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.

CE228931 – PSoC™ 4 CapSense CSX Button Tuning

Objective

This code example demonstrates how to manually tune a mutual capacitance CSX-based button widget in PSoC™ 4 devices using CAPSENSE™ Tuner.

Requirements

Tools: PSoC Creator 4.4

Programming Language: C (Arm® GCC 5.4.1)

Associated Parts: All PSoC 4 MCU parts that supports CapSense CSX sensing.

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 CSX-based button using CapSense Tuner.

This document details the following:

1. A high-level overview of the CSX widget tuning flow
2. An example to show how to manually tune a CSX button widget.
3. How to use the CapSense Tuner to monitor the CapSense raw data and fine-tune the CSX button for optimum performance such as reliability, power consumption, and response time

The code scans a single button widget using the CSX sensing method. It also sends the CapSense raw data over an I2C interface to the on-board KitProg, which then could be received using the CapSense Tuner. The tuning of the CSX button is performed with the help of data that is received in the CapSense Tuner. The successful tuning of the button is indicated by a user LED in the kit, the LED is turned ‘ON’ when the finger touches the button and turned ‘OFF’ when the finger is lifted from the CSX button.

Hardware Setup

The example project is set up for CY8CKIT-145-40XX PSoC 4000S CapSense 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.

Table 1. Supported Development Kits

Development Kit	Series	Device
CY8CKIT-149	PSoC 4100S Plus	CY8C4147AZI-S475
CY8CKIT-145-40XX	PSoC 4000S	CY8C4045AZI-S413

The pin assignments for the supported kits are provided in Table 2. For these kits, the project includes control files to automatically assign pins with respect to the kit hardware connections during the project build. 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 Assignments

Pin Name	CY8CKIT-145-40XX PSoC 4000S CapSense Prototyping Kit	CY8CKIT-149 PSoC 4100S Plus Prototyping Kit
CapSense:CintA	P4[2]	P4[2]
CapSense:CintB	P4[3]	P4[3]
CapSense:Rx (BTN0_Rx0)	P1[4]	P4[6]
CapSense:Tx (BTN0_Tx)	P1[3]	P0[2]
EZI2C:scl	P1[0]	P3[0]
EZI2C:sda	P1[1]	P3[1]
LED_Output	P3[4]	P5[2]

Software Setup

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

Pre-requisites

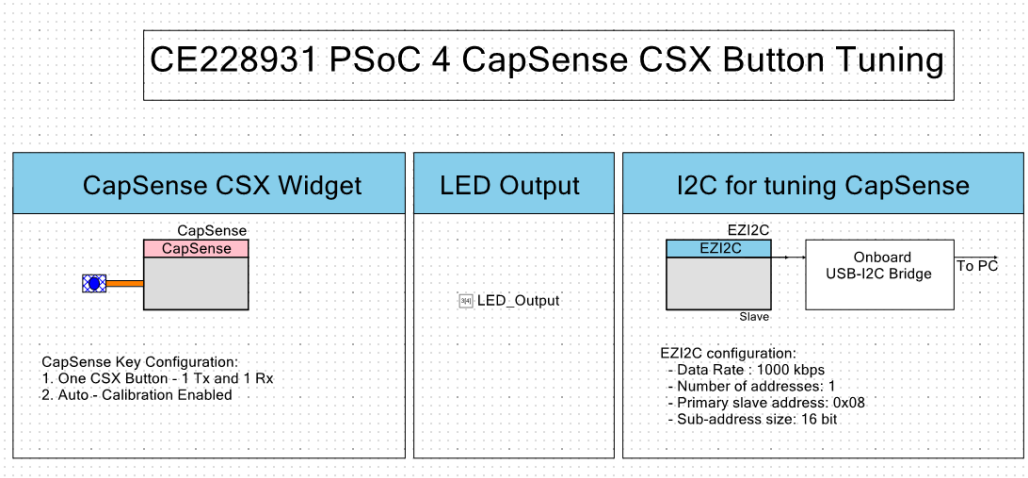
The user of this code example is assumed to be familiar with CapSense technology and basic terms such as raw counts, baseline, and difference counts. If you are new to CapSense, see the application note AN85951 – PSoC 4 and PSoC 6 MCU CapSense Design Guide.

Design and Implementation

This code example has a single workspace, CE228931.

Figure 1 shows the PSoC Creator schematics for the “CE228931_PSoC4_CapSense_CSX_Tuning” code example. This code example uses CapSense and SCB (configured as EZI2C slave) Components.

Figure 1. PSoC Creator Schematics



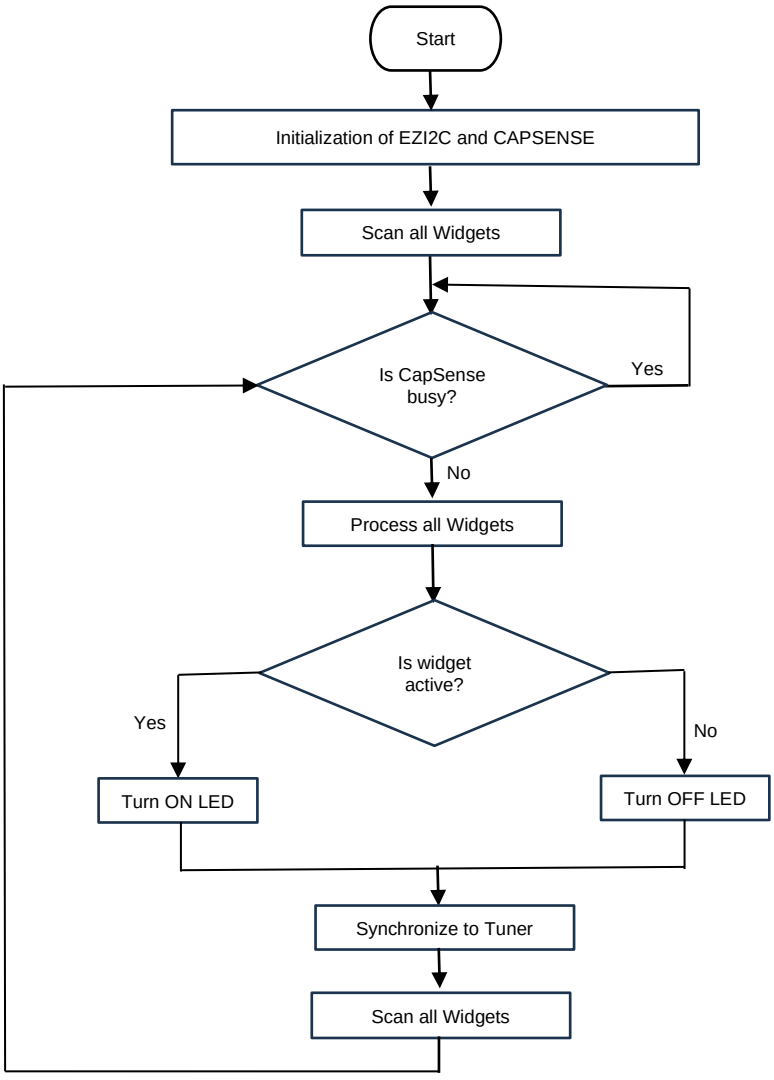
The project’s top design contains a CapSense Component that has a single button widget configured in CSX sensing mode. See the section “CapSense CSX Sensing Method” in the PSoC 4 and PSoC 6 MCU CapSense Design Guide. See the Operation section for step-by-step instructions to configure other settings of the CapSense Component.

