AN76000 - CY8CMBR2110

CapSense® Design Guide

Doc. No. 001-76000 Rev. *G

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

Contents

1. Introduction

1.1 Abstract

This document describes how to implement capacitive sensing functionality using Cypress's CapSense® Express CY8CMBR2110 device. The following topics are covered in this guide:

- Features of the CY8CMBR2110
- CapSense principles of operation
- Configuration options of the CY8CMBR2110 device
- Using the Design Toolbox with the CY8CMBR2110
- System electrical and mechanical design considerations for the CY8CMBR2110
- Low-power design considerations for the CY8CMBR2110
- Additional resources and support for designing CapSense into your system

1.2 Cypress's CapSense Documentation Ecosystem

Figure 1-1 and Table 1-1 summarize the CapSense documentation ecosystem. These resources allow the implementers to quickly access the information they need to complete a CapSense product design. Figure 1-1 shows a typical product design cycle with capacitive sensing; this document covers the topics highlighted in green. Table 1-1 offers links to supporting documents for each of the numbered tasks in Figure 1-1.

Figure 1-1. Typical CapSense Product Design Flow

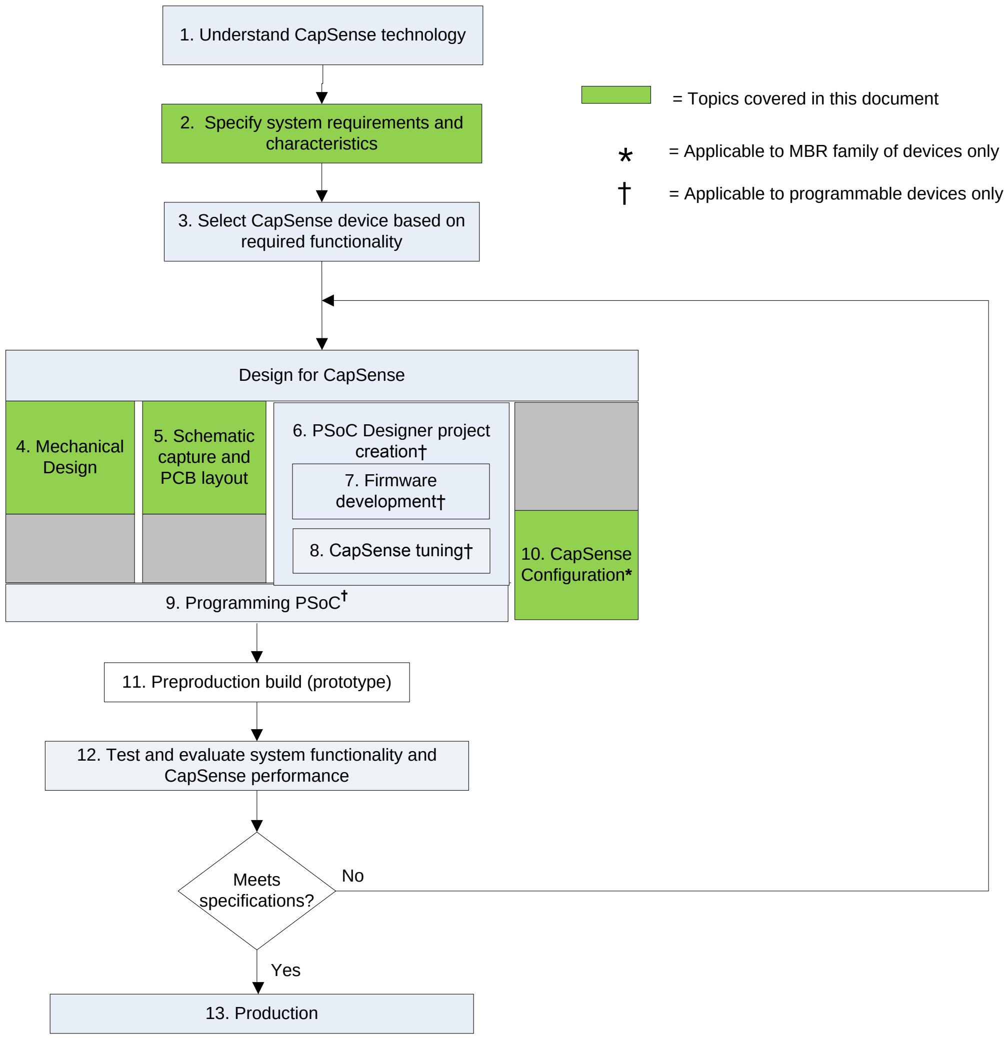


Table 1-1. Cypress Documents That Support Numbered Design Tasks of Figure 1-1

Numbered Design Task of Figure 1-1	Supporting Cypress CapSense Documentation
1	Getting Started with CapSense
2	CY8CMBR2110 Device Datasheet
3	Getting Started with CapSense
4	This document
5	This document
6	Not applicable for CY8CMBR2110
7	Not applicable for CY8CMBR2110
8	Not applicable for CY8CMBR2110
9	Not applicable for CY8CMBR2110
10	This document

1.3 CY8CMBR2110 CapSense Express Device Features

Cypress's low-power CapSense controller can easily add capacitive touch sensing to your user interface. The device's features include:

- Register configurable CapSense Controller
  - Does not require firmware or device programming
  - Ten button solution configurable through I^2C protocol
  - Ten general purpose outputs (GPOs)
  - GPOs are linked to CapSense buttons
  - GPOs support direct LED drive
- SmartSense™ Auto-Tuning
  - CapSense algorithm that continuously compensates for system, manufacturing, and environmental changes
  - Dynamically sets CapSense parameters
  - Eliminates the need for manual system tuning
  - Wide parasitic capcitiance(C_P) range (5-40 pF)
- Advanced features
  - Flanking Sensor Suppression (FSS)
    - Distinguishes between signals from closely spaced buttons
  - User-configurable LED effects
    - On system power-on
    - On button touch
    - LED ON Time after button release
    - Standby mode LED Brightness
  - Buzzer signal output
  - Analog voltage output
    - Using external resistor bridge
  - Attention line interrupt to host to indicate any CapSense button status change
  - CapSense performance data through I^2C interface
    - Simplifies production line testing and system debug
- Noise immunity
  - Specifically designed for superior noise immunity to external radiated and conducted noise
  - Low radiated noise emission
- System diagnostics
  - Button shorts
  - Improper value of modulating capacitor (C_{MOD})
  - Parasitic capacitance (C_P) value out of range
- EZ-Click™ Customizer Tool
  - Simple graphical configuration
  - Dynamically configures all features
  - Configurations can be saved and reused later

- I^{2}C interface
  - No clock stretching
  - Supports up to 100-kHz speed
- Wide operating voltage range
  - 1.71—5.5 V
  - Ideal for both regulated and unregulated battery applications
- Low power consumption
  - Average current consumption of 23 µA[1] per button
  - Deep sleep current: 100 nA
- Industrial temperature range: –40 °C to +85 °C
- 32-pin QFN package (5 mm x 5 mm x 0.6 mm)

1.4 Document Conventions

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

1.5 Acronyms

Acronym	Description
AC	Alternating current
ARST	Auto Reset
C_F	Finger capacitance
C_P	Parasitic capacitance
CS	CapSense
CSD	CapSense Sigma Delta

[1] Four buttons used, 180 button touches per hour, average button touch time = 1000 ms, buzzer disabled, Button Touch LED Effects disabled, 10 pF < (C_P of all buttons) < 20 pF, Button Scan Rate = 541 ms, power consumption optimized, Noise Immunity level "Normal", CSx sensitivity "Medium".

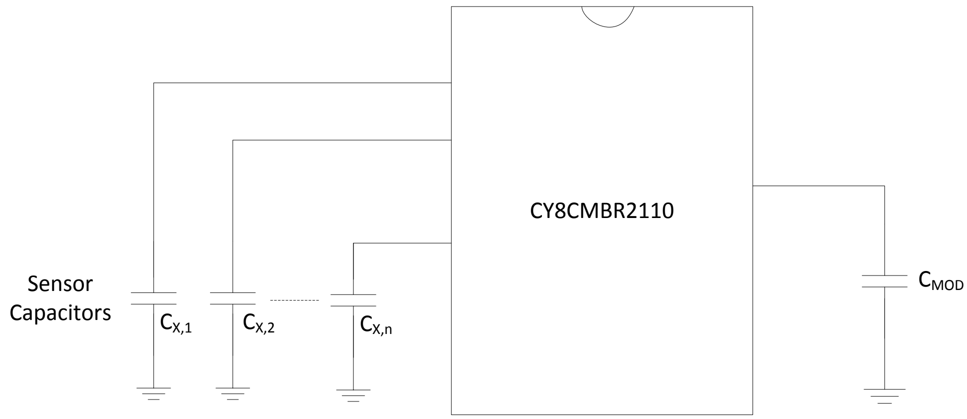
Acronym	Description
EMC	Electromagnetic Compatibility
ESD	Electrostatic Discharge
FSS	Flanking Sensor Suppression
GPO	General-Purpose Output
MSB	Most significant bit
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
LSB	Least significant bit
PCB	Printed Circuit Board
POR	Power on Reset
POST	Power on Self-Test
RF	Radio Frequency
SNR	Signal to Noise Ratio
SMPS	Switched Mode Power Supply

2. CapSense Technology

2.1 CapSense Fundamentals

CapSense is a touch sensing technology that works by measuring the capacitance of each sensor input pin on the CapSense controller. The total capacitance on each of the sensor pins can be modeled as equivalent lumped capacitors with values of CX,1 through CX,n as shown in Figure 2-1. Circuitry internal to the CY8CMBR2110 device converts the magnitude of each CX into a digital code that is stored for post-processing. A modulating capacitor, CMOD, is used by the CapSense controller's internal circuitry. CMOD will be discussed in more detail in Capacitive Sensing Method.

Figure 2-1. CapSense Implementation in a CY8CMBR2110 Device



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

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

Where:

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

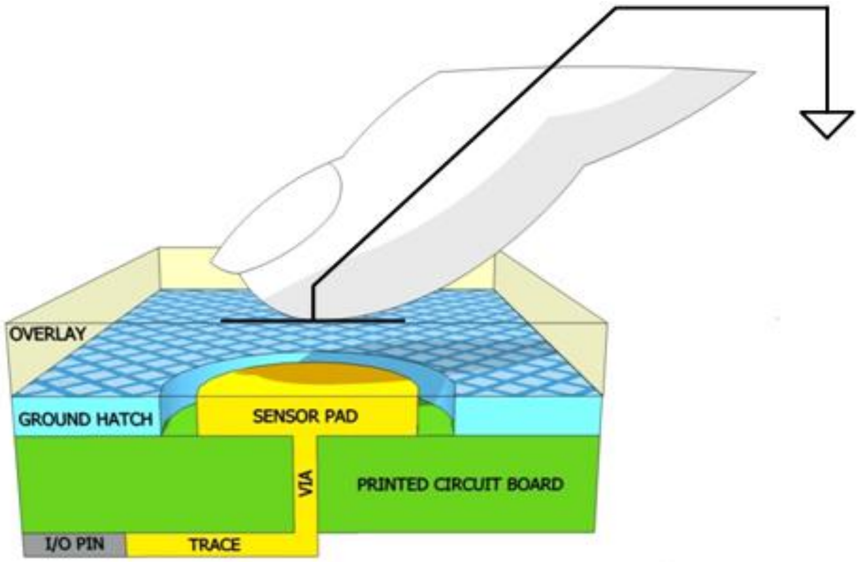
ε_0 = Free space permittivity

ε_r = Dielectric constant (relative permittivity) of overlay

A = Area of finger and sensor pad overlap

D = Overlay thickness

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



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

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

When a finger is not touching a sensor, use this equation:

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

With a finger on the sensor, C_X equals the sum of C_P and C_F:

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

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 40 pF. C_F usually ranges from 0.1—0.4 pF.

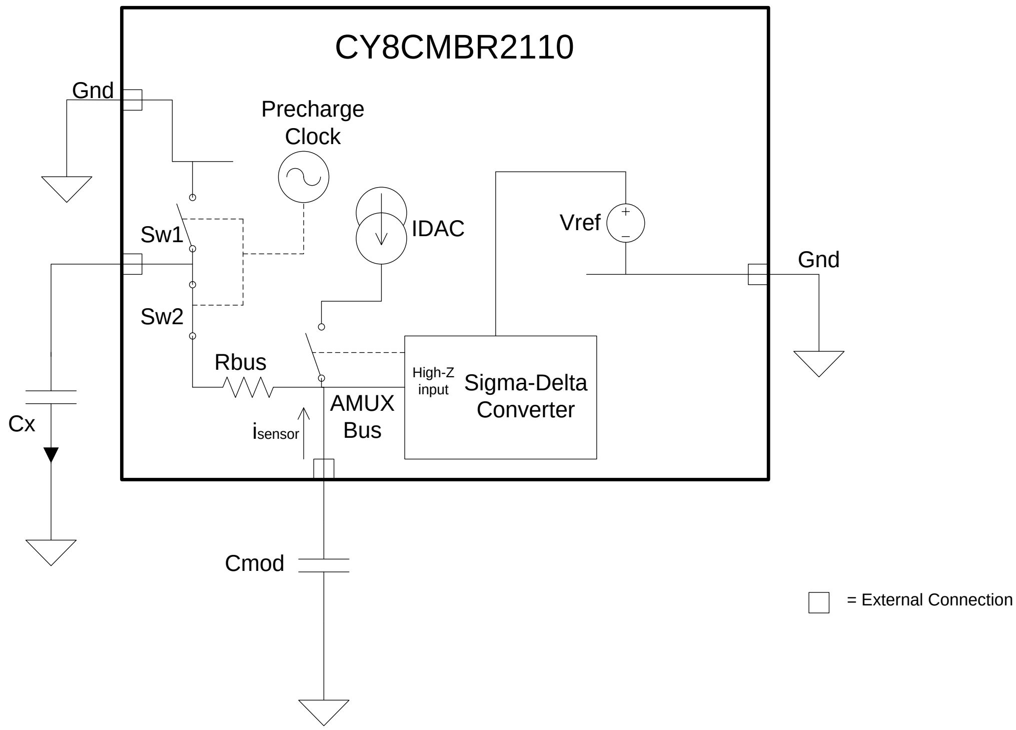
2.2 Capacitive Sensing Method

CY8CMBR2110 device supports the CapSense Sigma Delta (CSD) with SmartSense Auto-Tuning for converting sensor capacitance (C_X) into digital counts. The CSD method is described in the following sections.

2.2.1 CapSense Sigma-Delta (CSD)

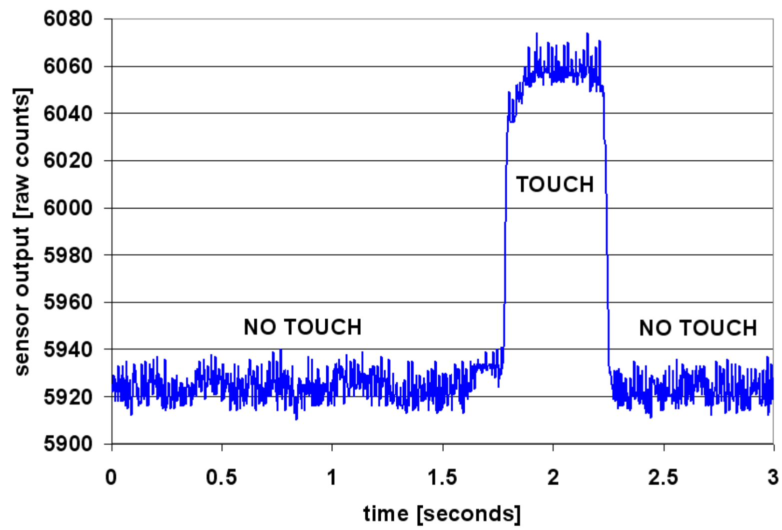
The CSD method in the CY8CMBR2110 device incorporates C_X into a switched capacitor circuit, as shown in Figure 2-3. C_X is alternatively connected to Gnd and the AMUX bus by the non-overlapping switches Sw1 and Sw2. Sw1 and Sw2 are driven by the Precharge Clock to bleed a current, i_{sensor} from the 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 Vref. The sensor bleeds off i_{sensor} from CMOD, which, in combination with Rbus, forms a low-pass filter that attenuates precharge switching transients at the sigma-delta converter input.

Figure 2-3. CSD Block Diagram



In order to maintain the AMUX bus voltage at Vref, the sigma-delta converter matches IDAC to i_{sensor} by controlling the bit stream duty cycle. The sigma-delta converter stores the bit stream over the duration of a sensor scan, and the accumulated result is a digital output, raw count, which is directly proportional to C_X. This raw count is interpreted by high-level algorithms to resolve the sensor state. Figure 2-4 plots the CSD raw counts from a number of consecutive scans during which the 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 steady state raw count level to a predetermined threshold, the high-level algorithms can determine whether the sensor is in an ON (Touch) or OFF (No Touch) state. To learn more about Raw Counts, Finger Threshold, and Signal-to-Noise Ratio (SNR), refer to Getting Started with CapSense.

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



2.3 SmartSense Auto-Tuning

Tuning the touch-sensing user interface is critical for proper system operation and a pleasant user experience. Unfortunately, tuning is time-consuming because it is an iterative process. In a typical development cycle, the interface is tuned in the initial design phase, during system integration, and before production ramp. SmartSense Auto-Tuning was developed to simplify the user interface development cycle. It is a CapSense algorithm that continuously compensates for system, manufacturing, and environmental changes. 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 variation in PCBs and overlays and automatically tunes out noise from sources such as LCD inverters, AC lines, and switch-mode power supplies.

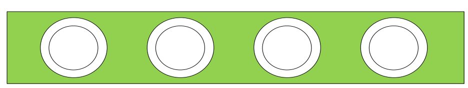
2.3.1 Process Variation

The CY8CMBR2110 device's SmartSense Auto-Tuning is designed to work with C_P values in the range of 5—40 pF. The sensitivity parameter for each button is set automatically, based on its characteristics. This parameter improves yield in mass production because every button maintains a consistent response regardless of C_P variation between the buttons. C_P can vary due to PCB layout and trace length, PCB manufacturing process variation, or vendor-to-vendor PCB variation within a multi-sourced supply chain. The sensitivity of a button depends on C_P; higher C_P values decrease sensitivity, resulting in decreased finger touch signal amplitude. A change in C_P can result in a button becoming too sensitive, not sensitive enough, or non-operational. When this happens, you must retune the system and, in some cases, re-qualify the user interface subsystem. SmartSense Auto-Tuning resolves these issues.

SmartSense Auto-Tuning makes platform designs possible. For example, consider the capacitive touch sensing multimedia keys on a laptop computer. The parasitic capacitance of the CapSense buttons can vary in different models of the same platform design depending on the size of the laptop and the keyboard layout. In this example, a wide-screen laptop model would have larger spaces between the buttons than a standard-screen model. Therefore, a wide-screen model would have longer traces between each button and the CapSense controller, which would result in higher C_P values. Though the buttons' functionality is identical for all of the laptop models, the buttons must be tuned for each model. SmartSense Auto-Tuning lets you do platform designs using the recommended practices shown in the PCB Layout in Getting Started with CapSense.

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

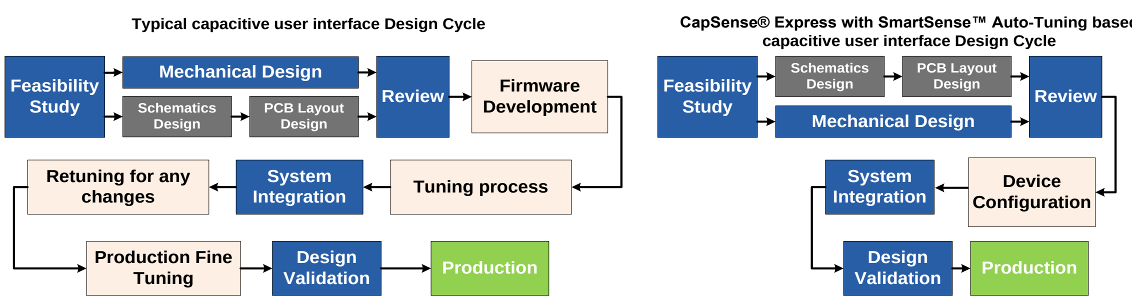
Figure 2-6. Design of Laptop Multimedia Keys for a 15-Inch Model with Identical Functionality and Button Size



2.3.2 Reduced Design Cycle Time

When you design a capacitive button interface, the most time-consuming tasks are firmware development, layout, and button tuning. With a typical touch-sensing controller, the buttons must be retuned when the design is ported to different models or when there are changes to the mechanical dimensions of the PCB or the button PCB layout. A design with SmartSense Auto-Tuning meets these challenges because it does not require firmware development, manual tuning, or retuning. In addition, SmartSense Auto-Tuning speeds up a typical design cycle. Figure 2-7 compares the design cycles of a typical touch-sensing controller and a SmartSense Auto-Tuning-based design.

Figure 2-7. Typical Capacitive User Interface Design Cycle Comparison

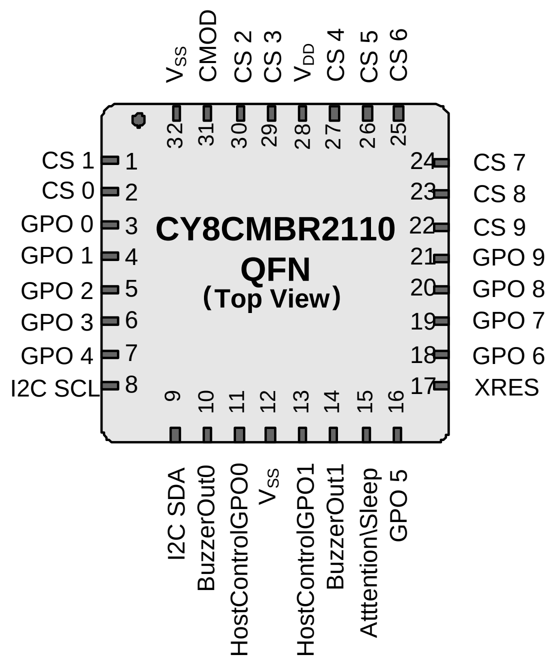


3. CapSense Schematic Design

Cypress's CY8CMBR2110 device is configured using hardware and the EZ-Click Customizer Tool via the I^2C interface. This section gives an overview of the CapSense controller pins and registers and how to configure them.

3.1 CapSense Controller Pins

Figure 3-1. CY8CMBR2110 Pin Diagram



3.1.1 CapSense Buttons (CSx)

The CY8CMBR2110 controller has ten capacitive sense inputs, CS0—CS9. Each capacitive button requires a connection to one of the capacitive sense inputs. You must ground all unused CapSense (CSx) input pins.

3.1.2 General-Purpose Outputs (GPOx)

There are ten active low outputs on the CY8CMBR2110 controller, GPO0—GPO9. Each output is driven by its corresponding capacitive sensing input, CSx. You can use GPOs to directly drive LEDs or to replace mechanical switches. GPOs are in strong drive[2] mode. All unused GPO pins must be floated.

[2] When a pin is in strong drive mode, it is pulled up to V_{DD} when the output is HIGH and pulled down to Ground when the output is LOW.

