**Please note that Cypress is an Infineon Technologies Company.**
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

# CE227719 – CapSense with Multi-Frequency Scan

## Objective

This example demonstrates the use of the multi-frequency scan (MFS) for PSoC® 4 CapSense® applications to prevent false touch detection in the presence of external noise. This code example demonstrates this feature for both CSD (self-capacitance) and CSX (mutual-capacitance) designs.

## Requirements

**Tool:** PSoC Creator™ 4.2, PSoC Programmer, and Bridge Control Panel

**Programming Language:** C (Arm® GCC 5.4.1)

**Associated Parts:** PSoC 4100S Plus

**Related Hardware:** CY8CKIT-149 PSoC 4100S Plus Prototyping Kit

## Overview

This example demonstrates the use of the multi-frequency scan feature of CapSense for PSoC 4 devices. It demonstrates this feature for both CSD (self-capacitance) and CSX (mutual-capacitance) designs.

The code scans all the buttons and sliders on the CY8CKIT-149 and drives the LEDs depending on the button and slider touch status.

The code also sends the CapSense raw data over an $I^2C$ interface to the on-board KitProg, which in turn enables reading the data from CapSense Tuner GUI or Bridge Control Panel. The CapSense Tuner and Bridge control Panel are used to observe the changes when the multi-frequency scan feature is enabled.

This document details the following:

1. The use of the multi-frequency-scan feature to prevent false touch detection in the presence of external noise at a particular frequency.
2. The effects of multi-frequency scan such as scanning at three different sense clock frequencies and increased scan time.
3. A method to test the improvement in touch detection accuracy brought in by MFS, by manually injecting external noise and showcasing that there is no false touch detected when the MFS feature is enabled.

## Pre-requisite

The user of this code example is assumed to be familiar with CapSense technology and basic terms such as raw counts, baseline, and difference counts. If you are new to CapSense, see the application note AN85951 – PSoC® 4 and PSoC 6 MCU CapSense® Design Guide and the code example CE220891 – CapSense with Breathing LED (In PSoC Creator, select **File** > **Code example** > **Device family** > **PSoC 4100S Plus**).

## Hardware Setup

This code example uses the kits default configuration. Refer to the kit user guide to ensure that the kit is configured correctly.

## Software Setup

Ensure that you have all the software tools as mentioned in Requirements section installed.

# When to Use Multi-Frequency Scan?

Multi-frequency scan (MFS) is used to prevent false touch detection in the presence of external noise at a particular frequency.

If MFS is disabled, whenever there is an external noise at the same frequency as the sense clock or a harmonic of the sense clock frequency (in case of CSD), or Tx clock frequency (in case of CSX), raw counts may fluctuate from scan to scan and this may cause the sensor to false-trigger.

When MFS is enabled, each sensor is scanned three times, at three different frequencies called channels. For each sensor, the CapSense component calculates three raw counts and baselines, one for each of these channels, and then calculates the difference-count between each raw count and baseline pair. Finally, a touch is detected on a sensor, based on the median difference-count amongst the three channels of that sensor. This prevents false touch detection if there is noise at the same frequency or harmonic of one of the channels; raw counts produced by other two channels are minimally affected by noise.

## Points to Consider While Using MFS

1. The sensor is scanned three times, each time with a different sense clock frequency (for CSD) or Tx clock frequency (for CSX). The base channel (zero channel) is the nominal frequency as configured in the Component, the second and the third channel frequencies are chosen by the Component at run time. These could be any of the following pairs: +5% and +10% or –5% and +5% or –5% and –10%. The second and third channel frequencies are generated by varying the IMO frequency of the device.
2. Because each sensor is scanned three times, the scan time of each sensor becomes three times compared to the case if MFS is disabled. Figure 1 shows the scan time when the MFS feature is disabled; Figure 2 shows the scan time when the MFS feature is enabled.
3. Enabling this feature increases the RAM and flash usage. See the Component datasheet for more information on memory usage when multi-frequency scan feature is enabled.
4. The `CapSense_IsBusy ()` API will return `CapSense_SW_STS_BUSY`. This means that the previous scanning is not completed, and the hardware block is busy until the Component completes the scan at all the three frequencies. This implies that the scan time has increased.
5. When other filters such as IIR filter and average filter are used, these filters are applied on each channel; therefore, the memory usage and the processing time will be three times to the case when MFS is not enabled.
6. SmartSense™ and the MFS feature cannot be used together. They are mutually exclusive features.

## Component-Specific Considerations

1. If MFS is enabled, the current CapSense Component (v6.0 and earlier) changes the IMO clock frequency during each sensor scan. All components which use IMO for critical time-dependent operations such as SysTick, CapSense gestures, and asynchronous communication such as UART should be taken care.

   **Note**: It is also possible to change the sense clock frequency without modifying the IMO clock by changing the sense clock divider. This ensures that time-critical operations are not affected by MFS. However, this is a custom technique and doesn't provide as good noise immunity as the IMO modification method. Contact Cypress technical support for support on this method.
2. If MFS is enabled, the current CapSense Component (v6.0 and earlier) sets the CSX Tx Clock frequency to fixed 300 kHz in third-generation devices and to 1 MHz in fourth-generation CapSense devices. See the 'CapSense Selector Guide' section in AN64846 – Getting Started with CapSense to know the CapSense generation present in each PSoC device.

   **Note:** Because the Tx clock frequency is fixed, ensure that the Tx and Rx electrode's parasitic capacitance is charged and discharged completely with this Tx clock frequency. See the "CapSense Performance Tuning (CSX Sensing Method)" section in AN85951 – PSoC® 4 and PSoC 6 MCU CapSense® Design Guide for information on dependence of Tx Clock frequency on sensor parasitic capacitance.

Figure 1. Scan Time when Multi-frequency Scan Is Disabled

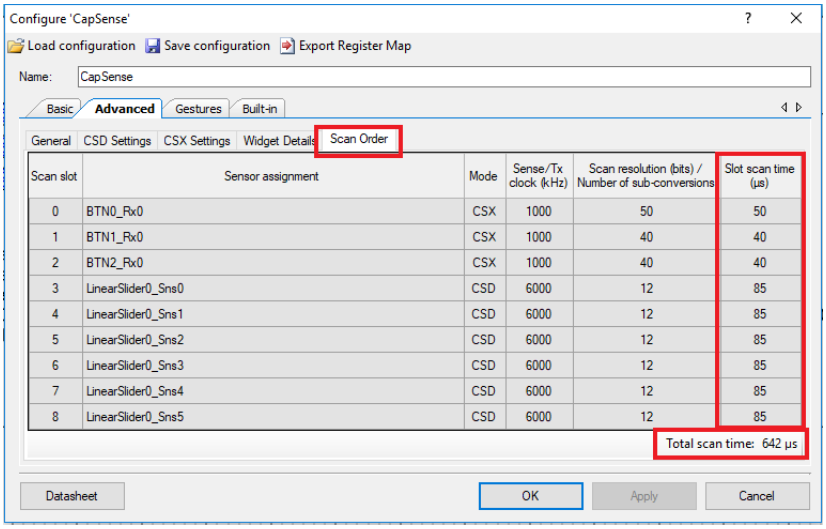
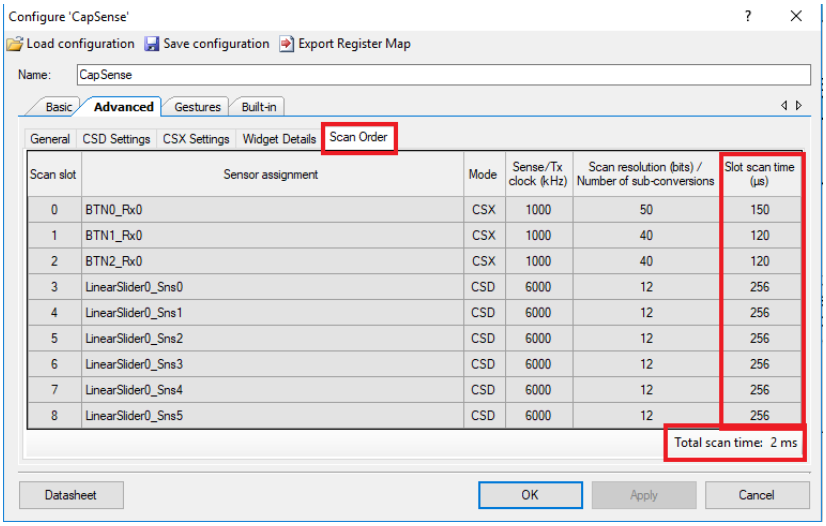