The CapSense Component provides a GUI-based tuner application for debugging and tuning the CapSense system. The CapSense Tuner application works with the EZI2C and UART Communication Components. This project has an SCB block configured in EZI2C mode to establish communication with the on-board KitProg, which in turn enables reading the CapSense raw data by the CapSense Tuner. See Configure EZI2C Component.

The CapSense data structure that contains the CapSense raw data is exposed to the CapSense Tuner by setting up the I2C communication data buffer with the CapSense data structure. This enables the CapSense Tuner to access the raw data for tuning and debugging.

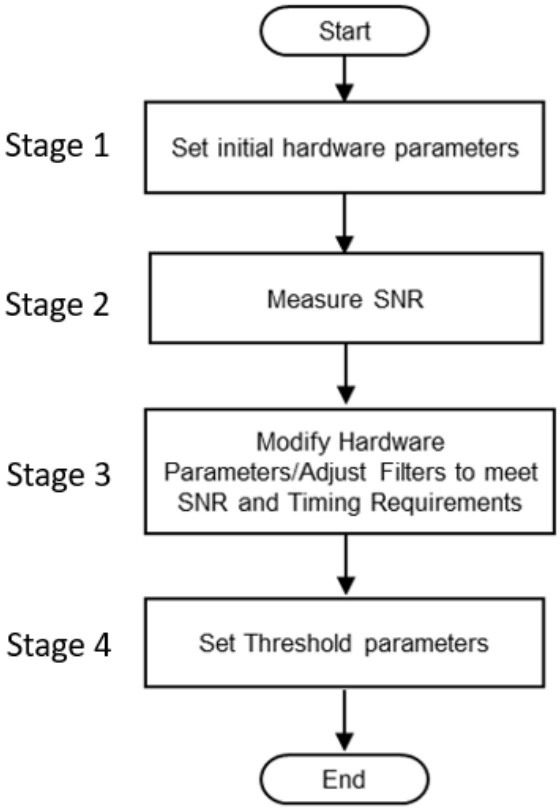
The Figure 2 below gives the firmware flow of this project.

Figure 2. Firmware Flow



Tuning Flow Summary

The following flowchart gives a high-level summary on how to tune a CSX-based CapSense button in PSoC 4 devices. See the “Manual Tuning” section in AN85951 – PSoC 4 and PSoC 6 MCU CapSense Design Guide for information on the hardware and threshold parameters that determines the CapSense touch performance.



To manually tune CSX button, see the Operation section.

Operation

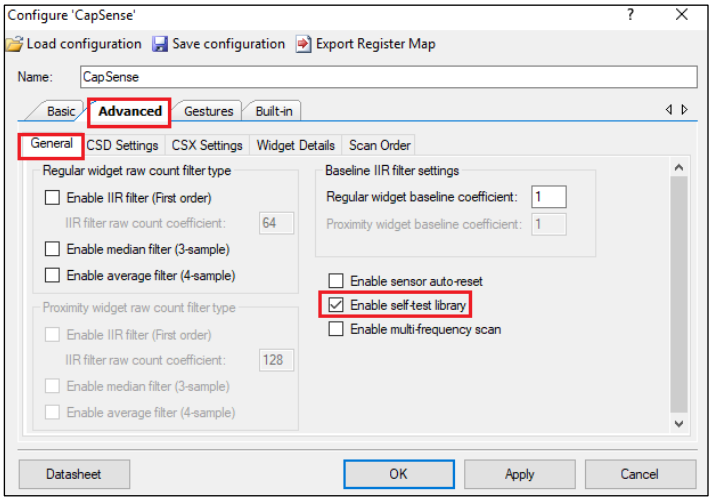
This process involves the following stages:

1. Stage 1. Set Initial Hardware Parameters
2. Stage 2. Measure SNR
3. Stage 3. Modify Hardware Parameters or Adjust Filter Settings
4. Stage 4. Set Threshold Parameters

Stage 1. Set Initial Hardware Parameters

1. Connect the board to your PC using the provided micro-USB cable through the KitProg2 USB connector.
2. In the **Workspace Explorer**, double-click on the *TopDesign.cysch* file to open the schematic editor window. Double-click the CapSense Component, or right-click the CapSense Component and select **Configure** to open the CapSense Component configuration window.
3. In the **Basics** tab, you will find a single widget 'BTN0' configured as a CSX button.
4. Navigate to the **Advanced** tab and then select the **General** settings sub-tab. Leave all the filter parameters at their default settings. Filters will be enabled depending on the SNR and system time requirements in Stage 3. Modify Hardware Parameters/Adjust Filters to meet SNR and Timing Requirements. Click **Enable self-test library** checkbox as shown in Figure 3 to perform sensors capacitance measurement as explained in section Calculate the Maximum Tx Clock Frequency.

Figure 3. Advanced Tab - General Settings



5. Select **Advanced** tab and the select the **CSX Settings** sub-tab and configure the parameters as Table 3 and Figure 4 show.

Figure 4. Advanced - CSX Settings

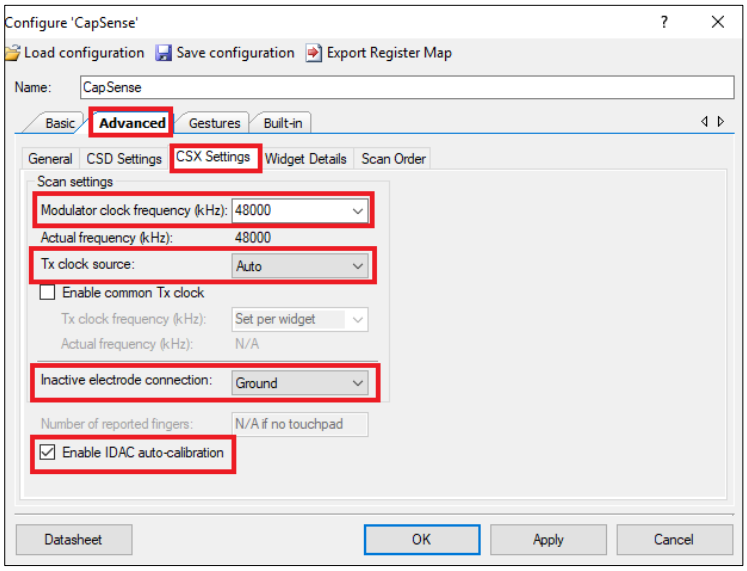


Table 3 CSX Settings

Parameter	Value	Remarks
Modulator clock frequency (kHz)	Maximum allowed by the selected device	A higher modulator clock frequency reduces Flat spot while increasing the measurement accuracy and sensitivity. Thus, it is recommended to select the highest possible available modulator clock frequency.
Tx clock source	Auto	Enabling Auto (chooses between spread spectrum clock or direct clock). The SSCx clock source option is available only for PSoC 4 S-Series, PSoC 4100S Plus, and PSoC 4100PS. It helps to deal with EMC/EMI. For other devices, the Tx clock is always Direct.
Enable common Tx clock	Unchecked	When selected, all CSX widgets share the same Tx clock with the frequency specified in the Tx clock frequency (kHz) parameter. Otherwise, the Tx clock frequency is entered separately for each CSX widget in the Widget Details tab. If the electrode capacitance of all the CSX widgets is the same, using the common Tx clock for all CSX widgets results in a lower power consumption and optimized memory usage; it is the recommended setting for CSX widgets. However, if the electrode capacitance is significantly different for each widget, a common Tx clock may not produce the optimal performance.
Inactive electrode connection	Ground	Inactive sensors are connected to ground to provide good shielding from noise sources.
Enable IDAC auto-calibration	Checked	Enabling auto-calibration allows the device to automatically choose the optimal IDAC calibration point (for CSX, this is 40 percent of the maximum value).