3.1.3 Modulating Capacitor (CMOD)

Connect a 2.2 nF (±10%) capacitor to the CMOD pin.

3.1.4 Buzzer Signal Outputs (BuzzerOut0, BuzzerOut1)

The buzzer signal outputs are used to give audio feedback when a CapSense button is touched. This is helpful in designs where audio sensors are used. Use piezoelectric buzzers for buzzer signal outputs.

The buzzer signal outputs are strong drive outputs. The outputs are driven commonly by all of the CSx buttons. If a buzzer is not used, BuzzerOut0 and BuzzerOut1 can be used as Host-Controlled GPOs. Table 3-2 shows the various buzzer settings.

The buzzer signal outputs can have two configurations:

1. AC 1-pin Buzzer: A buzzer is connected to the BuzzerOut0 pin of the device as shown in Figure 3-2. A square wave with a specified frequency and duty cycle is driven on this pin. The BuzzerOut1 pin can be left floating, or it can be used as a host-controlled GPO.
2. AC 2-pin Buzzer: A buzzer is connected to the BuzzerOut0 and BuzzerOut1 pins of the device as shown in Figure 3-3. Two out-of-phase square waves with a specified frequency and duty cycle are driven on these pins.

Figure 3-2. AC 1-pin Buzzer Configuration

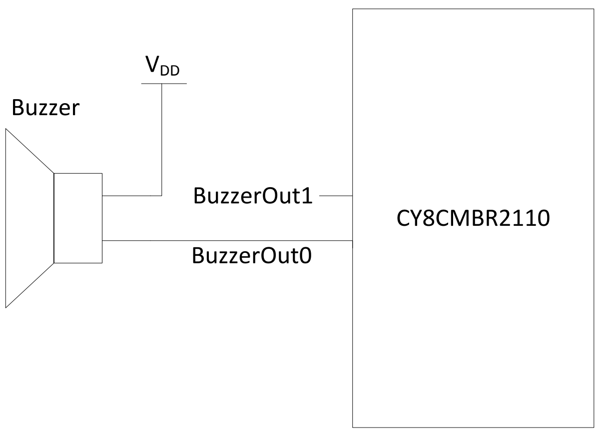
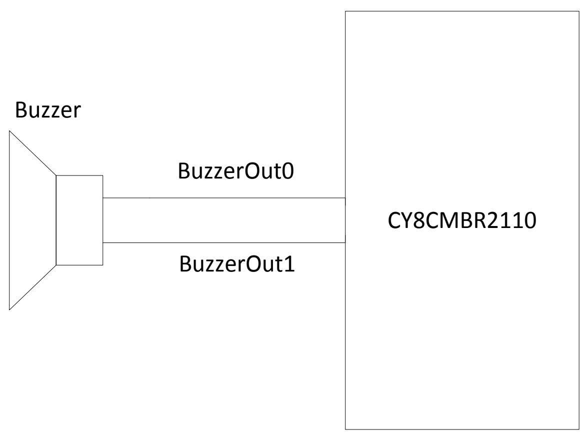


Figure 3-3. AC 2-pin Buzzer Configuration



The buzzer signal frequency is configurable. Table 3-1 lists the various frequencies and the corresponding output duty cycle.

Table 3-1. Buzzer Signal Output frequency and duty cycle

Buzzer Signal Output Frequency (kHz)	Duty Cycle
1.00	50%
1.14	57.14%
1.33	50%
1.60	60%
2.00	50%
2.67	66.7%
4.00	50%

Buzzer ON time has a range of (1 to 127) x Button Scan Rate Constant. To learn more about this constant, refer Button Scan Rate. The buzzer signal output is driven for the configured time and does not depend on the button touch time. The output goes to the idle state after the Buzzer ON time elapses, even if the button remains touched as shown in Figure 3-4. The idle state of the buzzer pin can be configured to either V_{DD} or Ground. The buzzer signal output restarts immediately if a button is touched before the Buzzer ON time elapses as shown in Figure 3-5.

When you enable Buzzer Signal Output, the Buz_Op_Duration register (in the Device Configuration mode) should have a minimum value of 1. To learn more about this register, refer to the CY8CMBR2110 Datasheet Appendix.

Figure 3-4. Buzzer Time-out

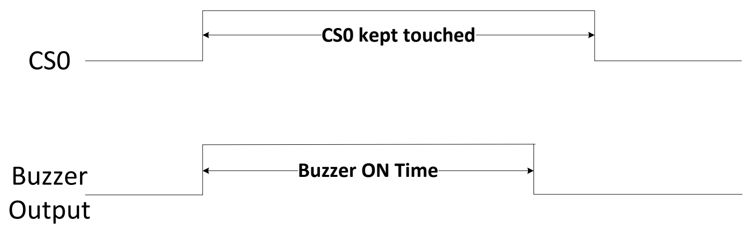
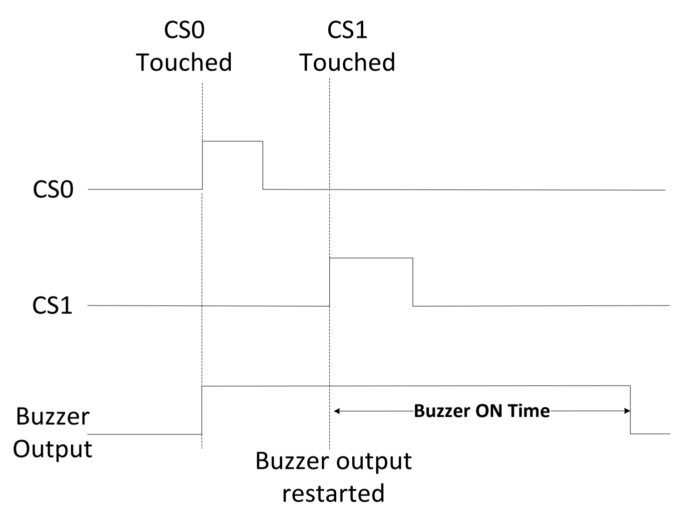


Figure 3-5. Buzzer Signal Output Restart



3.1.5 Host-Controlled GPOs (HostControlGPO0, HostControlGPO1)

The Host Controlled GPOs' logic states can be controlled by the host. These outputs are in strong drive mode.

If a buzzer is not used in your design, the BuzzerOut0 and BuzzerOut1 pins also can be used as host-controlled GPOs. If an AC 1-pin buzzer is used, the BuzzerOut1 pin can be used as a host-controlled GPO.

The host can control these GPOs in the Operating mode, Production Line Test mode, and Debug Data mode.

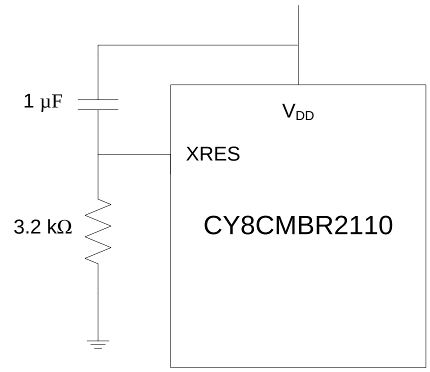
Host-Controlled GPOs are in the LOW state at power-on and have to be configured after reset. The configuration settings cannot be saved to flash memory, unlike other feature configuration settings.

HostControlGPO1 has a positive going pulse of 16 ms during power-on. To eliminate this pulse, use an external RC network (± 5% tolerance) on the XRES pin as shown in Figure 3-6. This keeps the device in XRES reset during every power-on. When the device comes out of XRES reset after 16 ms, normal operation occurs.

Table 3-2. Buzzer and Host-Controlled GPOs

Buzzer Configuration	BuzzerOut0 pin	BuzzerOut1 pin	Maximum Available Host Controlled GPOs
No buzzer	Floating / Host-Controlled GPO3	Floating / Host-Controlled GPO2	4
AC 1-pin	Buzzer pin 0	Floating / Host-Controlled GPO2	3
AC 2-pin	Buzzer pin 0	Buzzer pin 1	2

Figure 3-6. XRES Pin Configuration to Avoid HostControlGPO1 Pulse During Power-On



3.1.6 Attention/Sleep

Attention/Sleep is a bidirectional line in Open Drain Low drive mode that can be controlled by both the host and the device. Attention/Sleep is used to read CapSense data from the device and to enter and exit Low-Power Sleep and Deep Sleep modes.

3.1.6.1 Read Device Data

Two steps are required for the host to read data from the device.

1. The host pulls the Attention/Sleep line low.
2. The host initiates I^2C communication with the device.

When the Attention/Sleep line is pulled high, the device is in Low-Power Sleep or Deep Sleep mode (if the Deep Sleep bit in Host_Mode register is set). The device can NACK I^2C communication at this time. Keep the Attention/Sleep line pulled HIGH to conserve power.
To read the device data, the host can pull the Attention/Sleep line low at any time. When the Attention/Sleep line is low, the device can NACK I^2C communication, but very infrequently.

If any CapSense button is touched, the device pulls the Attention/Sleep line low to interrupt the host, as shown in Figure 3-7. The host then can read the CapSense data using I^2C communication with the device. If more than one button is touched simultaneously, the Attention/Sleep line is pulled low for the duration, as shown in Figure 3-8. The Attention/Sleep line goes high when the button is released.

The host should have both a falling edge and a rising edge triggered interrupt for the Attention/Sleep line, so it can recognize both a button touch and a button release. If a rising edge triggered interrupt is not available, the host should continuously poll the button status after the Attention/Sleep line goes low. Polling should be done at the Button Scan Rate constant.

Figure 3-7. Attention/Sleep Line Status with CS0 and CS1 Touched Separately

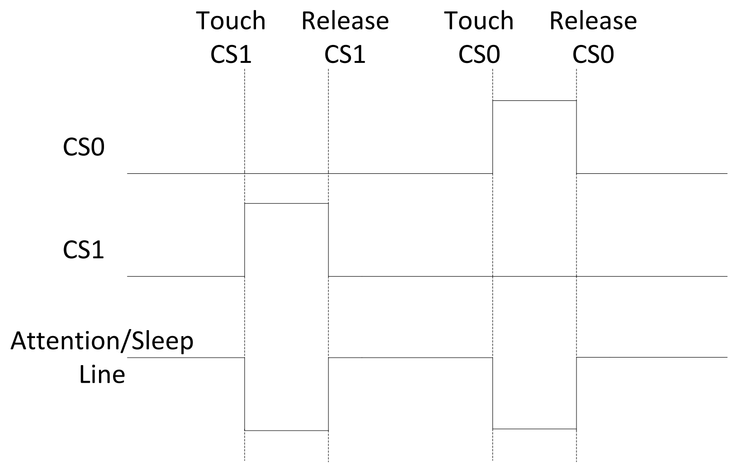
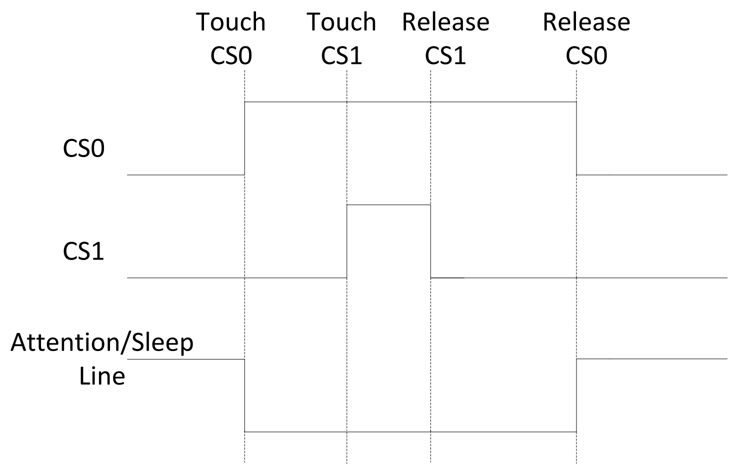


Figure 3-8. Attention/Sleep Line Status with CS0 and CS1 Touched Simultaneously



3.1.6.2 Sleep Modes

There are two possible sleep mode configurations

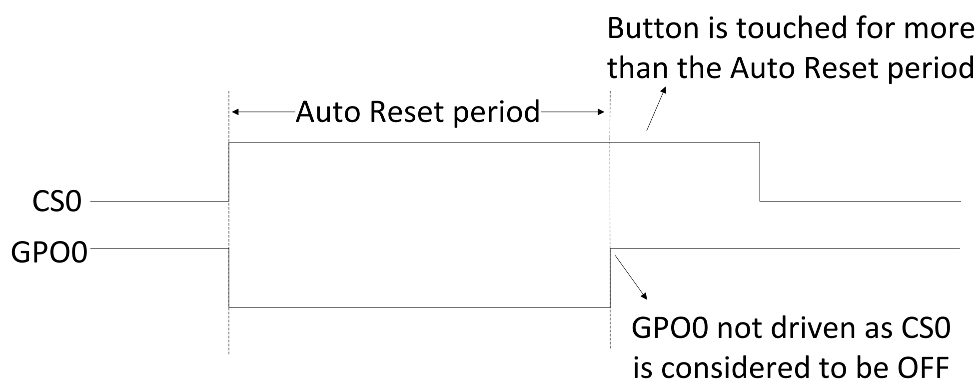
1. Pull the Attention/Sleep line to V_{DD} to enable Low Power Sleep Mode.
2. Pull the Attention/Sleep line to V_{DD} and set the Deep Sleep bit in Host_Mode register (in Operating Mode) to enable Deep Sleep Mode.

3.2 CapSense Controller Configuration

3.2.1 Button Auto Reset (ARST)

Button Auto Reset determines the maximum time a button is considered to be ON when CSx is continuously touched. The button is turned OFF after the ARST period. This feature prevents a button from getting stuck if a metal object is placed too close to it. The ARST period can be configured to either 5 seconds or 20 seconds. The Button Auto Reset is shown in Figure 3-9.

Figure 3-9. Button Auto Reset



After the CSx is turned off because of Button Auto Reset and after the button is released, do not touch the button for a time equal to the Button Scan Rate.

3.2.2 Noise Immunity

This setting 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.

In a system without major noise concerns, select "Normal" Noise Immunity. For a system in a high-noise environment, select "High" Noise Immunity. Power consumption and response time increase when Noise Immunity is "High". If you require the same response time with "High" Noise Immunity, reduce the button debounce value. For more information, refer to Debounce Control.

3.2.3 Automatic Threshold

As explained in CapSense Sigma-Delta (CSD), the sensor ON or OFF state is determined by comparing the shift in raw counts to a predetermined threshold, called the Finger Threshold. Finger Threshold is configurable and decides the other thresholds for the device. To learn more about the Finger Threshold, refer to Getting Started with CapSense.

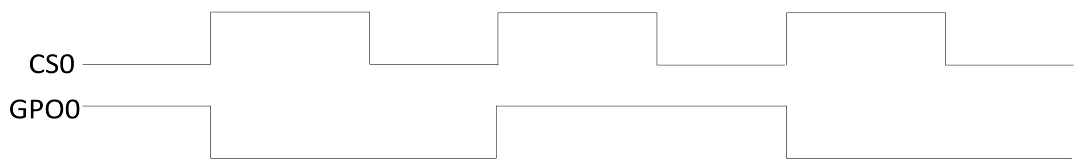
You can configure the Finger Threshold for each button individually or use the Automatic Threshold feature. The Automatic Threshold sets the various thresholds dynamically for each button, depending on the noise in the environment. For a variable noise environment, use Automatic Threshold. If you need to manually adjust the finger threshold, disable Automatic Threshold and set the finger threshold to the desired level.

3.2.4 Toggle ON/OFF

When Toggle ON/OFF is enabled, the state of GPOx changes on every rising edge of CSx. Toggle ON/OFF configuration is shown in Figure 3-10.

You can enable the toggle ON/OFF feature on each CapSense button individually.

Figure 3-10. Example of Toggle ON/OFF Feature



3.2.5 Flanking Sensor Suppression (FSS)

FSS distinguishes between signals from closely spaced buttons, eliminating false touches. It ensures that the system recognizes only the first button touched. FSS allows only one CSx to be in the TOUCH state at a time. If a finger contacts multiple CSx buttons, only the first one to sense a TOUCH state will turn ON.

FSS also is useful when nearby buttons can produce opposite effects such as an interface with two buttons for brightness control (UP or DOWN).

FSS can be enabled for each button individually. FSS configuration is shown in Figure 3-11 and Figure 3-12.

In applications such as washing machine panels, buttons can be separated into two groups: one with FSS enabled and one with FSS disabled. This allows you to distinguish between closely spaced buttons at one end of the design, while accommodating multi-touch functionality at the other end.

Figure 3-11. FSS When Only One Button is Touched

No button is ON prior to the touch

CS1 is reported as ON upon touch

Figure 3-12. FSS When Multiple Buttons are Touched With One Button ON Previously

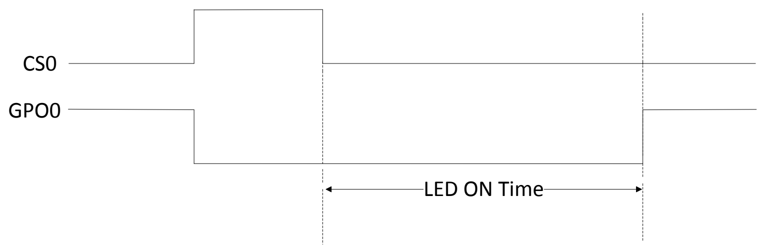
CS1 is touched; reported ON

CS2 is also touched along with CS1;
only CS1 is reported ON

3.2.6 LED ON Time

LED ON Time specifies the duration for which GPOx is driven low after CSx is released as shown in Figure 3-13. LED ON Time can range from 0—5100 ms, with a resolution of 20 ms.

Figure 3-13. LED ON Time

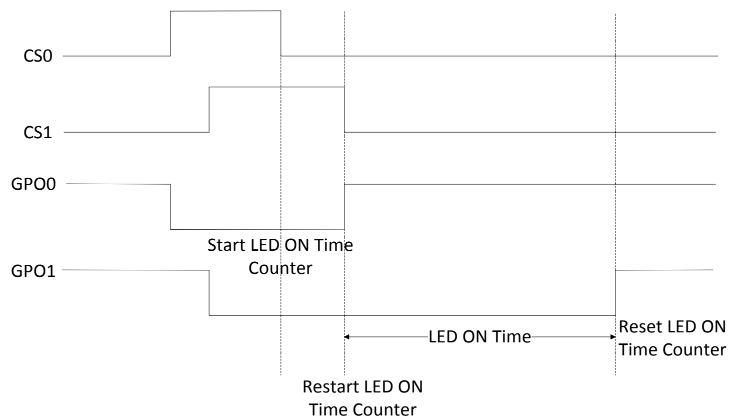


LED ON Time varies from device to device. Accuracy is ±10% at a range of -40 °C to +85 °C.

If a Button Auto Reset (ARST) is triggered for CSx, LED ON Time is not applied on GPOx. LED ON Time is disabled if Toggle ON/OFF is enabled.

LED ON Time applies only to one GPOx at a time, meaning the LED ON Time counter resets every time a CSx transitions to a NO TOUCH state. Figure 3-14 illustrates how LED ON Time operates when multiple buttons are touched. CS1 resets the LED ON Time counter, causing GPO0 to turn OFF prematurely.

Figure 3-14. LED ON Timing for Multiple Buttons



3.2.7 LED Effect Parameters

Power-On LED Effects and Button Touch LED Effects use the following parameters:

- Low-brightness: Minimum LED intensity
- Low-brightness time: The time period the LED remains in a low-brightness state
- Ramp-up time: The time period the LED transitions from low-brightness to high-brightness
- High-brightness: Maximum LED intensity
- High-brightness time: The time period the LED remains in a high-brightness state
- Ramp-down time: The time period the LED transitions from high-brightness to low-brightness
- Repeat rate: The number of times the effects are repeated

GPOs are configured in groups to have the same parameters. The different groups are:

- {GPO1, GPO2, GPO3}
- {GPO4, GPO5, GPO6}
- {GPO7, GPO8, GPO9}

GPO0's parameters can be configured separately. This functionality is useful in designs where the CS0 button has a special function such as the power button.

The brightness levels can range from 0—100%. The time range is 0—1600 ms. High-brightness should be kept higher than low-brightness.

3.2.7.1 Power-On LED Effects

If this feature is enabled, all LEDs connected to GPOs show dimming and fading effects for an initial time, at system power-on. You can configure these effects and the effect time. During this time, all CapSense buttons are disabled. The device responds to any button touch only after the effects are complete.

The effects are seen after the device initialization time from power-on. This time is less than 350 ms if Noise Immunity is "Normal" and less than 1000 ms if Noise Immunity is "High".

After power-on, system diagnostics, including a power-on self-test, are performed. If any CapSense button fails, the effects are not seen on the corresponding GPO. To learn more about this test, see System Diagnostics.

During Power-On LED Effects, the device ACKs I^2C communication, but all write commands are ignored. The host can only read Operating Mode data.

Power-On LED Effects can be configured to occur concurrently or sequentially on all the GPOs as shown in Figure 3-15 and Figure 3-16.

Figure 3-15. Example Power-On LED Effects (Concurrent)[3]

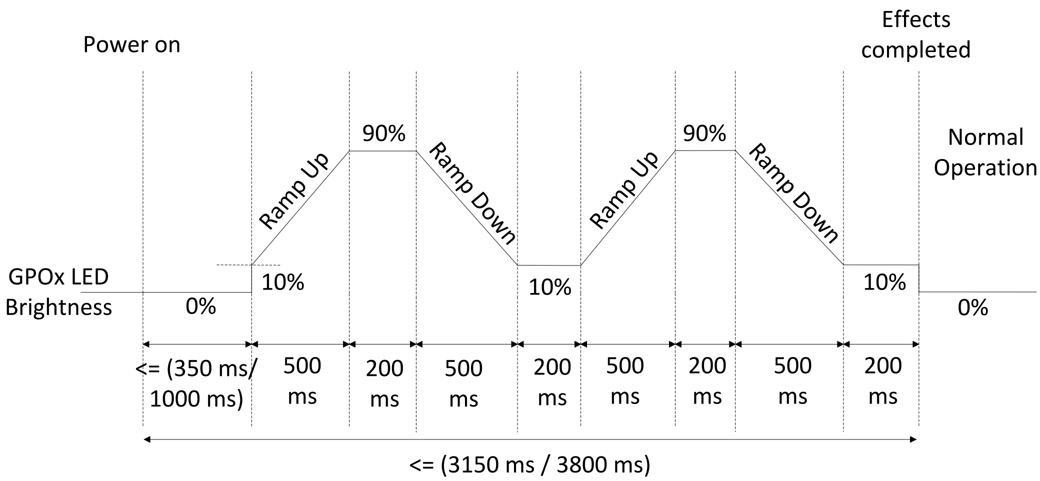
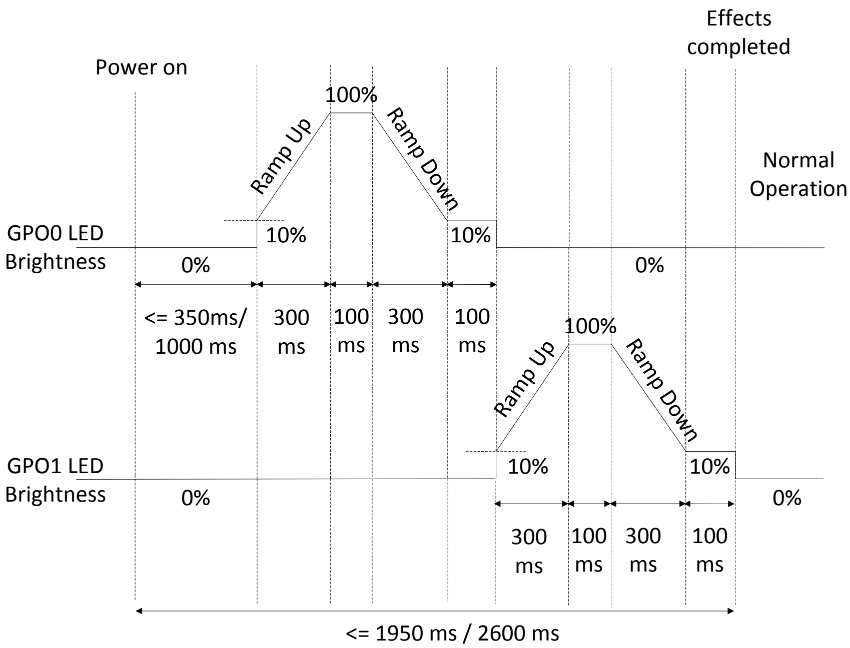


Figure 3-16. Example Power-On LED Effects (Sequential) with Two-Button Design[4]



3 Ramp up time = 500 ms; High-brightness = 90%; High-brightness time = 200 ms; Ramp down time = 500 ms; Low-brightness = 10%; Low-brightness time = 200 ms; Repeat rate = 1

4 Ramp up time = 300 ms; High-brightness = 100%; High-brightness time = 100 ms; Ramp down time = 300 ms; Low-brightness = 10%; Low-brightness time = 100 ms; Repeat rate = 0

3.2.7.2 Button Touch LED Effects

When this feature is enabled if a button is touched, the associated LEDs connected to GPOs show dimming and fading effects. You can configure these effects and the effect time.

