**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.

# CE229521 – PSoC 4 CapSense CSD Slider Tuning

## Objective

This code example demonstrates how to manually tune CapSense® Sigma Delta (CSD)-based slider widget in PSoC® 4 devices using CapSense Tuner GUI.

## Requirements

**Tool:** PSoC Creator™ 4.3

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

**Associated Parts:** PSoC 4100S Plus, PSoC 4000S

**Related Hardware:** CY8CKIT-149 PSoC 4100S Plus Prototyping Kit, CY8CKIT-145-40XX PSoC 4000S CapSense Prototyping Kit.

## Overview

This example demonstrates how to manually tune a CSD-based slider using CapSense Tuner GUI. This document includes:

- A high-level overview of the CSD slider widget tuning flow.
- An example to manually tune a CSD slider widget.
- A procedure on how to use the CapSense Tuner GUI to monitor the CapSense raw data and fine-tune the CSD slider for optimum performance such as response time and linearity.

The code scans a slider widget using the CSD sensing method and sends the CapSense raw data over an I2C interface to the CapSense Tuner GUI tool on a PC using the onboard KitProg USB-I2C bridge.

## Hardware Setup

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

The example project is set up for CY8CKIT-149 PSoC 4100S Plus Prototyping Kit. The project can be migrated to any supported kits as listed in Table 1 by changing the target device with Device Selector called from the project's context menu and setting the pin configuration as shown in Table 2.

**Note:** As the CY8CKIT-145 slider supports only five segments, double-click the CapSense Component and decrease the **Sensing element(s)** to **5**.

Table 1. Supported Development Kits

| Development Kit | Series | Device |
|---|---|---|
| CY8CKIT-149 | PSoC 4100 S Plus | CY8C4147AZI-S475 |
| CY8CKIT-145 | PSoC 4000 S | CY8C4045AZI-S413 |

Table 2 lists the pin assignment of the PSoC Creator Components. To change the pin assignments, override the control file selections in the Pin Editor of the Design Wide Resources by selecting the new port or pin number.

Table 2. Pin Assignment for the CE229521 PSoC4 CapSense CSD Slider Tuning Code Example

| Pin Name | CY8CKIT-149 PSoC 4100S Plus | CY8CKIT-145 PSoC 4000S |
|---|---|---|
| \Capsense:Cmod\ (Cmod) | P4[1] | P4[1] |
| \CapSense:Sns[0]\ (SLD_Sns0) | P2[7] | P0[0] |
| \CapSense:Sns[1]\ (SLD _Sns1) | P6[0] | P0[1] |
| \CapSense:Sns[2]\ (SLD _Sns2) | P6[1] | P0[2] |
| \CapSense:Sns[3]\ (SLD _Sns3) | P6[2] | P0[3] |
| \CapSense:Sns[4]\ (SLD _Sns4) | P6[4] | P0[6] |
| \CapSense:Sns[5]\ (SLD _Sns5) | P6[5] | - |
| \EZI2C:scl\ | P3[0] | P1[0] |
| \EZI2C:sda\ | P3[1] | P1[1] |

## Software Setup

Make sure to install the software tools mentioned in Requirements.

## Prerequisite

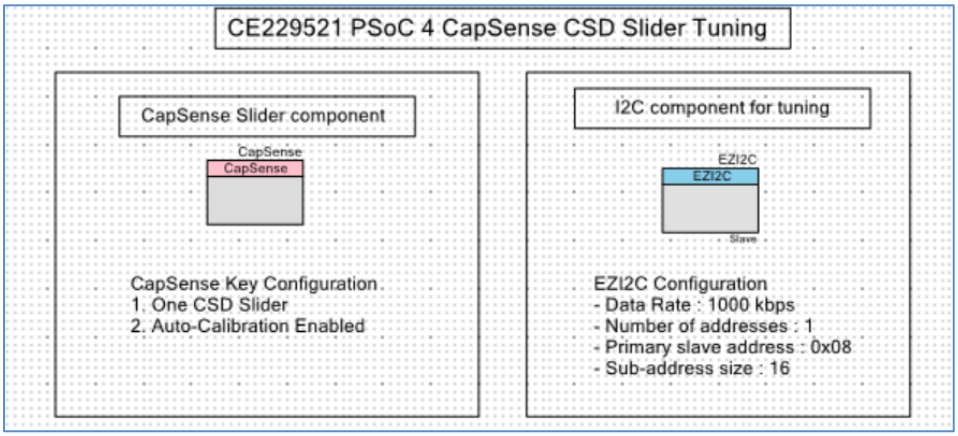
You should be familiar with CapSense technology, PSoC Creator, and basic terms such as CSD, raw counts, baseline, and difference counts. If you are new to CapSense, see AN85951 – PSoC® 4 and PSoC 6 MCU CapSense® Design Guide.

## Design and Implementation

This code example has a single workspace – CE229521. The project demonstrates the core functionality of the CapSense Component in line with the objective.

Figure 1 shows the PSoC Creator schematics of this code example. This code example uses CapSense and EZI2C Slave Components. The CapSense Component is configured with one slider widget. The project uses the CSD Manual tuning mode. The EZI2C Slave Component is used to monitor the sensor data and slider touch position information on a PC using the CapSense tuner available in the PSoC Creator IDE via I2C communication.

Figure 1. PSoC Creator Schematics for CapSense CSD Slider Tuning



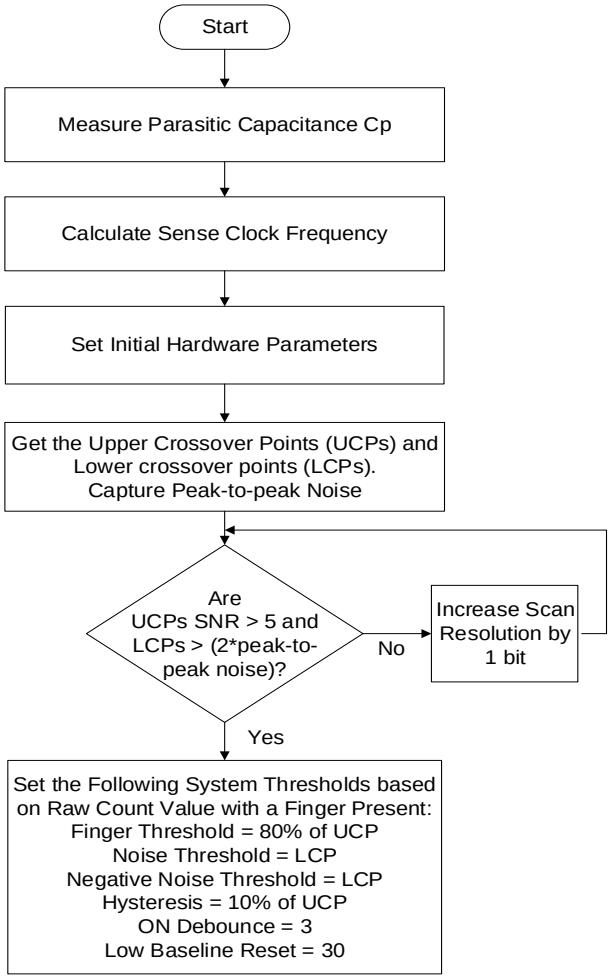

## Operation

The following steps explain the tuning procedure. Since the project already has the necessary settings by default, you can skip this procedure and go to Testing the Basic Operation to verify the operation. If you want to understand the tuning process and follow the steps for this kit or your own board, see Tuning Procedure.

