



AN90071 - CY8CMBR3xxx

CapSense® Design Guide

Doc. No. 001-90071 Rev. *H

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
<http://www.cypress.com>

Copyrights

© Cypress Semiconductor Corporation, 2014-2017. 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. Other names and brands may be claimed as property of their respective owners.

Contents



1. Introduction.....	5
1.1 Abstract	5
1.2 Introduction.....	5
1.3 CY8CMBR3xxx Family Feature Overview	5
1.4 CY8CMBR3xxx Family Feature Comparison	7
1.5 CY8CMBR3xxx CapSense System Overview	8
1.6 CapSense Design Flow	8
2. CapSense Technology	11
2.1 CapSense Fundamentals	11
2.2 Capacitive Touch-Sensing Method in CY8CMBR3xxx	13
2.3 CapSense Tuning	14
2.4 SmartSense Auto-Tuning	14
2.5 Sensor Types	15
2.6 Liquid Tolerance	17
3. CapSense Schematic Design	20
3.1 Schematic Design Considerations	20
3.2 Schematic Checklist	24
3.3 Sample Schematics	24
4. CapSense Layout Guidelines	29
4.1 Design Toolbox.....	29
4.2 Layout Guidelines for Proximity Sensor.....	38
4.3 Layout Guidelines for Liquid Tolerance	41
4.4 Sample Layouts.....	43
5. Configuring CY8CMBR3xxx.....	47
5.1 Configurations in CY8CMBR3xxx.....	47
5.2 Configuring CY8CMBR3xxx	57
6. CapSense Performance Tuning	76
6.1 General Considerations.....	76
6.2 Tuning the Buttons, Sliders, and Guard Sensor	78
6.3 Tuning a Proximity Sensor	83
7. Low-Power Design Considerations	91
7.1 CY8CMBR3xxx Operating Modes	91
7.2 Factors Affecting Power Consumption	93

7.3	System Design Recommendations for Low Power Consumption.....	94
8.	Resources	95
8.1	Website	95
8.2	Datasheet	95
8.3	Registers Technical Reference Manual.....	95
8.4	Design Toolbox.....	95
8.5	EZ-Click™ 2.0	95
8.6	Development Kit	95
8.7	Design Support.....	96
9.	Appendix	97
9.1	High-Level APIs.....	97
9.2	Low-Level APIs.....	103
	Glossary.....	105
	Revision History	111

1. Introduction



1.1 Abstract

This guide shows how to design capacitive touch-sensing applications with the CY8CMBR3xxx CapSense controller family. The CY8CMBR3xxx CapSense controller family offers unprecedented signal-to-noise ratio (SNR), best-in-class liquid tolerance, and a wide variety of sensors such as buttons, sliders, and proximity sensors. This guide explains the CY8CMBR3xxx CapSense design flow, design tools, design considerations, and performance tuning.

1.2 Introduction

Capacitive touch sensors are user-interface devices that use the capacitance of the human body to detect the presence of a finger on, or near, a sensor. Capacitive sensors are aesthetically superior, easy-to-use, and have long lifetimes. Cypress CapSense solutions bring elegant, reliable, and easy-to-use capacitive touch-sensing functionality to your product. Cypress CapSense solutions have replaced more than four billion mechanical buttons.

The CY8CMBR3xxx CapSense Express™ controllers enable advanced, yet easy-to-implement, user interface solutions using capacitive touch sensing. This register-configurable family, which supports up to 16 capacitive sensing inputs, eliminates time-consuming firmware development and tuning process from design cycles. These controllers are ideal for implementing capacitive buttons, sliders, and proximity sensing solutions with minimal development-cycle time. The CY8CMBR3xxx controller enables the development of robust liquid-tolerant CapSense applications by eliminating false touches due to mist, moisture, liquid droplets, or streaming liquid.

The CY8CMBR3xxx CapSense Design Guide explains how to implement capacitive touch sensing functionality using the CY8CMBR3xxx family of CapSense controllers. This guide assumes that you are familiar with Cypress' CapSense technology. If you are new to CapSense technology, refer to the [Getting Started with CapSense](#) guide to understand its basics.

This design guide helps you understand:

- Features of the CY8CMBR3xxx CapSense controller family
- CapSense technology used in the CY8CMBR3xxx family
- Schematic and layout design considerations
- How to configure the CY8CMBR3xxx family
- Performance tuning
- Low-power design considerations

1.3 CY8CMBR3xxx Family Feature Overview

The CY8CMBR3xxx family is a high-performance, low-power CapSense Express controller that enables advanced, yet easy-to-implement, user interface solutions using capacitive touch sensing.

The CY8CMBR3xxx family features include:

- Register-configurable CapSense Express controller
 - ☐ Configurable through the I²C interface
 - ☐ No firmware development or device programming required
 - ☐ Supports up to 16 capacitive sensing inputs

- ☐ Supports up to eight general purpose outputs (GPOs)
- ☐ GPOs are either linked to the CapSense sensors or controlled by the host processor
- ☐ GPOs support direct LED drive
- SmartSense™ Auto-Tuning
 - ☐ CapSense algorithm that continuously compensates for system, manufacturing, and environmental changes
 - ☐ Automatically configures capacitive button parameters for optimal performance
 - ☐ Eliminates the need for manual system tuning
 - ☐ Wide parasitic capacitance (C_P) range (5-45 pF)
 - ☐ Allows user to set thresholds manually if required
- Advanced features
 - ☐ Wake-on-approach
 - Wakes the system from a low-power mode to active mode on a proximity event.
 - ☐ Liquid-tolerance
 - Enables capacitive buttons to work properly in the presence of liquid droplets, liquid stream, or mist
 - ☐ PWM output on GPO pins
 - User-configurable LED brightness during sensor ON/OFF state
 - ☐ Flanking Sensor Suppression (FSS)
 - Distinguishes between touches from closely spaced buttons
 - ☐ Firmware filters to improve SNR
 - ☐ Buzzer signal output for audible touch feedback
 - ☐ Analog voltage output using external resistor bridge
 - ☐ Host interrupt output to alert the host of any change in sensor status
 - ☐ System-diagnostics data through the I²C interface
 - Simplifies production line testing and system debug
- Noise immunity
 - ☐ Pseudo-Random Sequence (PRS) clock source to minimize electromagnetic interference
 - ☐ Electro Magnetic Compatibility (EMC) feature provides superior noise immunity against external radiated and conducted noise
- System diagnostics that can detect:
 - ☐ Sensor shorted to VDD and Ground
 - ☐ Sensor-to-Sensor shorts
 - ☐ Sensor shorted to Shield
 - ☐ Improper value of modulating capacitor (C_{MOD})
 - ☐ Parasitic capacitance (C_P) of the sensors that is out of range
- I²C interface
 - ☐ Supports up to 400-kHz speed
 - ☐ Wake-on-hardware address match
 - ☐ No clock stretching
- Wide operating voltage range
 - ☐ 1.71-V to 5.5-V operation

- Low power consumption
 - Supports different modes of operation to reduce the power consumption of the device
 - Average current consumption of 22 μ A per sensor at 120-ms scan period
 - Deep sleep current with I²C ON : 2.5 μ A
- Industrial temperature range: –40 °C to +85 °C
- Package options
 - 8-pin SOIC (150 mil)
 - 16-pin SOIC (150 mil)
 - 16-pin QFN (3 × 3 × 0.6 mm)
 - 24-pin QFN (4 × 4 × 0.6 mm)

1.4 CY8CMBR3xxx Family Feature Comparison

Table 1-1 compares the features supported by different controllers of the CY8CMBR3xxx family. These controllers are differentiated based on the number of sensors and features supported. Use this table to select the appropriate device that meets your design requirements.

Table 1-1. Comparison of CY8CMBR3xxx Controllers

#	Feature	MBR3116	MBR3106S	MBR3110	MBR3108	MBR3102	MBR3002
1	Maximum number of buttons	16	11	10	8	2	2
2	Maximum number of slider segments	×	10	×	×	×	×
3	Maximum number of proximity sensors	2	2	2	2	2	×
4	Shield electrode	✓	✓	✓	✓	✓	×
5	Guard Sensor	✓	×	✓	✓	×	×
5	Wake-on-approach	✓	✓	✓	✓	✓	×
6	Liquid tolerance	✓	×	✓	✓	✓	×
7	Maximum number of GPOs/LED drive outputs	8	0	5	4	1	2
8	LED brightness control	✓	×	✓	✓	✓	×
9	I ² C Interface	✓	✓	✓	✓	✓	×
10	Buzzer drive output	✓	✓	✓	✓	×	×
11	Host interrupt output	✓	✓	✓	✓	×	×
12	FSS	✓	✓	✓	✓	✓	×
13	Median & IIR filter	✓	✓	✓	✓	✓	✓
14	Advanced-Low-Pass Filter	✓	×	✓	✓	✓	×
15	Electro Magnetic Compatibility (EMC)	✓	✓	✓	✓	✓	×
16	Sensitivity control	✓	✓	✓	✓	✓	×
17	Auto finger threshold	✓	✓	✓	✓	✓	✓
18	Flexible finger threshold	✓	✓	✓	✓	✓	×
19	LED ON time	✓	×	✓	✓	✓	×

#	Feature	MBR3116	MBR3106S	MBR3110	MBR3108	MBR3102	MBR3002
20	Toggle Mode	✓	✗	✓	✓	✓	✗
21	System diagnostics	✓	✓	✓	✓	✓	✓
22	Sensor auto-reset	✓	✓	✓	✓	✓	✓

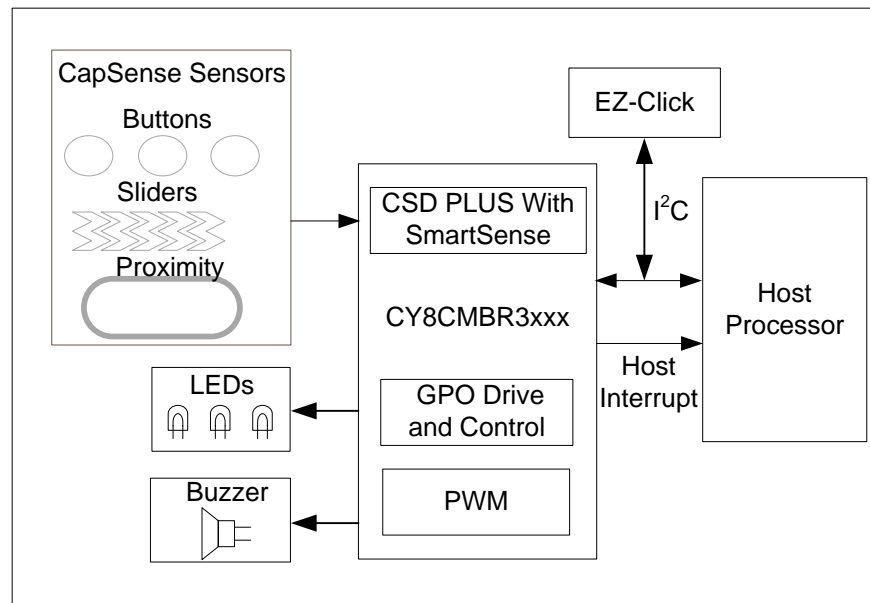
Notes:

- The CY8CMBR3102 controller does not support the GUARD sensor and hence it provides liquid tolerance only for liquid droplets and not for liquid flow.
- The sensor auto-reset for CY8CMBR3002 is fixed to 20 s and is not configurable.

1.5 CY8CMBR3xxx CapSense System Overview

Figure 1-1 shows the typical system-level block diagram of a CapSense application with the CY8CMBR3xxx CapSense controller. The CY8CMBR3xxx controller acts as a slave device that detects finger touches on the user interface panel and reports the touch status to the host processor. The CY8CMBR3xxx controller sends an interrupt pulse to the host processor whenever the sensor status changes, so that the host can read the sensor status through the I²C interface. The CY8CMBR3xxx controller provides touch feedback by driving GPOs and dedicated buzzer output. The EZ-Click™ tool is used to configure and debug the CY8CMBR3xxx controller using the I²C interface.

Figure 1-1. Block Diagram of a CapSense System With CY8CMBR3xxx Controller



1.6 CapSense Design Flow

Figure 1-2 shows the typical flow of a product design cycle with capacitive sensing using the CY8CMBR3xxx controller. Table 1-2 lists the supporting documentation and ecosystem for designing the CapSense application with the CY8CMBR3xxx controller.

Figure 1-2. Typical CapSense Product Design Flow

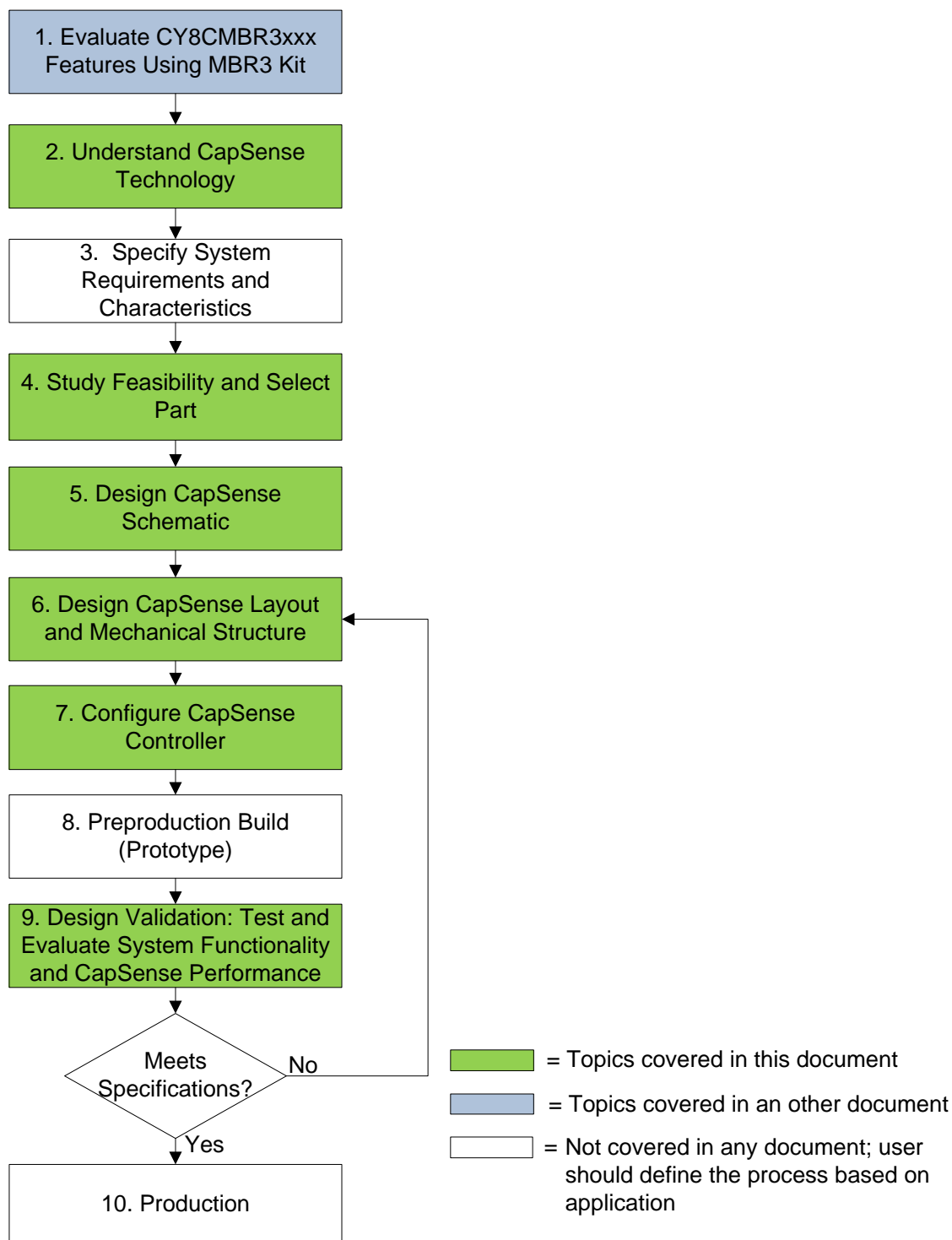


Table 1-2. Supporting Documentation and Ecosystem

Steps in the Flowchart	Supporting Cypress Documentation	
	Name	Chapter
1. Evaluate Features of CY8CMBR3xxx Using MBR3 Kit	CY3280-MBR3 Evaluation Kit User Guide	NA
2. Understand CapSense Technology	Getting Started With CapSense CY8CMBR3xxx CapSense Design Guide (This document)	Chapter 2
3. Specify System Requirements and Characteristics	Not covered in any document; user should define the process based on application	NA
4. Feasibility Study: Part Selection	CY8CMBR3xxx CapSense Design Guide (This document) CY8CMBR3xxx Datasheet	Chapter 1
5. Design CapSense Schematic	CY8CMBR3xxx CapSense Design Guide (This document)	Chapter 3
6. Design CapSense Layout and Mechanical Structure	CY8CMBR3xxx CapSense Design Guide (This document)	Chapter 4
7. Configure CapSense Controller	CY8CMBR3xxx CapSense Design Guide (This document)	Chapter 5
8. Preproduction Build (Prototype)	Not covered in any document; user should define the process based on the application	NA
9. Design Validation	CY8CMBR3xxx CapSense Design Guide (This document)	Chapter 6
10. Production	Not covered in any document; user should define the process based on the application	NA

2. CapSense Technology

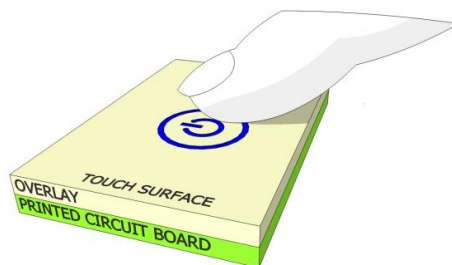


The capacitive touch sensing technology enables measurement of the change in the capacitance between a plate (the sensor) and its environment to detect the presence of a finger on or near a touch surface.

2.1 CapSense Fundamentals

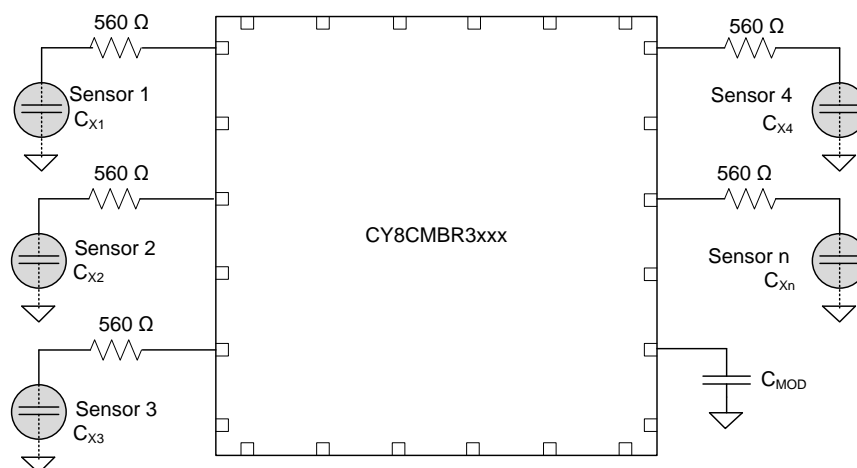
A typical CapSense sensor consists of a copper pad etched on the surface of a Printed Circuit Board (PCB). A nonconductive overlay serves as the touch surface for the sensor, as [Figure 2-1](#) shows.

Figure 2-1. Capacitive Touch Sensor



PCB traces and vias connect the sensor pads to the CapSense input pins of the CY8CMBR3xxx controller. As [Figure 2-2](#) shows, the total amount of capacitance on each of the sensor pins is modeled as the equivalent lumped capacitor with values of C_{X1} , C_{X2} , through C_{Xn} . The CapSense circuitry within the CY8CMBR3xxx controller converts these capacitance values into equivalent digital counts. These digital counts are then processed by the controller to detect touches.

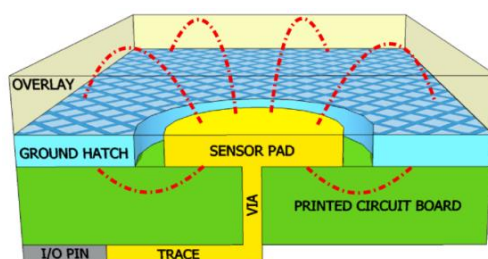
Figure 2-2. CapSense Implementation in the CY8CMBR3xxx Device



Note: The CY8CMBR3xxx controller requires an external capacitor C_{MOD} , which is connected between the CMOD pin and ground.

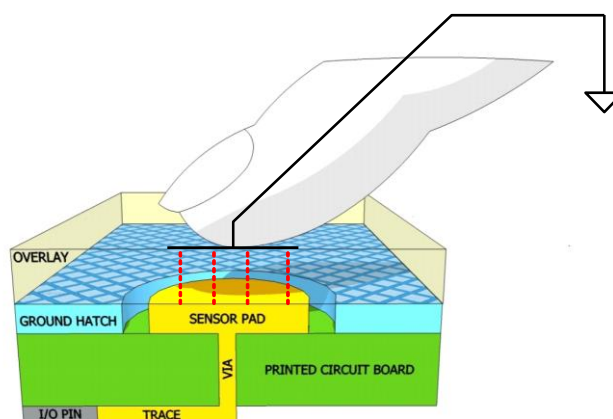
The capacitance of the sensor in the absence of a touch is called the *parasitic capacitance*, C_P . Parasitic capacitance results from the electric field between the sensor (including the sensor pad, traces, and vias) and other conductors in the system such as the ground planes, traces, any metal in the product's chassis or enclosure, and so on. The sensor pin and internal capacitances of the CY8CMBR3xxx controller also contribute to the parasitic capacitance. However, these internal capacitances are typically very small compared to the sensor capacitance. Figure 2-3 shows how a CY8CMBR3xxx sensor pin is connected to a sensor pad with traces and vias. Typically, a ground hatch surrounds the sensor pad to isolate it from other sensors and traces. Although this diagram shows some field lines around the sensor pad, the actual electric field distribution is complex.

Figure 2-3. Parasitic Capacitance



When a finger is present on the overlay, the conductive nature and large mass of the human body forms a grounded, conductive plane parallel to the sensor pad, as Figure 2-4 shows.

Figure 2-4. Section of Typical CapSense PCB With the Sensor Being Activated by a Finger



This arrangement forms a parallel plate capacitor. The capacitance between the sensor pad and the finger is:

$$C_F = \frac{\epsilon_0 \epsilon_r A}{d} \quad \text{Equation 1}$$

Where: ϵ_0 = Free space permittivity

ϵ_r = Dielectric constant (relative permittivity) of the overlay

A = Area of finger and sensor pad overlap

d = Overlay thickness

C_F is known as the finger capacitance. The parasitic capacitance C_P and finger capacitance C_F are parallel to each other because both represent the capacitances between the sensor pin and ground. Therefore, the total capacitance C_X of the sensor, when the finger is present on the sensor, is the sum of C_P and C_F .

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

In the absence of touch, C_X is equal to C_P .

The CY8CMBR3xxx controller converts the capacitance C_X into equivalent digital counts called *raw counts*. Because a finger touch increases the total capacitance of the sensor pin, an increase in the raw counts indicates a finger touch. As the parasitic capacitance C_P increases, the ratio of C_F to C_P decreases – the per unit change in capacitance corresponding to a finger touch decreases. Therefore, as C_P increases, touch detection becomes more difficult.

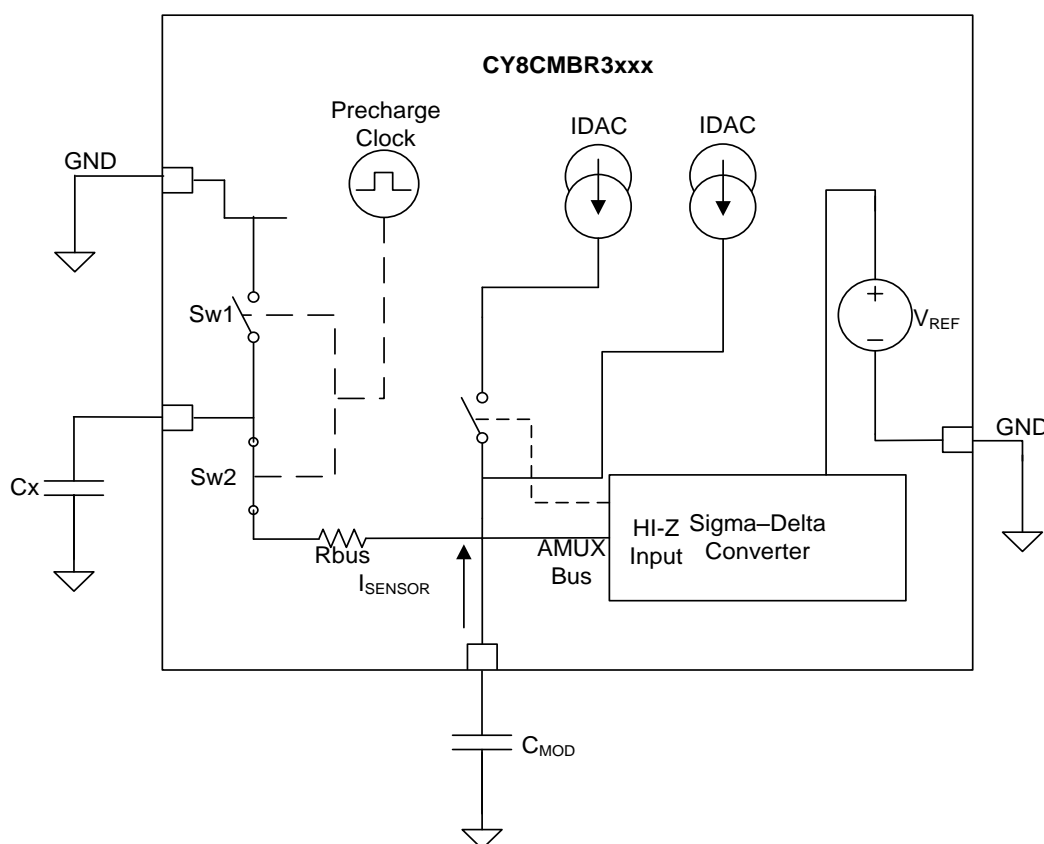
In general, C_P is an order of magnitude greater than C_F . C_P usually ranges from 10-20 pF, but in extreme cases it can be as high as 45 pF. C_F usually ranges from 100-400 fF.

2.2 Capacitive Touch-Sensing Method in CY8CMBR3xxx

The CY8CMBR3xxx controller uses a capacitive touch-sensing method - CapSense Sigma Delta PLUS (CSD PLUS) - to convert the changes in capacitance to digital counts. The CSD PLUS touch-sensing method delivers an SNR that ensures touch accuracy even in extremely noisy environments.

The CSD PLUS sensing method in the CY8CMBR3xxx device incorporates C_X into a switched capacitor circuit, as Figure 2-5 shows. The sensor (C_X) is alternatively connected to GND and the Analog MUX (AMUX) bus by the non-overlapping switches Sw1 and Sw2. Sw1 and Sw2 are driven by the Precharge Clock to bleed a current (I_{SENSOR}) from the AMUX bus. The magnitude of I_{SENSOR} is directly proportional to the magnitude of C_X . The sigma-delta converter samples the AMUX bus voltage and generates a modulating bit stream that controls the constant current source, IDAC. The IDAC charges AMUX such that the average AMUX bus voltage is maintained at V_{REF} . The sensor bleeds off the current I_{SENSOR} from the modulating capacitor (C_{MOD}), which, in combination with R_{bus} , forms a low-pass filter that attenuates precharge-switching transients at the sigma-delta converter input.

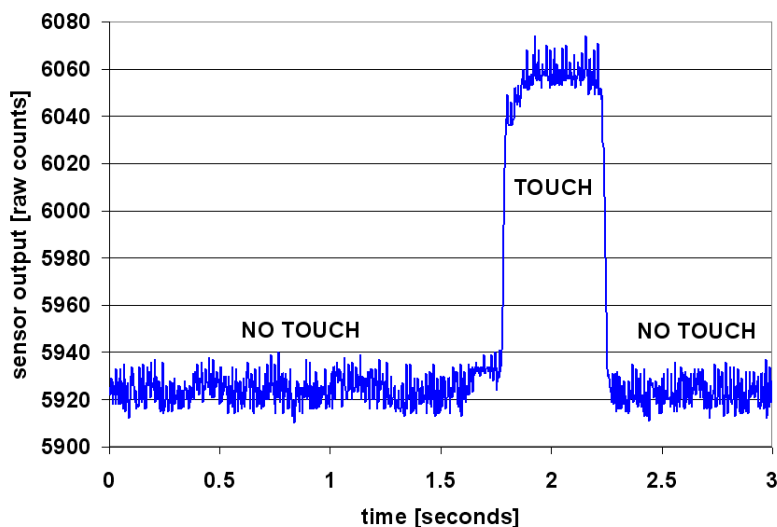
Figure 2-5. CSD PLUS Block Diagram



To maintain the AMUX bus voltage at V_{REF} , the sigma-delta converter matches the IDAC output current to I_{SENSOR} by controlling the bit stream duty cycle. The sigma-delta converter stores the bit stream over the duration of a sensor scan, and the accumulated result is a digital raw count output, which is directly proportional to C_X . This raw count is interpreted by high-level algorithms to resolve the sensor state. Figure 2-6 plots the CSD PLUS 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 raw count level from the steady state to a predetermined threshold, the high-level algorithms can determine whether the sensor is in an ON (Touch) or OFF (No Touch) state. To learn more about raw counts, finger threshold, and signal-to-noise ratio (SNR), refer to the [Getting Started with CapSense](#) guide.

Figure 2-6. Raw Counts during Finger Touch



2.3 CapSense Tuning

Optimal CapSense system performance depends on the board layout, sensor dimensions, overlay material, and application requirements. These factors are discussed in [CapSense Layout Guidelines](#). In addition to these factors, the CSD PLUS sensing method requires a few hardware and software parameters to be set for robust operation. The process of setting the optimum value for these parameters is called manual tuning. Manual tuning is a laborious and time consuming process. Cypress' SmartSense Auto-Tuning algorithm eliminates the manual tuning process by automatically setting all of the hardware and software parameters.

2.4 SmartSense Auto-Tuning

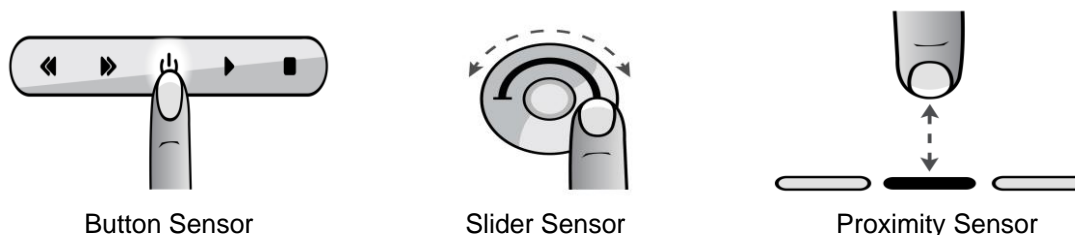
Tuning the touch-sensing user interface is critical for proper system operation and a pleasant user experience. The typical design flow involves tuning the sensor interface in the initial design phase, during system integration, and finally production fine-tuning before the production ramp. Tuning is an iterative process and can be time-consuming.

SmartSense Auto-Tuning is a CapSense algorithm that automatically configures capacitive button parameters for optimum performance. It continuously compensates for system, manufacturing, and environmental changes, and simplifies the user interface development cycle. It is easy to use and reduces design cycle time by eliminating manual tuning during the prototype and manufacturing stages. SmartSense Auto-Tuning tunes each CapSense button automatically at power up and maintains optimum button performance during runtime. SmartSense Auto-Tuning adapts for manufacturing variations in PCBs and overlays. It automatically tunes out noise from various sources such as LCD inverters, AC lines, and switch-mode power supplies. Refer to the [Getting Started with CapSense](#) guide to understand how the SmartSense algorithm works.

2.5 Sensor Types

CY8CMBR3xxx supports three types of capacitive sensors - buttons, sliders, and proximity sensors, as [Figure 2-7](#) shows.

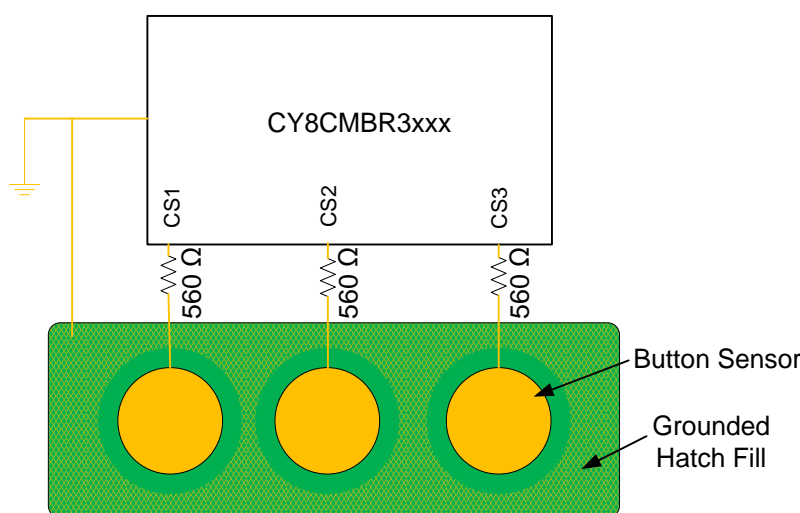
Figure 2-7. Types of Capacitive Sensors Supported in CY8CMBR3xxx



2.5.1 Buttons (Zero-Dimensional)

CapSense buttons replace mechanical buttons in a wide variety of applications such as home appliances, medical devices, lighting controls, and many other products. It is the simplest type of CapSense sensor, consisting of a single sensor. A CapSense button gives one of two possible output states: active (finger is present) or inactive (finger is not present). These two states are also called ON and OFF states respectively. A simple CapSense button consists of a circular copper pad. It connects to a pin marked for CapSense button functionality on the CY8CMBR3xxx controller as shown in [Figure 2-8](#). In CY8CMBR3xxx, a CSx pin indicates that a button sensor can be connected to it. The button is surrounded by grounded copper hatch to isolate it from other buttons and traces. A circular gap separates the button pad and the ground hatch. See the [General Layout Guidelines](#) section for layout recommendations.

Figure 2-8. Simple CapSense Buttons



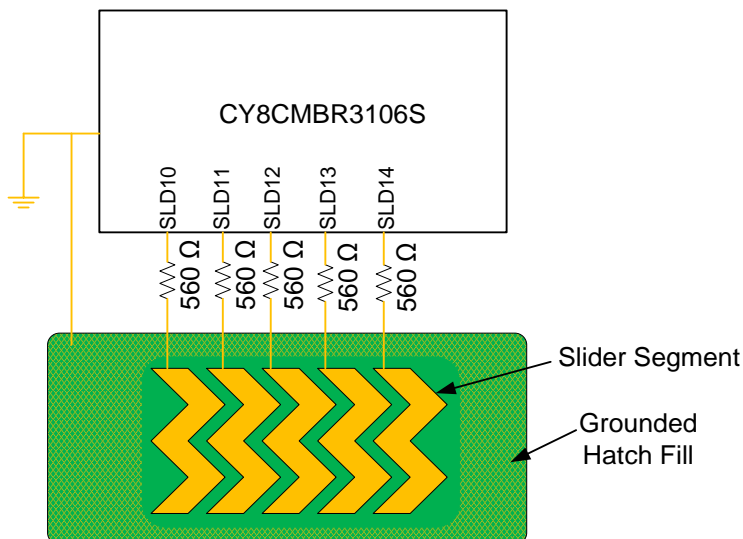
2.5.2 Sliders (One-Dimensional)

Sliders are used when the required output is in the form of a gradual increment or decrement. Examples include lighting control (dimmer), volume control, graphic equalizer, and speed control. A slider consists of a one-dimensional array of capacitive sensors called segments, which are placed adjacent to one another. Touching one segment also results in partial activation of adjacent segments. The firmware processes the raw counts from the touched segment and nearby segments to calculate the position of the geometric center of the finger touch, which is known as the **centroid position**.

The actual resolution of the calculated centroid position is much higher than the number of segments in a slider. For example, a slider with five segments can resolve at least 100 physical finger positions. This high resolution gives smooth transitions of the centroid position as the finger glides across a slider. In a linear slider, the segments are arranged inline, as [Figure 2-9](#) shows.

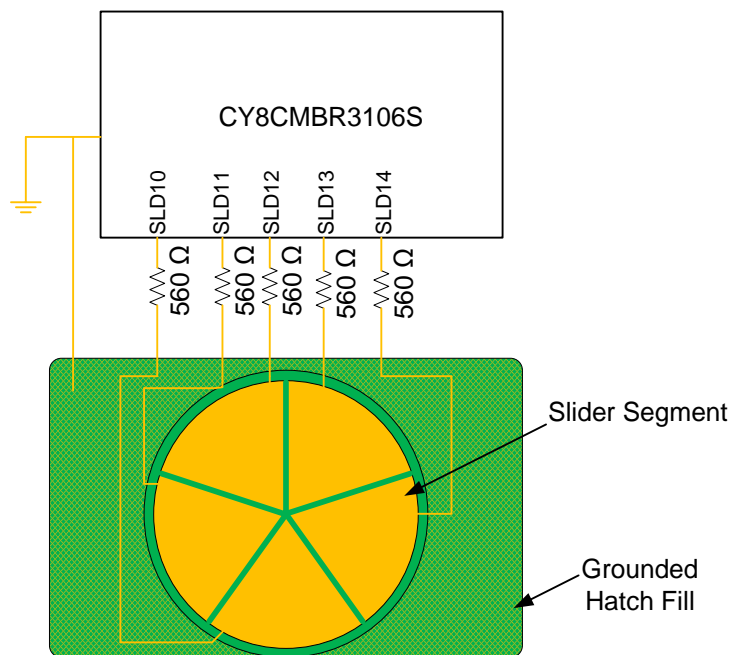
Each slider segment connects to a pin marked for slider functionality on the CY8CMBR3106S controller. In CY8CMBR3106S, a SLDxx pin indicates that a slider segment can be connected to it. A zigzag pattern (double chevron) is recommended for slider segments. This layout ensures that when a segment is touched, the adjacent segments are also partially touched, which helps in the calculation of the centroid position. See the [General Layout Guidelines](#) section for layout recommendations.

Figure 2-9. Linear Slider



Radial sliders are similar to linear sliders except that radial sliders are continuous. [Figure 2-10](#) shows a typical radial slider.

