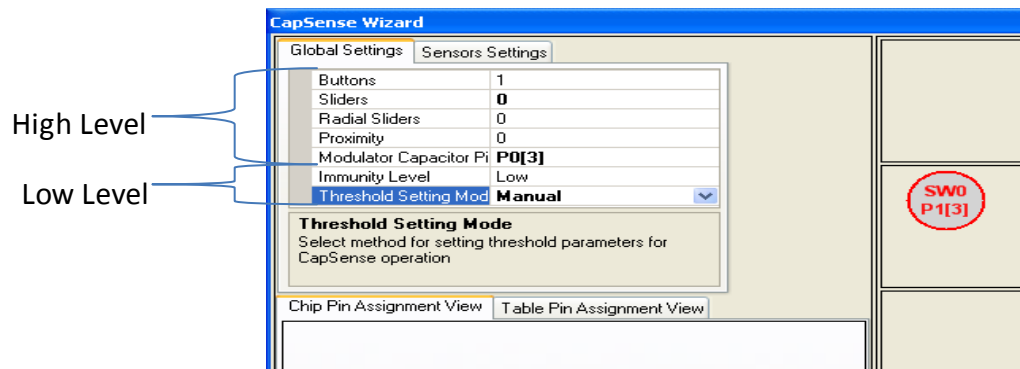
Possible values are P0[7] or P1[2].

#### 3.4.5.2 Immunity Level

This parameter defines the immunity level of the user module against the external noise. Selecting a High immunity level provides maximum immunity against the external noise. A Medium immunity level provides moderate immunity. Setting the Immunity level to Medium consumes two times the scan time and RAM memory, and setting the immunity level to High consumes three times the scan time and RAM memory for sensor implementation compared to the Low immunity mode.

Possible values are Low, Medium, and High.

Figure 3-9. PSoC Designer SmartSense\_EMC Global Setting

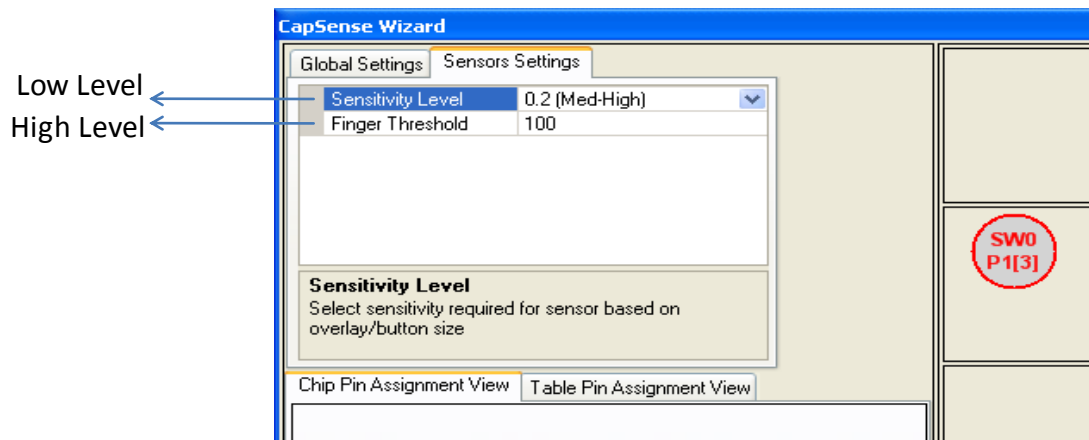


#### 3.4.5.3 Threshold Setting Mode

Selecting Manual threshold mode provides flexibility in setting the finger threshold for each sensor. Selecting Automatic threshold mode causes the SmartSense\_EMC User Module to automatically set the thresholds for each sensor. Automatic threshold mode consumes more RAM than Manual threshold mode.

Possible values are Manual and Automatic.

Figure 3-10. PSoC Designer SmartSense\_EMC Sensor Setting



#### 3.4.5.4 Sensor Sensitivity

This parameter is used to increase and decrease the sensitivity of a sensor. Possible values are 0.1 pF, 0.2 pF, 0.3 pF, and 0.4 pF.

## 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

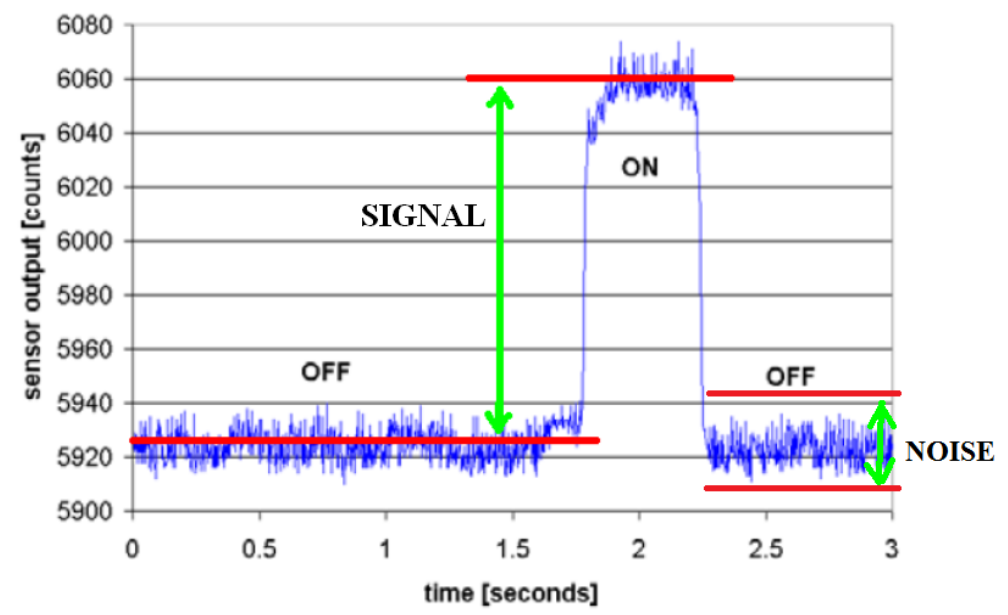
#### 4.1.1 Signal, Noise, and SNR

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 CapSense signal is compared to the CapSense noise by using a quantity called signal-to-noise ratio (SNR). Before discussing the meaning of SNR for CapSense, let us 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 a finger is placed on the sensor, the average output increases to 6060 counts. Because the CapSense signal tracks the change in counts due to the finger,  $\text{Signal} = 6060 - 5925 = 135$  counts.

Figure 4-1. CapSense Signal and Noise



#### 4.1.1.2 CapSense Noise

CapSense noise is the peak-to-peak variation in sensor response when a finger is not present, as demonstrated in [Figure 4-1](#). 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.

#### 4.1.1.3 CapSense SNR

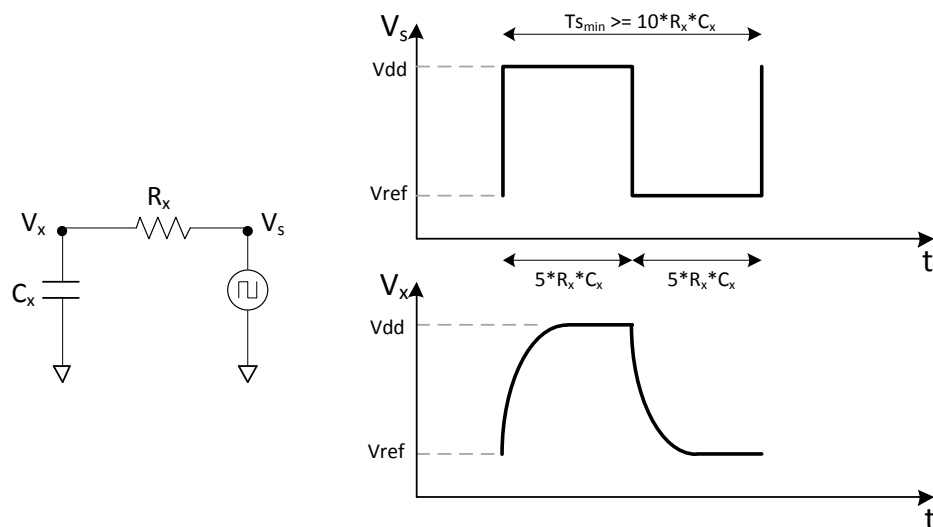
CapSense SNR is the simple ratio of signal and noise. Continuing with the example, if the signal is 135 counts and noise is 26 counts, and then SNR is 135:26, which reduces to an SNR of 5.2:1. The minimum recommended SNR for CapSense is 5:1, which means the signal is five times larger than the noise. 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 fully charged and discharged during each cycle. The charge/discharge path switches between two states at a rate set by a user module parameter called Clock in the CSA\_EMC User Module, and Precharge Clock in the CSD 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 4-2](#).

Figure 4-2. Charge/Discharge Waveforms



Set the charge/discharge rate to a level that is compatible with this RC time constant. You should 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:

$$Ts_{min} = 10 \times R_x C_x \quad \text{Equation 6}$$

$$fs_{max} = \frac{1}{10 \times R_x C_x} \quad \text{Equation 7}$$

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:

$$R_x = 1.4 \text{ k}\Omega$$

$$C_x = 24 \text{ pF}$$

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

$$Ts_{min} = 0.34 \text{ }\mu\text{s}$$

$$fs_{max} = 3 \text{ MHz}$$

### 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.

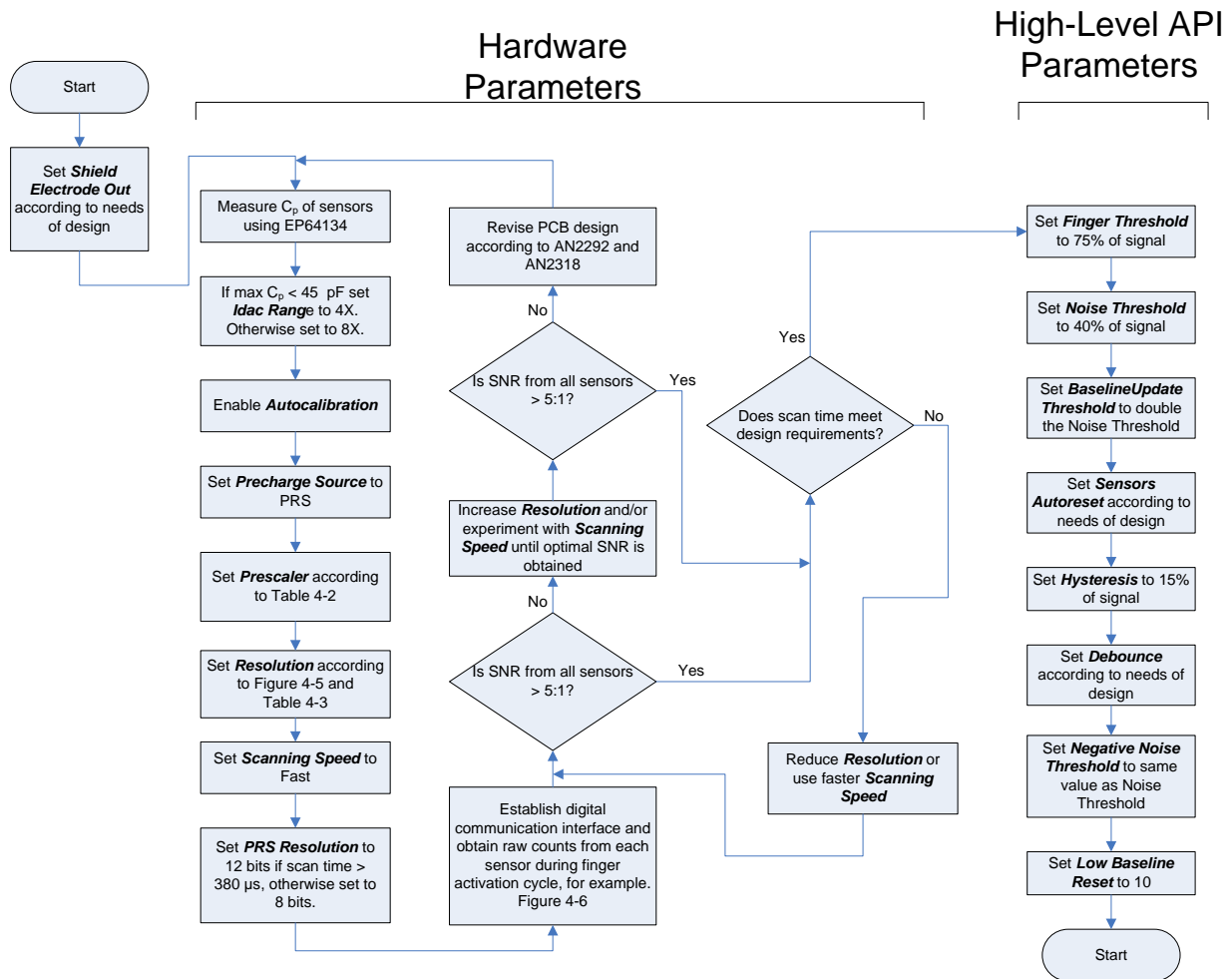
Baseline tracks the change in counts at a rate set by the Baseline Update Threshold parameter. Make sure to match the update rate to the intended application. If the update rate is too fast, the baseline will compensate for any changes introduced by a finger, and the moving finger will not be detected. 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 CSA\_EMC User Module

Figure 4-3 is a flowchart that shows the tuning process of CSA\_EMC parameters. CSA\_EMC User Module parameters can be separated into two broad categories: low-level (hardware) parameters and high-level parameters. The parameters in these categories affect the behavior of the capacitive sensing system in different ways. There is, however, a complementary relationship between the sensitivity of each sensor as determined by the hardware parameter settings and many of the high-level parameter settings. When any hardware parameter is changed, you must make sure that any corresponding high-level parameters are adjusted accordingly. Tuning of CSA\_EMC User Module parameters should always begin with the hardware parameters.



Figure 4-3. CSA\_EMC User Module Parameters Tuning Flowchart



### 4.3 Recommended $C_{INT}$ Value for CSA\_EMC

Start the tuning process with the recommended  $C_{INT}$  value of 1.2 nF. In the process of tuning, if you find that the sensor signal is not adequate to get 5:1 SNR, you can increase  $C_{INT}$ . The recommended maximum value of  $C_{INT}$  is 5.6 nF. X7R or NPO type capacitors are recommended for  $C_{INT}$  stability over temperature and capacitor should have voltage rating not less than 5 V.

### 4.4 Measuring Sensor $C_P$

The first step in the tuning procedure is to measure the sensor parasitic capacitance ( $C_P$ ). The step-by-step procedure on how to do this is as follows:

1. Set **CPU\_CLK** equal to SysClk/2.
2. Set **Clock** equal to IMO/8.
3. Set **Settling Time** equal to 255.
4. Read back the  $I_{DAC}$  code set by the algorithm for the particular sensor. The value will be stored in the array `CSA_EMC_baDACCodeBaseline[]`
5. Measure the  $I_{DAC}$  current corresponding to the  $I_{DAC}$  code.

Create a PSoC Designer project with the following code. The code routes the I<sub>DAC</sub> to port pin P1[4].

```
//configure P1[4] to HI-Z
PRT1DM0 &= ~ (1<<4);
PRT1DM1 |= (1<<4);
//connect P1[4] to analog mux bus
MUX_CR1 = (1<<4);
// set IDAC to read back IDAC Code
IDAC_D = <IDAC CODE>
// turn ON IDAC
CS_CR2 = 0xD0;
```

Place a current meter between pin P1[4] and ground, and measure current. Let I<sub>MEASURED</sub> be its value.

6. Calculate C<sub>P</sub> using the equation  $C_P = I_{MEASURED} / ((IMO/8) * 1.3)$

Sensor C<sub>P</sub> can also be measured using an LCR meter. Connect one terminal of the LCR meter to the sensor pin and the other to GND to measure C<sub>P</sub>.

## 4.5 Estimating CSA\_EMC Clock

Table 4-1 shows the recommended precharge clock frequency as a function of sensor C<sub>P</sub>. Set the CSA\_EMC clock to get the recommended precharge clock frequency. Precharge clock frequency depends on the selected IMO, CSA\_EMC clock setting, and the C<sub>P</sub> of the sensor.

Make sure that the precharge clock frequency does not go higher than the recommended value.

Table 4-1. CSA\_EMC Clock Setting Based on C<sub>P</sub> and IMO

C <sub>P</sub> (pF)	CSA_EMC Clock		
	IMO = 24 MHz	IMO = 12 MHz	IMO = 6 MHz
< 5	IMO/2	IMO	IMO
5 to 10	IMO/4	IMO/2	IMO
10 to 15	IMO/4	IMO/4	IMO/2
15 to 20	IMO/4	IMO/4	IMO/2
20 to 25	IMO/16	IMO/8	IMO/4
25 to 30	IMO/16	IMO/8	IMO/4
30 to 35	IMO/16	IMO/8	IMO/4
35 to 40	IMO/16	IMO/8	IMO/4
40 to 45	IMO/16	IMO/8	IMO/4
45 to 50	IMO/16	IMO/8	IMO/8

## 4.6 Setting Settling Time

Minimum value for the settling time parameter is estimated using Equation 8.

$$\text{Settling Time} = \frac{(5 \cdot C_{int})}{\left( \text{Clock} \cdot C_P \cdot 25 \cdot \left( \frac{1}{F_{cpu}} \right) \right)} \quad \text{Equation 8}$$

Where:

- C<sub>INT</sub> = Value of integration capacitor
- Clock = Precharge clock frequency (CSA\_EMC Clock)
- C<sub>P</sub> = Sensor parasitic capacitance value
- F<sub>CPU</sub> = CPU clock frequency

## 4.7 Monitoring CapSense Data

See [CapSense Data Viewing Tools](#).

## 4.8 Methods to Increase SNR

This section explains the methods available to increase SNR.

### 4.8.1 Reduce Noise

One way to increase SNR is to reduce the noise counts. You can use one of the following options to achieve this:

- Using software filters – see [Software Filtering](#) for details.
- Enabling Spread Spectrum – see [Spread Spectrum](#) for details.
- Increasing Immunity Level – see [Freq Num](#) for details.

### 4.8.2 Increase Signal

Improve SNR by increasing the signal, in one of two ways:

- Increase the value defined by macro `CSA_EMC_BASELINE`. This macro is located in file `CSA_EMC.inc`. By default, the macro is assigned a value of 0x0800
- Increase the value of `C_INT` capacitor

## 4.9 Tuning the CSD User Module

[Figure 4-4](#) is a flowchart showing the tuning process for CSD UM parameters. CSD UM parameters can be separated into two broad categories: low-level (hardware) parameters and high-level API parameters. The parameters in these categories affect the behavior of the capacitive sensing system in different ways. However, there is a complementary relationship between the sensitivity of each sensor as determined by the hardware parameter settings and many of the high-level parameter settings. You must consider this fact when you change any hardware parameter to make sure that the corresponding high-level parameters are adjusted accordingly. Tuning CSD User Module parameters should always begin with the hardware parameters.

```

graph TD
    subgraph Hardware_Parameters [Hardware Parameters]
        Start1([Start]) --> SetShield[Set Shield Electrode Out according to needs of design]
        SetShield --> MeasureCp[Measure Cp of sensors using EP64134]
        MeasureCp --> IdacRange{If max Cp < 45 pF set Idac Range to 4X. Otherwise set to 8X.}
        IdacRange --> EnableAuto[Enable Autocalibration]
        EnableAuto --> SetPrecharge[Set Precharge Source to PRS]
        SetPrecharge --> SetPrescaler[Set Prescaler according to Table 4-2]
        SetPrescaler --> SetResolution[Set Resolution according to Figure 4-5 and Table 4-3]
        SetResolution --> SetScanningSpeed[Set Scanning Speed to Fast]
        SetScanningSpeed --> SetPRSResolution[Set PRS Resolution to 12 bits if scan time > 380 μs, otherwise set to 8 bits.]
        SetPRSResolution --> IsSNR1{Is SNR from all sensors > 5:1?}
        IsSNR1 -- No --> RevisePCB[Revise PCB design according to AN2292 and AN2318]
        RevisePCB --> MeasureCp
        IsSNR1 -- Yes --> IncreaseRes[Increase Resolution and/or experiment with Scanning Speed until optimal SNR is obtained]
        IncreaseRes --> IsSNR2{Is SNR from all sensors > 5:1?}
        IsSNR2 -- No --> IncreaseRes
        IsSNR2 -- Yes --> EstablishInterface[Establish digital communication interface and obtain raw counts from each sensor during finger activation cycle, for example. Figure 4-6]
        EstablishInterface --> DoesScanTimeMeet{Does scan time meet design requirements?}
        DoesScanTimeMeet -- Yes --> SetFingerThreshold
        DoesScanTimeMeet -- No --> ReduceRes[Reduce Resolution or use faster Scanning Speed]
        ReduceRes --> IsSNR2
    end

    subgraph HighLevelAPI_Parameters [High-Level API Parameters]
        Start2([Start]) --> SetFingerThreshold[Set Finger Threshold to 75% of signal]
        SetFingerThreshold --> SetNoiseThreshold[Set Noise Threshold to 40% of signal]
        SetNoiseThreshold --> SetBaselineUpdateThreshold[Set BaselineUpdate Threshold to double the Noise Threshold]
        SetBaselineUpdateThreshold --> SetSensorsAutoreset[Set Sensors Autoreset according to needs of design]
        SetSensorsAutoreset --> SetHysteresis[Set Hysteresis to 15% of signal]
        SetHysteresis --> SetDebounce[Set Debounce according to needs of design]
        SetDebounce --> SetNegativeNoiseThreshold[Set Negative Noise Threshold to same value as Noise Threshold]
        SetNegativeNoiseThreshold --> SetLowBaselineReset[Set Low Baseline Reset to 10]
        SetLowBaselineReset --> Start2
    end

```

The flowchart is divided into two main sections: **Hardware Parameters** and **High-Level API Parameters**.