Figure 2. Scan Time when Multi-frequency Scan is Enabled

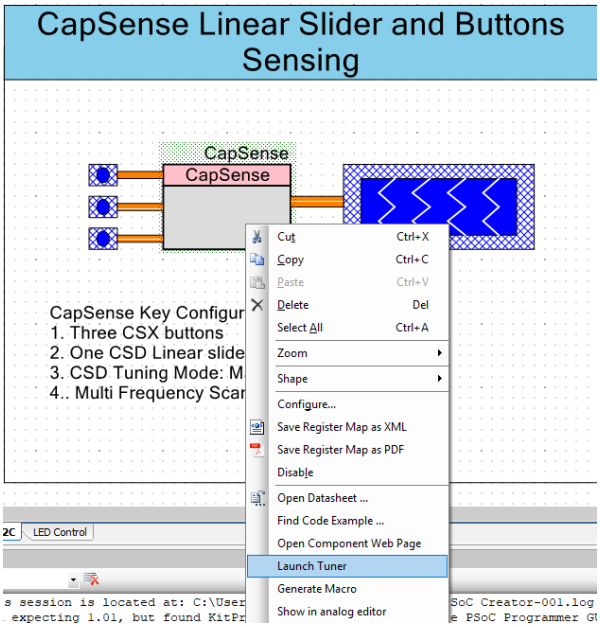



# Operation

## Testing the Basic Operation

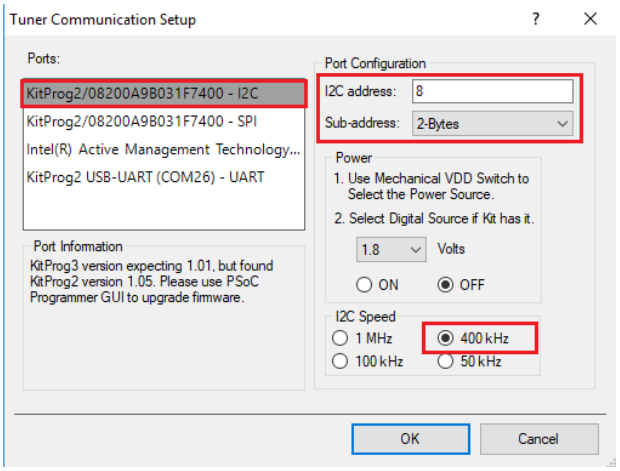
1. Power the PSoC 4100S Plus device by plugging a USB 2.0 Type A to Micro-B Cable on J8 (USB Micro-B connector).
2. Build the code example by selecting **Build** > **Build CE227719_CapSense_with_Multi_frequency_scan** in PSoC Creator.
3. Push the SWD Select switch (**SW4**) to the PSoC position to program the PSoC 4100S Plus device.
4. Program the PSoC 4100S Plus device (select **Debug** > **Program**).
5. Touch the CapSense buttons and observe that the corresponding LEDs turn ON.
6. Slide your finger over the CapSense linear slider and observe that the LEDs turn ON up to the position of touch.
7. To monitor CapSense raw data using the Sense Tuner GUI, launch the tuner by right-clicking on the CapSense Component and choose **Launch Tuner** from the drop-down menu as shown in Figure 3.

Figure 3. Launch Tuner GUI




8. A new window called “Sense Tuner” appears on the screen. Open the Tuner Communication Setup dialog by selecting **Tools** > **Tuner Communication Setup** in the menu, or clicking the **Tuner Communication Setup** icon.
9. Select the appropriate I²C communication device and set the following parameters as shown in Figure 4.

Figure 4. Tuner Communication Setup Parameters




10. Click **Connect** to establish connection.
11. Click **Start** to start data streaming from the device. The Tuner GUI displays data from the sensor in the widget view and Graph view tabs. See the “CapSense Tuner” section in CapSense Component datasheet for detailed information on Tuner GUI.
12. Observe from the Widget/ Sensor parameters section in the Sense Tuner window that the CapSense Component calculates three different modulator IDAC values for each of the channels as shown in Figure 5 and Figure 6.

Figure 5. Three Different MOD IDAC Values for the CSX Button

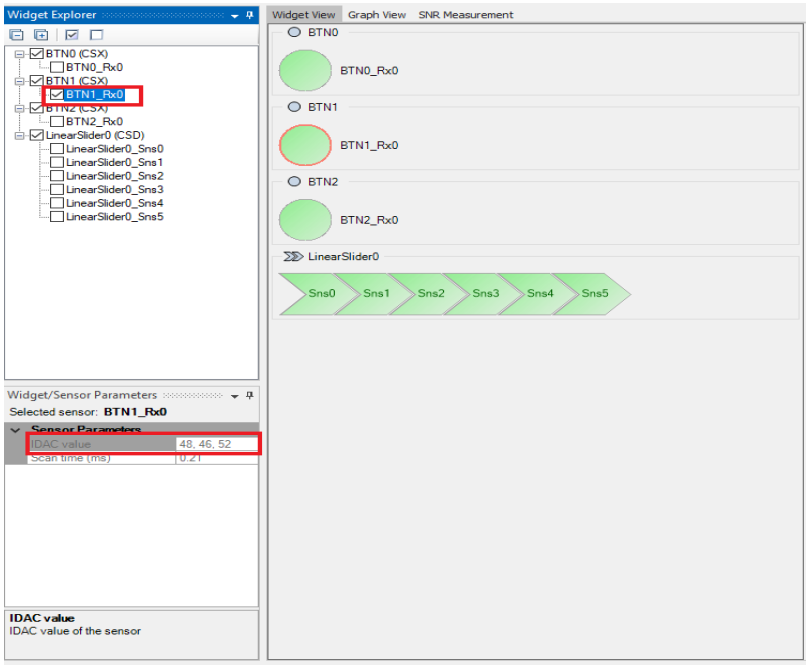
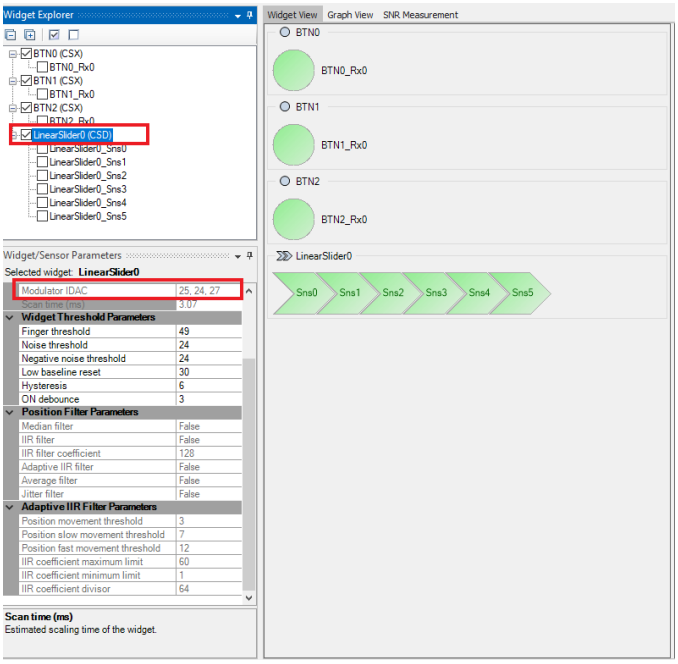


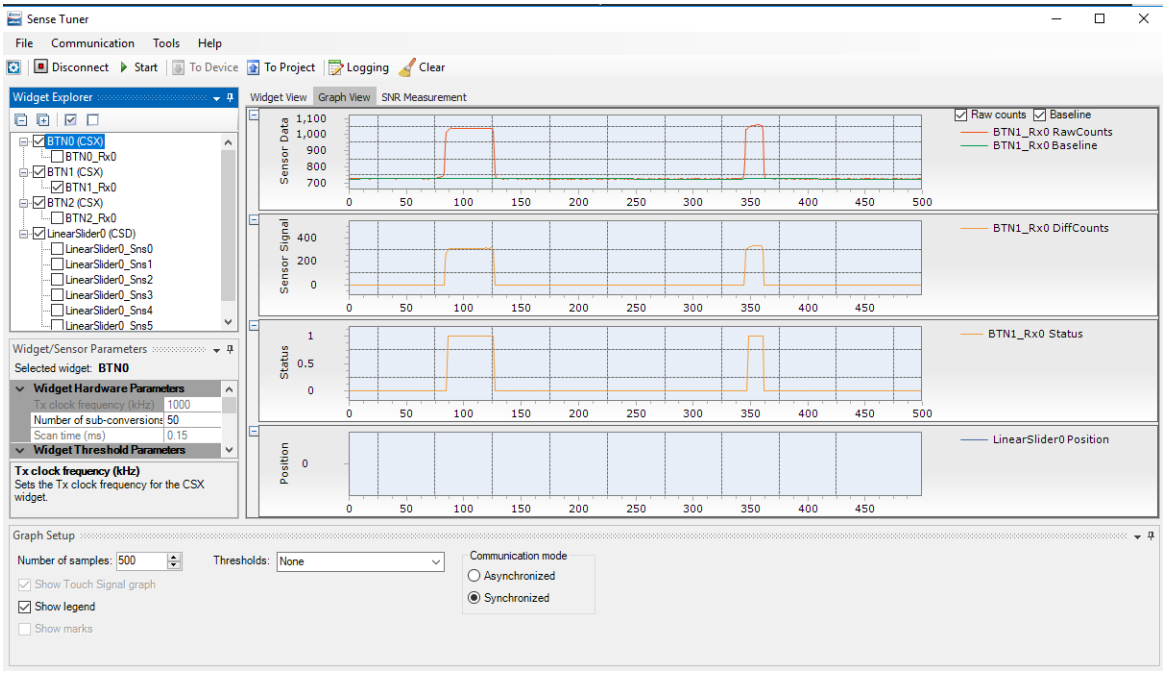

Figure 6. Three Different Modulation IDAC Values for the CSD Slider Widget




**Note:** If compensation IDAC is enabled, the three different compensation IDAC values can be observed for each sensor if CSD sensing method is used.

