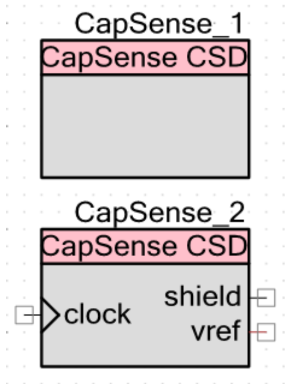
Capacitive Sensing (CapSense® CSD)

2.10

Features



- Supports user defined combinations of button, slider, touch pad, and proximity capacitive sensors
- Provides automatic SmartSense tuning or manual tuning with integrated PC GUI.
- High immunity to AC power line noise, EMC noise, and power supply voltage changes
- Optional two scan channels (parallel synchronized) increases sensor scan rate.
- Shield electrode support for reliable operation in the presence of water film or droplets
- Guided sensor and terminal assignments using the CapSense customizer

General Description

Capacitive Sensing using a Delta-Sigma Modulator (CapSense CSD) component provides a versatile and efficient means for measuring capacitance in applications such as touch sense buttons, sliders, touch pad, and proximity detection.

The following Application notes are recommended reading after reading this datasheet. Application notes can be found on the Cypress Semiconductor web site at www.cypress.com:

- CapSense Best Practices – AN2394
- Signal-to-Noise Ratio Requirements for CapSense Applications – AN2403
- EMC Design Consideration for PSoC CapSense Applications – AN2318
- Layout Guidelines for PSoC CapSense – AN2292
- Waterproof Capacitive Sensing – AN2398

When to Use a CapSense Component

Capacitance sensing systems can be used in many applications in place of conventional buttons, switches, and other controls, even in applications that are exposed to rain or water. Such applications include automotive, outdoor equipment, ATMs, public access systems, portable devices such as cell phones and PDAs, and kitchen and bathroom applications.

Input/Output Connections

This section describes the various input and output connections for the CapSense CSD component. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

clock – Input *

Supplies the clock for the CapSense CSD component. The clock input is only visible if the **Enable clock input** is checked.

shield – Output *

The shield electrode signal is connected to this output. It is only available if shield electrode is enabled. Details on shield use are provided in the Parameters and Setup section.

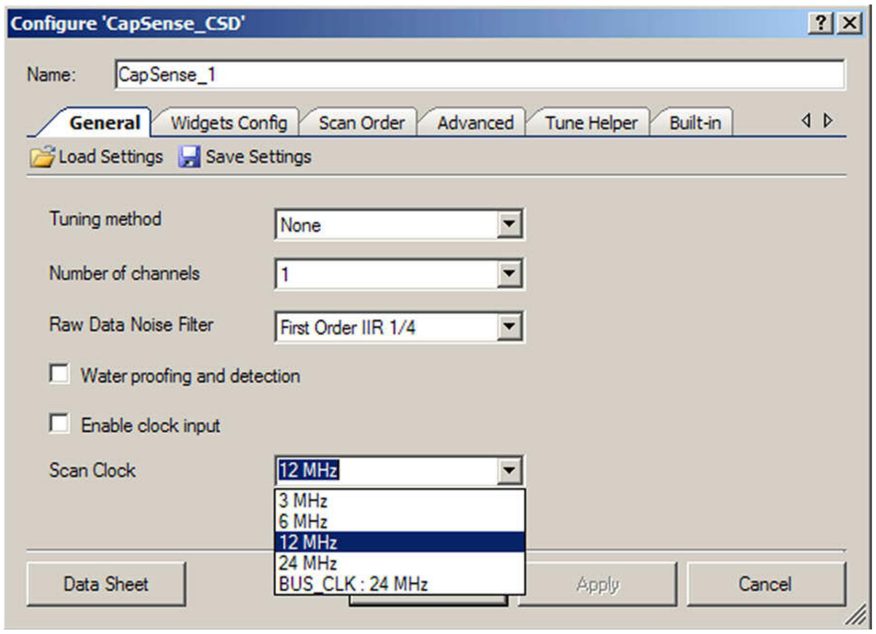
vref – Output *

The analog reference voltage is connected to this output. It can be used to adjust the shield signal amplitude. It is only available if the **Shield** option is enabled in IDAC **Sourcing mode**. Vref output should be connected to SIO reference when SIO is used for a shield signal. Details on vref use are provided in the Functional Description section of this datasheet.

Parameters and Setup

Drag a CapSense CSD component onto your design and double-click it to open the Configure dialog. This dialog has several tabs to guide you through the process of setting up the CapSense CSD component.

General Tab



Name: CapSense_1

Tuning method: None

Number of channels: 1

Raw Data Noise Filter: First Order IIR 1/4

☐ Water proofing and detection

☐ Enable clock input

Scan Clock: 12 MHz

3 MHz
6 MHz
12 MHz
24 MHz
BUS_CLK : 24 MHz

Load Settings/Save Settings

Save Settings is used to save all settings and tuning data configured for a component. This allows quick duplication in a new project. **Load Settings** is used to load previously saved settings.

The stored settings can also be used to import settings and tuning data into the Tuner GUI.

Tuning method

This parameter specifies the tuning method. There are three options:

- Auto (SmartSense) – Provides automatic tuning of the CapSense CSD component and is the recommended tuning method for all designs. . Firmware algorithms determine the best tuning parameters continuously at run time. Additional RAM and CPU resources are required in this mode.
  Important – Only one CapSense_CSD component in SmartSense mode can be placed onto the project schematic. SmartSense tuning may be used with an EZI2C communication component which is specified on the **Tuner Helper** tab to transmit data from the target device to the tuner GUI.

- Manual – Allows manual tuning of the CapSense CSD component using the Tuner GUI.

  To launch the GUI, right-click on the symbol and select **Launch Tuner**. For more information about Manual tuning refer to the CapSense Tuner GUI User Guide section in this data sheet. Manual tuning requires an EZI2C communication component which is specified on the **Tuner Helper** tab to transmit data between the target device and the tuner GUI.