## Tuning Procedure

Figure 2 illustrates the high-level procedure to tune the CSD slider.

Figure 2. High-level Overview of CSD Slider Widget Tuning Flow




Here are the steps to tune the slider.

### Step 1. Measuring Cp

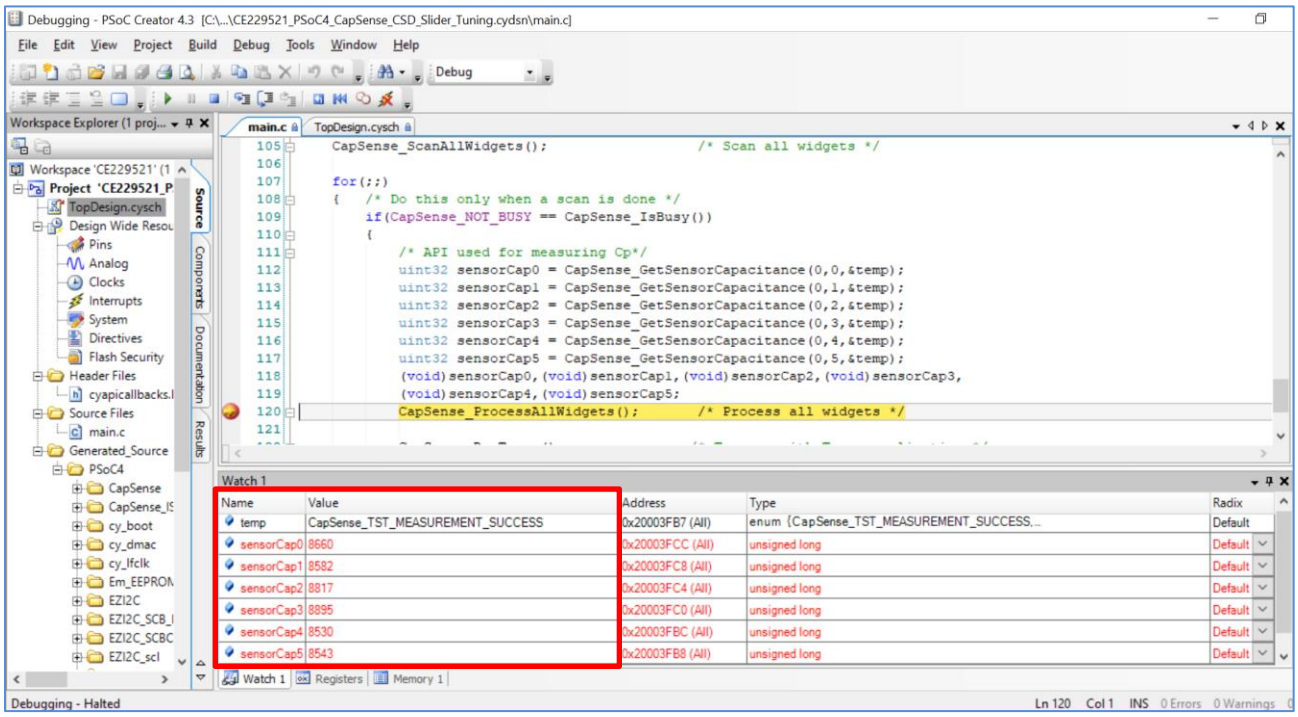
Use the `CapSense_GetSensorCapacitance ()` function to measure the parasitic capacitance, Cp, of each slider element to determine the slider element with maximum Cp. You can also use an LCR meter to measure Cp of the slider segments.

Follow these steps to determine Cp values in debug mode:

1. Follow the menu path **Project** > **Build Settings**. From the Tool Chain name, **ARM GCC5.4-2016-q2-update**, expand **Compiler** to locate **Optimization**. Set the **Optimization Level** to **None** to make sure that the sensor capacitance value measurement variables are not optimized out.
2. Program the device in debug mode (select **Debug** > **Debug**).
3. Place a breakpoint next to line 120 (after all capacitance measurements).
4. Follow the menu path **Debug** > **Windows** > **Watch** > **1** to open the Watch window.
5. In Watch 1, add the Cp measurement variables **sensorCap0**, **sensorCap1**, **sensorCap2**, **sensorCap3**, **sensorCap4**, **sensorCap5**. Also, add the measurementStatus pointer variable **temp** to **Watch 1**.

6. Click the resume execution icon (green arrow) to reach the breakpoint. Note that the **temp** variable reads **CapSense_TST_MEASUREMENT_SUCCESS** and the measurement variables provide capacitance of the sensor elements in **femtofarads** as shown in Figure 3.

Figure 3. Sensor Capacitance Measurement Values Obtained in Debug Mode




7. Follow the menu path **Debug → Stop Debugging** to exit the debug mode.

Table 3 shows the Cp values obtained for CY8CKIT-149 and CY8CKIT-145.

Table 3. Cp Values Obtained for CY8CKIT-149 and CY8CKIT-145 Kits

| Slider Segment | CY8CKIT-145 Parasitic Capacitance (Cp) | CY8CKIT-149 Parasitic Capacitance (Cp) |
|---|---|---|
| Sns0 | 13 pF | 9 pF |
| Sns1 | 12 pF | 8 pF |
| Sns2 | 14 pF | 9 pF |
| Sns3 | 14 pF | 9 pF |
| Sns4 | 14 pF | 8 pF |
| Sns5 | - | 8 pF |

### Step 2. Calculating Sense Clock Frequency

Calculate sense clock frequency using

$$Fs < \frac{1}{10 R_{SeriesTotal} Cp}$$ Equation 1

Where,

- $C_P$ is the sensor parasitic capacitance.

- $R_{SeriesTotal}$ is the total series-resistance. This includes the 500 Ω resistance of the internal switches, the recommended external series resistance of 560 Ω/ 2 kΩ (connected on PCB trace connecting sensor pad to the device pin), and trace resistance if using highly resistive materials (for example ITO or conductive ink); that is, a total of 1.06 kΩ/ 2.5 kΩ plus the trace resistance.

**Note:** The external series resistance value is different for both the kits.

Table 4 shows the sense clock frequency values for CY8CKIT-149 and CY8CKIT-145 kits calculated using Equation 1.