Button-Controlled LED Effects can be breathing effect enabled or disabled. Both are shown in Figure 3-17.

Breathing Effect Enabled: With the breathing effect enabled, LED intensity changes from Standby Mode LED Brightness to low-brightness immediately when a button is touched. The LED then ramps up to high-brightness and stays at that level for high-brightness time. The LED then ramps down to low-brightness and stays at that level for low-brightness time. This effect repeats as long as the button is touched. When the button is released, the breathing effect cycle continues until it is complete. The breathing effects cycle may repeat depending on the repeat rate.

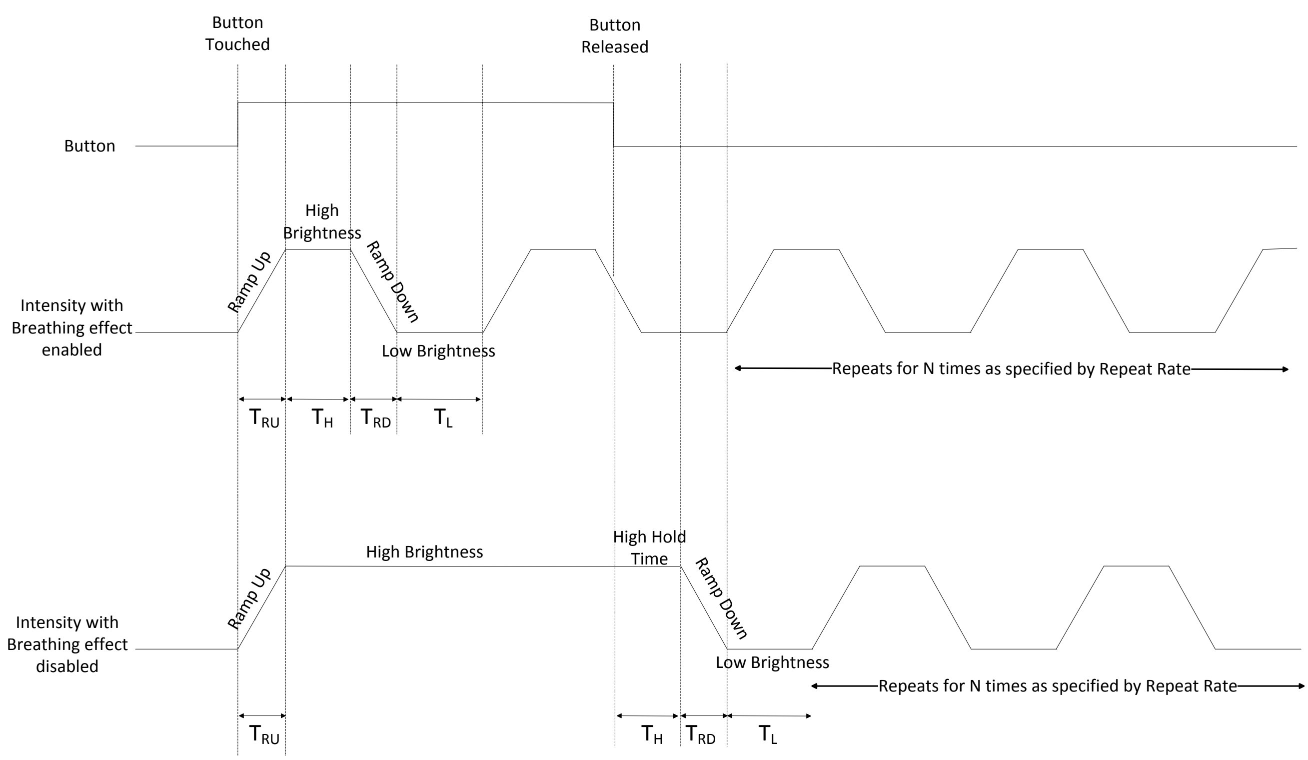
Breathing Effect Disabled: With the breathing effect disabled, the LED intensity changes from Standby Mode LED Brightness to low-brightness immediately when a button is touched. The LED then ramps up to high-brightness and stays at that level as long as the button is touched. When the button is released, the LED maintains high-brightness for high-brightness time then ramps down to low-brightness and stays at that level for low-brightness time. This effect may repeat depending on the repeat rate.

If the Button Touch LED Effects are ongoing on a GPOx and the corresponding CSx is touched again, then the pattern restarts on GPOx.

If the Toggle ON/OFF feature is enabled, the LEDs toggle between Standby Mode LED Brightness and high-brightness on successive button touches as shown in Figure 3-18.

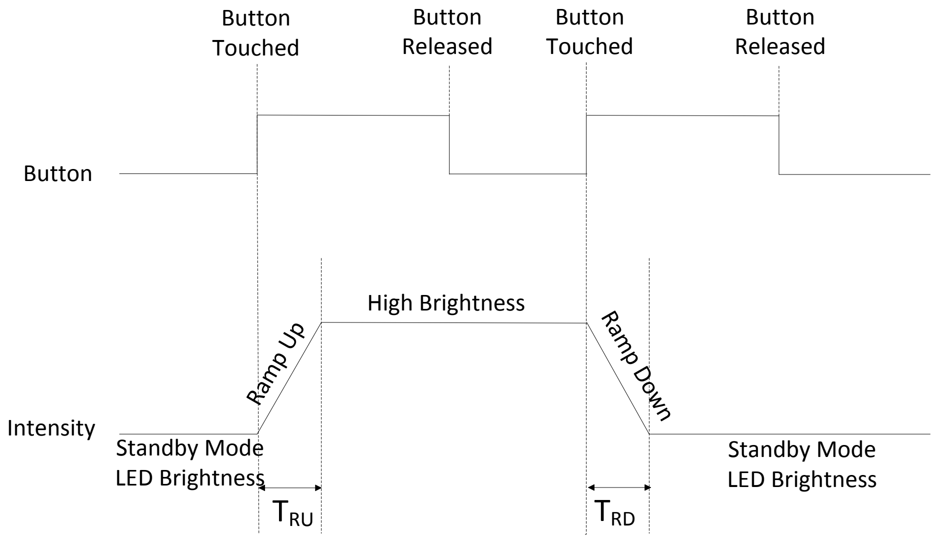
When Button Touch LED Effects are enabled, the LED ON Time is automatically disabled. When the device goes into Deep Sleep, ongoing Button Touch LED Effects are immediately disabled.

Figure 3-17. Button Touch LED Effects[5]



5 T_{RU} = Ramp Up Time
T_{RD} = Ramp Down Time
T_H = High-Brightness
T_L = Low-Brightness

Figure 3-18. Button Touch LED Effects with Toggle ON/OFF Enabled

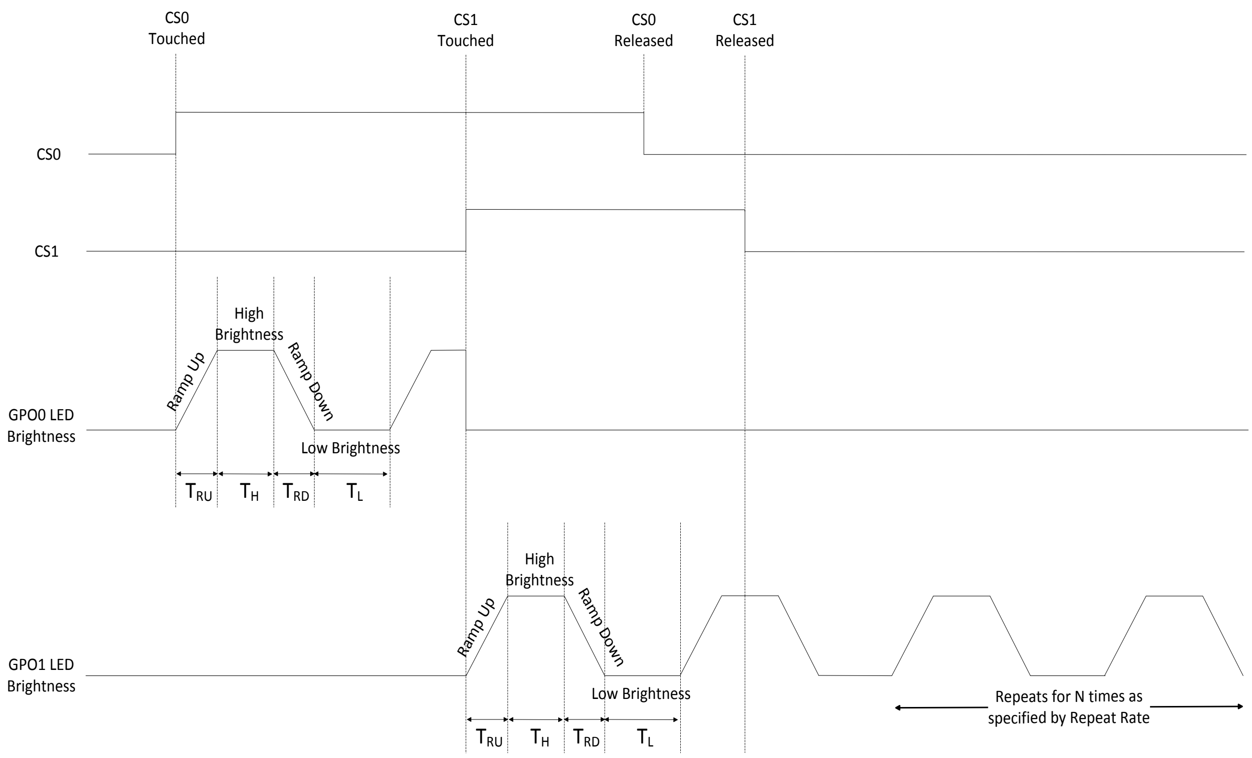


3.2.7.3 Last Button LED Effect

You can configure Button Touch LED Effects to be interrupted on one GPO if any other button in touched. The effects reset on the first GPO and start on the GPO associated with the last button touched as shown in Figure 3-19. This feature is disabled by default.

If Toggle ON/OFF is also enabled for some buttons, the Last Button LED Effect is disabled for those buttons. If Flanking Sensor Suppression (FSS) is enabled, and two buttons are touched simultaneously, Last Button LED Effect does not apply, as the second button touched does not turn ON.

Figure 3-19. Button Touch LED Effects (Breathing Enabled) with Last Button LED Effect Enabled



3.2.7.4 Standby Mode LED Brightness

When the CapSense button CSx is OFF, you can configure the LED associated with the corresponding GPOx to have a Standby Mode LED Brightness for LED backlighting. This configuration improves the look-and-feel of the design.

Standby Mode LED Brightness can be configured to be 0%, 20%, 30%, or 50%. Standby Mode LED Brightness should be the same as low-brightness.

The LEDs associated with GPOx remain on Standby Mode LED Brightness after the conclusion of Power-On LED Effects or Button Touch LED Effects, when the CSx is OFF.

Standby Mode LED Brightness increases the power consumption of the device because the device does not go into Low-Power Sleep mode. When the device goes into Deep Sleep mode, Standby Mode LED Brightness is disabled.

3.2.8 Latch Status Read

When a CapSense button CSx is touched, the device generates an interrupt to the host by pulling the Attention/Sleep line low. Then, the host processor can read the device Register Map through I^2C communication to learn the CapSense button status. To learn more refer to Attention/Sleep. To learn more about I^2C communication, refer to the CY8CMBR2110 Datasheet.

When the device interrupts the host, the host may not be able to service the interrupt immediately. As a result, the host could miss the button touch. To avoid missing any button touches, the host needs to read both the button status (CS) and the latch status (LS) for the proper information about any button touch. CS is stored in Button_Current_Stat0 and Button_Current_Stat1 registers in Operating Mode. LS is stored in Button_Latch_Stat0 and Button_Latch_Stat1 registers in Operating Mode. For register map details, refer to the CY8CMBR2110 Datasheet Appendix.

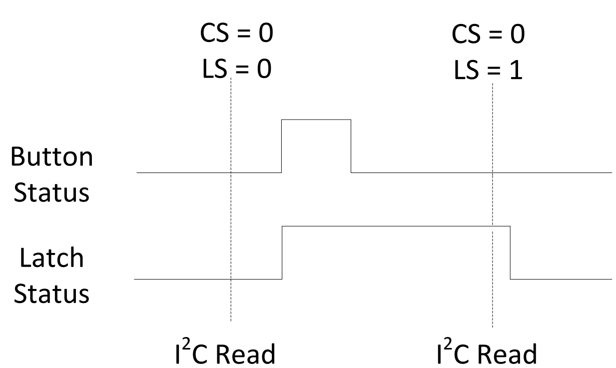
The Button Status bit is set on a button touch and cleared on button release. The Latch Status bit is set on a button touch. This bit is automatically cleared when the host reads the Button status.

Table 3-3 lists the various cases for a button touch and its acknowledgment. These cases are shown in Figure 3-20 and Figure 3-21.

Table 3-3. Latch Status Read

Button Status (CS)	Latch Status (LS)	Interpretation
0	0	CSx is not touched during the current I^2C read Host has already acknowledged any previous CSx touch in the last I^2C read
0	1	CSx was touched before the current I^2C read This CSx touch was missed by the host
1	0	CSx was touched and acknowledged by the host during the previous I^2C read This CSx is still touched during current I^2C read
1	1	CSx is touched during the current I^2C read

Figure 3-20. Latch Status Read Case 1



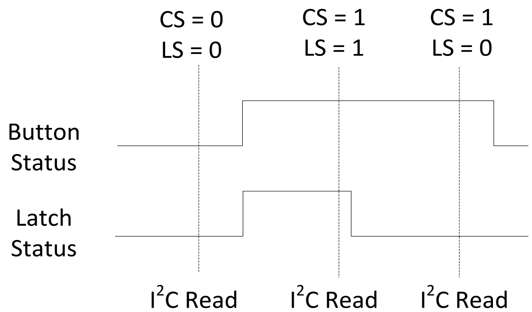
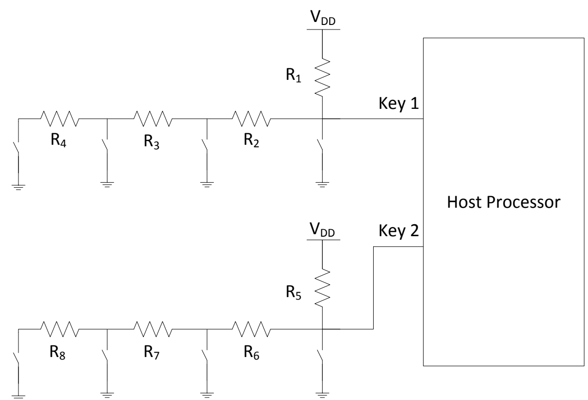


Figure 3-21. Latch Status Read Case 2

3.2.9 Analog Voltage Support

A general external resistive network with a host processor, such as the one shown in Figure 3-22, can configure the host to perform different functions based on the voltage level seen at the input pin. You can vary this voltage level using a combination of resistors and switches between V_{DD} and ground.

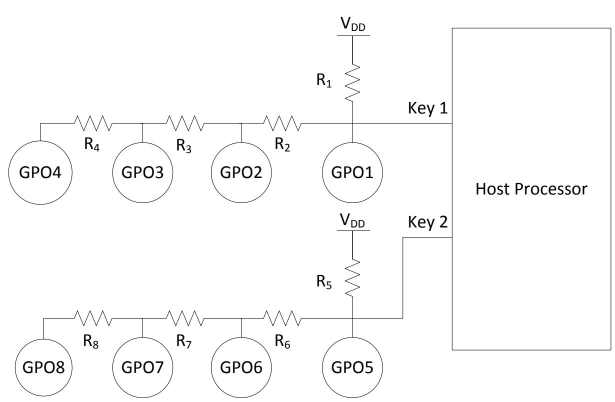
Figure 3-22. A General External Resistive Network



The analog voltage support feature of CY8CMBR2110 gives you the option to control these switches using CapSense buttons. Each switch can be replaced with one GPOx. When a CSx button is touched, the corresponding GPOx goes low; therefore, the switch is closed (shorted to ground). When the button is released, the corresponding switch is left open. This is shown in Figure 3-23.

If this feature is enabled, the GPOs cannot be used simultaneously in the external resistive network and for the LED drive. If only one button needs to be ON for analog voltage support, enable FSS with this feature. Usually, the GPO pins are in strong drive mode, however, when this feature is enabled, the GPOs are in Open Drain Low drive mode.

Figure 3-23. Analog Voltage Support from CY8CMBR2110



3.2.10 Sensitivity Control

The sensitivity of each CapSense button can be set individually. Sensitivity determines the minimum C_F required to turn ON a button. The following factors affect the button's sensitivity:

1. Overlay thickness: The thicker the overlay, the higher the sensitivity requirement.
2. System noise: As system noise increases, sensitivity needs to be lower, to avoid false button triggers.
3. Form factor of the design: A relatively large button size is required to support a low sensitivity (Higher C_F). For small-button diameters, the sensitivity needs to be high.
4. Power Consumption: Power consumption increases for high sensitivity buttons. For low power consumption needs, the sensitivity needs to be low.

The different sensitivity settings available are "High", "Medium", and "Low".

3.2.11 Debounce Control

The Debounce feature avoids false button triggering from noise spikes or system glitches, by specifying the minimum time a button has to be touched for a valid touch input.

The debounce time can vary depending on the button's function. For example, the power button should have a long debounce time to avoid inadvertently switching the system ON/OFF. Shorter debounce times speed up the device's response to a button touch.

The debounce value for the CS0 button can be set separately from the CS1—CS9 buttons. This functionality is useful in designs where the CS0 button has a special function such as the power button. The debounce can range from 1—255.

The device's Response Time depends on the button debounce. Table 3-4 lists some examples of device Response Time for different debounce values[6].To calculate the Response Time for any debounce value, refer to Response Time.

Table 3-4. Example Response Times for Debounce Values

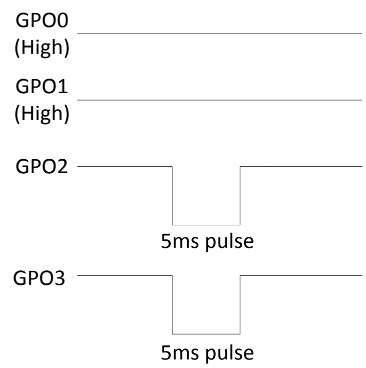
Debounce Value	Response Time for Consecutive Button Touch (ms)
1	70
4	105
7	140
10	175
100	1225
200	2380
255	3010

3.2.12 System Diagnostics

A built-in power-on self-test (POST) mechanism performs five tests at power-on reset (POR), which can be useful in production testing. If any button fails, a 5-ms pulse is sent out on the corresponding GPO within 350 ms if Noise Immunity is "Normal" and 1000 ms if Noise Immunity is "High".

[6] 8-buttons, Noise Immunity level "Normal", Response Time optimized design

Figure 3-24. Example Showing CS0, CS1 Passing the POST and CS2, CS3 Failing



To find out the result of the System Diagnostics, use the EZ-Click Customizer Tool. To learn more about the tool, refer to the EZ-Click Customizer Tool User Guide.

If you need to read the entire device's data, you can change the device's Register Map mode to "Production Line Test" mode and read the data through the I^2C lines. To learn more about changing Register Map modes, refer to the CY8CMBR2110 Datasheet Register Map Modes. To learn more about device data, refer to I2C Communication.

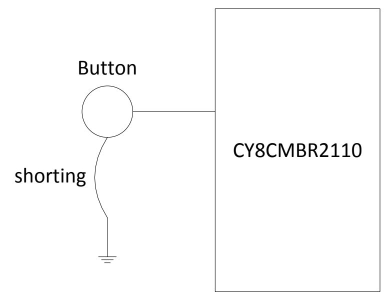
Because you can read the GPOs' status using I^2C, you do not need to create an interface between the GPOs and the host controller pins.

The following tests are performed on all of the buttons.

3.2.12.1 Button Shorted to Ground

If any button is found to be shorted to ground, it is disabled.

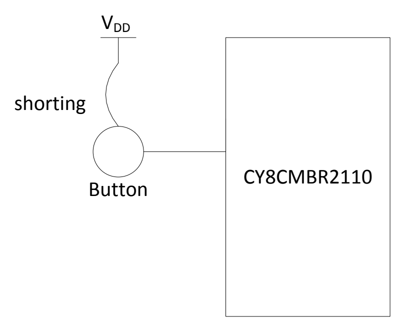
Figure 3-25. Button Shorted to Ground



3.2.12.2 Button Shorted to V_{DD}

If any button is found to be shorted to V_{DD}, it is disabled.

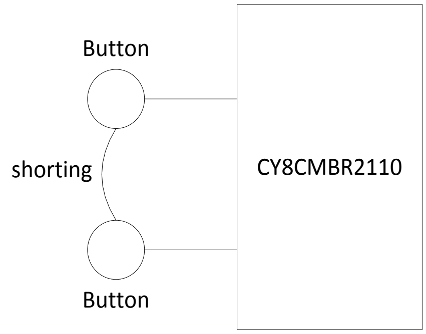
Figure 3-26. Button Shorted to V_{DD}



3.2.12.3 Button-to-Button Short

If two or more buttons are found to be shorted to each other, all of these buttons are disabled.