  - None (Default) – Disables tuning. All tuning parameters are stored in flash. This option should only be used after all parameters of the CapSense component are tuned and finalized. If this option is used, the communication functions are still provided but do nothing so Tuner won't work in this mode.

Number of channels

This parameter specifies the number of hardware scanning channels implemented.

Default. Best used for 1-20 sensors. The component is capable of performing one capacitive scan at a time. One sensor is scanned at a time in succession. Since only a single channel is implemented in hardware this option results in the minimum hardware resources being utilized.

- The AMUX buses are tied together.

  Note If all capacitive sensors are allocated on one side of the chip Left (#even ports GPIO for example: P0[X], P2[X], P4[X]) or Right (#odd ports GPIO for example: P1[X], P3[X], P5[X]) the AMUX buses don't tie together; the one half of AMUX bus is used.

  Note The port pins P15[0-5] have connections to different AMUX busses Left and Right. P12[X] and P15[6-7] do not have a connection to the AMUX bus. Refer to the TRM for the selected part.

- The component is capable of scanning 1 to (#GPIO – 1) capacitive sensors.

- One Cmod external capacitor is required.

Best used for over 20 sensors. The component is capable of performing two simultaneous capacitive scans. Both the Left and Right AMUX buses are used, one for each channel. Right and Left sensors are scanned two at time (one Right sensor and one Left sensor) in succession. If one channel has more sensors than the other, the channel with the greater number of sensors will finish scanning the remaining sensors in its array one at a time until done while the other channel performs no scans. Two channels doubles the resource used vs 1 channel but in return doubles the sensor scan rate.

- The Left AMUX bus can scan 1 to (#even ports GPIO – 1) capacitive sensors.

- The Right AMUX bus can scan 1 to (#odd ports GPIO – 1) capacitive sensors.

- Two Cmod external capacitors are required, 1 for each channel.

- Parallel scans run at the same scan rate.

Raw Data Noise Filter

This parameter selects the raw data filter. Only one filter may be selected and it is applied to all sensors. A filter is recommended to reduce the effect of noise during sensor scans. Details on the types of filters can be found in the Functional Description – Filters sections in this document.

- None – No filter is provided. No filter firmware or SRAM variable overhead is incurred.
- Median – Sorts the last three sensor values in order and returns the middle value.
- Averaging – Returns the simple average of the last three sensor values
- First Order IIR 1/2 – Returns 1/2 of the most current sensor value added to 1/2 of the previous filter value. IIR filters require the lowest firmware and SRAM overhead of all the filter types.
- First Order IIR 1/4 – Default – Returns 1/4th of the most current sensor value added to 3/4th of the previous filter value.
- First Order IIR 1/8 – Returns 1/8th of the most current sensor value added to 7/8th of the previous filter value.
- First Order IIR 1/16 – Returns 1/16th of the most current sensor value added to 15/16th of the previous filter value.
- Jitter – If the most current sensor value is greater than the last sensor value then the previous filter value is incremented by 1, if it is less then it is decremented.

Water proofing and detection

This feature configures the CapSense CSD to support water proofing (Unchecked - default). This feature sets the following parameters:

- Enables the Shield output terminal
- Adds a Guard widget

Note If the Guard widget is not desired with water proofing it can be removed on the **Advanced** tab.

Enable clock input

This parameter selects whether the component uses an internal clock or displays an input terminal for a user supplied clock connection (Unchecked – Default).

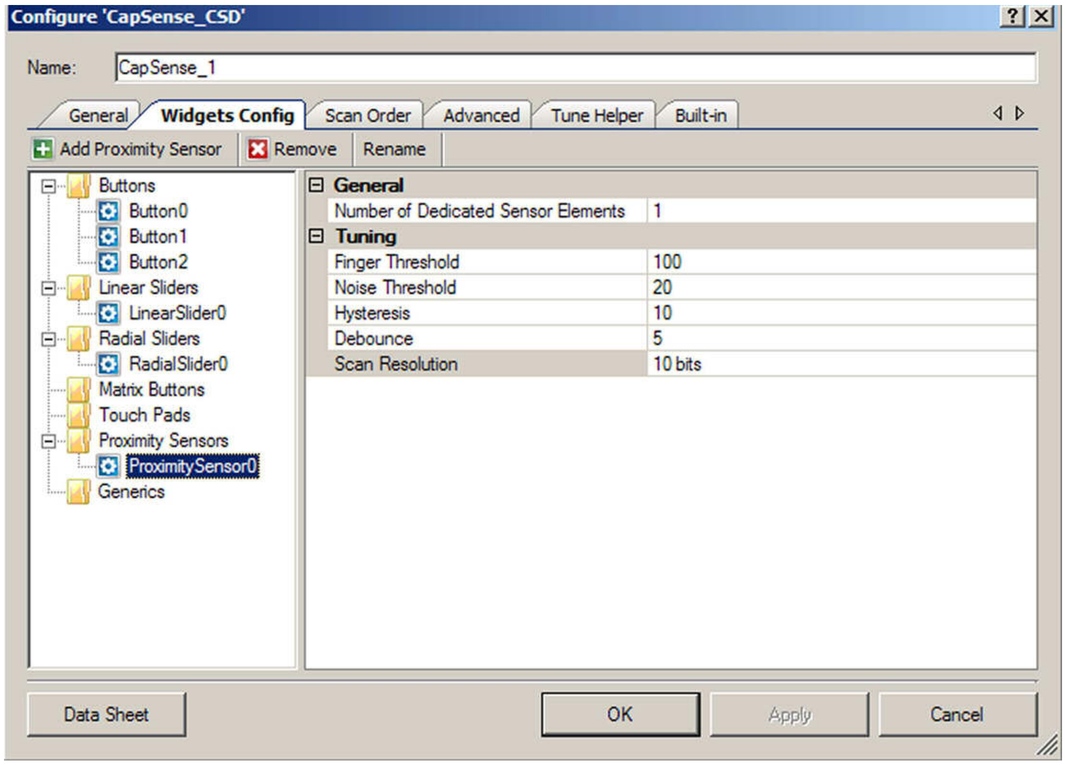
Note This option is disabled if the tuning method is Auto (SmartSense) as customizer required to know clock frequency to calculate internal data.