**Hardware Parameters:**

- Start
- Set **Shield Electrode Out** according to needs of design
- Measure  $C_p$  of sensors using EP64134
- If max  $C_p$  < 45 pF set **Idac Range** to 4X. Otherwise set to 8X.
- Enable **Autocalibration**
- Set **Precharge Source** to PRS
- Set **Prescaler** according to Table 4-2
- Set **Resolution** according to Figure 4-5 and Table 4-3
- Set **Scanning Speed** to Fast
- Set **PRS Resolution** to 12 bits if scan time > 380  $\mu$ s, otherwise set to 8 bits.
- Is SNR from all sensors > 5:1?
  - No: Revise PCB design according to AN2292 and AN2318 (loop back to Measure  $C_p$ )
  - Yes: Increase **Resolution** and/or experiment with **Scanning Speed** until optimal SNR is obtained
- Is SNR from all sensors > 5:1?
  - No: Increase **Resolution** and/or experiment with **Scanning Speed** until optimal SNR is obtained
  - Yes: Establish digital communication interface and obtain raw counts from each sensor during finger activation cycle, for example. Figure 4-6
- Does scan time meet design requirements?
  - Yes: Set **Finger Threshold** to 75% of signal
  - No: Reduce **Resolution** or use faster **Scanning Speed** (loop back to Is SNR from all sensors > 5:1?)

**High-Level API Parameters:**

- Start
- Set **Finger Threshold** to 75% of signal
- Set **Noise Threshold** to 40% of signal
- Set **BaselineUpdate Threshold** to double the Noise Threshold
- Set **Sensors Autoreset** according to needs of design
- Set **Hysteresis** to 15% of signal
- Set **Debounce** according to needs of design
- Set **Negative Noise Threshold** to same value as Noise Threshold
- Set **Low Baseline Reset** to 10
- Start

By default, hardware parameters are global settings that apply to all CapSense sensors in a design. In designs where total parasitic capacitance of each sensor ( $C_P$ ), sensor sensitivity, or both, vary over a wide range, there may not be global hardware parameter settings that are suitable for all sensors. In such cases, the respective hardware parameters for each sensor can be set by calling the `SetIadcValue()`, `SetPrescaler()`, and `SetScanMode()` API functions before calling the `ScanSensor()` API function.

#### 4.9.1 Recommended $C_{MOD}$ Value for CSD

AN65973 - CY8C20xx6 A/H/AS CapSense® Design Guide, Doc. No. 001-65973 Rev. \*J

## 4.9.2 ShieldElectrodeOut

Enable the ShieldElectrodeOut for this design.

## 4.9.3 I<sub>DAC</sub> Range

For projects where the maximum sensor C<sub>P</sub> is less than 45 pF, use 4X; otherwise, use 8X.

## 4.9.4 Autocalibration

Autocalibration should always be set to Enabled in CY8C20xx6A CSD designs. The autocalibration algorithm can successfully set the I<sub>DAC</sub> if the prescaler is set properly and C<sub>MOD</sub> is the recommended size.

## 4.9.5 I<sub>DAC</sub> Value

This parameter determines the current output of I<sub>DAC</sub> when autocalibration is disabled. When autocalibration is enabled, as recommended, this parameter is overridden and has no effect. When autocalibration is disabled, raising this parameter lowers the raw count baseline and vice versa.

## 4.9.6 Precharge Source

This parameter selects the sensor switching clock source. The available options are Prescaler, which uses the IMO through a divider, or PRS, which passes the divided IMO clock through a pseudo random generator, providing a spread-spectrum clock. PRS provides superior noise immunity and lower noise emissions and is therefore the recommend default setting for Precharge Source. In some instances, the prescaler precharge source can provide higher SNR. However, when using copper circuitry, this SNR improvement is usually marginal and rarely justifies foregoing the benefits of PRS.

## 4.9.7 Prescaler

Prescaler is the divider applied to the IMO to develop the precharge clock. This is the most critical hardware UM parameter for properly tuning a CSD design. Prescaler depends on the selected precharge source, IMO, and the C<sub>P</sub> of the sensors being scanned. [Table 4-2](#) gives recommended prescaler settings based on these parameters.

Table 4-2. Prescaler Setting Based on Precharge Source, IMO, and C<sub>P</sub>

C <sub>P</sub> (pF)	Precharge Source = PRS			Precharge Source = Prescaler		
	Prescaler IMO = 24 MHz	Prescaler IMO = 12 MHz	Prescaler IMO = 6 MHz	Prescaler IMO = 24 MHz	Prescaler IMO = 12 MHz	Prescaler IMO = 6 MHz
<6	1	Note 1	Note 1	2	1	1
7–11	2	1	Note 1	4	2	1
12–15	2	1	Note 1	4	2	1
16–19	4	2	1	8	4	2
20–22	4	2	1	8	4	2
23–26	4	2	1	8	4	2
27–30	4	2	1	8	4	2
31–34	4	2	1	8	4	2
35–37	8	4	2	16	8	4
38–41	8	4	2	16	8	4
42–45	8	4	2	16	8	4
46–49	8	4	2	16	8	4
50–52	8	4	2	16	8	4
53–56	8	4	2	16	8	4
57–60	8	4	2	16	8	4

**Note 1** This combination of Precharge Source, Prescaler, and C<sub>P</sub> is not recommended.

### 4.9.8 Resolution

Available choices are 9 to 16 bits. Raising the resolution raises sensitivity, SNR, and noise immunity at the expense of scan time. The maximum raw count (full scale range) for scanning resolution  $n$  is  $2^n - 1$ . Table 4-3 gives recommended resolution settings based on  $C_P$  and the finger capacitance  $C_F$ .  $C_F$  is the change in capacitance of a sensor when a finger is placed on the sensor.  $C_F$  depends on overlay thickness, sensor size, and proximity of the sensor to other large conductors. Figure 4-5 gives  $C_F$  values as a function of overlay thickness and circular sensor diameter.

Figure 4-5. Finger Capacitance ( $C_F$ ) Based on Overlay Thickness and Circular Sensor Diameter

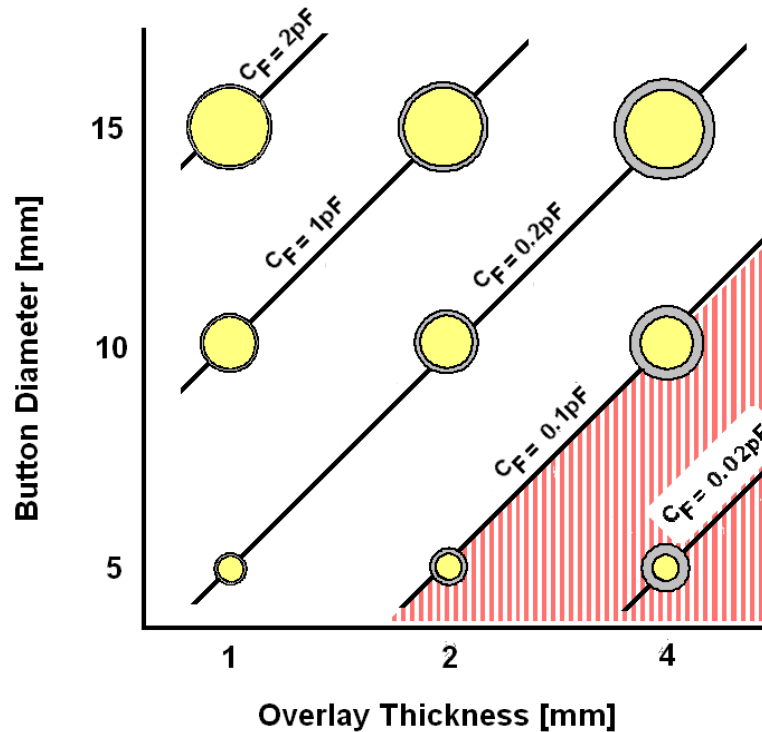


Table 4-3. Resolution Setting Based on Finger Capacitance and  $C_P$

$C_P$ (pF)	$C_F = 0.1\text{pF}$	$C_F = 0.2\text{ pF}$	$C_F = 0.4\text{pF}$	$C_F = 0.8\text{pF}$
<6	12	11	10	9
7–12	13	12	11	10
13–24	14	13	12	11
25–48	15	14	13	12
>49	16	15	14	13

### 4.9.9 Scanning Speed

This parameter controls the integration time for each LSB of the scan result. The choices are Ultra Fast, Fast, Normal, and Slow. Fast is generally a good starting point. In some, but not all, cases slower scanning speed can yield higher SNR at the expense of longer scan time and more power consumption. Table 4-4 shows the actual scan time in microseconds for a single sensor based on resolution and scanning speed.

Table 4-4. Scan Time for a Single Sensor in  $\mu$ s Based on Resolution and Scanning Speed

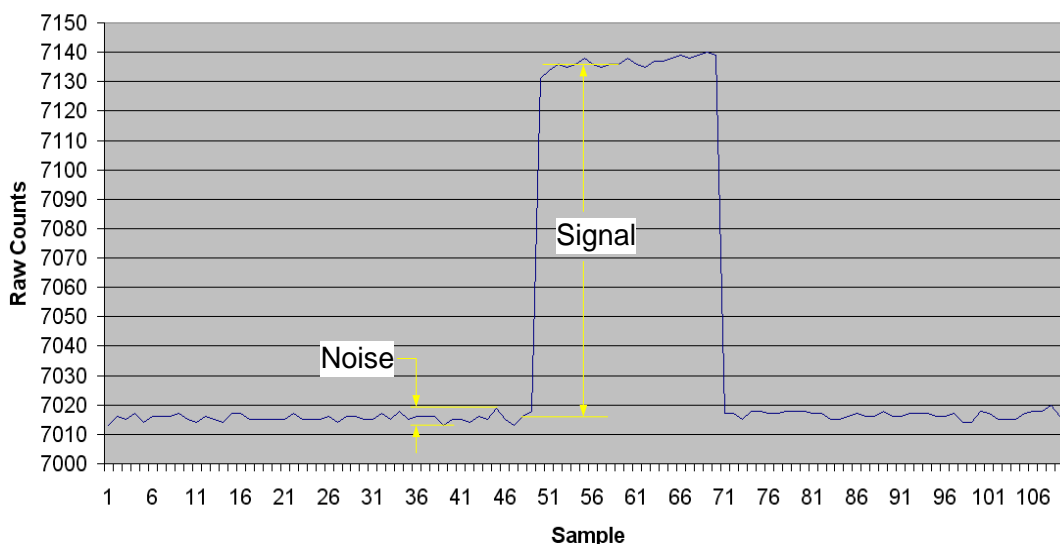
Resolution (bits)	Scanning Speed			
	Ultra Fast	Fast	Normal	Slow
9	57	78	125	205
10	78	125	205	380
11	125	205	380	720
12	205	380	720	1400
13	380	720	1400	2800
14	720	1400	2800	5600
15	1400	2800	5600	11000
16	2800	5600	11000	22000

#### 4.9.10 High-Level API Parameters

High-level API parameters determine the behavior of high-level firmware algorithms that discriminate between sensor activations and noise, and compensate for signal drift caused by environmental conditions. To determine proper values for these parameters, you must establish a digital communication interface with the system to monitor raw counts, baseline, and difference counts during a finger activation event for each sensor. This data is stored in arrays named `CSD_waSnsBaseline[]`, `CSD_waSnsResult[]`, and `CSD_waSnsDiff[]`, respectively. The high-level API parameter settings are based primarily on ambient noise and finger signal strength, as indicated by this data. Noise and signal strength depend on EMI environment, PCB layout, overlay thickness, and other physical characteristics of the system. Therefore, the data used as the basis for setting these parameters must be taken in its original position with the system in its final assembled state and in the same EMI environment as will exist in use.