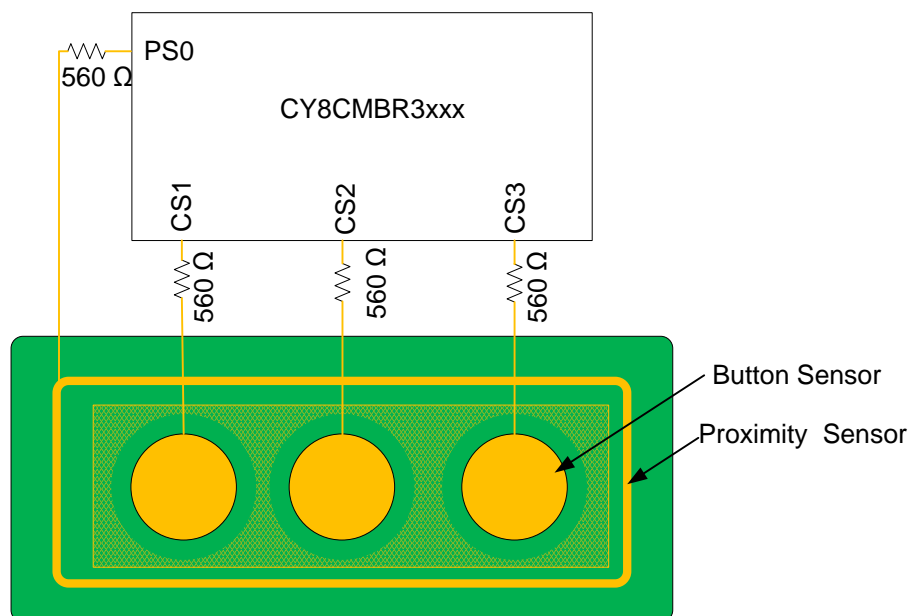
Figure 2-10. Radial Slider



2.5.3 Proximity (Three-Dimensional)

Proximity sensors detect the presence of a hand in the three-dimensional space around the sensor. However, the actual output of the proximity sensor is an ON/OFF state similar to a CapSense button. Proximity sensing can detect a hand at a distance of several centimeters to tens of centimeters depending on the sensor size and construction. Each proximity sensor connects to a pin marked for proximity sensing functionality on the CY8CMBR3xxx controller. In the CY8CMBR3xxx controller, a PSx pin indicates that a proximity sensor can be connected to it. Proximity sensing requires electric fields that are projected to much larger distances than buttons and sliders, requiring a large sensor area. The proximity sensor can either be a trace on the PCB or a simple wire loop, as [Figure 2-11](#) shows. See the [Layout Guidelines for Proximity Sensor](#) section for more details.

Figure 2-11. Proximity Sensor



2.6 Liquid Tolerance

In a CapSense design, false sensing of touch may happen due to the presence of a film of liquid or droplets on the touch surface. Home appliances, automotive applications, and industrial applications are examples of systems that must perform in environments that include water, ice, and humidity changes. For such applications, a shield electrode and guard sensor can provide robust touch sensing.

If your application requires tolerance to liquid droplets and moisture, use a shield electrode in your design. If your application also requires tolerance to liquid flow on the touch surface, use a guard sensor together with a shield electrode. The inactive sensors are connected to ground even when the SHIELD is enabled. Therefore, it is recommended to place the button sensors far apart such that liquid droplets present on one sensor do not cause false triggering of neighboring sensors.

Note: Liquid tolerance is not supported for slider sensors.

Diagram illustrating the connection of the CY8CMBR3xxx device to a sensor array. The device has pins labeled GUARD, SH, CS1, CS2, and CS3. Each of the CS pins (CS1, CS2, CS3) is connected to a 560 Ω resistor, which is then connected to a corresponding yellow circular sensor element on a green PCB. The GUARD pin is connected to a 560 Ω resistor, which is connected to a yellow rectangular guard sensor. The SH pin is connected to a 560 Ω resistor, which is connected to a yellow rectangular shield electrode. Labels on the right point to the Button Sensor, Guard Sensor, and Shield Electrode.

Shield electrodes protect CapSense button sensors from detecting false touches caused by liquid drops. When liquid drops are present on the overlay surface, the coupling between the shield electrode and the sensor pad is increased by C_{WD} , as [Figure 2-13](#) shows.

A 3D schematic diagram illustrating the experimental setup. A blue water droplet is shown on the top surface of a PCB. A yellow shield electrode is positioned on the bottom surface of the PCB, directly beneath the droplet. The PCB is composed of a green base layer and a blue top layer. Labels with arrows point to the 'Water Drop', 'Overlay' (the blue top layer), 'Shield Electrode' (the yellow electrode), and 'PCB' (the green base layer). A capacitor symbol is labeled C_{WD} , representing the capacitance between the droplet and the electrode.

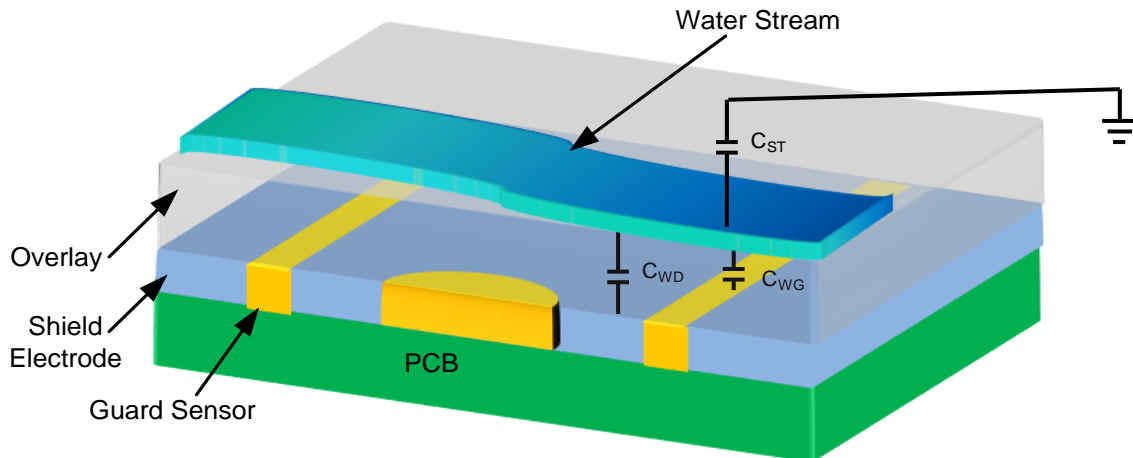
- The purpose of the shield electrode is to set up an electric field around the touch sensors that helps attenuate the effects of liquid. The shield electrode works by mirroring the voltage of the touch sensor on the shield. Since the shield electrode and sensors are driven with the same signals, the potential difference between them is zero and any capacitance between the sensors and the shield electrode cannot cause a charge transfer. Therefore, any liquid film or droplets that are present partially on both the sensor and shield area do not change the sensor capacitance, allowing CapSense to work in the presence of liquid film or droplets.

2.6.2 Guard Sensor

A guard sensor is a copper trace that surrounds all the sensors on the PCB, as [Figure 2-12](#) shows. It is used to detect the presence of streaming liquid. When a liquid stream is present on the sensing surface, a large capacitance (C_{ST}) is added to the system, as [Figure 2-14](#) shows. This capacitance may be several times larger than C_{WD} . Because of this, the effect of the shield electrode is completely masked, resulting in the same or even higher raw counts measured by the sensor than a finger touch. In this situation, when the guard sensor detects a liquid stream, it blocks other sensors from triggering. Therefore, touch cannot be detected when streaming liquid is present.

In MBR3, the guard sensor (if enabled) is scanned only in [Active](#) and [Look-for-Touch](#) mode. Therefore, when the device is in [Look-for-Proximity](#) mode, false proximity triggering can occur when a liquid stream is present.

Figure 2-14. Capacitance Measurement With Liquid Stream



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

To make your design liquid-tolerant, follow these steps:

1. If your application requires tolerance to liquid droplets and moisture, choose a CY8CMBR3xxx controller that has the shield feature. If your application also requires tolerance to liquid flow, choose a CY8CMBR3xxx controller that has both shield and guard features. Refer to [Table 1-1](#) to select the controller.
2. Follow the schematic and layout guidelines explained in the [Layout Guidelines for Liquid Tolerance](#) section to construct the shield electrode and guard sensor.
3. Enable the shield and guard sensor features in [EZ-Click](#) as explained in the [Shield and Guard Sensor](#) configuration section.
4. Tune the guard sensor as explained in the [Tuning the Buttons, Sliders, and Guard Sensor](#) section.

3. CapSense Schematic Design



The CY8CMBR3xxx controller supports buttons, sliders, and proximity sensors on dedicated pins. Also, the number of buttons, sliders, and proximity sensors supported is fixed in this controller. Selecting the sensor pins is the first step in designing a CapSense application. This chapter provides guidelines on selecting sensor pins for designing a CapSense schematic.

3.1 Schematic Design Considerations

Designing a schematic with CY8CMBR3xxx involves the following steps:

1. Special output pin selection
2. Sensor pin selection
3. GPO pin selection
4. Power and ground connections
5. I²C lines connection

This chapter explains how to select the special output, sensor, and GPO pins. Refer to the [CY8CMBR3xxx datasheet](#) for information on power, ground, and I²C lines connection.

3.1.1 Selecting the Special Output Pins

The shield electrode, guard sensor, buzzer, and host interrupt pins are called special output pins (SPOs). These features are multiplexed with sensor pins and are supported only on dedicated pins. Therefore, it is important to select these special output pins before connecting sensor pins.

For example, consider a CapSense design which requires seven buttons, buzzer output, and the liquid-tolerance feature. As listed in [Table 1-1](#), the CY8CMBR3116 and CY8CMBR3110 controller fit the design requirement. For illustration purpose, CY8CMBR3110 is selected. The first step in the schematic design is to select the Buzzer (BUZ), Shield (SH), and Guard (GUARD) pins because these pins are multiplexed with the sensor and GPO pins, as [Figure 3-1](#) shows. Refer to the “Pinouts” section in the [CY8CMBR3xxx datasheet](#) to know the multiple features supported on the same pin. After the special output pins are selected, select the button sensors CS0, CS1, CS3, CS5, CS6, CS7, and CS8, as [Figure 3-2](#) shows.

Figure 3-1. Multiple Features Supported on a Single Pin

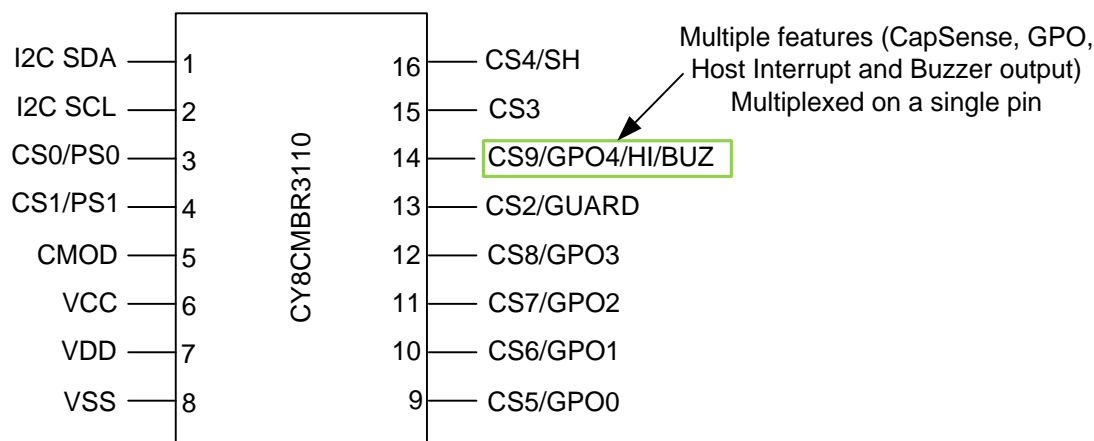
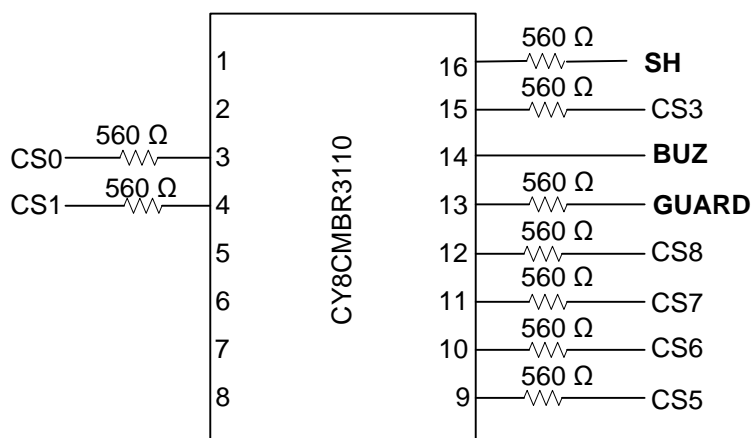


Figure 3-2. Special Output (SPO) Pin Selection



3.1.2 Selecting the Sensor Pin

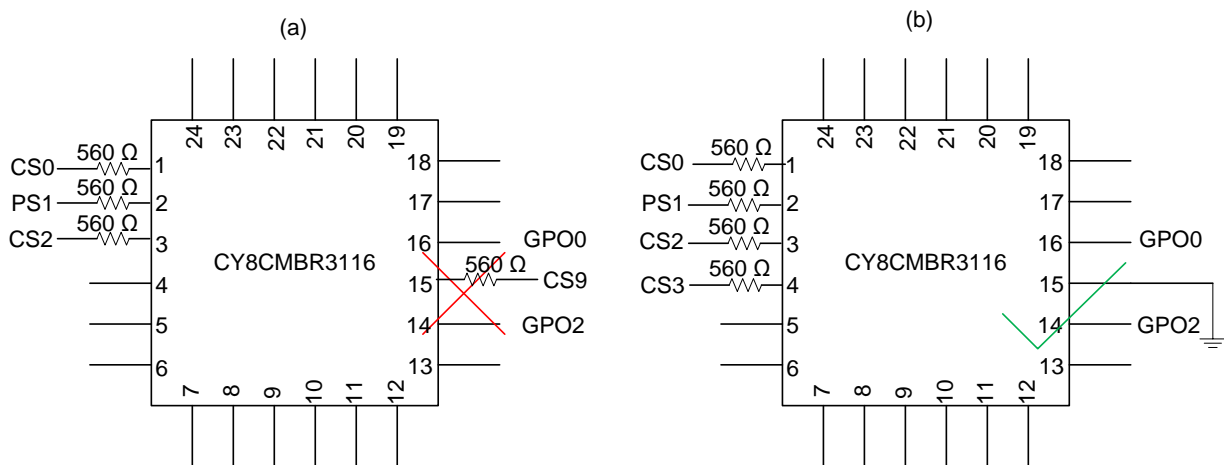
While selecting the sensor pins, do not select the pins (CSx, SLDx, PSx, or GUARD) that are next to or in between the GPO pins. Selecting a sensor pin next to a GPO pin might result in cross-talk. If the sensor trace and GPO trace run parallel to each other, then the GPO switching may cause the sensor next to GPO pin to false trigger. Refer to the “PCB Layout Guidelines” section in [Getting Started with CapSense](#) guide for guidelines to avoid crosstalk.

For example, [Figure 3-3 \(a\)](#) shows a schematic in which the sensor CS9 is in between GPO0 and GPO2 pin. In this case, if the CS9 trace runs parallel to the GPO0 or GPO2 trace, switching of GPO0 or GPO2 might cause the sensor CS9 to false-trigger. This false trigger can be avoided by choosing a sensor pin CS3, which is not next to a GPO pin as shown in [Figure 3-3 \(b\)](#) or by ensuring that the sensor trace and GPO trace do not run in parallel,.

If your design requires a proximity sensor, select the proximity sensor pins (PSx) before selecting the button (CSx) pins because the proximity sensors are supported only on two pins.

To increase noise immunity, connect a 560-Ω series resistor to the sensor pins. If any CapSense pin is not used, it is recommended to configure it as sensor pin and connect it to ground. Refer to the [Sensor Configuration](#) section for details on how to configure a pin as CapSense input.

Figure 3-3. Sensor Pin Selection Guidelines



3.1.3 Selecting the Slider Pins

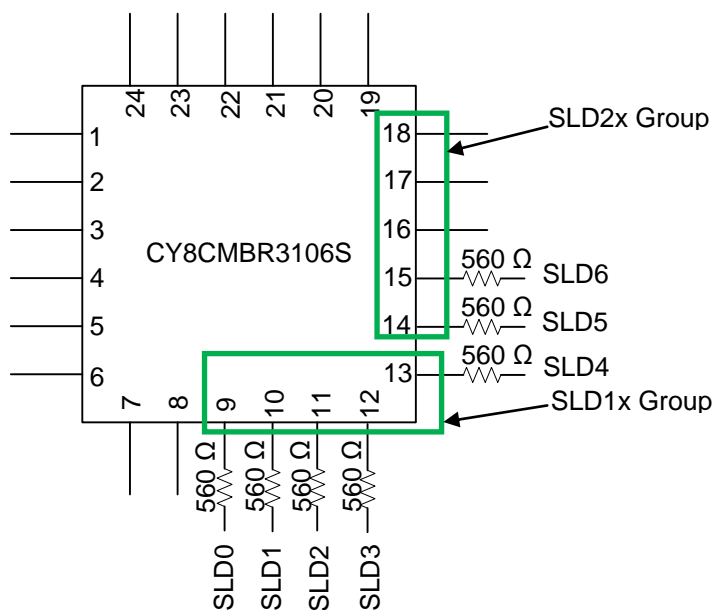
To select the slider pins, refer to [Table 3-1](#).

Table 3-1. Slider Pin Selection Guidelines

Number of Segments	Recommendation
Less than or equal to five	Select the pins from the SLD1x group. SLD10 should be the first slider segment and SLD14 should be the fifth segment.
More than five	Select all the five pins from the SLD1x group for first five slider segments and then select the pins from the SLD2x group for the remaining segments. SLD10 should be the first segment of the slider, SLD20 should be the sixth segment, and SLD24 should be the tenth segment.

[Figure 3-4](#) shows an example of a schematic designed for a slider with seven segments. In this schematic, the slider segments SLD0 to SLD4 use the pins from the SLD1x group of the controller and the SLD5 and SLD6 segments use the pins from the SLD2x group. The remaining pins in the SLD2x group (SLD22, SLD23, and SLD24) can be configured as buttons. If unused, they should be configured as buttons and connected to ground¹.

Figure 3-4. Slider Pins Selection

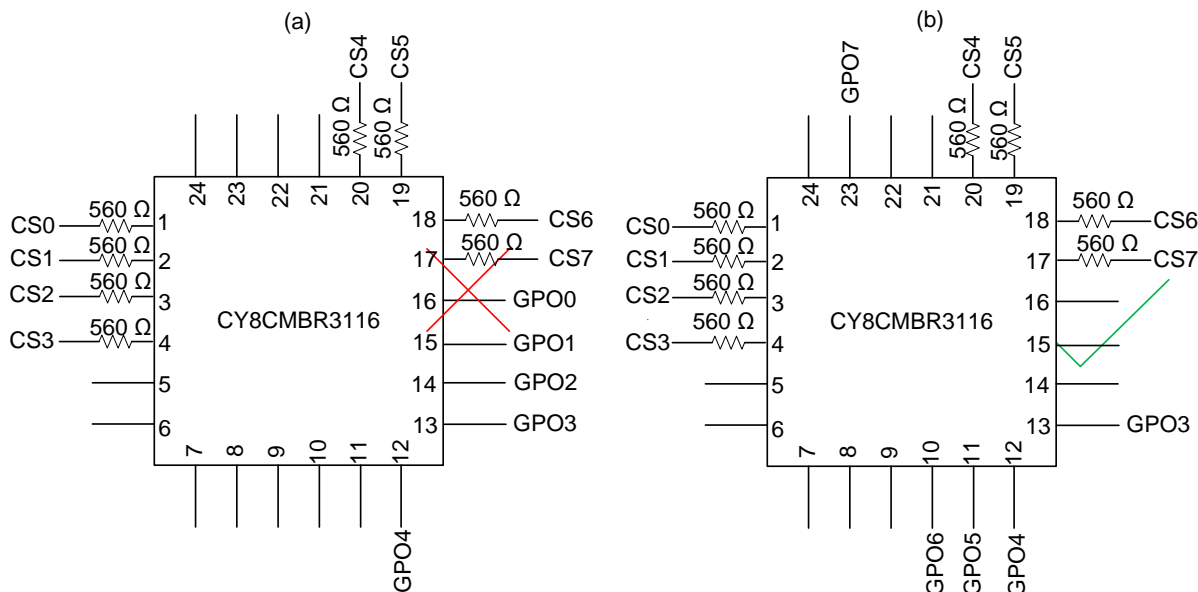


¹ Pin#18, that is, CS15/SLD24 in CY8CMBR3106S, should be left floating when it is not used. This I/O functions as a reset (AXRES) pin during boot-up. Make certain that this pin is not grounded during power-up for the device to boot up properly. After boot-up, this I/O functions as indicated by the pin name.

3.1.4 Selecting the GPOs

If the GPOs are controlled by the host controller, any available GPOs can be used to provide touch feedback. It is recommended to select GPOs in the order GPO7 to GPO0. For example, if your design requires eight buttons and five GPOs, select CS0-CS7 for buttons and GPO7-GPO3 for GPOs. This avoids selecting a sensor pin, which is next to a GPO pin, as [Figure 3-5](#) shows.

Figure 3-5. Selecting GPO Pins When GPO Host Control Is Enabled



If GPO Host Control is disabled, then each sensor from CS0-CS7 will be mapped to GPO0 to GPO7 respectively. In this case, select the GPOs corresponding to the selected sensor. For example, when GPO host control is disabled and CS0, CS1, and CS2 are selected as buttons, select GPO0, GPO1, and GPO2 as GPO pins to drive LEDs.

The CY8CMBR3xxx controller supports both sourcing and sinking configuration for driving LEDs on GPO pins. Since the sinking capability of GPO is much higher than the sourcing capability, it is recommended to connect the LED in sinking configuration. Refer to the DC I/O port specifications in the [CY8CMBR3xxx datasheet](#) for maximum sourcing and sinking current limits.

To limit the current, add a series resistor of appropriate value depending on the supply voltage. If your design requires a sinking current of more than the maximum limit, connect an external transistor to the GPO pin and configure the GPO in the strong drive mode using [EZ-Click](#).

3.2 Schematic Checklist

Use the checklist in [Table 3-2](#) to verify your schematic design.

Table 3-2. Schematic Design Checklist

No.	Category	Recommendations/Remarks
1.	Decoupling capacitor on the VDD ² pin	Connect a 1-μF and 0.1-μF capacitor in parallel
2.	Decoupling capacitor on the VCC ³ pin	Connect a 0.1 μF capacitor if VDD > 1.8 V If VDD is 1.71 V to 1.89 V, short this pin to VDD
3.	Decoupling capacitor on the VDDIO ⁴ pin	Connect a 1 μF and 0.1 μF capacitor in parallel
3.	C _{MOD}	Connect a 2.2 nF, 5 V, X7R or NPO capacitor
4.	Series resistor on CSx, PSx, SLDx, GUARD, and SH pins	Connect a 560-Ω resistor in series with the pin
5.	Series resistor on I ² C lines	Connect a 330-Ω resistor in series with the SDA ⁵ and SCL ⁶ lines
6.	Pull-up on I ² C lines	Select an appropriate value depending on the I ² C interface voltage
7.	Pull-up on the HI line	Select an appropriate value depending on the interface voltage
8.	Unused CapSense pins ⁷ (CSx, PSx, SLDx, and GUARD)	Refer to the Pinouts section in the CY8CMBR3xxx Datasheet for details on the pin connection when the pin is not used.

3.3 Sample Schematics

3.3.1 Touch Buttons in Mobile Phones

Touch buttons are used in mobile phones for quick access to various applications. A typical mobile phone has up to four touch buttons and requires LED backlighting.

The schematic shown in [Figure 3-6](#) is an example of how to implement touch buttons in mobile phones.

In [Figure 3-6](#), the CY8CMBR3108 controller is configured in the following manner:

- CS0-CS3: CapSense buttons
 - All CapSense pins must have a 560-Ω series resistance (placed close to the chip) for improved noise immunity.
- GPO0-GPO3: To external LEDs
 - LEDs are connected in sinking mode because the CY8CMBR3xxx controller has high sink current capability.
 - Series resistances are connected to limit the GPO current to be with I_{IL} limits.
- CMOD pin: 2.2 nF to ground
- VCC pin: 0.1 μF to ground.
 - Supply voltage is 3.3 V and therefore, the VCC pin is not shorted to the VDD pin.
- VDD pin: To external supply voltage
 - 1 μF and 0.1 μF decoupling capacitor connected to VDD pin.
- VDDIO pin: To supply voltage, which is ≤ VDD
 - VDDIO powers I²C and HI lines.
- I²C_SCL and I²C_SDA pins: 330 ohms to the I²C header

² VDD: This is the primary supply to the chip and can be powered from 1.8 V ±5% or 1.8 to 5.5 V

³ VCC: This is the internal regulator output, which powers the core and capacitive sensing circuits

⁴ VDDIO: This is the supply input for I²C SDA, I²C SCL, and HI lines in CY8CMBR3108. The signal levels of these I/Os are referenced with respect to VDDIO. For more details on power supply information refer to [CY8CMBR3xxx Datasheet](#)

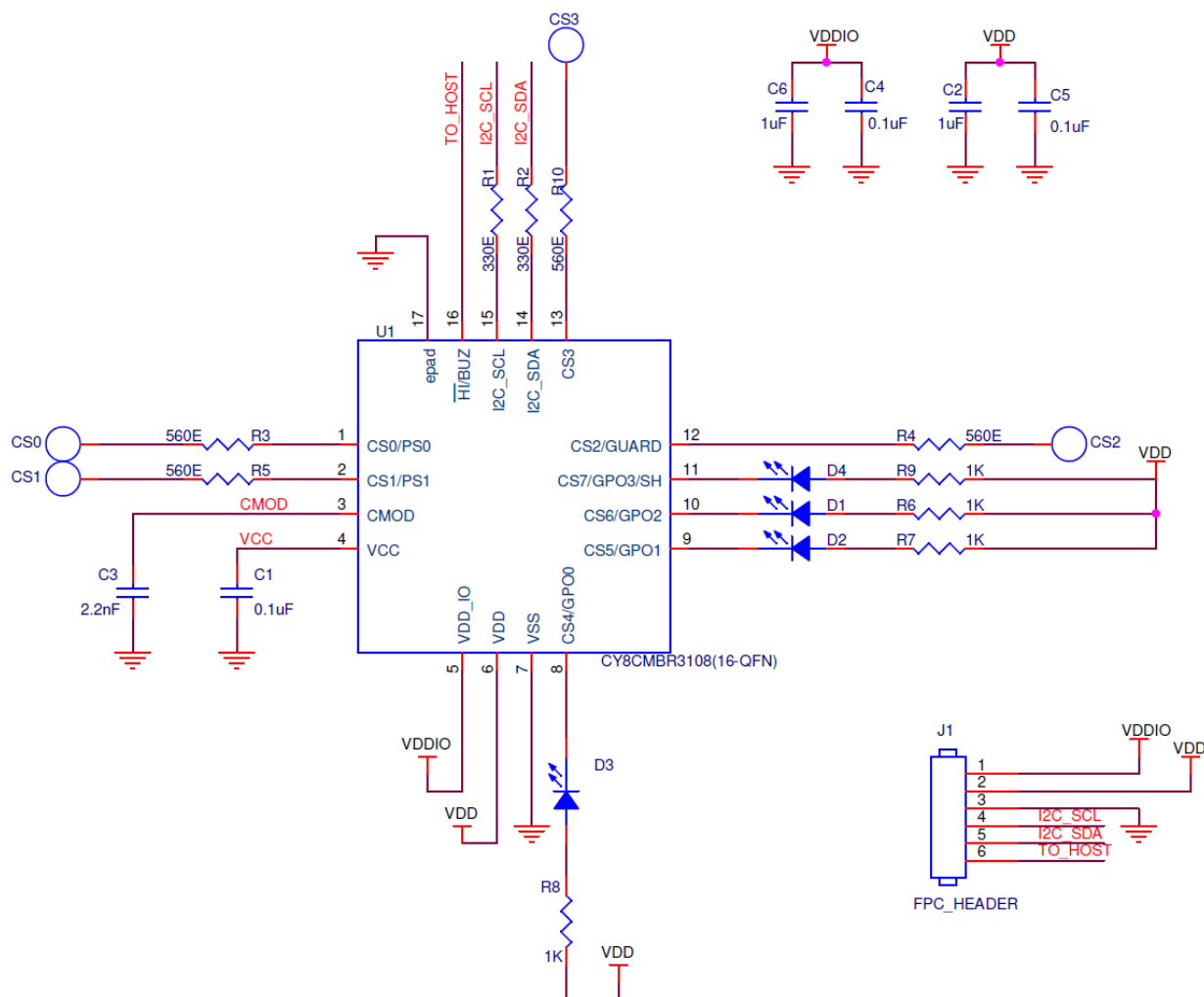
⁵ SDA – Serial Data (SDA) line is used to send data between the I²C master and the I²C slave

⁶ SCL – Serial Clock (SCL) line is used to synchronize the I²C slave with the I²C master

⁷ Refer to the Pinouts section in the [CY8CMBR3xxx Datasheet](#) for details on the pin connection when not used

- HI pin: To host
 - To prompt the host to initiate an I²C transaction for reading the changed sensor status.

Figure 3-6. Schematic for Touch Buttons in Mobile Phone


Notes:

- Place capacitors C1, C2, C3, C4, C5, C6, and 560-Ω series resistors as close as possible to the CapSense IC
- I²C lines (SDA and SDL) and HI requires external pull-up resistors on the host side.

Table 3-3. Bill of Materials

No.	Qty.	Reference	Description	Manufacturer	Manufacturer Part No.
1	2	C1, C4, C5	CAP CER 0.1 μF 10 V 10% X5R 0402	Taiyo Yuden	LMK105BJ104KV-F
2	1	C2, C6	CAP CER 1 μF 10 V 10% X5R 0402	TDK Corporation	C1005X5R1A105K050BB
3	1	C3	CAP CER 2200 pF 50 V 5% X7R 0402	Murata	GRM155R71H222JA01J
4	2	R1, R2	RES 330 Ω 1/10 W 5% 0402 SMD	Panasonic Electronic Components	ERJ-2GEJ331X
5	4	R3, R4, R5, R10	RES 560 Ω 1/16 W 5% 0402 SMD	Yageo Corporation	RC0402JR-07560RL

No.	Qty.	Reference	Description	Manufacturer	Manufacturer Part No.
6	4	R6, R7, R8, R9	RES 1 k Ω 1/10 W 5% 0402 SMD	Panasonic Electronic Components	ERJ-2GEJ102X
7	1	U1	CY8CMBR3108-LQXI-16QFN	Cypress Semiconductor	CY8CMBR3108-LQXI-16QFN
8	4	D3, D2, D1, D4	LED CHIP ANGAN HER RA 0603	Avago Technologies US Inc.	516-2277-1-ND

3.3.2 Touch Buttons in Home Appliances

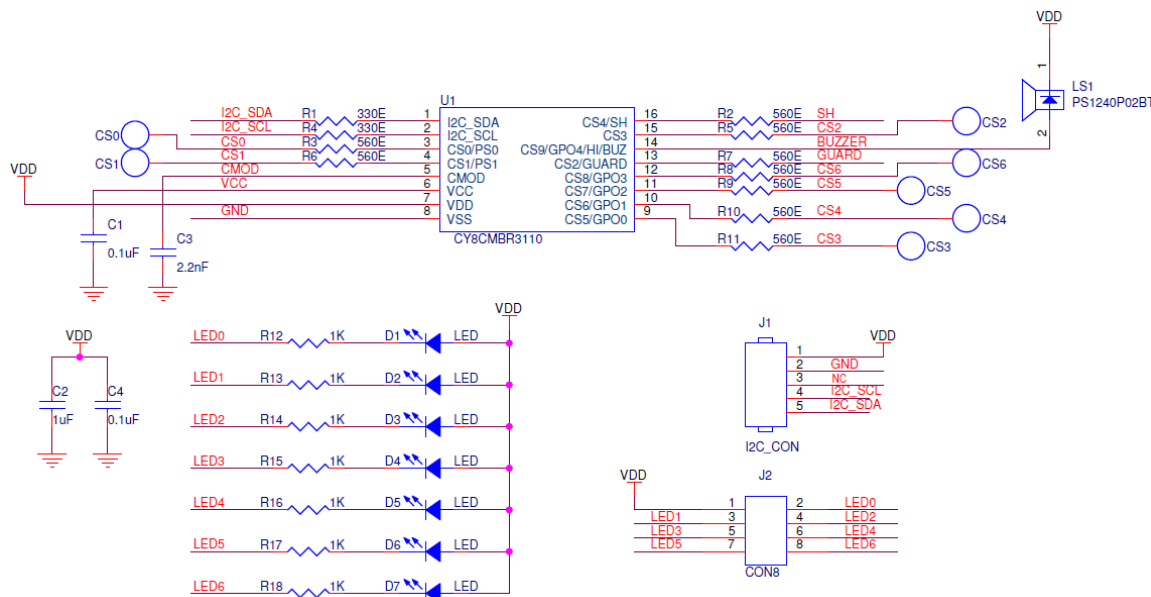
Touch buttons are used in home appliances to replace mechanical buttons. The key requirements in these applications are liquid-tolerant CapSense buttons and audio feedback.

The schematic shown in [Figure 3-7](#) is an example of implementing touch buttons in an induction cooktop.

In [Figure 3-7](#), the CY8CMBR3110 controller is configured in the following manner:

- CS0, CS1, CS3, CS5, CS6, CS7, CS8: CapSense buttons
 - All CapSense pins must have a 560- Ω series resistance (placed close to the chip) for improved noise immunity.
- SH: Shield pin
 - The shield pin must have a 560- Ω series resistance (placed close to the chip) for improved noise immunity.
- GUARD: Guard sensor
 - The guard sensor must have a 560- Ω series resistance (placed close to the chip) for improved noise immunity.
- BUZZER: To connect 1-pin buzzer
- CMOD pin: 2.2 nF to ground
- VCC pin: 0.1 μ F to ground (Device)
 - Supply voltage is 5 V and therefore, the VCC pin is not shorted to the VDD pin.
- VDD pin: To external supply voltage
 - 1 μ F and 0.1 μ F decoupling capacitor connected to the VDD pin.
- I²C_SCL and I²C_SDA pins: 330 Ω to the I²C header.

Figure 3-7. Schematic for Touch Buttons in Induction Cooker



Notes:

- Place the capacitors C1, C2, C3, C4, and 560- Ω series resistors as close as possible to CapSense IC

- The LEDs shown in this schematic are driven by the host controller based on the sensor status.
- I²C lines (SDA and SDL) require external pull-up resistors on the host side.

Table 3-4. Bill of Materials

No.	Qty.	Reference	Description	Manufacturer	Manufacturer Part No.
1	2	R1, R4	RES 330 Ω 1/10 W 5% 0603 SMD	Yageo Corporation	RC0603JR-07330RL
2	9	R3, R6, R2, R5, R7, R8, R9, R10, R11	RES 560 Ω 1/10 W 5% 0603 SMD	Yageo Corporation	RC0603JR-07560RL
3	1	C1, C4	CAP CER 0.1 μ F 10 V 10% X5R 0603	AVX Corporation	0603ZD104KAT2A
4	1	C2	CAP CER 1 μ F 10 V 10% X5R 0603	TDK Corporation	C1608X5R1A105K080AC
5	1	C3	CAP CER 2200 pF 50 V 5% NP0 0603	Murata	GRM1885C1H222JA01D
6	1	U1	CY8CMBR3110-SX2I-16SOIC	Cypress Semiconductor	CY8CMBR3110-SX2I-16SOIC
7	1	LS1	Buzzer Piezo 4 kHz 12.2 mm PC MNT	TDK Corporation (VA)	PS1240P02BT
8	7	D1, D2, D3, D4, D5, D6, D7, D8	LED Red Clear 1206 REAR MNT SMD	Stanley Electric Co	BR1111R-TR
9	7	R12, R13, R14, R15, R16, R17, R18	RES 1.0 k Ω 1/10 W 5% 0603 SMD	Yageo	RC0603JR-071KL
10	1	J1	CONN HDR 100" SNGL PCB 5POS	Sullins Connector Solutions	SWR25X-NRTC-S05-ST-BA
11	1	J2	CONN HEADER .100" DUAL STR 80POS	Sullins Connector Solutions	PRPC040DAAN-RC

3.3.3 Proximity Sensing

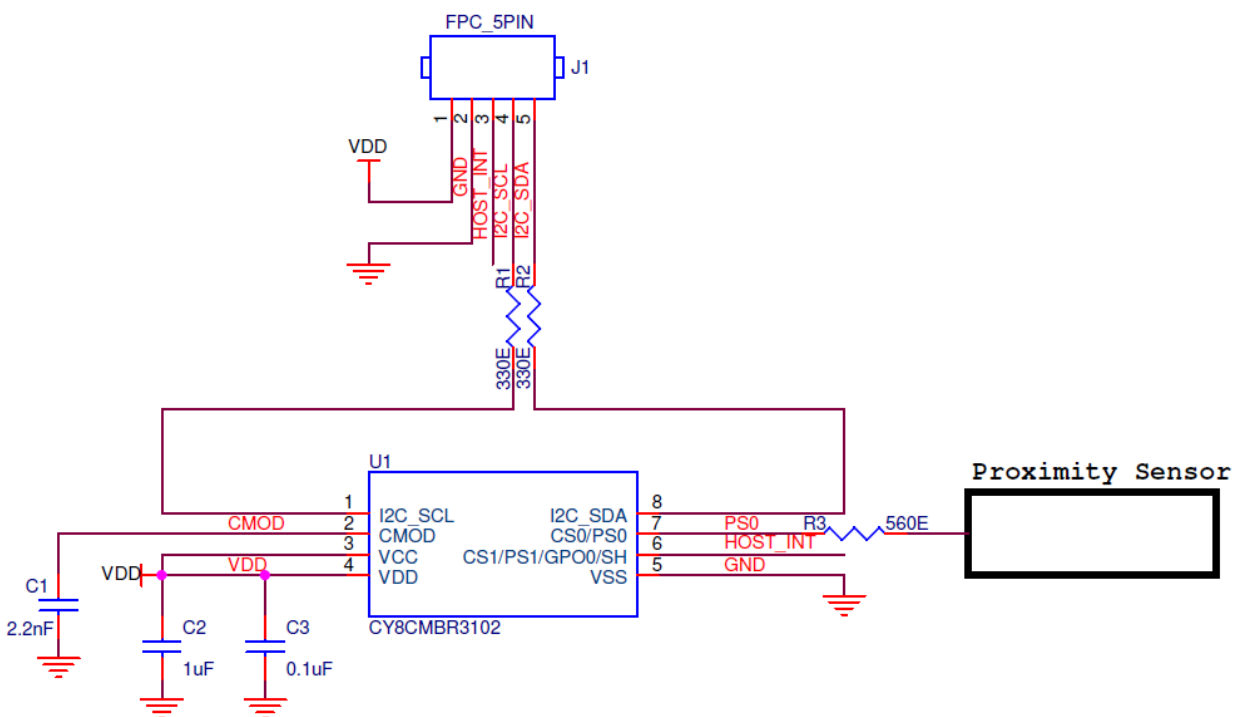
Proximity sensing is used in mobile phones/tablets to turn OFF the display/disable keypad when the user is in a call and the phone is close to the ear. Proximity sensing is also used in mobile phones/tablets for Specific Absorption Rate (SAR) regulation.