Scan Clock

This parameter specifies the internal CapSense component clock frequency. The range of values is 3 MHz to 24 MHz (12 MHz – default). This feature is disabled if the **Enable clock input** is checked.

Note Setting **Analog Switch Drive Source** to "FF Timer (Default)" and/or **Digital Implementation** to "FF Timers," does not support the CapSense CSD clock less than or equal too BUS_CLK, therefore BUS_CLK should be selected.

Widgets Config Tab



Name: CapSense_1

Buttons
- Button0
- Button1
- Button2

Linear Sliders
- LinearSlider0

Radial Sliders
- RadialSlider0

Matrix Buttons

Touch Pads

Proximity Sensors
- ProximitySensor0

Generics

General	
Number of Dedicated Sensor Elements	1
Tuning	
Finger Threshold	100
Noise Threshold	20
Hysteresis	10
Debounce	5
Scan Resolution	10 bits

Definitions for various parameters are provided in the Functional Description section.

Toolbar

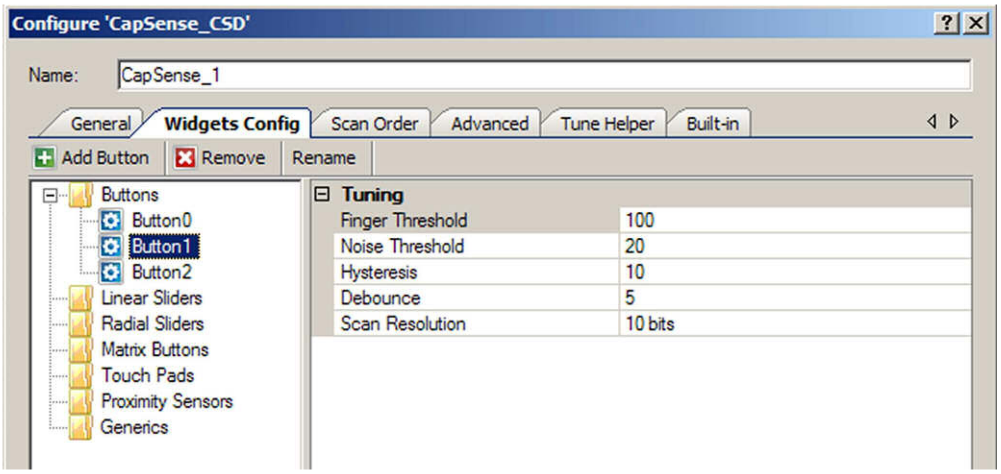
The toolbar contains the following commands:

- **Add widget** – Adds the selected type of widget to the tree. The widget types are:
  - ❑ Button – A button detects a finger press on a single sensor and provides a single mechanical button replacement
  - ❑ Linear Slider – A linear slider provides an integer value based on interpolating the location of a figure press on a small number of sensors.
  - ❑ Radial Slider – A radial slider is similar to a linear slider except that the sensors are placed in a circle.

- ❑ Matrix Button – A matrix button detects a finger press at the intersection formed by a row sensor and column sensor. Matrix buttons provide an efficient method of scanning a large number of buttons.
- ❑ Touch Pad – A touch pad returns the X and Y coordinates of a figure press within the touchpad area. A touchpad is comprised of multiple row and column sensors.
- ❑ Proximity Sensor – A proximity sensor is optimized to detect the presence of a finger hand or other large object at a large distance from the sensor avoiding the requirement for an actual touch to occur.
- ❑ Generic Sensor – Generic sensor provide raw data from a single sensor allowing the creation of unique or advanced sensors no otherwise possible with processed outputs of the other sensor types.

- **Remove widget** – Removes the selected widget from the tree.
- **Rename** – Opens a dialog to change the selected widget name. You can also double-click a widget to open the dialog.

Buttons



Tuning

Finger Threshold	100
Noise Threshold	20
Hysteresis	10
Debounce	5
Scan Resolution	10 bits

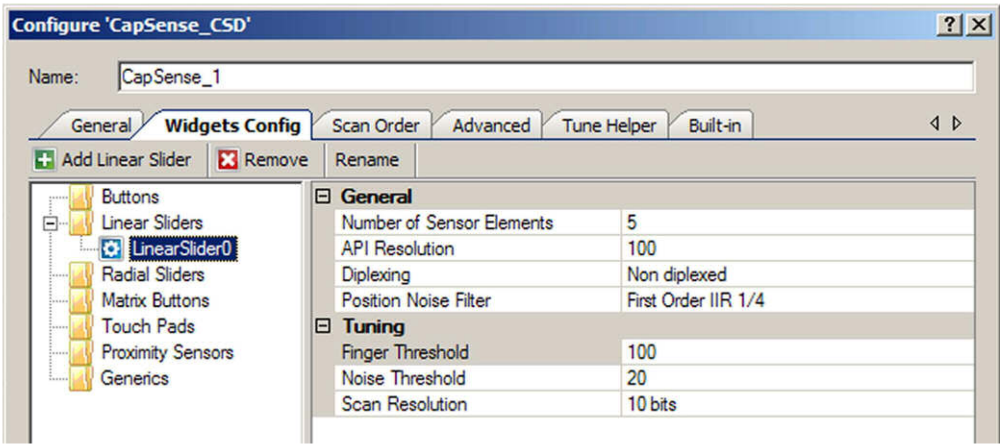
Tuning:

- **Finger Threshold** – Defines sensor active threshold resulting in increased or decreased sensitivity to touches. When the sensor scan value is greater than this threshold the button is reported as touched. Default value is 100. Valid range of values is [1…255]. Finger Threshold + Hysteresis cannot be more than 254.