Note: The Modulator Clock Frequency can be changed to 48 MHz only after the IMO clock is changed to 48 MHz. Under **Design Wide Resources**, click **Clocks** > **System IMO**. Select the drop down on the IMO Clock and select **48 MHz**.

6. Set the Tx clock frequency:

   a. Navigate to the **Advanced** tab and the select **Widget Details** window.

   b. Set the Tx clock frequency in such a way that it completely charges and discharges the sensor parasitic capacitance for maximum sensitivity. It can be verified by checking the signal in an oscilloscope or it can be set using the Equation 1.

Equation 1. Condition for Selecting Tx Clock Frequency

$$F_{Tx} < Min(\frac{1}{10R_{SeriesTx}C_{PTx}}, \frac{1}{10R_{SeriesRx}C_{PRx}})$$

C_{PTx} and C_{PRx} are the parasitic capacitances of the Tx electrode and Rx electrode respectively. $R_{SeriesTx}$ and $R_{SeriesRx}$ are the recommended external series resistances (connected on the PCB trace connecting the sensor pad to the device pin), and trace resistance if using highly resistive materials (for example, ITO or conductive ink).

Calculate the Maximum Tx Clock Frequency

Note: While using this procedure, ensure that you have enabled the **Enable self-test library** option in the CapSense Component. After you obtained the Cp value, you can disable this option.

i) Estimate the parasitic capacitance (Cp) of Tx and Rx electrodes. Use the `CapSense_GetSensorCapacitance()` function to measure the Cp of Tx and Rx electrodes of the CSX button. The C_P can be measured using an LCR meter also.

Note: The BIST feature is available only from CapSense Component v3.10 and above.

ii) Program the board in Debug mode. Click the **Debug** icon or press **F5** to program in debug mode. For more details, see Programming and Debugging section in PSoC Creator User Guide

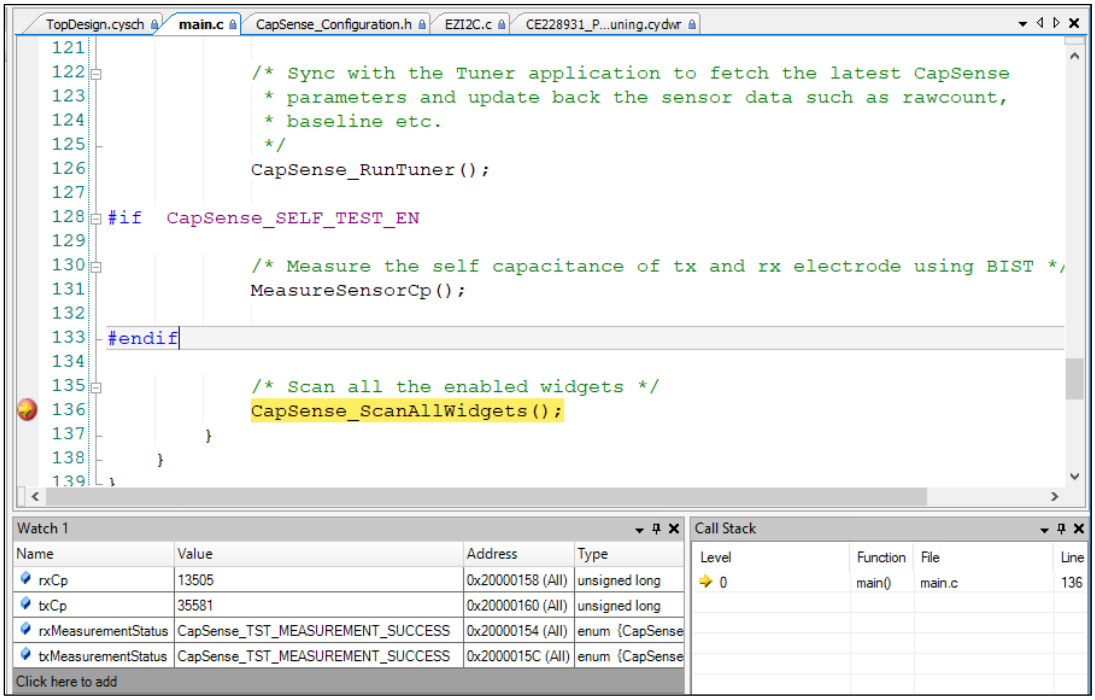
iii) Place a breakpoint after capacitance measurement.

iv) In the watch window, add the Cp measurement variables *rxCp* and *txCp*.

v) Click the **resume execution** icon (green arrow) to reach the breakpoint. Note that the function argument *measurementStatus* reads **CapSense_TST_MEASUREMENT_SUCCESS** and the function returns the capacitance of the sensor elements in femtofarads (fF) as shown in Figure 5.

vi) Click the Stop Debugging icon (blue box) to exit Debug mode.

Figure 5. Cp Measurement Using BIST



c. Using BIST, the C_P of the Tx and Rx electrodes were estimated to be as shown in Table 4. Set the Tx Clock frequency in the CapSense Component to the nearest possible Tx clock frequency as supported by the device. The maximum value of F_{Tx} depends on the selected device. For the PSoC 4 S-Series, PSoC 4100S Plus, and PSoC 4100PS family of devices, the maximum F_{Tx} is 3000 kHz; for other PSoC 4 devices, it is 300 kHz.

Table 4. Cp of Tx Electrode, Calculated Maximum Frequency and Tx Frequency Setting in Configurator

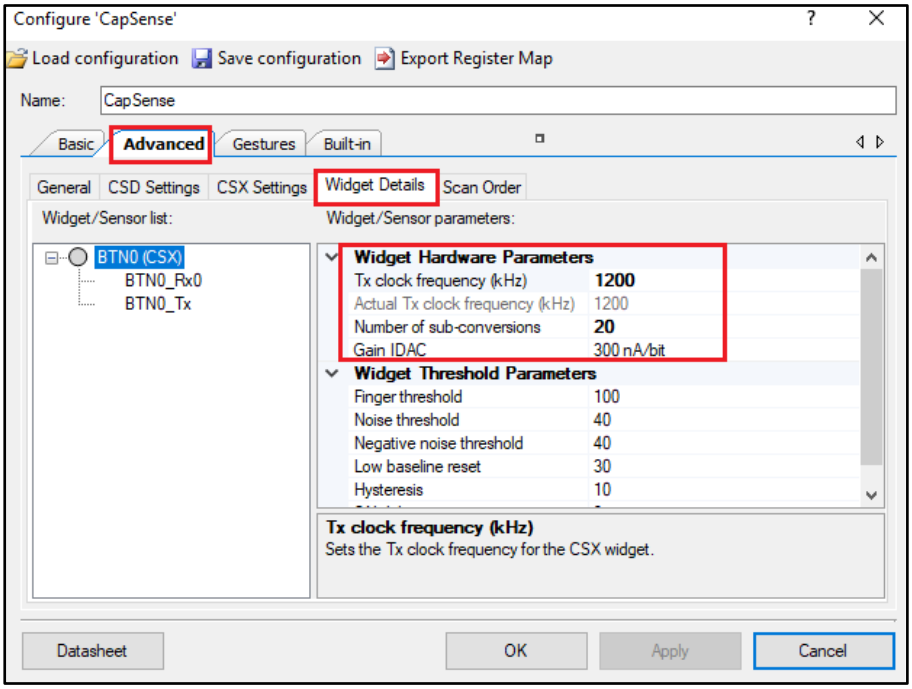
Development kit	C_P of BTN0_Tx Electrode (pF)	C_P of BTN0_Rx0 Electrode (pF)	External Resistance (Ω)	Calculated Maximum Frequency (kHz)	Tx Clock Frequency (kHz) setting in Configurator
CY8CKIT-149	36	13	2000	1388	1200
CY8CKIT-145	22	12	560	8116	3000

d. Ensure the following in addition to this condition:

- The auto-calibrated IDAC value should lie in the mid-range (for example, 30 - 90) for the selected F_{Tx}. If the auto-calibrated IDAC value lies out of the recommended range, the Gain IDAC and F_{Tx} is tuned such that IDAC falls in the recommended range as explained in section Ensure that Auto-calibrated IDAC Falls in the Recommended Range.
- If you are using the SSCx clock source, ensure that you select the Tx clock frequency that meets the conditions mentioned in CapSense Component datasheet in addition to these conditions.