It can be seen from that that graph view of the Sense Tuner that the only the raw count and baselines for Channel 0 are displayed as shown in Figure 7. The Tuner displays the difference count graph after applying the median filter. You can use the Bridge Control Panel tool to monitor the raw counts and baseline for all the three channels, as the next section Using Bridge Control Panel to Monitor Raw Counts and Baselines explains.

Figure 7 Graph View of Sense Tuner



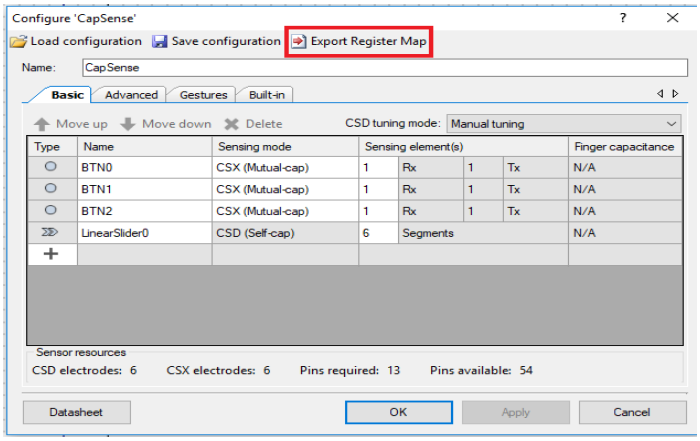

## Using Bridge Control Panel to Monitor Raw Counts and Baselines

The Bridge Control Panel is a data monitoring and logging tool that is automatically installed along with the PSoC Programmer. If you are unable to find this tool installed on your PC, install the latest PSoC Programmer version.

Refer to the following steps to use Bridge Control Panel to monitor raw counts and baselines of all channels:

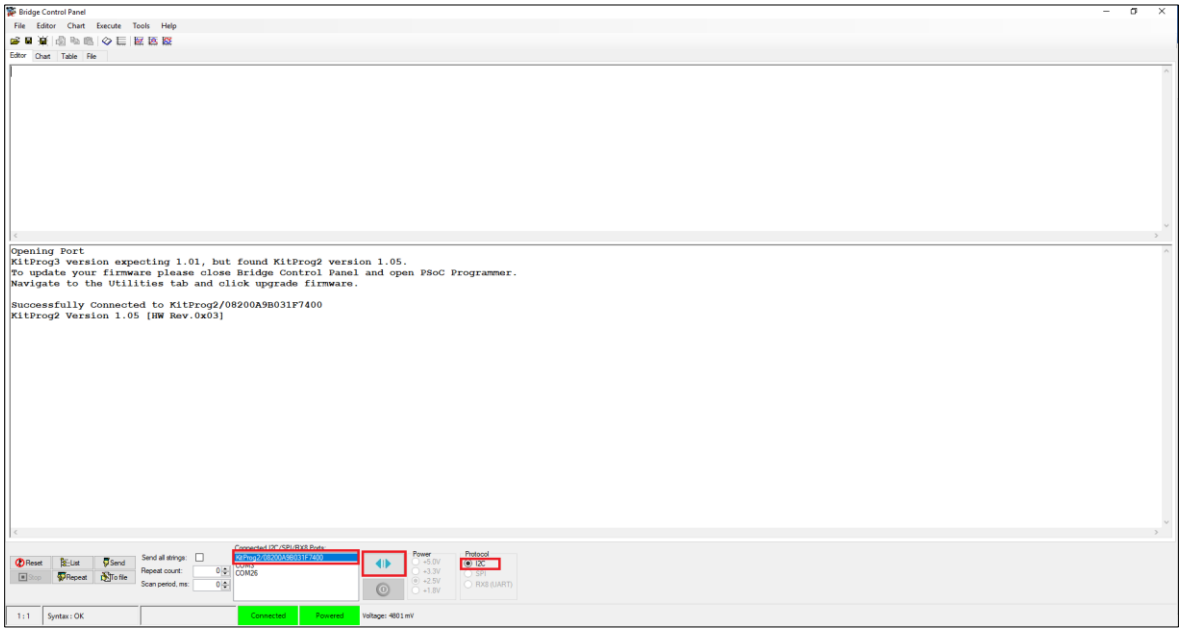
1. Create a register map file for the CapSense configuration by clicking on the **Export Register Map** option in the Configure 'CapSense' window as shown in Figure 8.

Figure 8. Export Register Map Option in CapSense Configuration Window



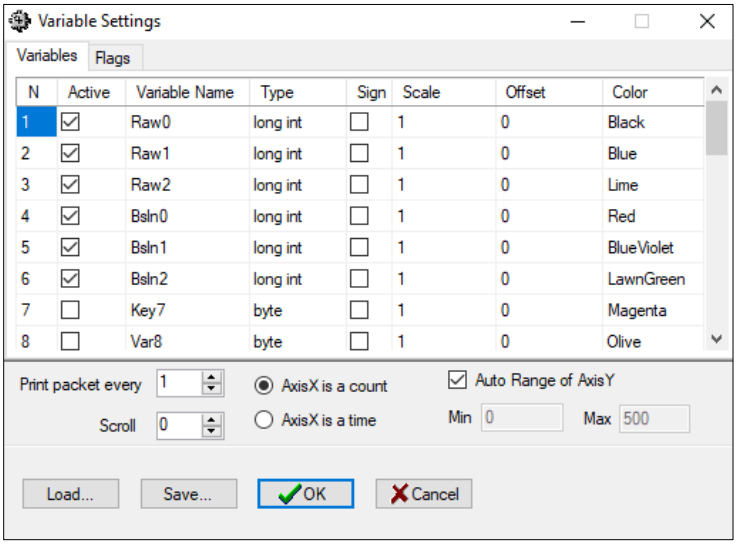

2. Power the CY8CKIT-149 PSoC 4100S Plus Prototyping Kit by plugging a USB 2.0 Type A to Micro-B Cable on J8 (USB Micro-B connector).
3. Open the BCP tool (choose **Start** > **All Programs** > **Cypress** > **Bridge Control Panel** > **Bridge Control Panel**).
4. In the BCP tool, select the CY8CKIT-149 I2C-USB bridge port in **Connected I2C /SPI/RX8 Ports:** window and set I2C as its protocol, as Figure 9 shows.

Figure 9. Bridge Control Panel - Port and Protocol Selection




5. Select **Charts** > **Variable Settings** and provide the variable names for each parameter (raw counts and baselines for each channel) as shown in Figure 10 and click **OK** to exit.

Figure 10. Variable Settings in BCP




6. Do the following to read the raw counts and baselines for the three channels of BTN1 as illustrated in Figure 12:

   a. Choose **File** > **Open File** and select the *CE227719_CapSense_with_Multi_Frequency_Scan - Commands.iic* file supplied with the project and click **Open**. Alternatively, go to **Editor** and type or copy the following command:

   ```
   W 08 00 7A R 08 @0Raw0 @1Raw0 @0Raw1 @1Raw1 @0Raw2 @1Raw2 @0Bsln0 @1Bsln0 @0Bsln1 @1Bsln1
   @0Bsln2 @1Bsln2 p
   ```

   The offset (00 7A) in the above command is entered based on the register map file generated in step1. Figure 11 shows the parameters referred from the register map file (register offset for raw0 = 122 (007A in hexadecimal).

Figure 11. Offset of BTN0's Raw Count from Register Map File

**BTN1_RX0_RAW**

The sensor raw counts.

| Register/struct name | Type | Offset | Size |
|---|---|---|---|
| BTN1_RX0_RAW0 | uint16 | 122 | 2 |
| BTN1_RX0_RAW1 | uint16 | 124 | 2 |
| BTN1_RX0_RAW2 | uint16 | 126 | 2 |

b. Place the cursor on the command line and then click **Repeat**.

c. Observe the data packets in the output window.

7. To view the sensor data in a graphical format, click on the **Chart** tab. Figure 13 shows the raw counts and baselines for the three channels.