Figure 4-6 shows the typical raw counts obtained from a sensor during a finger activation cycle; that is, the sensor is activated and then deactivated. Labels are superimposed over the data that indicate how noise and signal are to be calculated based on the raw data. Where appropriate, the high-level parameter descriptions that follow include information about how to set each parameter based on these noise and signal values. According to CapSense design best practice, the ratio of signal-to-noise (SNR) must be at least 5:1 for robust CapSense system operation. If SNR is less than 5:1, the hardware parameters must be adjusted, the PCB layout changed according to the guidelines of [Getting Started with CapSense](#) to raise SNR to at least 5:1, or both.

Figure 4-6. Typical Raw Counts from a Sensor during Finger Activation Cycle



### 4.9.11 Set High-Level Parameters

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

- **Finger Threshold:** Set to 75 percent of raw counts with sensor ON
- **Noise Threshold:** Set to 40 percent of raw counts with sensor OFF
- **Negative Noise Threshold:** Set equal to (Noise Threshold/2)
- **Baseline Update Threshold:** Set to two times Noise Threshold
- **Hysteresis:** Set to 15 percent of raw counts with sensor ON
- **Low Baseline Reset:** Set to 50
- **Sensors Autoreset:** Based on design requirements
- **Debounce:** Based on design requirements

## 4.10 Using the SmartSense User Module

SmartSense allows you to create a CapSense design that requires no tuning, as long as the sensor parasitic capacitance is in the range from 5 pF to 45 pF with a minimum 0.1-pF finger touch. You can create a SmartSense design by using the SmartSense User Module in PSoC Designer 5.1. This section also shows you how to migrate an existing CSD CapSense design to SmartSense.

### 4.10.1 Guidelines for SmartSense

Follow these guidelines when using the SmartSense User Module in an application:

- SmartSense requires that the capacitive user interface design follows the layout and system design best practices documented in the previous sections of this design guide.
- All of the CSD User Module parameters (such as  $I_{DAC}$  value, prescaler period, clock divider, scan speed, resolution) are determined at runtime by the SmartSense User Module. You should not use APIs that modify these CSD parameters in firmware, unless you know exactly what effect it has in your design.
- To migrate an existing design from CSD to SmartSense,
  - ☐ Ensure that all APIs that set or modify the CSD parameters are first removed from the program.
  - ☐ Ensure that the parasitic capacitance of all CapSense sensors in the design is between 5 pF and 45 pF over environmental and PCB production process variations.
  - ☐ Make sure recommended  $C_{MOD}$  capacitor (X7R, 2.2-nF, voltage rating more than 5 V) is connected to the  $C_{MOD}$  port pin selected in the user module wizard.

### 4.10.2 Understanding the Difference

The differences between the SmartSense User Module and the standard CSD User Module are:

- The SmartSense User Module supports the same APIs that a standard CSD User Module supports. Thus, no change is required in placing, configuring, starting, or calling other APIs except the user module instance name.
- There is no need to set any user module parameters for tuning, as all the parameters related to tuning are automatically set at runtime by the SmartSense User Module.
- The  $C_{MOD}$  capacitor value is restricted to 2.2 nF. Use of an X7R capacitor with a voltage rating higher than 5 V is recommended in all CapSense applications.
- The SmartSense algorithm maintains the signal SNR of each sensor between 5:1 and 11:1 to ensure robust CapSense operation while maximizing performance.
- The scanning time of the SmartSense User Module is restricted by the algorithm to be between 410  $\mu$ s and 2.8 ms per sensor in 24-MHz operating mode, based on the parasitic capacitance of the sensor.



### 4.10.3 Recommended $C_{MOD}$ Value for SmartSense

The recommended  $C_{MOD}$  value for a SmartSense-based design is 2.2 nF. X7R or NPO type capacitors are recommended for  $C_{INT}$  stability over temperature. The capacitor should have voltage rating not less than 5 V.

### 4.10.4 SmartSense User Module Parameters

Only four parameters must be set for this user module. These are:

- Sensors Autoreset
- Debounce
- Modulator Capacitor Pin
- Sensitivity Level

#### 4.10.4.1 Sensors Autoreset

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

#### 4.10.4.2 Debounce

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

#### 4.10.4.3 Modulator Capacitor Pin

This parameter selects the pin to which the 2.2 nF/X7R/voltage rating more than 5 V  $C_{MOD}$  capacitor is connected. The available pins are P0[1] and P0[3].

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

#### 4.10.4.4 Sensitivity Level

Sensitivity is used to increase or decrease strength of signal from a sensor. A lower value for sensitivity (0.1 pF) leads to a stronger signal from the sensor. Designs with thicker overlays require stronger signals from sensors for proper implementation. The available options for sensitivity selection are High (0.1 pF), Medium High (0.2 pF), Medium Low (0.3 pF) and Low (0.4 pF).

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

Tuning best practice is to find the highest sensitivity value for the sensor to produce the required 5:1 SNR. You may start the tuning with the highest sensitivity value (0.4 pF) and reduce the value as required to meet the 5:1 SNR

### 4.10.5 SmartSense\_EMC User Module Specific Guidelines

All guidelines applicable to the SmartSense User Module apply to the SmartSense\_EMC User Module. For general guidelines about CapSense design and SmartSense-based design, see the [CapSense Getting Started Guide](#). This section documents a few important aspects of the SmartSense\_EMC User Module.

#### 4.10.5.1 Sensor Scan Time, Response Time, and Memory Utilization

When a sensor is implemented using the SmartSense\_EMC User Module, the scan time of a sensor, response time of the sensor, and RAM memory usage depends on the immunity mode selected in the user module.

- With immunity mode Medium, sensor scan time is two times higher than a sensor with immunity mode Low. With immunity mode High, the scan time of a sensor is three times higher than the scan time of a sensor with immunity mode Low.
- Increase in scan time proportionally increases the response time of a sensor. With immunity mode Medium, response time is two times higher than that of a sensor with immunity mode Low. Similarly, response time of a sensor with immunity mode High is three times higher than that of a sensor with immunity mode Low.

- To implement a robust electromagnetic compliant algorithm, the SmartSense\_EMC User Module uses RAM memory. As a result, the highest immunity mode (High) needs approximately three times the RAM memory used in immunity mode Low. Immunity mode Medium uses only about two times more RAM memory than that of immunity mode Low.

#### 4.10.5.2 IMO Tolerance and Time Critical Task

IMO tolerance for SmartSense\_EMC-enabled parts is +5% and –20%.

- When implementing time-critical algorithms and logic, you must consider IMO tolerance to make sure that the firmware logic or algorithm does not break.
- If a project uses interrupts, you need to consider IMO tolerance while analyzing interrupt latency, ISR execution time, and so on.
- Every timing analysis that depends on the IMO (for example, a Timer clocked by IMO, delay created using loop in firmware, API execution time) must take into account the IMO tolerance to ensure robust application firmware.

#### 4.10.5.3 I<sup>2</sup>C Operating Speed

I<sup>2</sup>C interface operation frequency is limited to a maximum of 80 percent of the actual operating frequency of the user module in the SmartSense\_EMC-enabled parts. This limitation is caused by the 20-percent IMO tolerance.

- This means, when a clock speed of 400 kHz is selected in the I<sup>2</sup>C User Module, the I<sup>2</sup>C interface can be operated to maximum of 320 kHz. Similarly, operating frequency is limited to a maximum of 80 kHz and 40 kHz when 100-kHz and 50-kHz clock modes, respectively, are selected in the I<sup>2</sup>C User Module.
- While using the I<sup>2</sup>C slave interface, the master clock should operate within the reduced specification mentioned earlier. Not doing this will lead to data corruption, I<sup>2</sup>C bus conjunction, or inconsistent behavior from the I<sup>2</sup>C User Module.
- Using the I<sup>2</sup>C master module impacts only the throughput of the interface.

#### 4.10.6 Scan Time of a CapSense Sensor

To maintain the consistent finger response sensitivity over a wide range of parasitic capacitance, the SmartSense User Module automatically determines the hardware parameters of the user module. As a result of this, sensor scan time does not remain constant. For a design in mass production, it can vary based on the parasitic capacitance variation of the PCB. The total scan time of a sensor is decided by four factors. They are parasitic capacitance of sensor, IMO frequency, CPU operating frequency, and sensitivity level of the SmartSense User Module.

Scan time of a sensor can be found using Equation 9 and the following tables.

$$\text{Scan time} = \text{Sampling time (ST)} + \text{Processing time (PT)} \quad \text{Equation 9}$$

The following tables show the sampling time value with various IMO and sensitivity levels.

Table 4-5. Sampling Time for a Sensor with IMO = 24 MHz

Sensitivity = 0.2 pF		Sensitivity = 0.3 pF		Sensitivity = 0.4 pF	
C <sub>P</sub> (pF)	ST (μs)	C <sub>P</sub> (pF)	ST (μs)	C <sub>P</sub> (pF)	ST (μs)
8 to 10	340	8 to 17	340	8 to 10	170
10 to 23	680	17 to 35	680	10 to 23	340
23 to 41	1360	35 to 41	1360	23 to 41	680
41 to 45	2730	41 to 45	2730	41 to 45	1360

Table 4-6. Sampling Time for a Sensor with IMO = 12 MHz

Sensitivity = 0.2 pF		Sensitivity = 0.3 pF		Sensitivity = 0.4 pF	
C <sub>P</sub> (pF)	ST (μs)	C <sub>P</sub> (pF)	ST (μs)	C <sub>P</sub> (pF)	ST (μs)
8 to 10	680	8 to 17	680	8 to 10	340
10 to 23	1360	17 to 35	1360	10 to 23	680
23 to 41	2730	35 to 41	2730	23 to 41	1360
41 to 45	5460	41 to 45	5460	41 to 45	2730

Table 4-7. Sampling Time for a Sensor with IMO = 6 MHz

Sensitivity = 0.2 pF		Sensitivity = 0.3 pF		Sensitivity = 0.4 pF	
C <sub>P</sub> (pF)	ST (μs)	C <sub>P</sub> (pF)	ST (μs)	C <sub>P</sub> (pF)	ST (μs)
8 to 11	680	8 to 10	680	8 to 11	680
11 to 23	1360	10 to 17	1360	11 to 23	1360
23 to 42	2730	17 to 35	2730	23 to 41	2730
42 to 45	5460	35 to 41	5460	41 to 45	5460
		41 to 45	10920		

Table 4-8 shows the value for processing time with various CPU frequencies.