7. Set the **Number of sub-conversions** to an initial low value of 20 as Figure 6 shows. This will be modified in Stage 3. Modify Hardware Parameters/Adjust Filters to meet SNR and Timing Requirements based on Signal-to-Noise ratio (SNR) and system timing requirements. Leave all other values in this tab to their default settings.

Figure 6. Advanced Tab - Widget Details Sub-Tab



Note: CapSense initialization may fail at this stage. Ensure that CapSense initialization passes after the final tuning parameters are set.

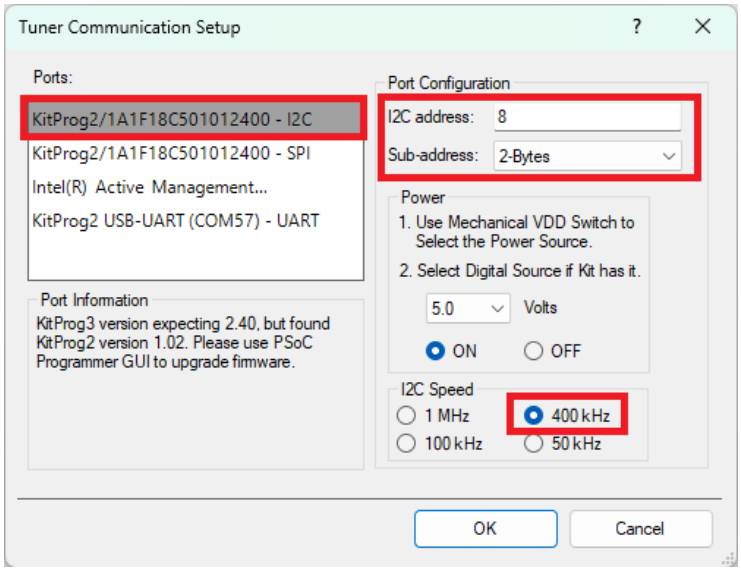
8. Build and program the project onto the kit. Select **Debug** > **Program.** This builds the project and then programs your kit.

Stage 2. Measure SNR

Set up CapSense Tuner to View Sensor Data

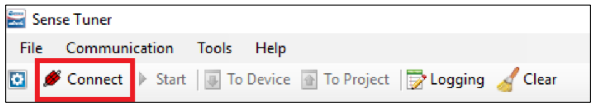
1. In the *TopDesign.cysch* file, right-click the **CapSense** Component and select **Launch Tuner**.
2. Go to **Tools** > **Tuner Communication Setup**… and set the parameters as Figure 7 shows and click **OK**.

Figure 7. Tuner Communication Setup



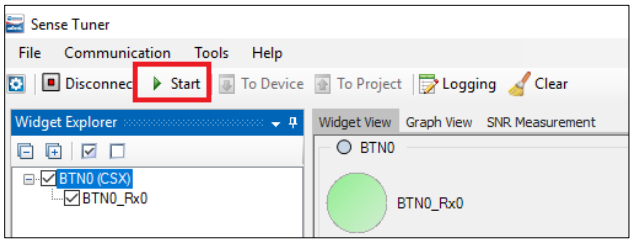
3. Click **Connect** as Figure 8 shows.

Figure 8. CapSense Tuner Window



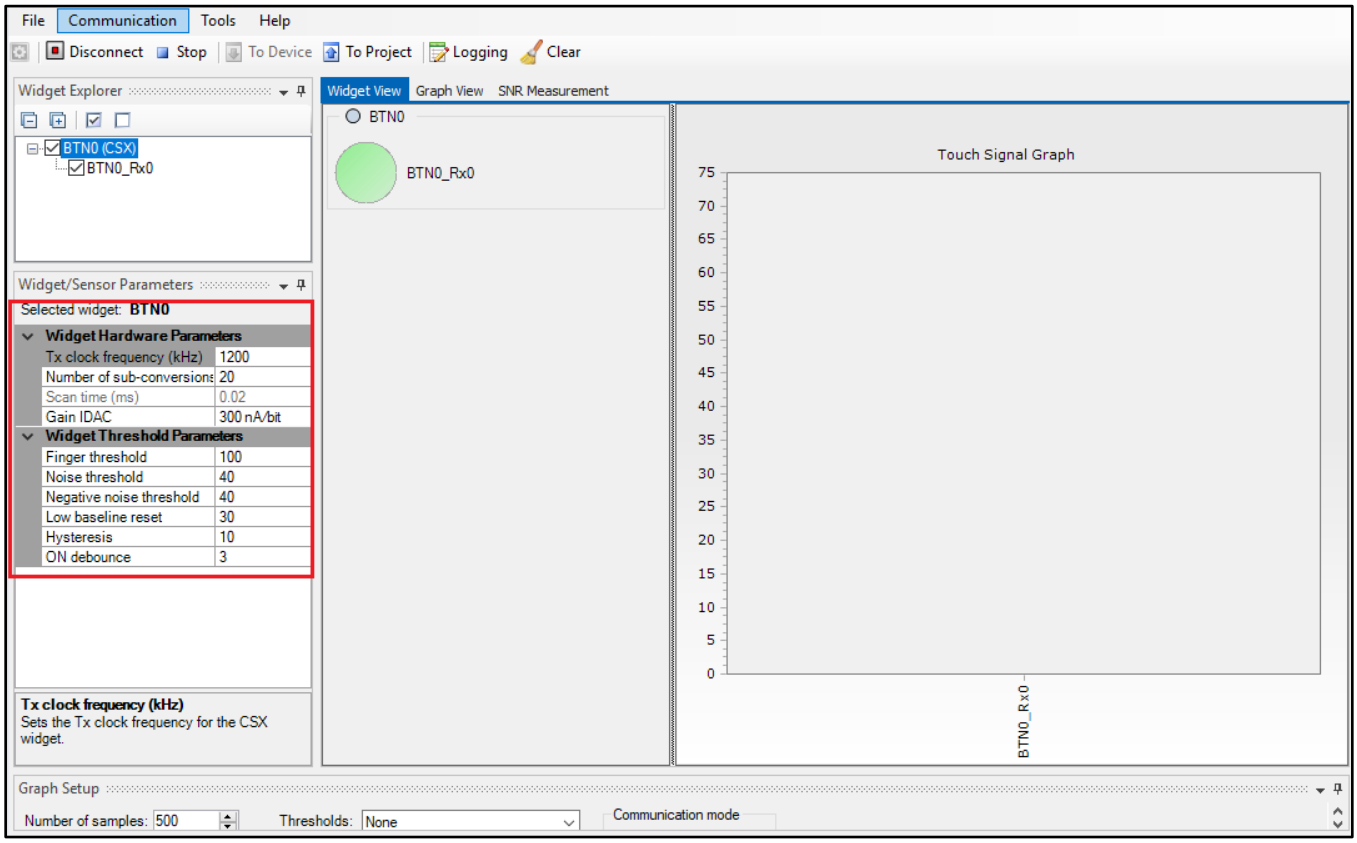
4. Click **Start** as Figure 9 shows.