- **Noise Threshold** – Defines sensor noise threshold. Count values above this threshold do not update the baseline. If the noise threshold is too low sensor and thermal offsets may not be accounted for resulting in false or missed touches. If the noise threshold is too high a finger touch may be interpreted as noise and artificially increase the baseline resulting in missed finger touches. Default value is 20. Valid range of values is [1…255].

- **Hysteresis** – Adds differential hysteresis for sensor active state transitions. If the sensor is inactive, the difference count must overcome the finger threshold plus hysteresis. If the sensor is active, the difference count must go below the finger threshold minus hysteresis. Hysteresis helps to ensure that low amplitude sensor noise and small finger moves do not cause cycling of the button state. Default value is 10. Valid range of values is [1…255]. Finger Threshold + Hysteresis cannot be more than 254.
- **Debounce** – Adds a debounce counter for detection of the sensor active state transition. In order for the sensor to transition from inactive to active, the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified. Default value is 5. Debounce ensures that high frequency high amplitude noise does not cause false detection of a pressed button. Valid range of values is [1…255].
- **Scan Resolution** – Defines the scanning resolution. This parameter has an effect on the scanning time of the sensor within the button widget. The maximum raw count for the scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the Signal to Noise Ratio (SNR) of touch detection but increases scan time. Default value is 10 bits.

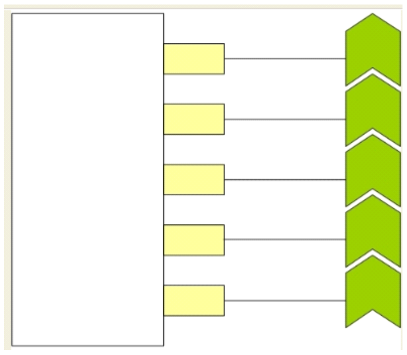
Linear Sliders

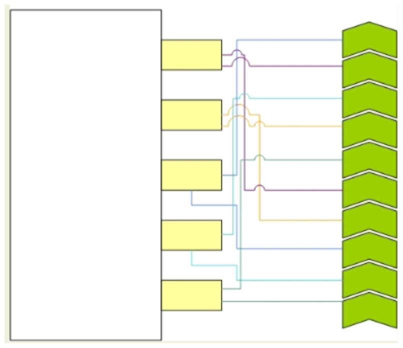


General	
Number of Sensor Elements	5
API Resolution	100
Diplexing	Non diplexed
Position Noise Filter	First Order IIR 1/4
Tuning	
Finger Threshold	100
Noise Threshold	20
Scan Resolution	10 bits

General:

- **Numbers of Sensors Elements** – Defines the number of elements within the slider. A good ratio of API resolution to sensor elements is 20:1. Increasing the ratio of API resolution to sensor elements too much can result in increased noise on the calculated finger position. Valid range of values is [2…32]. Default value is 5 elements.
- **API Resolution** – Defines the slider resolution. The position value will be changed within this range. Valid range of values is [1…255].
- **Diplexing –** Non Diplexed (Default) or Diplexed. Diplexing allows two slider sensors to share a single device pin reducing the total number of pins required for a given number of slider sensors.


Non Diplexed


Diplexed

- **Position Noise Filter** – Selects the type of noise filter to perform on position calculations. Only one filter may be applied for a selected widget. Details on the types of filters can be found in the Functional Description – Filters sections in this document.
  - ❑ None
  - ❑ Median Filter
  - ❑ Averaging Filter
  - ❑ First Order IIR 1/2
  - ❑ First Order IIR 1/4 – Default
  - ❑ Jitter Filter

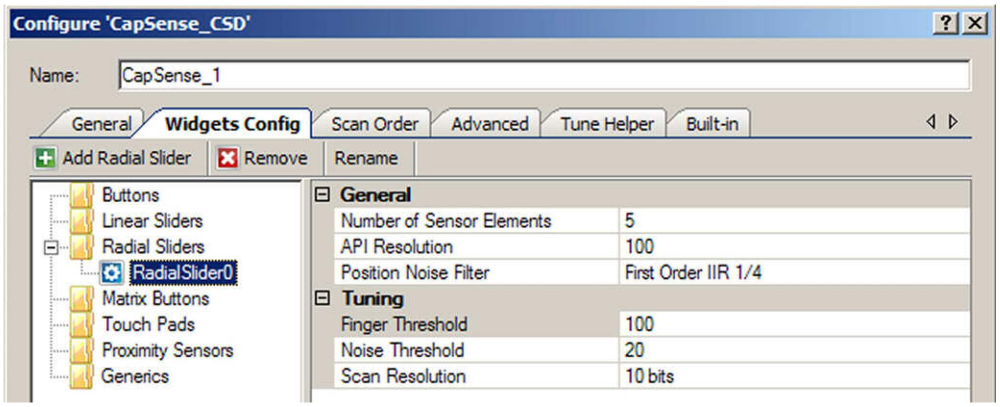
Tuning:

- **Finger Threshold** – Defines sensor active threshold resulting in increased or decreased sensitivity to touches. When the sensor scan value is greater than this threshold the button is reported as touched. Default value is 100. Valid range of values is [1…255].

- **Noise Threshold** – Defines the sensor noise threshold for slider elements. Count values above this threshold do not update the baseline. If the noise threshold is too low sensor and thermal offsets may not be accounted for resulting in false or missed touches. If the noise threshold is too high a finger touch may be interpreted as noise and artificially increase the baseline resulting in centroid location calculation errors. Count values below this threshold are not counted in the calculation of the centroid. Default value is 20. Valid range of values is [1…255].

- **Scan Resolution** – Defines the scanning resolution. This parameter has an effect on the scanning time of all sensors within the linear slider widget. The maximum raw count for scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection but increases scan time. Default value is 10 bits.

Radial Slider



General	
Number of Sensor Elements	5
API Resolution	100
Position Noise Filter	First Order IIR 1/4
Tuning	
Finger Threshold	100
Noise Threshold	20
Scan Resolution	10 bits

General:

- **Numbers of Sensors Elements** – Defines the number of elements within the slider. A good ratio of API resolution to sensor elements is 20:1. Increasing the ratio of API resolution to sensor elements to much can result in increased noise on the resolution calculation. Valid range of values is [2…32]. Default value is 5 elements.