The schematic shown in [Figure 3-8](#) is an example of implementing proximity sensing in mobile phones/tablets.

In [Figure 3-8](#), the CY8CMBR3102 controller is configured in the following manner:

- PS0: Proximity Sensor
 - All proximity sensor pins must have a 560 Ω series resistance (placed close to the chip) for improved noise immunity.
- GPO0: To Host
 - GPO is used to indicate proximity sensor status to host controller
- CMOD pin: 2.2 nF to ground
- VCC pin: Shorted to the VDD pin for 1.71 to 1.89 V operation
- VDD pin: To external supply voltage
 - 1 μ F and 0.1 μ F decoupling capacitor connected to the VDD pin.
- I²C_SCL and I²C_SDA pins: 330 Ω to the I²C.

Figure 3-8. Schematic for Proximity-Sensing Application


Notes:

- Place the capacitors C1, C2, C3, and a 560-Ω series resistor as close as possible to the CapSense IC
- I²C lines (SDA and SCL) and the Host_INT pin require a pull-up resistor on the host side.

Table 3-5. Bill of Materials

No.	Qty.	Reference	Description	Manufacturer	Manufacturer Part No.
1	1	C1	CAP CER 2200 pF 50 V 5% X7R 0402	Murata	GRM155R71H222JA01J
2	1	C2	CAP CER 1 μF 10 V 10% X5R 0402	TDK Corporation	C1005X5R1A105K050BB
3	1	C3	CAP CER 0.1 μF 10 V 10% X5R 0402	Taiyo Yuden	LMK105BJ104KV-F
4	1	R3	RES 560 Ω 1/16 W 5% 0402 SMD	Yageo Corporation	RC0402JR-07560RL
5	2	R1, R2	RES 330 Ω 1/10 W 5% 0402 SMD	Panasonic Electronic Components	ERJ-2GEJ331X
6	1	U1	CY8CMBR3102-SX11-8SOIC	Cypress Semiconductor	CY8CMBR3102-SX11-8SOIC

4. CapSense Layout Guidelines



In a typical CapSense application, capacitive sensors are constructed with traces on a FR4/FR2 or flexible printed circuit (FPC). Designing the CapSense layout is an important step in the design phase and following the CapSense layout best practices will help your design achieve higher noise immunity, lower parasitic capacitance (C_P), and higher SNR.

The following factors should be considered while designing the CapSense layout:

- Parasitic capacitance (C_P): A high sensor C_P makes it more difficult to sense small changes in sensor capacitance and thereby reduces sensitivity. The CapSense layout should be such that the C_P of the sensor is kept to the minimum.
- Trace length: Longer trace lengths add to the sensor C_P and reduce the sensor sensitivity. Also, a long trace acts like an antenna and reduces the noise immunity of the sensor.
- Sensor dimensions: The sensor dimensions depend on the overlay thickness. A thicker overlay requires larger sensor dimensions.
- Power consumption: The power consumption of the sensor depends on the C_P of the sensor. Higher sensor C_P increases the scan time of the sensor, and results in higher overall power consumption. To achieve lower power consumption, reduce the sensor C_P .

To achieve lower C_P , higher SNR, and lower power consumption, follow the layout best practices. The [CY8CMBR3xxx Design Toolbox](#) helps in achieving these goals.

4.1 Design Toolbox

The CY8CMBR3xxx Design Toolbox is a spreadsheet that provides general layout guidelines, estimation of sensor dimensions, parasitic capacitance estimation, average current consumption, and response time of sensors. The design toolbox can be accessed from [EZ-Click](#). The latest version of the design toolbox can be downloaded from [CY8CMBR3xxx Design Toolbox](#).

This toolbox has four different sections:

1. General Layout Guidelines
2. Layout Estimator
3. C_P , Power Consumption, and Response Time Calculator
4. Design Validation

4.1.1 General Layout Guidelines

This sheet provides recommendations on the sensor shape, minimum and maximum sensor dimensions, placement of the sensor, and routing traces on the PCB or FPC. Follow these guidelines for designing the CapSense layout.

[Table 4-1](#) summarizes the layout guidelines for designing a CapSense application. For a detailed review of these guidelines, see the [Getting Started with CapSense](#) guide.

Table 4-1. Layout Recommendations

#	Category	Details	Min	Max	Recommendations/Remarks
1	Button	Button Parasitic Capacitance (C_P)	5 pF	45 pF	Lower the button C_P , higher the sensitivity and lower the power consumption
2		Button shape	NA	NA	Solid round pattern, round with LED hole, or rectangle with round corners
3		Button size	5 mm	15 mm	See the Layout Estimator sheet
4		Button-button spacing	Equal to button ground clearance	NA	8 mm
5		Button-ground clearance	0.5 mm	2 mm	Equal to overlay thickness, but within min/max limits
6	Slider	Slider segment Parasitic Capacitance (C_P)	5 pF	45 pF	Lower the slider segment C_P , higher the sensitivity and lower the power consumption
7		Slider Segment Shape (Linear Slider)	NA	NA	Zigzag pattern (double chevron)
8		Width of Slider Segment	2 mm ⁸	8 mm	8 mm ⁹
9		Height of the Slider segment	7 mm	15 mm	12 mm
10		Clearance between Slider segments	0.5 mm	2 mm	0.5 mm
11		Air gap between the hatch fill and the slider	0.5 mm	2 mm	Equal to overlay thickness, but within the min/max limits
12	Proximity Sensor	Proximity sensor Parasitic Capacitance (C_P)	8 pF	45 pF	Lower the proximity sensor C_P , higher the proximity detection range
13		Proximity sensor shape	NA	NA	<ul style="list-style-type: none"> Circular or rectangular (curved edges) loop on PCB for large area Circular or rectangular solid fill for small area. It is recommended to have a ground loop surrounding the proximity sensor.
14		Proximity sensor trace width	1.5 mm		1.5 mm
15		Clearance between proximity loop and ground loop	1 mm		1 mm
16		Ground loop trace width	1.5 mm		1.5 mm

⁸ The recommended minimum slider segment width depends on the overlay thickness and the overlay material used. Refer to section “Slider Design” in Getting Started with CapSense design guide for information on the minimum slider segment width for a given overlay thickness and overlay material.

⁹ The recommended slider-segment width is based on an average human finger diameter of 9 mm. Refer to the section “Slider-Segment Shape, Width, and Air Gap” in *Getting Started with CapSense design guide* for more details.

#	Category	Details	Min	Max	Recommendations/Remarks
17		Proximity sensor loop diameter	As described in Recommendation Column	As described in Recommendation Column	Loop diameter should at least be equal to required proximity distance if ALP filter is disabled. If ALP filter is enabled, loop diameter should be at least equal to half of required proximity distance. Any ground metal/plane near the proximity loop sensor will decrease the proximity distance.
18	Guard	Guard sensor Parasitic Capacitance (C_P)	5 pF	45 pF	Lower the guard sensor C_P , higher the sensitivity and lower the power consumption
19		Guard sensor shape	NA	NA	Rectangular PCB trace with curved edges and should surround all the button sensor
20		Guard sensor thickness	2 mm		2 mm
21	Ground/ Shield	Ground /Shield flood- top layer	As described in Recommendation Column	As described in Recommendation Column	Hatched pattern 0.17 mm (7 mil) trace and 1.143 mm (45 mil) grid (25% filling)
22		Ground/Shield flood - bottom layer	As described in Recommendation Column	As described in Recommendation Column	Hatched pattern 0.17 mm (7 mil) trace and 1.778 mm (70 mil) grid (17% filling). For CapSense designs on FPC there should not be any ground fill in the bottom layer.
23	Routing	Placement of components in two layer PCB	NA	NA	<ul style="list-style-type: none"> Top layer: Sensors Bottom layer: Device, other components, and traces.
24		Placement of components in four layer PCB	NA	NA	<ul style="list-style-type: none"> Top layer: Sensors Second layer: CapSense traces and V_{DD}. Avoid V_{DD} traces below sensors Third layer: Hatched ground Bottom layer: Device, other components and non-CapSense traces
25		Trace length from sensor pad to device pin		520 mm	520 mm is for FR4 PCB, with a button diameter of 5 mm. For a different design, refer to Layout Estimator sheet. Shorter the trace lower will be the sensor C_P .
26		Trace width (FR4/FR2/FPC)	0.17 mm	0.20 mm	0.17 mm (7 mil)
27		Trace routing	NA	NA	Traces should be routed on the non sensor side. If any non-CapSense trace crosses the CapSense trace, ensure that the intersection is orthogonal.
28		Via position for the sensors	NA	NA	Via should be placed near the edge of the button to reduce trace length, thereby increasing the sensitivity.

#	Category	Details	Min	Max	Recommendations/Remarks
29		Via hole size for sensor traces			10 mil
30		Number of via on sensor trace	1	2	1
31		CapSense series resistor placement		10 mm	Place CapSense series resistors close to the device for noise suppression. CapSense resistors have higher priority than LED resistors. Place them first.
32		Distance between any CapSense trace to ground flood	0.254 mm (10 mil)	0.508 mm (20 mil)	0.508 mm (20 mil)
33		Device placement	NA	NA	Mount the device on the layer opposite to the sensor. The CapSense trace length between the device and sensors should be minimum (see trace length above)
34		LED backlighting	NA	NA	Cut a hole in the sensor pad and use rear mountable LEDs.
35		Board thickness	As described in Recommendation Column	As described in Recommendation Column	Standard board thickness for CapSense FR4 based designs is 1.6 mm
36	Overlay	Overlay thickness	As per design requirement	5 mm	Use layout Estimator sheet to decide overlay. Given maximum limit is for plastic overlay
37		Overlay material	NA	NA	Should be non-conductive material (glass, ABS plastic, Formica, wood, and so on). No air gap should be there between the PCB and overlay. Use adhesive to stick the PCB and overlay.
38		Overlay adhesives	NA	NA	Adhesive should be nonconductive and dielectrically homogenous. 467MP and 468MP adhesives made by 3M are recommended.

4.1.2 Layout Estimator

The Layout Estimator sheet provides recommendations for minimum sensor size and maximum trace length based on the end-system requirements and industrial design, as [Figure 4-1](#) shows.

This sheet takes the following parameters as inputs:

- Overlay thickness: Enter the overlay thickness that will be used in your design. CY8CMBR3xxx supports up to 5 mm thick plastic overlay for a button sensor and up to 2 mm plastic overlay for slider sensor.
- Overlay dielectric constant: Enter the overlay dielectric constant. Refer to Table C in the sheet to know the dielectric constant of several materials. Materials with high dielectric constant results in higher sensitivity of sensors.
- Capacitance of trace per inch: Enter the capacitance of trace per inch by referring to Table C in this sheet. FR4 PCB adds less C_p when compared to FPC.
- Button sensitivity: This parameter decides the button diameter and maximum overlay thickness. Selecting 400 fF tunes the system for lowest sensitivity and selecting 100 fF tunes the system for highest sensitivity.
- Slider sensitivity: If your design has a slider, then enter the sensitivity of the slider. Thicker overlays require lower values of sensitivity.

- Noise condition: The noise condition is a factor in deciding the sensor dimension. If your design is required to operate in a high-noise environment, then select “High”. If the noise condition is not known, select “Medium”.
- Proximity sensor type: If your design requires a proximity sensor, then select the proximity sensor type.
- Proximity distance required: Select the required proximity distance. Valid range is 1-30 cm.
- ALP filter: If your design requires a higher proximity distance with smaller loop size, then select “Enabled” otherwise, select “Disabled”. When the Advanced Low-Pass (ALP) filter is enabled, the response time of the proximity sensor increases.

Based on the inputs entered, the toolbox estimates the following:

- Minimum button diameter required to achieve 5:1 SNR: If the button diameter required to achieve 5:1 SNR is greater than 15 mm, the toolbox shows a warning to reduce the overlay thickness or increase the sensitivity of the sensor. For the CY8CMBR3002 CapSense controller, the sensitivity parameter is fixed at 100 fF. To avoid the sensor being oversensitive, you need to start with the button diameter equal to the recommended minimum. If touch is not detected, increase the button diameter until the touch response is optimum. If the button diameter is too large, it might result in hover effect and turn on even before the finger touches the sensor.
- Minimum slider dimension required to achieve 5:1 SNR: The toolbox shows a warning when the required slider dimensions are out of range. In such cases, reduce the overlay thickness or decrease the sensitivity of the sensor.
- Maximum allowed trace length in the design for button and slider sensors.
- Clearance between slider segments and the button-to-ground clearance.
- Minimum proximity loop diameter required to achieve required proximity distance with 5:1 SNR.

Figure 4-1. Layout Estimator

Layout Estimator

Table A: Layout Estimator

Input Parameters	Value	Units	Comments
Overlay Thickness	2	mm	
Overlay - Dielectric constant	2.8	farad/m	
Capacitance of trace per inch	2	pF	
Button Sensitivity	200	fF	
Slider Sensitivity	100	fF	
Noise Condition	Medium		
Minimum Recommended Button Diameter	8	mm	
Minimum Recommended Slider Width	2	mm	
Minimum Recommended Slider Height	14	mm	
Maximum Trace Length for Button	508	mm	
Maximum Trace Length for Slider Segments	520	mm	
Clearance Between Slider Segments	2	mm	
Button to Ground clearance	2	mm	

Table B: Proximity Sensor Layout Estimator

Input Parameters	Value	Units	Comments
Proximity Sensor Type	PCB TRACE		
Required Proximity Distance	5	cm	
ALP Filter	Disabled		
Minimum Recommended Proximity Sensor Loop Diameter	5	cm	

Table C: Industry Standard Reference Values

Overlay Material	Dielectric constant
Plastic	2.8
Plexi glass	8
Formica	4.6 - 4.9
Glass (Standard)	7.6 - 8.0
Glass (Ceramic)	6
Mylar	3.2
ABS Plastic	3.8 - 4.5
Wood	1.2 - 2.5
Trace and board type	Capacitance per inch in pF
Copper trace, PCB, 2 layer, 64m	2
Copper trace, flex PCB, 2 layer	8

Note: Search online for the dielectric constant of the materials which are not listed in Table C.

	input cells, edit with actuals
	output cells, based on inputs

4.1.3 C_P , Power Consumption, and Response Time Calculator

After the board layout has been completed, the C_P , average current consumption and response time of your design can be estimated using the C_P , Power Consumption, and Response Time Calculator sheet.

4.1.3.1 Estimating the C_P Value

Follow these steps to estimate the C_P of the sensor:

1. In Table A, select the part number from the drop down list.
2. If the part number selected is “CY8C3106S” and sliders are required in the design, enter the number of sliders and number of segments in each slider group.

3. In Table B, select the type of the sensor from the “Sensor Type” column. For a pin that is not connected to any sensor or not applicable for the selected part number, select NC (no connection). Refer to the [CY8CMBR3xxx datasheet](#) for the pin details.
4. Enter the sensor dimensions for the selected sensor. The “Slider Length” column is applicable only for slider segments. If the entered sensor dimension exceeds the maximum range, the toolbox alerts to reduce the sensor dimension. Refer to the [General Layout Guidelines](#) to find the minimum and maximum sensor dimensions.
5. Enter the trace length of each sensor in the “Trace Length” column.
6. When all these parameters are entered, the toolbox estimates the sensor C_P value and displays it in the “Parasitic capacitance (C_P) of sensors” column. If the estimated C_P value is more than 45 pF, then decrease the trace length until the C_P value is less than 45 pF. Reducing the sensor dimension is not recommended if its value is within the maximum limit.

Figure 4-2. Estimating the C_P of Sensors

Table B: Cp Calculator

Sensor	Sensor Dimension				Trace Length		Sensitivity		Parasitic capacitance (Cp) of sensors (Approx)		Comments
	Slider Length		Button Diameter/Slider Width/Proximity Loop								
PS0			10	mm	50	mm	100	fF	7	pF	
PS1			10	mm	55	mm	100	fF	7	pF	
CS2			10	mm	58	mm	100	fF	10	pF	
CS3			10	mm	56	mm	100	fF	10	pF	
CS4			10	mm	54	mm	100	fF	10	pF	
CS5			10	mm	57	mm	100	fF	10	pF	
SLD10	12	mm	2	mm	62	mm	100	fF	9	pF	
SLD11	12	mm	2	mm	64	mm	100	fF	9	pF	
SLD12	12	mm	2	mm	67	mm	100	fF	10	pF	
SLD13	12	mm	2	mm	69	mm	100	fF	10	pF	
SLD14	12	mm	2	mm	71	mm	100	fF	10	pF	
SLD20	10	mm	2	mm	45	mm	100	fF	7	pF	
SLD21	10	mm	2	mm	47	mm	100	fF	7	pF	
SLD22	10	mm	2	mm	49	mm	100	fF	8	pF	
SLD23	10	mm	2	mm	52	mm	100	fF	8	pF	
SLD24	10	mm	2	mm	54	mm	100	fF	8	pF	

4.1.3.2 Estimating the Average Current

After the sensor dimensions are entered as shown in the [Estimating the CP Value](#) section, use the following steps to estimate the average current consumption of the controller:

1. In Table B, select the sensitivity of each sensor from the “Sensitivity” column. The higher the value of sensitivity, the lower the current consumption.
2. In Table C, select the scan period. The higher the scan period value, the lower the average current consumption.
3. Select the [State Timeout](#) period. The lower the timeout period value, lower the power consumption. Select this value based on your design requirements.
4. If EMC is enabled, select “Enabled”; otherwise select “Disabled”.
5. Select the Buzzer ON Time value if the buzzer is enabled; otherwise select 0. When the buzzer is enabled, the average current consumption increases with the increase in the Buzzer ON Time value.
6. If PWM is enabled on the GPOs, select “Enabled”; otherwise select “Disabled”.
7. Enter the average number of touches in an hour and the duration of each sensor touch. This value depends on how frequently the user touches the sensors.
8. When all these parameters are entered, the toolbox displays the estimated average current consumption per hour, as [Figure 4-3](#) shows.

Figure 4-3. Average Current Consumption Estimation

Table C: Power Calculator

Scan Period	200	ms
Time Out Period	8	s
EMC	Enabled	
Buzzer ON Time	0	ms
PWM on GPOs	Disabled	
Wake On Proximity	Disabled	
Average Number of Button Touch per Hour	10	
Average Button Touch Time	100	ms
Estimated Average Current Consumption	1.08	mA

4.1.3.3 Estimating the Response Time

The response time of the sensors depends on the values entered in Table B and Table C. To estimate the response time of buttons and sliders, enter the parameters in Table B and Table C as explained in section 4.1.3.1 and section 4.1.3.2 respectively.

Follow these steps to estimate the response time of buttons and sliders:

1. In Table D, select the [Debounce](#) parameter value.
2. If infinite impulse response (IIR) and median filter are enabled, select “Enabled” in their respective cells; otherwise select “Disabled”.
3. When these parameters are entered, the toolbox displays the first touch and consecutive touch response times for buttons and sliders respectively, as [Figure 4-4](#) shows.

To estimate the response time of the proximity sensor and device timing parameters, follow these steps:

1. To estimate the response time of the proximity sensor in different operating modes, select the operating state of the controller. Refer to the [CY8CMBR3xxx Operating Modes](#) section for details on each operating mode.
2. If the Advanced Low-Pass (ALP) filter is enabled, then select “Enabled”; otherwise select “Disabled”.
3. Select the [Debounce](#) parameter value.
4. When these parameters are entered, the toolbox displays the response time for proximity detection and proximity release, as [Figure 4-4](#) shows.
5. To estimate the LED duration minimum and maximum values, select the LED duration value.

Figure 4-4. Response Time Estimation

Table D: Response Time Calculator

Button and Slider Touch and Release Response Time		
Debounce	3	
IIR Filter	Enabled	
Median Filter	Enabled	
Response time for first button touch	754	ms
Response time for consecutive button touch	268	ms
Response time for first slider touch	665	ms
Response time for consecutive slider touch	178	ms
Response time for button/slider release	178	ms

Proximity Sensor Response Time				
Operating State	Look-for-Touch			
ALP Filter	Disabled			
Debounce	3			
	Minimum	Typical	Maximum	
Response time for proximity detection	354	354	354	ms
Response time for proximity release	400	400	400	ms

Device Timing Parameters		
LED Duration	500	ms
LED Duration (Minimum)	500	ms
LED Duration (Maximum)	594	ms
GPO Control Status Delay	200	ms
Device Wakeup Time	50	ms

4.1.4 Validating the Design

After the prototype board is built and tested, use the design validation sheet to verify the design. To use the design validation sheet, you must enter the parameters in Table-B of the [CP, Power Consumption, and Response Time Calculator](#) sheet.

The “Design Validation” sheet validates only the design of buttons and slider sensors; it does not validate or provide recommendations for proximity sensor design because the design of proximity sensors depends on various external factors.

To validate your design, follow these steps:

1. Use [EZ-Click](#) to measure the raw counts, noise, and C_P value for buttons and sliders.
2. Enter the raw counts, noise, and C_P value in Table C for each sensor, as [Figure 4-5](#) shows.
3. Based on these parameters, the toolbox provides recommended sensor dimension. If the sensor dimension entered in Table-B of the [CP, Power Consumption, and Response Time Calculator](#) sheet is greater than or equal to the recommended sensor dimension, the design is shown as “Pass”; otherwise it is shown as “Fail”.

If the design passes, the prototype stage is complete and the design is production-ready.

Note: If the design has a proximity sensor, the toolbox will not provide any recommendation for the proximity sensor and will not validate the entire design.

If the design fails, follow these steps:

1. Reduce the sensitivity parameter value in Table B.
2. If the design still fails, reduce the overlay thickness in Table A, as [Figure 4-6](#) shows.

3. If the design still fails after reducing the sensitivity value and reducing overlay thickness, enter the new sensor dimensions in Table B, based on the recommended values from Table C, as Figure 4-7 shows.

Note: The layout has to be re-designed based on the recommended values from Table C.

Figure 4-5. Design Validation - Sensor Dimension Recommendation Table

Table C: Sensor Dimension Recommendation

Sensor	Values taken from I2C						Improvement Recommendations					
	Noise		Rawcount		Cp		Recommended Sensor Dimension			Comments	Maximum Trace Length	
PS0	20	counts	45000	counts	12	pF			mm			mm
PS1	25	counts	45400	counts	13	pF			mm			mm
CS2	10	counts	3000	counts	12	pF			6 mm		520	mm
CS3	10	counts	3000	counts	12	pF			6 mm		520	mm
CS4	10	counts	3000	counts	13	pF			6 mm		520	mm
CS5	10	counts	3000	counts	15	pF			7 mm		514	mm
SLD10	12	counts	3100	counts	9	pF	14	mm	2 mm		520	mm
SLD11	13	counts	3125	counts	10	pF	14	mm	2 mm		520	mm
SLD12	15	counts	3150	counts	10	pF	14	mm	2 mm		520	mm
SLD13	14	counts	3140	counts	9	pF	14	mm	2 mm		520	mm
SLD14	12	counts	3170	counts	10	pF	14	mm	2 mm		520	mm
SLD20	16	counts	3158	counts	8	pF	14	mm	2 mm		520	mm
SLD21	12	counts	3100	counts	7	pF	14	mm	2 mm		520	mm
SLD22	14	counts	3140	counts	8	pF	14	mm	2 mm		520	mm
SLD23	13	counts	3125	counts	9	pF	14	mm	2 mm		520	mm
SLD24	16	counts	3158	counts	7	pF	14	mm	2 mm		520	mm

Figure 4-6. Design Validation – Actual Design Values Table

Table A: Actual Design Values

Input Parameters	Initial Value	New Value	Units
Overlay Thickness	2	1	mm
Overlay - Dielectric constant	2.8		farad/m
Capacitance of trace per inch	2		pF
Scan Period	200	200	ms
Time Out Period	8	8	s
EMC	Enabled	Enabled	
Buzzer ON Time	0	0	ms
Wake On Proximity	Disabled	Disabled	
Average Number of Button Touch per	10	10	
Average Button Touch Time	100	100	ms

New overlay thickness

Figure 4-7. Design Validation – Actual Sensor Dimensions Table

Table B: Actual Sensor Dimensions

Sensor	Sensor Dimension								Sensitivity			
	Slider Length		Button Diameter/Slider Width/Proximity Loop Diameter		Slider Length		Button Diameter/Slider Width/Proximity Loop Diameter					
Initial Value					New Value				Initial Value		New Value	
CS0			10	mm				mm	100	fF		fF
CS1			10	mm				mm				fF
CS2			10	mm				mm				fF
CS3			10	mm				mm				fF
CS4			10	mm				mm				fF
CS5			10	mm				mm	100	fF		fF
SLD10	12	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD11	12	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD12	12	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD13	12	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD14	12	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD20	10	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD21	10	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD22	10	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD23	10	mm	2	mm	14	mm	2	mm	100	fF		fF
SLD24	10	mm	2	mm	14	mm	2	mm	100	fF		fF

New sensor dimension value

4.2 Layout Guidelines for Proximity Sensor

A capacitive proximity sensor senses nearby conductive objects without any physical contact. A proximity sensor differs from a button sensor in two different aspects: sensor construction and sensor sensitivity. Proximity sensor construction should be such that the electric field lines are projected to a maximum distance to achieve higher proximity distance. The proximity distance depends on the sensor's electrical field propagation (electrical field strength). A longer propagation distance provides a longer detection range.

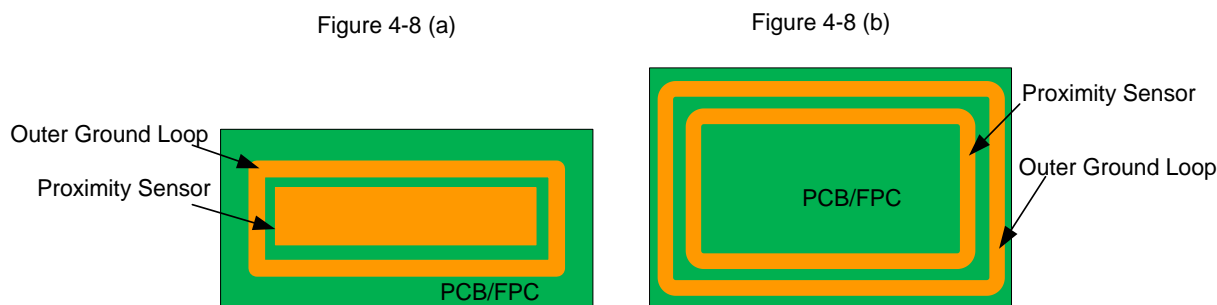
4.2.1 Sensor Design Guidelines

Proximity sensor construction depends on the intended application and the required proximity range. Applications such as face proximity detection in mobile phones and Specific Absorption Rate (SAR) regulation in tablets require proximity distance of up to 3 cm. To achieve this proximity distance, a sensor with solid fill and small area is sufficient. For applications such as photo frames, monitors, and keyboards, which require higher proximity distance, use a large sensor loop covering the circumference of the device to achieve higher proximity distance.

A proximity sensor can be constructed using the following:

- Copper trace on a PCB or FPC: A long PCB trace can be used as a proximity sensor. The trace can be a straight line, or it can surround the perimeter of a system's user interface, as [Figure 4-8](#) shows. Proximity sensor construction using PCB trace is appropriate for mass production, but it is not as sensitive as a wire sensor. It is recommended to have a ground loop around the proximity sensor to reduce the noise and provide immunity against ESD events. There is a trade-off between proximity range and noise immunity, ESD performance. For proximity sensor and ground loop layout recommendations, refer to the [General Layout Guidelines](#) sheet of the design toolbox.

Figure 4-8. Proximity Sensor Construction on PCB/FPC

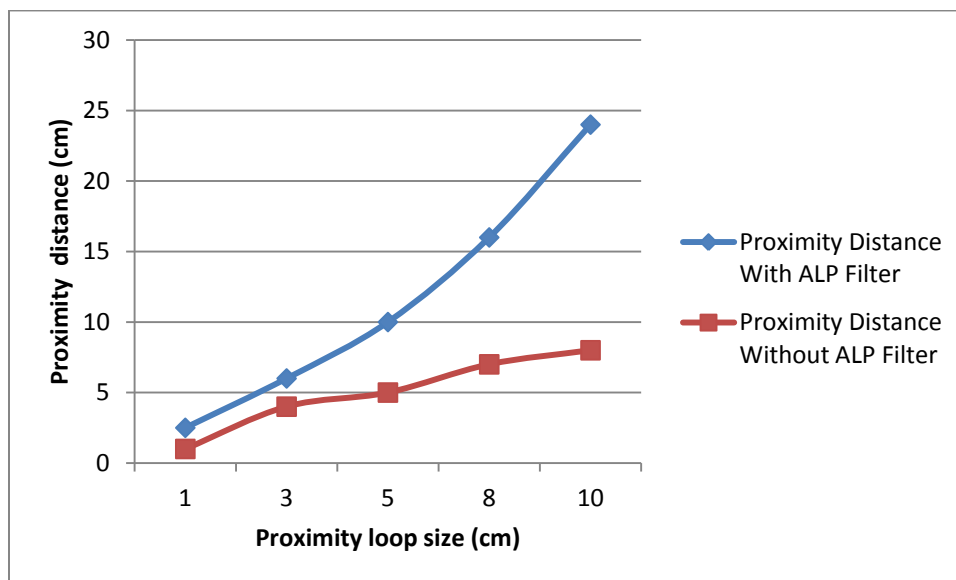


- **Wire:** A single length of wire works well as a proximity sensor. Because detecting a hand relies on the capacitance change from electric field changes, any stray capacitance or objects affecting the electrical field around the wire will affect the range of the proximity sensor. Using a wire sensor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

To increase the proximity range, follow these guidelines:

- **Increase the proximity sensor loop size:** The proximity distance increases with increase in the proximity sensor area, as [Figure 4-9](#) shows.
- **Set the proximity resolution to maximum value:** The proximity distance increases with increase in the proximity sensor resolution.
- **Enable the Advanced Low-Pass (ALP) filter:** The ALP filter attenuates the noise in proximity sensor raw counts and increases the proximity distance, as [Figure 4-9](#) shows.
- **Shield the proximity sensor from any nearby metal objects:** Metal objects placed close to the proximity sensor reduce the sensitivity and decrease the proximity distance. Use a shield electrode to eliminate the effect of metal objects and increase the proximity distance.

Figure 4-9. Relationship between Proximity Loop Size and Proximity Distance



Notes:

- The proximity distance shown in [Figure 4-9](#) is measured under ideal conditions. The distance might reduce in practical applications depending on the operating environment.
- The proximity distance shown in [Figure 4-9](#) is measured with the ground loop surrounding the proximity sensor. The clearance between the proximity loop and ground loop is kept to 1 mm.

4.2.2 Effect of Metal Objects on Proximity Distance

The proximity detection range reduces drastically if there is a metal surface near to the sensor. Most home appliances and automotive applications have a metal frame or case, so achieving a higher proximity range in these applications is very challenging.

The following factors cause the proximity detection range to reduce drastically when conductive objects are placed close to the sensor:

- The parasitic capacitance of the sensor increases. Larger stray capacitance often requires reducing the sensor operation frequency, causing the proximity distance to decrease.
- A grounded metal plane catches a part of the sensor electric field and reduces the capacitance added by the palm.

The influence of a nearby metal surface on a proximity sensor is decreased by placing a shield electrode between the proximity sensor and the metal object, as [Figure 4-12](#) shows.

Figure 4-10. Electrical Field Propagation for a Single Sensor Configuration without a Metal Object

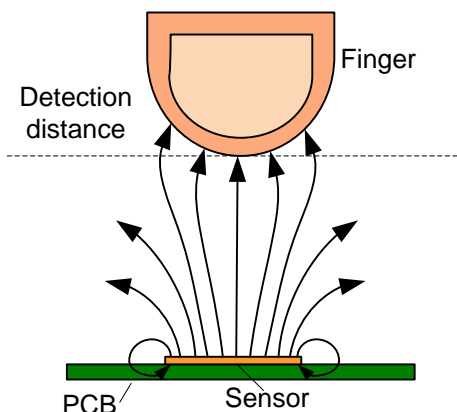


Figure 4-11. Electrical Field Propagation for a Single Sensor Configuration with a Solid Metal Object

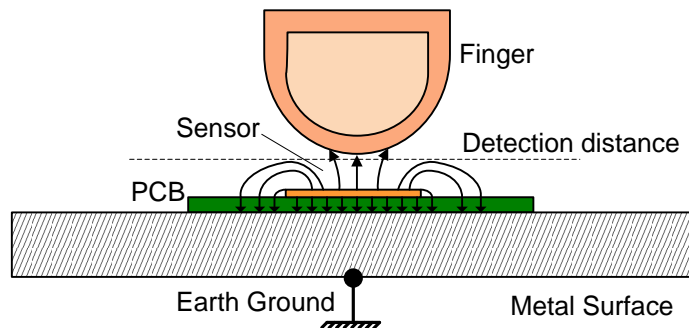
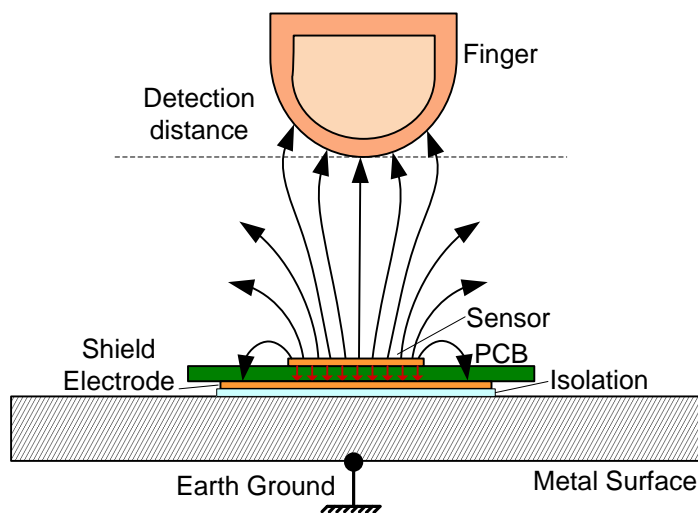


Figure 4-12. Using a Shield Electrode to Decrease the Metal Object's Influence



4.3 Layout Guidelines for Liquid Tolerance

Presence of liquid droplets or continuous liquid flow on the overlay can cause capacitive touch sensors to falsely trigger. Using a shield electrode eliminates the false touches caused by the liquid droplets on the sensor. Similarly, using a guard sensor eliminates the false touches due to a continuous flow of liquid. Refer to the [Liquid Tolerance](#) section to understand the working of shield and guard sensors.

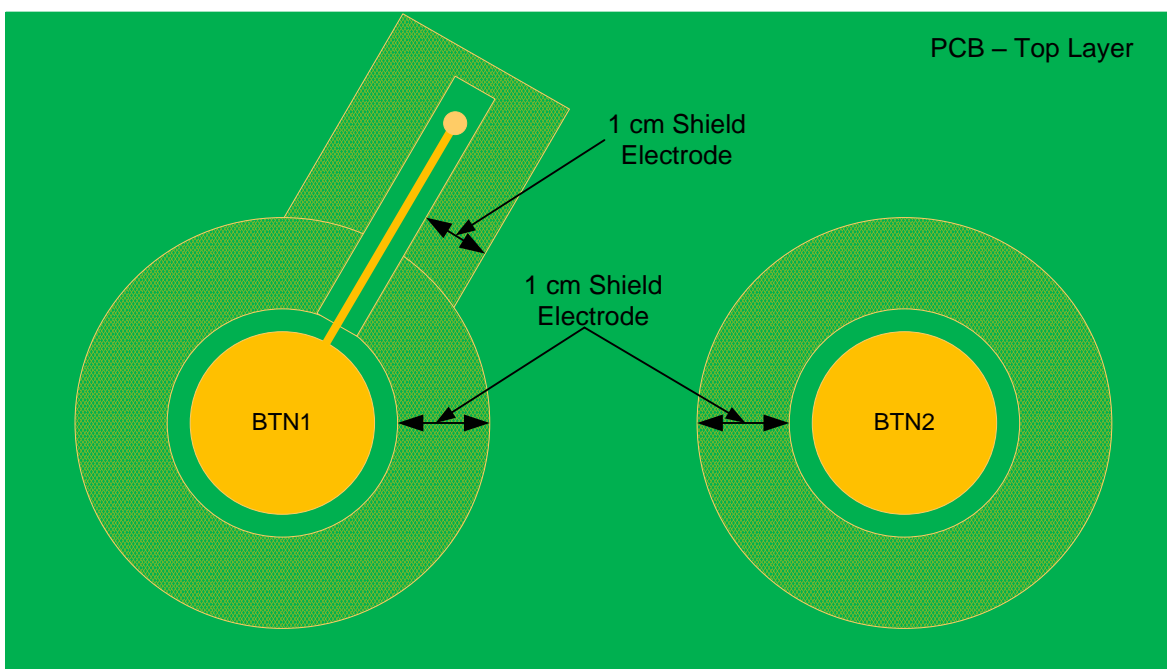
4.3.1 Shield Electrode Construction

Keep in mind the following guidelines to design a liquid-tolerant capacitive sensing application:

- Use a hatch ground of 0.17 mm (7 mil) trace and 1.143 mm (45 mil grid) in the top layer of the PCB and drive this hatch pattern with shield signal (SH pin).
- Surround the sensor pads and exposed traces (for single-layer PCB) with the shield electrode.