Table 4. Sense Clock Frequency for CY8CKIT-149 and CY8CKIT-145

| Kit | $R_{SeriesTotal}$ (kΩ) | $C_P$ (pF) | Maximum Sense Clock Frequency (kHz) | Actual Sense Clock Frequency (kHz) |
|---|---|---|---|---|
| CY8CKIT-149 | 2.5 | 9 | 4444 | 3000 |
| CY8CKIT-145 | 1.06 | 14 | 6500 | 6000 |

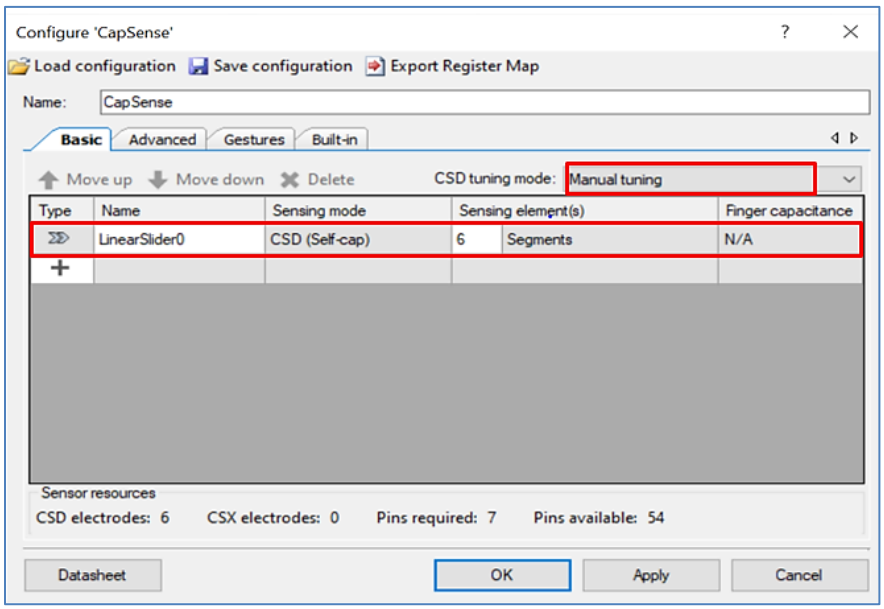
**Note:** Actual sense clock frequency corresponds to the nearest possible sense clock frequency supported by the device in the CapSense Configurator.

The calculated value, listed in Table 4, ensures maximum possible sense clock frequency (for good gain) while allowing the sensor capacitance to fully charge and discharge during each sense clock cycle.

### Step 3. Setting Initial Hardware Parameters

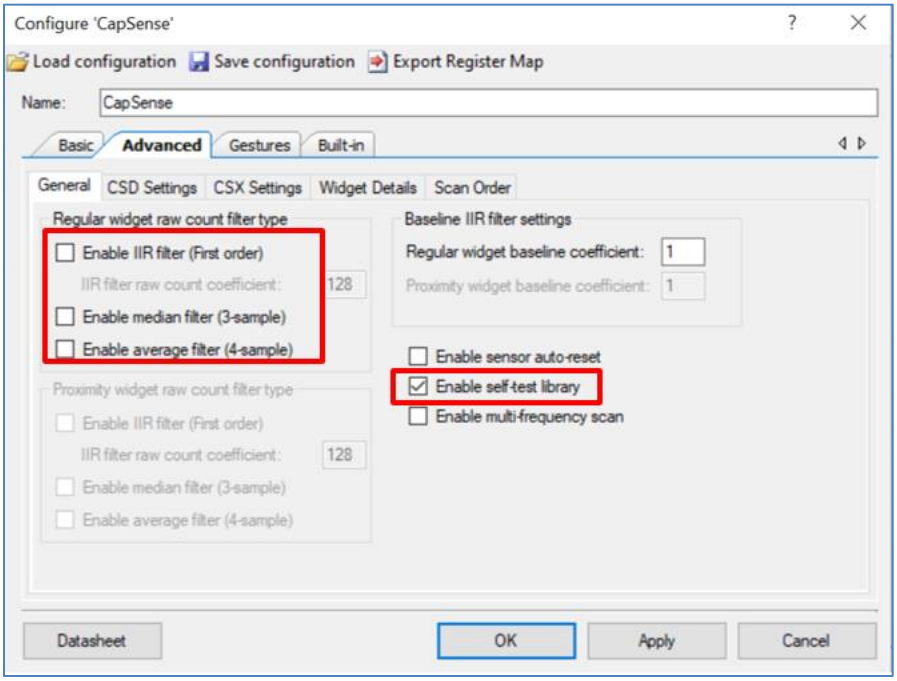
1. Plug the kit board into your computer's USB port as shown in Figure 15.
2. In the Workspace Explorer, double-click the *TopDesign.cysch* file to open the Schematic Editor window. Double-click the CapSense Component to open the Configure 'CapSense' window. In the **Basic** tab, note that a single slider **LinearSlider0** is configured as a **CSD (Self-cap)** and the **CSD tuning mode** is configured as **Manual tuning**.

Figure 4. CapSense Component - Basic Tab




3. Go to the **Advanced** tab, and then the **General** tab.
   - ☐ Select **Enable self-test library**.
   - ☐ Retain the default settings for all filters. You might later enable the filters depending on the signal-to-noise ratio (SNR) requirements in Step 5.

Figure 5. CapSense Component – General Settings of Advanced Tab




4. Go to the **CSD Settings** tab. Make the following changes:

   ☐ Set **Modular clock frequency (kHz)** as **48000** (that is, the maximum available in the drop-down list).

   **Note:** You can change **Modulator clock frequency** to **48,000** kHz only after changing the IMO clock frequency to 48 MHz. Under **Design Wide Resources**, click **Clocks** > **System IMO**. Select **48 MHz** from the **IMO** drop-down list.

   ☐ Select **Auto** from the **Sense clock source** drop-down list**.**

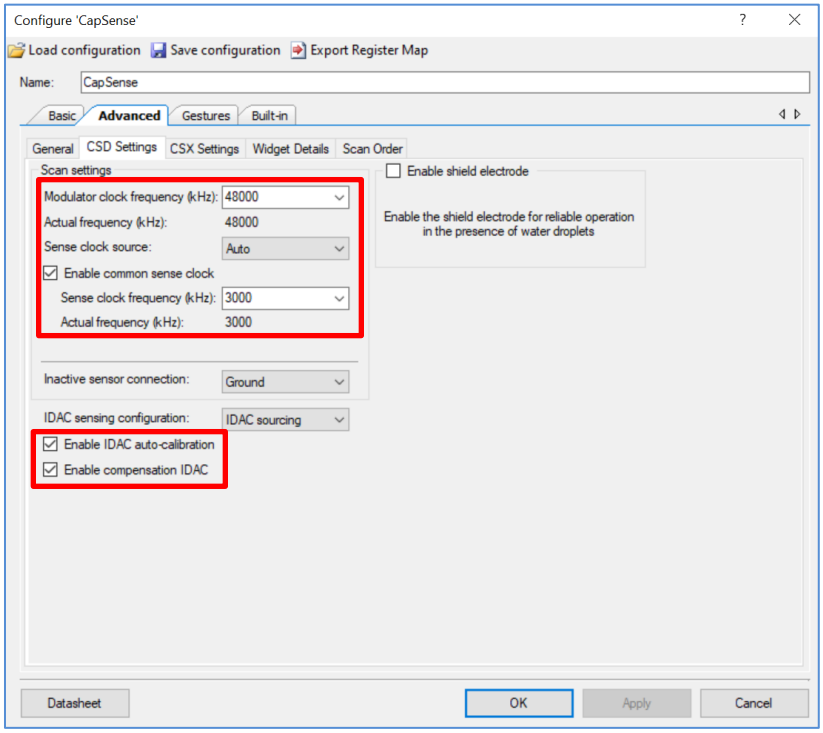
   ☐ Check **Enable common sense clock**.

   If the electrode capacitance of all CSD widgets is the same, using the common sense clock for all CSD widgets will result in lower power consumption and optimized memory usage; hence, it is the recommended setting for CSD widgets.

   ☐ Select **Enable IDAC auto-calibration** and **Enable Compensation IDAC**.