- **API Resolution** – Defines the resolution of the slider. The position value will be changed within this range. Valid range of values is [1…255].

- **Position Noise Filter** – Selects the type of noise filter to perform on position calculations. Only one filter may be applied for a selected widget. Details on the types of filters can be found in the Functional Description – Filters sections in this document.
  - None
  - Median Filter
  - Averaging Filter
  - First Order IIR 1/2
  - First Order IIR 1/4 – Default
  - Jitter Filter

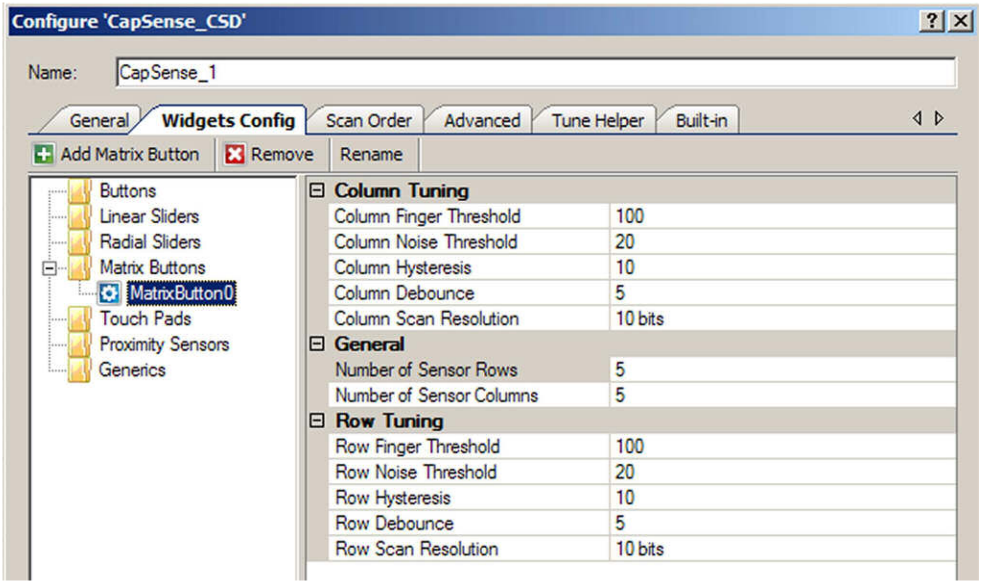
Tuning:

- **Finger Threshold** – Defines the sensor active threshold resulting in increased or decreased sensitivity to touches. When the sensor scan value is greater than this threshold the button is reported as touched. Default value is 100.

- **Noise Threshold** – Defines the sensor noise threshold for slider elements. Count values above this threshold do not update the baseline. If the noise threshold is too low sensor and thermal offsets may not be accounted for resulting in false or missed touches. If the noise threshold is too high a finger touch may be interpreted as noise and artificially increase the baseline resulting in centroid location calculation errors. Count values below this threshold are not counted in the calculation of the centroid. Default value is 20. Valid range of values is [1…255].

- **Scan Resolution** – Defines the scanning resolution. This parameter has an effect on the scanning time of all sensors within a radial slider widget. The maximum raw count for scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection but increases scan time. Default value is 10 bits.

Matrix Buttons



Name: CapSense_1

Column Tuning	
Column Finger Threshold	100
Column Noise Threshold	20
Column Hysteresis	10
Column Debounce	5
Column Scan Resolution	10 bits
General	
Number of Sensor Rows	5
Number of Sensor Columns	5
Row Tuning	
Row Finger Threshold	100
Row Noise Threshold	20
Row Hysteresis	10
Row Debounce	5
Row Scan Resolution	10 bits

Tuning

- **Column/Row Finger Threshold** – Defines the sensor active threshold for matrix button columns and rows resulting in increased or decreased sensitivity to touches. When the sensor scan value is greater than this threshold the button is reported as touched. Default value is 100. Valid range of values is [1…255]. Finger Threshold + Hysteresis cannot be more than 254.

- **Column/Row Noise Threshold** – Defines the sensor noise threshold for matrix button columns and rows. Count values above this threshold do not update the baseline. If the noise threshold is too low, sensor and thermal offsets may not be accounted for resulting in false or missed touches. If the noise threshold is too high a finger touch may be interpreted as noise and artificially increase the baseline resulting in missed finger touches. Default value is 20. Valid range of values is [1…255].

- **Column/Row Hysteresis** – Adds differential hysteresis for sensor active state transitions for matrix button columns and rows. If the sensor is inactive, the difference count must overcome the finger threshold plus hysteresis. If the sensor is active, the difference count must go below the finger threshold minus hysteresis. Hysteresis helps to ensure that low amplitude sensor noise and small finger moves do not cause cycling of the button state. Default value is