Table 4-8. Processing Time for a Sensor

CPU CLK	Processing Time (PT) in μs
24	71
12	142
6	284
3	568

For example, if a CapSense system is designed with a 24-MHz IMO frequency, a 6-MHz CPU clock (IMO/4), and a SmartSense sensitivity level of 0.3 pF, the scan time of the sensor that has parasitic capacitance around 15 pF can be calculated from the previous tables using Equation 9.

Sampling time for the previously mentioned configuration (24 MHz of IMO, 0.3 pF of sensitivity) is chosen from Table 4-5; it is 680 μs. Processing time for the previously mentioned configuration (CPU clock of 6 MHz) is chosen from Table 4-8; it is 284 μs.

Thus, the total scan time in this configuration is 680 + 284 = 964 μs. Scan time for more than one sensor is the sum of the scan time of each sensor.

#### 4.10.7 SmartSense Response Time

Consider the following application with standard CSD along with typical CapSense scanning firmware.

- Three CapSense sensors with parasitic capacitance of sensor between 5 pF and 10 pF
- IMO of 12 MHz and CPU clock of 12 MHz
- Sensor sensitivity level of 0.4 pF
- Debounce = 3

According to the previous tables, scanning of each sensor requires 482  $\mu$ s and three sensors have a scan time of 1.45 ms. The following firmware example requires 1 ms for additional firmware execution; thus, the loop execution time is 2.45 ms.

```
while (1)
{
    SmartSense_ScanAllSensors();
    SmartSense_UpdateAllBaselines();

    if(SmartSense_bIsAnySensorActive() )
    {
        //lms firmware routines
    }
}
```

This means that, when a CapSense sensor is activated, firmware produces the sensor ON status within 7.35 ms (the sensor should be active for Debounce number of consecutive scans). This is often referred to as the response time of a CapSense system.

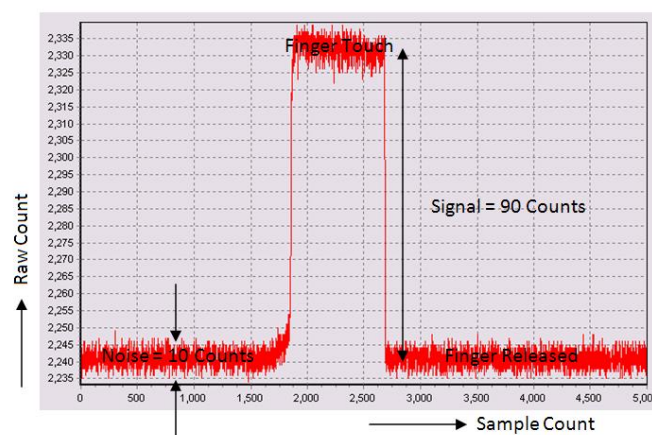
If the scan time varies with respect to the parasitic capacitance to maintain consistent, what is the impact on response time if the parasitic capacitance of the sensor changes because of the process variation? Response time may be increased (slow response) in this case. This can have a negative impact on sensor performance. Guidelines to build a robust firmware design are provided in the next section.

#### 4.10.8 Method to Ensure Minimum SNR Using the SmartSense EMC User Module

SmartSense EMC is an advanced electromagnetic compliance design of a CSD-based SmartSense User Module that does not require a tedious tuning process. However there are two simple steps to ensure robustness of the design while using the SmartSense EMC UM.

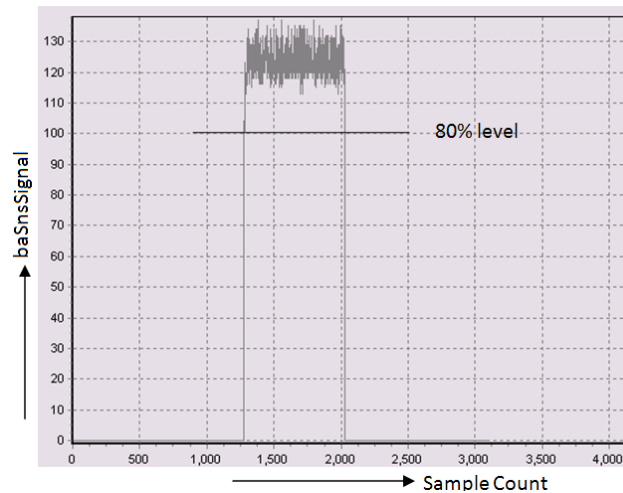
1. Set up a real-time monitoring tool to monitor CapSense User Module parameters to measure the sensor signals. The sensor raw count (SmartSense EMC\_waSnsResult), sensor normalized signal (SmartSense EMC\_baSnsSignal), and sensor finger threshold (SmartSense EMC\_baBtnFThreshold) must be observed during the tuning process. Do not use the LCD or any other numerical display to monitor data because they are slow and do not allow visualizing the data dynamics. Recommended data monitoring tools are multi-chart or the I<sup>2</sup>C USB Bridge Control panel.
2. Set the sensitivity level to 0.4 pF (Low), and calculate the SNR. [Figure 4-7](#) shows a typical raw count graph with a finger touch. According to CapSense best practices, SNR for a robust design should be greater than 5:1. If measured SNR is more than 10:1, reduce sensitivity level value to the next possible step until SNR is more than 5:1 and less than 10:1.

Figure 4-7. Raw Count Graph for a Typical Sensor with a Finger Touch



3. If you are using automatic finger threshold in the design, the process is complete with completion of the previous step. If you are using flexible finger threshold, you should also set finger threshold to complete the process. To set finger threshold, monitor the sensor signal (SmartSense EMC\_baSnsSignal) and the set finger threshold value to 80 percent of the sensor signal value when the sensor is touched. This completes the process. [Figure 4-8](#) shows a typical sensor signal and finger threshold value.

Figure 4-8. Sensor Signal for a Typical Sensor with a Finger Touch



#### 4.10.9 Firmware Design Guidelines

The response time of the CapSense sensors may change due to the increased parasitic capacitance of the sensor. It is also important to watch the loop execution time (see the following example code), which may also increase. When the parasitic capacitance of all sensors is less than 10 pF, the firmware routine is executed at a rate of 2.45 ms. This rate will change if the sensor scan time is increased because of the increase in the parasitic capacitance of the sensor based on the process variation.

The following is example code for toggling a port pin based on the main loop execution time.

```
while (1)
{
    SmartSense_ScanAllSensors();
    SmartSense_UpdateAllBaselines();

    if(SmartSense_bIsAnySensorActive() )
    {
        //1ms firmware routines
    }

    PRT0DR_Shadow ^= 0x01;
    PRT0DR = PRT0DR_Shadow;
}
```

The period of the signal on the Port\_0[1] pin is 4.9 ms (the period is twice the loop time as the port pin is toggled). If the parasitic capacitance of one sensor is increased to approximately 15 pF, the scan time will change to 1.78 ms; thus, the period of the signal on Port\_0[1] will be 5.6 ms.

If the parasitic capacitance of the sensor is close to the boundary of the SmartSense capacitance banks (for example, 9 pF, which is very close to the 10-pF boundary), SmartSense may choose a neighboring scan time in an application because of process variation. Because of this, different production parts of the same design can have two different main loop execution times and response times.

Based on the above discussions, the firmware should not rely on the scan time of the sensor for implementing other features (such as, software PWM, software delay, and so on). Programs implementing a watchdog timer (WDT) should consider this fact while setting the WDT expiration time

A simple firmware implementation example to get a consistent main loop execution time using the Timer16 User Module follows.

```
// Main program
BYTE bTimerTicks = 0;

#pragma interrupt_handler myTimer_ISR_Handler;
void myTimer_ISR_Handler( void );
```

```

void main()
{
    M8C_EnableGInt;

    SmartSense_Start();
    SmartSense_ScanAllSensors();
    SmartSense_SetDefaultFingerThresholds() ;

    Timer16_EnableInt();
    Timer16_SetPeriod (TIMEOUT_10MS) ;
    Timer16_Start();

    while( 1 )
    {
        /* Scan all 3 sensors and update
        Baseline */
        SmartSense_ScanAllSensors();
        SmartSense_UpdateAllBaselines();

        /* Wait till timer expires or
        sleep here */

        while (bTimerTicks != 1) ;
        bTimerTicks = 0 ;

        if(CSDAUTO_bIsAnySensorActive() )
        {
            //lms firmware routines
        }

        // Toggle Port_0[1]
        PRT0DR_Shadow ^= 0x01 ;
        PRT0DR = PRT0DR_Shadow ;
    }
}

// Timer16 ISR program
void myTimer_ISR_Handler(void)
{
    bTimerTicks++;
}

```

In the previous example, the program waits for the Timer to expire even if the sensor scanning is complete. The period of the Timer should be chosen based on the worst-case main loop execution time. This is the sum of the worst-case scan times of the individual CapSense sensors. If the parasitic capacitance of the sensor is close to the boundary of the SmartSense capacitance bank, choose higher scan time (using [Table 4-6](#)) for the calculation.

The SmartSense User Module enables you to easily implement the capacitive touch-sensing user interface into a system. It removes the difficulties of the tuning process and also helps to increase the yield in production against manufacturing process variations of the PCB, and other variations. Therefore, the preferred option is to migrate the existing CSD-based CapSense designs to SmartSense and to use SmartSense for new designs.

The main loop execution time and scan time of SmartSense vary based on the process variations. Though it does not affect the performance of CapSense in any way, the firmware developer should consider this when implementing CapSense PLUS applications with SmartSense Auto-Tuning technology.

## 5. Design Considerations



When designing capacitive touch-sense technology into your application, it is crucial to keep in mind that the CapSense device exists within a larger framework. Careful attention to every level of detail from PCB layout to user interface to end-use operating environment will lead to robust and reliable system performance. For more in-depth information, see the [Getting Started with CapSense](#).

### 5.1 Overlay Selection

In [CapSense Fundamentals](#), Equation 1 is presented for finger capacitance

$$C_F = \frac{\epsilon_0 \epsilon_r A}{D}$$

Where:

$\epsilon_0$  = Free space permittivity

$\epsilon_r$  = Dielectric constant of overlay

A = Area of finger and sensor pad overlap (mm<sup>2</sup>)

D = Overlay thickness (mm)

To increase CapSense signal strength, choose an overlay material with a 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
Polyimide 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 by-product 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<sup>®</sup> 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 close 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  $\Omega$ .