Figure 9. Starting Tuning with CapSense Tuner



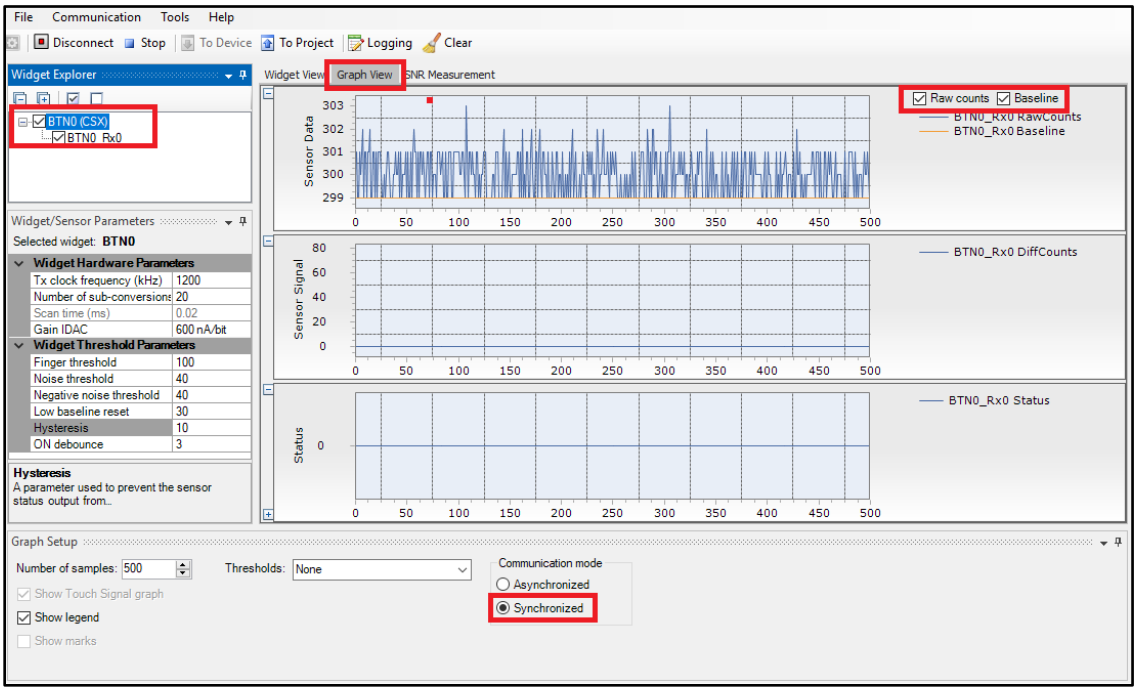
The Widget/Sensor Parameters tab gets updated with the parameters configured in the CapSense Component as Figure 10 shows.

Figure 10. Widget/Sensor Parameters Updated in Sense Tuner



5. Select the **BTN0_Rx0** check box, select **Synchronized** under **Communication mode**, and then navigate to the Graph View as Figure 11 shows. The graph view displays the raw counts and baseline for BTN0_Rx0 in the Sensor Data window. Ensure that the **Raw Counts** and **Baseline** checkboxes are selected to view the sensor data.

Figure 11. Graph View of Tuner GUI

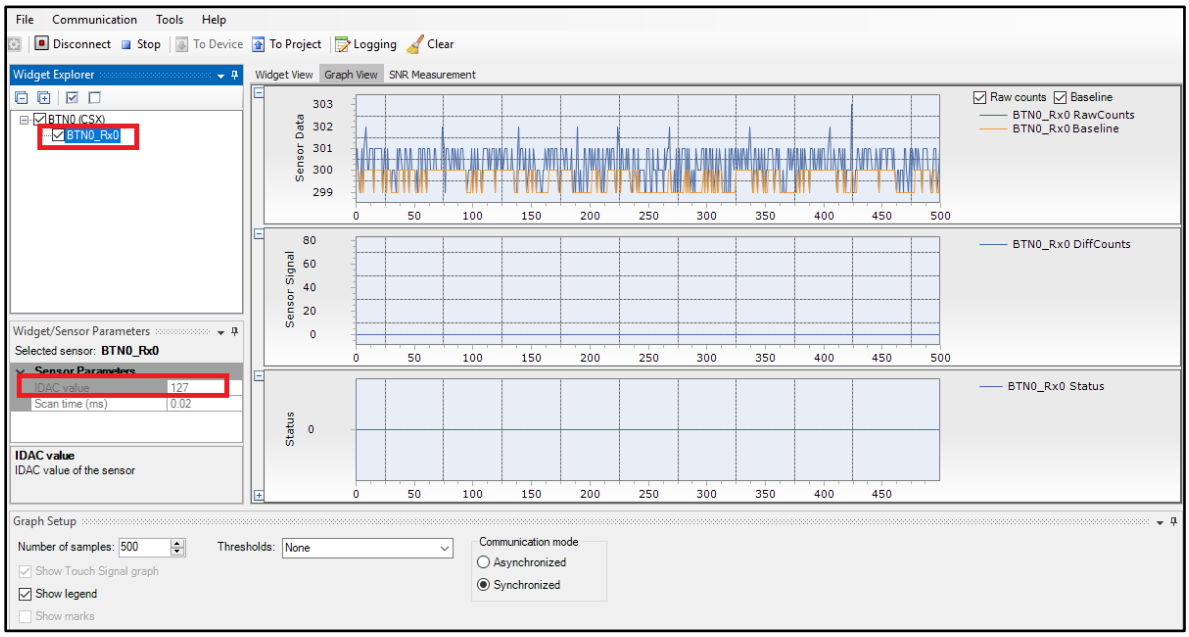


Note: At this point, when the configured button is touched, you may or may not notice the touch signal in the **Sensor Signal graph**. There are chances that the sensor may false trigger, which can be seen in the touch status going from 0 to 1 in the Status window.

Ensure that Auto-calibrated IDAC Falls in the Recommended Range

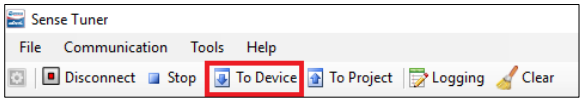
6. As discussed in step 6 of Stage 1. Set Initial Hardware Parameters, the Gain IDAC and Tx clock frequency will be tuned to bring the IDAC value to the recommended range in this step. Click on **BTN0_Rx0** in the Widget Explorer to view the IDAC value in the sensor parameters window as shown in Figure 12. If the IDAC value is within the range 30 to 90, skip to Step 8 (Measure SNR); if not, follow the next steps to modify the Gain IDAC until the IDAC value is in the desired range.

Figure 12. IDAC Value



7. Do the following to modify the Gain IDAC:
   a. Click on **BTN0(CSX)** in the Widget Explorer.
   b. Increase or decrease the Gain IDAC value in the **Widget Hardware Parameters** window. Decreasing the Gain IDAC value will increase the IDAC and vice versa.
   c. Click **To Device** to apply the changes to the device as Figure 13 shows.

Figure 13. Apply Changes to Device



   d. Observe the IDAC value in the **Sensor Parameters** section of Widget/Sensor Parameters window.

Repeat steps 7 a to 7 d until you obtain the IDAC value in the range of 30 to 90.