Keep the shield electrode spread within 1 cm from the sensor. Spreading the shield electrode beyond 1 cm has negligible effect on system performance and might increase radiated emissions. If the board area is very large, the area outside the 1-cm shield electrode should be left empty, as Figure 4-13 shows. In some applications, there might not be sufficient area available on the PCB for shield electrode implementation. In such cases, the shield electrode can spread less than 1 cm; the minimum area for shield electrode can be the area remaining on the board after implementing the sensor.

Figure 4-13. Shield Electrode Pattern



Follow these guidelines to implement the shield electrode in two-layer and four-layer PCBs:

Two-Layer PCB:

- Top layer: Hatch fill with 7-mil trace and 45-mil grid (25 percent fill). Hatch fill should be connected to the driven-shield signal.
- Bottom layer: Hatch fill with 7-mil trace and 70-mil grid (17 percent fill). Hatch fill should be connected to the driven-shield signal.

Four-Layer PCB:

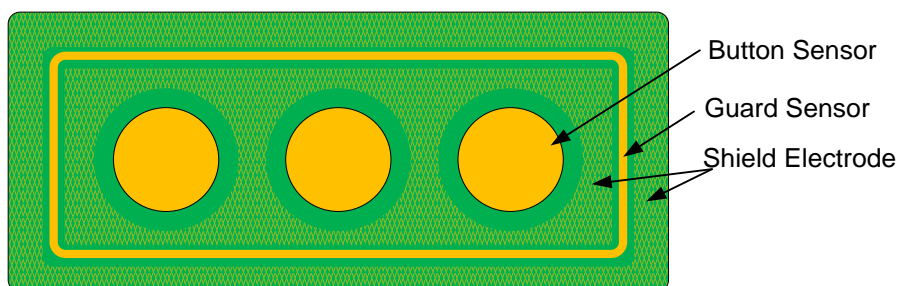
- Top layer: Hatch fill with 7-mil trace and 45-mil grid (25 percent fill). Hatch fill should be connected to the driven-shield signal.
- Layer-2: Hatch fill with 7-mil trace and 70-mil grid (17 percent fill). Hatch fill should be connected to the driven-shield signal.
- Layer-3: V_{DD} Plane
- Bottom layer: Hatch fill with 7-mil trace and 70-mil grid (17 percent fill). Hatch fill should be connected to ground.

The recommended air gap between the sensor and the shield electrode is 1 mm.

4.3.2 Guard Sensor

The guard sensor is a copper trace that surrounds all sensors, as [Figure 4-14](#) shows. Ensure that the shield electrode pattern surrounds the guard sensor, exposed traces, and spreads no further than 1 cm from these features. The guard sensor should be in the shape of a rectangle with curved edges. The recommended thickness of a guard sensor is 2 mm, and the maximum distance between the guard trace and the shield hatch is 1 mm.

Figure 4-14. PCB Layout With Shield Electrode and Guard Sensor



4.4 Sample Layouts

4.4.1 Touch Buttons on FPC

Figure 4-15 and Figure 4-16 show the top and bottom layer of a layout designed for implementing touch buttons on FPC. For the schematic of this design, see Figure 3-6.

The following layout best practices are followed in this layout:

1. C_p of the sensor is minimized by pouring ground only in the top layer.
2. The trace length of the sensors is minimized by placing the chip close to the sensors.
3. VDD, LED and I²C lines are isolated from sensor traces.
4. The clearance (for LED backlighting) at the center of sensor is kept to a minimum value to achieve maximum sensitivity.

Figure 4-15. Touch Buttons on FPC – Top Layer

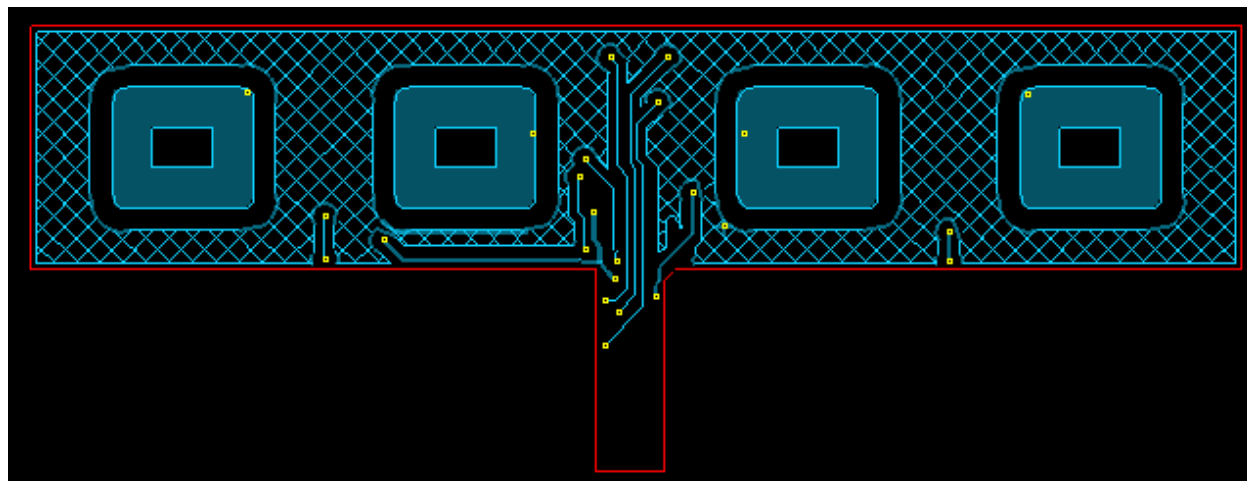
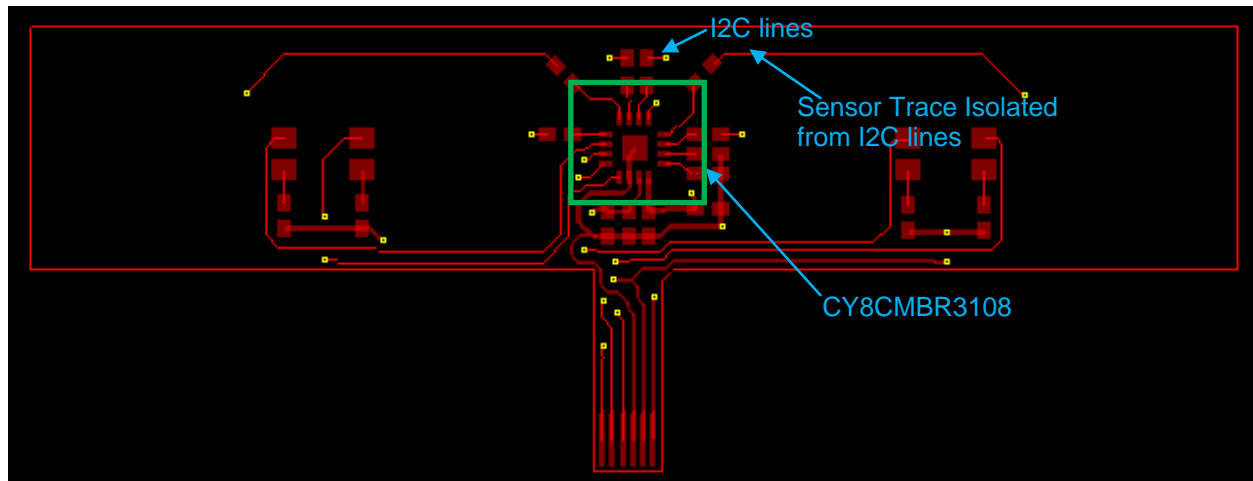


Figure 4-16. Touch Buttons on FPC – Button Layer



4.4.2 Touch Buttons on FR4 PCB

Figure 4-17 and Figure 4-18 show the top and bottom layer of a layout designed for implementing liquid tolerant touch buttons on FR4 PCB. For the schematic of this design, see Figure 3-7.

The following layout best practices are followed in this layout:

1. The shield and guard sensor layout is constructed as explained in the [Layout Guidelines for Liquid Tolerance](#) section.
2. The trace length of the sensors is minimized by placing the chip close to the sensors.
3. VDD, LED and I²C lines are isolated from sensor traces.
4. The clearance (for LED backlighting) at the center of sensor is kept to a minimum value to achieve maximum sensitivity.

Figure 4-17. Liquid-Tolerant Touch Buttons on FR4 PCB – Top Layer

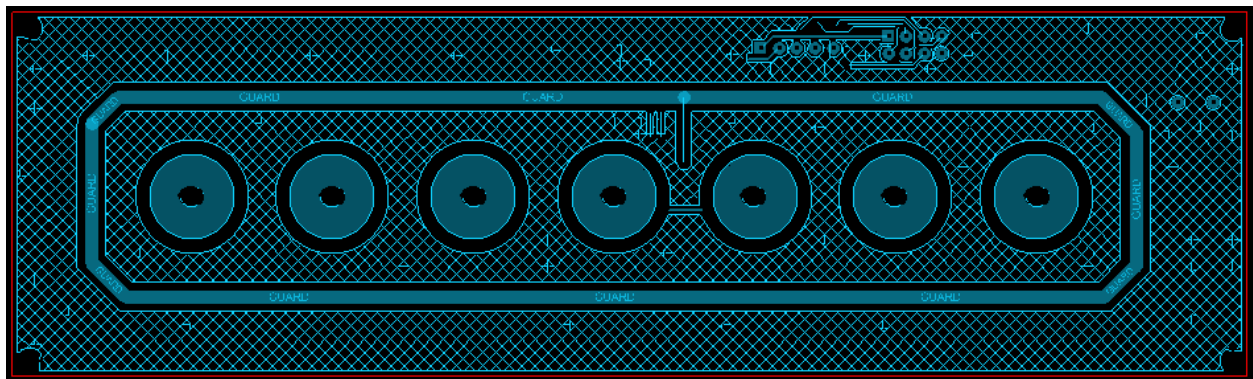
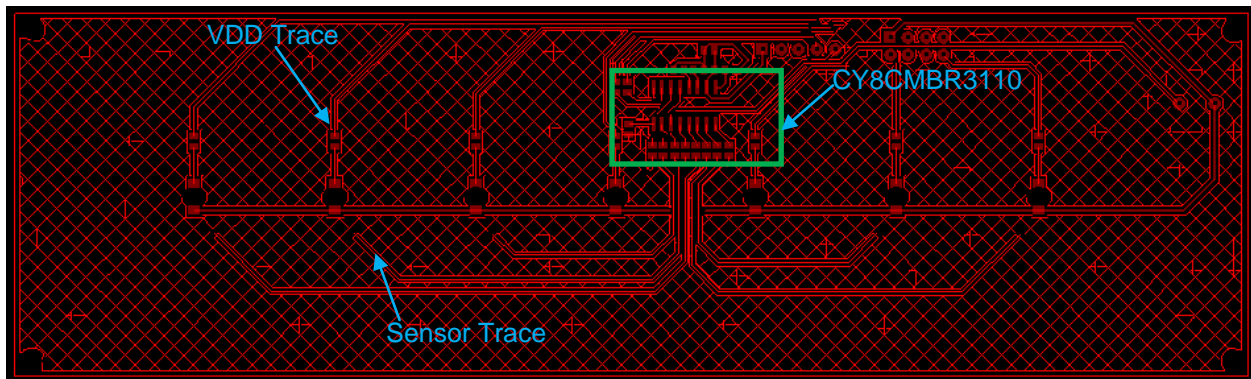


Figure 4-18. Liquid-Tolerant Touch Buttons on FR4 PCB – Bottom Layer



4.4.3 Proximity Sensor on FPC

Figure 4-19 and Figure 4-20 show the top and bottom layer of a layout designed for implementing a proximity sensor on FPC. For the schematic of this design, see Figure 3-8.

The following layout best practices are followed in this layout:

1. The proximity sensor loop is surrounded by a ground loop for improved noise immunity and ESD performance.
2. C_p of the sensor is minimized by using a loop sensor.
3. The proximity sensor is surrounded by a ground ring to minimize the noise. This also helps in providing immunity against ESD events.
4. The trace length of the sensor is minimized by placing the chip close to the sensor.
5. VDD and I²C lines are isolated from sensor traces.

Figure 4-19. Proximity Sensor on FPC –Top Layer

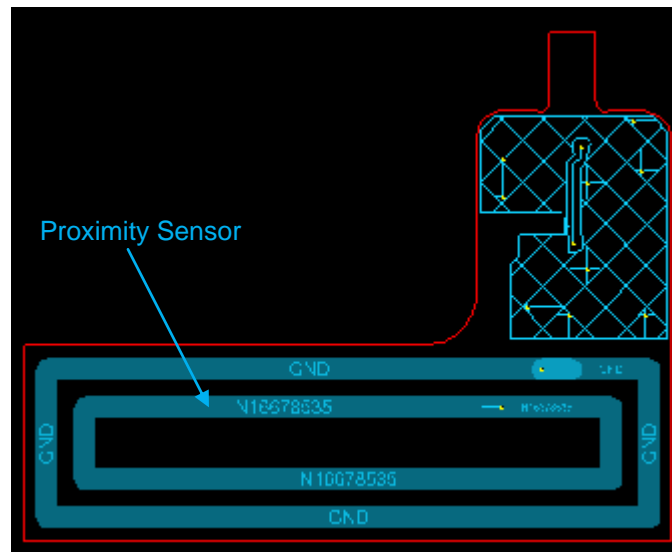
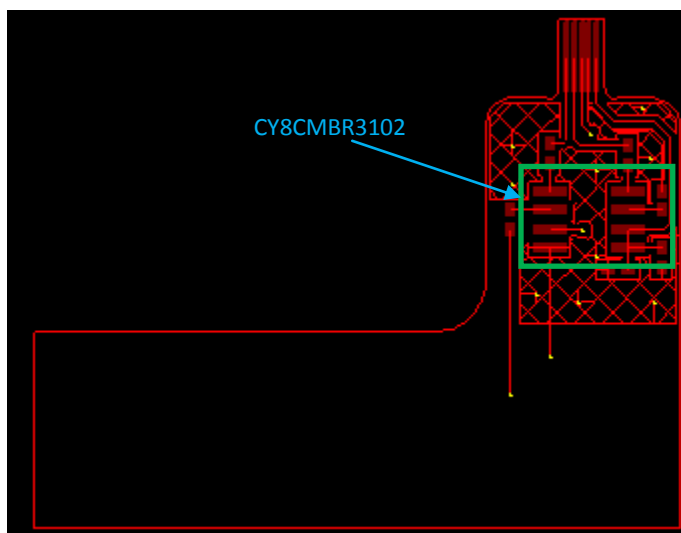


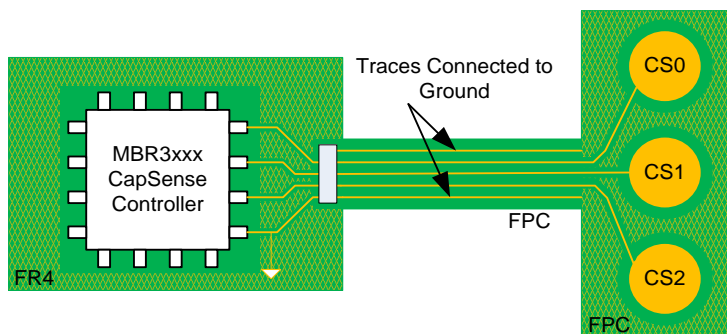
Figure 4-20. Proximity Sensor on FPC – Bottom Layer



4.4.4 CapSense Design on FR4 and FPC

Figure 4-21 shows a CapSense design in which the CapSense sensors are implemented on an FPC and the CapSense controller is placed on an FR4 PCB. The CapSense sensors are connected to the controller using a FPC connector.

Figure 4-21. CapSense Design on an FR4 and FPC



Follow these guidelines for a CapSense design where the sensor is on an FPC and the CapSense controller is on an FR4 PCB:

1. In the FPC, to minimize the sensor parasitic capacitance, you should have a hatch fill only in the top layer. The hatch fill can be connected to either ground or driven shield signal. The recommended hatch fill in the top layer is 7-mil trace and 45-mil grid.
2. In the FPC connector, to minimize the noise coupling to the CapSense system via sensor trace, reduce the sensor trace length and place a ground trace in parallel to the sensor trace, as Figure 4-21 shows.
3. In the FR4 PCB, you can place the CapSense controller in the top layer and have a grounded hatch fill in the unoccupied area. The recommended hatch fill in the top layer is 7-mil trace and 45-mil grid. In the bottom layer, you can have a hatch ground of 7-mil trace and 70-mil grid.

5. Configuring CY8CMBR3xxx



CY8CMBR3xxx is an I²C-configurable CapSense controller. This controller provides various features and can be easily configured using [EZ-Click](#), which is a simple, intuitive, graphical user interface tool. All the controllers in the CY8CMBR3xxx family, except the CY8CMBR3002, can be configured using [EZ-Click](#). The CY8CMBR3002 controller is a fixed-function device and cannot be configured. The following section explains the various configurations supported in the CY8CMBR3xxx family and how to configure them.

5.1 Configurations in CY8CMBR3xxx

The configurations supported in CY8CMBR3xxx controller can be divided into the following four categories:

1. Sensor configuration
2. GPO configuration
3. Buzzer configuration
4. Global device configuration

5.1.1 Sensor Configuration

5.1.1.1 Sensitivity Control

The sensitivity parameter is used to increase or decrease the strength of the sensor signal (difference count). A lower value of sensitivity (100 fF) setting leads to a stronger signal from the sensor, but increases the response time and average power consumption. The different sensitivity settings available are 100 fF, 200 fF, 300 fF, and 400 fF. The sensitivity parameter can be configured for each button sensor, guard sensor, and slider group. The sensitivity parameter is not applicable to proximity sensors.

The following factors affect the sensor sensitivity:

1. Overlay thickness: Thicker overlay requires lower value of sensitivity setting.
2. System noise: As system noise increases, sensitivity needs to be lower to avoid false trigger of sensors.
3. Form factor of the design: A relatively large sensor size requires higher value of sensitivity. Sensors with smaller size require lower value of sensitivity.
4. Power consumption: Power consumption increases for lower values of sensitivity setting. For low power consumption, the sensitivity parameter should be high.

Power consumption decreases in the order 100 fF > 200 fF > 300 fF > 400 fF.

The sensitivity parameter can be set in the “CapSense sensor Configuration” tab in [EZ-Click](#). Refer to the [CapSense Performance Tuning](#) section to understand how to set the sensitivity parameter. This parameter is fixed at 100 fF for the CY8CMBR3002 CapSense controller and cannot be configured.

5.1.1.2 Proximity Resolution

The proximity distance depends on the proximity resolution parameter. Higher proximity resolution results in higher proximity distance. The resolution parameter should be specified for each proximity sensor. This parameter can range from 12 to 16 bits. The proximity resolution parameter can be set in the “CapSense sensor Configuration” tab in [EZ-Click](#). Refer to the [CapSense Performance Tuning](#) section to understand how to set the proximity resolution parameter. This parameter is not applicable to the CY8CMBR3002 CapSense controller.

CS2 is reported as ON when only CS2 is touched

5.1.1.3 Slider Resolution

The resolution of the slider indicates the number of unique positions that the controller can resolve. A slider with higher resolution can resolve more number of positions. This parameter is valid only for the CY8CMBR3106S controller. CY8CMBR3106S supports up to two sliders: the SLD1x group and SLD2x group. The resolution parameter for each slider group should be specified. However, if both the sliders are combined in to one single slider, specify

only one resolution. The value of this parameter can range from 1 to 254. The slider resolution parameter can be set in the “CapSense sensor Configuration” tab in [EZ-Click](#).

5.1.1.4 Electromagnetic Compatibility (EMC)

The EMC feature determines the device’s immunity to external, radiated, and conducted noise such as audio-frequency noise from power amplifiers, radio-frequency noise from wireless transmitters, ESD, and power-line surges. For a system in a high-noise environment, enable the EMC feature. Power consumption and response time increase when the EMC feature is enabled.

For designs based on CY8CMBR3106S, EMC feature is applicable only for sensors CS0/PS0, CS1/PS1 and CS2 to CS5. All the slider segments and buttons from CS11-CS15 work without EMC.

For designs based on CY8CMBR3116, EMC feature is applicable only for sensors CS0/PS0, CS1/PS1 and CS2 to CS9.

Disable the EMC feature in the following cases:

- If any button from CS10 to CS15 is enabled in CY8CMBR3116 controller.
- If filters are enabled (*In designs based on CY8CMBR3106S only*)
- If your design will operate in a low-noise environment
- If the buzzer is enabled

You can enable the EMC feature in the “Global Configuration” tab in [EZ-Click](#). The EMC feature is always disabled in the CY8CMBR3002 CapSense controller.

5.1.1.5 Flanking Sensor Suppression (FSS)

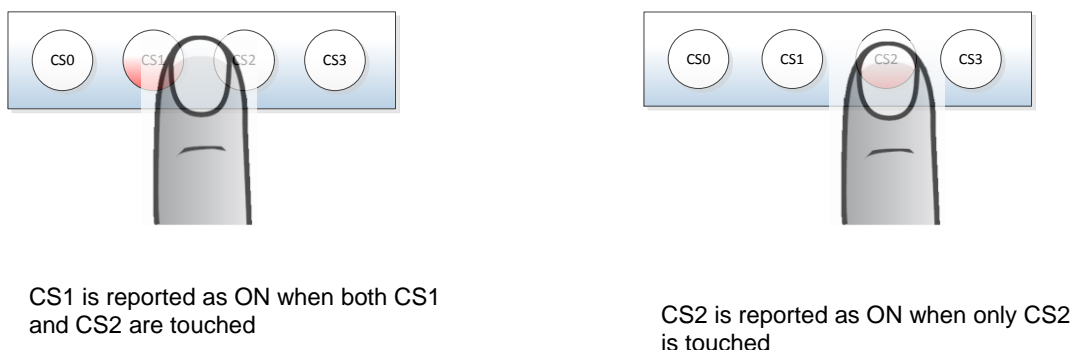
The FSS feature distinguishes among sensor touches from closely-spaced buttons, thus eliminating false touches. It ensures that the system recognizes only the first button touched when multiple sensors are touched. FSS ensures that only one button is reported as ON at any time, as [Figure 5-2](#) shows. If a finger contacts multiple buttons, only the first one to sense a TOUCH state will be reported as ON. If more than one button is touched at the same time, that is, if multiple buttons are touched simultaneously, then the sensor with the lower number is reported as ON.

The FSS feature is only applicable to buttons, and can be enabled or disabled for each button in “CapSense sensor Configuration” tab in [EZ-Click](#). The FSS feature is not applicable to the CY8CMBR3002 CapSense controller.

Figure 5-1. FSS When Only One Button Is Touched



Figure 5-2. FSS When Multiple Buttons are Touched With One Button ON Previously



5.1.1.6 Firmware filters

To reduce the noise in sensor raw counts, the CY8CMBR3xxx controller supports the following filters:

- Median Filter
- IIR Filter
- Advanced Low-Pass (ALP) Filter

The Median and IIR filter is applicable only to buttons, sliders, and the guard sensor. The ALP filter is applicable only to proximity sensors.

For designs based on CY8CMBR3106S, note the following considerations:

- ALP filter is not supported
- If EMC feature is enabled, disable the Median and IIR filter

5.1.1.6.1 Median Filter

The median filter is a nonlinear filter that computes the median value from a buffer of size N . This filter removes the spike noise from the sensor raw counts. The median filter implemented in the CY8CMBR3xxx controller is a third-order moving median filter. Refer to the [CapSense Performance Tuning](#) section to know when to enable this filter.

If the median filter is enabled, it is recommended to disable the [Automatic Threshold](#) feature and specify the [Finger Threshold](#) parameter manually. The median filter is always enabled in the CY8CMBR3002 CapSense controller.

5.1.1.6.2 Infinite Impulse Response (IIR) filter

The IIR filter produces a step response similar to RC filters. This filter attenuates high-frequency white noise from sensor raw counts. Refer to the [CapSense Performance Tuning](#) section to know when to enable this filter. The IIR filter is always enabled in the CY8CMBR3002 CapSense controller.

When both the median and IIR filter are enabled, the median filter is applied first on the raw counts, followed by the IIR filter.

5.1.1.6.3 Advance Low-Pass (ALP) Filter

The ALP filter is used in attenuating the noise in proximity sensor raw counts. To achieve higher proximity distance, the proximity sensor requires higher resolution; however, a higher resolution setting results in increased noise in the raw counts. To attenuate this high noise and achieve an SNR greater than 5:1, enable the ALP filter. Refer to the [Tuning a Proximity Sensor](#) section to know when to enable the ALP filter and how to tune it when it is enabled. This feature is not applicable for the CY8CMBR3106S and CY8CMBR3002 CapSense controllers.

5.1.1.7 Automatic Threshold

The automatic threshold feature dynamically sets all the threshold parameters for button sensors, depending on the noise in the environment. For a variable-noise environment, enable this feature. This feature is applicable only to button sensors. This feature is always enabled in the CY8CMBR3002 CapSense controller.

Consider the following on the automatic threshold feature:

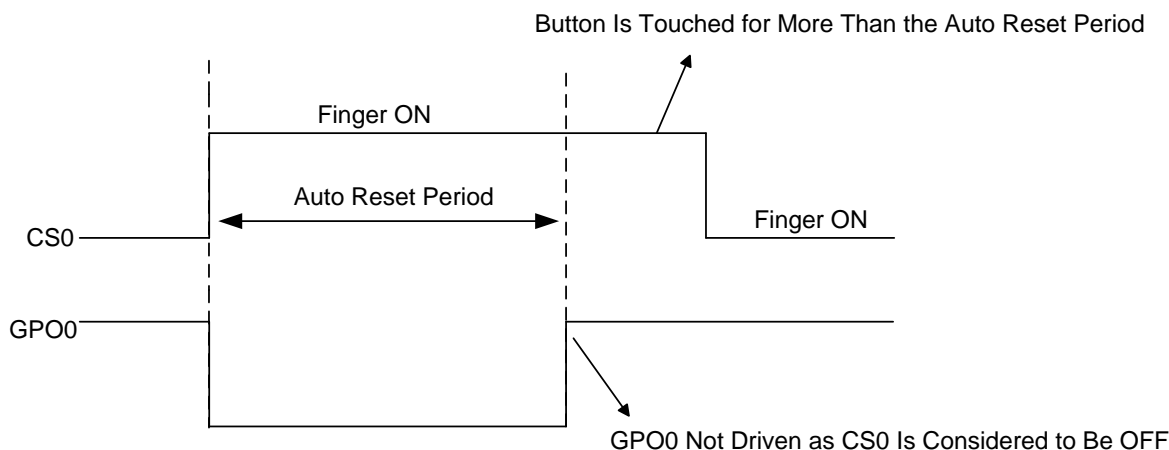
- If the EMC feature is enabled, the automatic threshold feature will be disabled. In this case, manually set the finger threshold parameter.
- If this feature is disabled, set only the finger threshold parameter. All other threshold parameters are automatically set based on the finger threshold parameter.
- If the median filter is enabled, disable the automatic threshold feature and manually set the finger threshold parameter.

5.1.1.8 Sensor Auto-Reset Time

The Sensor Auto-Reset time determines the maximum time a sensor is considered to be ON when the sensors are activated. The sensor will be turned OFF after the auto-reset period, as [Figure 5-3](#) shows. This feature prevents a sensor from getting stuck ON in situations such as a metal object placed too close to it.

There are two sets of auto-reset parameters: auto-reset and proximity auto-reset. The auto-reset parameter is applicable to buttons and sliders. The proximity auto-reset is applicable to proximity sensors. Auto-reset is not applicable to guard sensors. The auto-reset period can be disabled, or set to 5 or 20 seconds. This parameter is set to 20 s in the CY8CMBR3002 CapSense controller.

Figure 5-3. Sensor Auto Reset



5.1.1.9 Debounce

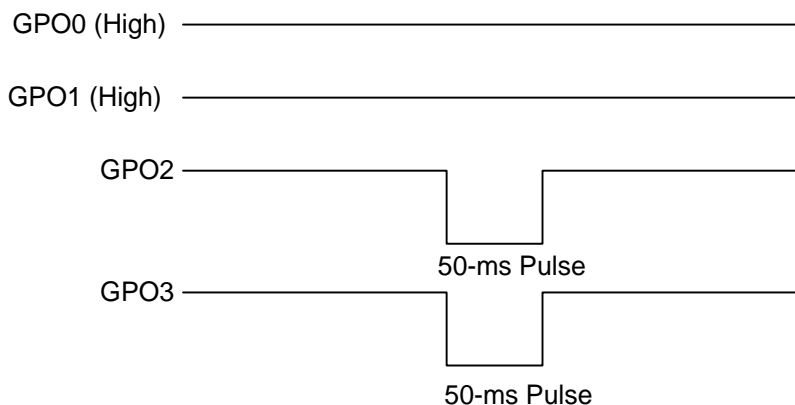
The debounce feature avoids false triggering of sensors due to noise spikes or system glitches by specifying the minimum time a sensor has to be touched for a valid ON status.

The debounce value can vary depending on the sensor function. For example, the power button should have a long debounce time to avoid inadvertently switching the system ON or OFF. Shorter debounce times speed up the device's response to a sensor touch or proximity events. The debounce parameter is applicable to all types of sensors, except slider sensors, and can range from 1 to 15. Set this parameter depending on the response time requirement. See the [Estimating the Response Time](#) section for more details. This parameter is set to 3 in the CY8CMBR3002 CapSense controller.

5.1.1.10 System Diagnostics

System diagnostics is a built-in power-on self-test (BIST) mechanism. When enabled, it performs several tests at power-on reset (POR) or after software rest, which are useful in production testing. If any sensor is detected faulty, it will be disabled and a 50-ms pulse will be sent out on the corresponding GPO (if GPO is enabled) during power-up, as [Figure 5-4](#) shows. The System Diagnostics status is also updated in the register map so that the host can read the test results through the I²C interface. Refer to the [CY8CMBR3xxx Registers TRM](#) for more details.

Figure 5-4. Example Showing CS0, CS1 Passing the BIST and CS2, CS3 Failing



The following tests are performed during power-up:

1. Sensor shorted to Ground: If any sensor is found to be shorted to ground, it is disabled.
2. Sensor shorted to VDD: If any sensor is found to be shorted to VDD, it is disabled.
3. Sensor-to-sensor short: If two or more sensors are found to be shorted to each other, all of these sensors are disabled.
4. Improper value of C_{MOD} : Recommended value of C_{MOD} is $2.2 \text{ nF} \pm 10\%$. If the value of C_{MOD} is found to be less than 1 nF or greater than 4 nF , all sensors are disabled and 50 ms pulse is sent out on all GPOs.
5. Button $C_P > 45 \text{ pF}$: If the sensor C_P is greater than 45 pF , then it is disabled. The C_P estimated by CY8CMBR3xxx has an error of $\pm 2 \text{ pF}$. Therefore, the sensor may be disabled if its C_P is in the range of $43\text{-}47 \text{ pF}$.
6. If shield is shorted to V_{DD} , Ground, or a sensor, all sensors will be disabled and a 50 ms pulse will be sent out on all the enabled GPOs.
7. If any slider segment fails the above tests, the entire slider will be disabled.

If system diagnostics is enabled, the boot-up time of the CY8CMBR3xxx controller increases. Refer to the System Specifications section in the [CY8CMBR3xxx datasheet](#) for more details.

Note: CY8CMBR3106S does not support GPOs; therefore, read the system diagnostics results from the register map using I²C interface.

5.1.1.11 Shield and Guard Sensor

Shield and guard sensor are required for a liquid-tolerant CapSense design. If your design has a shield electrode, select the “Enable shield” option in the “Global configuration” tab of [EZ-Click](#). This selection allows the controller to drive the shield signal on the SH pin.

If your design has guard sensors, select the “Enable guard sensor” option in the “Global configuration” tab of the [EZ-Click](#). This selection allows the controller to configure the pin as a guard sensor. See [Table 1-1](#) to know the devices that support the Shield and Guard sensor features.

5.1.2 GPO Configuration

The CY8CMBR3xxx controller supports up to eight GPOs to drive LEDs or to drive an external resistive network to form a basic D/A converter. A GPO can output a PWM signal or a DC voltage level depending on the configuration. Since the GPO pins are multiplexed with the CapSense sensor pins, specify whether a CY8CMBR3xxx pin is a CapSense pin or a GPO pin in [EZ-Click](#). The CY8CMBR3106S CapSense controller does not support GPOs and, therefore, this feature is not applicable to the device.

5.1.2.1 GPO Drive Mode

The GPOs in CY8CMBR3xxx support two drive modes: open-drain low and strong-drive mode. The open-drain low mode is equivalent to a normally opened (n/o) switch. When the sensor is inactive, the corresponding GPO will be in the HI-Z state; when it is active, the GPO is pulled LOW. This drive mode is useful for driving an external resistive network to perform a D/A conversion. The strong-drive mode can be used for driving LEDs. The GPO drive mode for the CY8CMBR3002 CapSense controller is fixed to open-drain low.

The GPO drive mode is global and applies to all the enabled GPOs.

5.1.2.2 GPO Logic Level

CY8CMBR3xxx allows the user to configure the logic level (Active HIGH or Active LOW) of the GPO pin. This feature provides flexibility in controlling the output voltage of GPOs based on the sensor status, as [Table 5-1](#) shows. This feature is global and applies to all the enabled GPO pins. The GPO logic level for the CY8CMBR3002 CapSense controller is “Active Low”. If the drive mode is set to open-drain low drive mode then the [Table 5-1](#) is applicable only when the pin is externally pulled to V_{DD} .

Table 5-1. GPO Logic Level

GPO Logic Level	Sensor Status	Output Voltage on GPO
Active HIGH	ON	V_{DD}
	OFF	GND
Active LOW	ON	GND
	OFF	V_{DD}

5.1.2.3 PWM Output on GPO

CY8CMBR3xxx provides a PWM output on the GPO pins. The PWM signal is useful for controlling the brightness of LEDs. To enable PWM output on GPOs, specify the active-state duty cycle and inactive-state duty cycle percentage in the “Global configuration” tab in [EZ-Click](#). An active duty cycle of 100% and inactive duty cycle of 0% results in a DC output on the GPO. This feature is not applicable to the CY8CMBR3106S and CY8CMBR3002 CapSense controllers.

The PWM output feature is global and applies to all enabled GPOs. The frequency of the PWM signal is 106.6 Hz. The PWM duty cycle can be configured for each GPO with a step size of 6.67%.

5.1.2.4 GPO Host Control

In CY8CMBR3xxx, each GPO (GPO0-GPO7) is logically linked to the CapSense pins (CS0-CS7). When a sensor is activated, the corresponding GPO will be automatically turned ON. However, if the host processor wants to control the GPO pins, the GPO host control feature should be enabled. This feature allows the host processor to control the GPO pins by writing to the GPO_OUTPUT_STATE register. Refer to [CY8CMBR3xxx Registers TRM](#) for more details. This feature is not applicable to the CY8CMBR3106S and CY8CMBR3002 CapSense controllers.

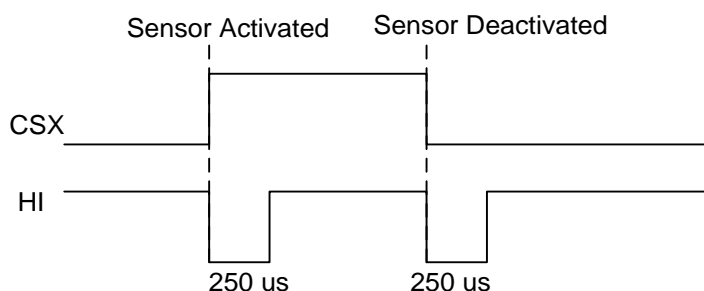
The delay between the host writing the command and the CY8CMBR3xxx controller responding to these commands is equal to one [Scan Period](#) when the controller is operating in [Look-for-Touch Mode](#) and [Look-for-Proximity Mode](#). If the controller is operating in Active Mode, then the delay depends on the total scan time of all sensors.

Note In EZ-Click, even if a GPO is not enabled and if the corresponding GPOx/CSx pin is not configured as a CapSense sensor, the GPO will be driven (turned ON/OFF) based on the corresponding sensor ON/OFF status. To disable a GPO pin, you need to configure the pin as a CapSense input in EZ-Click and connect the pin to ground.

5.1.2.5 Host Interrupt

The host interrupt feature generates an active low pulse signal with duration of 250 μ s on the \overline{HI} pin to the host processor whenever the sensor status or slider position changes, as [Figure 5-5](#) shows. To enable this feature specify the Host Interrupt pin in the [EZ-Click](#). The host interrupt pin has open-drain low-drive mode configuration and requires an external pull-up resistor for proper operation. See [Table 1-1](#) to know the list of devices that support this feature.

Figure 5-5. Host Interrupt Line when CSx Button is Touched



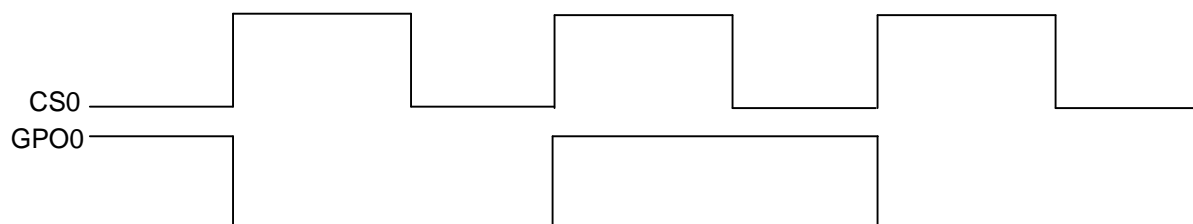
5.1.2.6 Toggle

When the Toggle feature is enabled, the state of GPO pins changes on every rising edge of the button and proximity sensor status. This feature implements the mechanical push button switch behavior in CapSense controller. The toggle configuration is shown in [Figure 5-6](#). The toggle feature can be enabled or disabled for each GPO pin. The toggle feature is not applicable for the CY8CMBR3002 CapSense controller.

Take into account the following considerations for using the toggle feature:

- The toggle and LED ON time features are mutually exclusive. If both the features are enabled, the toggle feature takes higher priority and so the LED ON time feature will be disabled.
- If the toggle feature is enabled and any GPO is turned ON, the device will not enter the [Look-for-Proximity Mode](#). Disable the toggle feature if your design requires the wake-on-approach feature.