Figure 3-27. Button-to-Button Short



3.2.12.4 Improper Value of CMOD

Recommended value of CMOD is 2.2 nF, ±10%.

If the value of CMOD is found to be less than 1 nF or greater than 4 nF, all of the buttons are disabled.

3.2.12.5 Button C_P > 40 pF

If any button's C_P is greater than 40 pF, that button is disabled.

3.2.13 Button Scan Rate

The button scan rate specifies the time between successive button scans by the device. Use the following equation to calculate the rate:

Button Scan Rate = Button Scan Rate constant + Button Scan Rate offset   Equation 4

The Button Scan Rate is configurable from 25—561 ms.

The Button Scan Rate constant depends on the number of buttons and the Noise Immunity level selected. For a higher number of buttons, the constant is higher. Similarly, for "High" Noise Immunity, the constant is higher.

If you use a maximum of five buttons, the Button Scan Rate constant depends on how you optimize your design:

Response Time Optimization: The time between consecutive button scans is shorter. As more scans occur in a fixed time, the device responds more quickly to a button touch. However, power consumption increases.

Power Consumption Optimization: The time between consecutive button scans is longer. As fewer scans occur in a fixed time, the device takes longer to respond to a button touch. As a result, power consumption decreases.

You can configure the Button Scan Rate offset using the EZ-Click Customizer Tool. The Button Scan Rate constant is given in Table 3-5.

Table 3-5. Button Scan Rate Constant

Button Count	Button Scan Rate Constant			
	Response Time-Optimization		Power Consumption Optimization	
	Noise Immunity "Normal"	Noise Immunity "High"	Noise Immunity "Normal"	Noise Immunity "High"
≤ 5	25	35	35	55
> 5	35	55	35	55

As an example, consider a design with four buttons and the following parameters:

- C_P between 10—20 pF for all buttons
- Sensitivity is high for all buttons
- Noise Immunity is "Normal"
- Debounce for each button is set to 10

- Average button touches per hour = 200
- Average touch time = 1000 ms
- Buzzer and Button Touch LED Effects are disabled
- Button Scan Rate offset = 0.
- The current consumption per button is:
  - Response Time Optimized = 0.3075 mA
  - Power Consumption Optimized = 0.2204 mA

The response times for first button touch as well as consecutive button touches are:

- Response Time Optimized = 125 ms
- Power Consumption Optimized = 175 ms

Note that the response time optimized design consumes a lot more power and responds more quickly to a button touch when compared to the power consumption optimized design. To find the response time for your design, refer to the Design Toolbox.

Button scan rate varies from device to device, and it is ±10% accurate at a temperature range of -40 °C to +85 °C.

3.2.14 I^2C Communication

I^2C is the interface used to communicate between the CY8CMBR2110 (I^2C slave) and the host (I^2C master).

To learn more about the protocol and the communication procedure, refer to the CY8CMBR2110 Datasheet I^2C Communication section.

For proper I^2C communication between the host and the device, follow these guidelines:

- The host processor should pull the Attention/Sleep line low before initiating any I^2C communication or the device might NACK the host.
- The host processor should not initiate or continue an I^2C communication with the device unless:
  - The host needs to configure the device.
  - The device interrupts the host.
  - The host needs to read and verify the device register map contents.
- To reduce power consumption, avoid prolonged I^2C communication with the device.
- The host should wait for 350 ms if Noise Immunity is “Normal” or 1000 ms if Noise Immunity is “High” after device power-on before initiating any I^2C communication. Otherwise, the device NACKs any such communication.
- The host should wait for a minimum of 60 ms after any I^2C transaction before initiating a new transaction.
- The host should wait for 350 ms if Noise Immunity is “Normal” or 1000 ms if Noise Immunity is “High” after “Save to Flash” or “Software reset” commands are issued before initiating any I^2C communication.
- The device should be in Operating Mode in runtime.
- The host should not initiate a new START condition for the device without initiating a STOP condition for the previous I^2C communication. This is also called Repeat Start condition.
- The host should maintain a minimum of 60 ms between consecutive I^2C transactions.
  - If the host initiates another I^2C transaction before this time, it will receive the same data as in the previous transaction.
  - If the host writes to the same register as the one in the previous transaction within this time, the old data is lost.
  - If the host writes to a different register than the one in the previous transaction within this time, the register keeps this data. The data from the previous transaction is not lost.

3.3 Design Toolbox

The Design Toolbox helps you to design a CY8CMBR2110 CapSense solution. It offers basic information about the board layout and feature settings and recommends whether the design is fit for mass production.

3.3.1 General Layout Guidelines

Figure 3-28 summarizes the layout guidelines for the CY8CMBR2110. These guidelines are discussed in Electrical and Mechanical Design Considerations. For a thorough treatment of this material, see Getting Started with CapSense.

Figure 3-28. Design Layout Recommendations

General Layout Guidelines

Sl. No.	Category	Min	Max	Recommendations/Remarks
1	Button shape			Solid round pattern, round with LED hole, rectangle with round corners
2	Button size	5 mm	15 mm	Given in Layout Estimator sheet
3	Button-button spacing	equal to button ground clearance		8 mm
4	Button-ground clearance	0.5 mm	2 mm	Given in Layout Estimator sheet
5	Ground flood - top layer			Hatched ground 7 mil trace and 45 mil grid (15% filling)
6	Ground flood - bottom layer			Hatched ground 7 mil trace and 70 mil grid (10% filling)
7	Trace length from sensor pad to device pin		450 mm	450 mm is for FR4 PCB, with a button diameter of 5 mm and a pin capacitance of 7 pF. For a different design, refer to Layout Estimator sheet.
8	Trace width	0.17 mm	0.20 mm	0.17 mm (7 mil)
9	Trace routing			Traces should be routed on the non sensor side. If any non CapSense trace crosses CapSense trace, ensure that the intersection is orthogonal.
10	Via position for the sensors			Via should be placed near the edge of the button to reduce trace length thereby increasing sensitivity.
11	Via hole size for sensor traces			10 mil
12	Number of via on sensor trace	1	2	1
13	CapSense series resistor placement		10 mm	Place CapSense series resistors close to the device for noise suppression. CapSense resistors have highest priority compared to LED resistors. Place them first.
14	Distance between any CapSense trace to ground flood	10 mil	20 mil	20 mil
15	Device placement			Mount the device on the layer opposite to the sensor. The CapSense trace length between the device and the sensors should be minimum (see trace length above)
16	Placement of components in two layer PCB			Top layer - sensors and bottom layer - device, other components and traces.
17	Placement of components in four layer PCB			Top layer-sensors, 2nd Layer – CapSense traces & Vdd and avoid the Vdd traces below the sensors, 3rd Layer-hatched ground, Bottom layer- device other components and non CapSense traces
18	Overlay thickness	0 mm	5 mm	Use layout Estimator sheet to decide on overlay, given maximum limit is for plastic overlay.
19	Overlay material			Should be non-conductive material. Glass, ABS Plastic, Formica, wood etc. No air gap should be there between PCB and overlay. Use adhesive to stick the PCB and overlay.
20	Overlay adhesives			Adhesive should be non conductive and dielectrically homogenous. 467MP and 468MP adhesives made by 3M are recommended.
21	LED backlighting			Cut a hole in the sensor pad and use rear mountable LEDs.
22	Board thickness			Standard board thickness for CapSense FR4 based designs is 1.6 mm.

3.3.2 Layout Estimator

The Layout Estimator provides the minimum button size and maximum trace length recommendation based on the intended end-system requirements and industrial design. The inputs include the overlay material, overlay thickness, trace capacitance of circuit board material, and CapSense button sensitivity. See Figure 3-29, Table B, to learn the dielectric constants for different overlay materials and the trace capacitance per unit length for different PCBs. Table A calculates the minimum button diameter and maximum trace length for the design, based on three system noise conditions. “Low”, “Medium”, and “High” noise conditions are relative figures of merit to help you with button development. Noise conditions can vary from button to button based on the end-system environment. If the noise conditions are unknown, use medium as the starting point. The actual noise seen at each button will be determined during Design Validation .

Use the outputs of this sheet to guide the button board layout process, and then check the design prior to prototyping with the C_P, Power Consumption and Response Time Calculator sheet, as detailed in CP, Power Consumption and Response Time Calculator.

Figure 3-29. Layout Estimator

Layout Estimator

TableA: Layout Estimator

Input Parameters	Value	Units	Comments
Overlay Thickness	1.5	mm	
Overlay - Dielectric constant	4	farad/m	
Capacitance of trace per inch	2	pF	
CSx Sensitivity	High		If the power consumption is critical, select "Low" sensitivity. If the board form factor is critical, select "High" sensitivity.
Minimum Recommended Button Diameter			
Noise Conditions - Low (0.05 pF Noise)	5	mm	
Noise Conditions - Medium (0.075 pF Noise)	6	mm	
Noise Conditions - High (0.1 pF Noise)	7	mm	
Maximum Trace length			
Noise Conditions - Low (0.05 pF Noise)	412	mm	
Noise Conditions - Medium (0.075 pF Noise)	406	mm	
Noise Conditions - High (0.1 pF Noise)	400	mm	
Button to Ground clearance	1.5	mm	

(green)	input cells, edit with actuals
(blue)	output cells, based on inputs

Note: Button diameter of all the buttons CS1 to CS9 will be same with respect to overlay thickness, but can differ with respect to noise conditions

TableB - 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, 64mil, FR4	2
Copper trace, flex PCB, 2 layer	8

Inputs

- Overlay thickness
- Overlay dielectric constant
- Capacitance of trace per inch of board
- CSx sensitivity

Outputs

- Recommended minimum button diameter and maximum trace length for different noise conditions
- Button-to-ground clearance

The diameter of each button can vary based on the variation in noise in each button.

3.3.3 C_P, Power Consumption and Response Time Calculator

After the board layout has been completed, the Power Consumption and Response Time Calculator shown in Figure 3-30 checks the design before building the button board prototype. To verify the C_P value of each button, insert the button diameters and trace lengths into Table A. After you enter the information, the toolbox confirms whether each button is within the specified C_P range of 5—40 pF.

The power calculator in Table B is used to optimize power consumption. Power consumption is a function of the button scan rate, noise immunity level, and the percentage of active time. Active time is calculated by multiplying the average number of button touches per hour by the maximum of the following three values: Button touch time, Buzzer ON time or Button Touch LED Effects. This is converted into the percentage of active time, and the power consumption is calculated accordingly. Ensure that you do not keep all the following input cells empty (or zero) at the same time:

1. Average number of button touches per hour
2. Average button touch time
3. Average Buzzer ON time
4. Average Button Touch LED Effects time

Table C outputs the button response time based on the inputs in Tables A and B. The debounce value affects the button response time.

Figure 3-30. C_P, Power Consumption and Response Time Calculator

Cp, Power Consumption and Response Time Calculator

Table A: Cp Calculator

Sensor	Button diameter		Trace length		Sensitivity	Parasitic capacitance (Cp) of sensors (Approx)		Comments
CS0		mm		mm	Medium	0	pF	
CS1		mm		mm	Medium	0	pF	
CS2		mm		mm	Medium	0	pF	
CS3		mm		mm	Medium	0	pF	
CS4		mm		mm	Medium	0	pF	
CS5		mm		mm	Medium	0	pF	
CS6		mm		mm	Medium	0	pF	
CS7		mm		mm	Medium	0	pF	
CS8		mm		mm	Medium	0	pF	
CS9		mm		mm	Medium	0	pF	
Total No of buttons	0	Nos						

Table B: Power calculator

Button Scan Rate offset	506	ms
Design optimization	Response Time	
Noise Immunity level	High	
Approximate Button Scan Rate value	541	ms
Average number of button touch per hour	50	
Average button touch time	500	ms
CS0 Debounce	1	
CS1 - CS9 Debounce	1	
Average Buzzer ON time	0	ms
Average Button Touch LED Effects time	1000	ms
Standby Mode LED Brightness	Disabled	
Current consumption calculation factor	Typical	
Sleep Current	0.00952	mA
Active Current	3.4	mA
Average Current without Finger	0	mA
Average Current with Finger	0	mA
Actual average current consumption	0	mA
Actual average current consumption per butt	0	mA

Table C: Response time calculator

CS0 First button press	576	ms
CS0 Consecutive button press	70	ms
CS1-CS9 First button press	576	ms
CS1-CS9 Consecutive button press	70	ms

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

Note: The power values given here are for the worst case, the actual power values will be lower.

Inputs

- Button diameter and trace length of CS0—CS9 as designed in layout
- Sensitivity of CS0—CS9
- Button Scan Rate offset
- Design optimization
- Noise immunity level
- CS0 Debounce
- CS1—CS9 Debounce
- Average number of button touch per hour
- Average button touch time
- Average Buzzer ON time
- Average Button Touch LED Effects time
- Standby Mode LED Brightness
- Current consumption calculation factor

Outputs

- C_P for each button (confirms whether the C_P values are within the specified range of 5—40 pF)
- Current consumption per button
- Button response time

3.3.4 Design Validation

After you have built and tested the prototype board, use the EZ-Click Customizer Tool to capture the raw count, noise count, and C_P for all buttons (See EZ-Click User Guide). You can use this information and the design validation sheet to validate the design, as detailed in Design Validation .

Table A shows the various design parameter values, taken from the previous sheets, so you do not need to enter any data in this sheet. This sheet provides a pass/fail grade for the prototype board. If your design fails, you can redesign your system by entering new values in Table A, and you will receive further recommendations and results. If your design passes, leave blank the “New value” column in Table A.

Table B shows the button sensitivity values, taken from the C_P, Power Consumption and Response Time Calculator Sheet. If your design fails, you can redesign your button sensitivity by entering the new values. If your design passes, you can leave blank the “New value” column in Table B.

Figure 3-31. Design Validation

Design Validation

Table A: Actual Design values

Input Parameters	Initial value	New value	Units
Overlay Thickness (in mm)	1.5		mm
Dielectric constant, overlay	4		farad/m
Capacitance of trace per inch in	2		pF
Button Scan Rate offset	506	506	ms
Design Optimization	Response Time	Response Time	
Noise Immunity Level	High	High	
Button Scan Rate Value	541	541	ms
Average number of button touch	50	50	
Average button touch time	500	500	ms
Average Buzzer ON time	0	0	ms
Average Button Touch LED Effect	1000	1000	ms
Standby Mode LED Brightness	Disabled	Disabled	
Current consumption calculatio	Typical	Typical	
No of buttons	0	0	Nos
CS0 Button diameter actual			mm
CS1 Button diameter actual			mm
CS2 Button diameter actual			mm
CS3 Button diameter actual			mm
CS4 Button diameter actual			mm
CS5 Button diameter actual			mm
CS6 Button diameter actual			mm
CS7 Button diameter actual			mm
CS8 Button diameter actual			mm
CS9 Button diameter actual			mm

Table B: Button Sensitivity

Button	Initial value	New value
CS0	Medium	
CS1	Medium	
CS2	Medium	
CS3	Medium	
CS4	Medium	
CS5	Medium	
CS6	Medium	
CS7	Medium	
CS8	Medium	
CS9	Medium	

Table C: Reference values

Overlay Material	Dielectric constant
Plastic	2.8
Plexi glass	2.6-3.5
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, 64mil,FR4	2
copper trace , flex, 2 layer	8

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

For Table A: The Initial values of "Input Parameters" are the ones you have entered in the previous sheets. If your design passes, leave the "New value" column blank. If your design fails, enter the New values for the

Table D: Power consumption, Button diameter actuals

Sensor	Values taken from I2C						Average Current		Improvement Recommendations				
	Noise		Cp		Raw				Minimum			Maximum	
CS0		counts		pF	0	counts	0	mA	0	mm		0	mm
CS1		counts		pF	0	counts	0	mA	0	mm		0	mm
CS2		counts		pF	0	counts	0	mA	0	mm		0	mm
CS3		counts		pF	0	counts	0	mA	0	mm		0	mm
CS4		counts		pF	0	counts	0	mA	0	mm		0	mm
CS5		counts		pF	0	counts	0	mA	0	mm		0	mm
CS6		counts		pF	0	counts	0	mA	0	mm		0	mm
CS7		counts		pF	0	counts	0	mA	0	mm		0	mm
CS8		counts		pF	0	counts	0	mA	0	mm		0	mm
CS9		counts		pF	0	counts	0	mA	0	mm		0	mm
			Actual average current consumption				0	mA					

Note: While logging debug data for this sheet, make sure there is no finger present on the sensors for the log duration

To use the EZ-Click Customizer Tool to enter data into Table D, follow these steps:

1. Power-on the device and connect it to your computer using the USB-I²C Bridge (CY3240-I2USB Bridge). Refer to AN2397 – CapSense Data Viewing Tools for (USB-I²C Bridge) (CY3240-I2USB Bridge) details.

2. Open the EZ-Click Customizer Tool and create a new project. Select Cypress device CY8CMBR2110. Select the port you are using from the Port selection window and click **Connect**.
3. Go to Device Config tab and select the number of buttons in your design. Assign the CapSense pins to the corresponding buttons if required. Set the finger threshold or select Automatic Threshold.
4. Go to CapSense output tab and select Button Specific Output view.
5. Select the button whose CapSense output you want to see. Select the “Raw Count vs Baseline” graph.
6. Observe the raw count graph and note the average Raw Counts for 300 samples. Also note the Button C_P.
7. Calculate Noise Counts based on the following equation:
   Noise Counts = Maximum Raw Counts - Minimum Raw Counts (for 300 samples)
8. Enter this data in Table D to find the current consumption values and determine if your design is ready for mass production.

Inputs

- Raw Counts
- Noise Counts
- Button C_P
- If the design fails, note the following:
  - New overlay thickness, overlay material permittivity, button diameter for each individual button, and trace capacitance
  - CSx sensitivity

Outputs

- Current consumption per button
- Design change recommendations. The Design Toolbox makes recommendations based on the actual values from the design if the button size or trace lengths are outside of best design practices.

If the button board does not pass, the Design Toolbox will offer recommendations to guide you to a passing outcome. You can change four areas to remedy a failing design: button size, trace length, overlay material, and overlay thickness. Changing the button size or trace length requires a board spin, while changing the overlay material, thickness, or both, may result in a passing design. The best solution depends on where your design is in the development cycle as well as your end-system requirements.

3.4 Configuring the CY8CMBR2110

CY8CMBR2110 can be configured using one of the following methods:

1. EZ-Click Customizer Tool
2. Configuring the Device using a Host Processor
3. Third-party Programmer

The general procedure to configure the CY8CMBR2110 device is listed in steps. These procedures are common for all the configuration methods. The EZ-Click Customizer Tool takes care of this procedure automatically but the host processor must follow these procedures:

1. Change the device mode to LED Configuration mode.
2. Wait 55 ms.
3. Write to all of the configuration registers in the LED Configuration mode.
4. Wait 55 ms.
5. Change the device mode to Device Configuration mode.
6. Wait 55 ms.
7. Write to all of the configuration registers in the Device Configuration mode.
8. Calculate the checksum and enter it in the “Checksum_MSB” (0x1E) and “Checksum_LSB” (0x1F) registers (in the Device Configuration mode).

Checksum: The checksum is the sum of the values of the registers (0x01—0x1F) in the LED Configuration mode and the registers (0x01—0x1D) in the Device Configuration mode. The checksum also includes the values of any reserved register bits. The host should not write to these bits and should add 0 for any such bit, when calculating the checksum.

Checksum_Flash_xxx registers (in Operating mode) indicate the checksum stored in the flash.

Checksum_RAM_xxx registers (in Operating mode) indicate the checksum calculated by the device for the current configuration and stored in the RAM.

9. Wait 55 ms.
10. Read the "Checksum matched" bit in the Host_Mode register (in Device Configuration mode), and verify that it is set to 1. If this bit is not set, restart at step one and reconfigure the device. The host should keep a backup of the configuration data if this is needed.

    "Checksum matched" bit: The CY8CMBR2110 calculates the checksum and compares that with the Checksum register value entered by the host. If both the values match, the "Checksum matched" bit in the Host_Mode register is set to 1. If the values do not match, it indicates a possible I^2C write error, and this bit is cleared to 0. The host can read the Checksum_RAM_xxx register (in Operating mode) to get the device calculated checksum.

11. If the "Checksum matched" bit is set to 1, then set the "Save to Flash" bit in the Host_mode register.

    Save to Flash: On a "save to flash", the following sequence is executed:

    (i) The device copies the 64-byte data (in LED Configuration mode and Device Configuration mode) to the flash.

    (ii) A software reset is done.

    (iii) After the software reset, the device mode is Operating mode.

    Any configuration changes are not applicable unless you save to flash. A "save to flash" is useful when the device has to be configured only once for all future operations. During a save to flash, the device's power supply must be stable, with V_{DD} fluctuations limited to ±5%.

12. After a "save to flash", wait for (T_{SAVE_FLASH} + Device initialization) time. T_{SAVE_FLASH} is mentioned in the Flash Write Time Specifications in the CY8CMBR2110 Datasheet. The device initialization time is 350 ms if the Noise Immunity is "Normal" or 1000 ms if the Noise Immunity is "High".
13. Read the "Factory defaults loaded" bit in Device_Stat register (in Operating mode).

    Factory Defaults Loaded: On every reset, the device loads the RAM with the flash content and verifies the RAM checksum with the flash checksum to ensure there is no flash corruption. If the checksum differs, then the device identifies it as a flash corruption, loads the factory defaults value in the RAM, and sets the "Factory defaults loaded" bit. This resets any register values previously changed by the host. Factory default values for each register are given in the Register Map.

    If the factory defaults are loaded, the I^2C address of the device also changes from the current address, set by the host, to the default address, 37h. The host must use the default I^2C address on the I^2C bus to communicate with the CY8CMBR2110 after factory defaults are loaded.

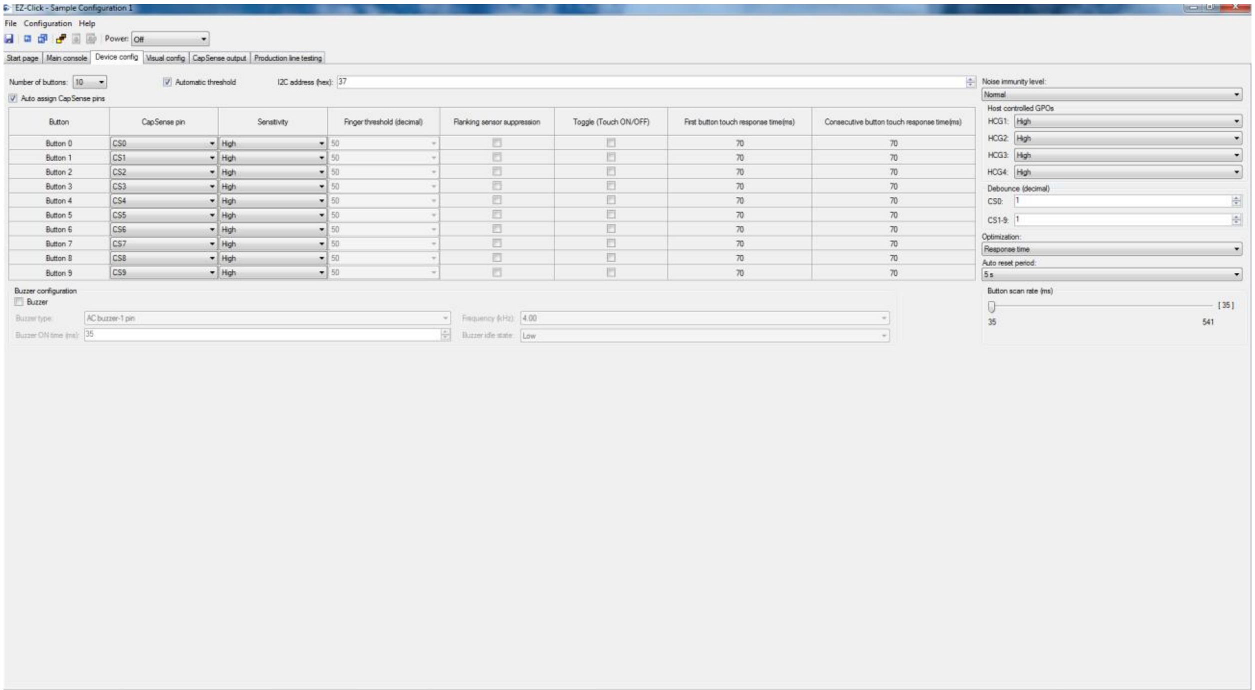
14. If the "Factory defaults loaded" bit is set, then the flash is corrupted, and the host needs to reconfigure the device from step one. If this bit is clear, device configuration is successful.