Note: If changing the Gain IDAC doesn't bring the IDAC value to the recommended range, set the Gain IDAC that is closest to achieving this IDAC range (30 to 90), and then modify the Tx clock frequency until the IDAC value falls within recommended range. Decreasing the Tx clock frequency will decrease IDAC value. The Tx Clock frequency should be modified in the **Widget/ hardware parameters** section of the CapSense Tuner.

After performing these steps, you will arrive at the following Gain IDAC and Tx clock frequency settings with the corresponding IDAC values for different development kits as shown in Table 5.

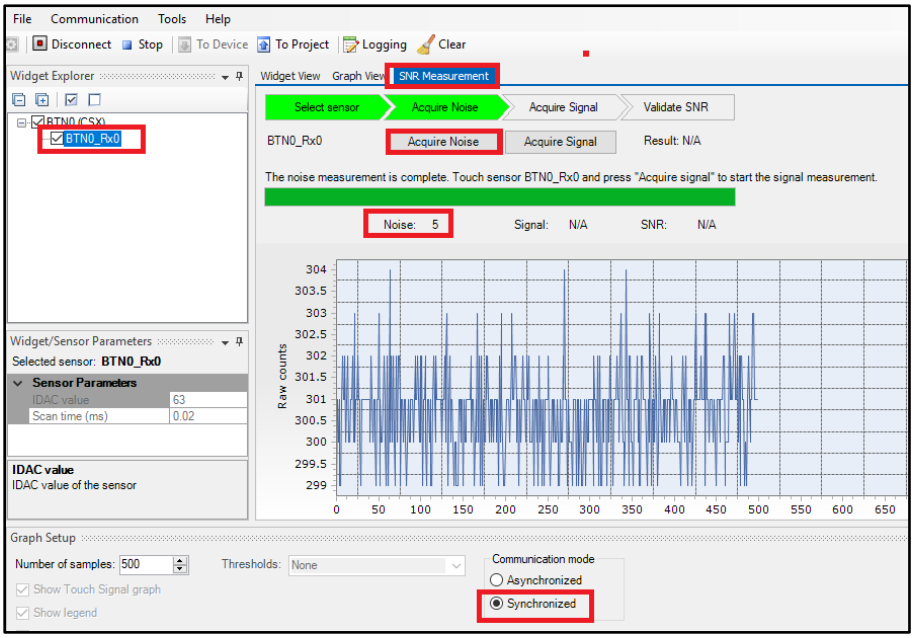
Table 5. Final Tx Clock Frequency and Gain IDAC Settings for Different Development Kits

Development Kit	Tx Clock Frequency (kHz)	Gain IDAC	IDAC value
CY8CKIT-149	1200	600 nA/bit	63
CY8CKIT-145	3000	2400 nA/bit	86

Measure SNR

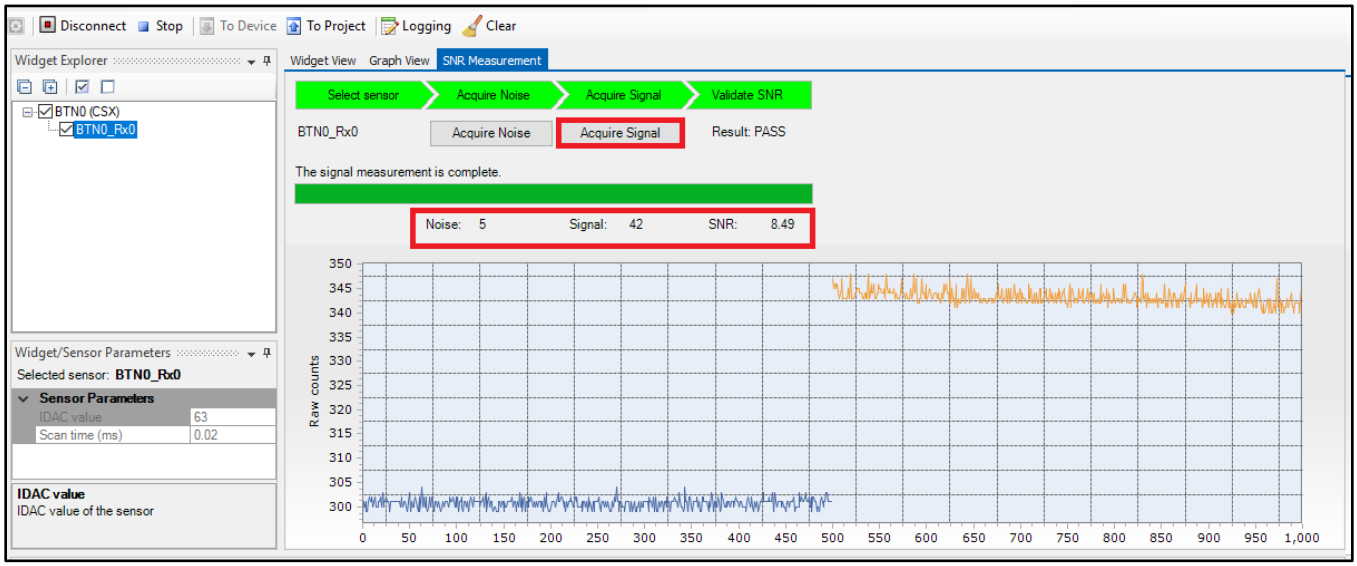
8. Switch to the **SNR Measurement tab**, select the **BTN0_Rx0** sensor, and then click **Acquire Noise** as Figure 14 shows.

Figure 14. SNR Measurement Using CapSense Tuner



9. Once the noise is acquired, touch button BTN0 on the kit, and then click **Acquire Signal**. Ensure that the finger remains on the button as long as the signal acquisition is in progress. You will now be able to see the calculated SNR on this button as Figure 15 shows. Based on your end-system design, test with a finger that matches the size of your normal use case. Typically, finger size targets are ~8 to 9 mm.

Figure 15. Acquiring Signal and Calculating SNR Using Tuner



Stage 3. Modify Hardware Parameters/Adjust Filters to meet SNR and Timing Requirements

1. Skip to Stage 4. Set Threshold Parameters if the following conditions are met:
    - Measured SNR from Stage 2. Measure SNR is greater than 5:1
    - Signal count is greater than 50
    - Response time requirement is met

Perform steps 2 to 4 to ensure that SNR and timing requirements are met. You should tune the buttons for an SNR > 5:1 to avoid triggering on noise and ensure that all intended touches are registered.

Modify Hardware Parameters

2. Increasing the number of sub-conversions increases the signal without increasing the noise at the same rate. However, increasing the number of sub-conversions will in turn increase the sensor scan time.

    Do the following to update the number of sub-conversions:

    a. Update the number of sub-conversions directly in the **Widget/Sensor Parameters** tab of the CapSense Tuner.

    b. Increase the number of sub-conversions in steps of 10 and repeat steps in the Measure SNR section until the minimum SNR of 5:1 and at least a signal count greater than 50 are achieved.