   This helps in achieving the required IDAC calibration levels for all segments in the widget while maintaining same sensitivity across segments.

Figure 6. CSD Settings in Advanced Tab




5. Go to the **Widget Details** tab. Select **LinearSlider0 (CSD)** from the left pane. Set the following:

   ☐ **Sense clock frequency** – same value as the **Actual Sense Clock Frequency** calculated in Step 2 (see Table 4).

   ☐ **Scan resolution** value – **8 bits**.

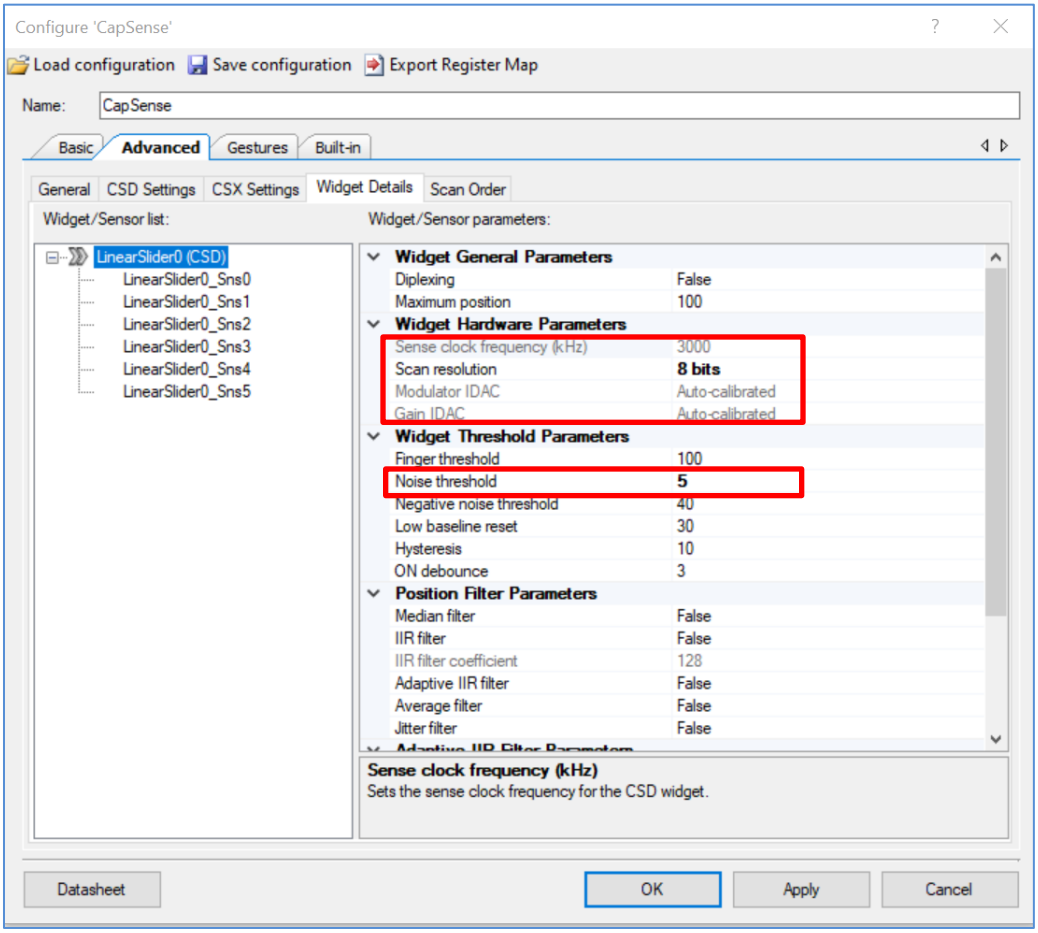
   8 bits is a good starting point to ensure fast scan time and sufficient signal. This value will be adjusted as needed in Step 5.

   ☐ **Noise Threshold** – **5**

   This reduces the influence of baseline on the sensor signal, which helps to get the true difference count. Retain the default values for all other threshold parameters; these parameters are set in Step 6.

6. Click **Apply** and **OK** to apply the settings.

Figure 7. Widget/Sensor Parameters in Advanced Tab




### Step 4. Obtaining Cross Over Points and Noise

Do the following to capture the difference count of all segments for setting the software parameters for the slider widget:

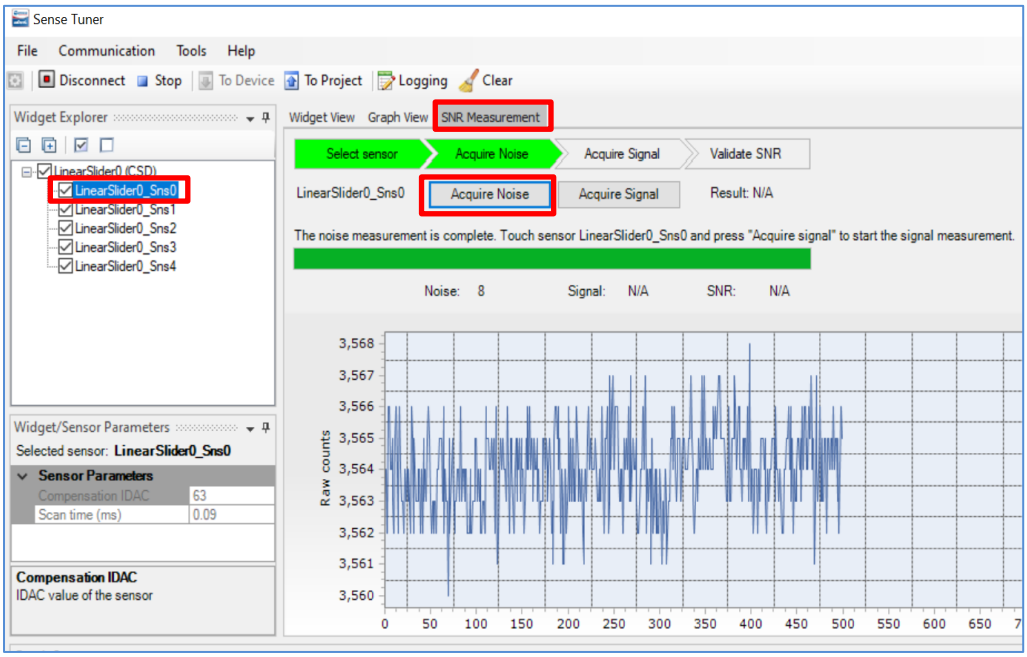
1. Program the device (select **Debug** > **Program**).
2. Right-click the CapSense Component and select **Launch Tuner** from the pop-up menu. The Sense Tuner window appears.
3. Capture and note the peak-to-peak noise of each segment of the sliders. Table 5 lists the values for each segment in CY8CKIT-149 and CY8CKIT-145.