Note The details of different modes and registers referred to in these steps are available in the CY8CMBR2110 Datasheet.

3.4.1 EZ-Click Customizer Tool

The EZ-Click Customizer Tool is a simple and intuitive graphical user interface used to configure the device. It takes all the required parameters and configures the device using an I^2C interface.

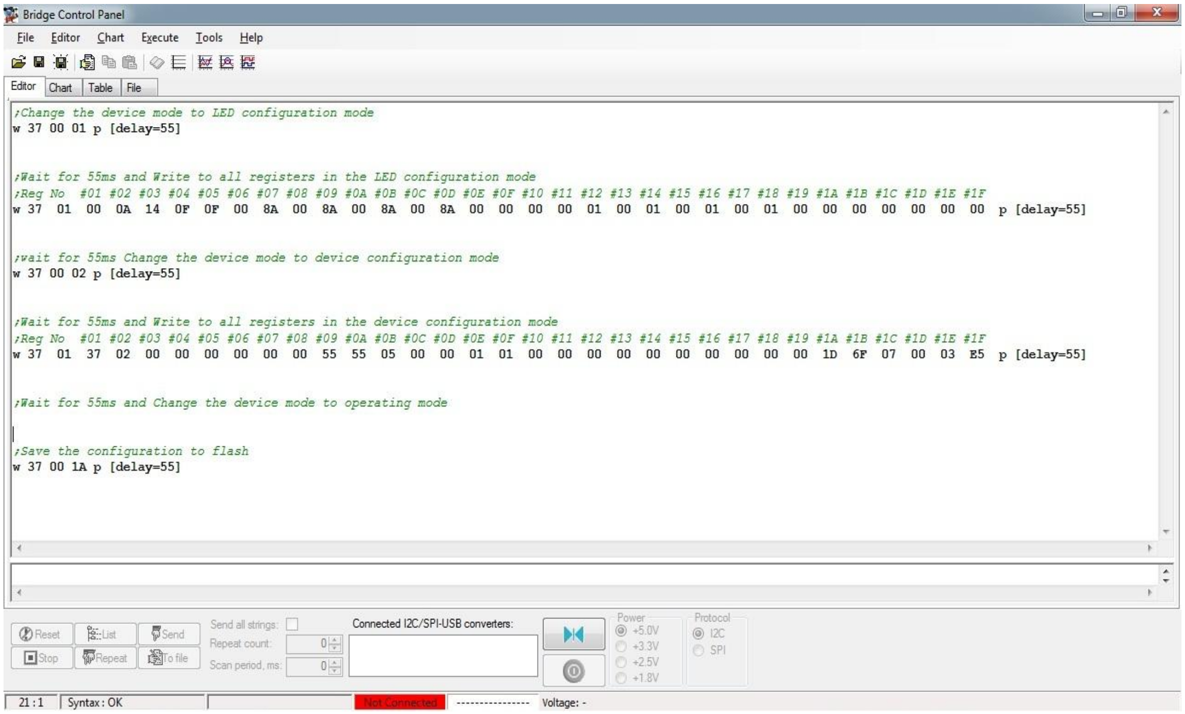
Figure 3-32: EZ-Click Customizer Tool



The EZ-Click Customizer Tool displays real-time CapSense data from the device. You can see both button-specific and parameter-specific data, including CapSense button status, C_P, Raw Counts, Finger threshold, and SNR. The tool can be used for production line testing because it displays System Diagnostics results and CapSense button SNR, and indicates whether the SNR meets your requirements. For more information on this tool, refer to the EZ-Click User Guide.

You can save the configuration and use it on a different sample. You can also use the tool to generate a configuration file, including the required I^2C instructions, and use it to configure the device. To do this, open the configuration file in Bridge Control Panel (refer to AN2397 - CapSense Data Viewing Tools to learn more about Bridge Control Panel) and send the commands to the device over the USB-I^2C Bridge (CY3240-I2USB Bridge). Figure 3-33 shows an example configuration file.

Figure 3-33. Example Configuration File Generated by the EZ-Click Customizer Tool



3.4.2 Configuring the Device using a Host Processor

To configure the CY8CMBR2110 device using a Host processor, there is a comprehensive list of APIs and these APIs are to be called from the Host processor in a specific order. These APIs use I^2C communication to configure the device features, read CapSense data, drive host control GPOs, perform production line tests, configure power consumption settings, and so on. You can download the source code from http://www.cypress.com/?rID=74590.

The advantages of using a Host processor to configure the CY8CMBR2110 device are as follows:

- In-system configuration - no need to take the device (chip) out of the board
- Run time configuration - modifying the features dynamically by a host processor

The APIs are primarily divided as high-level APIs and low-level APIs. High-level APIs are hardware (platform) independent and work on any host processor. The low-level APIs are developed for the CY8C29466-24PXI device, and it is hardware (platform) dependent. If you have a different host processor in your application, you need to modify the low-level API firmware.

3.4.2.1 High-Level APIs

High-level APIs can be used to enable or disable Button Touch LED Brightness, set Finger Threshold parameters, configure scan rate, change I^2C address, and many other functions.

High-level APIs contain code to read or write the appropriate register of the CY8CMBR2110 and calculate the checksum of the configurations. They call low-level APIs that are host processor specific and implement the physical I^2C communication between the host processor and the device.

The high-level API header file (High_Level_API.h) contains function prototypes for all of the high-level APIs. This header file needs to be included in the required .C file when configuring the CY8CMBR2110 device. High-level APIs use the macros defined in High_Level_API.h for internal configuration. You must not change the macros.

For example:

```
#define   I2C_CFG_REG    (0x01)
```

3.4.2.2 Low-Level APIs

Low-level APIs are used in the host processor to enable physical I^2C communication with the device. The low-level APIs provided here use the PSoC I2CHW User Module to perform read and write operations. You may need to modify the low-level API code depending on how you implement I^2C protocol.

The low-level API header file (Low_Level_API.h) contains function prototypes for the low-level APIs and macros used by the low-level APIs. The macros are mainly used for I^2C communication and the software delay routine. These macros are defined for the CY8C29466-24PXI device. You need to change the definitions to work with your host I^2C implementation.

For example, if the CY8CMBR2110 device NACKs, the I2CHW User Module in CY8C29466-24PXI (PSoC1) returns 0x00. Therefore, the macro **I2C_NACK** is defined as 0x00. If you are using a different host processor that returns a different value when it NACKs, you need to modify **I2C_NACK** to match.

The software delay API is required to provide a delay equal to the Button Scan Rate. This delay is required after every write instruction. If you wish to implement this delay using a hardware resource (timer), you can disable the software delay routine by clearing the corresponding macro as described in Table 3-7.

Macros that do not depend on the host controller are listed in Table 3-6. Macros that you may need to change based on the host controller you are using are listed in Table 3-7.

Table 3-6: Macros Not Dependent on the Host Controller

Macro Name	Usage
FLASH_WRITE_TIME	The amount of time it takes the CY8CMBR2110 device to properly save the data after a save to flash command is issued
TOTAL_BUTTON_COUNT	The maximum number of buttons in the CY8CMBR2110 device
FACTORY_DEFAULT_CHECKSUM	The factory default checksum of the CY8CMBR2110 device
DEFAULT_SLAVE_ADDRESS	The factory default I^2C address of the CY8CMBR2110 device
DELAY_CONST	Used to calculate number of iterations required for the software delay
SLAVE_NACK	Used to clear the I^2C flag, when the CY8CMBR2110 device NACKs
SLAVE_ACK	Used to set the I^2C flag, when the CY8CMBR2110 device ACKs
SLAVE_BUF_PTR	Used to set the host I^2C buffer pointer to the specific register address on the register map

Table 3-7: Macros Dependent on the Host Controller

Macro Name	Usage
I2C_WRITE_COMPLETE	Checks if the I2C write operation to the CY8CMBR2110 device is complete. The I2CHW User Module returns 0x50 when the write operation is complete.
NACK_RETRY_LIMIT	Defines the number of times the host processor retries when the CY8CMBR2110 device NACKs. The typical value is 20. You may change this value to work with your application.
DELAY_ROUTINE_USED	Used to enable/disable the software delay routine. A value of 1 enables the software delay, while 0 disables it. **If you are using a hardware resource to implement the delay, you should disable the software delay routine.** **Note** The software delay routine is a blocking code. It stalls the CPU for a definite time.
I2C_NACK	Used to check if the CY8CMBR2110 device NACKed the current I^2C operation. The I2CHW User Module returns 0x00 when the write/read operation is NACKed.
I2C_READ_COMPLETE	Checks if the I^2C write operation to the CY8CMBR2110 device is complete. The I2CHW User Module returns 0x15 when the write operation is complete.
NEW_SLAVE_ADDRESS	The value of the new slave address. If the host changes the default slave address of the CY8CMBR2110 device using the MBR_SetI2CSlaveAddress API, it needs to re-define this macro with the new slave address.
CLOCK_FREQUENCY	The host controller clock frequency in MHz. For the PSoC 1 Host device, the clock frequency is 24 MHz.
MACHINE_CYCLES	The number of machine cycles taken to execute the while loop in the software delay routine. The value of MACHINE_CYCLES is 97 on building with the ImageCraft compiler (refer to MBR_Delay).

3.4.2.3 MBR_WriteBytes

This API initiates an I^2C write operation between the CY8CMBR2110 device and host processor. The function prototype is given in Section 7.2.2.

Note For the write operation, there is a buffer defined in the host. The high-level API passes the buffer to the write API and the buffer is in the form of a BYTE array (refer to Data Types). Upon writing, the first BYTE (byte[0]) holds the base pointer and rest of the bytes (byte[1], byte[2]…) have the data. Because the base pointer is set to "location to be written in the register map of CY8CMBR2110", the write operation begins from that location.

High-level APIs pass the I^2C buffer pointer and the number of bytes to be written. MBR_WriteBytes does the following:

1. Initiates an I^2C write operation to the CY8CMBR2110 device
2. Waits until the transaction ends
3. Checks if the transaction worked properly
4. If the transaction did not work properly, it retries the write operation for up to the value of the macro NACK_RETRY_LIMIT

3.4.2.4 MBR_ReadBytes

This API initiates an I^2C read operation between the CY8CMBR2110 device and host processor. The function prototype is given in section 7.2.2.

Note Upon reading, the host buffer is updated with the required data from the location 0x00 of the device register map as byte[0] will contain the data in location 0x00, byte [1] will have data in location 0x01,etc.The read operation always begin from location 0x00 of all the register maps.

High level APIs pass the I^2C buffer and the number of bytes to be read. MBR_ReadBytes does the following:

1. Gets the I^2C buffer address and the number of bytes that will be read from the device
2. Sets the slave pointer to the location 0x00
3. Initiates an I^2C read operation from the CY8CMBR2110 device
4. Waits until the transaction ends
5. Checks if the transaction worked properly
6. If the transaction did not work properly, it retries the read operation for up to the value of the macro NACK_RETRY_LIMIT

3.4.2.5 MBR_Delay

This API generates a software delay using a while loop that is executed a specified number of times. The function prototype is given in section 7.2.2. The number of loop iterations can be calculated using the following formula:

$$number\ of\ loop\ iterations = \frac{\text{required delay time (ms)} \times \text{clock frequency (MHz)} \times 1000}{\text{machine cycles required to execute the while loop}} \quad \text{Equation 5}$$

You need to calculate the number of machine cycles (total assembly-level instruction cycles) required to execute the while loop in the host machine. For a PSoC 1 host using the ImageCraft Pro compiler, the macro MACHINE_CYCLES is 97. You need to modify this value based on the compiler and host processor you are using.

Note The CPU is completely blocked for the entire delay time.

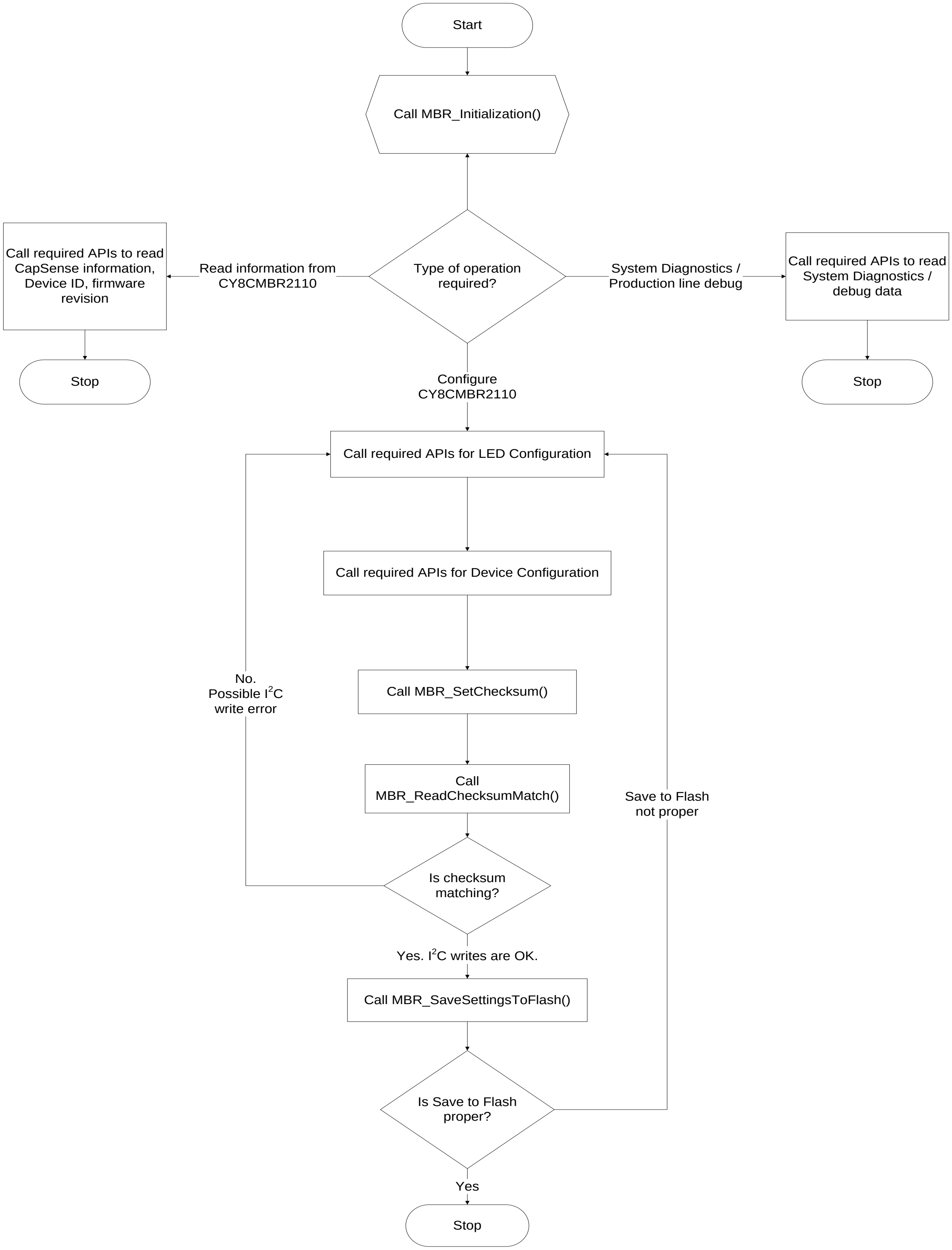
3.4.2.6 Guidelines to Configure the CY8CMBR2110 Device

- The high-level APIs need to be called in a specific order when configuring the CY8CMBR2110 device. Figure 3-34 illustrates the correct order.

- Check your I^2C communication status in the host processor after calling the MBR_Initialization API. This API should be called before any other API call. For example, when the transaction gets ACK, the variable "gbI2CFlag" in low-level APIs is set to 1, otherwise it will be set to zero. You can check this variable for proper transaction. You can also check your own I^2C registers in your host processor for the indication of NACK or ACK.

- Do not switch between register map modes until you have completed configuring all of the features for that register map mode. For example, do not configure one feature in the LED configuration mode, switch to the device configuration mode, and then return to configuring features in the LED configuration mode. Switching between register map modes consumes time. Therefore, configure all the features in the LED configuration mode and then switch to device configuration.
- Pass the correct arguments to the high-level APIs as defined in the section, APIs for CY8CMBR2110 Configuration.
- Since the high-level APIs themselves calculate the checksum of the configurations, you need not take care of checksum calculations.
- Host controlled GPOs must be configured after the save to flash because the save to flash command issues a software reset, which clears the Host controlled GPO configurations.
- LED effects are defined in groups of GPOs (GPO123, GPO456, and GPO789) except for GPO0. The configuration must match for all of the GPOs in a group. For example, do not pass different LED configurations to GPO1 and GPO2. After you configure GPO1, the configuration applies to GPO2 and GPO3 because they share a register and if you again configure different LED effects for GPO2, that will be applicable to GPO1,3.
- When setting the Finger threshold values of the buttons, clear or disable the Automatic Threshold feature using the MBR_SetAdaptiveThreshold API (see High-Level APIs).
- When using LED effects, enable the effect before configuring the features of that effect. For example, enable button touch LED effects and then configure all the features corresponding to button touch LED effects.
- The deep sleep API must be called using the procedure in Deep Sleep Mode.
- Do not configure the LED ON time and also enable Toggle ON/OFF. LED ON time will be disabled if Toggle ON/OFF is enabled.
- Do not configure the LED ON time and also enable Button Touch LED Effects. LED ON time will be disabled if Button Touch LED Effects is enabled.
- Do not enable Toggle ON/OFF and Last Button LED Effect. The Last Button LED Effect will be disabled if Toggle ON/OFF is enabled.
- All of the read APIs such as System Diagnostics, Sensor Current Status, Sensor Latch Status, Sensor SNR, and Debug Data can be called directly without saving to flash.

Figure 3-34: High-Level API Flow Chart



3.4.2.7 Input Header

Inputs.h includes macro definitions for high-level API inputs. Use these macros when passing arguments to high-level APIs. For example, pass the FEATURE_ENABLE macro as an argument when you enable a feature. Some high-level APIs do not have macros for their input. For example, the MBR_SetScanRate() API does not have any macro definition for the input, you need to pass the decimal value of 0 to 31 as the input to the function parameter. Refer to the function prototype of every high-level API in the section, APIs for CY8CMBR2110 Configuration, before passing the parameters. You should not change these macro definitions. These macros help you to pass proper parameters to the high-level APIs.

Note The header of every high-level API also lists all of the possible macros that can be passed to it as arguments.

3.4.2.8 Data Types

The amount of memory allocated for each data type depends upon the complier. Data types char, int, and long are type-defined and used by the high-level APIs to configure the CY8CMBR2110 device. The data types are as follows:

- unsigned char type-defined to BOOL
- unsigned char type-defined to BYTE
- unsigned int type-defined to WORD
- unsigned long type-defined to DWORD
- signed char type-defined to CHAR
- signed int type-defined to INT
- signed long type-defined to LONG

These values are based on the assumption that char, int, and long data types take 8, 16, and 32 bits of memory respectively. If these assumptions are not valid for your host complier, modify the type-definitions in Low_Level_API.h and High_Level_API.h.

3.4.2.9 Sample Project

The sample project is created to configure the CY8CMBR2110 device using CY8C29466-24PXI (PSoC) as the host device, which can be downloaded from http://www.cypress.com/?rID=74590. This code is implemented with PSoC Designer 5.2 and ImageCraft compiler in CY3210-PSoC-EVAL-kit. The sample code configures the following features:

1. Reads the number of working sensors (number of valid sensors passed the system diagnostics)
2. Enables concurrent power-on LED effects for all the GPOs with 600 ms of ramp-up, ramp-down time
3. Enables a high time of 600 ms with 80% brightness level for GPO0, GPO123, 20% brightness level for GPO456, and 100% for GPO789 on Power-On LED Effects
4. Sets the repeat rate equal to one for the GPO0 on Power-On LED Effect
5. Configures AC-1 pin Buzzer in LOW idle state with 4-KHz buzzer frequency and 200 ms of buzzer duration
6. Sets the debounce value to 100 (response time for consecutive button touches to 1225 ms) for the CS0 button
7. Sets sensitivity value of 2 (Medium) for the CS0 button
8. Enables toggle feature for button CS0 button
9. Enables the FSS feature for all the buttons
10. Writes the checksum calculated by the host to CY8CMBR2110 device
11. Verifies the checksum match condition
12. Save the configurations to the Flash if the checksum match condition is true
13. Sets the HGPO1 state to HIGH

Note HGPO1 is configured to be HIGH after save to flash is complete. On the next reset, HGPO1 is cleared to LOW. If you need to see the Power-On LED Effects, you must give a hardware reset to the device, which clears the HGPO1.

3.4.3 Third-party Programmer

To configure the large number of devices, Cypress recommends a third-party vendor to perform automated programming on the devices. For this, you must give the hex file of your configuration, generated by EZ-Click Customizer Tool, to Hilo systems (a third-party programmer).

Contact http://www.hilosystems.com.tw/en/index.aspx for further information.

3.5 CY8CMBR2110 Reset

The CY8CMBR2110 can be reset using hardware or software.

3.5.1 Hardware Reset

On a hardware reset, the LED Configuration mode and Device Configuration mode register values are loaded from the flash into the RAM. All of the device blocks are initialized, System Diagnostics are performed, and an initial 5 ms pulse is sent out on any GPOx associated with a failing CSx. This is done within 350 ms if Noise Immunity is “Normal” or 1000 ms if Noise Immunity is “High”. If Power-On LED Effects are enabled, they are then seen on all the remaining GPOs. After the LED Effects, the device is in Operating mode, and normal operation begins.

Hardware reset is done by toggling power on the CY8CMBR2110 pins using the power supply or XRES.

3.5.1.1 Power Reset

For a power reset, turn off the external power supply to the device’s V_{DD} line, ensuring that V_{DD} drops below 100 mV, and then turn power back on. On a power reset, a high-going pulse of 16 ms is seen on the HostControlGPO1 pin.