Figure 5-6. Example of Toggle Feature

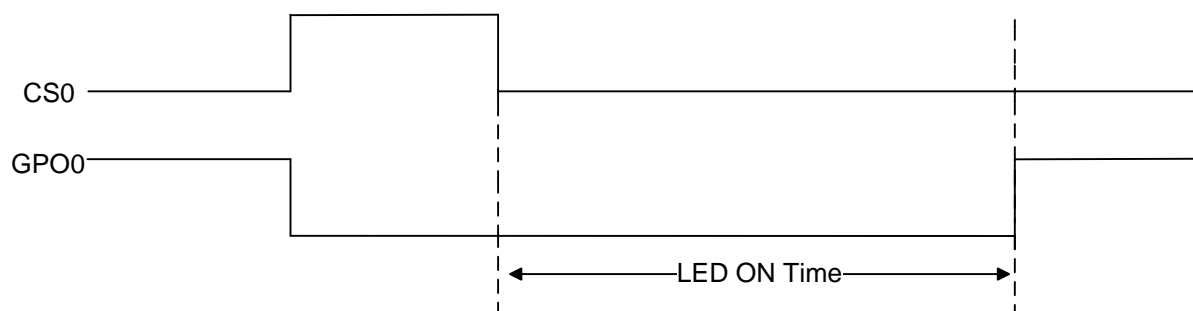


5.1.2.7 LED ON Time

The LED ON time specifies the duration for which the GPO pin is driven LOW or HIGH after the corresponding sensor is released, as shown in Figure 5-7. The LED ON time feature can be enabled or disabled on each GPO pin. LED ON time is global for all GPOs and can range from 0 to 2000 ms, with a resolution of 20 ms. The LED ON time feature is not applicable for the CY8CMBR3002 CapSense controller.

The toggle and LED ON time features are mutually exclusive. If both features are enabled, the toggle feature takes the higher priority and so the LED ON time feature will be disabled.

Figure 5-7. LED ON Time



5.1.2.8 Analog Voltage Output

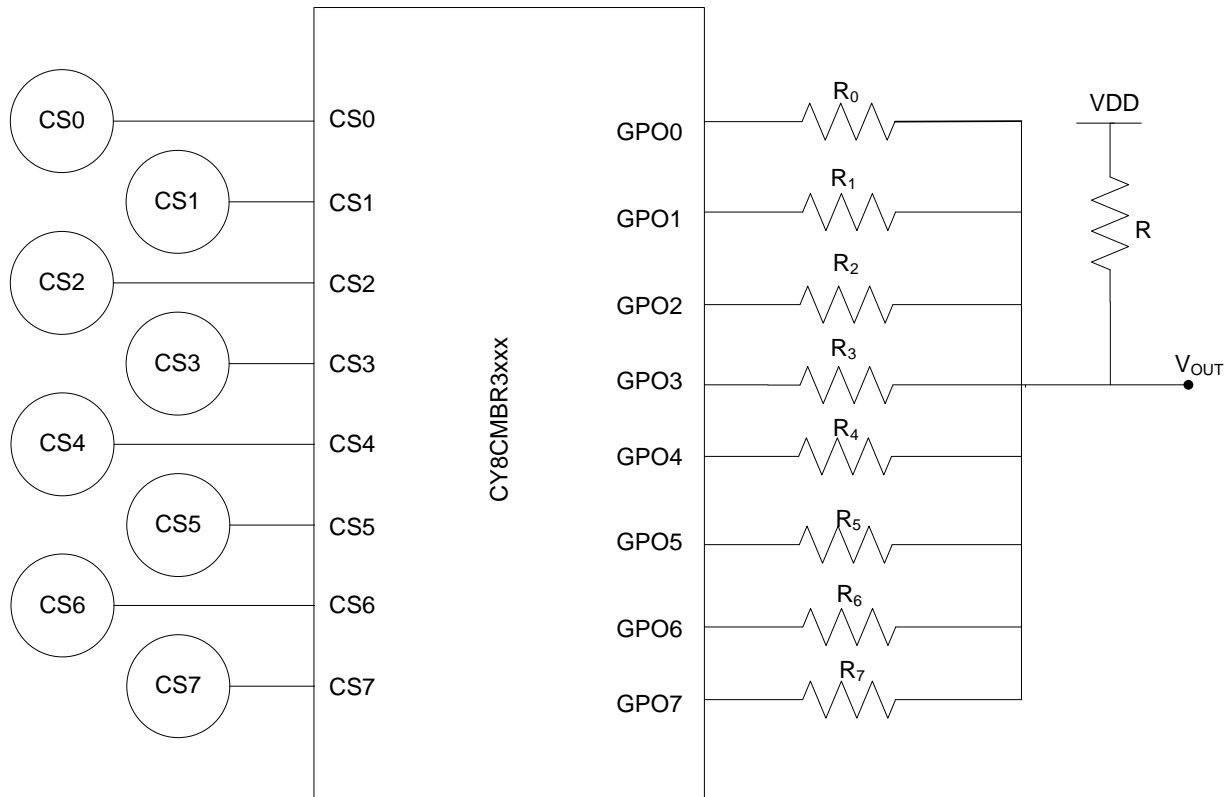
The analog voltage output feature uses a resistor network to indicate the sensor status as an analog voltage to the host, as Figure 5-8 shows. In this configuration, the GPO drive mode should be set to open-drain LOW-drive mode. When the sensor is touched, the corresponding GPO is driven to a logic LOW signal and a voltage divider is formed at V_{OUT} . When the sensor is released, the corresponding GPO is in HI-Z state and the voltage V_{OUT} will be at V_{DD} . The FSS feature can be enabled so that only one button is reported ON at a time.

The analog voltage V_{OUT} is given by:

$$V_{out} = \frac{V_{DD} \times R_n}{R + R_n} \quad \text{Equation 3}$$

Where: R is the resistance between V_{DD} and V_{OUT}
 R_n is the effective resistance between V_{OUT} and Ground

Figure 5-8. Analog Voltage Output Using GPO and Resistor Network



5.1.3 Buzzer Configuration

5.1.3.1 Buzzer Output

CY8CMBR3xxx supports a single-input Piezo-buzzer. The buzzer is driven by a PWM signal; its frequency is configurable. To enable the buzzer, select the buzzer pin in [EZ-Click](#). This feature is not applicable for the CY8CMBR3102 and CY8CMBR3002 CapSense controllers.

The CY8CMBR3xxx supports buzzer frequencies of 1.0, 1.1, 1.3, 1.6, 2.0, 2.6, and 4.0 kHz to meet different Piezo-buzzer drive requirements and to provide different tones. The duty cycle of buzzer output is fixed at 50%.

5.1.3.2 Buzzer ON Time

The Buzzer ON Time feature allows the user to specify the duration for which the buzzer output should be active when the sensor is activated, as [Figure 5-9](#) shows. The buzzer signal output is driven for the configured time and does not depend on the sensor touch time. The output goes to the idle state after the Buzzer ON-Time elapses, even if the sensor is active.

The buzzer signal output does not restart if the same sensor or any other sensor is touched before the Buzzer ON - Time elapses, as [Figure 5-10](#) and [Figure 5-11](#) show. The Buzzer ON-Time can range from 100 ms to 12.7 s with a resolution of 100 ms.

Figure 5-9. Buzzer Activation on Touch Event

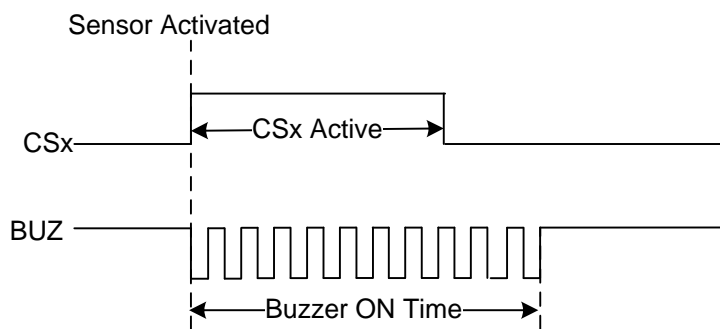


Figure 5-10. Buzzer Operation with Consecutive Touches of the Same Sensor

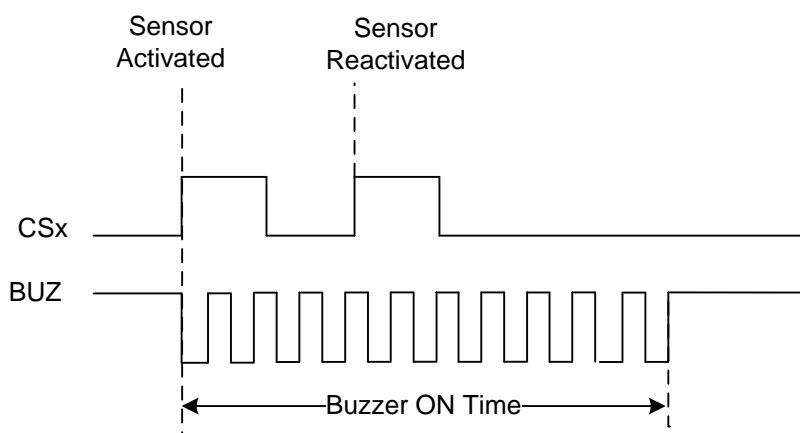
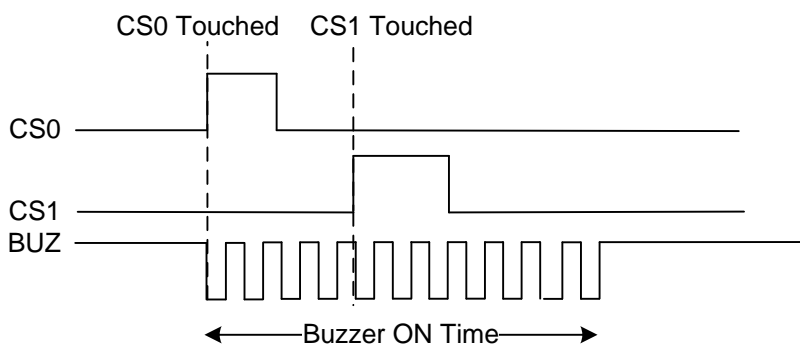


Figure 5-11. Buzzer Operation When Multiple Sensors Are Activated



5.1.4 Device Configuration

5.1.4.1 Supply Voltage

The supply voltage of the CY8CMBR3xxx controller should be specified in [EZ-Click](#) for proper operation. If the operating voltage is in the range of 1.71 to 1.89 V, select the 1.8(±5%) V option; otherwise select the 1.8-5.5 V option. This feature is not applicable to the CY8CMBR3002 CapSense controller.

5.1.4.2 I²C Address

The CY8CMBR3xxx controller communicates to the host processor through the I²C interface. In a typical I²C communication, there will be more than one slave device on the I²C bus. To avoid conflicts in the I²C address among the slave devices, CY8CMBR3xxx provides an option to specify the I²C address of the controller. The I²C address is the 7-bit value and can range from 8 (0x08) to 119 (0x77). The CY8CMBR3002 controller does not support I²C communication.

5.1.4.3 Scan Period

The scan period parameter specifies the amount of time between successive sensor scans in [Look-for-Touch Mode](#) and [Look-for-Proximity Mode](#). This parameter can range from 20 to 500 ms with a resolution of 20 ms. This parameter is fixed at 120 ms for CY8CMBR3002 CapSense controller.

The response time and power consumption of the controller depends on the scan period. See the [Low-Power Design Considerations](#) section for more details.

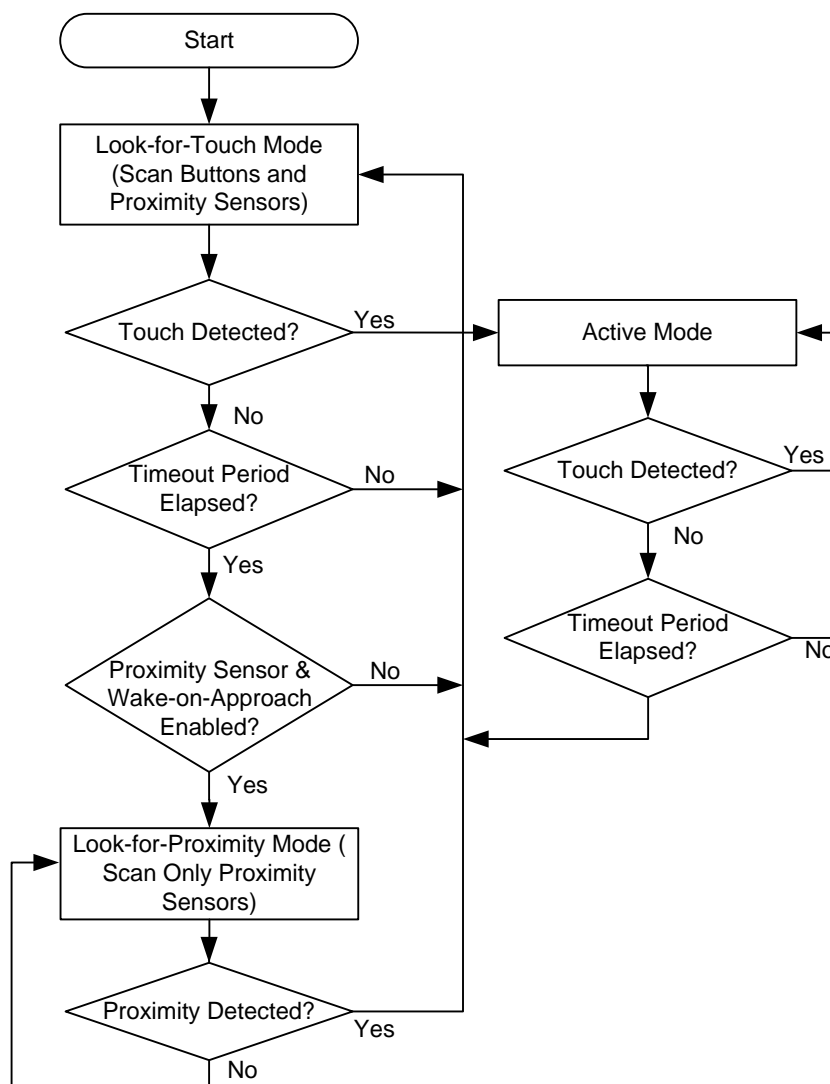
5.1.4.4 State Timeout

The State Timeout interval decides the time taken by the device to transition from one operating mode to another when the sensors are not touched. This parameter affects the response time of the sensors for consecutive touches. This parameter should be set based on the design requirements. This allows the controller to stay in the active mode and provide a fast response to consecutive touches. This parameter can range from 0 to 63 s with a resolution of 1 s. This parameter is fixed at 10 s for CY8CMBR3002 CapSense controller.

5.1.4.5 Wake-on-Approach

To reduce power consumption, CapSense designs use the wake-on-approach feature. This feature allows the device to enter the [Look-for-Proximity Mode](#) when proximity sensors are enabled. In the look-for-proximity mode, only proximity sensors are scanned; when proximity is detected, the controller enters the [Look-for-Touch Mode](#) in which all the sensors are scanned, as shown in [Figure 5-12](#). See the [Low-Power Design Considerations](#) section for more details. This feature is not applicable to the CY8CMBR3002 CapSense controller.

Figure 5-12. Transition From Look-for-Touch Mode to Look-for-Proximity Mode



5.2 Configuring CY8CMBR3xxx

The CY8CMBR3xxx family is a register-configurable CapSense controller. This controller has 128 bytes of configuration registers starting from address 0x00 to 0x7E. Except the CY8CMBR3002 CapSense controller, all other controllers in the CY8CMBR3xxx family can be configured through the I²C interface. The configuration data can be saved to nonvolatile memory so that controller retains the configuration data even when it is powered-off.

For more information on the CY8CMBR3xxx registers, refer to the [CY8CMBR3xxx Registers TRM](#).

The CY8CMBR3xxx configuration registers are configured via I²C interface using one of the following methods:

1. EZ-Click 2.0
2. Third-party Programmer
3. Host APIs
4. Bridge Control Panel

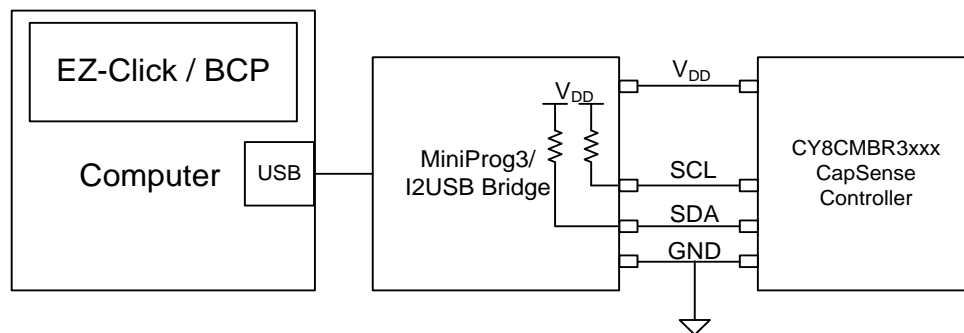
5.2.1 Configuring CY8CMBR3xxx Using EZ-Click

EZ-Click is a simple and intuitive graphical user interface (GUI) used to configure the CY8CMBR3xxx CapSense controller. It takes all the required parameters, and configures the device via I²C interface.

5.2.1.1 Hardware Connection

Figure 5-13 shows the hardware setup required to configure the CY8CMBR3xxx CapSense controller using EZ-Click 2.0 or Bridge Control Panel through an I²C interface. The EZ-Click/BCP communicates to the CY8CMBR3xxx controller using either MiniProg3 or CY3240-I2USB Bridge. The connection between the MiniProg3 or I2USB Bridge and the CY8CMBR3xxx controller is shown in Figure 5-13. Because the SCL and SDA pins in the MiniProg3 or I2USB Bridge are internally pulled to V_{DD} (refer to the MiniProg3/I2USB Bridge kit user guide for the pull-up resistor values), you should not have any external pull-up resistors connected on these lines. It is not mandatory to supply V_{DD} to the CY8CMBR3xxx controller from the MiniProg3 or CY3240-I2USB Bridge. The CY8CMBR3xxx controller can also be externally powered. In such cases, in the EZ-Click and BCP, ensure that the power is turned ON and the voltage settings for the MiniProg3/I2USB Bridge is selected as same as the CY8CMBR3xxx V_{DD}.

Figure 5-13. Hardware Setup Required for Configuring CY8CMBR3xxx Using EZ-Click or BCP



Note: In the CY3280-MBR3 Kit, the PSoC 5LP device is used as an I2USB Bridge and therefore, you do not need the MiniProg3 or CY3280-I2USB Bridge to communicate with the CY8CMBR3116 CapSense controller. See CY3280-MBR3 Kit User Guide for more details.

5.2.1.2 EZ-Click

This section provides a quick introduction to **EZ-Click**. Refer to the **EZ-Click user guide** for details on how to configure the CY8CMBR3xxx controller.

EZ-Click provides the following features:

- A Graphical User Interface (GUI) to configure the CY8CMBR3xxx controller.
- Allows tuning the sensor parameters and viewing the sensor debug data in real time.
- Supports production-line testing and displays system diagnostics results and the CapSense sensor SNR.
- Allows to save the configuration and use it on multiple samples.
- Generates a header file which consists of an array with 128 bytes of configuration data to configure the CY8CMBR3xxx controller using Host APIs, as Figure 5-16 shows. For more details, go to Section 5.2.3
- Generates hex file of your configuration to configure the device using third-party programming tools. For more details, refer to section 5.2.2
- Generates the configuration file, including the required I²C instructions to configure the device using the Bridge Control Panel, as Figure 5-23 shows. For more details, refer to section 5.2.4

EZ-Click has five tabs, as Figure 5-15 shows:

- **Start Page:** This page shows the steps to create a new project, generate configuration files, and configure the MBR family of controllers.
- **CapSense sensor configuration:** In this tab, the number of sensors can be selected and various parameters and features can be set for each sensor.
- **Global configuration:** In this tab, the GPOs can be enabled and the global parameters can be specified.

- CapSense output: In this tab, the sensor debug data can be monitored for debugging purpose.
- System diagnostics: In this tab, the system diagnostics results can be viewed for each sensor.

Note: The “CapSense sensor configuration”, “Global configuration”, “CapSense output”, and “System diagnostics” tab can be accessed only if a new project is created or an existing project is opened.

Figure 5-14. Configuring CY8CMBR3xxx Using Various Files Generated by EZ-Click

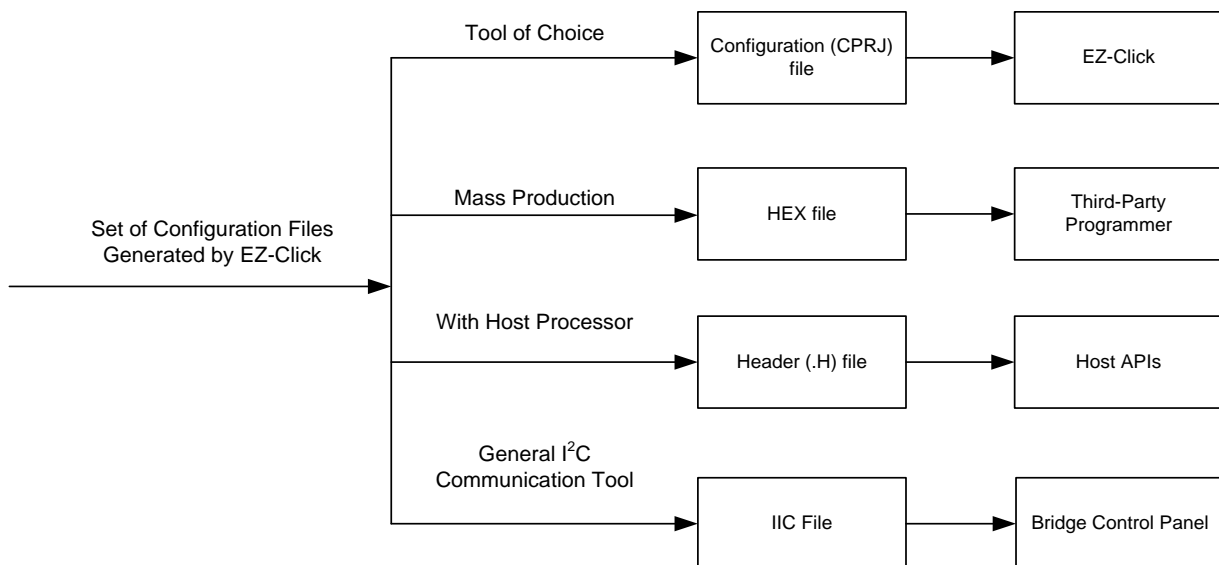
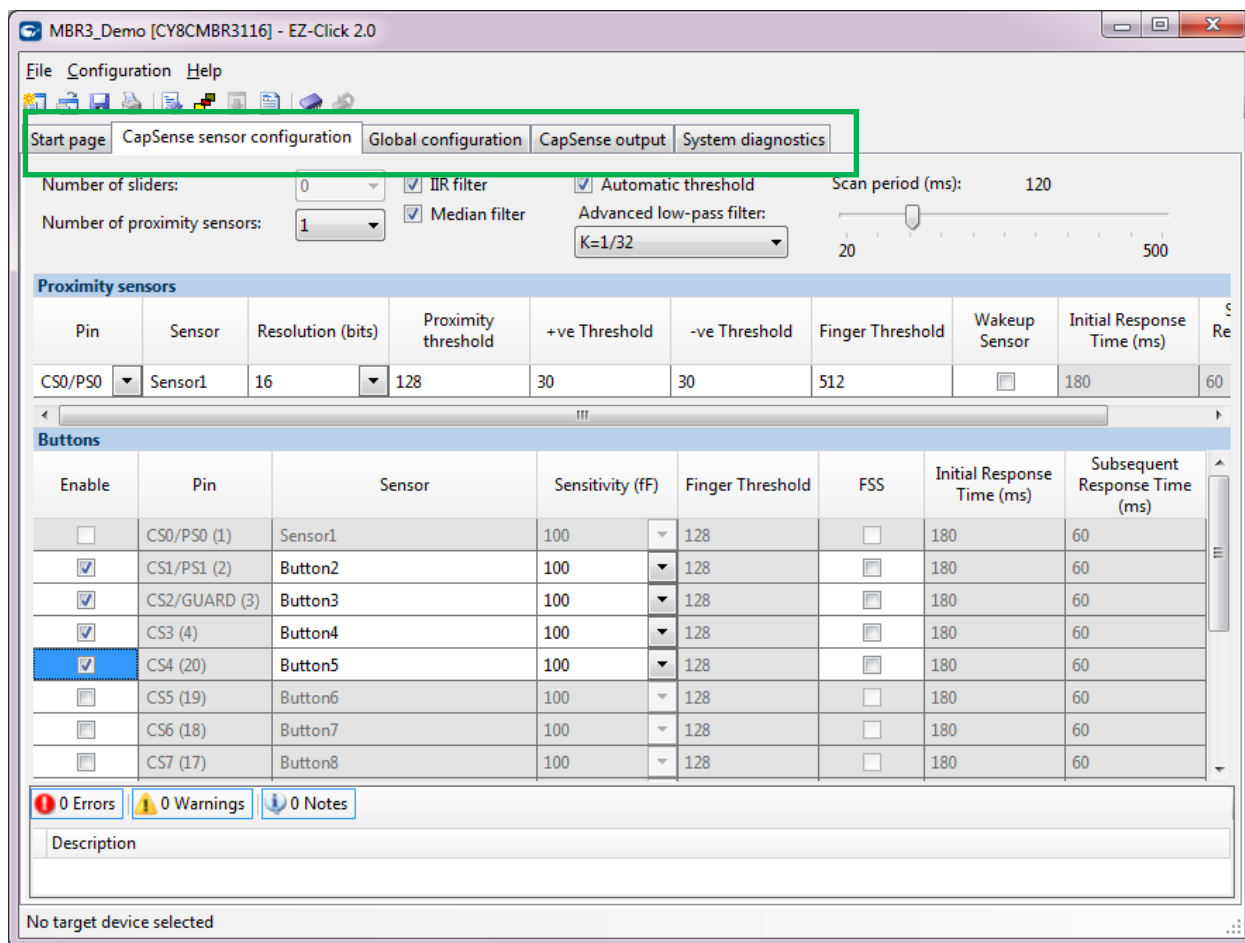


Figure 5-15. EZ-Click Customizer Tool



MBR3_Demo [CY8CMBR3116] - EZ-Click 2.0

File Configuration Help

Start page CapSense sensor configuration Global configuration CapSense output System diagnostics

Number of sliders: 0 ☐ IIR filter ☒ Automatic threshold Scan period (ms): 120

Number of proximity sensors: 1 ☒ Median filter Advanced low-pass filter: K=1/32 20 500

Proximity sensors

Pin	Sensor	Resolution (bits)	Proximity threshold	+ve Threshold	-ve Threshold	Finger Threshold	Wakeup Sensor	Initial Response Time (ms)	Re
CS0/PS0	Sensor1	16	128	30	30	512	<input type="checkbox"/>	180	60

Buttons

Enable	Pin	Sensor	Sensitivity (fF)	Finger Threshold	FSS	Initial Response Time (ms)	Subsequent Response Time (ms)
<input type="checkbox"/>	CS0/PS0 (1)	Sensor1	100	128	<input type="checkbox"/>	180	60
<input checked="" type="checkbox"/>	CS1/PS1 (2)	Button2	100	128	<input type="checkbox"/>	180	60
<input checked="" type="checkbox"/>	CS2/GUARD (3)	Button3	100	128	<input type="checkbox"/>	180	60
<input checked="" type="checkbox"/>	CS3 (4)	Button4	100	128	<input type="checkbox"/>	180	60
<input checked="" type="checkbox"/>	CS4 (20)	Button5	100	128	<input type="checkbox"/>	180	60
<input type="checkbox"/>	CS5 (19)	Button6	100	128	<input type="checkbox"/>	180	60
<input type="checkbox"/>	CS6 (18)	Button7	100	128	<input type="checkbox"/>	180	60
<input type="checkbox"/>	CS7 (17)	Button8	100	128	<input type="checkbox"/>	180	60

0 Errors 0 Warnings 0 Notes

Description

No target device selected

Figure 5-16. C Header File Generated by the EZ-Click Customizer Tool

```

1  /* Project: C:\MBR3_Demo\MBR3_Demo.cprj
2  * Generated: 02/18/2014 11:20:08 AM +05:30 */
3  const unsigned char CY8CMBR3116_configuration[128] = {
4      0x78u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u,
5      0x00u, 0x00u, 0x00u, 0x00u, 0x7Fu, 0x7Fu, 0x7Fu, 0x80u,
6      0x80u, 0x80u, 0x80u, 0x7Fu, 0x7Fu, 0x7Fu, 0x7Fu, 0x7Fu,
7      0x7Fu, 0x7Fu, 0x7Fu, 0x7Fu, 0x03u, 0x00u, 0x00u, 0x00u,
8      0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x80u,
9      0x05u, 0x00u, 0x00u, 0x02u, 0x00u, 0x02u, 0x00u, 0x00u,
10     0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u,
11     0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x01u, 0x01u,
12     0x00u, 0xFFu, 0xFFu, 0xFFu, 0xFFu, 0xFFu, 0xFFu, 0xFFu,
13     0xFFu, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x03u, 0x01u, 0x58u,
14     0x00u, 0x37u, 0x06u, 0x00u, 0x00u, 0x0Au, 0x00u, 0x00u,
15     0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u,
16     0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u,
17     0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u,
18     0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u,
19     0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x00u, 0x41u, 0x41u
20 };

```

5.2.2 Configuring CY8CMBR3xxx Using a Third-Party Programmer

To configure large number of devices rapidly during mass production, Cypress recommends using a third-party programmer from RPM Systems Corporation. To configure the CY8CMBR3xxx controller using the third-party programmer, use the hex file of the configuration generated by [EZ-Click](#). Contact [RPM Systems Corporation](#) for further information.

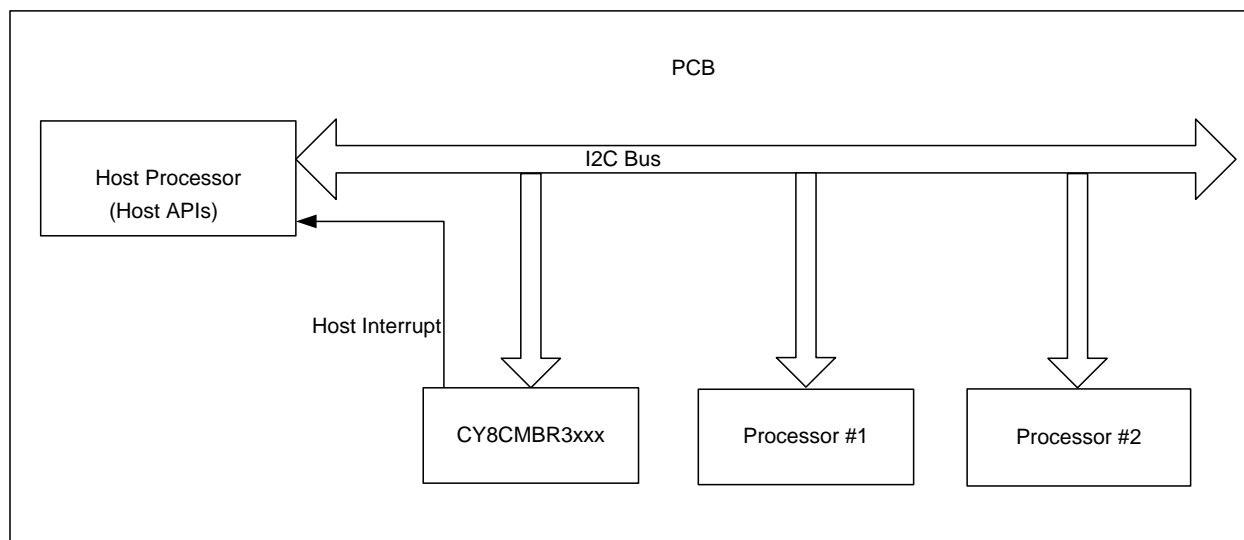
For information on the CY8CMBR3xxx family programming specifications refer to the [CY8CMBR3xxx Device Programming Specifications](#).

5.2.3 Configuring CY8CMBR3xxx Using Host APIs

[Figure 5-17](#) shows a system where the host processor communicates via I²C interface with the CY8CMBR3xxx controller along with few other processors on the same PCB. The host can read back the sensor status from the CY8CMBR3xxx controller during run-time, tune its operation if required, and also reconfigure it to suit a different requirement.

To configure the CY8CMBR3xxx controller on-board using a host processor, Cypress provides a set of library files called “[Host APIs](#)”, which can be integrated into the host processor firmware. This section provides the description of Host APIs and explains how to use them to configure the CY8CMBR3xxx Controller.

Figure 5-17. System Demonstrating Host API Usage



5.2.3.1 Host APIs Overview

[Host APIs](#) are a set of C-style functions provided by Cypress to enable the host processor to interact with the CY8CMBR3xxx controller, directly via the I²C interface.

These APIs can be used to:

1. Configure the CY8CMBR3xxx controller
2. Issue I²C commands to the CY8CMBR3xxx controller
3. Read information from the CY8CMBR3xxx register map

5.2.3.2 Host API Description and Usage

Host APIs are divided into two sections: [High-Level APIs](#) and [Low-Level APIs](#).

5.2.3.2.1 High-Level APIs

High-level APIs implement the required functionality purely in C-style software, and are in turn dependent on low-level APIs for physical interaction with the CY8CMBR3xxx controller. [Table 5-2](#) lists the high-level APIs and their usage. The API prototypes, parameters, and return values are mentioned in the [Appendix](#).

Table 5-2. High-Level Host APIs

#	API	Description and Usage
1	CY8CMBR3xxx_WriteData()	General purpose API to write data to the device. Use this API to write to one or more registers of the CY8CMBR3xxx controller.
2	CY8CMBR3xxx_ReadData()	General purpose API to read data from the CY8CMBR3xxx controller. Use this API to read from one or more registers of the CY8CMBR3xxx controller.
3	CY8CMBR3xxx_WriteDualByte()	API to write to a two-byte register of the CY8CMBR3xxx controller. For example, the 16-bit sensor enable register can be written using this API.
4	CY8CMBR3xxx_ReadDualByte()	API to read from a two-byte register of the CY8CMBR3xxx controller. For example, the 16-bit sensor status register can be read using this API.
5	CY8CMBR3xxx_SendCommand()	API to issue commands to the CY8CMBR3xxx controller, such as to save configuration to nonvolatile memory, reset the CY8CMBR3xxx controller, etc.
6	CY8CMBR3xxx_CheckCommandStatus()	API to check the status of the last command sent to the CY8CMBR3xxx controller. It is recommended to call this API in the host processor before sending a new command to the CY8CMBR3xxx controller.
7	CY8CMBR3xxx_Configure()	API to configure the entire 128 bytes of configuration registers in the CY8CMBR3xxx controller. To use this API, create a configuration in EZ-Click and use the generated header file in the host firmware.
8	CY8CMBR3xxx_CalculateCrc()	API to calculate CRC checksum of a given configuration.
9	CY8CMBR3xxx_VerifyDeviceOnBus()	API to verify that the CY8CMBR3xxx device on the I ² C bus is the intended one ¹⁰ . Call this API from the host processor to confirm that the right I ² C slave device is on the bus.
10	CY8CMBR3xxx_SetDebugDataSensorId()	This API sets the sensor number for which the sensor debug data has to be read from the CY8CMBR3xxx controller.
11	CY8CMBR3xxx_ReadSensorDebugData()	API to read sensor debug data. This API helps in knowing the sensor performance.
12	CY8CMBR3xxx_ReadDiffCounts()	API to read the difference counts of all sensors of the CY8CMBR3xxx controller.
13	CY8CMBR3xxx_ReadSensorStatus()	API to read the status of all sensors of the CY8CMBR3xxx controller. When the CY8CMBR3xxx controller issues a Host Interrupt pulse , call this API to know which sensor is triggered.

5.2.3.2.2 Low-Level APIs

Low-level APIs are responsible for the physical communication between the host processor and the CY8CMBR3xxx device via I²C interface, and are thus hardware-dependent. These APIs support [PSoC4](#) architecture by default, and require to be modified to support a specific host processor.

¹⁰ The Host APIs cannot be used to configure different controllers of the CY8CMBR3xxx family present on the same I²C bus.

When the CY8CMBR3xxx controller is in [Deep Sleep Mode](#), the first I²C transaction will be NACKed. If the CY8CMBR3xxx controller NACKs the I²C transaction, these low-level APIs must retry twice within the next 340 ms for a successful transaction. Refer to the “Host Communication Protocol” in the [CY8CMBR3xxx Datasheet](#). To achieve this either disable all the interrupts in the host processor except for I²C, or ensure that all pending interrupts complete within 340 ms. If there is a delay of more than 340 ms between two subsequent I²C transactions, the device might go into [Deep Sleep Mode](#) and the host might receive a NACK.

There are three low-level APIs – Host_LowLevelWrite(), Host_LowLevelRead(), and Host_LowLevelDelay(). The API prototypes, parameters, and return values are mentioned in the [Appendix](#).

Table 5-3. Low-Level Host APIs

#	API	Description and Usage
1	Host_LowLevelWrite()	API to write to the register map of the CY8CMBR3xxx controller via I ² C interface.
2	Host_LowLevelRead()	API to read data from the register map of the CY8CMBR3xxx controller via I ² C interface.
3	Host_LowLevelDelay()	This API implements a time-delay function to be used by the High-level APIs. The delay period is in milliseconds. This delay is achieved by a code-execution block for the required amount of time.

5.2.3.3 Host API File Structure

The files listed in [Table 5-4](#) form the library of Host APIs.

Table 5-4. File Structure for Host APIs

#	File Name	Description
1	CY8CMBR3xxx_Device.h	This file contains the macros to define the various devices in CY8CMBR3xxx family. Also, it defines which device is currently being used for the Host APIs.
2	CY8CMBR3xxx_Registers.h	This file contains the macros to define the offset addresses for all the registers of the CY8CMBR3xxx family. Use these macros to refer to the registers, in place of their addresses directly. Each register's macro is accessible only if the register is available for the currently used device.
3	CY8CMBR3xxx_CommandsAndConfig.h	This file contains the macros to define the various op-codes of the command register (CTRL_CMD).
4	CY8CMBR3xxx_APIS.h	This file contains the declarations of all the high-level APIs. This file also declares the structure definitions for holding the sensor debug data and sensor status read from the device.
5	CY8CMBR3xxx_CRC.c	This file contains the API to calculate CRC for a given configuration of CY8CMBR3xxx device.
6	CY8CMBR3xxx_APIS.c	This file contains the definitions of all the high-level APIs, except the API to calculate CRC.
7	CY8CMBR3xxx_HostFunctions.h	This file contains the declarations of the host-dependent, low-level APIs.
8	CY8CMBR3xxx_HostFunctions.c	This file contains the definitions of the low-level APIs. Modify the content of these APIs to suit the host processor's I ² C implementation.