10. Valid range of values is [1…255]. Finger Threshold + Hysteresis cannot be more than 254.

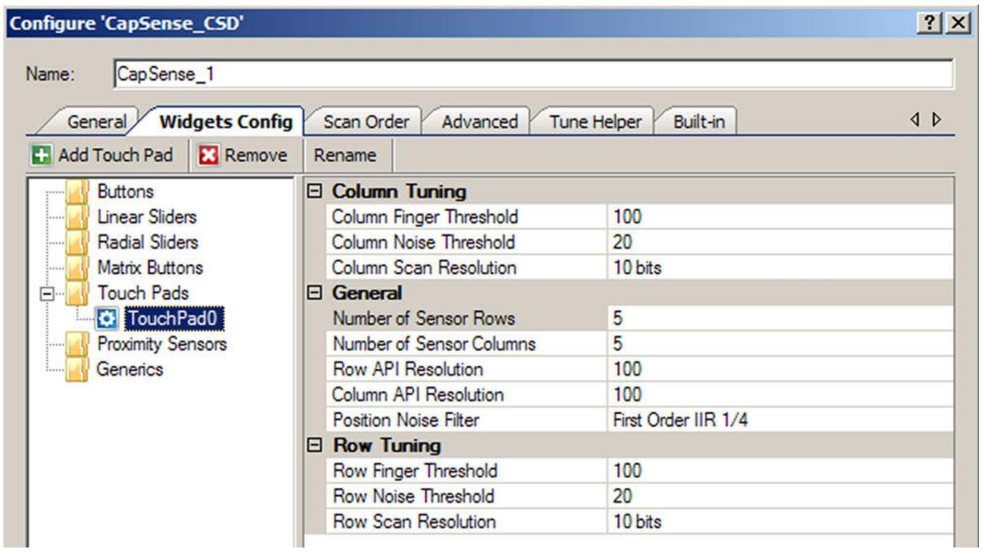
- **Column/Row Debounce** – Adds a debounce counter for detection of the sensor active state transition for matrix buttons column/row. In order for the sensor to transition from inactive to active, the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified. Default value is 5. Debounce ensures that high frequency high amplitude noise does not cause false detection of a pressed button. Valid range of values is [1…255].

- **Column/Row Scan Resolution** – Defines the scanning resolution of matrix button columns and rows. This parameter has an effect on the scanning time of all sensors within a column/row of a matrix button widget. The maximum raw count for scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection but increases scan time. The Column and Row scanning resolutions should be the same to get the same sensitivity level. Default value is 10 bits.

General

- **Number of Sensor Columns/Rows** – Defines the number of columns and rows that form the matrix. Valid range of values is [2…32]. Default value is 5 elements for both Column and Row.

Touch Pads



Name: CapSense_1

Column Tuning	
Column Finger Threshold	100
Column Noise Threshold	20
Column Scan Resolution	10 bits
General	
Number of Sensor Rows	5
Number of Sensor Columns	5
Row API Resolution	100
Column API Resolution	100
Position Noise Filter	First Order IIR 1/4
Row Tuning	
Row Finger Threshold	100
Row Noise Threshold	20
Row Scan Resolution	10 bits

Tuning

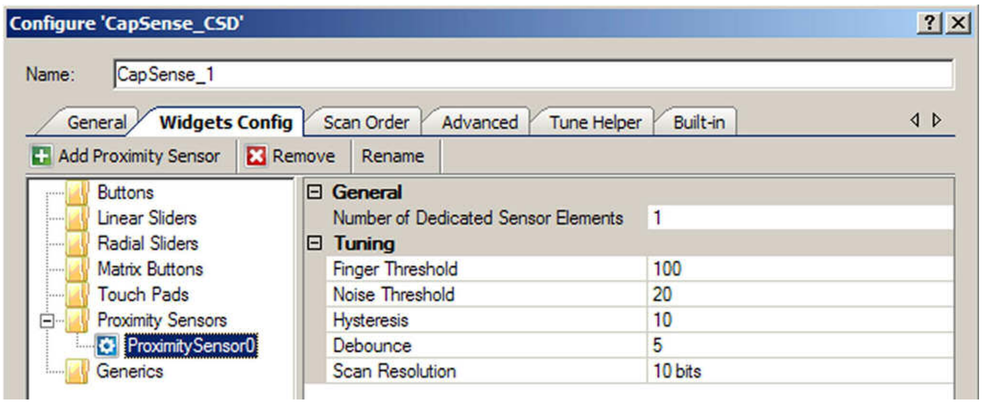
- **Column/Row Finger Threshold** – Defines the sensor active threshold for touchpad columns and rows resulting in increased or decreased sensitivity to touches. When the sensor scan value is greater than this threshold the touchpad reports the touch position. Default value is 100. Valid range of values is [1…255].

- **Column/Row Noise Threshold** – Defines the sensor noise threshold for touchpad columns and rows. Count values above this threshold do not update the baseline. Count values below this threshold are not counted in the calculation of the centroid location. If the noise threshold is too low sensor and thermal offsets may not be accounted for resulting in false or missed touches. If the noise threshold is too high a finger touch may be interpreted as noise and artificially increase the baseline resulting in centroid calculation errors. Default value is 20. Valid range of values is [1…255].
- **Column/Row Scan Resolution** – Defines the scanning resolution of touchpad columns and rows. This parameter has an effect on the scanning time of all sensors within a column/row of a touchpad widget. The maximum raw count for scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection but increases scan time. The Column and Row scanning resolution should be equal to get the same sensitivity level. Default value is 10 bits.

General

- **Numbers of Sensors Colum/Row** – Defines the number of columns and rows that form the touchpad. Valid range of values is [2…32]. Default value is 5 elements for both the column and row.
- **API Resolution Colum/Row** – Defines the resolution of the touch pad columns and rows. The finger position values will be reported within this range. Valid range of values is [1…255].
- **Position Noise Filter** – Adds noise filter to position calculations. Only one filter may be applied for a selected widget. Details on the types of filters can be found in the Functional Description – Filters sections in this document.
  - None
  - Median Filter
  - Averaging Filter
  - First Order IIR 1/2
  - First Order IIR 1/4 – Default
  - Jitter Filter

Proximity Sensors



General	
Number of Dedicated Sensor Elements	1
Tuning	
Finger Threshold	100
Noise Threshold	20
Hysteresis	10
Debounce	5
Scan Resolution	10 bits

General

- **Numbers of Dedicated Sensors elements** – Selects the number of dedicated proximity sensors. These sensor elements are in addition to all the other sensors instantiated for other Widgets. Any Widget sensors may be used individually or connected together in parallel to create proximity sensors.
  - 0 – The proximity sensor will only scan one or more existing sensors to determine proximity. No new sensors are allocated for this widget.
  - 1 – Number of dedicated proximity sensors in the system. – Default.

Tuning

- **Finger Threshold** – Defines the sensor active threshold resulting in increased or decreased sensitivity to the proximity of a touch. When the sensor scan value is greater than this threshold the proximity sensor is reported as touched. Default value is 100. Valid range of values is [1…255]. Finger Threshold + Hysteresis cannot be more than 254..