3.5.1.2 XRES Reset

For a XRES reset, pull the device’s XRES pin HIGH and then LOW. On a XRES reset, a pulse is not seen on HostControlGPO1 pin.

3.5.2 Software Reset

Software reset is done by writing a 1 to the “Software Reset” bit in the Host_Mode register (in Operating mode). On a software reset, the LED Configuration mode and Device Configuration mode register values are loaded from the flash into the RAM. The device auto-clears the “Software Reset” bit, and all of the device blocks are initialized. This is done within 350 ms if Noise Immunity is “Normal” or 1000 ms if Noise Immunity is “High”. The device is in Operating mode, and normal operation begins. No System Diagnostics are performed, and Power-On LED Effects do not occur. If the user has configured the device for Power-On LED Effects and saved the settings to flash, a hardware reset must be done to see the Power-On LED Effects.

4. Electrical and Mechanical Design Considerations

When designing a capacitive touch sense technology into your application, it is crucial to remember that the CapSense device exists within a larger framework. Careful attention to every detail, including PCB layout, user interface, and end-user operating environment, leads to robust and reliable system performance. For in-depth information, refer to Getting Started with CapSense.

4.1 Overlay Selection

In CapSense Schematic Design, Equation 1 describes finger capacitance:

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

Where:

ε_0 = Free space permittivity

ε_r = Dielectric constant of overlay

A = Area of finger and button pad overlap

D = Overlay thickness

To increase the CapSense signal strength, choose an overlay material with a higher dielectric constant, decrease the overlay thickness, and increase the button diameter. The Design Toolbox helps you design a robust and reliable CY8CMBR2110 solution, as discussed in the chapter CapSense Schematic Design.

Table 4-1. Overlay Material Dielectric Strength

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

Conductive material cannot be used as an overlay because it interferes with the electric field pattern. Therefore, do not use paint containing metal particles.

Bonding Overlay to PCB

Because the dielectric constant of air is very low, an air gap between the overlay and the button degrades the performance of the button. To eliminate the gap, use a nonconductive adhesive to bond the overlay to the CapSense

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

4.2 ESD Protection

Robust ESD tolerance is a natural byproduct of thoughtful system design. By considering how the contact discharge occurs in your end product, particularly in your user interface, you can withstand an 18-kV discharge event without damaging the CapSense controller.

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

4.2.1 Prevent

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

4.2.2 Redirect

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

4.2.3 Clamp

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

The recommended series resistance value is 560 Ω.

A more effective method is to put special-purpose ESD protection devices on the vulnerable traces. Note that ESD protection devices for CapSense need to be low in capacitance. Table 4-2 lists devices recommended for use with CapSense controllers.

Table 4-2. Low-Capacitance ESD Protection Devices Recommended for CapSense

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

4.3 Electromagnetic Compatibility (EMC) Considerations

4.3.1 Radiated Interference

Radiated electrical energy can influence system measurements and the operation of the processor core. The interference enters the CY8CMBR2110 chip at the PCB level, through CapSense button traces and any other digital or analog inputs. The layout guidelines for minimizing the effects of RF interference follow:

- **Ground plane**: provide a ground plane on the PCB.
- **Series resistor**: place series resistors within 10 mm of the CapSense controller pins.
  - The recommended series resistance for CapSense input lines is 560 Ω.
- **Trace length**: Minimize trace length whenever possible.
- **Current loop area**: Minimize the return path for current. To reduce the impact of parasitic capacitance, hatched ground is given within 1 cm of the buttons and traces, instead of solid fill.
- **RF source location**: Partition systems with noise sources, such as LCD inverters and switched-mode power supplies (SMPS), to keep the interference separated from CapSense inputs. Shielding the power supply is another common technique to prevent interference.

4.3.2 Conducted Immunity and Emissions

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

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

4.4 PCB Layout Guidelines

The Design Toolbox will help you design a robust CY8CMBR2110 CapSense PCB layout, as discussed in the General Layout Guidelines.

If your design uses the GPOs to sink current to the CapSense controller, and there is a lot of noise in the CapSense system, use series resistors on all of the GPOs to limit sink current. Sink current limit is determined by the maximum button C_P in your design at 5 V, as show in Table 4-3.

Table 4-3. GPO Sink Current Limit for Low Output Voltage

Button C_P Range	Sink Current Limit per GPO	Sink Current Limit for Device
5 pF ≤ C_P ≤ 12 pF	25 mA	120 mA
12 pF ≤ C_P ≤ 21 pF	20 mA	20 mA
21 pF ≤ C_P ≤ 40 pF	6 mA	6 mA

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

5. Low-Power Design Considerations

5.1 System Design Recommendations

Cypress's CY8CMBR2110 is designed to meet the low-power requirements of battery-powered applications.

To minimize power consumption, take these steps:

- Ground all unused CapSense inputs
- Minimize C_P using the design guidelines in Getting Started with CapSense
- Reduce supply voltage
- Reduce the sensitivity of CSx buttons, refer to Sensitivity Control
- Configure the design to be power consumption-optimized, refer to Button Scan Rate
- Use "High" noise immunity level only if required, refer to Noise Immunity
- Use a higher Button Scan Rate or Deep Sleep operating mode, refer to Button Scan Rate

5.2 Calculating Average Power

The Design Toolbox automates the power optimization calculations described in this section. The average power consumed by the CY8CMBR2110 is determined by calculating the parameters below:

- Button scan rate, T_R
- Scan time, T_S
- Average current in a NO TOUCH state, I_{AVE_NT}
- Average current in a TOUCH state, I_{AVE_T}
- Percentage of active time, P
- Average use current, I_{AVE_U}
- Average current, I_{AVE}
- Average power, P_{AVE}

5.2.1 Button Scan Rate (T_R)

You control the button scan rate through the Register Map settings in the CY8CMBR2110. Based on the register value, an offset is obtained and added to a constant to get the actual button scan rate. The range of the offset value is 0—506 ms.

$$T_R = Button\ Scan\ Rate\ Constant + Button\ Scan\ Rate\ offset \quad \text{Equation 6}$$

Table 3-5 shows how to determine the Button Scan Rate constant.

5.2.1.1 Response Time

Response time is the minimum time the button CSx should be touched for the device to detect as valid button touch and produce a signal on GPOx.

Response times are calculated using the following equation:

Equation 7

If Noise Immunity is "Normal":

$$RT_{CBT} = Button\ Scan\ Rate\ constant + \left[Button\ Scan\ Rate\ constant \times \left\{Round_{down}\left((Debounce - 1)/3\right) + 1\right\}\right]$$

$$RT_{FBT} = Button\ Scan\ Rate + \left[Button\ Scan\ Rate\ constant \times \left\{Round_{down}\left((Debounce - 1)/3\right) + 1\right\}\right]$$

If Noise Immunity is "High":

$$RT_{CBT} = Button\ Scan\ Rate\ constant + [Button\ Scan\ Rate\ constant \times Debounce]$$

$$RT_{FBT} = Button\ Scan\ Rate + [Button\ Scan\ Rate\ constant \times Debounce]$$

Where:

RT_{CBT} = response time for consecutive button touch after first button touch

RT_{FBT} = response time for first button touch

Debounce for CS1—CS9 = 1—255

Debounce for CS0 = 1—255

$Round_{down}$ is the greatest integer less than or equal to ((Debounce – 1)/3)

If you need to change your design configuration from "Normal" Noise Immunity to "High" Noise Immunity, reduce the debounce value to maintain the Response Time.

5.2.2 Scan Time (T_S)

To calculate approximate scan time, use the following equation:

Equation 8

When Noise Immunity is “Normal”:

$$T_S = [0.375\ ms \times (K_{CS0} + K_{CS1} + K_{CS2} + \cdots + K_{CS9})] + T_{FW}$$

When Noise Immunity is “High”:

$$T_S = [0.375\ ms \times (K_{CS0} + K_{CS1} + K_{CS2} + \cdots + K_{CS9}) \times 3] + T_{FW}$$

Where:

K_{CSX} = button sensitivity constant for CSx, from Table 5-1.

T_{FW} = Firmware execution time, from Table 5-2.

Table 5-1. Button Sensitivity Constant

CSx Sensitivity (pF)	C_P (pF)[7]	Button Sensitivity Constant (K)
High	Button connected to GND	0
	5 pF ≤ C_P ≤ 10 pF	1
	10 pF < C_P ≤ 22 pF	2
	22 pF < C_P ≤ 40 pF	4
Medium	Button connected to GND	0
	5 pF ≤ C_P ≤ 18 pF	1
	18 pF < C_P ≤ 38 pF	2
	38 pF < C_P ≤ 40 pF	4
Low	Button connected to GND	0
	5 pF ≤ C_P ≤ 12 pF	0.5
	12 pF < C_P ≤ 26 pF	1
	26 pF < C_P ≤ 40 pF	2

Table 5-2. Average Current Parameters

Parameter	Typical	Maximum
T_{FW}	6.00 ms	6.50 ms
T_S	From Equation 7	+5% from TYP value
T_R	From Equation 5	+10% from TYP value
I_{SLEEP}	9.52 µA	14.2 µA
I_{ACTIVE}	3.4 mA	4.00 mA

[7] C_P limits are approximate and can have ±2 pF variation

5.2.3 Average Current in NO TOUCH State (I_{AVE_NT})

$$I_{AVE\_NT} = \left(\frac{T_R - T_S}{T_R} \times I_{SLEEP}\right) + \left(\frac{T_S}{T_R} \times I_{ACTIVE}\right) \qquad \text{Equation 9}$$

Where:

T_R = button scan rate

T_S = scan time

I_{SLEEP} = current consumed by CY8CMBR2110 during Low Power Sleep mode, from Table 5-2.

I_{ACTIVE} = current consumed by CY8CMBR2110 during active operation, from Table 5-2.

If Standby Mode LED Brightness is enabled:

$$I_{AVE\_NT} = I_{ACTIVE}$$

5.2.4 Average Current in TOUCH State (I_{AVE_T})

$$I_{AVE\_T} = \left(\frac{C_{BS} - T_S}{C_{BS}} \times I_{SLEEP}\right) + \left(\frac{T_S}{C_{BS}} \times I_{ACTIVE}\right) \qquad \text{Equation 10}$$

Where:

T_S = Scan time

C_{BS} = Button scan rate constant, from Table 3-5.

I_{SLEEP} = current consumed by CY8CMBR2110 during Low Power Sleep mode, from Table 5-2.

I_{ACTIVE} = current consumed by CY8CMBR2110 during active operation, from Table 5-2.

If Standby Mode LED Brightness is enabled:

$$I_{AVE\_T} = I_{ACTIVE}$$

5.2.5 Percentage of Active Time (P)

When you touch a button, the device's active time is calculated (in ms) using the number of button touches per hour and the maximum of the following three values:

1. Average button touch time
2. Average Buzzer ON time
3. Average Button Touch LED Effects time

Equation 11

$$Active\ time = Max(Button\ touch\ time, Buzzer\ ON\ time, Button\ Touch\ LED\ Effects\ time) \times (Number\ of\ button\ touches\ per\ hour)$$

The percentage of active time is:

$$P = \frac{Active\ time}{(3600 \times 1000)} \times 100 \qquad \text{Equation 12}$$

Using this method to find P assumes that each button touch occurs after any Buzzer Signal Output or Button Touch LED Effects have finished and no other button is touched. If this is not the case, using this value for P will result in a higher power consumption calculation than the actual value.

5.2.6 Average Use Current (I_{AVE_U})

$$I_{AVE\_U} = \left(\frac{100 - P}{100} \times I_{AVE\_NT}\right) + \left(\frac{P}{100} \times I_{AVE\_T}\right) \qquad \text{Equation 13}$$

Where:

P = percentage of active time

I_{AVG_NT} = average current in the NO TOUCH state

I_{AVG_T} = average current in the TOUCH state

5.2.7 Average Current (I_{AVE})

$$I_{AVE} = \left[I_{AVE\_U} \times \left(\frac{T_{SA}}{T_{DS}+T_{SA}}\right)\right] + 0.1\ \mu A \qquad \text{Equation 14}$$

Where:

T_{SA} = time device is not in deep sleep mode

T_{DS} = time device is in deep sleep mode

5.2.8 Average Power (P_{AVE})

$$P_{AVE} = V_{DD} \times I_{AVE} \qquad \text{Equation 15}$$

Where:

I_{AVE} = average current

V_{DD} = supply voltage

5.2.9 Example Calculation

As an example of how to calculate average power, consider a CapSense user interface with eight well-designed buttons and the following parameters:

- C_P for all eight buttons is between 10—20 pF
- Sensitivity of each button is high
- Design is response time-optimized
- Noise Immunity is "Normal"
- Button scan rate offset is set to 506 ms
- Standby Mode LED Brightness is disabled
- Typical current consumption values measured

The button scan rate constant can be obtained from Table 3-5:

$$C_{BS} = 35\ ms$$

The button scan rate is calculated using Equation 5:

$$T_R = 35 + 506 = 541\ ms$$

The scan time can be calculated using Equation 7, with the button sensitivity constant obtained from Table 5-1, and the typical value for firmware execution time from Table 5-2.

$$T_S = [0.375 \times (8 \times 2)] + 6.00 = 12.0\ ms$$

The average current in NO TOUCH state is calculated as follows using Equation 8 and the maximum values for I_{SLEEP} and I_{ACTIVE} from Table 5-2.

$$I_{AVE\_NT} = \left(\frac{541-12}{541} \times 9.52\ \mu A\right) + \left(\frac{12}{541} \times 3.4\ mA\right) = 84.7\ \mu A$$

The average current in TOUCH state is calculated as follows using Equation 9:

$$I_{AVE\_T} = \left(\frac{35-12}{35} \times 9.52\ \mu A\right) + \left(\frac{12}{35} \times 3.4\ mA\right) = 1172\ \mu A$$

To calculate the active time using Equation 10, assume that a button is touched once a minute (60 button touches per hour). On average, button touch time is 1000 ms, Button Touch LED Effects time is 3000 ms, and there are no buzzer outputs.

$Active\ time = 3000ms \times 60 = 180\ s$

The percentage of active time is calculated using Equation 11:

$$P = \frac{180}{3600} \times 100 = 5\%$$

The average current consumption of the design is calculated as follows using Equation 12:

$$I_{AVE\_U} = \left(\frac{100-5}{100} \times 84.7\ \mu A\right) + \left(\frac{5}{100} \times 1172\ \mu A\right) = 139.1\ \mu A$$

Assuming this design does not utilize deep sleep mode and that it operates at 1.71 V, the average power is calculated as follows using Equation 14:

$$P_{AVE} = 1.71 \times 139.1\ \mu A = 237.8\ \mu W$$

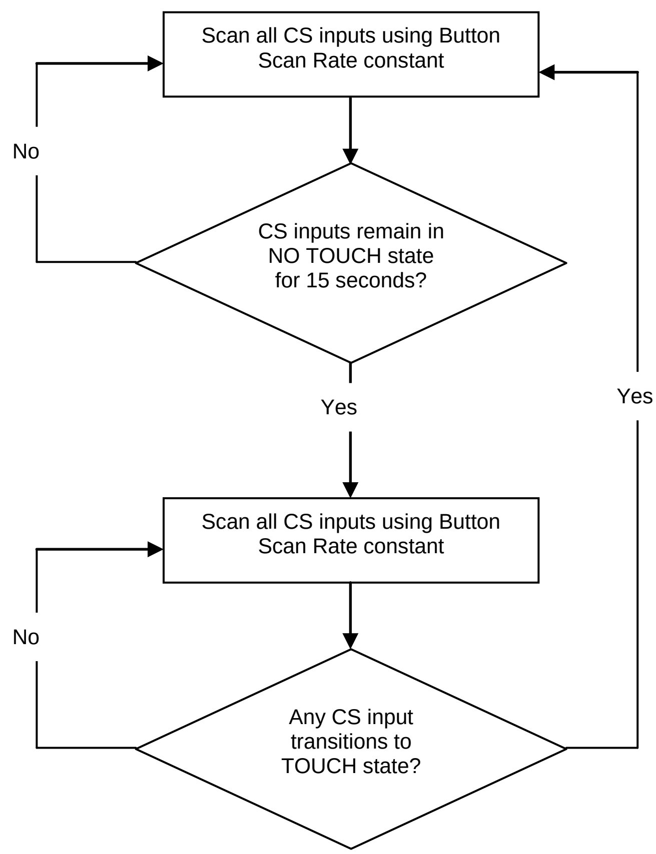
5.3 Sleep Modes

Cypress's CY8CMBR2110 can be configured to operate in either low-power sleep mode or deep sleep mode. These modes reduce the power consumption of the device.

5.3.1 Low-Power Sleep Mode

The behavior of the CY8CMBR2110 controller in Low-Power Sleep mode is described in Figure 5-1.

Figure 5-1. Low-Power Sleep Mode



5.3.2 Deep Sleep Mode

If you use the CY8CMBR2110 in a system with a host processor, the Attention/Sleep line can operate the device in Deep Sleep mode. For CY8CMBR2110 to go into Deep Sleep mode, follow these steps:

1. Pull the Attention/Sleep line low
2. Set the “Deep Sleep” bit in Host_Mode register (in Operating Mode) to 1
3. Wait for 50 ms
4. Pull the Attention/Sleep pin high

All communication is suspended. In Deep Sleep mode, the device consumes ~0.1-µA. After the device enters Deep Sleep mode, the Deep Sleep bit is automatically cleared. To wake up, the Attention/Sleep line is pulled low by the host. After it wakes up, the CY8CMBR2110 goes into active mode. The host processor can then pull the Attention/Sleep pin high to put the device into Low-Power Sleep mode. After waking up from Deep Sleep mode, the device takes some time before the button scanning restarts, this period is called re-initialization. During this time, any button touch is not reported. Re-initialization takes 20 ms if Noise Immunity is "Normal" or 50 ms if Noise Immunity is "High".

6. Resources

6.1 Website

Visit Cypress's CapSense Controllers website to access all of the reference material discussed in this section.

Find a variety of technical resources on the CY8CMBR2110 web page.

6.2 Datasheet

The datasheet for the CapSense CY8CMBR2110 device is available at www.cypress.com.

- CY8CMBR2110

6.3 Design Toolbox

The interactive Design Toolbox will enable you to design a robust and reliable CY8CMBR2110 CapSense solution.

6.4 EZ-Click™ Customizer Tool

The interactive EZ-Click Customizer Tool will help you configure your CY8CMBR2110 CapSense solution.

6.5 Design Support

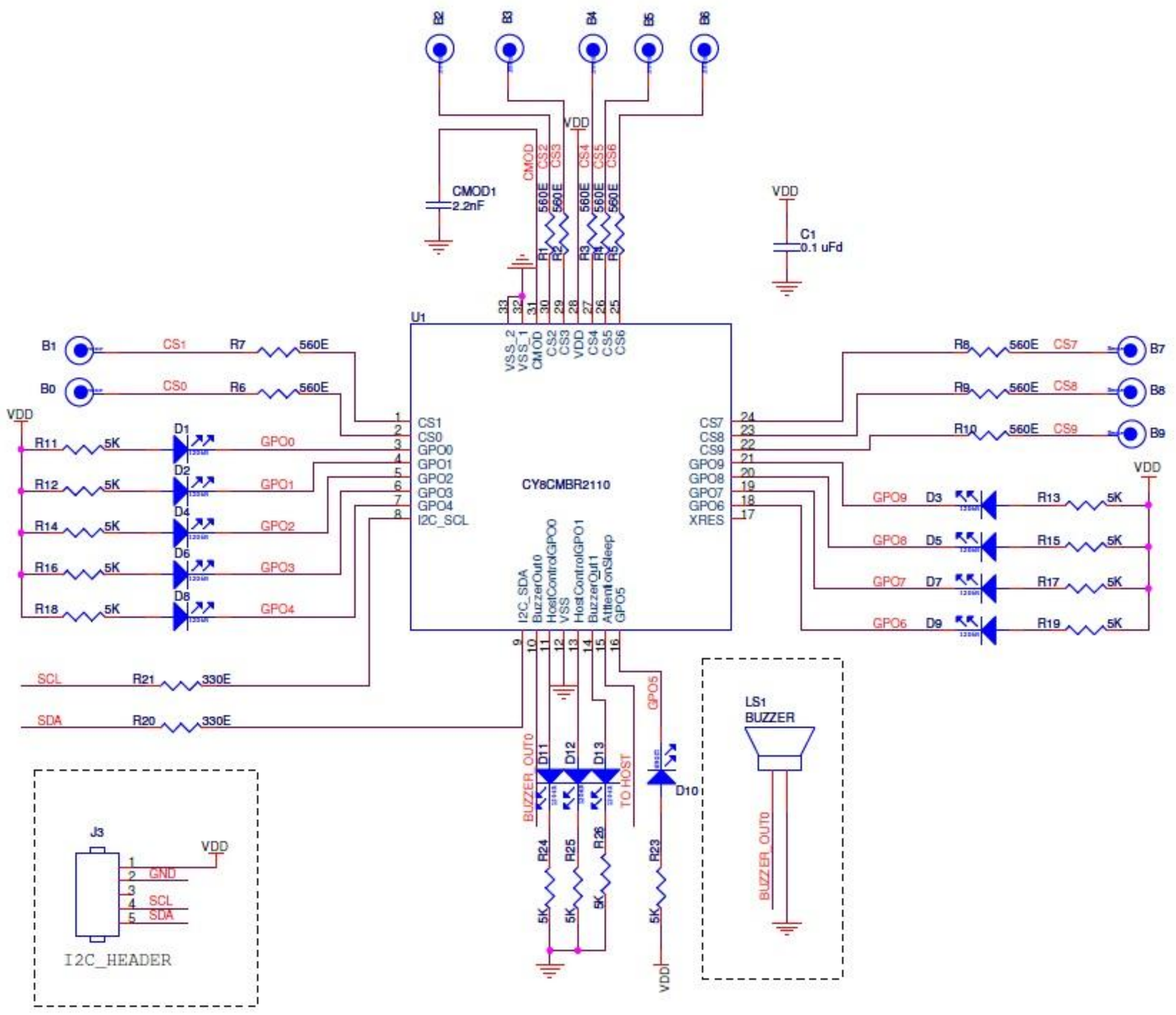
To ensure the success of your CapSense solutions, Cypress has a variety of design support channels.

- 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.