Table 5. Peak-to-peak Noise Obtained for Each Segment in CY8CKIT-149 and CY8CKIT-145

| Slider Segment | Peak-to-peak Noise (CY8CKIT-149) | Peak-to-peak Noise (CY8CKIT-145) |
|---|---|---|
| Sns0 | 8 | 6 |
| Sns1 | 7 | 7 |
| Sns2 | 8 | 6 |
| Sns3 | 6 | 6 |
| Sns4 | 8 | 9 |
| Sns5 | 8 | - |

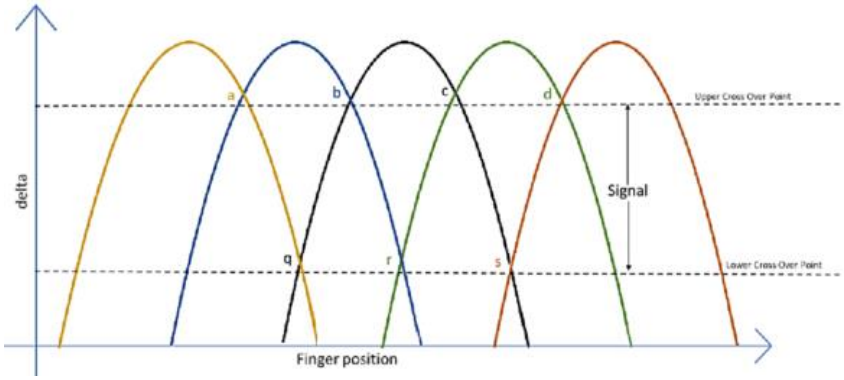
From the Widget Explorer section, select a sensor, for instance **LinearSlider0_Sns0**. Go to the **SNR Measurement** tab. Click **Acquire Noise** to capture the noise.

Figure 8. SNR Measurement Tab in Tuner Window




4. Use the metal finger (grounded) typically 8 mm or 9 mm and swipe it slowly at a constant speed from the start to end of the slider. Go to the **Graph View** tab to view a graph similar to Figure 9.Get the upper crossover point (UCP) and lower crossover point (LCP) as shown in Figure 10.

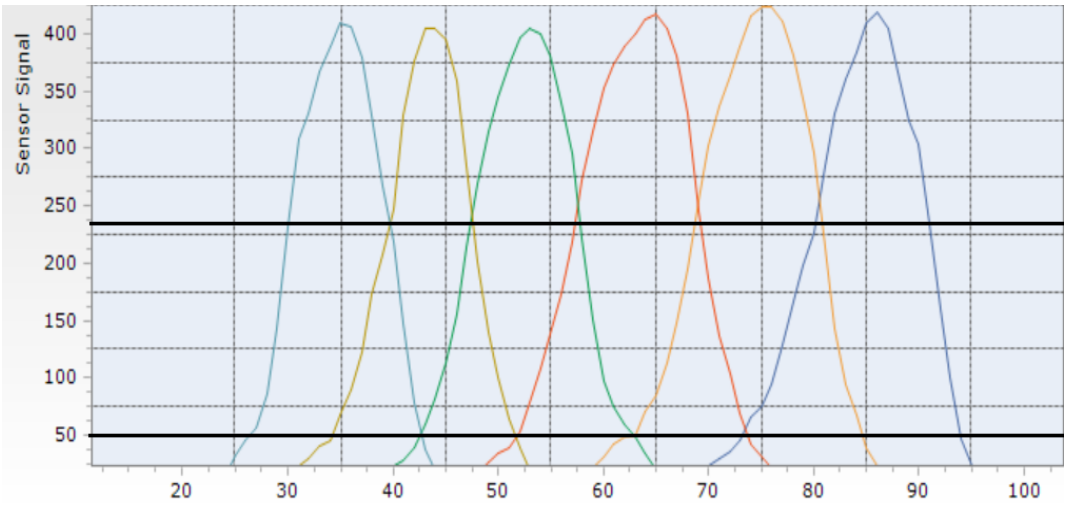
Figure 9. Difference Count (Delta) vs. Finger Position




Sensor signal values at points a, b, c, and d are expected to be at approximately the same level. If the values are slightly different, consider the lowest value as the UCP.

Sensor signal values at points q, r, and s are expected to be at approximately the same level. If the values are slightly different, consider the lowest value as the LCP.

Figure 10. Sensor Signal (Difference Counts) Displayed in Graph View tab




### Step 5. Using CapSense Tuner to Fine Tune Sensitivity for 5:1 SNR

The CapSense system may be required to work reliably in adverse conditions such as a noisy environment. The slider segments need to be tuned with SNR > 5:1 to avoid triggering false touches and to make sure that all intended touches are registered in these adverse conditions.

1. Make sure that all UCPs meet at least 5:1 SNR (using Equation 2) and all LCPs are greater than twice the peak-to-peak noise for all slider segments.

   In the Sensor Tuner window, you can increase the **Scan resolution** (located in the **Widget/Sensor Parameters** section, under **Widget Hardware Parameters)** by one, until you achieve this requirement.

$$SNR_N = \frac{Sensor\ signal_N}{Pk - Pk\ noise_N} \qquad \text{Equation 2}$$

2. After changing the **Scan resolution**, click **To Device** to send the setting to the device, and the change is reflected in the graphs.

   **Note:** The option **To Device** is enabled when the **Scan resolution** is changed.

3. If the SNR condition is not achieved even with the highest resolution, enable the filters in the General settings (go to the **Advanced** tab of the CapSense Component.

This is generally not required for these kits.

### Step 6. Using CapSense Tuner to Tune Threshold Parameters

After confirming that your design meets the timing parameters, and the SNR is greater than 5:1, set your threshold parameters.

1. Set the recommended thresholds values for the slider widget using the LCP and UCP obtained in Step 5:

   - ☐ Finger Threshold – 80% of UCP
   - ☐ Noise Threshold – LCP
   - ☐ Negative Noise Threshold – LCP
   - ☐ Hysteresis – 10% of UCP
   - ☐ ON Debounce – 3
   - ☐ Low Baseline Reset - 30

   Table 6 lists the threshold parameters obtained for both the kits.

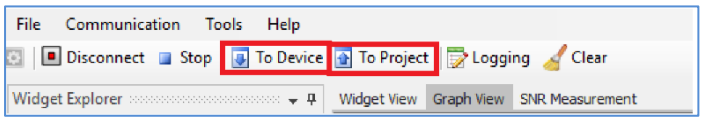
Table 6. Threshold Parameters Obtained for CY8CKIT-145 and CY8CKIT-149

| Parameter | CY8CKIT-145 | CY8CKIT-149 |
|---|---|---|
| Scan Resolution | 12 | 12 |
| Finger Threshold | 200 | 100 |
| Noise Threshold | 40 | 20 |
| Hysteresis | 25 | 10 |
| ON Debounce | 3 | 3 |
| Low Baseline Reset | 30 | 30 |
| Negative Noise Threshold | 40 | 20 |

## Applying Settings to Firmware

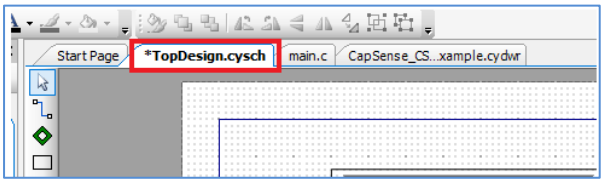
1. Click **To Device** and **To Project** in the Sense Tuner window to apply the settings to the device and project, respectively. Close the tuner.