5.2.3.4 Macros

There are two macros “CY8CMBR3XXX_DEVICE” and “CY8CMBR3xxx_SYNC_COUNTER_MATCH_RETRY” defined in “CY8CMBR3xxx_Device.h” and “CY8CMBR3xxx_APIs.h” respectively. These two macros need to be modified to define the CY8CMBR3xxx controller and the I²C retry counts in case of sync counter mismatch.

1. CY8CMBR3xxx_DEVICE

This macro defines the controller accessed by the host. This controller can be one of the different controllers in the CY8CMBR3xxx family, except for CY8CMBR3002 (since this controller does not support I²C communication). The different controllers in CY8CMBR3xxx family are defined in CY8CMBR3xxx_Device.h.

2. CY8CMBR3xxx_SYNC_COUNTER_MATCH_RETRY

This macro defines the number of times the APIs CY8CMBR3xxx_ReadSensorDebugData() and CY8CMBR3xxx_ReadDiffCounts() retry I²C communication with the CY8CMBR3xxx controller in case the sync counters do not match during a read operation. Typically the read operation will be successful in the second attempt, if not in the first attempt. It is recommended to set a value which is < 10. To know more about sync counters, refer to [CY8CMBR3xxx Registers TRM](#).

5.2.3.5 Demo Project and its Usage

To quickly get familiarized with using Host APIs, a demo project is created with PSoC 4 as the host processor. This demo showcases how to use Host APIs to configure the CY8CMBR3116 controller on the [CY3280-MBR3 Evaluation Kit](#) using the [PSoC4 Pioneer Kit](#).

The following hardware is required for this demo:

- [CY3280-MBR3 Evaluation Kit](#)
- [CY8CKIT-042 PSoC4 Pioneer kit](#)
- USB A to mini-B cable
- PC/Laptop

The following software is required for this demo:

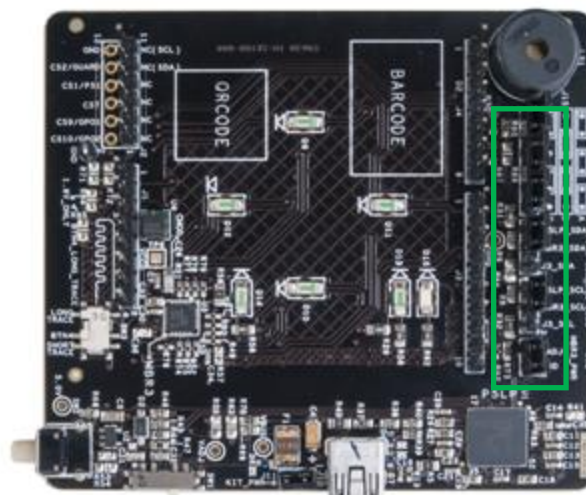
- [PSoC Programmer 3.20.0](#)

Note: The kit driver for the PSoC4 pioneer kit should already have been installed. Refer to the [CY8CKIT-042 PSoC 4 Pioneer Kit Guide](#) for details on installing kit driver.

To configure the MBR3 Evaluation Kit using the [PSoC4 Pioneer Kit](#) follow this procedure:

1. Change the jumper position on header J13 on MBR3 Evaluation Kit to connect pins J13-2 and J13-3.
2. Change the jumper position on header J14 on MBR3 Evaluation Kit to connect pins J14-2 and J14-3.
3. Change the jumper position on header J15 on MBR3 Evaluation Kit to position A. This will enable the kit to support Buzzer and Host Interrupt output.

Figure 5-18. Jumper J13, J14 and J15 Location on the MBR3 Evaluation Kit



4. Mount the MBR3 Kit on top of the PSoC4 Pioneer Kit, such that the USB ports of both kits align with each other. When the USB ports are aligned, the J1, J2, J3, and J4 headers on the PSoC 4 Pioneer Kit connect to J1, J2, J3, and J4 headers on the CY3280-MBR3 EVK.
5. Connect the PSoC4 Pioneer Kit to the PC via the USB A to mini-B cable, available as part of the MBR3 Evaluation Kit. The power LED (red) of the MBR3 kit glows. The final connection is shown in Figure 5-19. Once the hardware is set up, the project can be downloaded on to the PSoC4 host

Figure 5-19. MBR3 Kit Mounted on Top of PSoC4 Pioneer Kit



6. Download the “CY8CMBR3xxx_Host_APIs_Demo_Project.zip” project from [here](#) and extract the contents from the .zip file.
7. Open PSoC Programmer. It is located at Windows start menu > All programs > Cypress > PSoC Programmer 3.20.0. If PSoC Programmer is not installed in your PC, download it from [PSoC Programmer 3.20.0](#) and install.
8. In the PSoC Programmer click on File > File Load and navigate to the directory where the “CY8CMBR3xxx_Host APIs_Demo_Project” is extracted.

9. Navigate to CY8CMBR3xxx_Host_APIs_Demo_Project > PSoC4_Pioneer_Kit_Demo > PSoC4_Pioneer_Kit_Demo.cydsn > CortexM0 > ARM_GCC_473 > Release > PSoC4_Pioneer_Kit_Demo.hex and click on open.
10. Make sure that PSoC Programmer is connected to KitProg as shown in [Figure 5-20](#). Click on the Program button to program the [PSoC4 Pioneer Kit](#) with the given hex file, as shown in [Figure 5-20](#).
11. Once the PSoC4 Pioneer Kit is programmed, the PSoC4 Pioneer Kit configures the CY8CMBR3116 controller on the MBR3 Evaluation Kit with the demo configuration. If the CY8CMBR3116 controller is successfully configured, the RGB LED D9 on the PSoC4 Pioneer Kit glows Green. If the CY8CMBR3xxx controller is not successfully configured, the RGB LED D9 on the PSoC4 Pioneer Kit glows Red.

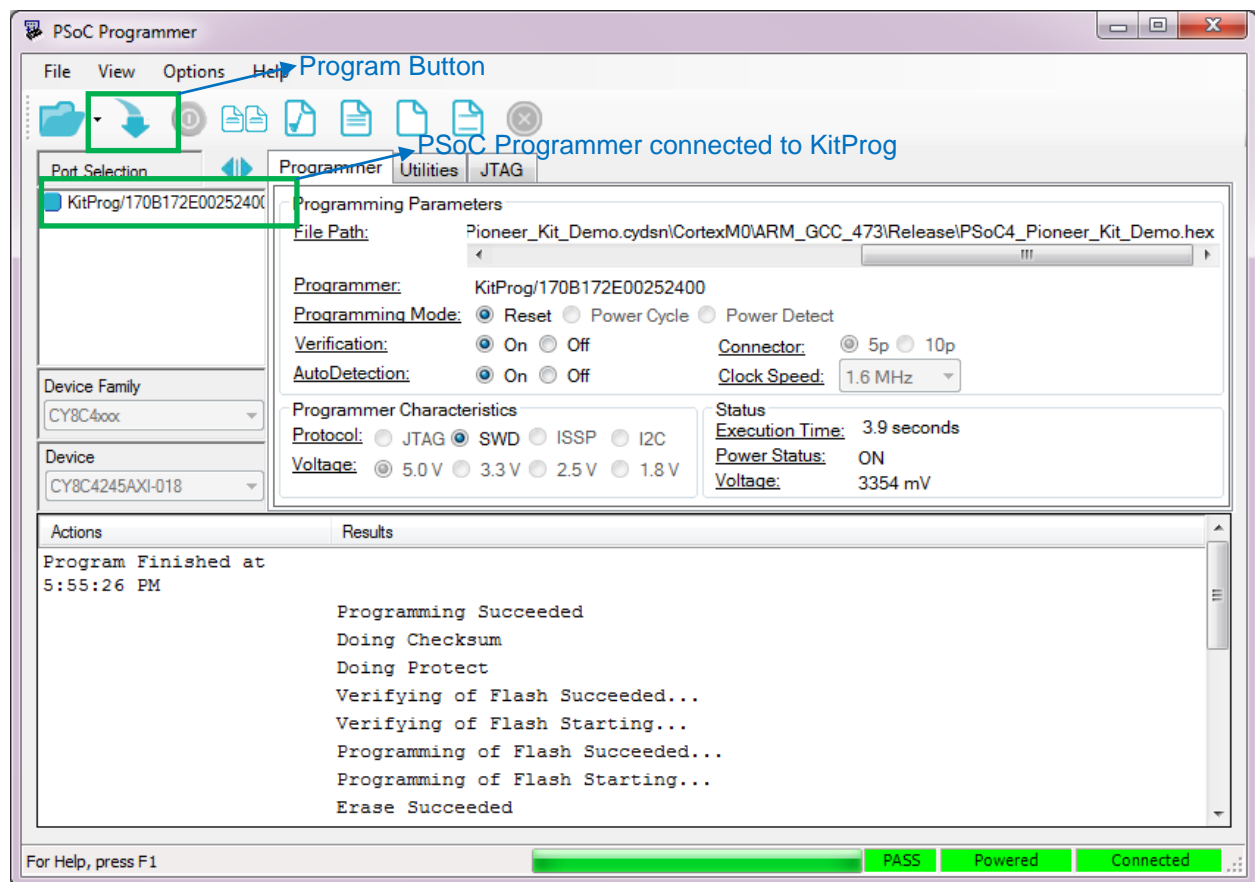
The MBR3 Evaluation Kit is configured for the following:

- Four CapSense button sensors
- One proximity sensor
- One proximity LED
- Buzzer output
- Host Interrupt output
- Host controlled GPOs.

The LEDs in the MBR3 Evaluation Kit are turned ON/OFF by the host processor (PSoC 4) whenever the CY8CMBR3116 controller sends a Host Interrupt pulse. When a Host Interrupt is triggered, the host reads the status of all the sensors of the CY8CMBR3116 controller and turns ON the corresponding LED if the sensor status is set to '1'.

12. After the PSoC 4 host configures the CY8CMBR3116 controller, verify the following features on the MBR3 Evaluation Kit:
 - Proximity: Hover your hand/finger (3-cm distance) over the MBR3 Evaluation kit; the proximity LED blinks once, to indicate proximity detection. Touch the proximity sensor (marked by a white rectangle at the edge of the MBR3 Kit); the buzzer beeps for 500 ms and the proximity LED remains ON as long as the proximity sensor is touched. When the hand/finger is no longer touching the proximity sensor, the proximity LED turns OFF to indicate that proximity touch is no longer detected. Moving the hand/finger further away results in the proximity LED blinking once more to indicate that the proximity is no longer detected.
 - CapSense buttons: Touch any CapSense button; the corresponding LED turns ON and buzzer beeps for 500 ms.

Figure 5-20. Programming the PSoC4 Pioneer Kit Using PSoC Programmer



5.2.4 Configuring CY8CMBR3xxx Using Bridge Control Panel

The [Bridge Control Panel](#) (BCP) is a software tool that can be used as I²C Master to communicate with the I²C slave devices using CY3240-I2USB or MiniProg3 hardware, as [Figure 5-21](#) shows.

 Figure 5-21. Bridge Control Panel Communication with I²C Slave Device


Note: The CY3280-MBR3 Evaluation kit does not require an additional I2USB/MiniProg3 hardware. This kit has an on-board I2C-USB bridge to connect to BCP. To connect the on-board I2C-USB bridge to BCP change the jumper position on header J13 on the MBR3 Evaluation Kit to connect pins J13-1 and J13-2 and on header J14 to connect pins J13-1 and J13-2.

This tool can be used to configure the CY8CMBR3xxx controller and read debug data using I²C Interface. It is highly recommended to use [EZ-Click](#) to configure the CY8CMBR3xxx controller instead of BCP. However, there might be situations where [BCP](#) can be used to configure the CY8CMBR3xxx controller. For example, to override the threshold parameter and manually specify them, use [BCP](#) tool. For more information on using the BCP tool, click on BCP "Help" menu in the BCP tool.

This section refers to various register names in the CY8CMBR3xxx controller. Refer to the [CY8CMBR3xxx Registers TRM](#) for definitions and address of these registers.





5.2.4.1 Procedure to Configure CY8CMBR3xxx Using Bridge Control Panel

The flowchart for configuring the CY8CMBR3xxx using the [BCP](#) tool is shown in [Figure 5-24](#). Refer to [Figure 5-13](#) for details on the hardware connection required for communicating with the CY8CMBR3xxx controller using BCP.

The procedure to configure the CY8CMBR3xxx using the [BCP](#) tool can be divided into four steps:

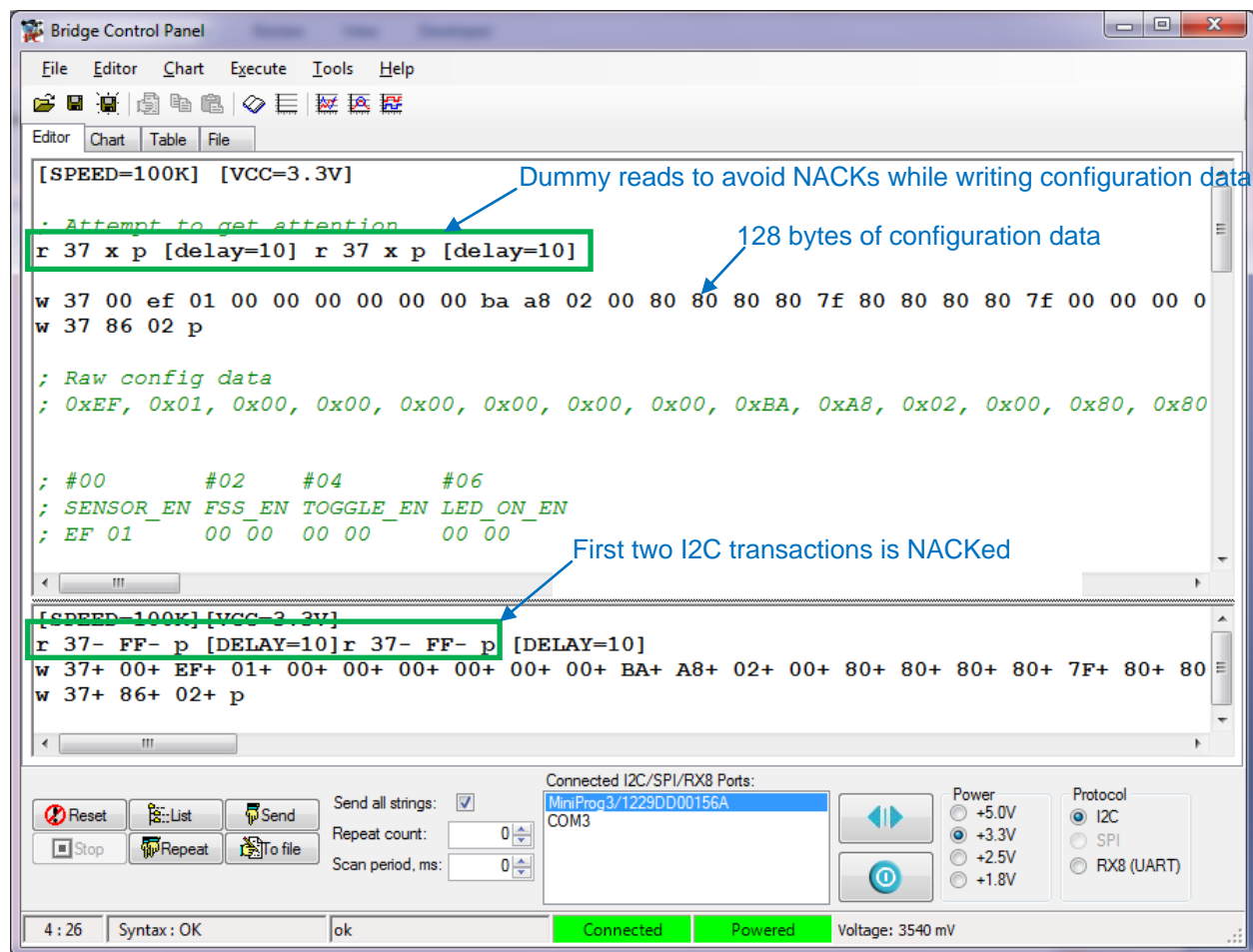
1. Send the configuration data to the CY8CMBR3xxx controller. The configuration data can be one of the following:
 - a) The configuration file (<project_name>.iic file) generated by the [EZ-Click](#):
 Configuring the CY8CMBR3xxx controller using the configuration file generated by EZ-Click is shown in [Figure 5-22](#). This method is rarely used since the CY8CMBR3xxx can be directly configured using [EZ-Click](#). Click on the “Help” option in the BCP tool for instructions on opening any IIC (configuration) file in the [BCP](#) tool and for sending the commands to the slave device.

Figure 5-22. IIC File generated by EZ-Click to Configure the CY8CMBR3xxx Controller using BCP

Name	Date modified	Type	Size
 MBR3_Demo			
 MBR3_Demo	02/17/2014 8:29 PM	H File	2 KB
 MBR3_Demo.hex	02/17/2014 8:29 PM	HEX File	1 KB
 MBR3_Demo.iic	02/17/2014 8:29 PM	IIC File	5 KB

.IIC file generated by EZ-Click

Figure 5-23. Example Configuration File Generated by the EZ-Click Customizer Tool



b) Custom commands entered in the BCP tool :

Configuring CY8CMBR3xxx by entering custom commands in BCP is shown in Figure 5-26. In this method all the 128-byte configuration registers or any specific register can be configured. When the CY8CMBR3xxx is configured using the custom command, it is required to compute the Cyclic Redundancy Check (CRC) of the first 126 bytes of the configuration data and write it to CONFIG_CRC (0x7E) register. This CRC value will be used by the CY8CMBR3xxx controller to verify the integrity of the configuration data.

Note: When an I²C command (read/write) is sent to the CY8CMBR3xxx controller it might NACK the I²C transaction. In this case insert dummy I²C read transactions while writing or reading any register, as shown in Figure 5-26.

The CRC value of the first 126 bytes of configuration data can be calculated in two ways:

- Use the CY8CMBR3xxx controller to calculate the CRC value: The CY8CMBR3xxx controller calculates the checksum of the configuration data when a CMD_OP_CODE value of '3' is written to the CTRL_CMD (0x86) register, as shown in Figure 5-25. The calculated CRC value will be stored in the CALC_CRC (0x94) register after 220 ms. The user has to read the CRC value from the CALC_CRC (0x94) register and manually write it to the CONFIG_CRC (0x7E) register. This method should be used only during testing and debugging the CY8CMBR3xxx controller. For configuring the CY8CMBR3xxx controller in production, use the [CRC API](#) provided with the Host APIs.
- Use the Host APIs to calculate the CRC value: Cypress provides Host APIs to configure the CY8CMBR3xxx using host processor. The Host APIs have source code, which can be used to compute the CRC value of the 126 bytes of the configuration data. The user should use this code to calculate the

CRC and write the calculated CRC value to the CONFIG_CRC (0x7E) register, as shown in [Figure 5-26](#). The procedure to calculate the CRC is explained in the section [5.2.4.2](#). It is highly recommended to use this method to calculate the CRC value.

2. Once the configuration data and the CRC value is written to the CY8CMBR3xxx controller registers, write a CMD_OP_CODE value of '2' to the CTRL_CMD (0x86) register to save the configuration data to the nonvolatile memory.
3. After writing a CMD_OP_CODE value of '2' to the CTRL_CMD (0x86) register, wait for 220 ms and read the CTRL_CMD_STATUS (0x88) register to check if the configuration data is successfully saved to nonvolatile memory.

If the configuration data is successfully saved to nonvolatile memory, the value in the CTRL_CMD_STATUS (0x88) register will be '0'. If the value in the CTRL_CMD_STATUS register is '0', issue reset command by writing CMD_OP_CODE value of 255 to the CTRL_CMD (0x86) register.

If the value in the CTRL_CMD_STATUS (0x88) register is '1', it means the configuration data is not saved to the nonvolatile memory. In this case read the CTRL_CMD_ERR (0x89) register to know the reason for failure to save the configuration data to nonvolatile memory.

- A value of '253' in the CTRL_CMD_ERR indicates write to nonvolatile memory failed. In this case re-write the configuration data and repeat from step 2.
- A value of '254' in the CTRL_CMD_ERR register indicates CRC value written to CONFIG_CRC (0x7E) is incorrect. In this case re-calculate the CRC for the entire 126 bytes of configuration data, write to the CONFIG_CRC (0x7E) register and repeat from step 2.
- A value of '255' in the CTRL_CMD_ERR register indicates the CMD_OP_CODE value written to the CTRL_CMD (0x86) register is not valid. In this case send a valid CMD_OP_CODE = '2' and repeat step 3.

Figure 5-24. Flowchart for Configuring the CY8CMBR3xxx Using BCP

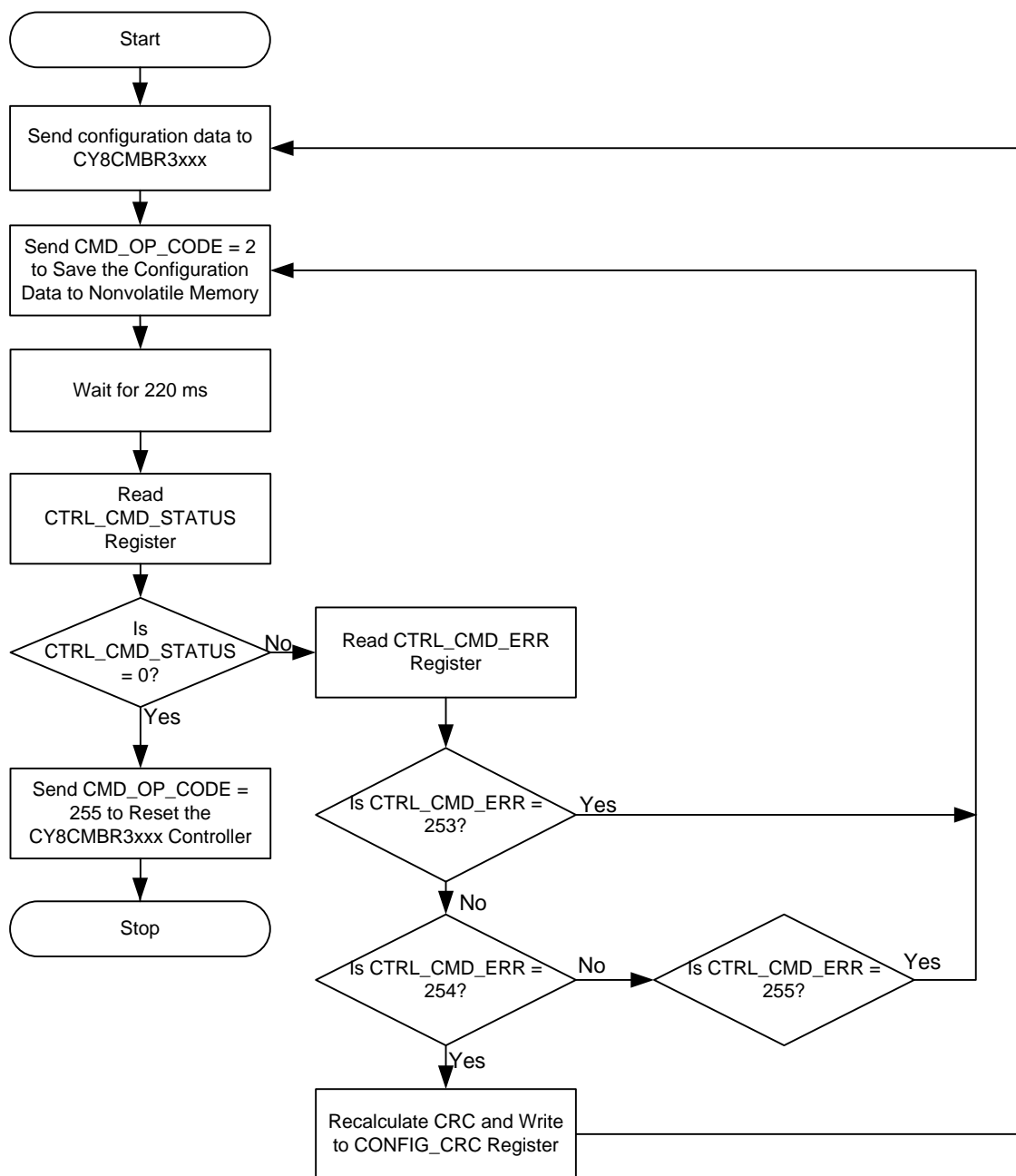
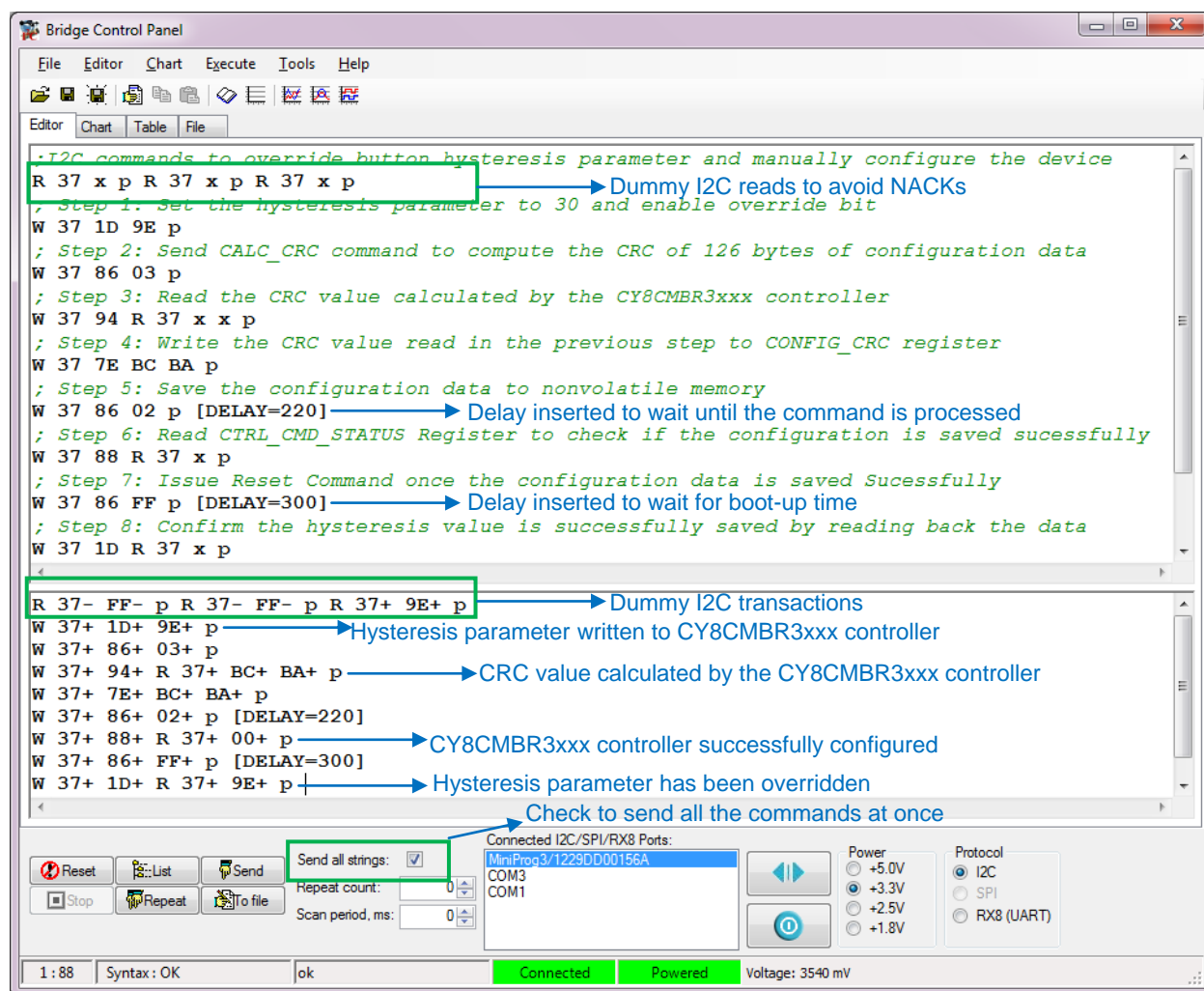


Figure 5-25. Calculating the CRC Using CY8CMBR3xxx Controller



Note: All the commands entered in the BCP can be sent at once by checking the “send all strings” box and clicking send button.


```

/* Variable to store the 16-bit CRC value*/
unsigned short crcValue;

/* Call the "CY8CMBR3xxx_CalculateCrc" API with configuration data address as the argument*/
crcValue = CY8CMBR3xxx_CalculateCrc(CY8CMBR3xxx_configuration);

/* Print the calculated 16-bit CRC value*/
printf("The CRC for the given configuration data is: 0x%x",crcValue);

return 0;
}

```

5. Read the 126 bytes of configuration data from the CY8CMBR3xxx controller using BCP and copy this into a text editor software (such as Notepad) to format the data i.e. to remove all the "+" symbols from the copied array and replace it with ",0x" character in front of each byte data as shown in [Figure 5-27](#) and [Figure 5-28](#).

Figure 5-27. CY8CMBR3xxx Configuration Data Copied from BCP to Notepad

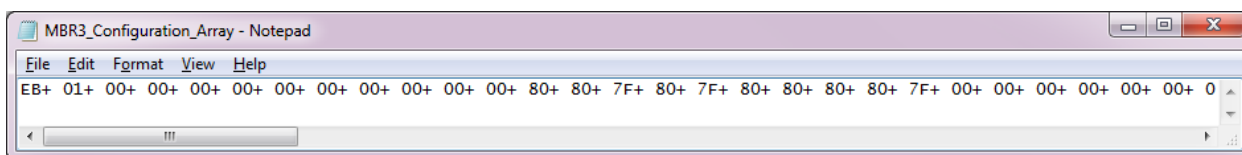
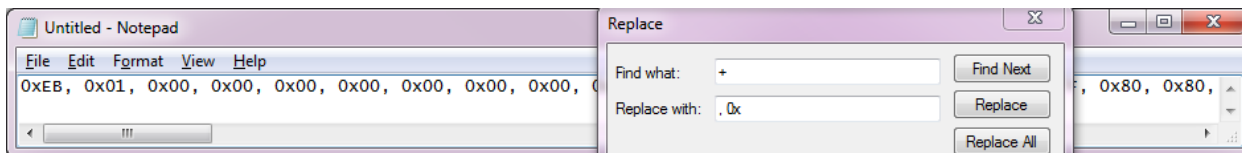


Figure 5-28. CY8CMBR3xxx Configuration Data Formatted in Notepad



6. Copy this formatted array into the "CY8CMBR3xxx_configuration [126]" array in the main.c, as shown in [Figure 5-28](#).
7. Save and build the project. Ensure there are no errors.
8. Once the project is successfully built, click the "Run" button to get the CRC value.

The screenshot shows the Eclipse IDE with the following components:

- Project Explorer (Left):** Shows the project structure. The file `main.c` is selected, and a green box highlights the `CY8CMBR3xxx_APIs.h` and `CY8CMBR3xxx_CRC.c` files.
- Main Editor:** Displays the source code for `main.c`. The code includes `<stdio.h>` and defines a 126-byte configuration data array. The main function calls the `CY8CMBR3xxx_CalculateCrc` API to compute the CRC for the 126 bytes of configuration data. The console output shows the result: `The CRC for the given configuration data is: 0xbabc`.
- Console (Bottom):** Shows the output of the program, which is `The CRC for the given configuration data is: 0xbabc`. A green box highlights this output.

6. CapSense Performance Tuning



As explained in the [CapSense Technology](#) chapter, Cypress' SmartSense Auto-tuning algorithm eliminates the need for manual tuning by automatically tuning the hardware and firmware parameters. However, the SmartSense algorithm requires setting the sensitivity parameter value of the sensor to ensure the SNR of all the sensors is greater than 5:1. The CY8CMBR3xxx allows manually setting the threshold parameters. This section explains how to set the sensitivity of the sensor and threshold parameters for reliable touch detection. The CY8CMBR3002 CapSense controller does not support manual tuning. In this controller the SmartSense algorithm automatically tunes all the hardware and firmware parameters. Therefore, the following section is not applicable to the CY8CMBR3002 controller.

6.1 General Considerations

6.1.1 Signal, Noise, and SNR

A well-tuned CapSense system reliably discriminates between ON and OFF sensor states. To achieve a robust 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). For more in-depth information on SNR refer to the "CapSense Technology" chapter in [Getting Started with CapSense](#). According to CapSense best practices, it is required to have a minimum SNR of 5:1 for all sensors. The SNR of the sensor is given by the following equation:

$$\text{SNR} = \frac{\text{Signal}}{\text{Peak-to-Peak Noise}} \quad \text{Equation 4}$$

Where: Signal = Raw count – Baseline

Peak-to-Peak Noise = Peak-to-Peak noise of raw count measured over 3000 samples

Therefore, the key to achieve a high SNR is to increase the signal and decrease the noise.

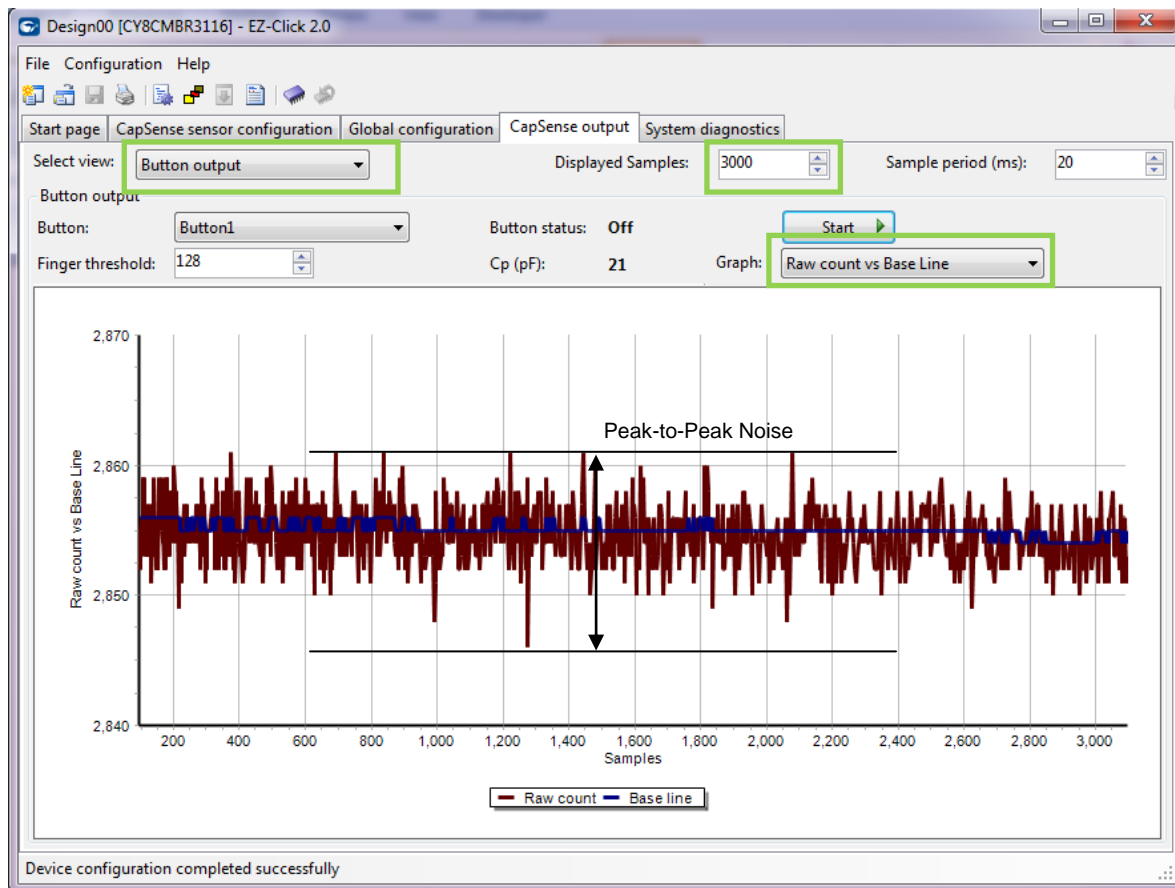
6.1.1.1 Noise Counts Measurement

To compute the SNR, it is required to measure the signal and noise counts of the sensor.

To measure the noise counts using [EZ-Click](#), follow these steps:

1. Generate the configuration file and download the configuration to the device.
2. Go to the "CapSense output" tab and select the "Raw count vs Baseline" graph. Set the "Displayed Samples" field to 3000.
3. Click on the "Start" button and wait until 3000 samples are acquired.
Note: The sensor should not be touched while acquiring raw count samples to measure noise counts.
4. Measure the noise count which is equal to the raw count (max) – raw count (min), as [Figure 6-1](#) shows. The noise counts for the graph shown in [Figure 6-1](#) is 2860 – 2845 = 15 counts.

Figure 6-1. Measuring Noise in EZ-Click Customizer



6.1.1.2 Signal Measurement

After the noise count is measured, it is required to measure the signal of the sensor to compute the SNR.