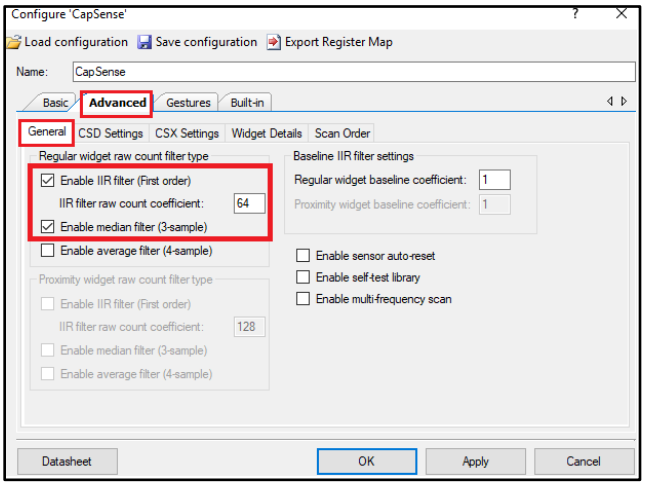
Adjust Filter Settings

3. If your system is very noisy (raw counts > 20), add a filter. Filters help reduce the noise without increasing the signal. Adding a filter adds to the processing time and memory usage, because this is implemented in firmware and gets executed by the CPU. This also results in increased power consumption and response time. Filters must be added such that the response time, power consumption, and SNR requirements are satisfied. For more details on filters and selecting the filter coefficients, see the CapSense Component datasheet.

    a. To adjust the filter settings, open **CapSense Configurator** and select the appropriate filter as shown in Figure 16.

    b. Reprogram the device to update filter settings.

Figure 16. Filter Settings in CapSense Configurator



Use Table 6. Filter Settings for Different Development KitsTable 6 to set the filter settings for different development kits:

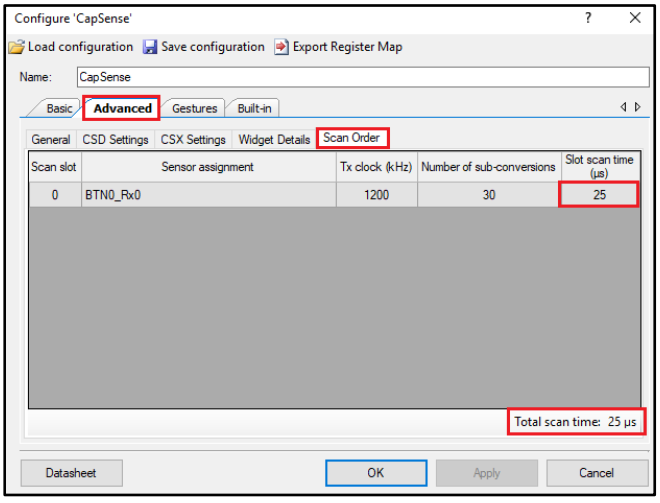
Table 6. Filter Settings for Different Development Kits

Development Kit	IIR Filter	IIR Filter Raw Count Coefficient	Median Filter	Average Filter
CY8CKIT-149	Disabled	NA	Disabled	Disabled
CY8CKIT-145-40XX	Disabled	NA	Disabled	Disabled

Timing Requirements

4. If the total sensor scan time meets your requirements, skip to Stage 4. Set Threshold Parameters. If not, adjust the tuning to speed up the scan time. If the SNR is greater than 10 on any sensor, you can lower the number of sub-conversions or remove filters to decrease the scan time but keep the SNR greater than 5:1. It is best to find a balance between the number of sub-conversions and filters to achieve a proper overall tuning. The hardware scan time can be seen from the **Scan Order** tab in the **Advanced** settings of the CapSense Component as shown in Figure 17. The total scan time of the sensor includes the sensor initialization, which is the time taken by the sensor to write to the internal registers and initiate a scan.

Figure 17. Hardware Scan Time of the Sensor



5. Use Table 7 to set the final hardware tuning parameters for the supported kits to achieve 5:1 SNR.

Table 7. Final Hardware Tuning Parameters to Achieve 5:1 SNR on Different Development Kits

Development Kit	Tx Clock Frequency (kHz)	Number of Sub-conversions	Gain IDAC	IDAC value
CY8CKIT-149	1200	30	600 nA/bit	63
CY8CKIT-145	3000	200	2400 nA/bit	59

* Firmware filters enabled.

Stage 4. Set Threshold Parameters

After you have confirmed that your design meets the timing parameters, and the SNR is greater than 5:1, set your threshold parameters as follows:

1. Switch to the **Graph View** tab and ensure that **BTN0 (CSX)** is selected.
2. Touch the sensor and monitor the touch signal in the **Sensor Signal** graph.

   The Sensor Signal graph should show the signal as Figure 18 shows. Ensure that you observe the difference count (that is, the **signal** output) in the **Graph View** tab in Figure 18, not the raw count output for setting these thresholds. Based on your end system design, test the signal with a finger that matches the size of your normal use case. Typically, finger size targets are ~8 to 9 mm. Consider testing with smaller sizes that should be rejected by the system to ensure that they do not reach the finger threshold.
3. When the signal is measured, set the thresholds according to the following recommendations:

   Finger Threshold = 80 percent of signal

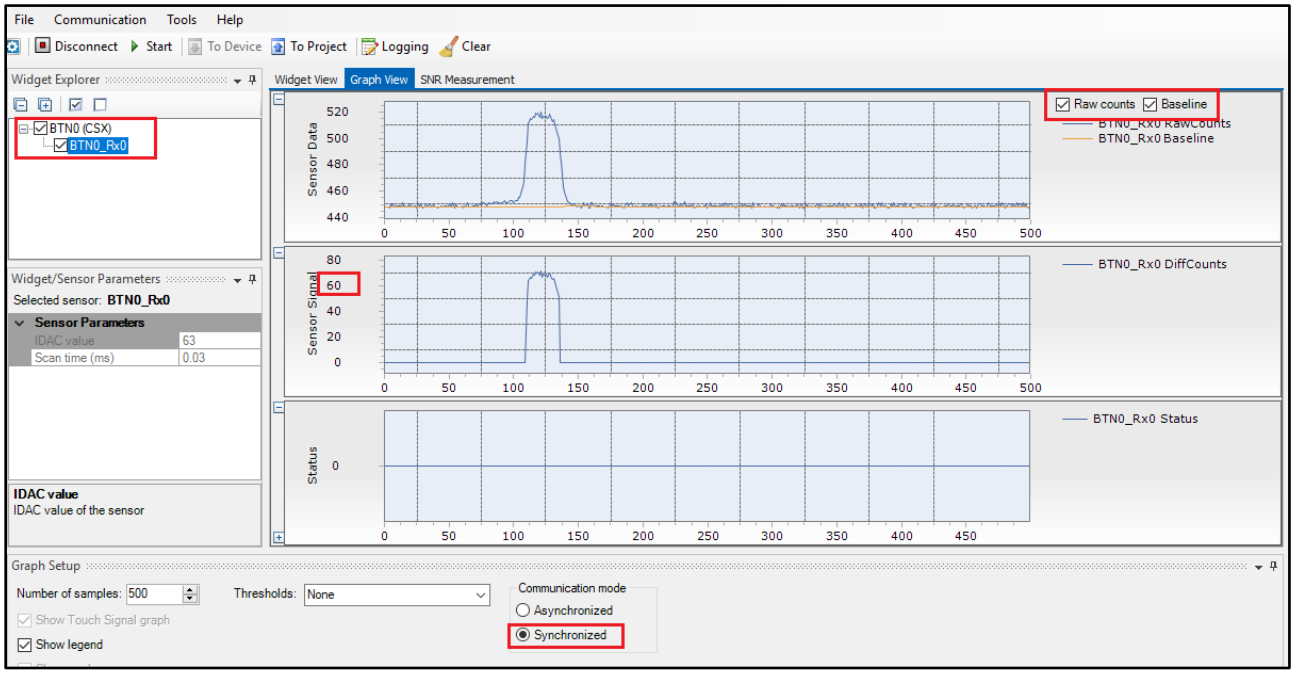
   Noise Threshold = 40 percent of signal