Figure 12. Reading CapSense Sensor Data in Bridge Control Panel

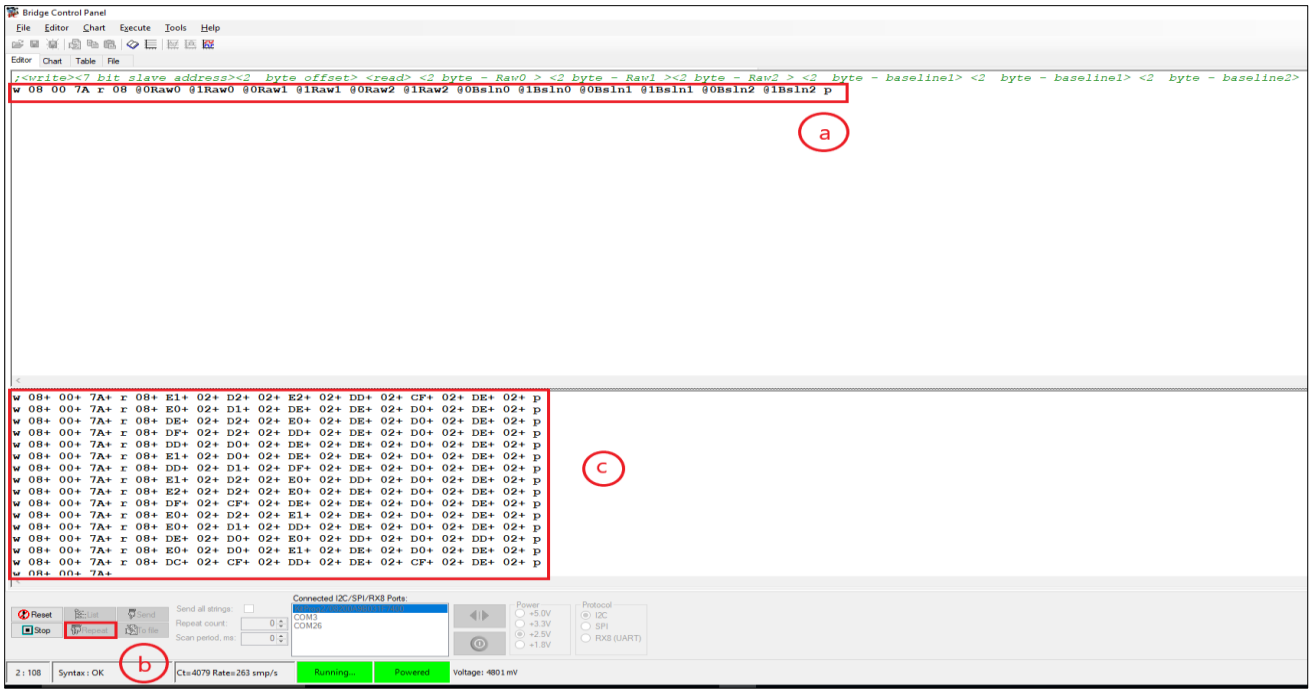
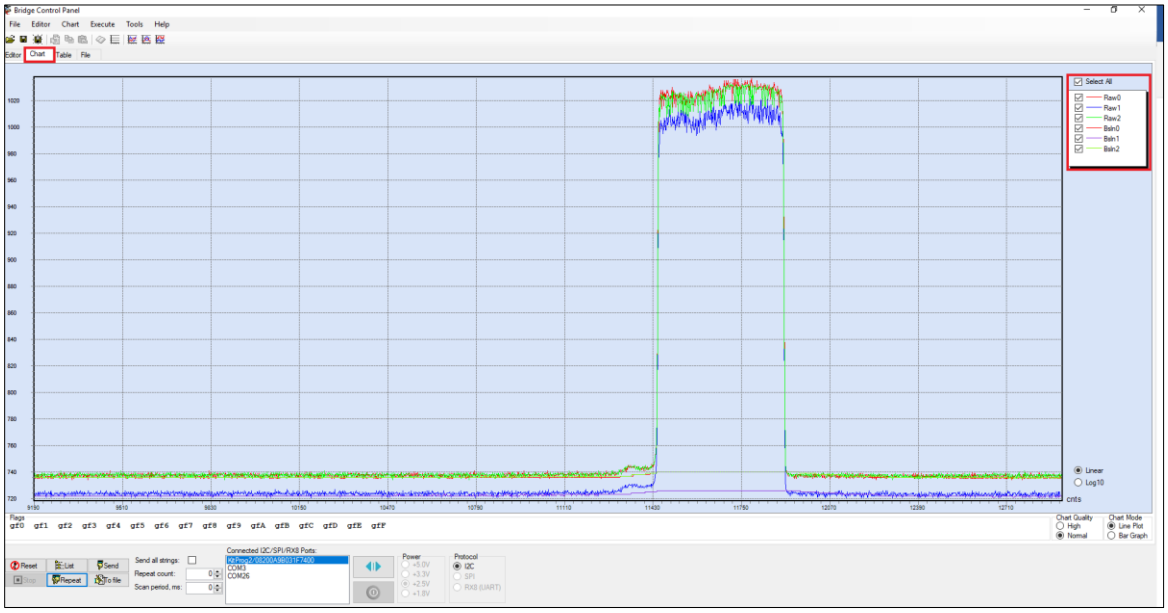


Figure 13. Chart Showing Three Different Raw Counts and Baselines

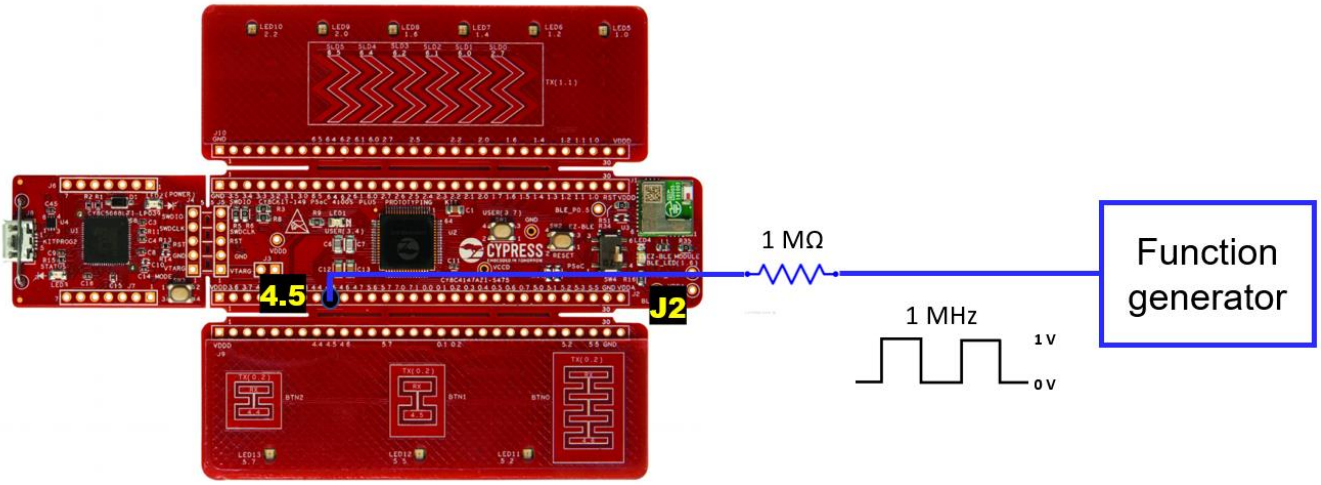


## Noise Injection to Test Effectiveness of MFS

This section explains how to inject extreme external noise into the system and show the effectiveness of MFS in reducing false triggers caused by noise injection.

1. Inject a square wave of 1 MHz (same as Tx Clock frequency) with an amplitude of 1 V into the system at the Rx pin of BTN1 (Pin 4.5 in J2 header) using a function generator as shown in Figure 14.

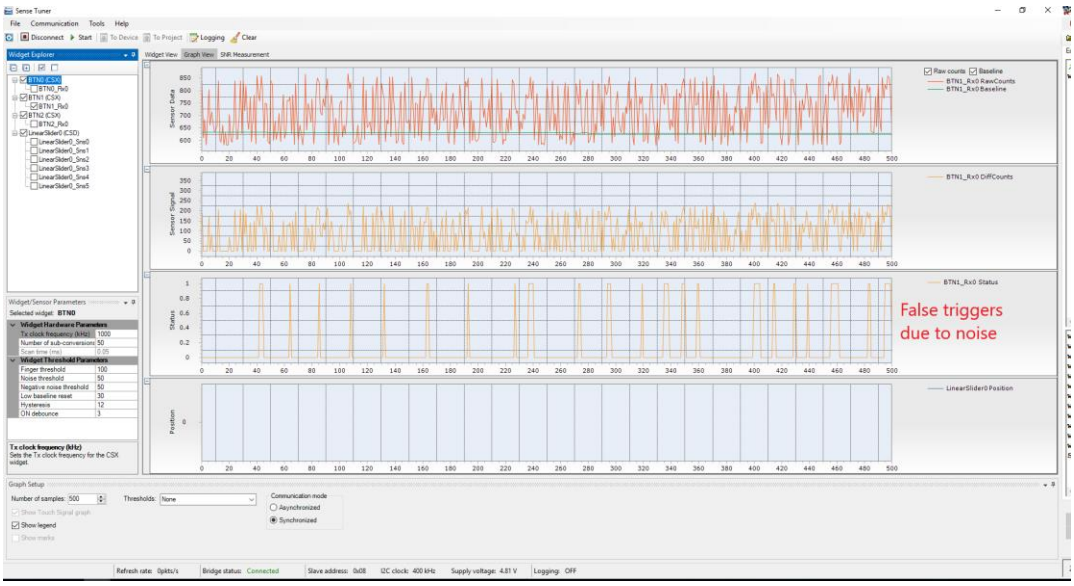
Figure 14. Test Circuit to Inject External Noise into the System Using Function Generator