Figure 11. Apply to Project



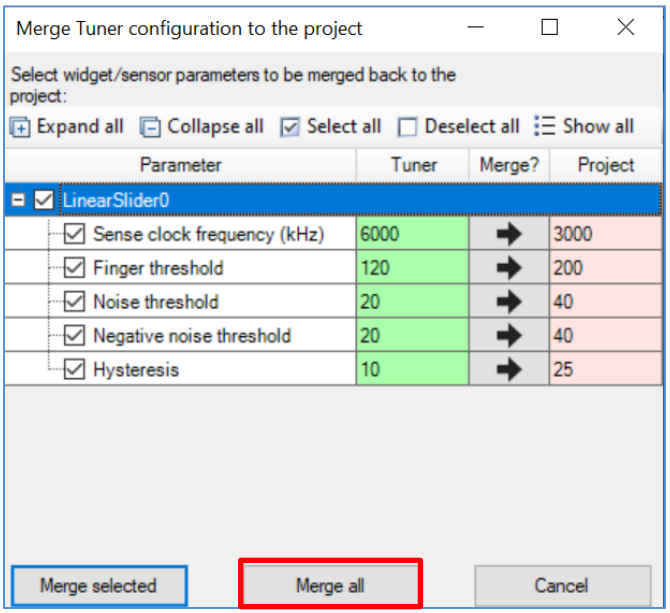

2. The * sign in the *TopDesign* file indicates that TopDesign has been updated.

Figure 12. TopDesign Updated when Settings are Sent to Project



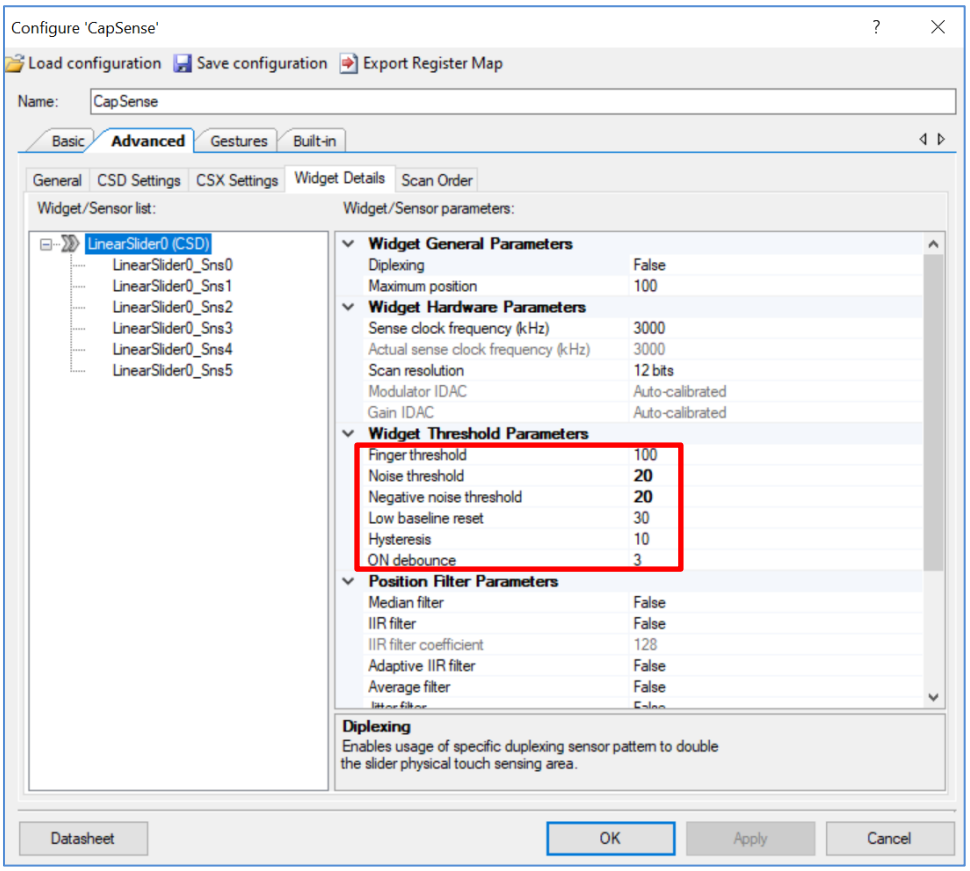

3. Double-click the CapSense Component to open the Configuration window. Click **Merge all** to accept the changes applied by the CapSense Tuner. Click **OK** to close the component configuration window.

Figure 13. Accept Changes to the Parameters



The changes are reflected in the CapSense Component as shown in Figure 14.

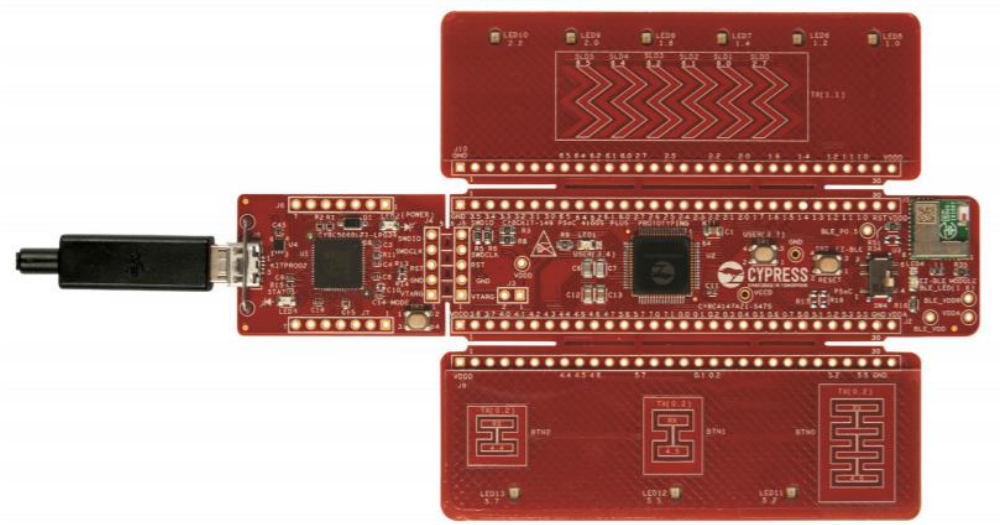
Figure 14. CapSense Component LinearSlider0 CSD Widget Details for CY8CKIT-149




## Testing the Basic Operation

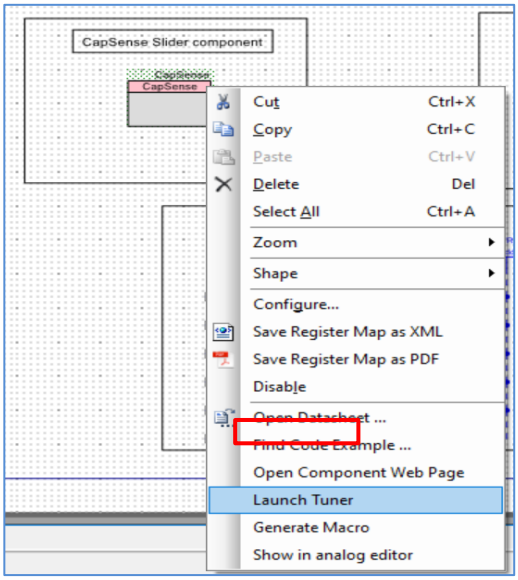
1. Power the device by plugging a USB 2.0 Type A to Micro-B Cable on J8 (USB Micro-B connector) as shown in Figure 15.