   Negative Noise Threshold = 40 percent of signal

   Hysteresis = 10 percent of signal

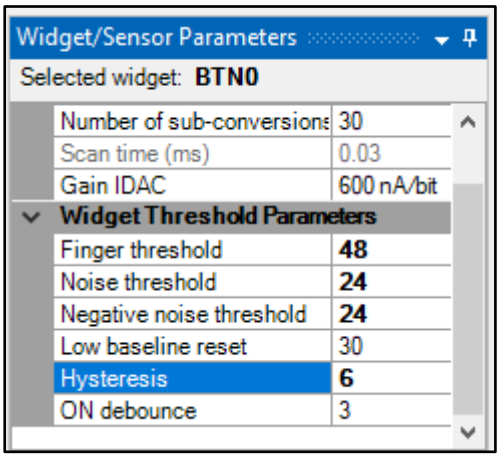
   Debounce = 3

Figure 18. Sensor Signal when the Sensor Is Touched



4. Set the threshold parameters in the **Widget/Sensor Parameters** section of the Tuner GUI, as Figure 19 shows.

Figure 19. Widget Threshold Parameters



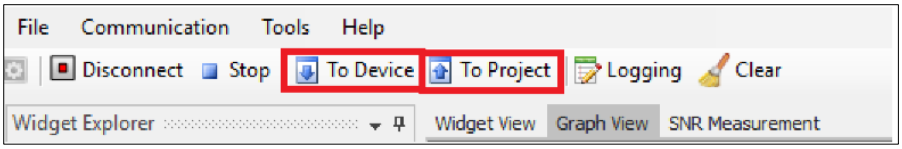
See Table 8 to set the threshold parameters in the CapSense Tuner for different development kits.

Table 8. Threshold Parameters for Different Development Kits

Development Kit	Difference Counts	Finger Threshold	Noise Threshold	Negative Noise Threshold	Hysteresis	Low Baseline Reset	Debounce
CY8CKIT-149	60	48	24	24	6	30	3
CY8CKIT-145	50	40	20	20	5	30	3

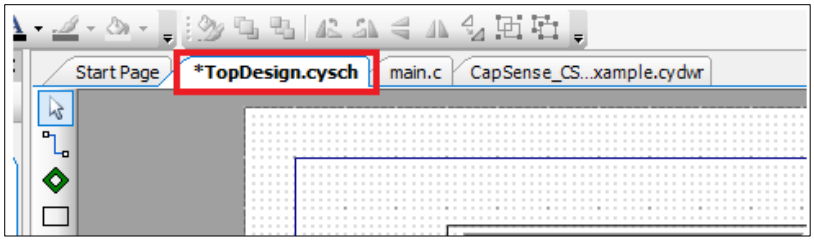
5. Apply the settings to the device and to the project by clicking on **To Device** and **To Project** as Figure 20 shows and close the tuner.

Figure 20. Apply to Project



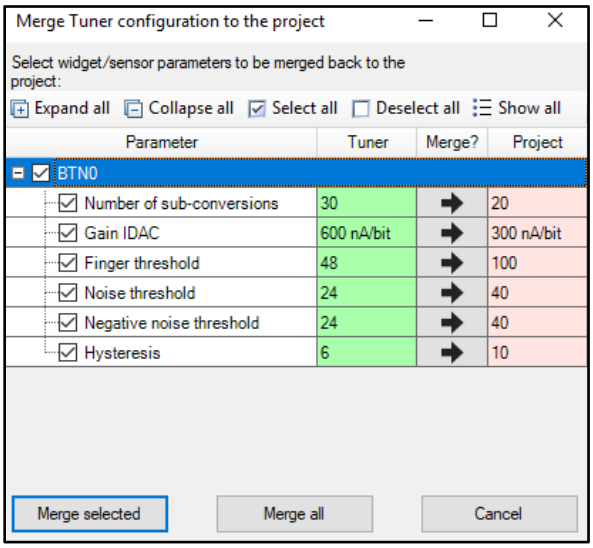
6. If your sensor is tuned correctly, you will observe the touch status go from 0 to 1 in the **Status** sub-window of the **Graph View** window. The successful tuning of the button is also indicated by a LED in the kit. The LED is turned 'ON' when the finger touches the button and turned 'OFF' when the finger is lifted from the button.
7. Notice the * sign on the **TopDesign** file, indicating that TopDesign has been updated, as Figure 21 shows.

Figure 21. TopDesign Gets Updated when To Project Is Clicked on Tuner



8. Double-click the CapSense Component to open the configuration window and choose **Merge All** to accept the changes applied by the CapSense Tuner, as Figure 22 shows. Click **OK** to close the Component Configuration window.

Figure 22. Accept Tuner Modified Parameters in CapSense Component



Components and Settings

See the Operation section for configuring CapSense component.

Configure EZI2C Component

Figure 23 shows the settings for the EZI2C Component.

Figure 23. EZI2C Component Settings

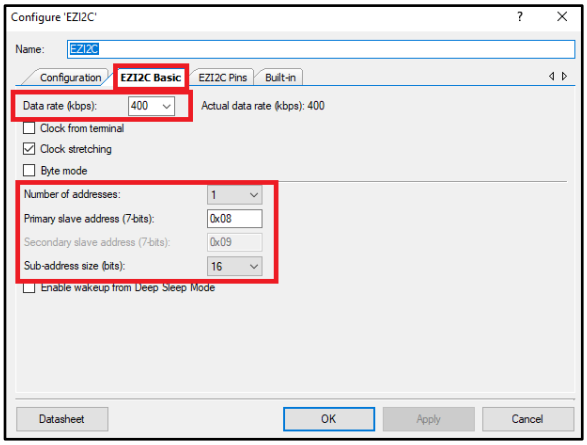


Table 9 lists the PSoC Creator Components used in this example.

Table 9. PSoC Creator Components

Component	Instance Name	Version	Hardware Resources
CapSense	CapSense	v7.0	One CSD
EZI2C Slave (SCB mode)	EZI2C	v4.0	One Serial Communication Block (SCB) and two GPIOs
Digital Output pin	LED_Output	v2.20	One GPIO

Reusing This Example

This example can be easily ported to the kits mentioned in the Requirements section. Refer to the Hardware Setup section for correct pin assignments.

To port the code example to a device that is not listed, in PSoC Creator, select **Project** > **Device Selector** and change to the target device.

This code example can be used in all CapSense-enabled devices in the PSoC 4 family of devices that support the CSX sensing method, 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 and PSoC 4100S Plus.

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 Manual PSoC 4100S Plus Register Technical Reference Manual
PSoC 4000S Datasheet		PSoC 4000S Architecture Technical Reference Manual PSoC 4000S Register Technical Reference Manual
Development Kit (DVK) Documentation		
CY8CKIT-149 PSoC 4100S Plus Prototyping Kit		
CY8CKIT-145-40XX PSoC 4000S CapSense Prototyping Kit		
Tool Documentation		
PSoC™ Creator user guide		
PSoC™ Creator quick start guide		

Document History

Document Title: CE228931 – PSoC™ 4 CapSense CSX Button Tuning

Document Number: 002-28931

Revision	ECN	Date	Description of Change
**	6824937	03/05/2020	New code example
*A	8046867	07/30/2024	Changed CE name to PSoC 4 CapSense CSX Button Tuning Added Cp measurement using BIST Updated Tuning Flow chart Added step to tune Gain IDAC Updated document for clarity Removed CY8CKIT-041-40XX from supported kits

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

Technical Support

cypress.com/support

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

© Cypress Semiconductor Corporation, 2016-2024. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. 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. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.