**Note**: The amplitude of the noise can be varied by changing the voltage output of the function generator.

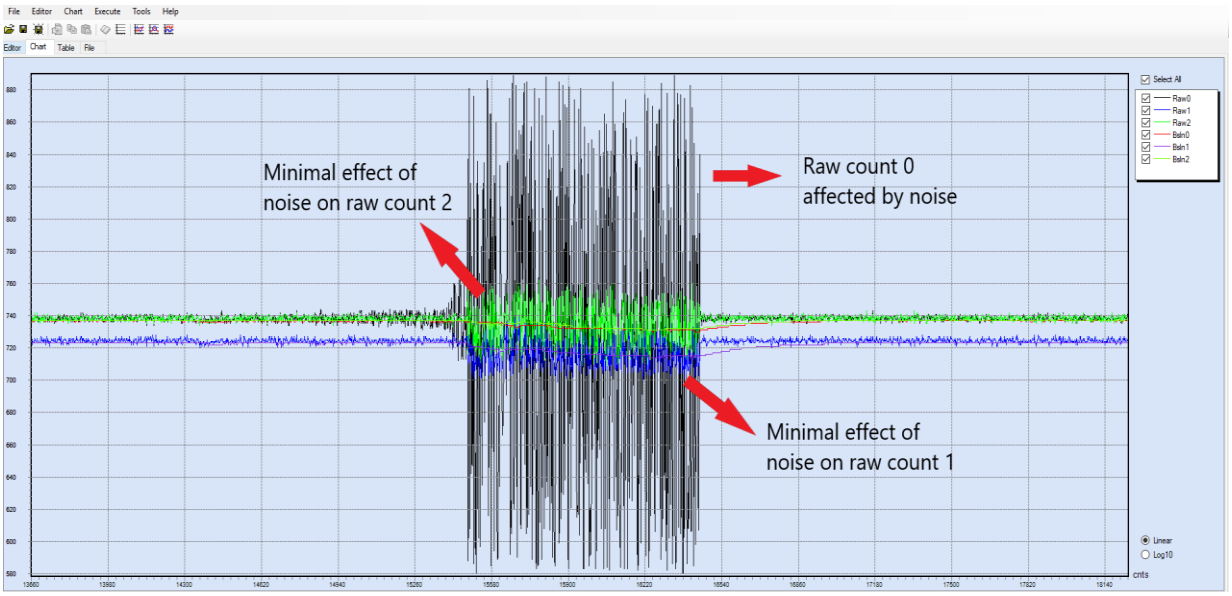
2. When the MFS feature is not enabled, the BTN1 sensor false triggers due to the induced noise. This can be seen using Sense Tuner GUI as shown in Figure 15. You may also be able to observe this by the flickering of the corresponding LED in the kit indicating false touch.

Figure 15. MFS Disabled: Sensor False Triggers Due To Induced Noise



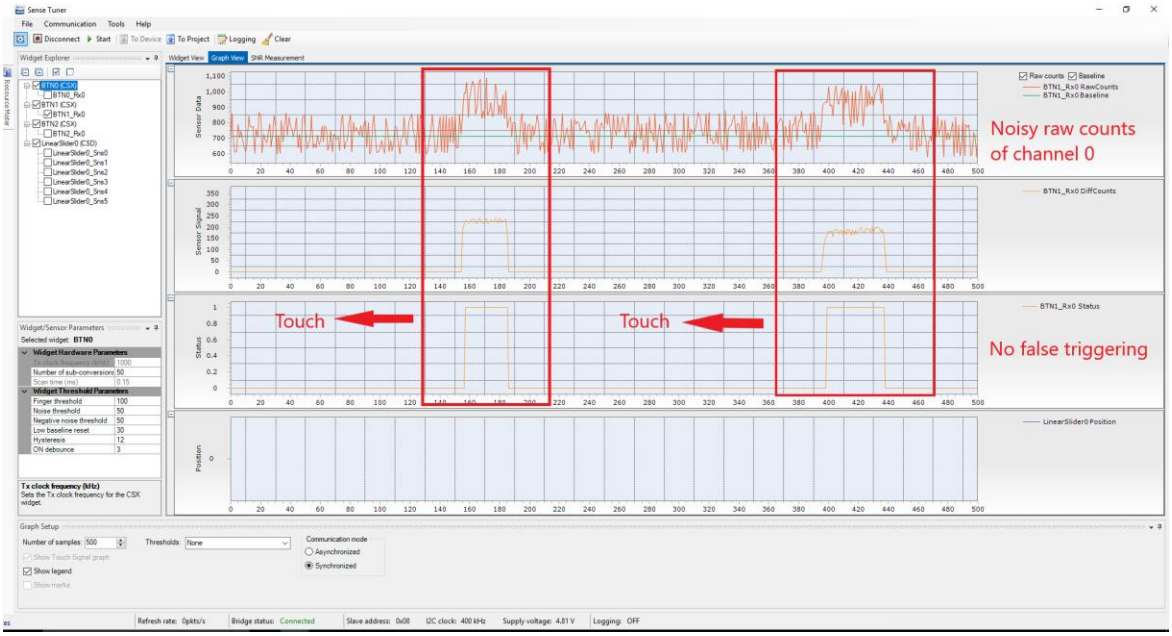

3. Now, MFS is enabled as shown in Figure 21. The effect of noise on the three different raw counts can be seen using BCP as shown in Figure 16. It is evident that only Channel 0 is affected by the external noise because it is closer to the noise frequency and the effect of noise on Channel 1 and Channel 2 is minimal because they have a different frequency when compared to the noise frequency.

Figure 16. F Enabled: Effect of Extreme Noise on Different Channels



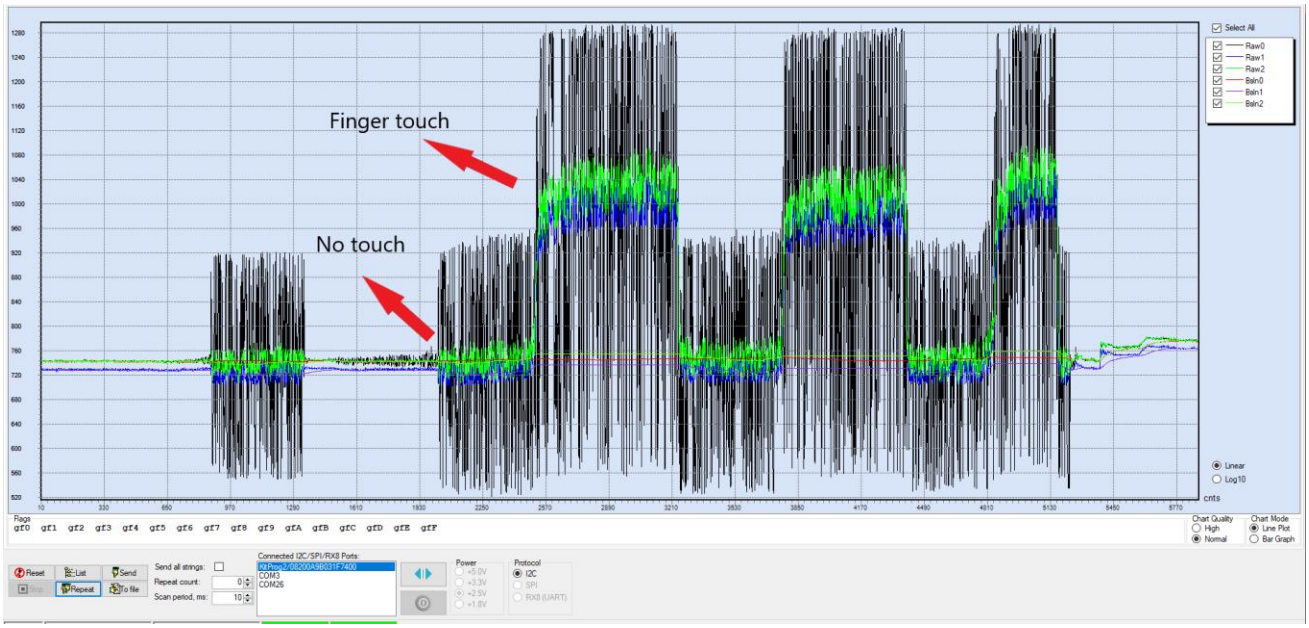

4. Touch the sensors to see of the effectiveness of MFS feature. It is clearly seen from the Tuner GUI graph view that there are no false triggers due to the induced noise as shown in Figure 17. The sensor responds well to finger touch.

Figure 17. MFS Enabled: No Sensor False Triggers Due to Induced Noise




5. It is seen from Figure 18 that channel 1 and channel 2 responds well to touch and only channel 0 is affected by the induced noise.