7. Appendix

7.1 Schematic Example

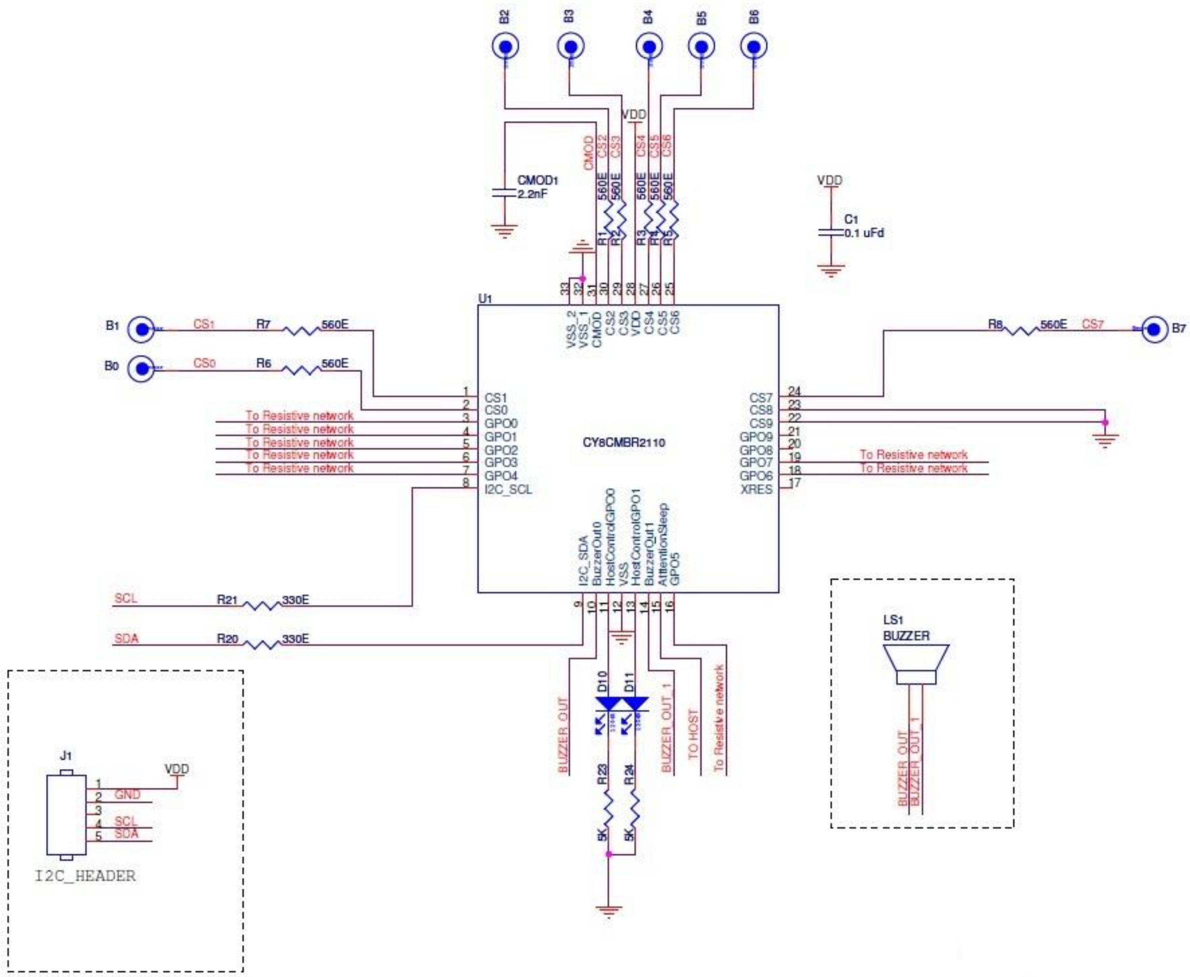
7.1.1 Schematic 1: Ten Buttons with Ten GPOs



In Schematic 1: Ten Buttons with Ten GPOs, CY8CMBR2110 is configured as follows:

- CS0—CS9 pins: 560 Ω to CapSense buttons
  - Ten CapSense buttons (CS0—CS9)
- GPO0—GPO9 pins: LED and 5 kΩ to V_{DD}
  - CapSense buttons driving ten LEDs (GPO0—GPO9)
- CMOD pin: 2.2 nF to Ground
  - Modulating capacitor
- XRES pin: Floating
  - For external reset
- BuzzerOut0 pin: To buzzer
  - AC buzzer (1-pin)
  - Buzzer second pin to Ground
- BuzzerOut1 pin: LED and 5 kΩ to Ground
  - Used as Host Controlled GPO
- HostControlGPO0, HostControlGPO1: LED and 5 kΩ to Ground
  - Two Host Controlled GPOs
- I2C_SDA, I2C_SCL pins: 330 Ω to I^2C Header
  - For I^2C communication
- Attention/Sleep pin: To Host
  - For controlling I^2C communication, power consumption, and device operating mode

7.1.2 Schematic 2: Eight Buttons with Analog Voltage Output



In Schematic 2: Eight Buttons with Analog Voltage Output, CY8CMBR2110 is configured as follows:

- CS0—CS7 pins: 560 Ω to CapSense buttons; CS8, CS9 pins: Ground
  - ☐ Eight CapSense buttons (CS0 – CS7)
  - ☐ CS8 and CS9 buttons not used in design
- GPO0—GPO7 pins: To external resistive network
  - ☐ Eight GPOs (GPO0 – GPO7) used for Analog Voltage Output
  - ☐ GPO8 and GPO9 not used in design
- CMOD pin: 2.2 nF to Ground
  - ☐ Modulating capacitor
- XRES pin: Floating
  - ☐ For external reset
- BuzzerOut0, BuzzerOut1 pins: To AC Buzzer
  - ☐ AC 2-pin Buzzer
- HostControlGPO0, HostControlGPO1 pins: LED and 5 kΩ to Ground

    - Two Host Controlled GPOs
- I2C_SDA, I2C_SCL pins: 330 Ω to I^{2}C Header
    - For I^{2}C communication
- Attention/Sleep pin: To Host
    - For controlling I^{2}C communication, power consumption, and device operating mode

7.2 APIs for CY8CMBR2110 Configuration

The following table lists 72 high-level APIs and 3 low-level APIs, which will be used at host processor (I^2C master) to configure CY8CMBR2110 (I^2C slave) through I^2C interface. These high-level APIs are independent of platforms and can be used on any host processor. The appropriate inputs are defined as macros for many high-level APIs in inputs.h.

Low-level APIs are platform dependent and used in the host processor to enable physical I^2C communication with the device. These low-level APIs are developed for the PSoC 1 host device; therefore, you may need to modify the low-level API code depending on your host processor. The sample project created using these APIs is explained in section 3.4.2.9.

7.2.1 High-Level APIs

	Prototype	**void MBR_Initialization(void);**		
1	**Description**	Initializes global variables used by the high-level APIs. You must call this API first before calling any other API.		
	Parameters	None		
	Return	None		
	Example	MBR_Initialization();		
2	**Prototype**	**void MBR_SetCustomData(BYTE bCustomData);**		
	Description	Writes the data given by the user to the custom data storage register in Device Configuration mode. User should call **MBR_SaveSettingsToFlash** API to store the data permanently.		
	Parameters	**Name**	**Description**	**Possible values**
		bCustomData	Date to be written in Custom register	0 to 255
	Return	None		
	Example	MBR_SetCustomData(200); Value 200 will be stored in Custom Data Storage registers.		
3	**Prototype**	**void MBR_IssueSWReset(void);**		
	Description	Issues software reset to the CY8CMBR2110 device. Refer to Software Reset		
	Parameters	None		
	Return	None		
	Example	MBR_IssueSWReset();		
4	**Prototype**	**WORD MBR_ReadFlashChecksum(void);**		
	Description	Reads the checksum stored in the flash of CY8CMBR2110 device.		
	Parameters	None		
	Return	Flash checksum of CY8CMBR2110 device		
	Example	MBR_ReadFlashChecksum();		
5	**Prototype**	**WORD MBR_ReadRAMChecksum(void);**		
	Description	Reads the checksum stored in the RAM of CY8CMBR2110 device.		
	Parameters	None		
	Return	RAM checksum of CY8CMBR2110 device		

	Example	MBR_ReadRAMChecksum();		
6	**Prototype**	**void MBR_SetChecksum(void);**		
	Description	Writes the checksum calculated by the host in to the CY8CMBR2110 device. Host itself calculates the check sum of the configurations		
	Parameters	None		
	Return	None		
	Example	MBR_SetChecksum();		
7	**Prototype**	**BYTE MBR_ReadChecksumMatch(void);**		
	Description	Checks whether RAM checksum calculated by CY8CMBR2110 device is same as that of the checksum entered by the Host.		
	Parameters	None		
	Return	0 or 1 0 for checksum mismatch 1 for checksum match		
	Example	MBR_ReadChecksumMatch();		
8	**Prototype**	**BYTE MBR_SaveSettingsToFlash(void);**		
	Description	Saves the current configuration of the CY8CMBR2110 device to flash (refer to the Configuring the CY8CMBR2110)		
	Parameters	None		
	Return	0 or 1 0 - save to flash is not successful 1 - save to flash is successful		
	Example	MBR_SaveSettingsToFlash();		
9	**Prototype**	**BYTE MBR_SettingsLoaded(void);**		
	Description	Indicates whether the factory default setting or the user configured setting is loaded		
	Parameters	None		
	Return	0 or 1 0 - user configured settings 1 - factory default settings		
	Example	MBR_SettingsLoaded( );		
10	**Prototype**	**void MBR_LoadFactoryDefaults(void);**		
	Description	Loads the factory default settings configuration in to the RAM of CY8CMBR2110 device.		
	Parameters	None		
	Return	None		
	Example	MBR_LoadFactoryDefaults();		
11	**Prototype**	**void MBR_ReadConfigData(BYTE abConfigData[]);**		
	Description	Loads the LED configuration and device configuration data from the CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**

		abConfigData	Pointer to 64-byte array to hold all the configuration data	
	Return	None		
	Example	MBR_ReadConfigData(abConfigData); abConfigData is a pointer to the 64-byte array abConfigData[64].The array is updated with all the configuration data.		
12	**Prototype**	**void MBR_LEDEffectsBreathing(BYTE bGPO, BYTE bBreath);**		
	Description	Enables or disables the button touch LED effects breathing. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		bBreath	Enable/disable the breathing effect	0 or 1 0 - disable breathing 1 - enable breathing
	Return	None		
	Example	MBR_LEDEffectsBreathing(GPO4, FEATURE_ENABLE); GPO4 is a macro with value 4, FEATURE_ENABLE is a macro with value 1 (inputs.h).		
13	**Prototype**	**void MBR_LEDEffectsRepeatRate(BYTE bGPO, BYTE bRepeatRate, BYTE bPwrOnOrBtnTch);**		
	Description	Sets the repeat rate of the LED effect for selected GPOs. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		bRepeatRate	Repeat rate of the LED effect	0 to 7 0 - Repeat rate of 0 1 - Repeat rate of 1 2 - Repeat rate of 2 3 - Repeat rate of 4 4 - Repeat rate of 6 5 - Repeat rate of 10 6 - Repeat rate of 15 7 - Repeat rate of 20
		bPwrOnOrBtnTch	Power on or button touch LED effects	1 or 2 1 - power on LED 2 - Button touch LED
	Return	None		
	Example	MBR_LEDEffectsRepeatRate(GPO1,REPEAT_RATE_20,POWER_ON_LED_EFFECTS); GPO1, REPEAT_RATE_20,POWER_ON_LED_EFFECTS are macros with values 1, 7, 1 (inputs.h).		
14	**Prototype**	**void MBR_LEDEffectsLowBrightness(BYTE bGPO, BYTE bLowBright, BYTE bPwrOnOrBtnTch);**		

	Description	Sets the LED low brightness for the GPOs. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		bLowBright	Low brightness level as per register map	0 to 7 0 - Low brightness 0% 1 - Low brightness 10% …………………………… …………………………... 7 – Low brightness 100%
		bPwrOnOrBtnTch	Power on or button touch LED effects	1 or 2 1 - power on LED 2 - Button touch LED
	Return	None		
	Example	MBR_LEDEffectsLowBrightness(GPO2, LOW_BRIGHT_80, BTN_TOUCH_LED_EFFECTS); GPO2, LOW_BRIGHT_80, BTN_TOUCH_LED_EFFECTS are macros with values 2 ,6, 2 (inputs.h).		
15	**Prototype**	**void MBR_LEDEffectsHighBrightness(BYTE bGPO, BYTE bHighBright, BYTE bPwrOnOrBtnTch);**		
	Description	Sets the LED high brightness for the GPOs. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		bHighBright	High brightness level as per the register map	0 to 7 0 - High brightness 100% 1 - High brightness 90% …………………………… …………………………... 7 - High brightness 0%
		bPwrOnOrBtnTch	Power on or button touch LED effects	1 or 2 1 - power on LED 2 - Button touch LED
	Return	None		
	Example	MBR_LEDEffectsHighBrightness(GPO9, HIGH_BRIGHT_50, BTN_TOUCH_LED_EFFECTS); GPO9, HIGH_BRIGHT_50, BTN_TOUCH_LED_EFFECTS are macros with values 9 ,4, 2 (inputs.h).		
16	**Prototype**	**void MBR_LEDEffectsLowTime(BYTE bGPO, BYTE bLowTime, BYTE bPwrOnOrBtnTch);**		
	Description	Sets the LED low time for GPOs. **Note** For LED effects ,GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9

		bLowTime	Global period register map to the get the low time value	0 to 1 0 - GLOBAL_PERIOD_1 1 - GLOBAL_PERIOD_2
		bPwrOnOrBtnTch	Power on or button touch LED effects	1 or 2 1 - power on LED 2 - Button touch LED
	Return	None		
	Example	MBR_LEDEffectsLowTime(GPO6, GLOBAL_PERIOD_1, BTN_TOUCH_LED_EFFECTS); GPO6, GLOBAL_PERIOD_1, BTN_TOUCH_LED_EFFECTS are macros with values 6, 0, 2 (inputs.h).		
17	**Prototype**	**void MBR_LEDEffectsHighTime(BYTE bGPO, BYTE bHighTime, BYTE bPwrOnOrBtnTch);**		
	Description	Set the LED high time for the GPOs. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		bHighTime	Global period register map to the get the high time value	0 to 1 0 - GLOBAL_PERIOD_1 1 - GLOBAL_PERIOD_2
		bPwrOnOrBtnTch	Power on or button touch LED effects	1 or 2 1 - power on LED 2 - Button touch LED
	Return	None		
	Example	MBR_LEDEffectsHighTime(GPO5, GLOBAL_PERIOD_2, BTN_TOUCH_LED_EFFECTS); GPO5, GLOBAL_PERIOD_2, BTN_TOUCH_LED_EFFECTS are macros with values 5, 1, 2 (inputs.h).		
18	**Prototype**	**void MBR_LEDEffectsRampDown(BYTE bGPO, BYTE bRampDown, BYTE bPwrOnOrBtnTch);**		
	Description	Sets the ramp down time for the GPOs. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		bRampDown	Global period register map to the get the ramp down time value	0 to 3 0 - GLOBAL_PERIOD_1 1 - GLOBAL_PERIOD_2 2 - GLOBAL_PERIOD_3 3 - GLOBAL_PERIOD_4
		bPwrOnOrBtnTch	Power on or button touch LED effects	1 or 2 1 - power on LED 2 - Button touch LED
	Return	None		
	Example	MBR_LEDEffectsRampDown(GPO8, GLOBAL_PERIOD_1, POWER_ON_LED_EFFECTS); GPO8, GLOBAL_PERIOD_1, POWER_ON_LED_EFFECTS are macros with values 8, 0, 1 (inputs.h).		

#				
19	**Prototype**	**void MBR_LEDEffectsRampUp(BYTE bGPO, BYTE bRampUp, BYTE bPwrOnOrBtnTch);**		
	Description	Sets the ramp up time for the GPOs. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789 Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		bRampUp	Global period register map to the get the ramp up time value	0 to 3 0 - GLOBAL_PERIOD_1 1 - GLOBAL_PERIOD_2 2 - GLOBAL_PERIOD_3 3 - GLOBAL_PERIOD_4
		bPwrOnOrBtnTch	Power on or button touch LED effects	1 or 2 1 - power on LED 2 - Button touch LED
	Return	None		
	Example	MBR_LEDEffectsRampUp(GPO8, GLOBAL_PERIOD_1, POWER_ON_LED_EFFECTS) GPO8, GLOBAL_PERIOD_1, POWER_ON_LED_EFFECTS are macros with values 8, 0, 1 (inputs.h).		
20	**Prototype**	**void MBR_PowerONLEDEffectSeq(BYTE bPwrOnSeq);**		
	Description	Sets the power-on LED Effects sequence (concurrent or sequential) .Make sure that you enabled the power on LED effects before calling this API.		
	Parameters	**Name**	**Description**	**Possible values**
		bPwrOnSeq	Type of Power on LED effect sequence	0 or 1 0 - concurrent 1 - sequential
	Return	None		
	Example	MBR_PowerONLEDEffectSeq(POWER_ON_SEQUENTIAL); POWER_ON_SEQUENTIAL is a macro with value 1 (inputs.h).		
21	**Prototype**	**void MBR_PowerONLEDEffects(BYTE bEnable);**		
	Description	Enables or disables power on LED effects.		
	Parameters	**Name**	**Description**	**Possible values**
		bEnable	Enable or disable the effect	0 to 1 0 - disable the effect 1 - enable the effect
	Return	None		
	Example	MBR_PowerONLEDEffects(FEATURE_ENABLE); FEATURE_ENABLE is a macro with value 1 (inputs.h).		
22	**Prototype**	**void MBR_ButtonLEDEffects(BYTE bEnable);**		
	Description	Enables or disables the Button Touch LED Effects		
	Parameters	**Name**	**Description**	**Possible values**
		bEnable	Enable or disable the effect	0 to 1 0 - disable the effect 1 - enable the effect
	Return	None		

	Example	MBR_ButtonLEDEffects(FEATURE_DISABLE); FEATURE_DISABLE is a macro with a value 0. (inputs.h)		
23	**Prototype**	**void MBR_StandbyModeLEDBrightness(BYTE bLEDBrightness);**		
	Description	Sets the standby mode LED Brightness level.		
	Parameters	**Name**	**Description**	**Possible values**
		bLEDBrightness	Standby mode brightness level as per the register map	0 to 3 0 - 0% brightness 1 - 20% brightness 2 - 30% brightness 3 - 50% brightness
	Return	None		
	Example	MBR_StandbyModeLEDBrightness(STDBY_LED_50) ; STDBY_LED_50 is a macro with value 3 (inputs.h)		
24	**Prototype**	**void MBR_LEDEffectLastButton(BYTE bEnable);**		
	Description	Enables or disables LED effects on last button touch feature.		
	Parameters	**Name**	**Description**	**Possible values**
		BYTE bEnable	Enable or disable the effect	0 or 1 0 - disable the effect 1 - enable the effect
	Return	None		
	Example	MBR_LEDEffectLastButton(FEATURE_DISABLE); FEATURE_DISABLE is a macro with value 0 (inputs.h)		
25	**Prototype**	**void MBR_SetGlobalPeriod(BYTE bPeriodReg, WORD wPeriodValue);**		
	Description	Sets the period value in the global period register		
	Parameters	**Name**	**Description**	**Possible values**
		bPeriodReg	Global period register map	0 to 3 0 - GLOBAL_PERIOD_1 1 - GLOBAL_PERIOD_2 2 - GLOBAL_PERIOD_3 3 - GLOBAL_PERIOD_4
		wPeriodValue	Global period value in (ms)	0 to 1600
	Return	None		
	Example	MBR_SetGlobalPeriod(GLOBAL_PERIOD_1,600) GLOBAL_PERIOD_1 is a macro with value 0 (inputs.h).		
26	**Prototype**	**void MBR_SetAllGlobalPeriods(WORD awPeriodValue[]);**		
	Description	Sets the period values of all the global period registers.		
	Parameters	**Name**	**Description**	**Possible values**
		awPeriodValue	Pointer to a 4-word array holding the period values in (ms)	0 to 1600
	Return	None		
	Example	MBR_SetAllGlobalPeriods(wTestBuffer); wTestBuffe is the base pointer of the 4-word array wTestBuffer[4].		

27	**Prototype**	**void MBR_SetAllLEDParameters(BYTE bGPO, BYTE abParam[]);**		
	Description	Sets all the LED effects parameters for any GPO. **Note** For LED effects, GPOs are grouped as: GPO0, GPO123, GPO456, GPO789. Configuring one GPO in a group also configures the other GPOs in that group.		
	Parameters	**Name**	**Description**	**Possible values**
		bGPO	GPO number	0 to 9
		awPeriodValue	Pointer to 9 byte array holding configuring parameters	byte[0] - Power On or Button Touch effects byte[1] - High brightness level byte[2] - Low brightness level byte[3] - Ramp up time mapping to global period registers byte[4] - Ramp down time mapping to global period registers byte[5] - High time mapping to global period registers byte[6] - Low time mapping to global period registers byte[7] - Repeat rate byte[8] - Breathing effect enable/disable
	Return	None		
	Example	MBR_SetAllLEDParameters(GPO2, bconfig); bconfig is a pointer of the 9-byte array bconfig[9].		
28	**Prototype**	**BYTE MBR_ReadDeviceID(void);**		
	Description	Reads the CY8CMBR2110 device ID.		
	Parameters	None		
	Return	Device ID of CY8CMBR2110.The ID is “0xA1”.		
	Example	MBR_ReadDeviceID();		
29	**Prototype**	**BYTE MBR_ReadFWRevision(void);**		
	Description	Reads the slave device firmware revision.		
	Parameters	None		
	Return	Device firmware revision		
	Example	MBR_ReadFWRevision();		
30	**Prototype**	**void MBR_SetDebugSensorNumber(BYTE bSensor);**		
	Description	Sets the sensor number for which the debug data has to be sent.		
	Parameters	**Name**	**Description**	**Possible values**
		bSensor	Sensor number	0 to 9
	Return	None		
	Example	MBR_SetDebugSensorNumber(CS0); CS0 is a macro with value 0 (inputs.h).		

31	**Prototype**	**void MBR_SetDebugDataParameter(BYTE bParameter);**		
	Description	Set the type of parameter to be sent in debug data out.		
	Parameters	**Name**	**Description**	**Possible values**
		bParameter	Type of parameter	0 to 4 0 - C_P 1 - Raw counts 2 - Difference counts 3 - Raw counts ,baseline 4 - All parameters(C_P, Raw count, difference count, base line, SNR)
	Return	None		
	Example	MBR_SetDebugDataParameter(DEBUG_PARAM_CP); DEBUG_PARAM_CP is a macro with value 0 (inputs.h).		
32	**Prototype**	**void MBR_ReadDebugData(BYTE abDebugData[]);**		
	Description	Reads the debug data of the selected parameter from the debug data register map		
	Parameters	**Name**	**Description**	**Possible values**
		abDebugData	Pointer to 25-byte array to hold the debug data	
	Return	None		
	Example	MBR_ReadDebugData(bgetdata); bgetdata is a pointer to the 25-byte array bgetdata[25]. The array is updated with the debug data.		
33	**Prototype**	**void MBR_SetBuzzer(BYTE bEnable);**		
	Description	Enables or disables the audio feedback (buzzer).		
	Parameters	**Name**	**Description**	**Possible values**
		bEnable	Enable or disable buzzer	0 or 1 0 – enable 1 – disable
	Return	None		
	Example	MBR_SetBuzzer(FEATURE_ENABLE) FEATURE_ENABLE is a macro with value 1 (inputs.h).		
34	**Prototype**	**void MBR_SetBuzzerPins(BYTE bBuzzerPins);**		
	Description	Sets the number of pins for the buzzer.		
	Parameters	**Name**	**Description**	**Possible values**
		bBuzzerPins	Number of buzzer output pins	0 or 1 0 - 1 pin buzzer 1 - 2 pin buzzer
	Return	None		
	Example	MBR_SetBuzzerPins(BUZZER_AC_2_PIN); BUZZER_AC_2_PIN is a macro with value 1 (inputs.h).		