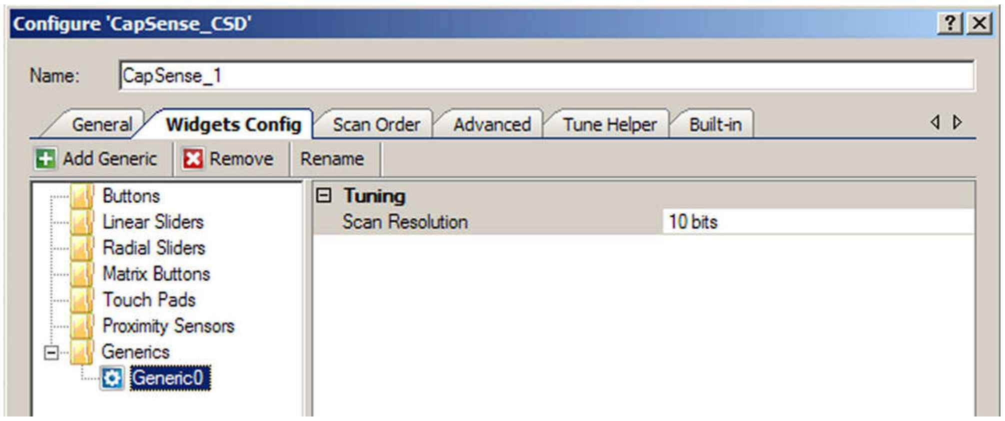
- **Noise Threshold** – Defines the sensor noise threshold. Count values above this threshold do not update the baseline. If the noise threshold is too low sensor and thermal offsets may not be accounted for resulting in false or missed proximity touches. If the noise threshold is too high a figure touch may be interpreted as noise and artificially increase the baseline resulting in missed finger touches. Valid range of values is [1…255].

- **Hysteresis** – Adds differential hysteresis for the sensor active state transition. If the sensor is inactive, the difference count must overcome the finger threshold plus hysteresis. If the sensor is active, the difference count must go below the finger threshold minus hysteresis. Hysteresis helps to ensure that low amplitude sensor noise and small finger/body moves do not cause cycling of the proximity sensor state. Valid range of values is [1…255].

- **Debounce** – Adds a debounce counter for detection of the sensor active state transition. In order for the sensor to transition from inactive to active, the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified. Debounce ensures that high frequency high amplitude noise does not cause false detection of a proximity event. Valid range of values is [1…255].

- **Scan Resolution** – Defines the scanning resolution. This parameter has an effect on the scanning time of a proximity widget. The maximum raw count for scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection but increases scan time. We recommend using a higher resolution for proximity detection than what is used for a typical button to increase detection range. Default value is 10 bits.

Generics



Tuning

Scan Resolution 10 bits

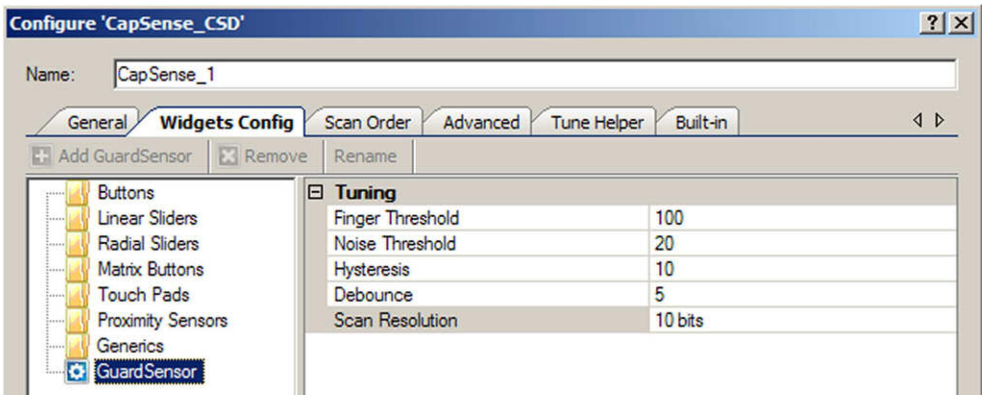
Tuning

- **Scan Resolution** – Defines the scanning resolution. This parameter has an effect on the scanning time of a generic widget. The maximum raw count for scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection but increases scan time. Default value is 10 bits.

Only one tuning option is available for a generic widget because all high Level handling is left to the customer to support CapSense sensors and algorithms that do not fit into any of the predefined widgets.

Guard Sensor

This special sensor is added or removed using the **Advanced Tab**. The Guard sensor does not report a finger press like other sensors but reports an invalid condition near the other widgets to suppress their update. For more information about this sensor type and when it should be used, refer to "Guard Sensor Implementation" in the Functional Description section of this data sheet.



Finger Threshold	100
Noise Threshold	20
Hysteresis	10
Debounce	5
Scan Resolution	10 bits

Tuning:

- **Finger Threshold** – Defines sensor active threshold resulting in increased or decreased sensitivity to touches. When the sensor scan value is greater than this threshold the guard sensor is reported as touched. Default value is 100. Valid range of values is [1…255]. Finger Threshold + Hysteresis cannot be more than 254.

- **Noise Threshold** – Defines the sensor noise threshold. Count values above this threshold do not update the baseline. If the noise threshold is too low sensor and thermal offsets may not be accounted for resulting in false or missed touches. If the noise threshold is too high a finger touch may be interpreted as noise and artificially increase the baseline resulting in missed finger touches. Default value is 20. Valid range is [1…255].