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				
Littlefuse	SP723	5 pF	2 nA	8 kV	15 kV
Vishay	VBUS05L1-DD1	0.3 pF	0.1 $\mu$ A <	$\pm$ 15 kV	$\pm$ 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 or core. The interference enters the PSoC chip at the PCB level, through CapSense sensor traces and any other digital or analog inputs. Layout guidelines for minimizing the effects of RF interference include:

- **Ground Plane:** Provide 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  $\Omega$ .
  - ☐ The recommended series resistance for communication lines such as I<sup>2</sup>C and SPI is 330  $\Omega$ .
- **Trace Length:** Minimize trace length whenever possible.
- **Current Loop Area:** Minimize the return path for current. Hatched ground instead of solid fill should be provided 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

Selecting a low frequency for the switched capacitor clock helps to reduce radiated emissions from the CapSense sensor. 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 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.
- Place a ferrite bead around power supply or communication lines to filter out high-frequency noise.

## 5.4 Software Filtering

Using software filters is 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

[Table 5-4](#) details 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

## 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 the high-power blocks.
- Optimize CPU speed for low power.
- Operate at a lower  $V_{DD}$ .

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

### 5.5.2 Sleep-Scan Method

In typical applications, the CapSense controller does not need to always 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.

$$I_{AVE} = \frac{(I_{Act} \times t_{Act}) + (I_{Slp} \times t_{Slp})}{T} \quad \text{Equation 10}$$

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

$$P_{AVE} = V_{DD} \times I_{AVE} \quad \text{Equation 11}$$

### 5.5.3 Response Time versus Power Consumption

As illustrated in Equation 11, 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 CapSense button response time. 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, an optimized method can be used 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 in the 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 10.
8. Calculate average power consumption using Equation 11.

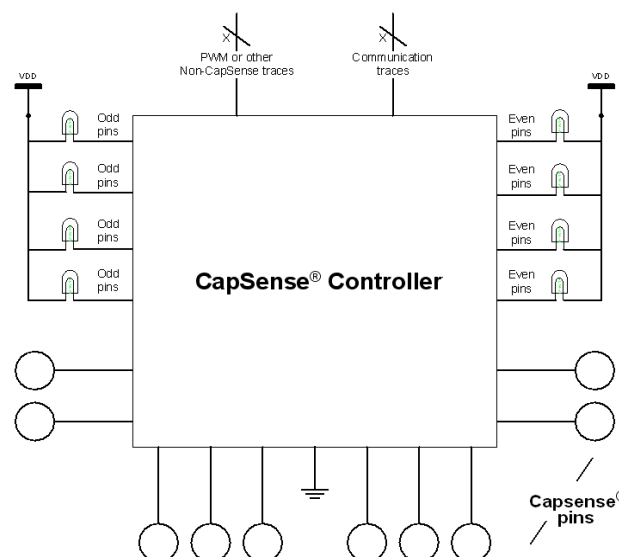
Figure 5-1. Global Resources Window

Global Resources	Value
Power Setting [ Vcc / SysClk freq ]	5.0V / 24MHz
CPU_Clock	SysClk/1
Sleep_Timer	64_Hz
VC1= SysClk/N	512_Hz
VC2= VC1/N	64_Hz
VC3 Source	1_Hz
VC3 Divider	1

## 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

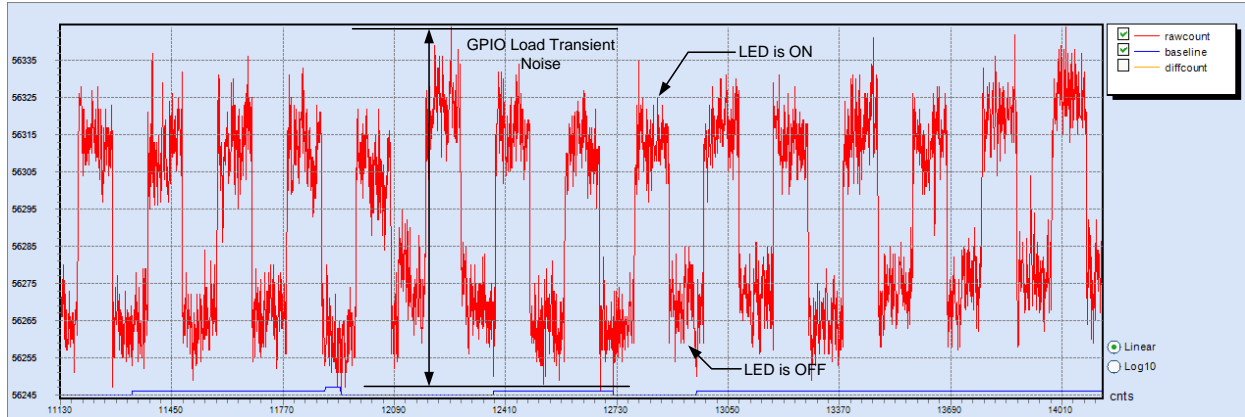


The architecture of the CapSense controller imposes a restriction on current budget for even and odd port pin numbers. An odd pin can be any port pin having an odd number as pin number. For a CapSense controller, if the current budget for the odd port pin is 100 mA, the total current drawn through all odd port pins should not exceed 100 mA. In addition to the total current budget limitation, there is also a maximum current limitation for each port pin that is defined in the CapSense controller datasheet.

## 5.7 GPIO Load Transient

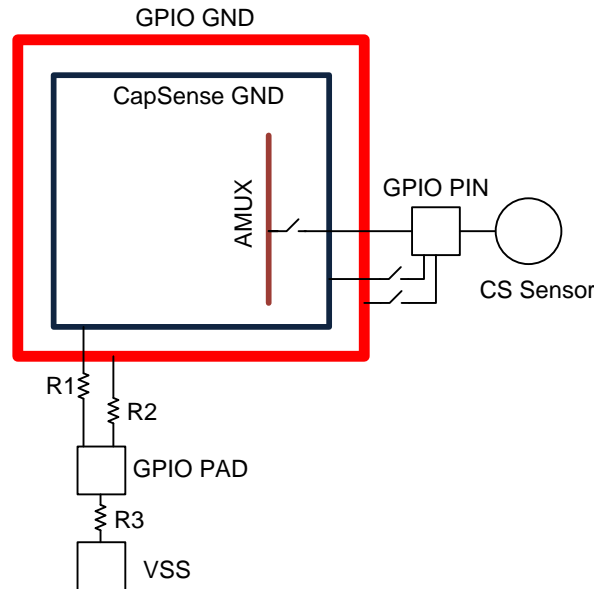
When GPIOs sink a large current ( $>10\text{ mA}$ ) to the ground of the chip by driving port pins to strong-low, noise will be introduced into the CapSense system. The instantaneous change in the amount of current flow to ground through the GPIOs is referred as GPIO load transient. The noise introduced into the CapSense system due to the GPIO load transient is called GPIO load transient noise, as shown in Figure 5-3. This section shows you how to reduce this noise using hardware techniques and compensate the noise using firmware techniques.

Figure 5-3. GPIO Load Transient Noise in a CapSense System



When current is sunk through a GPIO pin, the voltage at CapSense ground (GPIO PAD) will not be zero because of the non-zero bond-wire resistance,  $R_3$ . Because of the non-zero ground potential, the sensor will not be completely discharged when LED is sinking the current; this will cause an increase in the sensor raw count.

Figure 5-4. Ground Structure in CY8C20x66A/S



**Note:**  $R_1$ ,  $R_2$ ,  $R_3$  are bond-wire resistances

For a robust CapSense design, the worst-case GPIO transient noise should be less than 30% of the finger-touch signal. The worst-case noise appears in the CapSense system when the GPIO state is changed from a no-current-flow state (e.g., all LEDs OFF) to a maximum-current-flow state (e.g., all LEDs ON).

The GPIO load transient noise increases with the sensor scan resolution. CapSense sensors with high parasitic capacitance or proximity sensors require higher sensor-scan resolution to achieve an SNR > 5:1. In such systems, the effect of GPIO load transient is more pronounced. In some cases, the noise due to GPIO load transient might be higher than the signal due to finger-touch and causes sensor false triggers. The section below shows how to reduce GPIO load transient noise.

### 5.7.1 Hardware Guidelines to Reduce GPIO Load Transient Noise

- a) Reduce Sensor  $C_P$   
The sensor  $C_P$  determines the sensor-scan resolution parameter. The larger the  $C_P$ , the higher will be the resolution parameter required to achieve an SNR > 5:1. Setting the resolution parameter high causes the amplitude of the GPIO load transient noise to increase. Therefore, it is recommended to minimize the sensor  $C_P$  by following the layout guidelines mentioned in the [Getting Started with CapSense](#) design guide.
- b) Reduce LED sink current  
The GPIO load transient noise is directly proportional to the LED sink current. It is recommended to keep the LED sink current within the limits as specified in the [device datasheet](#). If the GPIO has to sink a current which is more than the maximum value specified in datasheet, use an external transistor or a driver IC.
- c) Select Appropriate Pins for LED  
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 the device ground pin to minimize the GPIO load transient noise.

### 5.7.2 Firmware Guidelines to Compensate GPIO Load Transient Noise

To prevent sensor false-triggers due to GPIO load transient noise, the sensor baseline can be updated using rule-based algorithms. One of the methods to compensate the baseline is explained below.

[Figure 5-5](#) shows a condition in which false triggers are seen due to GPIO load transient.

1. At instant 1, there is no finger on the sensor and the LED is in the OFF condition.
2. At instant 2, a finger is on the sensor and the shift in rawcount is greater than finger threshold.
3. Because the shift in rawcount is greater than the finger threshold, the LED is turned ON at instant 3.
4. When the LED is turned ON, because of GPIO load transient noise, the rawcount further shifts.
5. At instant 4, even if the finger is removed, the rawcount will not return to initial value because of the shift in the rawcount due to GPIO load transient noise. If this shift is greater than the finger threshold, the LED will remain ON permanently indicating a sensor false-trigger.

To prevent the sensor and the LED from remaining in the ON condition permanently, the sensor baseline should be compensated, which is explained in the steps below.