Figure 18. No False Triggering when MFS Is Enabled

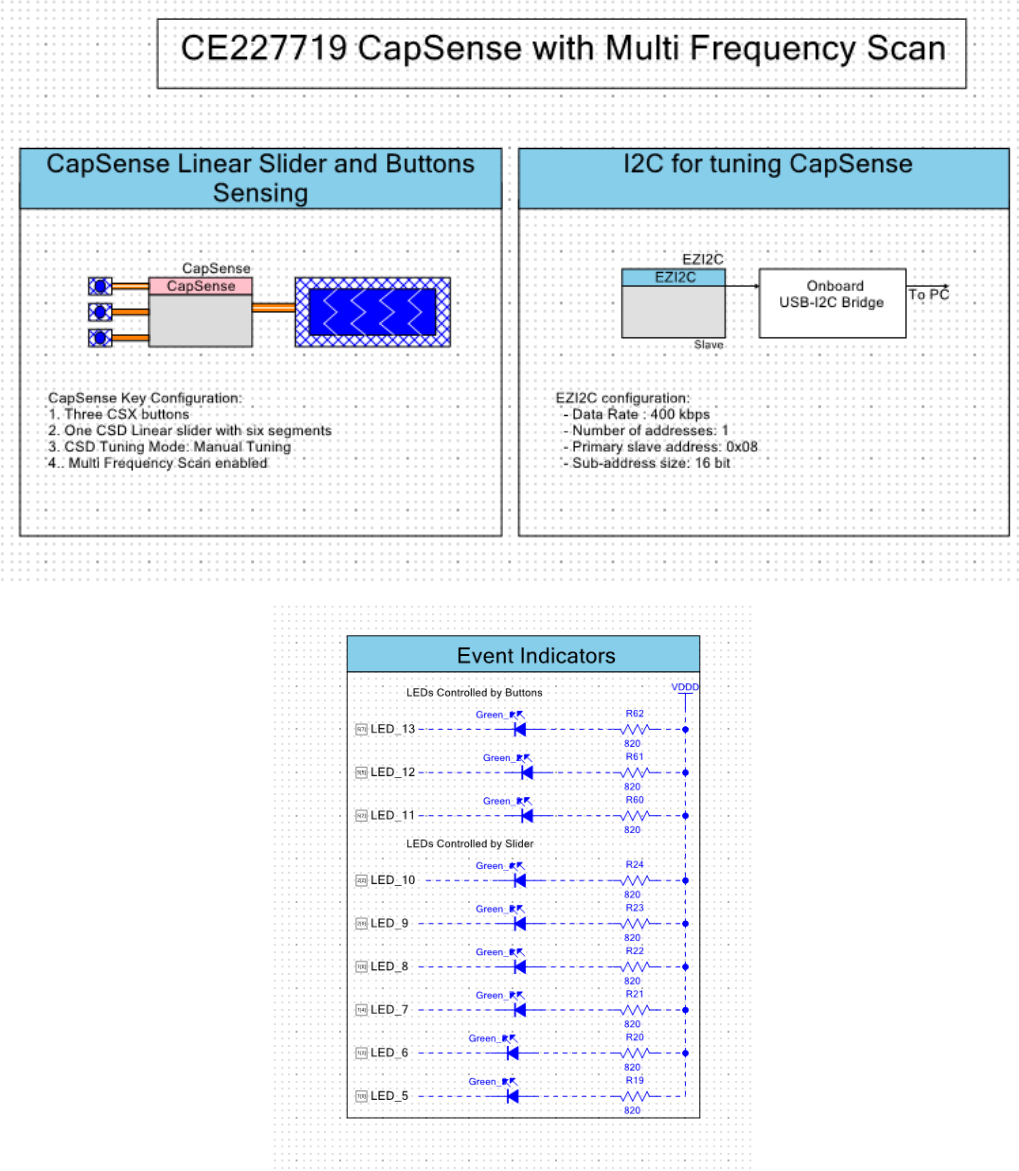


## Design and Implementation

This code example has a single workspace – *CE227719_CapSense_with_Multi_Frequency_Scan*.

Figure 19 shows the PSoC Creator schematics for the "CE227719_CapSense_with_Multi_Frequency_Scan" code example. This code example uses CapSense, SCB (configured as EZI2C Slave), Pin, and Clock.

Figure 19 . PSoC Creator Schematics for CapSense Linear Slider, Buttons, and LEDs




The CapSense Component is configured with one linear slider widget with six segments and three button widgets. This code example uses the CapSense Sigma-Delta (CSD) sensing method for sliders and Mutual Capacitive Sensing (CSX) sensing method for buttons. Both mutual and self capacitance have their own advantages. To understand the detailed comparison between them, refer to the application note, AN85951 – PSoC® 4 and PSoC 6 MCU CapSense® Design Guide. When the CapSense linear slider is touched, the LEDs turns ON to indicate the position of touch. When a touch is detected on CapSense buttons, the corresponding LEDs turn ON.

The code example is designed for the PSoC 4100S Plus device and associated CY8CKIT-149 PSoC 4100S Plus Prototyping Kit. Table 1 shows the pin assignment of the PSoC Creator Components.

Table 1. Pin Assignment for the CE227719 CapSense with Multi-frequency Scan Code Example

| Pin name | CY8CKIT-149 PSoC 4100S Plus |
|---|---|
| \Capsense:CintA\ | P4[2] |
| \Capsense:CintB\ | P4[3] |
| \Capsense:Cmod\ (Cmod) | P4[1] |
| \CapSense:Rx[0]\ (BTN0_Rx0) | P4[6] |
| \CapSense:Rx[1]\ (BTN1_Rx0) | P4[5] |
| \CapSense:Rx[2]\ (BTN2_Rx0) | P4[4] |
| \CapSense:Sns[0]\ (SLD_Sns0) | P2[7] |
| \CapSense:Sns[1]\ (SLD _Sns1) | P6[0] |
| \CapSense:Sns[2]\ (SLD _Sns2) | P6[1] |
| \CapSense:Sns[3]\ (SLD _Sns3) | P6[2] |
| \CapSense:Sns[4]\ (SLD _Sns4) | P6[4] |
| \CapSense:Sns[5]\ (SLD _Sns5) | P6[5] |
| \CapSense:Tx\ (BTN0_Tx) | P0[2] |
| \EZI2C:scl\ | P3[0] |
| \EZI2C:sda\ | P3[1] |
| LED1 | P3[4] |
| LED5 | P1[0] |
| LED6 | P1[2] |
| LED7 | P1[4] |
| LED8 | P1[6] |
| LED9 | P2[0] |
| LED10 | P2[2] |
| LED11 | P5[2] |
| LED12 | P5[5] |
| LED13 | P5[7] |

Table 2 lists the PSoC Creator Components used in this example, how they are used in the design, and the non-default settings required so they function as intended.

Table 2. List of PSoC Creator Components in the CapSense with Multi-Frequency Scan Code Example

| Component | Instance Name | Version | Hardware Resources |
|---|---|---|---|
| CapSense | CapSense | v6.0 | One CSD |
| EZI2C Slave (SCB mode) | EZI2C | v4.0 | One Serial Communication Block (SCB) and two GPIOs |
| Digital Output Pin | LED_1 | v2.20 | I/O |
| | LED_5 | v2.20 | I/O |
| | LED_6 | v2.20 | I/O |
| | LED_7 | v2.20 | I/O |
| | LED_8 | v2.20 | I/O |
| | LED_9 | v2.20 | I/O |
| | LED_10 | v2.20 | I/O |
| | LED_11 | v2.20 | I/O |
| | LED_12 | v2.20 | I/O |
| | LED_13 | v2.20 | I/O |

For more information on the hardware resources used by a Component, refer to the respective Component datasheets.

# Parameter Settings

## CapSense Component

Figure 20 through Figure 27 show the settings for the CapSense Component.