Figure 15. Connecting the PS0C 4100S Plus to a Computer



2. To use CY8CKIT-145, make sure to do the necessary settings as specified in Hardware Setup section and change the Pin Assignment as shown in Table 2.
3. Build the code example by selecting **Build** > **Build CE229521 – PSoC 4 CapSense CSD Slider Tuning** in PSoC Creator.
4. Program the device (select **Debug** > **Program**).
5. Slide your finger over the CapSense linear slider.
6. To monitor CapSense raw data using the Sense Tuner GUI, right-click the CapSense Component and select **Launch Tuner** from the pop-up.

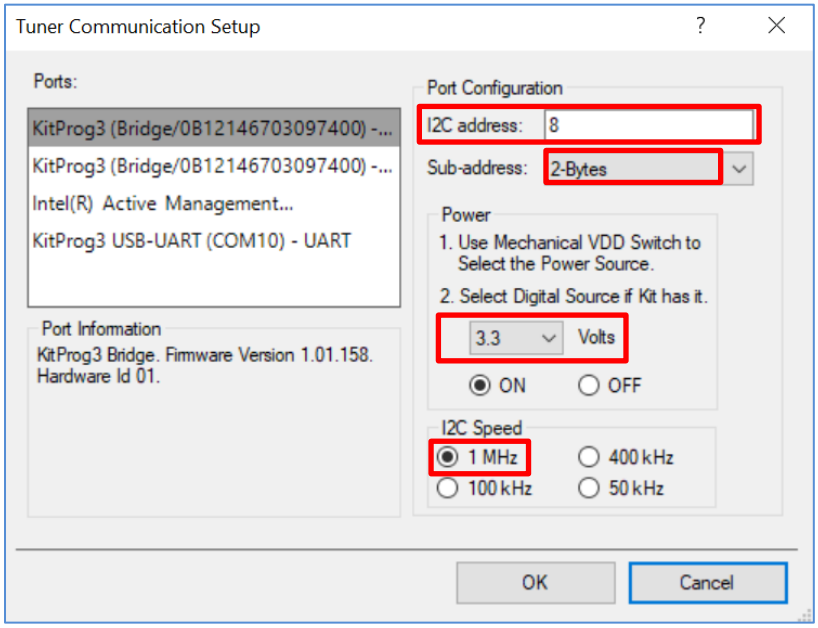
Figure 16. Launch Tuner GUI using CapSense Component




7. The Sense Tuner dialog appears. Follow the menu path **Tools** > **Tuner Communication Setup** or click the **Tuner Communication Setup** icon to open the Tuner Communication Setup dialog.
8. Select the appropriate $I^2C$ communication device and set the following parameters:
   - ☐ **I2C address** – **8**
   - ☐ **Sub-address** – **2-Bytes**
   - ☐ **Power** – **3.3** Volts
   - ☐ **I2C Speed** – **1 MHz**

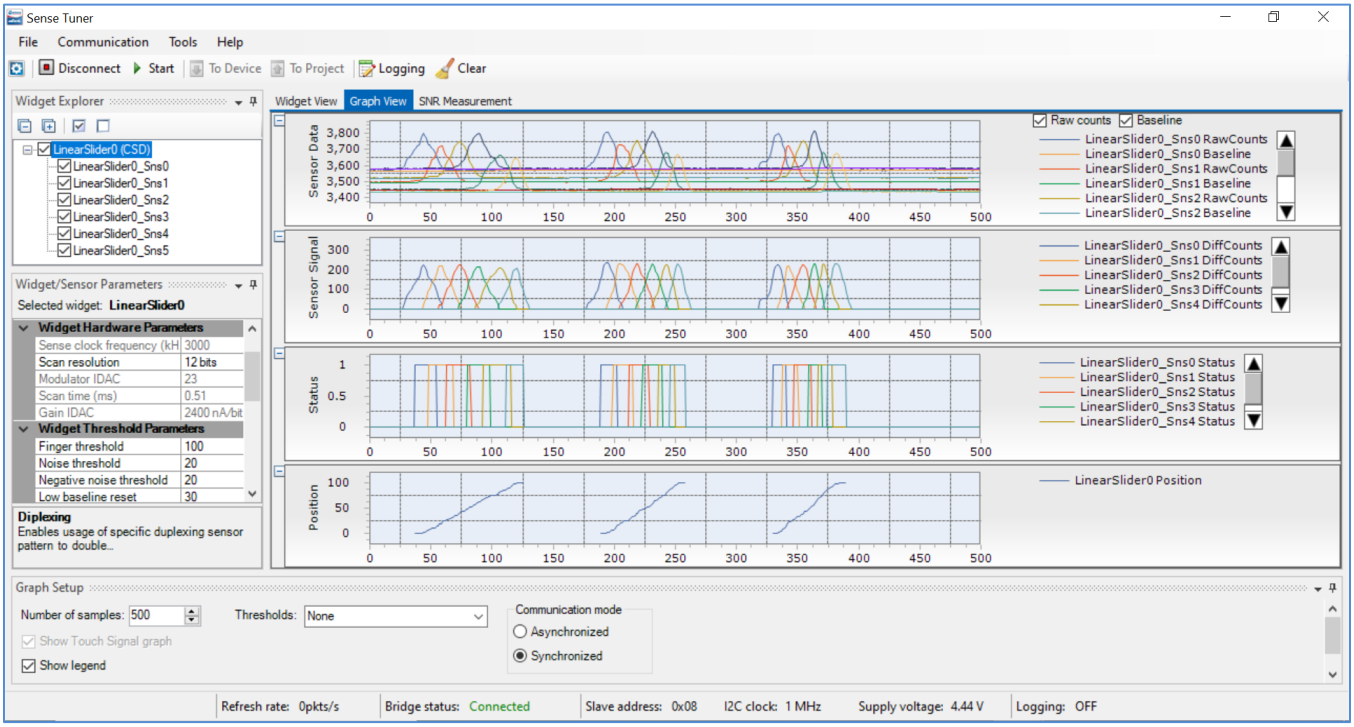
   These are the same values set in the EZI2C Component.
9. Click **OK**.