To measure the signal, follow these steps:

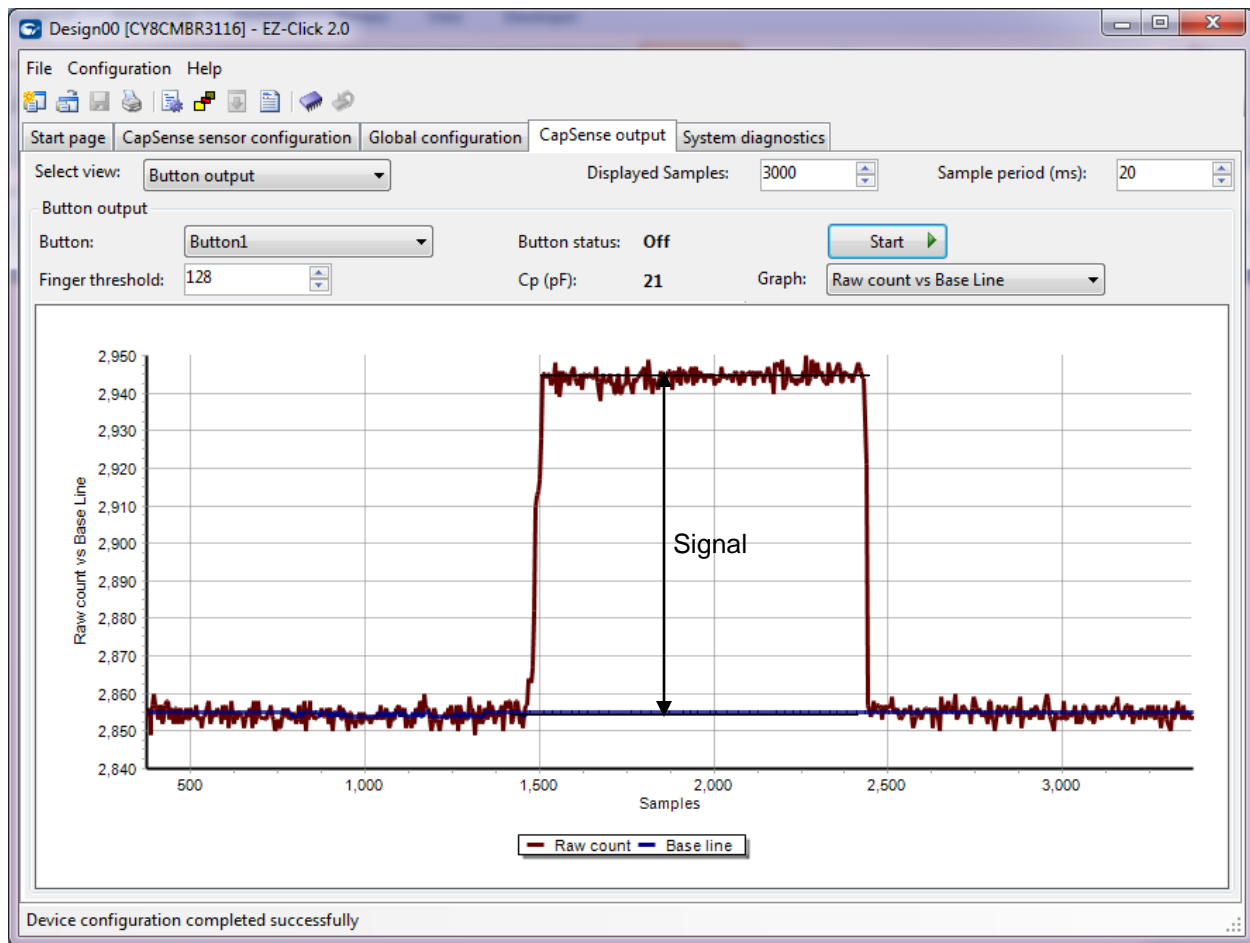
1. In the CapSense output tab, click on the “Start” button to view the “Raw count vs Baseline” graph.
2. Touch the sensor for 1000 samples and then release the finger.
3. Measure the shift in raw counts from the *no touch* to touch state, as [Figure 6-2](#) shows.

Signal = Average value of the raw count (When sensor is touched) - Average value of raw count (When sensor is not touched)

For the graph shown in [Figure 6-2](#), Signal = 2945 - 2855 = 90 counts

$$\text{SNR} = \frac{90}{15} = 6:1$$

Figure 6-2. Measuring the Signal in EZ-Click Customizer



6.2 Tuning the Buttons, Sliders, and Guard Sensor

Figure 6-3 shows the typical flow of tuning the buttons, sliders and guard sensors.

Tuning a sensor in the CY8CMBR3xxx controller is divided into two sections:

1. Ensure SNR is greater than 5:1
2. Set threshold parameters to the recommended value

6.2.1 Method to Ensure Minimum SNR

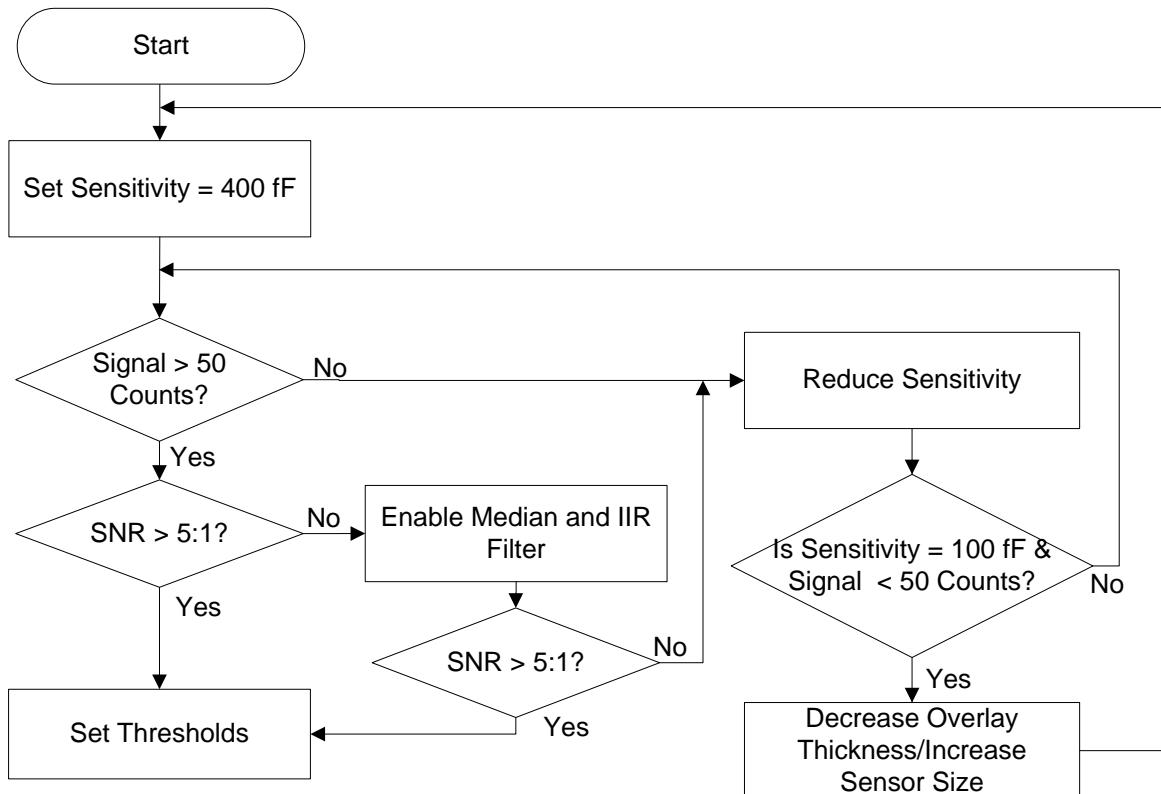
To ensure that the SNR is greater than 5:1, follow these steps:

1. Set the sensitivity of the sensor to 400 fF and measure the [signal](#).
2. If the signal is less than 50 counts, lower the sensitivity value until the signal is greater than 50 counts. If the signal is not greater than 50 counts even after setting the lowest sensitivity value (that is, 100 fF), reduce the overlay thickness or increase the sensor size.
3. When the signal is greater than 50 counts, measure the [noise counts](#). If the noise is greater than one-fifth of the signal, enable filters to reduce the noise. Enable the IIR filter and measure the noise counts. If the noise is still greater than one-fifth of the signal, enable the Median filter. If the median filter is enabled, the [Automatic Threshold](#) feature should be disabled and the [Finger Threshold](#) parameter should be manually specified.
4. Calculate the [SNR](#), and if it is greater than 5:1, proceed to set the threshold parameters.

5. If the SNR is not greater than 5:1 even after enabling filters, lower the sensitivity value and calculate the SNR. If the SNR is less than 5:1 after setting the lowest value of sensitivity (that is, 100 fF) and enabling filters, reduce the overlay thickness or increase the sensor size to achieve an SNR greater than 5:1.

Note: For sliders, the signal should always be less than 255 to get accurate centroid position. If the signal is greater than 255, increase the sensitivity parameter until the signal is less than 255.

Figure 6-3. Buttons, Sliders, and Guard Sensor Tuning



6.2.2 Button Threshold Parameters

After ensuring that all the sensors meet SNR greater than 5:1, the threshold parameters can be specified.

The following threshold parameters are applicable for buttons and guard sensor:

- Finger threshold
- Hysteresis
- Noise threshold
- Negative Noise threshold
- Low-Baseline Reset

Section 6.2.3 explains when and how to set these parameters.

6.2.2.1 Finger Threshold

The finger threshold parameter is used to judge the active/inactive state of a sensor. If the difference count value of a sensor is greater than the finger threshold + hysteresis, the sensor is judged as ON and if the difference count of the sensor is less than the finger threshold value – hysteresis, the sensor is judged as OFF. The finger threshold can range from 31 to 200; the default value is set to 128. See Table 6-2 to set this parameter.

The relationship between finger threshold and sensor ON/OFF state is given by:

$$\text{Sensor State} = \begin{cases} \text{ON} & \text{if } (\text{Signal} \geq \text{Finger Threshold} + \text{Hysteresis}) \\ \text{OFF} & \text{if } (\text{Signal} \leq \text{Finger Threshold} - \text{Hysteresis}) \end{cases} \quad \text{Equation 5}$$

6.2.2.2 Hysteresis

The hysteresis parameter is used along with the finger threshold parameter to determine the ON/OFF state of the sensor. Hysteresis provides immunity against noisy transitions of the sensor state. This parameter is applicable only for buttons and guard sensor. This parameter can range from 0 to 31; the default value is set to 12. See [Table 6-2](#) to set this parameter.

6.2.2.3 Noise Threshold

The noise threshold parameter sets the raw count limit above which the baseline is not updated, as [Figure 6-4](#) shows. In other words, the baseline remains constant as long as the raw count is above (baseline + noise threshold). This keeps the baseline from becoming too high due to a finger touch. This parameter can range from 0 to 127; the default value is set to 51. See [Table 6-2](#) to set this parameter.

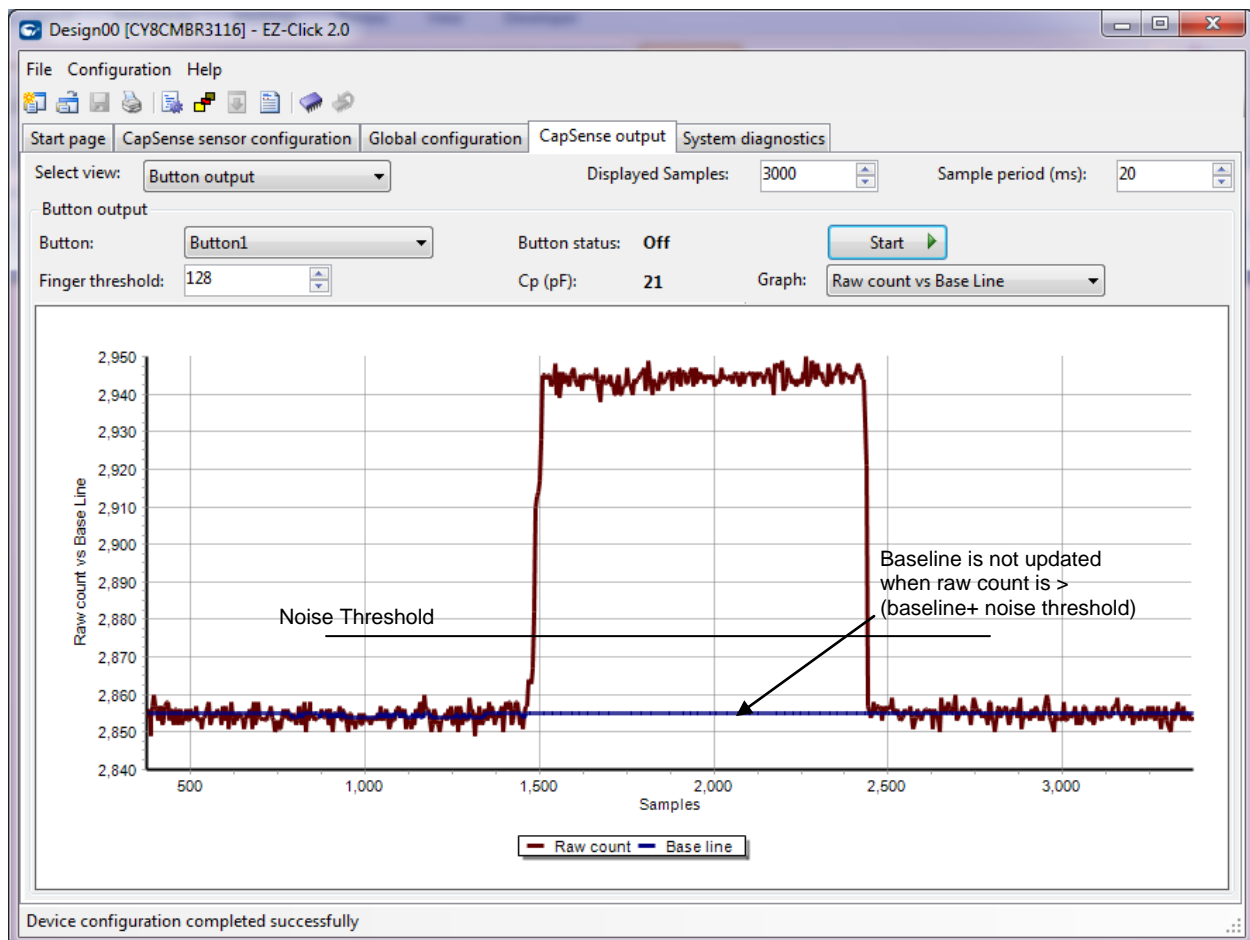
6.2.2.4 Negative Noise Threshold

If the raw count is less than (baseline – negative noise threshold) for the number of samples specified by the low-baseline reset parameter, the baseline is reset to the new raw count value. This change in baseline resets any sensor that is stuck in the active state during the device power ON i.e. sensor touched during controller power-up. This parameter can range from 0 to 127; default value is set to 51. See [Table 6-2](#) to set this parameter.

6.2.2.5 Low-Baseline Reset

This parameter is used along with the negative noise threshold parameter to reset the baseline when the baseline is higher than the raw count. It counts the number of abnormally low raw counts required to reset the baseline. It is used to reset the baseline if the finger is placed on the sensor during device startup and later removed. This parameter can range from 0 to 127; the default value is set to 50. See [Table 6-2](#) to set this parameter.

Figure 6-4. Noise Threshold



6.2.3 Setting the Thresholds for the Button, Slider, and Guard Sensors

Table 6-1 shows the threshold parameters that are applicable for button, slider, and guard sensor.

CY8CMBR3xxx provides the Automatic Threshold feature for buttons. This feature should always be enabled unless the user requires specifying the finger threshold manually.

Slider sensors require the user to specify the finger threshold parameter manually because the automatic threshold feature is not applicable to slider sensor. The SmartSense auto-tuning algorithm automatically sets all other threshold parameters.

To manually specify the threshold parameters such as noise threshold, negative noise threshold, and low-baseline reset, set the respective threshold override bit in the respective threshold register and specify these thresholds manually. It is recommended not to override the threshold parameters. To set the threshold override bit and specify the threshold parameters such as noise threshold, negative noise threshold, and low-baseline reset parameter, use Bridge Control Panel (BCP) tool. [EZ-click](#) does not provide option to override or specify threshold parameters.

Table 6-1. Threshold Parameters for Button, Slider, and Guard Sensors

Threshold Parameter	Sensor Type		
	Button	Slider	Guard
Finger Threshold	Set only if the Automatic Threshold feature is disabled	Should always be set manually	Set only if the Automatic Threshold feature is disabled
Noise Threshold	Set only if the Noise Threshold Override bit is enabled	Set only if the Slider Noise Threshold Override bit is enabled	Set only if the Noise Threshold Override bit is enabled
Negative Noise Threshold	Set only if the Negative Noise Threshold Override bit is enabled	Set only if the Slider Negative Noise Threshold Override bit is enabled	Set only if the Negative Noise Threshold Override bit is enabled
Hysteresis	Set only if the Hysteresis Threshold Override bit is enabled	Not Applicable	Set only if the Hysteresis Threshold Override bit is enabled
Low-Baseline Reset	Set only if the Low-Baseline Reset Override bit is enabled	Set only if the Slider Low-Baseline Reset Override bit is enabled	Set only if the Low-Baseline Reset Override bit is enabled

Note: CY8CMBR3xxx has separate registers for the button and slider sensors to set the noise threshold, negative noise threshold, and low-baseline reset parameters.

Follow the below procedure to set these threshold parameters:

1. In the CapSense output tab, select the “Diff count vs. Finger threshold” graph and click on the Start button.
2. Touch the sensor and measure the difference count, as [Figure 6-5](#) shows.

Notes:

- The difference count in CY8CMBR3xxx controller is normalized. It will not be equal to Raw counts (touched) - Baseline.
- If the difference count is greater than 255, it will be truncated to 255. In this case, threshold parameters cannot be set properly. Increase the sensitivity parameter value until the difference counts is less than 255. Ensure the SNR is greater than 5:1 after changing the sensitivity parameter value by following the procedure explained in section [6.2.1](#)

3. Set the threshold parameters to values as [Table 6-2](#) shows:

Table 6-2. Threshold Parameter Values

Threshold Parameter	Recommendation
Finger Threshold	80% of the Difference count
Noise Threshold	40% of the Difference count
Negative Noise Threshold	40% of the Difference count
Hysteresis	10% of the Difference count
Low-Baseline Reset	Set to 50

For the example shown in [Figure 6-5](#), the difference count is approximately 210 counts and the threshold parameters are set as shown below.

Finger threshold = $80\% \times 210 = 168$

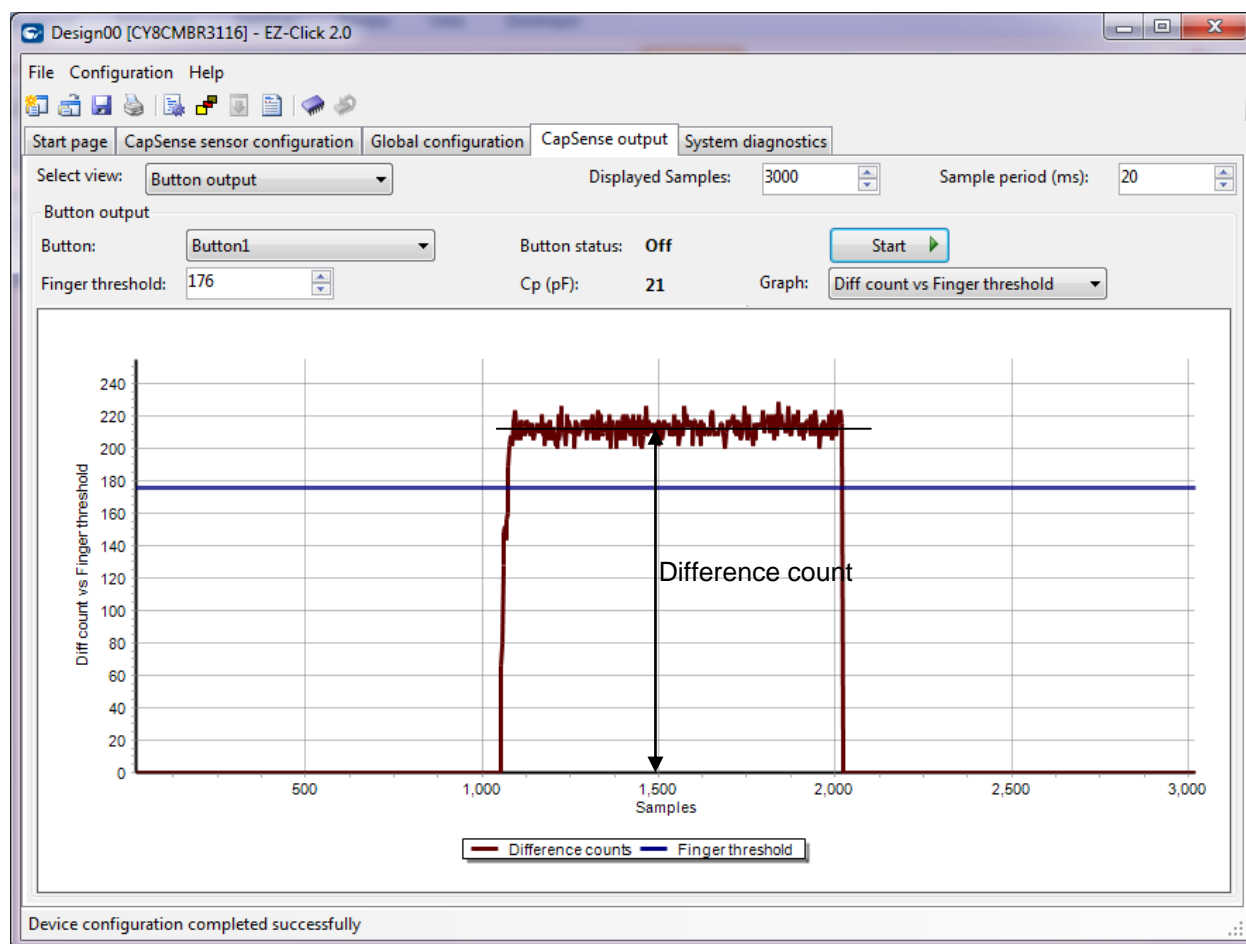
Noise threshold = $40\% \times 210 = 84$

Negative Noise threshold = Noise threshold = 84

Hysteresis = $10\% \times 210 = 21$

Low Baseline reset = 50

Figure 6-5. Difference Count Measurement



6.3 Tuning a Proximity Sensor

Figure 6-6 shows the typical flow of tuning a proximity sensor.

Tuning a proximity sensor in the CY8CMBR3xxx controller has two stages:

1. Ensure SNR is greater than 5:1
2. Set appropriate threshold parameters

6.3.1 Method to Ensure Minimum SNR

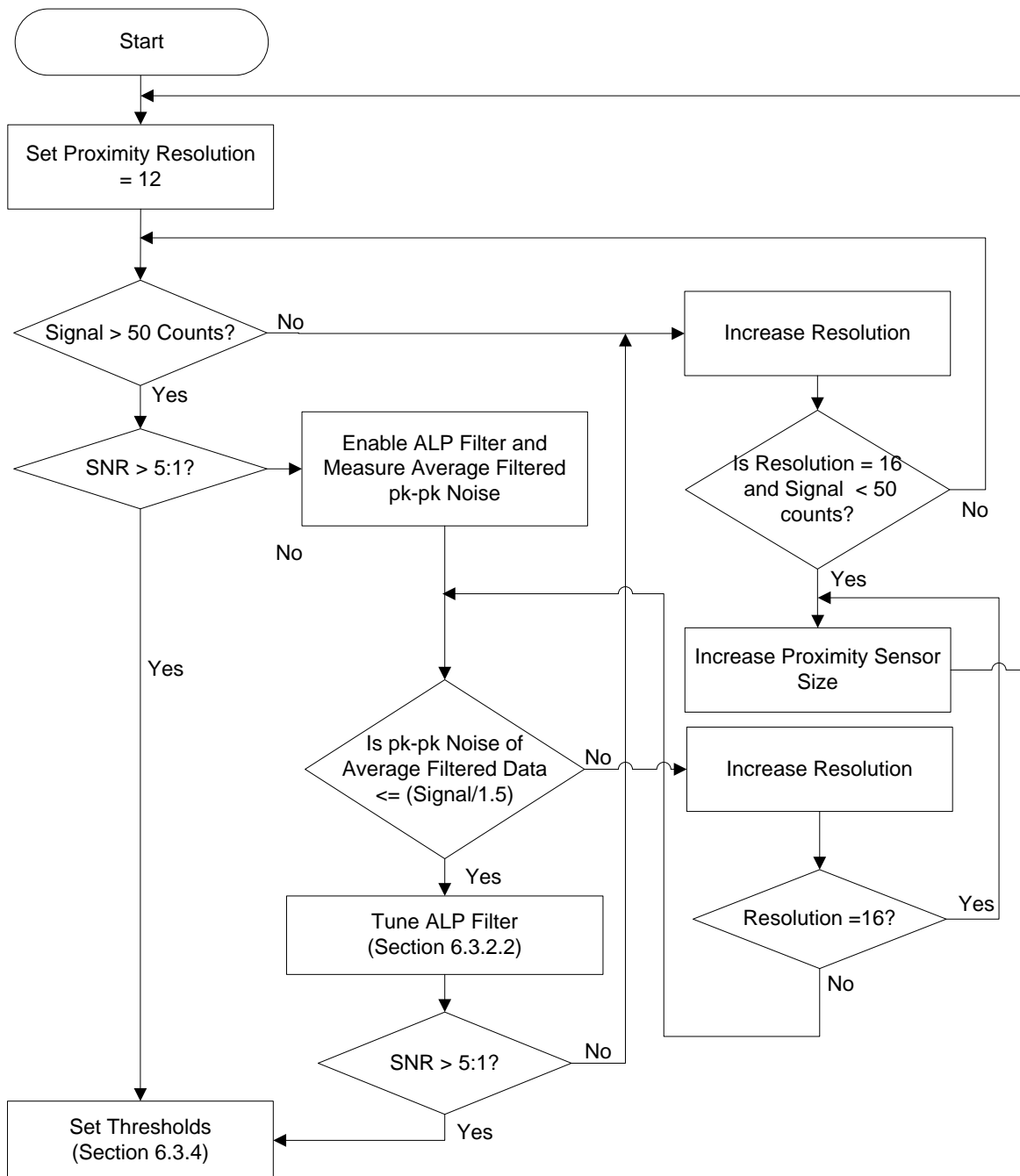
To ensure that the SNR is greater than 5:1, follow these steps:

1. Set the resolution of the proximity sensor to 12-bit and measure the signal at the required proximity distance.
2. If the signal is less than 50 counts, increase the proximity resolution until the signal is greater than 50 counts. If the signal is not greater than 50 counts even after setting the highest resolution (that is, 16-bit), increase the proximity sensor size.
3. After the signal is greater than 50 counts, measure the noise counts, as shown in Figure 6-7. If the noise is greater than one-fifth of the signal, enable the ALP filter to reduce the noise.

Note: For proximity sensors Median and IIR filters are not applicable.

4. Compute the SNR, and if the SNR is greater than 5:1, proceed to set the threshold parameters.
5. If SNR is not greater than 5:1 even after enabling the ALP filter, increase the resolution and compute the SNR. If the SNR is less than 5:1 even after setting the highest resolution (16-bit) and enabling the ALP filter, increase the proximity sensor size to achieve an SNR greater than 5:1.

Figure 6-6. Proximity Sensor Tuning



6.3.2 Advanced Low-Pass (ALP) Filter

The ALP filter is a combination of multiple low-pass filters specifically designed to attenuate the noise in proximity sensors. The ALP filter switches among multiple low-pass filters to achieve the maximum noise attenuation and provide optimum response time. Set the ALP parameters shown in [Table 6-3](#) for proper operation.

Table 6-3. ALP Filter Parameters

Parameter	Recommendation
K-value	Set when the ALP filter is enabled
Proximity Positive Threshold	Set when the ALP filter is enabled
Proximity Negative Threshold	Set when the ALP filter is enabled

6.3.2.1 ALP Filter Parameters

6.3.2.1.1 K-Value

The K-value of the ALP filter determines the attenuation of noise in proximity sensor raw counts. The CY8CMBR3xxx controller has three different K-values: 1/16, 1/32, and 1/64.

Noise attenuation decreases in the order $1/64 > 1/32 > 1/16$. Refer to the [ALP Filter Tuning](#) section to set this parameter.

6.3.2.1.2 Proximity Positive Threshold

This parameter determines the turn-on time of the proximity sensor, and is in the range 0-255; the default value is set to 30. Refer to the [ALP Filter Tuning](#) section to set this parameter.

6.3.2.1.3 Proximity Negative Threshold

This parameter determines the turn-off time of the proximity sensor, and is in the range 0-255; the default value is set to 30. Refer to the [ALP Filter Tuning](#) section to set this parameter.

6.3.2.2 ALP Filter Tuning

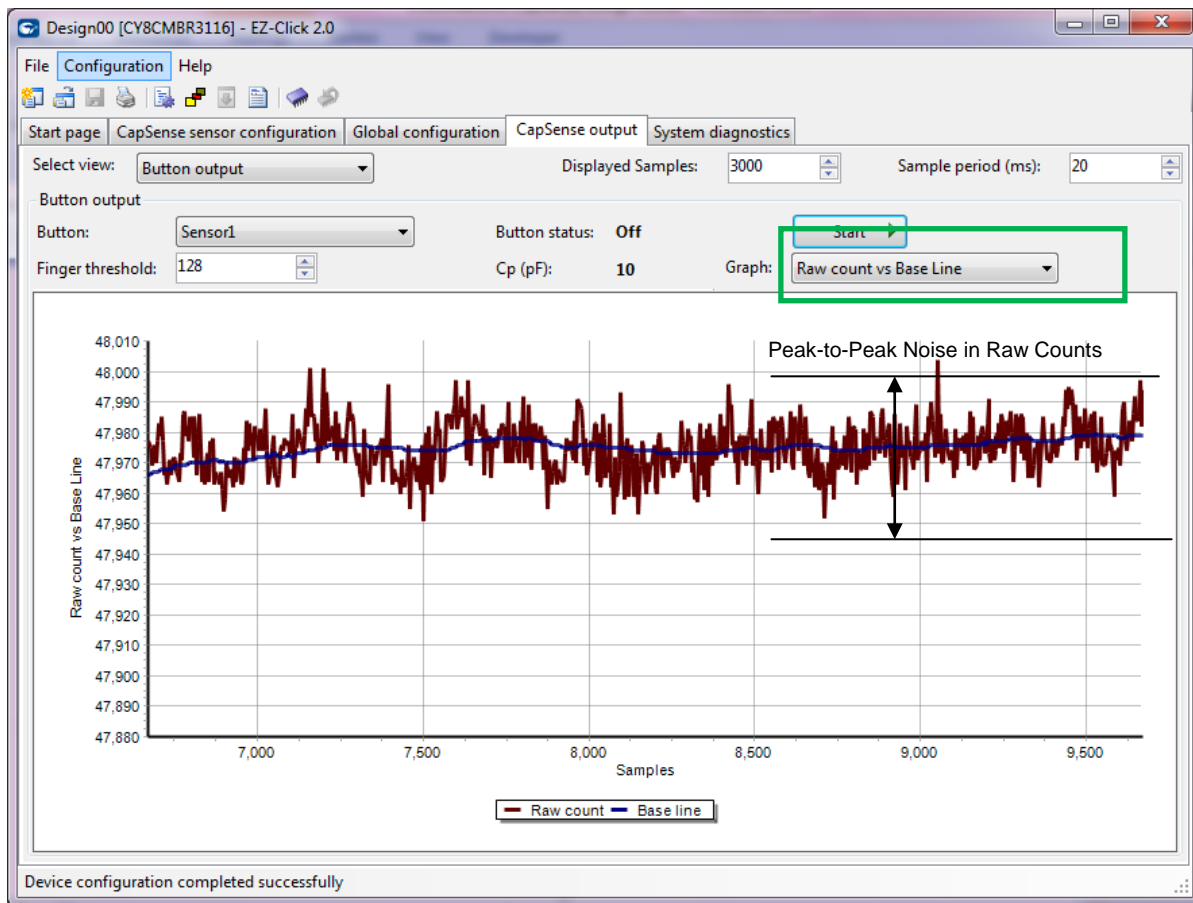
To tune the ALP filter, follow these steps:

1. Disable the ALP filter and measure the peak-to-peak noise in raw counts for 3000 samples, as [Figure 6-7](#) shows. See the [Noise Counts Measurement](#) section for more details.
2. Set the K-value in the “CapSense sensor configuration” tab of [EZ-Click](#) per the mapping in [Table 6-4](#).

Table 6-4. Selecting the K-Value

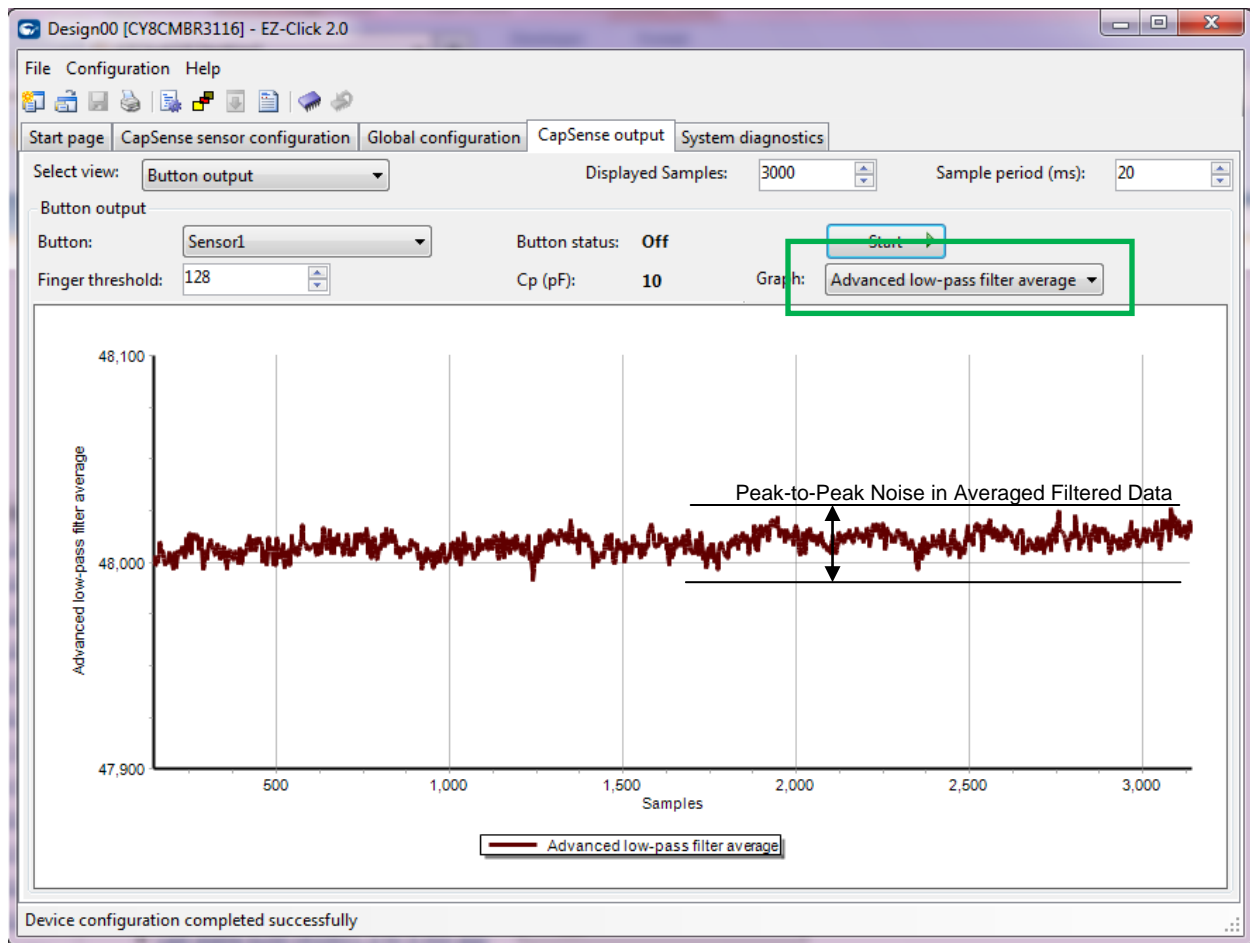
Peak-to-Peak Noise	Recommended K-Value
Less than 32 counts	1/16
Greater than 32 counts	1/32
Greater than 64 counts	1/64

Figure 6-7. Measuring Noise Counts of Proximity Sensor When ALP Filter is Disabled



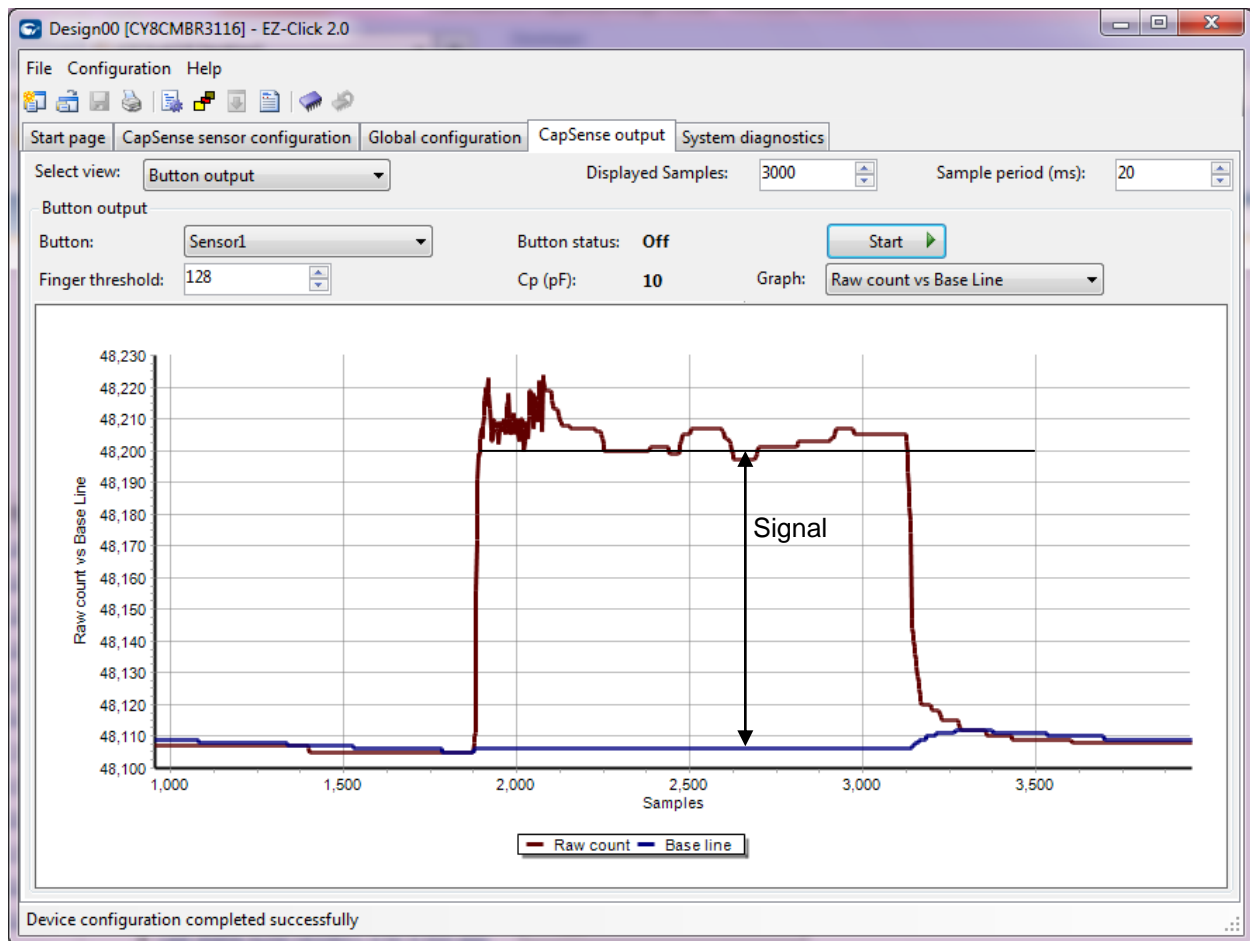
3. Enable the ALP filter (If K-Value is set, EZ-Click automatically enables ALP filter) and measure the peak-to-peak noise in the advanced low pass average filtered data for 3000 samples, as Figure 6-8 shows. The average filtered data can be viewed in the CapSense output tab in EZ-Click.