35	**Prototype**	**void MBR_SetBuzzerIdleState(BYTE bIdleState);**		
	Description	Sets the idle state of the buzzer pins.		
	Parameters	**Name**	**Description**	**Possible values**
		bIdleState	Buzzer idle state	0 or 1 0 - LOW 1 - HIGH
	Return	None		
	Example	MBR_SetBuzzerIdleState(BUZZER_IDLE_HIGH); BUZZER_IDLE_HIGH is a macro with value 1 (inputs.h)		
36	**Prototype**	**void MBR_SetBuzzerFrequency(BYTE bFrequency);**		
	Description	Sets the output frequency for the buzzer output.		
	Parameters	**Name**	**Description**	**Possible values**
		bFrequency	Buzzer output frequency	1 to 7 1 - 4000 Hz 2 - 2670 Hz 3 - 2000 Hz 4 - 1600 Hz 5 - 1330 Hz 6 - 1140 Hz 7 - 1000 Hz
	Return	None		
	Example	**MBR_SetBuzzerFrequency (**BUZZER_FREQ_1000); BUZZER_FREQ_1000 is a macro with value 7 (inputs.h).		
37	**Prototype**	**void MBR_SetBuzzerOutputDuration(WORD wDuration);**		
	Description	Sets the duration of the buzzer output.		
	Parameters	**Name**	**Description**	**Possible values**
		wDuration	Buzzer output duration in millisecond (ms)	(0 to 127) * Button Scan Rate
	Return	None		
	Example	MBR_SetBuzzerOutputDuration(1000);		
38	**Prototype**	**void MBR_SetAllBuzzerParameters(BYTE bEnable, BYTE bParameters[], WORD wOutputDuration);**		
	Description	Sets all the buzzer parameters of the CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bEnable	Enable or disable buzzer	0 or 1 0 - disable 1 - enable
		bParameters	Pointer to the 3-byte array holding the required inputs	Byte [0] – number of buzzer pins Byte [1] – buzzer idle state Byte [2] – buzzer output frequency
		wOutputDuration	Duration of buzzer output in ms	(0 to 127) * Button Scan Rate

	Return	None		
	Example	MBR_SetAllBuzzerParameters(FEATURE_ENABLE , bbuzzconfig ,1000); FEATURE_ENABLE is a macro with value 1 (inputs.h), bbuzzconfig is a pointer to a 3-byte array containing buzzer pins, idle state, and frequency details (1000 is the buzzer duration).		
39	**Prototype**	**void MBR_SetI2CSlaveAddress (BYTE bNewSlaveAddress);**		
	Description	Sets the I^{2}C address of the CY8CMBR2110 device. The default address is '37h'.		
	Parameters	**Name**	**Description**	**Possible values**
		bNewSlaveAddress	New address value to the device	0x00 to 0x7F
	Return	None		
	Example	MBR_SetI2CSlaveAddress(50);		
40	**Prototype**	**void MBR_SetAdaptiveThreshold(BYTE bSetRest);**		
	Description	Enables or disables automatic threshold feature of the CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bSetRest	Enable or disable automatic threshold feature	0 or 1 0 – to disable 1 – to enable
	Return	None		
	Example	MBR_SetAdaptiveThreshold(FEATURE_ENABLE); FEATURE_ENABLE is a macro with value 1 (inputs.h).		
41	**Prototype**	**void MBR_SetSensitivity(BYTE bButtonNumber, BYTE bButtonSensitivityLevel);**		
	Description	Sets the sensitivity value for a button.		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
		bButtonSensitivityLevel	Sensitivity level of the button	1 to 3 1 – high sensitivity 2 – medium sensitivity 3 – low sensitivity
	Return	None		
	Example	MBR_SetSensitivity (CS9, SENSITIVITY_MEDIUM); CS9 and SENSITIVITY_MEDIUM are macros with values 9 and 2 (inputs .h).		
42	**Prototype**	**void MBR_SetSensitivityAll(BYTE bsensitivity[]);**		
	Description	Sets the sensitivity value of all the buttons.		
	Parameters	**Name**	**Description**	**Possible values**
		bsensitivity	Pointer to the 10-byte array holding the sensitivity level for all the buttons	1 to 3 1 – high sensitivity 2 – medium sensitivity 3 – low sensitivity
	Return	None		
	Example	test_MBR_SetSensitivityAll(bBuffer); bBuffer is a pointer to the 10-byte array bBuffer[10] that holds the sensitivity values for all the buttons (the first byte corresponds to the button number 0,……..tenth byte corresponds to button 9).		

43	**Prototype**	**void MBR_SetDebounce(BYTE bButtonNumber, BYTE bDebouncevalue);**		
	Description	Sets the debounce level of buttons. Button numbers 1 to 9 are configured with the same value. They cannot be configured individually.		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 1 0 – for button number 0 1 – for button number (1-9)
		bDebouncevalue	Debounce value for the buttons	1 to 255
	Return	None		
	Example	MBR_SetDebounce(DEBOUNCE_FOR_CS0, 200); DEBOUNCE _FOR_CS0 is a macro with value 0 (inputs.h).		
44	**Prototype**	**void MBR_SetFingerThreshold(BYTE bButtonNumber, BYTE bFingerthreshold);**		
	Description	Sets the finger threshold level for a button.		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
		bFingerthreshold	Finger threshold level	0 to 15
	Return	None		
	Example	MBR_SetFingerThreshold(CS3, FINGER_THRESHOLD_180); CS3 and FINGER_THRESHOLD_180 are macros with values 3 and 10 (inputs.h).		
45	**Prototype**	**void MBR_SetFingerThresholdAll(BYTE bFingerthreshold[]);**		
	Description	Sets the finger threshold level for all the sensors.		
	Parameters	**Name**	**Description**	**Possible values**
		bFingerthreshold	Pointer to the 10-byte array holding the finger threshold values	0 to 15
	Return	None		
	Example	MBR_SetFingerThresholdAll(Buffer); Buffer is a pointer to the 10-byte array Buffer[10] holding the finger threshold level for all the buttons. The first byte corresponds to the button number 0,……..tenth byte corresponds to button 9).		
46	**Prototype**	**BYTE MBR_ReadSensorStatus(BYTE bButtonNumber);**		
	Description	Reads the current status of a button (to check current state of button touch).		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
	Return	0 or 1 0 – button is not pressed (OFF) 1 – button is pressed (ON)		
	Example	MBR_ReadSensorStatus(CS5); CS5 is a macro with value 5 (inputs.h).		
47	**Prototype**	**WORD MBR_ReadSensorStatusAll(void);**		
	Description	Reads the current status of all the buttons.		

	Return	Two bytes with the current status of all the sensors. LSB is buttons 0 to 7 First two bits of MSB are buttons 8 and 9 For example, 0x0301 indicates buttons 0, 8, 9 are touched (ON) and rest of the buttons are not touched (OFF).		
	Example	MBR_ReadSensorStatusAll();		
48	**Prototype**	**WORD MBR_ReadLatchStatusAll(void);**		
	Description	Reads the latched status of all the sensors.		
	Parameters	None		
	Return	Two bytes with the current latched status of all the sensors. LSB is buttons 0 to 7 First two bits of MSB are buttons 8 and 9 For example, 0x0301 indicates buttons 0, 8, 9 were touched (ON) and rest of the buttons were not touched (OFF) before the current I^2C read.		
	Example	MBR_ReadLatchStatusAll();		
49	**Prototype**	**void MBR_EnterDeepSleep(void);**		
	Description	Sets the Deep sleep bit as 1 in operating mode register so device will enter into deep sleep mode. Follow the procedures in Deep Sleep Mode to change the mode to deep sleep mode.		
	Parameters	None		
	Return	None		
	Example	MBR_EnterDeepSleep();		
50	**Prototype**	**void MBR_SetPowerOptimization(BYTE bOptimization);**		
	Description	Sets the power consumption optimized or response time optimized design for the CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bOptimization	Enables power consumption or response time optimization	0 or 1 0 - response time optimization 1 - power consumption optimization
	Return	None		
	Example	MBR_SetPowerOptimization(PWR_CONS_OPT); PWR_CONS_OPT is a macro with value 1 (inputs.h)		
51	**Prototype**	**void MBR_SetScanRate(BYTE bSetscanvalue);**		
	Description	Sets the scan rate of the CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bSetscanvalue	Scan rate values as per the register map	0 to 31 0 – 25 ms 31- 561 ms
	Return	None		
	Example	MBR_SetScanRate(30);		
52	**Prototype**	**void MBR_SetHGPOValue(BYTE bHGPO_Number, BYTE bDriveLogic);**		

	Description	Sets the drive logic of a HGPO.		
	Parameters	**Name**	**Description**	**Possible values**
		bHGPO_Number	Host controlled GPO (HGPO) number	0 to 3 0 - HGPO0 1 - HGPO1 2 - HGPO2 3 - HGPO3
		bDriveLogic	Drive logic level	0 or 1 1 - HIGH 0 - LOW
	Return	None		
	Example	MBR_SetHGPOValue(HOSTGPO_3, HOSTGPO_HIGH); HOSTGPO_3 and HOSTGPO_HIGH are macros with values 3 and 1 (inputs.h).		
53	**Prototype**	**void MBR_SetAllHGPOValue(BYTE bdriveGP0, BYTE bdriveGP1, BYTE bdriveGP2, BYTE bdriveGP3);**		
	Description	Sets the drive logic of all HGPOs.		
	Parameters	**Name**	**Description**	**Possible values**
		bdriveGP0	Drive logic of HGPO0	0 or 1
		bdriveGP1	Drive logic of HGPO1	0 or 1
		bdriveGP2	Drive logic of HGPO2	0 or 1
		bdriveGP3	Drive logic of HGPO3	0 or 1
	Return	None		
	Example	void MBR_SetAllHGPOValue(HOSTGPO_HIGH, HOSTGPO_HIGH, HOSTGPO_LOW, HOSTGPO_LOW); HOSTGPO_HIGH, HOSTGPO_HIGH, HOSTGPO_LOW, and HOSTGPO_LOW are macros with the values 1, 1, 0, 0 (inputs.h).		
54	**Prototype**	**void MBR_AnalogOutput(BYTE bSet_Reset)**		
	Description	Enables or disables analog output voltage feature of the CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bSet_Reset	Set or resets analog output voltage feature	0 or 1
	Return	None		
	Example	MBR_AnalogOutput(FEATURE_ENABLE); FEATURE_ENABLE is a macro with value 1 (inputs.h).		
55	**Prototype**	**void MBR_SetToggle (BYTE bButtonNumber, BYTE fSet_Reset);**		
	Description	Enables or disables the toggle feature for a button of the CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
		fSet_Reset	Enable or disable toggle	0 or 1 0 - disable 1 - enable
	Return	None		

	Example	MBR_SetToggle(CS0,FEATURE_ENABLE); FEATURE_ENABLE is a macro with value 1 (inputs.h).		
56	**Prototype**	**void MBR_SetToggleAll(BYTE afSet_Reset[]);**		
	Description	Enables or disables the toggle feature for all the buttons		
	Parameters	**Name**	**Description**	**Possible values**
		afSet_Reset	Pointer to 2-byte array holding values	Byte[1] - 0x00 to 0xFF Byte[2] - 0x00 to 0x03
	Return	None.		
	Example	MBR_SetToggleAll(Buffer); Buffer is a pointer to the 2-byte array Buffer Buffer[1] holds the values for buttons 0 to 7 The first two bits of Buffer[2] hold the values for buttons 8 and 9. For example, Buffer[1] = 0xFF enables toggle for buttons 0 to 7, and Buffer[2] = 0x01 enables toggle for button 8 and disables toggle for button 9.		
57	**Prototype**	**void MBR_LEDONTime(BYTE bSet_Reset);**		
	Description	Enables or disables LED on time feature of CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bSet_Reset	Enable or disable the feature	0 or 1 0 - disable 1- enable
	Return	None		
	Example	MBR_LEDONTime(FEATURE_DISABLE); FEATURE_DIASBLE is a macro with value 0 (inputs.h).		
58	**Prototype**	**BYTE MBR_ReadValidSensors(void);**		
	Description	Reads the valid sensor/button count Buttons may be disabled due to short to VDD, short to GND, improper C_P value, and improper CMOD value.		
	Parameters	None		
	Return	One byte of data indicating the valid sensor count A return value of 8 indicates eight buttons are valid and two buttons are disabled.		
	Example	MBR_ReadValidSensors();		
59	**Prototype**	**BYTE MBR_ReadFMEAGround(BYTE bButtonNumber);**		
	Description	Reads the system diagnostics data of one button for short to ground (checks if a button is shorted to ground).		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
	Return	0 or 1 0 - button is not shorted to ground 1 - button is shorted to ground		
	Example	MBR_ReadFMEAGround(CS9); CS9 is a macro with value 9 (inputs.h).		
60	**Prototype**	**WORD MBR_ReadFMEAGroundAll(void);**		
	Description	Reads the system diagnostics data of all the buttons for short to ground.		

	Parameters	None		
	Return	Two bytes to indicate which buttons are shorted to ground LSB is buttons 0 to 7 First two bits of MSB are buttons 8 and 9 For example, 0x02F1 indicates buttons 0,4,5,6,7,9 are shorted to ground		
	Example	MBR_ReadFMEAGroundAll();		
61	**Prototype**	**BYTE MBR_ReadFMEAVDD(BYTE bButtonNumber);**		
	Description	Reads the system diagnostics data of one button for short to VDD (checks if a button is shorted to VDD).		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
	Return	0 or 1 0 - button is not shorted to VDD 1 - button is shorted to VDD		
	Example	MBR_ReadFMEAVDD(CS8); CS8 is a macro with value 8 (inputs.h).		
62	**Prototype**	**WORD MBR_ReadFMEAVDDAll(void);**		
	Description	Reads the system diagnostics data of all the buttons for short to VDD.		
	Parameters	None		
	Return	Two bytes to indicate which buttons are shorted to VDD. LSB is buttons 0 to 7 First two bits of MSB are buttons 8 and 9. For example, 0x02F1 indicates buttons 0,4,5,6,7,9 are shorted to VDD.		
	Example	MBR_ReadFMEAVDDAll();		
63	**Prototype**	**WORD MBR_ReadFMEASnsToSnsAll(void);**		
	Description	Reads the system diagnostics data of all the buttons for button to button short.		
	Parameters	None		
	Return	Two bytes to indicate which buttons are shorted to another button. LSB is buttons 0 to 7 First two bits of MSB are buttons 8 and 9. For example, 0x02F1 indicates buttons 0,4,5,6,7,9 are shorted to another button.		
	Example	MBR_ReadFMEASnsToSnsAll();		
64	**Prototype**	**BYTE MBR_ReadFMEASensorCP(BYTE bButtonNumber);**		
	Description	Read the system diagnostics data of a button for high Cp value.		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
	Return	0 or 1 0 - Cp of the button is proper (Cp < 40 pF) 1 - Cp of the button is high (Cp > 40 pF)		
	Example	MBR_ReadFMEASensorCP(CS0); CS0 is a macro with value 0 (inputs.h).		

65	**Prototype**	**WORD MBR_ReadFMEASensorCpAll(void);**		
	Description	Reads the system diagnostics data of all the buttons for high C_P values.		
	Parameters	None		
	Return	Two bytes to indicate which buttons have a high Cp value. LSB is buttons 0 to 7 First two bits of MSB are buttons 8 and 9. For example, 0x02F1 indicates buttons 0,4,5,6,7,9 have Cp > 40 pF.		
	Example	MBR_ReadFMEASensorCpAll();		
66	**Prototype**	**BYTE MBR_ReadFMEACMOD(void);**		
	Description	Reads the system diagnostics of CMOD (checks the CMOD capacitance value).		
	Parameters	None		
	Return	0 - if CMOD is proper within (1- 4) nF 1 - if COMD is above 4 nF 2 - if CMOD is below 1 nF		
	Example	MBR_ReadFMEACMOD();		
67	**Prototype**	**BYTE MBR_ReadSensorSNR(BYTE bButtonNumber);**		
	Description	Reads the SNR value of the button.		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
	Return	One byte to indicate the button SNR. SNR value can range from 0 to 15 for a button		
	Example	MBR_ReadSensorSNR(CS9); CS9 is a macro with value 9 (inputs.h).		
68	**Prototype**	**void MBR_ReadSensorSNRAll(BYTE bSensor_SNR[TOTAL_BUTTON_COUNT]);**		
	Description	Reads the SNR value of all the buttons		
	Parameters	**Name**	**Description**	**Possible values**
		bSensor_SNR[TOTAL_BUTTON_COUNT]	Pointer to 10-byte array to hold the read values from the device	
	Return	None		
	Example	MBR_ReadSensorSNRAll(buffer); buffer is a pointer to a 10-byte array that is updated with the SNR values starting from button 0 (The first byte corresponds to button 0, the second byte corresponds to button 1,…. Tenth byte corresponds to button 9.).		
69	**Prototype**	**void MBR_SetAutoResetTime(BYTE bSet_time);**		
	Description	Sets the auto reset time of all the buttons.		
	Parameters	**Name**	**Description**	**Possible values**
		bSet_time	Auto reset time value	1 - no limit 2 - 5 sec 3 - 20 sec
	Return	None		

	Example	MBR_SetAutoResetTime(AUTO_RESET_20S); AUTO_RESET_20S is a macro with value 3 (inputs.h).		
70	**Prototype**	**void MBR_SetEMC(BYTE bSet_Reset);**		
	Description	Enables or disable EMC feature in CY8CMBR2110 device.		
	Parameters	**Name**	**Description**	**Possible values**
		bSet_Reset	Enable or disable EMC	0 or 1 0 - disable 1 - enable
	Return	None		
	Example	MBR_SetEMC(FEATURE_ENABLE); FEATURE_ENABLE is a macro with value 1 (inputs.h).		
71	**Prototype**	**void MBR_SetFSS(BYTE bButtonNumber, BYTE fSet_Reset);**		
	Description	Enables or disables the FSS (Flanking Sensor suppression) feature for a button.		
	Parameters	**Name**	**Description**	**Possible values**
		bButtonNumber	Button number	0 to 9
		fSet_Reset	Enable or disable feature	0 or 1 0 - disable 1 - enable
	Return	None		
	Example	MBR_SetFSS(CS0,FEATURE_DISABLE); CS0 and FEATURE_DISABLE are macros with values 0 and 1 (inputs.h).		
72	**Prototype**	**void MBR_SetFSSAllSensors(BYTE afSet_Reset[])**		
	Description	Enables or disables the FSS (Flanking Sensor suppression) feature for all the buttons.		
	Parameters	**Name**	**Description**	**Possible values**
		afSet_Reset	Pointer to 2-byte array holding the configuring values	Byte[1] - 0x00 to 0xFF Byte[2] - 0x00 to 0x03
	Return	None		
	Example	MBR_SetFSSAllSensors(Buffer); Buffer is a pointer to 2-byte array Buffer[2]. Buffer[1] holds the values for buttons 0 to 7 The first two bits of Buffer[2] hold the values for buttons 8 and 9 For example: Buffer[1] = 0xFF enables FSS for buttons 0 to 7 , Buffer[2] = 0x01 enables the feature for button 8, Buffer[2] =0x02 enables the feature for button 9 and disables the feature for button 8.		

7.2.2 Low-Level APIs

1	**Prototype**	**void MBR_WriteBytes(BYTE abWriteBuffer[], BYTE bNumberOfBytes)**		
	Description	Write the array of bytes to the CapSense slave device.		
	Parameters	**Name**	**Description**	**Possible values**
		abWriteBuffer	Pointer to the Host I^2C buffer array	1 to 31 bytes
		bNumberOfBytes	Number of bytes to be written	1 to 31
	Return	None		
	Example	MBR_WriteBytes(abHostI2CBuffer, 3);		
2	**Prototype**	**void MBR_ReadBytes(BYTE abReadBuffer[] , BYTE bNumberOfBytes)**		
	Description	Read the array of bytes from the CapSense slave device.		
	Parameters	**Name**	**Description**	**Possible values**
		abReadBuffer	Pointer to Host I^2C buffer array	1 to 32 bytes
		bNumberOfBytes	Number of bytes to be read	1 to 32
	Return	None		
	Example	MBR_ReadBytes(bHostI2CBuffer, 6);		
3	**Prototype**	**void MBR_Delay(WORD wDelayTime)**		
	Description	Implements software delay in milliseconds.		
	Parameters	**Name**	**Description**	**Possible values**
		wDelayTime	Delay time in (ms)	
	Return	None		
	Example	MBR_Delay(100);		

Glossary

AMUXBUS

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

SmartSense™ Auto-Tuning

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

Baseline

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

Button or Button Widget

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

Difference Count

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

Capacitive Sensor

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

CapSense®

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

CapSense Mechanical Button Replacement (MBR)

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

Centroid or Centroid Position

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

Compensation IDAC

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

CSD

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

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

Debounce

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

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

Driven-Shield

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

Electrode

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

Finger Threshold

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

Ganged Sensors

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

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

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

Gesture

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

Guard Sensor

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

Hatch Fill or Hatch Ground or Hatched Ground

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

Hysteresis

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

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

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

Liquid Tolerance

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

Linear Slider

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

Low Baseline Reset

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

Manual-Tuning

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

Matrix Buttons

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

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

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

Modulation Capacitor (CMOD)

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

Modulator Clock

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

Modulation IDAC

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

Mutual-Capacitance

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

Negative Noise Threshold

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

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

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

Noise (CapSense Noise)

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

Noise Threshold

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

Overlay

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

Parasitic Capacitance (C_P)

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

Proximity Sensor

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

Radial Slider

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

Raw Count

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

Refresh Interval

The time between two consecutive scans of a sensor.

Scan Resolution

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

Scan Time

Time taken for completing the scan of a sensor.

Self-Capacitance

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

Sensitivity

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

Sense Clock

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

Sensor

See Capacitive Sensor.

Sensor Auto Reset

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

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

Sensor Ganging

See Ganged Sensors.

Shield Electrode

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

Shield Tank Capacitor (C_{SH})

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

Signal (CapSense Signal)

Difference Count is also called Signal. See Difference Count.

Signal-to-Noise Ratio (SNR)

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

Slider Resolution

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

Touchpad

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

Trackpad

See Touchpad.

Tuning

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

V_{REF}

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

Widget

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

Revision History

Document Revision History

Document Title: AN76000 - CY8CMBR2110 CapSense® Design Guide Document Number: 001-76000			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**	08/01/2012	UDYG	New Design Guide
*A	09/10/2012	UDYG	Updated links to external documents
*B	03/14/2013	SEEE	Updated Section 3.4 and added Section 7.2.
*C	08/27/2013	UDYG/ZINE	Updated SmartSense Auto-Tuning features in Chapter 1. Updated FSS description. Updated screenshots for Figures 3-28, 3-29, 3-30 and 3-31. Added Ez-Click Customizer screenshot
*D	01/14/2015	SSHH	Updated references to USB-I2C Bridge Updated to new template.
*E	01/20/2016	VAIR	Added Glossary.
*F	09/15/2016	DIMA	Updated hyperlink in section 6.5. Updated template
*G	07/13/2017	AESATMP8	Updated logo and Copyright.