Figure 17. Tuner Communication Setup Parameters



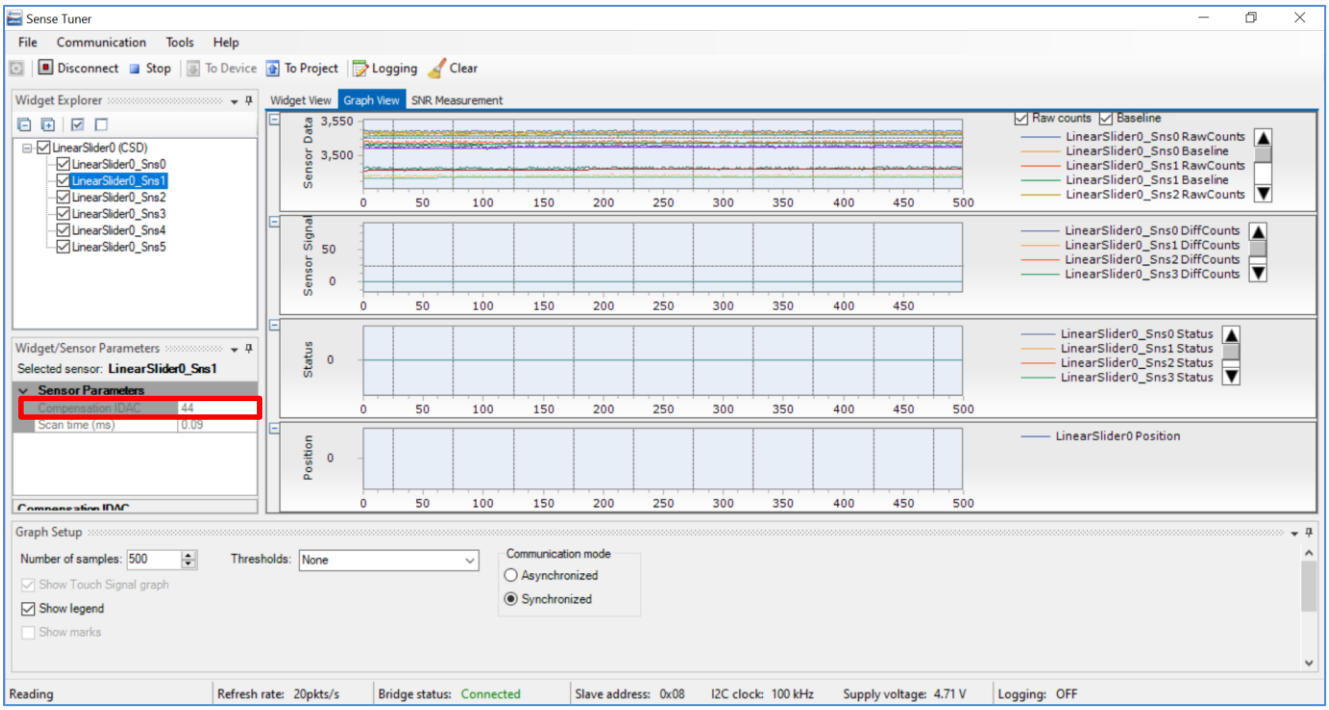

10. Click **Connect** to establish connection.
11. Click **Start** to start data streaming from the device. The Tuner GUI displays data from the sensor in the **Widget View** and **Graph View** tabs. For details on Tuner GUI, see the CapSense Tuner section of the CapSense Component datasheet. Set the **Communication Mode** to Synchronized mode. The **Graph View** tab shows the raw count, difference count, position, and baselines for each segment.

Figure 18. Graph View of Sense Tuner



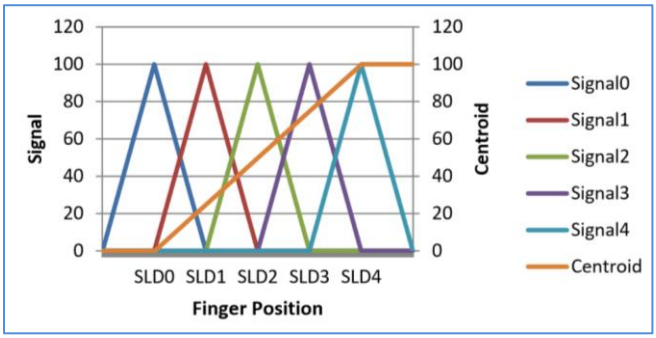
12. Observe the **Widget/Sensor Parameters** section in the Sense Tuner window. The **Compensation IDAC** values for each of the slider segment calculated by the CapSense Component is displayed as shown in Figure 19.

Figure 19. IDAC Values for the CSD Slider Widget




The position graph obtained must be linear with no flat spots ensuring that the slider has been tuned to have a linear response as shown in Figure 20.

Figure 20. Response of Centroid vs. Finger Location when Signals of All Slider Elements Are Equal




## Component Settings

The step-by-step instructions to configure the CapSense Component is discussed Operation.

### I2C Component

Figure 21 shows the settings for the EZI2C Component.

Figure 21. EZI2C Slave Component's Basic Tab

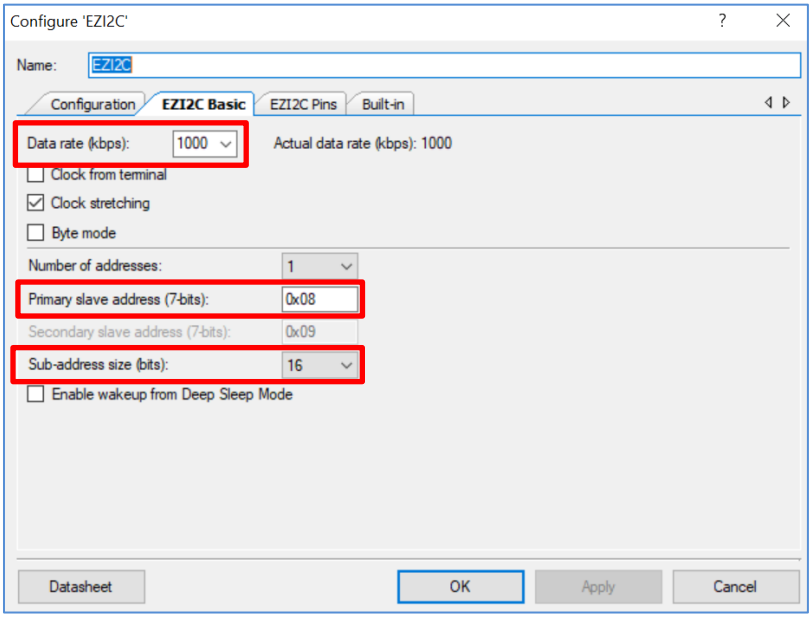


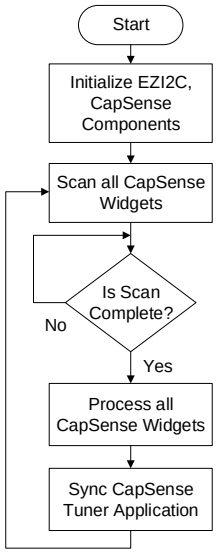

Table 7 lists the PSoC Creator Components used in this example, how they are used in the design, and the non-default settings required so they function as intended.

Table 7. List of PSoC Creator Components in PSoC 4 CapSense Slider Code Example

| Component | Instance Name | Purpose | Non-default Settings |
|---|---|---|---|
| CapSense | CapSense | Detect finger touch on the CapSense slider | See Figure 4 to Figure 7 |
| EZI2C Slave (SCB mode) | EZI2C | I2C for tuning and debug | See Figure 21 |

## Firmware Explanation

Figure 22. Firmware Flowchart



## Reusing This Example

This example is designed for CY8CKIT-149 PSoC® 4100S Plus Prototyping Kit. To port the design to a different PSoC 4 MCU device and/or kit, change the target device using the Device Selector and pin assignments in the Design Wide Resources Pins settings as needed.

## 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 |
| **Code Examples** | | |
| CE210709 | CapSense Linear Slider and Buttons | This code example demonstrates the implementation of CapSense® linear slider and buttons and LED control. |
| **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 |
| 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: CE229521 – PSoC 4 CapSense CSD Slider Tuning

Document Number: 002-29521

| Revision | ECN | Submission Date | Description of Change |
|---|---|---|---|
| ** | 6864447 | 04/24/2020 | New code example |
| *A | 6939003 | 07/31/2020 | Added a code example in the Related documents<br>Updated Figure 9<br>Changed default compiler settings in the project.<br>Updated source code |

## 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
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2020. 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.