Figure 5-5. CapSense Sensor Variables when Baseline is Not Compensated

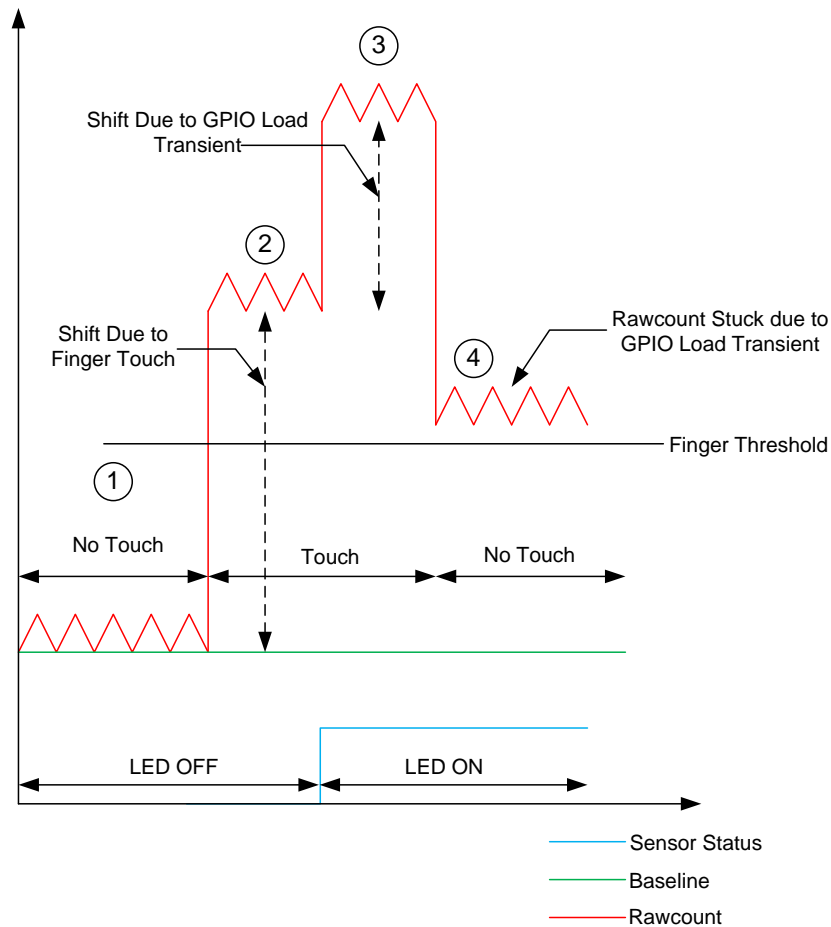
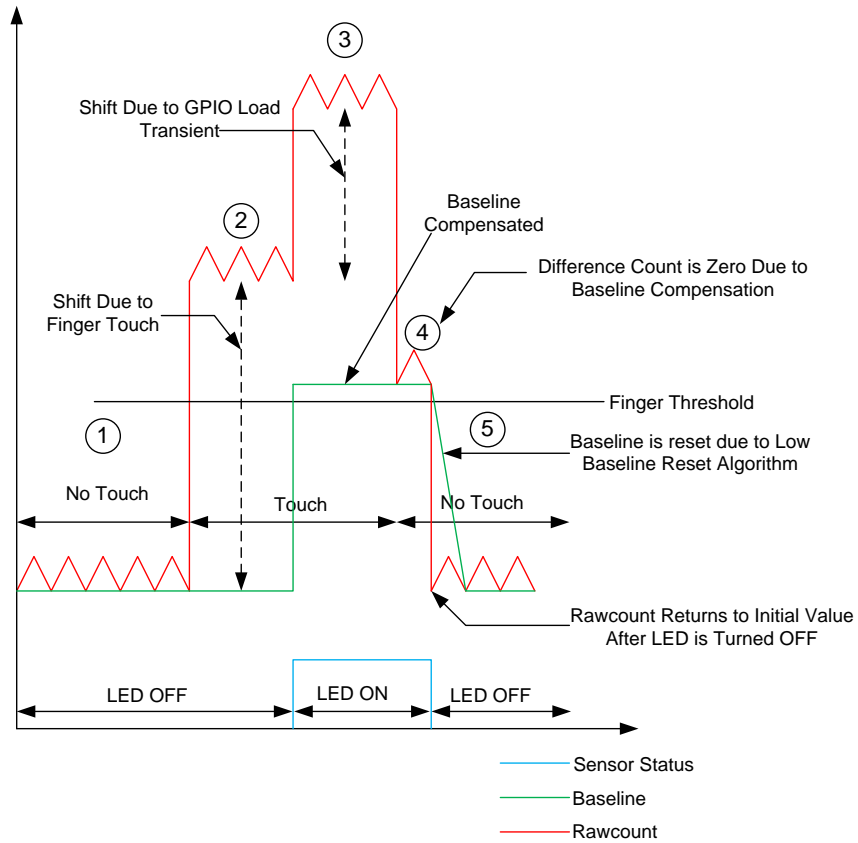


Figure 5-6 shows a condition in which false triggers are eliminated by compensating the sensor baseline.

1. At instant 1, there is no finger on the sensor and the LED is in the OFF condition.
2. At instant 2, a finger is on the sensor and the shift in rawcount (difference count) is greater than finger threshold.
3. Because the shift in difference count is greater than the finger threshold, the LED is turned ON at instant 3.
4. When the LED is turned ON, the noise due to GPIO load transient is calculated. Here, Noise = Rawcount (When LED in ON) – Rawcount (When LED is OFF)  
This noise count due to GPIO load transient is added to the baseline, and as a result, when the finger is removed, the difference count value will be zero and the LED will be turned OFF.
5. After the LED is turned OFF, the rawcount will return to the initial value and the baseline is reset due to the low-baseline reset algorithm.

Figure 5-6. CapSense Sensor Variables when Baseline is Compensated



## 5.8 PCB Layout Guidelines

Detailed PCB layout guidelines are available in [Getting Started with CapSense](#).

## 6. 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 Mode is disabled. The only circuits left in operation are supply voltage monitor and 32-kHz internal oscillator. Power saving techniques other than sleep mode are:

- Disable CapSense (PSoC) analog block references
- Disable CT and 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 correctly awakened from sleep when necessary, and the user must be aware that the device is sleeping to allow extra processing.

### 6.1 Additional Power Saving Techniques

All the power saving techniques, with the exception of sleep mode, are application-based. Some of them produce undesirable results. Each technique is discussed in detail in the following sections.

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

#### 6.1.1 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 effects 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 the 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. Therefore, 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. Three registers, PRTxDM0, PRTxDM1, and PRTxDM2, control the drive modes. One bit per register is assigned to a pin. Therefore, 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
```



### 6.1.2 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;
}
```

### 6.1.3 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. After 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.

### 6.1.4 Pending Interrupts

If an interrupt is pending, enabled, and scheduled to occur 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.

### 6.1.5 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;
```

## 6.2 Post Wakeup Execution Sequence

If the CapSense controller is awakened through a reset, 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 prefetched 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.

### 6.2.1 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
```

### 6.2.2 Execution of Global Interrupt Enable

Avoid interrupts on the instruction boundary of writing the SLEEP bit. This can 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, interrupts are temporarily disabled before sleep preparations and then re-enabled 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.

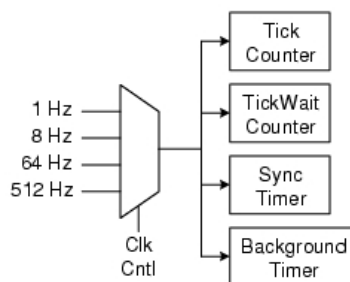
### 6.2.3 I<sup>2</sup>C Slave with Sleep Mode

There are a few complications using an I<sup>2</sup>C 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<sup>2</sup>C address. When an I<sup>2</sup>C 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<sup>2</sup>C bus. The master can then send a dummy START condition to wake up the CapSense controller. There is some lag time between waking up and being able to process an I<sup>2</sup>C address, so the master may need to delay up to 200  $\mu$ s 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<sup>2</sup>C falling-edge traffic, which causes more total active time and higher 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.

### 6.2.4 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 of 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 periodically wake the CapSense controller up 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. A simple block diagram for this functionality is shown in [Figure 6-1](#).

Figure 6-1. Sleep Timer User Module Block Diagram



# 7. Resources



## 7.1 Website

Visit [Cypress's CapSense Controllers website](#) to access all of the reference material discussed in this section.

Find a variety of technical resources for the CapSense CY8C20xx6A/H/AS family of devices on the [CY8C20xx6A/H](#) web page.

## 7.2 Datasheet

The datasheets for the CapSense CY8C20XX6A/H/AS family of devices are available at [www.cypress.com](http://www.cypress.com).

- [CY8C20x36A, CY8C20x46A, CY8C20x66A, CY8C20x96A, CY8C20x46AS, and CY8C20x66AS](#)
- [CY8C20336H, CY8C20446H](#)

## 7.3 Technical Reference Manual

Cypress created the following technical reference manual to give quick and easy access to information on CapSense controller functionality, including top-level architectural diagrams along with register and timing diagrams.

- [PSoC® CY8C20x66, CY8C20x66A, CY8C20x46/96, CY8C20x46A/96A, CY8C20x36, CY8C20x36A Technical Reference Manual \(TRM\)](#)

## 7.4 Development Kits

### 7.4.1 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<sup>2</sup>C-to-USB Bridge hardware are included for tuning and data acquisition.

- [CY3280-20xx6](#) Universal CapSense Controller

### 7.4.2 Universal CapSense Module Boards

#### 7.4.2.1 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

#### 7.4.2.2 Matrix Button Module Board

The [CY3280-BMM](#) Matrix Button Module consists of eight LEDs and eight CapSense sensors organized in a 4x4 matrix format to form 16 physical buttons. This module connects to any CY3280 Universal CapSense Controller Board.

#### 7.4.2.3 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.

#### 7.4.2.4 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

#### 7.4.2.5 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

### 7.4.3 In-Circuit Emulation (ICE) Kits

The ICE pod provides the interconnection between the CY3215-DK In-Circuit Emulator and the target PSoC device in a prototype system or PCB using package-specific pod feet, through a flex cable. The following pods are available.

- [CY3250-20246QFN In-Circuit Emulation \(ICE\) Pod Kit for Debugging CY8C20236/46A CapSense PSoC Devices](#)
- [CY3250-20346QFN In-Circuit Emulation \(ICE\) Pod Kit for Debugging CY8C20336/346A CapSense PSoC Devices](#)
- [CY3250-20666QFN In-Circuit Emulation \(ICE\) Pod Kit for Debugging CY8C20636/646/666A CapSense PSoC Devices](#)
- [CY3250-20566 In-Circuit Emulation \(ICE\) Pod Kit for Debugging CY8C20536/546/566A CapSense PSoC Devices](#)

## 7.5 Sample Board Files

Cypress offers sample schematic and board files, which can be used as a reference to quickly complete your PCB design process.