Figure 6-8. Peak-to-Peak Noise in Average Filtered Data



4. Set the proximity positive threshold as equal to $1.5 \times$ peak-to-peak noise of the average filtered data.
5. Set the proximity negative threshold as equal to $0.5 \times$ peak-to-peak noise of the average filtered data.
6. Set the finger threshold as equal to the proximity positive threshold. This is required to determine the response of the ALP filter.
7. Configure the device with these settings and measure the noise in raw counts.
8. Place the hand at the required proximity distance and measure the signal, as [Figure 6-9](#) shows.
9. Compute the SNR. If the SNR is greater than 5:1, then proceed to set the thresholds, as per [Table 6-5](#). Otherwise, increase the proximity resolution and repeat from Step 1.

Figure 6-9. Measuring Proximity Sensor Signal



6.3.3 Proximity Sensor Threshold Parameters

Table 6-5 lists the threshold parameters that are applicable to proximity sensors.

Table 6-5. Proximity Sensor Thresholds

Parameter	Recommendation
Proximity Threshold	Should always be set manually
Proximity Touch Threshold	Should always be set manually
Proximity Hysteresis	Set when hysteresis override is enabled
Proximity Noise Threshold	Set when Noise threshold override is enabled
Proximity Negative Noise Threshold	Set when Negative Noise threshold override is enabled
Proximity Low-Baseline Reset	Set when low baseline reset override is enabled

6.3.3.1 Proximity Threshold

The proximity threshold parameter is similar to the finger threshold of buttons. It determines the ON/OFF state of the proximity sensor. If the difference count is greater than proximity threshold + hysteresis, the proximity status is ON. If the difference count is less than proximity threshold - hysteresis, the proximity status is OFF. This threshold should be manually set. This parameter can range from 31 to 200; the default value is 128. See [Table 6-6](#) for the recommended value.

6.3.3.2 Proximity Touch Threshold

A proximity sensor has two events: proximity detect event and touch detect event (upon physically touching the proximity sensor). The proximity threshold parameter determines the ON/OFF status of the proximity event; the proximity touch threshold is used to detect the touch event of the proximity sensor. This parameter can range from 62 to 65000; the default value is 512. See [Table 6-6](#) for the recommended value.

6.3.3.3 Proximity Hysteresis

The proximity hysteresis parameter is similar to button hysteresis. It is used along with the proximity threshold parameter to determine the ON/OFF status of the proximity sensor. This parameter can range from 0 to 127; the default value is 5. See [Table 6-6](#) for the recommended value.

6.3.3.4 Proximity Noise Threshold

The proximity noise threshold parameter is similar to the [Noise Threshold](#) parameter for buttons. This parameter can range from 0 to 127; the default value is 20. See [Table 6-6](#) for the recommended value.

6.3.3.5 Proximity Negative Noise Threshold

The proximity negative noise threshold parameter is similar to the [Negative Noise Threshold](#) parameter for buttons. This parameter can range from 0 to 127; the default value is 20. See [Table 6-6](#) for the recommended value.

6.3.3.6 Proximity Low-Baseline Reset

The proximity low-baseline reset parameter is similar to the [Low-Baseline Reset](#) parameter for buttons. This parameter can range from 0 to 127; the default value is 50. See [Table 6-6](#) for the recommended value.

6.3.4 Setting Proximity Sensor Threshold Parameters

For a proximity sensor, only the proximity threshold and proximity touch threshold need to be specified. All other thresholds will be internally calculated by the SmartSense algorithm.

If, for some reason, these parameters need to be overridden, use the [Bridge Control Panel](#) to send I²C commands to override these parameters. Refer to the [CY8CMBR3xxx Registers TRM](#) for more details. It is strongly recommended not to override threshold parameters.

Follow the below procedure to set the threshold parameters for a proximity sensor:

1. In the CapSense output tab, select the Diff count vs. Finger threshold graph for proximity sensor and click on the Start button.
2. Place the hand at the required proximity distance and measure the difference count, as shown in [Figure 6-10](#).

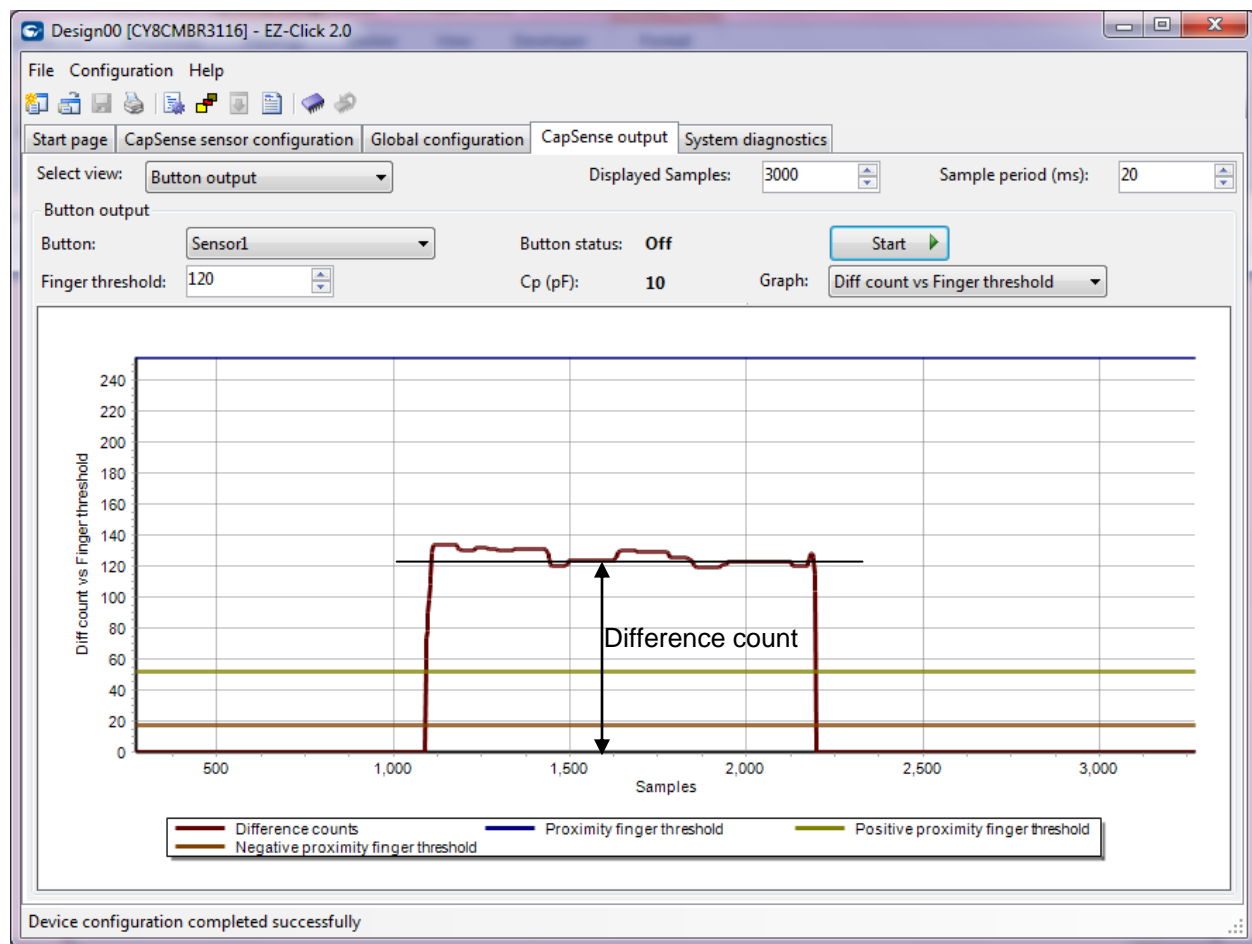
Notes:

- The difference count in CY8CMBR3xxx controller is normalized. It will not be equal to Raw counts (touched) - Baseline.
 - If the difference count is greater than 255, it will be truncated to 255. In this case, threshold parameters cannot be set properly. Increase the sensitivity parameter value until the difference counts is less than 255. Ensure the SNR is greater than 5:1 after changing the sensitivity parameter value.
3. Set the threshold parameters to values as shown in [Table 6-6](#).
 4. To set the proximity touch threshold, touch the proximity sensor and measure the difference counts (Proximity Sensor Touched).
 5. Set the proximity touch threshold to a value equal to 80% of difference counts (Proximity Sensor Touched)

Table 6-6. Proximity Threshold Settings

Threshold Parameter	Recommendation
Proximity Threshold	80% of Difference count
Proximity Touch Threshold	80% of Difference count (when proximity sensor is touched)
Proximity Noise Threshold	40% of Difference count
Proximity Negative Noise Threshold	40% of Difference count
Proximity Hysteresis	10% of Difference count
Proximity Low-Baseline Reset	Set to 50

Figure 6-10. Measuring Difference Count to Set Proximity Sensor Thresholds



7. Low-Power Design Considerations



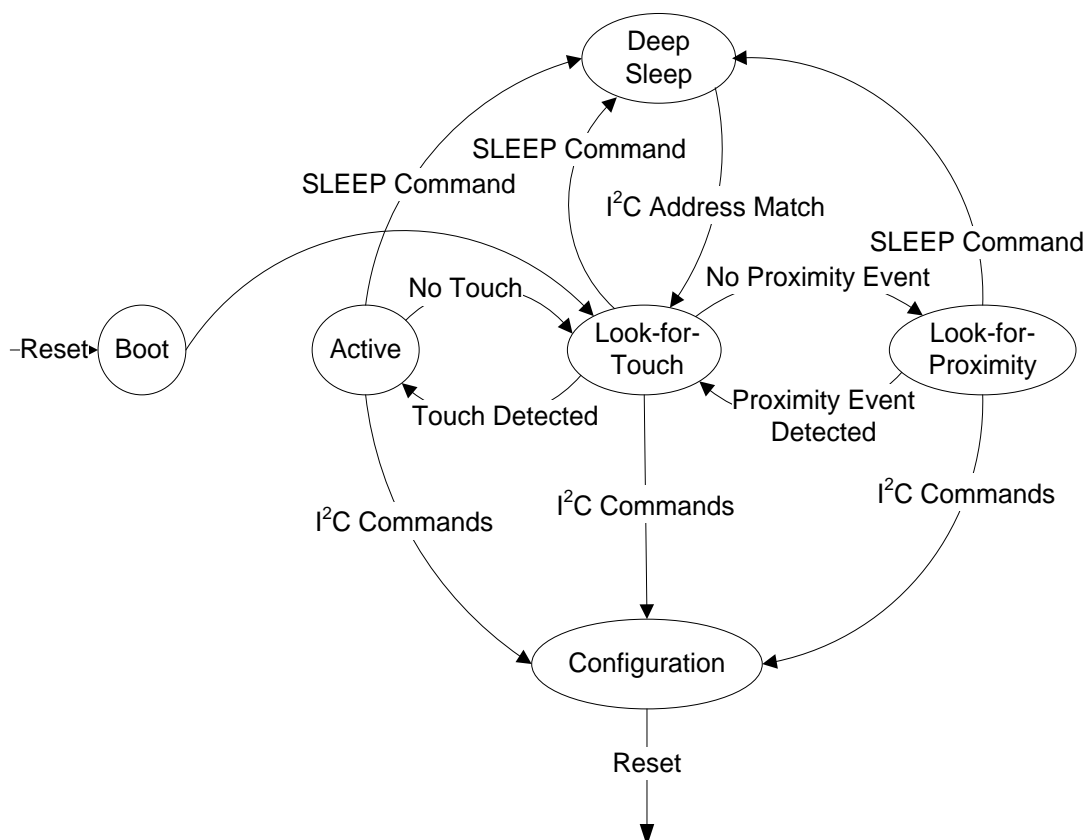
Capacitive touch sensing is used in battery-powered, handheld, and portable electronic devices. Because these devices are battery powered, power consumption and energy efficiency become very important. This section explains the factors that affect the power consumption of the CY8CMBR3xxx controller and provides guidelines on how to minimize the power consumption.

7.1 CY8CMBR3xxx Operating Modes

The CY8CMBR3xxx controller supports five operational modes:

1. Active Mode
2. Look-for-Touch Mode
3. Look-for-Proximity Mode
4. Deep Sleep Mode
5. Configuration Mode

Figure 7-1. CY8CMBR3xxx Operational Modes



7.1.1 Active Mode

The CY8CMBR3xxx controller enters the Active mode from the reset state, as [Figure 7-1](#) shows. In the Active mode, the controller scans all the sensors, updates the sensor data, and drives any configured outputs (GPOs, buzzer, and host interrupt) based on the sensor status. By default, the scan rate during the Active mode is set to 20 ms to ensure fast touch response. If the sensor scanning and processing is completed within 20 ms, then the controller is put to the sleep mode for the time duration equal to (20 ms – Active time). When the 20 ms timer expires, sensors are scanned again and the cycle continues, as [Figure 7-2](#) shows.

If the time taken to scan all the sensors and process the sensor data is more than 20 ms, then the device is not put in to the sleep mode. Instead, the next cycle of scanning will be started, as [Figure 7-3](#) shows.

Figure 7-2. Active Mode Scan-Sleep Sequence When Active Time < 20 ms

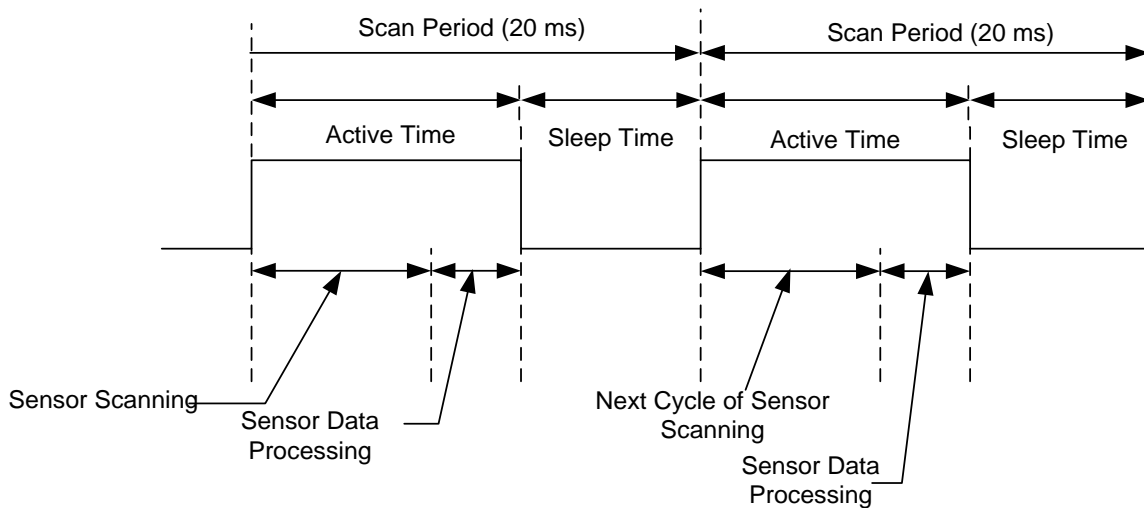
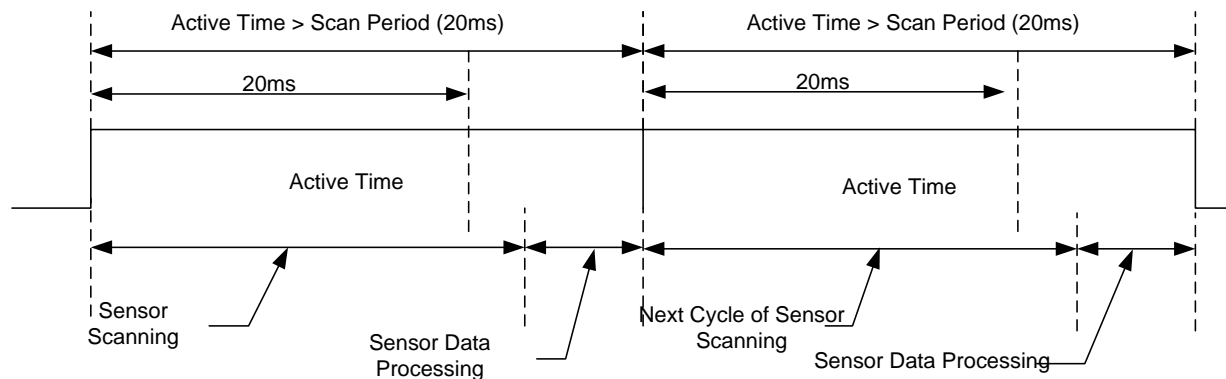


Figure 7-3. Active Mode Scan-Sleep Sequence When Active Time > 20 ms



7.1.2 Look-for-Touch Mode

If no touch is detected in the Active mode for duration equal to the State Timeout value, CY8CMBR3xxx enters the Look-for-Touch mode, as [Figure 7-1](#) shows. The Look-for-Touch mode is similar to the Active mode except that the scan period is configurable. The default value of scan period is 120 ms. In this mode, all the sensors are scanned and sensors data are processed. If any sensor is ON, outputs (GPOs, buzzer and host interrupt) are driven and the controller enters the Active mode.

7.1.3 Look-for-Proximity Mode

The Look-for-Proximity mode is applicable only if the design has a proximity sensor and Wake-on-Approach is enabled. If no touch is detected in the Look-for-Touch mode for a duration greater than or equal to the State Timeout value, then the controller enters the Look-for-Proximity mode, as [Figure 7-1](#) shows.

In the Look-for-Proximity mode, the controller scans only proximity sensors. If a proximity event is detected, then the controller enters the Look-for-Touch mode. If a touch event is detected in the Look-for-Proximity mode, then the controller enters the Active mode. The scan period in this mode is configurable and it is equal to the scan period in the Look-for-Touch mode; the default value is 120 ms.

7.1.4 Deep Sleep Mode

The Deep Sleep mode is the lowest power consumption mode. The CY8CMBR3xxx device enters deep sleep mode when the host issues a sleep command. Refer to the CTRL_CMD register in the [CY8CMBR3xxx Registers TRM](#) for more details. In the deep sleep mode, the device does not scan any sensor and all outputs are disabled. The device wakes up on the I²C address match event; NACKs the first I²C command, and then enters the Look-for-Touch mode.

7.1.5 Configuration Mode

The device enters the configuration mode when CMD_OP_CODE = '2' is written to CTRL_CMD register to save the configuration data to nonvolatile memory. Refer to [CY8CMBR3xxx Registers TRM](#) for more details. The device comes out of the configuration mode only through a device reset (either software or hardware reset).

7.2 Factors Affecting Power Consumption

Various factors that affect power consumption of the CY8CMBR3xxx controller are:

- **Number of Sensors:** The total scan time of sensors increases with the increase in the number of sensors. For a fixed scan period, if the scan time increases, then the duration in which the device is in sleep mode decreases. This results in increased power consumption of the device.
- **C_P of the Sensor:** This affects the scan time of the sensor. With a larger value of C_P, there will be higher sensor scan time, which results in increased power consumption. Follow the layout guidelines mentioned in the [Getting Started with CapSense](#) guide to minimize the sensor C_P.
- **Sensitivity:** This parameter affects the scan time of the sensor. The scan time of the sensor increases with a decrease in the sensor sensitivity parameter value. As a result, the lower the sensitivity parameter value, the higher will be the power consumption.

The power consumption decreases in the order 100 fF > 200 fF > 300 fF > 400 fF.

- **Electromagnetic Compatibility (EMC):** When the EMC feature is enabled, the scan time of the sensor increases and results in increased power consumption.
- **Buzzer Duration:** The CY8CMBR3xxx controller stays in the active state while driving the buzzer output. Longer buzzer duration results in increased active time and higher power consumption.
- **PWM on GPOs:** Enabling PWM on GPOs has the same effect as that of enabling the buzzer output. If PWM output is enabled on the GPO pins, the device is not put into sleep mode and this increases the power consumption.
- **Scan period:** The scan period parameter affects the duration for which the device is in sleep mode. The higher the scan period, the lower will be the power consumption.
- **State Timeout:** The state timeout parameter affects how quickly the device enters the look-for-touch mode from the active mode or the look-for-proximity mode from the look-for-touch mode. The shorter the state timeout interval, the lower will be the power consumption.
- **I²C Communication:** As explained in the Active Mode section, CY8CMBR3xxx enters sleep mode after the sensor scanning is completed. The CY8CMBR3xxx will not enter sleep mode or it will exit from deep sleep mode if there are any I²C transactions with the host processor. Therefore, frequent I²C transactions result in increased average power consumption.

To reduce the average power consumption, implement a dedicated host interrupt pin and initiate an I²C transaction only when the CY8CMBR3xxx sends a host interrupt pulse.

Use the [CY8CMBR3xxx Design Toolbox](#) to estimate the average current consumption of the controller based on the design inputs as explained in [Estimating the Average Current](#) section.

7.3 System Design Recommendations for Low Power Consumption

Cypress' CY8CMBR3xxx is designed to meet the low-power requirements of battery-powered applications. To minimize power consumption, follow these guidelines:

- Minimize the sensor C_P using the design guidelines mentioned in [Getting Started with CapSense](#) and [CY8CMBR3xxx Design Toolbox](#).
- Set optimum value of sensitivity for sensors. Refer to the [Sensitivity Control](#) section.
- Enable EMC only if required. Refer to the [Electromagnetic Compatibility \(EMC\)](#) section.
- Increase the scan period and decrease the state timeout period.
- Put the device in the Deep Sleep mode or turn off the power to the device using the power manager IC (PMIC) when sensor scanning is not required.
- Avoid frequent I²C reads/writes to the CY8CMBR3xxx controller. Use host interrupt pin and initiate I²C transaction only after the CY8CMBR3xxx sends a host interrupt pulse.
- Use the proximity sensor and enable the wake-on-approach feature.
- Reduce the Buzzer ON time duration.
- Enable PWM on GPOs only if required.

8. Resources



8.1 Website

Visit the [Cypress' CapSense Controllers](#) website to access the reference material discussed in this document. You can find a variety of technical resources on the [CY8CMBR3xxx](#) web page.

8.2 Datasheet

The datasheet for the CapSense CY8CMBR3xxx device is available at [CY8CMBR3xxx Datasheet](#)

8.3 Registers Technical Reference Manual

The registers technical reference manual is available at [CY8CMBR3xxx Registers TRM](#)

8.4 Design Toolbox

The interactive [Design Toolbox](#) enables you to design a robust and reliable CY8CMBR3xxx CapSense solution.

8.5 EZ-Click™ 2.0

[EZ-Click 2.0](#) is a simple, GUI-based software utility that can be used to customize CY8CMBR3xxx device configurations.

8.6 Development Kit

The [CY3280-MBR3 Evaluation Kit](#) is designed to showcase the abilities of the CY8CMBR3116 CapSense Controller. Use this kit to quickly evaluate the features of CY8CMBR3xxx solution.

8.7 Design Support

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

[Knowledge-Based Articles](#) – Browse technical articles by product family or search on CapSense topics.

[CapSense Application Notes](#) – Peruse 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 CapSense product line.

[Video Library](#) –Get up to speed quickly with tutorial videos

[Quality & Reliability](#) – Cypress is committed to customer satisfaction. At our Quality website, find reliability and product qualification reports.

[Technical Support](#) – World-class technical support is available online.

9. Appendix



9.1 High-Level APIs

Table 9-1. High-Level Host APIs

1	CY8CMBR3xxx_ReadSensorStatus	
	Description	<p>This API reads the button status, latched button status, proximity status, latched proximity status, slider 1 and 2 positions, and slider 1 and 2 liftoff positions from the CY8CMBR3xxx device.</p> <p>The slider positions and slider liftoff positions are valid for CY8CMBR3106S device only.</p> <p>You can use this API when the device asserts a Host Interrupt pulse, to understand the corresponding status change.</p>
	Prototype	<code>bool CY8CMBR3xxx_ReadSensorStatus(uint8 slaveAddress, CY8CMBR3XXX_SENSORSTATUS *status)</code>
	Parameters	<p>slaveAddress</p> <p>The I²C address of the CY8CMBR3xxx device.</p> <p>Valid range: 8 – 119</p>
		<p>*status</p> <p>Pointer to the user-defined buffer where the sensor status read from the CY8CMBR3xxx device is stored. The buffer structure is defined in CY8CMBR3xxx_APIs.h.</p>
	Return value	<p>TRUE – If the buffer was updated properly</p> <p>FALSE – If the buffer was not updated</p>
2	CY8CMBR3xxx_WriteData	
	Description	<p>This API writes one or more bytes of data to successive register locations in the CY8CMBR3xxx device. The initial register location is user specified.</p>
	Parameters	<p>SlaveAddress</p> <p>The I²C address of the CY8CMBR3xxx device.</p> <p>Valid range: 8 – 119</p>

		<p>*writeBuffer</p> <p>The buffer from which data is written to the device.</p> <p>The first element should always contain the location of the register of the device to write to. This value can be within 0 – 134.</p> <p>Each successive element should contain the data to be written to that register and the successive registers. These elements can have a value between 0 – 255. The number of data bytes can be between 1 to minimum of (128, and 135 – register location).</p>
		<p>numberOfBytes</p> <p>Number of bytes to be written, equal to the number of elements in the buffer.</p> <p>Valid range: 1 – min(129, 136 – register location)</p>
	Return value	<p>TRUE – If the write was successful</p> <p>FALSE – If the write was not successful</p>
3	CY8CMBR3xxx_ReadData	
	Description	This API reads one or more bytes of data from successive register locations in the CY8CMBR3xxx device. The initial register location is user specified.
	Prototype	bool CY8CMBR3xxx_ReadData(uint8 slaveAddress, uint8 registerAddress, uint8 *readBuffer, uint8 numberOfBytes)
	Parameters	<p>slaveAddress</p> <p>The I²C address of the CY8CMBR3xxx device.</p> <p>Valid range: 8 – 119</p>
		<p>registerAddress</p> <p>The register location to read from. The value passed to this argument should be one of the register offset macros defined in the file CY8CMBR3xxx_Registers.h. The macro for a register is defined only if it is supported by the device defined by CY8CMBR3xxx_DEVICE macro.</p>
		<p>*readBuffer</p> <p>The buffer to be updated with the data read from the device.</p> <p>Each successive element to contain the data read from successive registers. These elements can have a value between 0 – 255.</p>
		<p>numberOfBytes</p> <p>Number of bytes to be read, equal to the number of elements in the buffer.</p> <p>Valid range: 1 – 252</p>
	Return value	<p>TRUE – If the read was successful</p> <p>FALSE – If the read was not successful</p>
4	CY8CMBR3xxx_WriteDualByte	
	Description	This API writes to a two byte register in the CY8CMBR3xxx device. The register location is user – specified.

	Prototype	bool CY8CMBR3xxx_WriteDualByte(uint8 slaveAddress, uint8 registerAddress, uint16 writeData)
	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
		registerAddress The register location to write to. The value passed to this argument should be one of the register offset macros defined in the file CY8CMBR3xxx_Registers.h. The macro for a register is defined only if it is supported by the device defined by CY8CMBR3xxx_DEVICE macro.
		writeData The two-byte value to be written to the register. The LSB forms the first byte and the MSB forms the second byte. Valid range: 0 – 65535
	Return value	TRUE – If the write was successful FALSE – If the write was not successful
5	CY8CMBR3xxx_ReadDualByte	
	Description	This API reads from a two-byte register in the CY8CMBR3xxx device and updates the concatenated value to the read buffer. The register location is user – specified.
	Prototype	bool CY8CMBR3xxx_ReadDualByte(uint8 slaveAddress, uint8 registerAddress, uint16 *readData)
	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
		registerAddress The two-byte register to read from. The data will be read from this address and the successive address. The value passed to this argument should be one of the register offset macros defined in the file CY8CMBR3xxx_Registers.h. The macro for a register is defined only if it is supported by the device defined by CY8CMBR3xxx_DEVICE macro.
		*readData The 2-byte buffer to be updated with the value read from the device. The API fills the LSB with the first byte read from the device and the MSB with the second byte. The value can be between 0 – 65535.
	Return value	TRUE – If the read was successful FALSE – If the read was not successful

6	CY8CMBR3xxx_SendCommand	
	Description	This API issues a new command to the CY8CMBR3xxx device. Before calling this API, you should check whether the device is ready to accept a new command by calling CY8CMBR3xxx_CheckCommandStatus().
	Prototype	bool CY8CMBR3xxx_SendCommand(uint8 slaveAddress, uint8 command)
	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 - 119
		command The command op-code to issue. The value passed to this argument should be one of the op-code macros defined in the file CY8CMBR3xxx_CommandsAndConfig.h.
	Return value	TRUE – If the command was issued successfully FALSE – If the command was not issued successfully
7	CY8CMBR3xxx_CheckCommandStatus	
	Description	This API verifies whether the CY8CMBR3xxx device is ready to accept a new command, and whether the previously issued command has executed successfully.
	Prototype	uint8 CY8CMBR3xxx_CheckCommandStatus(uint8 slaveAddress, uint8 *errorCode)
	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 - 119
		*errorCode Pointer to the error code returned from the last command execution. If the last command failed, the API updates this data to show what caused the failure. Valid range is 0 to 255.
	Return value	0 – If the last command is still executing 1 – If the last command executed successfully 2 – If the last command did not execute successfully, or if the I ² C communication failed.
8	CY8CMBR3xxx_Configure	
	Description	This API writes a user – specified configuration to the CY8CMBR3xxx device, saves the configuration, and resets the device. To configure the device, you should first create a configuration using EZ-Click. The 128-byte configuration data (generated in the project specific .h file in the EZ-Click project) should be used directly for this API. This API blocks CPU execution during the configuration save command execution, and during CY8CMBR3xxx device reset.
	Prototype	bool CY8CMBR3xxx_Configure(uint8 slaveAddress, const unsigned char *configuration)

	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
		*configuration Buffer of 128 bytes to store the entire configuration for the device. This data should be taken directly from the EZ-Click generated configuration data. This includes 126-bytes of configuration registers data and two bytes of CRC.
	Return value	TRUE – If the device was configured successfully FALSE – If the device was not configured successfully
9	CY8CMBR3xxx_CalculateCrc	
	Description	This API calculates the CRC checksum of a given configuration setting for the CY8CMBR3xxx device.
	Prototype	uint16 CY8CMBR3xxx_CalculateCrc(uint8 *configuration)
	Parameters	*configuration This 126-byte buffer would hold the entire configuration setting for the device.
	Return value	A two-byte value (with the higher byte being the MSB) for the calculated CRC.
10	CY8CMBR3xxx_VerifyDeviceOnBus	
	Description	This API verifies that the device on the I ² C bus is a valid CY8CMBR3xxx device. It checks the Device ID, Family ID, and Device Revision to verify the device.
	Prototype	bool CY8CMBR3xxx_VerifyDeviceOnBus(uint8 slaveAddress)
	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
	Return value	TRUE – if the device is valid FALSE – if the device is not valid
11	CY8CMBR3xxx_SetDebugDataSensorId	
	Description	This API sets the Sensor ID for reading the corresponding debug data from the CY8CMBR3xxx device. To read the debug data from the device, you should first call this API, followed by a time wait equal to the device refresh interval. Then you should call CY8CMBR3xxx_ReadSensorDebugData() to read back the debug data from the device.
	Prototype	bool CY8CMBR3xxx_SetDebugDataSensorId(uint8 slaveAddress, uint8 sensorId)

	Parameters	SlaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
		sensorId The sensor number for which the data is requested. This number can be a maximum of one less than the total number of sensors supported by the device.
	Return value	TRUE – If the sensor ID write operation was successful FALSE – If the sensor ID write operation was not successful
12	CY8CMBR3xxx_ReadSensorDebugData	
	Description	<p>This API reads the Cp, Difference counts, Baseline, Raw counts, and Average Raw counts of a sensor and updates the data to a buffer.</p> <p>To read debug data, you should first call CY8CMBR3xxx_SetDebugDataSensorId(), and then wait for one refresh interval. After this time, you should call this API to read the data from the device.</p> <p>The API updates the buffer only if the sync counters of the device match. In case they don't, the API tries to read the data again. The maximum number of read attempts is defined by CY8CMBR3xxx_SYNC_COUNTER_MATCH_RETRY. You can set this value in CY8CMBR3xxx_APIs.h file.</p>
	Prototype	bool CY8CMBR3xxx_ReadSensorDebugData(uint8 slaveAddress, CY8CMBR3XXX_SENSORDATA *debugData)
	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
		*debugData Pointer to the user-defined buffer where the data read from the CY8CMBR3xxx device is stored. The buffer structure is defined in CY8CMBR3xxx_APIs.h.
	Return value	TRUE – If the buffer was updated properly FALSE – If the buffer was not updated
13	CY8CMBR3xxx_ReadDiffCounts	
	Description	<p>This API reads the Difference counts of all sensors of the CY8CMBR3xxx device and updates the data to a buffer.</p> <p>The API updates the buffer only if the sync counters of the device match. In case they don't, the API tries to read the data again. The maximum number of read attempts is defined by CY8CMBR3xxx_SYNC_COUNTER_MATCH_RETRY. You can set this value in CY8CMBR3xxx_APIs.h file.</p>
	Prototype	bool CY8CMBR3xxx_ReadDiffCounts(uint8 slaveAddress, uint16 *differenceCounts)

	Parameters	slaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
		*differenceCounts Buffer to store the difference counts of all sensors. The buffer size should be equal to the number of sensors in the device.
	Return value	TRUE – If the buffer was updated properly FALSE – If the buffer was not updated

9.2 Low-Level APIs

Table 9-2. Low-Level Host APIs

1	Host_LowLevelWrite	
	Description	This API writes to the register map of the CY8CMBR3xxx device using the I ² C communication protocol. The implementation is host processor dependent and requires modification to the API code to suit specific host processor. In case the device NACKs I ² C transaction on the first time, ensure that this API retries twice within the next 340 ms.
	Prototype	bool Host_LowLevelWrite(uint8 slaveAddress, uint8 *writeBuffer, uint8 numberOfBytes)
	Parameters	SlaveAddress The I ² C address of the CY8CMBR3xxx device. Valid range: 8 – 119
		*writeBuffer The buffer from which data is written to the device. The first element should always contain the location of the register of the device to write to. This value can be within 0 – 251. Each successive element should contain the data to be written to that register and the successive registers. These elements can have a value between 0 – 255. The number of data bytes can be between 0 and 128. Writing zero data bytes signifies that you are updating the register location from which you want to read device register map information.
		numberOfBytes Number of bytes to be written, equal to the number of elements in the buffer (i.e. number of data bytes + 1 for register location)
	Return value	TRUE – If the write process was successful FALSE – If the write process was not successful
2	Host_LowLevelRead	

	Description	<p>This API reads from the register map of the CY8CMBR3xxx device using the I²C communication protocol. The implementation is host processor dependent and requires modification to the API code to suit specific host processor.</p> <p>In case the device NACKs I2C transaction on the first time, you must ensure that this API retries twice within the next 340 ms.</p>
	Prototype	<pre>bool Host_LowLevelRead(uint8 slaveAddress, uint8 *readBuffer, uint8 numberOfBytes)</pre>
	Parameters	<p>slaveAddress</p> <p>The I²C address of the CY8CMBR3xxx device.</p> <p>Valid range: 8 – 119</p>
		<p>*readBuffer</p> <p>The buffer to be updated with the data read from the device.</p> <p>The register location to read from should be set using Host_LowLevelWrite API prior to calling this API. Each successive element to contain the data read from that register and the successive registers. These elements can have a value between 0 – 255.</p>
		<p>numberOfBytes</p> <p>Number of data bytes to be read, equal to the buffer size.</p> <p>Valid range: 1 – 252</p>
	Return value	<p>TRUE – If the read process was successful</p> <p>FALSE – If the read process was not successful</p>
3	Host_LowLevelDelay	
	Description	<p>This API implements a time-delay function to be used by the High-level APIs. The delay period is in milliseconds. This delay is achieved by a code-execution block for the required amount of time.</p> <p>The implementation is host processor dependent and requires modification to the API code to suit your host.</p>
	Prototype	<pre>void Host_LowLevelDelay(uint16 milliseconds)</pre>
	Parameters	<p>milliseconds</p> <p>The amount of time in milliseconds for which a wait is required.</p> <p>Valid range: 0 – 65535</p>
	Return value	None

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

CapSense Mechanical Button Replacement (MBR)

Cypress' 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_{REF}

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

Widget

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

Revision History



Document Revision History

Document Title: AN90071 - CY8CMBR3xxx CapSense® Design Guide

Document Number: 001-90071

Revision	Issue Date	Origin of Change	Description of Change
**	02/18/2014	DCHE	New Design Guide
*A	02/25/2014	DCHE/UDYG	Modified Figure 1-2 and Figure 2-13 Added Min and Max Cp range for each sensors in Table 4-1 Updated section 5.2 with Host APIs and Bridge Control Panel Added Appendix chapter
*B	03/11/2014	DCHE	Updated the Operating Voltage Specifications
*C	06/05/2014	DCHE	Added footnote#6 in page 27 Updated MBR3 Design Toolbox with Device Timing Parameters Table Updated “ Estimating the Response Time ” section with Device Timing Parameters.
*D	07/31/2014	DCHE	Updated Table 3-2 and Figure 5-21
*E	12/12/2014	DCHE	Updated Table 1-1 , Table 4-1 Updated Shield Electrode Construction section. Updated Figure 2-13 and Figure 2-14 Added section CapSense Design on FR4 and FPC Added information on CY8CMBR3002 in Configurations in CY8CMBR3xxx Added section Hardware Connection Updated Slider Layout Guidelines in MBR3 Design Toolbox Updated CY8CMBR3xxx_Host_APIS_Demo_Project folder structure to fix build errors.
*F	01/21/2016	VAIR	Added Glossary. Changed category from “Guides – Tech Pubs – English (EN)” to “Application Notes – English (EN)”.
*G	02/14/2017	DCHE	Updated Liquid Tolerance section.
*H	06/29/2017	AESATMP8	Updated logo and Copyright.