Figure 20. CapSense Component - Basic Tab

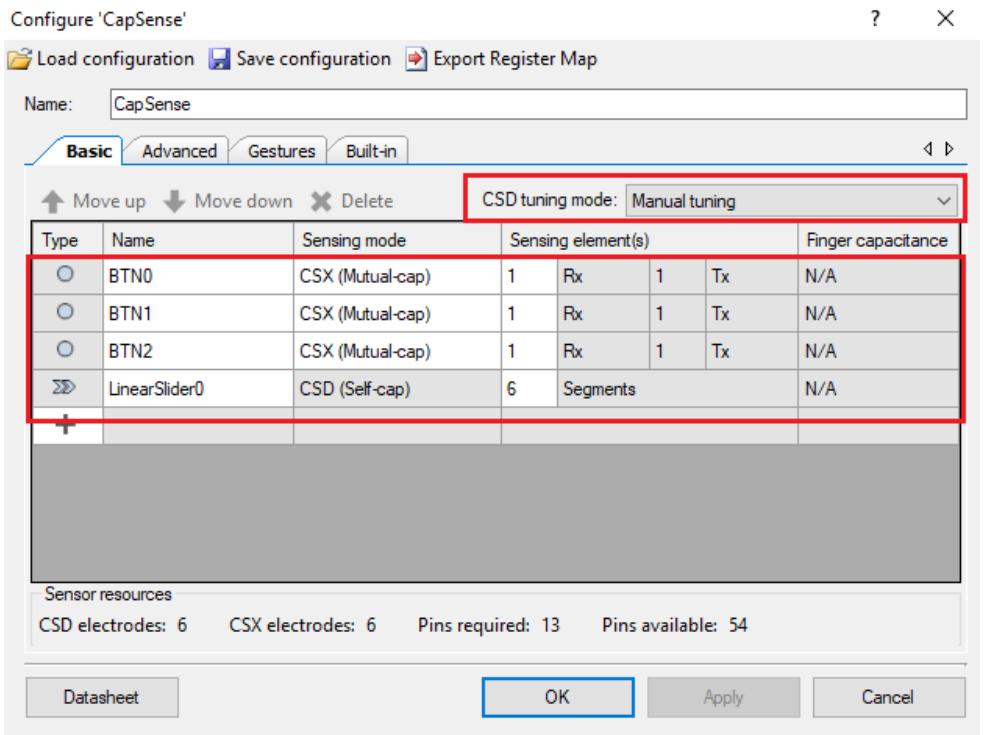


Figure 21. Enable Multi-Frequency Scan in General Settings of Advanced Tab

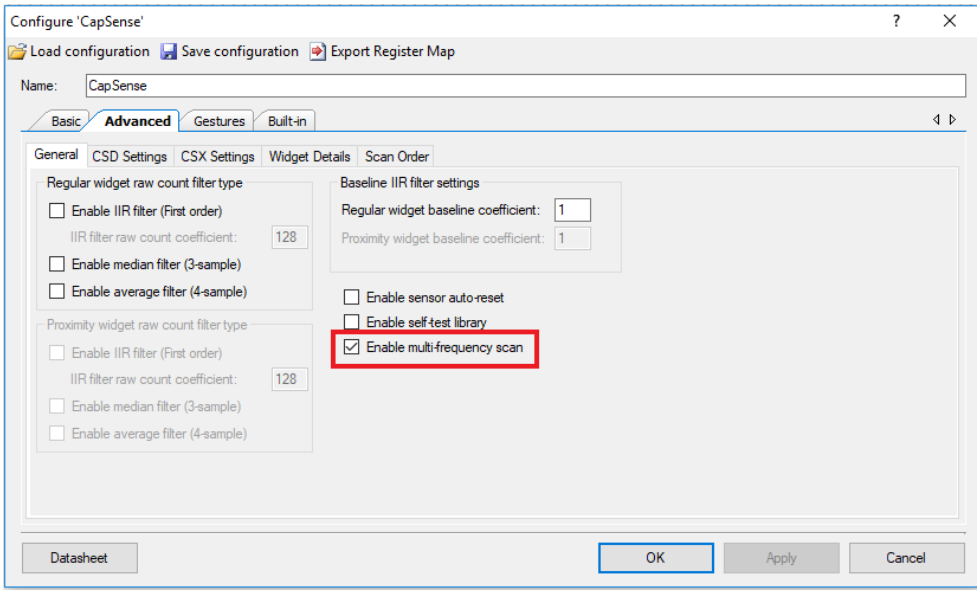



Figure 22. CSD Settings in Advanced Tab

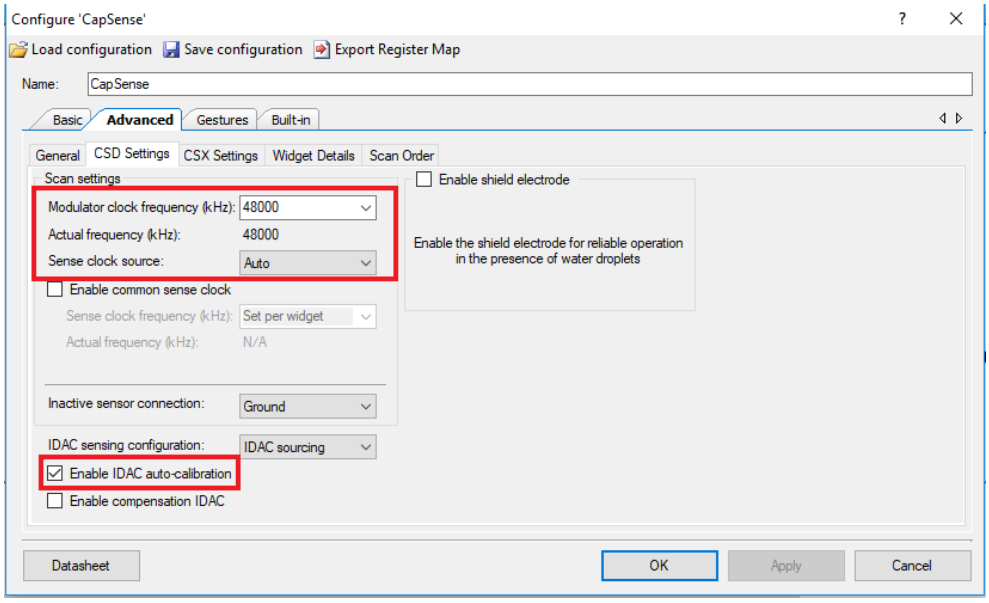


Figure 23. CSX Settings in Advanced Tab

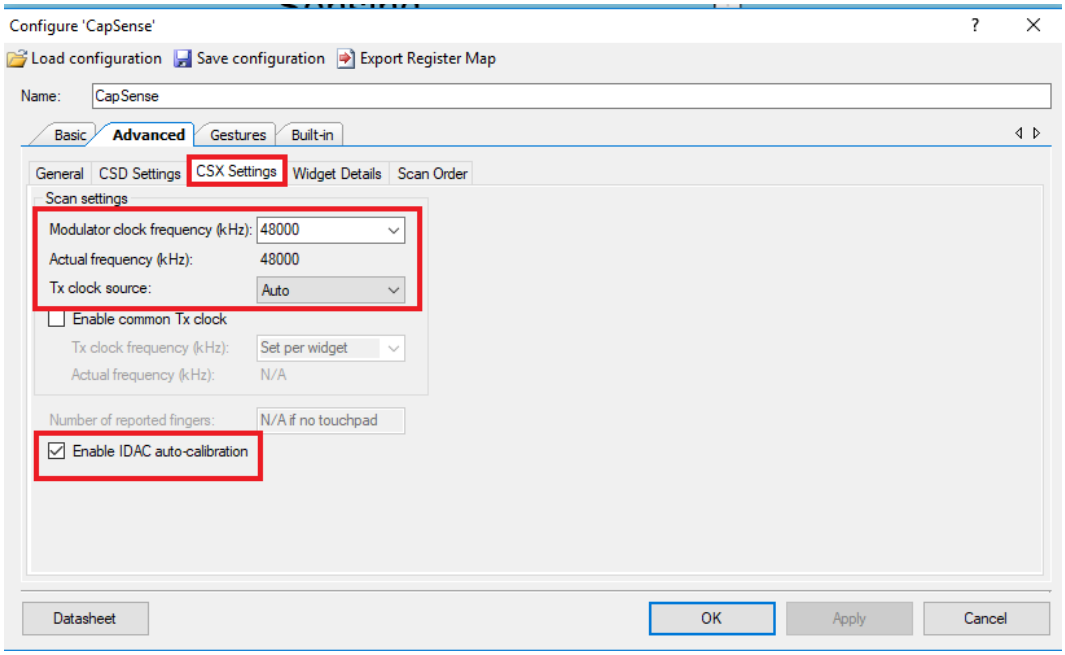



Figure 24. CapSense Component BTN0 CSX Settings

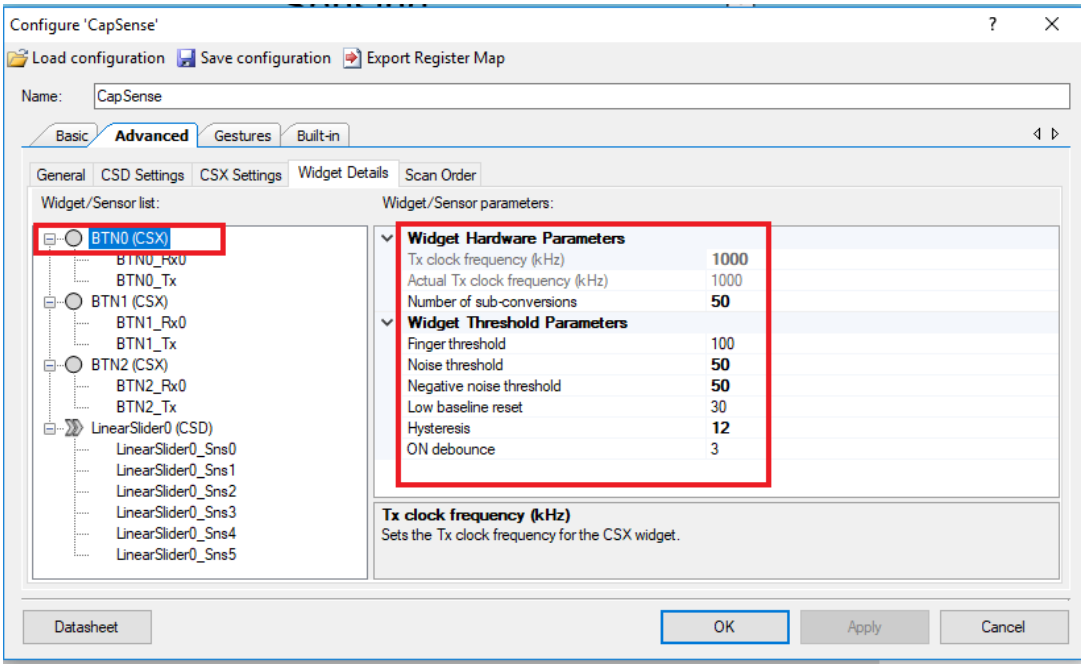


Figure 25. CapSense Component BTN1 CSX Settings

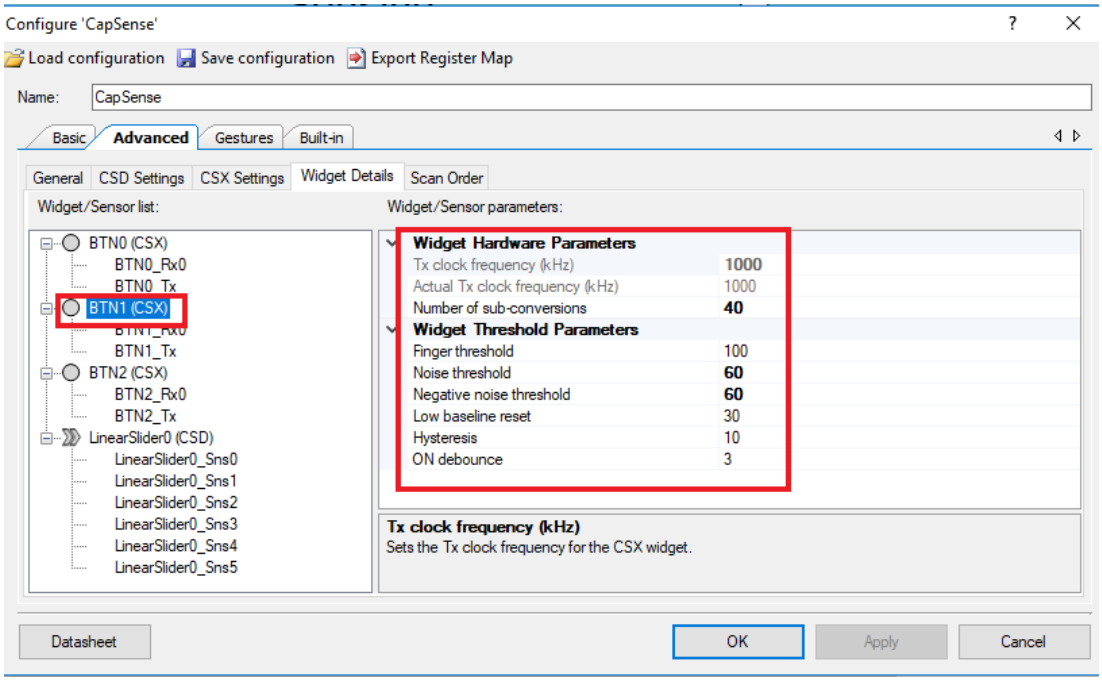



Figure 26. CapSense Component BTN2 CSX Settings

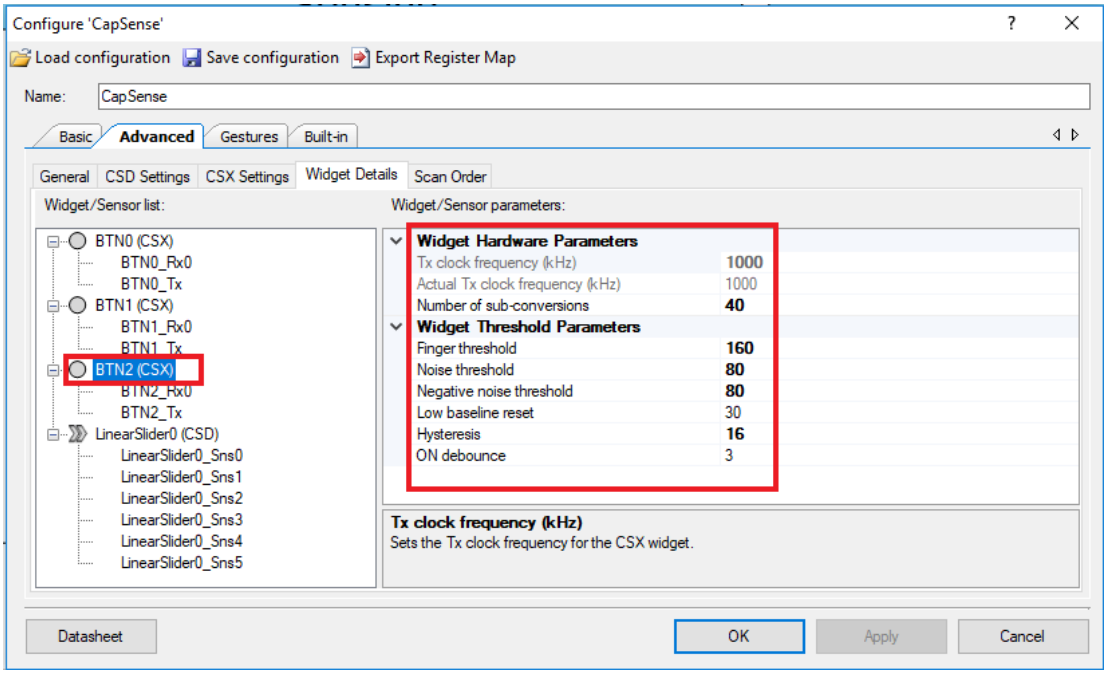
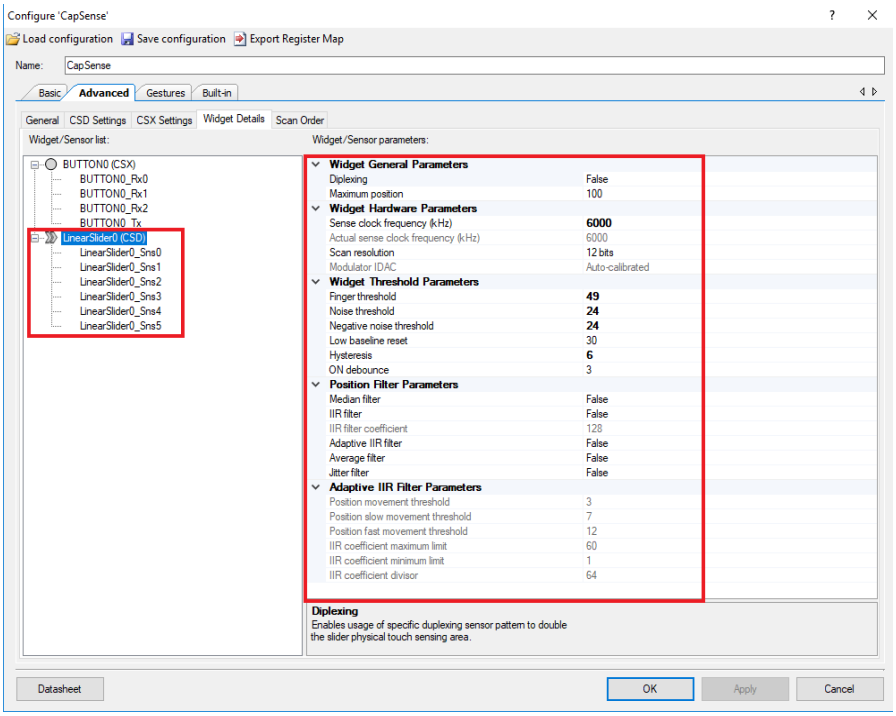


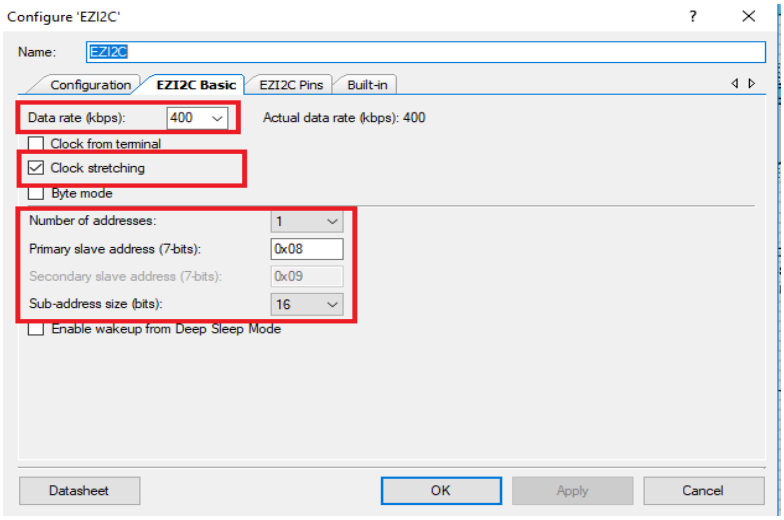
Figure 27. CapSense Component LinearSlider0 CSD Settings




## I2C Component

Figure 28 shows the settings for the EZI2C Component.

Figure 28. EZI2C Slave Component's Basic Tab



## Reusing This Example

The multi-frequency scan feature is available in CapSense Component versions v3.1 and above. This feature is supported in all CapSense-enabled devices in the PSoC 4 family of devices, including:

- **Third-generation CapSense**: PSoC 4000, PSoC 4100, PSoC 4200, PSoC 4100M, PSoC 4200M, PSoC 4200L, PSoC 4100 BLE and PSoC 4200 BLE.
- **Fourth-generation CapSense**: PSoC 4000S, PSoC 4100S PSoC 4100 S Plus and PSoC Analog Coprocessor.

  The tuning parameters must be changed in accordance with the kit used and sensor properties. See the PSoC 4 and PSoC CapSense Design Guide for more information on CapSense sensor tuning.