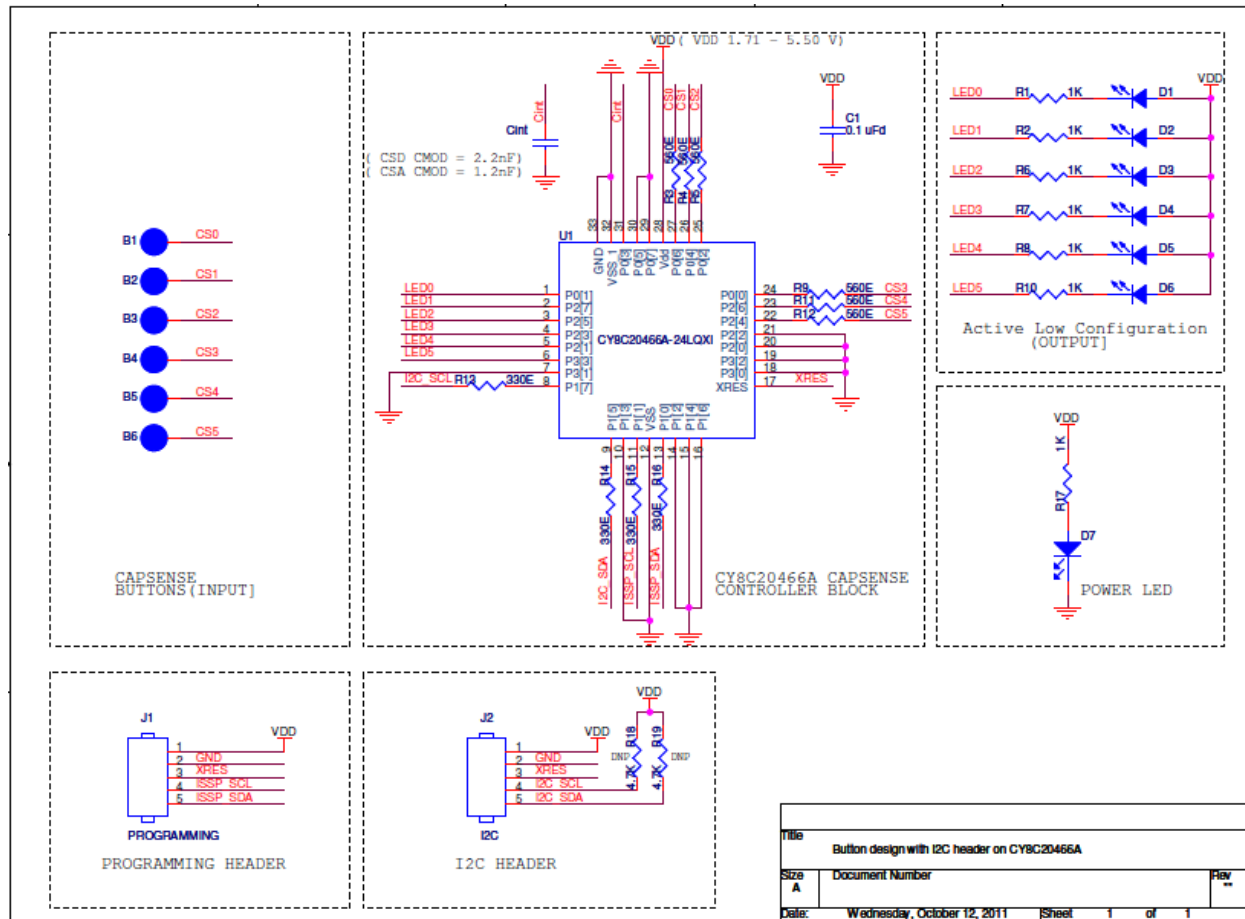
- Button design with I<sup>2</sup>C header on CY8C20466A
- Button and slider design with I<sup>2</sup>C header on CY8C20466A

**Note** The board files (schematic, layout, and Gerber files) will be placed in the landing page of this document.

[Figure 7-1](#) and [Figure 7-2](#) show the board schematics.

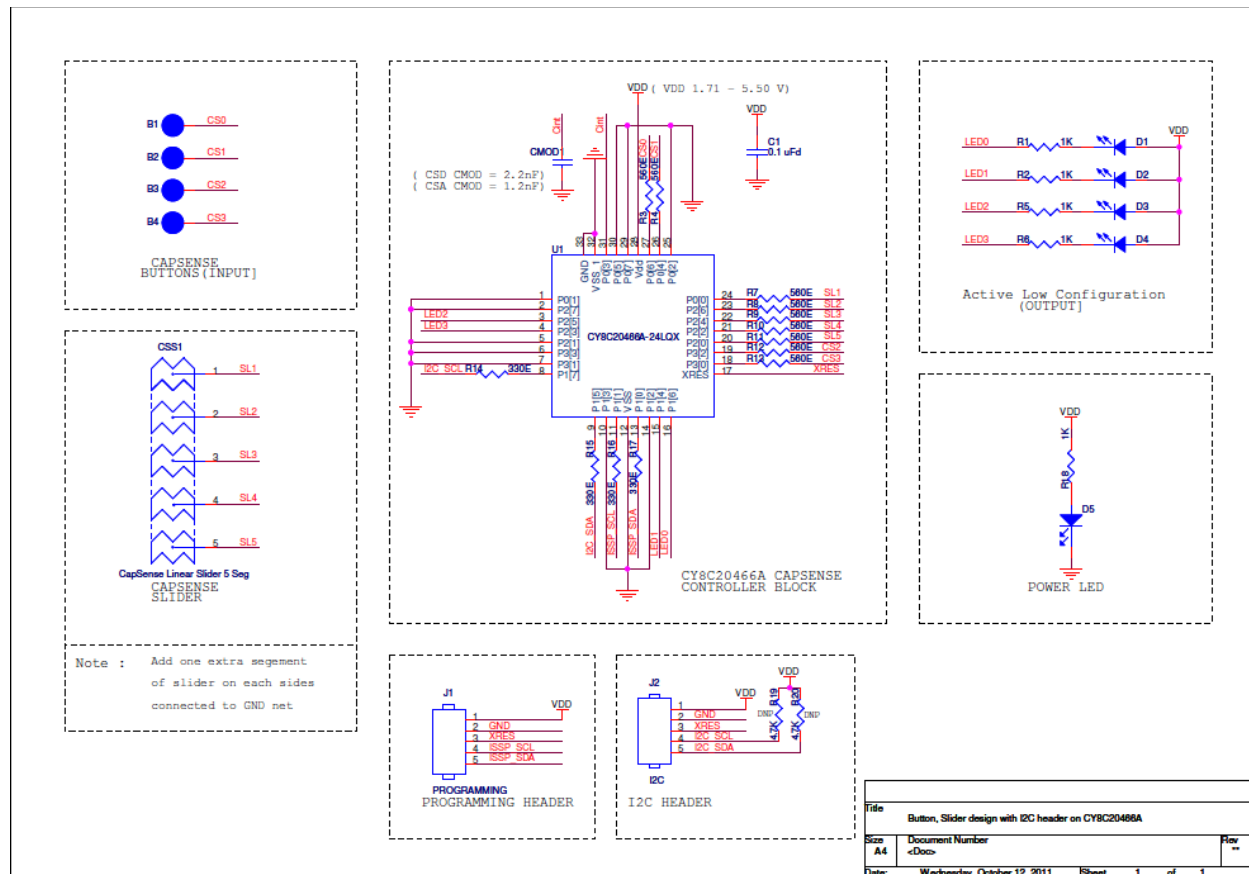
The following schematic is designed to support:

- Six CapSense sensors. The sensors are assigned to pins P0[6], P0[4], P0[2], P0[0], P2[6], and P2[4] of the CY8C20466A-24LQXI device.
- Six GPOs connected to pins P0[1], P2[1], P2[3], P2[5], P2[7], and P3[3] of CY8C20466A-24LQXI to drive LEDs D1, D2, D3, D4, D5, and D6.
- Programming of CY8C20466A-24LQXI by way of the programming header J1.
- I<sup>2</sup>C communication with CY8C20466A-24LQXI by way of the I<sup>2</sup>C header J2.

Figure 7-1. Button Design with I<sup>2</sup>C Header on CY8C20466A - Board Schematic


This following schematic is designed to support:

- Four CapSense sensors. The sensors are assigned to pins P0[6], P0[4], P3[2], and P3[0] of the CY8C20466A-24LQXI device.
- Linear slider with five segments. The segments are assigned to pins P0[0], P2[6], P2[4], P2[2], and P2[0] of the CY8C20466A-24LQXI device.
- Four GPOs connected to pins P1[6], P1[4], P2[5], and P2[7] CY8C20466A-24LQXI to drive LEDs D1, D2, D3, and D4.
- Programming of CY8C20466A-24LQXI by way of the programming header J1.
- I<sup>2</sup>C communication with CY8C20466A-24LQXI by way of the I<sup>2</sup>C header J2.

Figure 7-2. Button and Slider Design with I<sup>2</sup>C Header on CY8C20466A


## 7.6 PSoC Programmer

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

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

## 7.7 CapSense Data Viewing Tools

Often during CapSense design, you will want to monitor relevant CapSense data (raw counts, baseline, difference counts, and so on) for tuning and debugging purposes.

Application note [AN2397 – CapSense Data Viewing Tools](#) gives information to help you identify and use the right tools for CapSense data viewing and logging.

## 7.8 PSoC Designer

Cypress offers an exclusive integrated development 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 fully characterized 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.

## 7.9 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 CY8C20xx6A](#)

## 7.10 Design Support

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

- [Knowledge Base Articles](#) – See the technical articles by product family or perform a search on various CapSense topics.
- [CapSense Application Notes](#) – See 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 Cypress's CapSense product line.
- [Video Library](#) – Quickly get up to speed with tutorial videos.
- [Quality and 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  $(2^N - 1)/\text{Modulator Clock Frequency}$ , 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<sub>REF</sub>**

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: AN65973 - CY8C20xx6A/H/AS CapSense® Design Guide			
Document Number: 001-65973			
Revision	Issue Date	Origin of Change	Description of Change
**	12/14/2010	ANBA	New Design Guide
*A	03/04/2011	BVI	Multiple chapter enhancements for content and reader clarity
*B	06/14/2011	BVI	Part numbers for SmartSense_EMC enabled CapSense controller is updated. Best practice for SmartSense_EMC user module is updated
*C	12/23/2011	BVI	Added bulleted abstract Added Section 1.2 Moved Documented Revision History to end of document Rewrote Section 2 for clarity Multiple updates based on NPS audit
*D	02/20/2012	VAIR/ZINE	Corrected maximum raw counts value from $2^{(N-1)}$ to $(2^N) - 1$ Corrected reference to CSD UM APIs in section 4-9 Corrected references in Figure 4-4
*E	09/06/2012	ZINE	Updated links to external documents
*F	09/03/2014	SSHH	Corrected definition of Hysteresis in section 3.4.1.2. Updated the sensor auto-reset information in section 3.4.1.8.
*G	01/22/2015	DCHE/SSHH	Updated sections 3.4.1.2 and 3.4.1.8 Added section 5.7 – GPIO Load Transient Updated to new template. Completing Sunset Review .
*H	01/21/2016	VAIR	Added Glossary .
*I	02/07/2017	SSHH	Updated to new template. Completing Sunset Review .
*J	07/13/2017	AESATMP9	Updated logo and copyright.