## Related Documents

| | | |
|---|---|---|
| **Application Notes** | | |
| AN85951 | PSoC 4 and PSoC 6 MCU CapSense Design Guide | Describes how to design capacitive touch sensing applications with PSoC 4 and PSoC 6 MCU |
| AN79953 | Getting Started with PSoC 4 | Introduces the PSoC 4 device and explains how to build a PSoC Creator project |
| **PSoC Creator Component Datasheets** | | |
| CapSense | | Supports various interfaces such as Button, Matrix Buttons, Slider, Touchpad, and Proximity Sensor |
| EZI2C Slave | | Supports one or two address decoding with independent memory buffers |
| Pins | | Supports connection of hardware resources to physical pins. |
| I2C | | Supports $I_2C$ Slave, Master, Multi-Master, and Multi-Master-Slave configurations |
| **Device Documentation** | | |
| PSoC 4100S Plus Datasheet | | PSoC 4100S Plus Architecture Technical Reference Manuals<br>PSoC 4100S Plus Register Technical Reference Manuals |
| **Development Kit (DVK) Documentation** | | |
| CY8CKIT-149 PSoC 4100S Plus Prototyping Kit | | |

# Document History

Document Title: CE227719 – CapSense with Multi-Frequency Scan

Document Number: 002-27719

| Revision | ECN | Submission Date | Description of Change |
|---|---|---|---|
| ** | 6654094 | 09/13/2019 | New code example |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| Arm® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

## Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach"). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

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