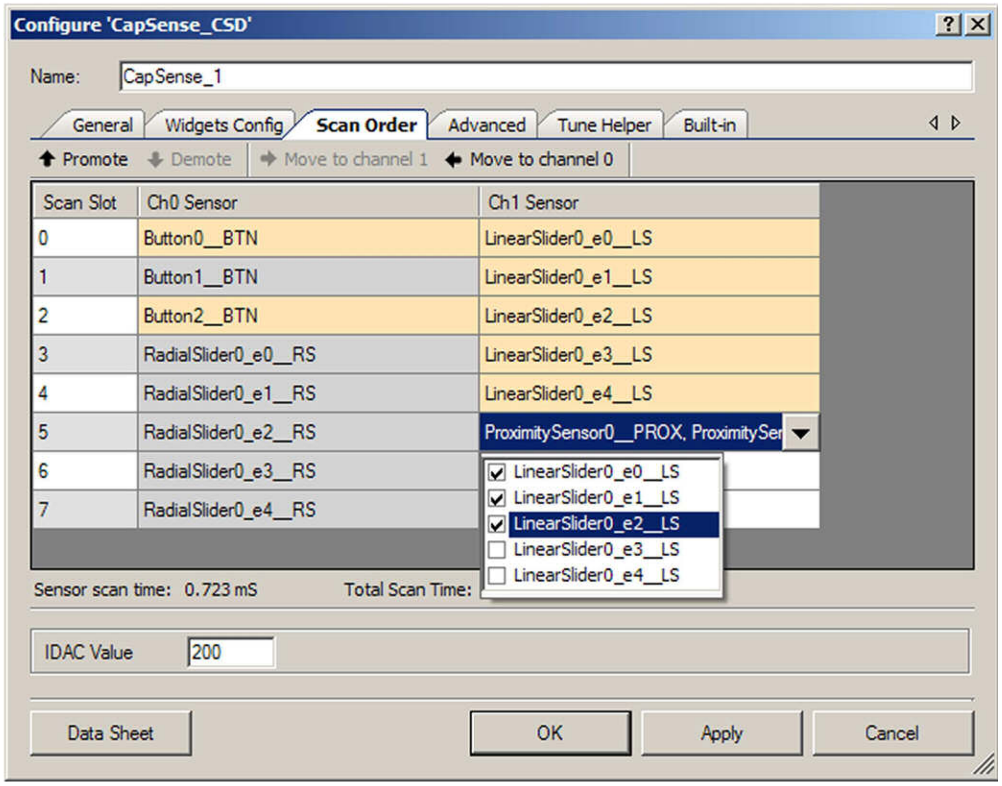
- **Hysteresis** – Adds the differential hysteresis for sensor active state transition. If the sensor is inactive, the difference count must overcome the finger threshold plus hysteresis. If the sensor is active, the difference count must go below the finger threshold minus hysteresis. Hysteresis helps to ensure that low amplitude sensor noise and small finger moves do not cause cycling of the button state. Default value is 10. Valid range of values is [1…255]. Finger Threshold + Hysteresis cannot be more than 254.

- **Debounce** – Adds a debounce counter for detection of the sensor active state transition. In order for the sensor to transition from inactive to active the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified. Debounce ensures that high frequency high amplitude noise does not cause false detection of the guard sensor. Default value is 5. Valid range of values is [1…255].

- **Scan Resolution** – Defines the scanning resolution. This parameter has an effect on the scanning time of a guard sensor. The maximum raw count for scanning resolution for N bits is

2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection but increases scan time. Default value is 10 bits.

Scan Order Tab



Name: CapSense_1

Scan Slot	Ch0 Sensor	Ch1 Sensor
0	Button0__BTN	LinearSlider0_e0__LS
1	Button1__BTN	LinearSlider0_e1__LS
2	Button2__BTN	LinearSlider0_e2__LS
3	RadialSlider0_e0__RS	LinearSlider0_e3__LS
4	RadialSlider0_e1__RS	LinearSlider0_e4__LS
5	RadialSlider0_e2__RS	ProximitySensor0__PROX, ProximitySer
6	RadialSlider0_e3__RS	
7	RadialSlider0_e4__RS	

☑ LinearSlider0_e0__LS
☑ LinearSlider0_e1__LS
☑ LinearSlider0_e2__LS
☐ LinearSlider0_e3__LS
☐ LinearSlider0_e4__LS

Sensor scan time: 0.723 mS	Total Scan Time:

IDAC Value 200

Toolbar

The toolbar contains the following commands:

- **Promote/Demote** – moves the selected widget up or down in the data grid. The whole widget is selected if one or more of its elements are selected.
- **Move to Channel 1/ Channel 0** – moves the selected widget to another channel. This option is active only in two channel designs. The whole widget is selected if one or more of its elements are selected

Note Pins should be re-assigned if scanning order is changed.

Note Proximity sensor is excluded from scanning process by default. Its scan must be started manually at run time as it is typically not scanned at the same time as the other sensors.

IDAC Value

Specifies the IDAC value of the selected sensors. This option is active only when IDAC is selected as the **Current Source** (under the **Advanced** tab). Valid range is 0-255, default value is 200.

Sensitivity

Sensitivity is the nominal change in Cs (sensor capacitance) required to activate a sensor. The valid range of values is [1…4] which corresponds to sensitivity levels: 0.1, 0.2, 0.3, and 0.4 pF. The default value is 2. Sensitivity sets the overall sensitivity of the sensors to account for different thicknesses of overlay material. Thicker material should use a lower sensitivity value.

This option is only available if the **Tuning method** parameter is set to "Auto (SmartSense)."

Sensor Scan Time

Shows the approximate scan time required for the selected sensor in typical systems.

When Auto(SmartSense) is selected as the tuning method the displayed value may be inaccurate because parameters are changed while tuning. 'Unknown' is shown when the CapSense CSD component input clock frequency is unknown.

The following parameters of the CapSense CSD component have effects on the scan time of sensor:

- Scan Speed
- Resolution
- CapSense CSD clock

Total Scan Time

Shows total scan time required to scan all of the sensors. This value is the approximate sensor scan time, so it could be slightly different from real.

When Auto(SmartSense) is selected as the tuning method the displayed value may be inaccurate because parameters are changed while tuning. 'Unknown' is shown when the CapSense CSD component input clock frequency is unknown.

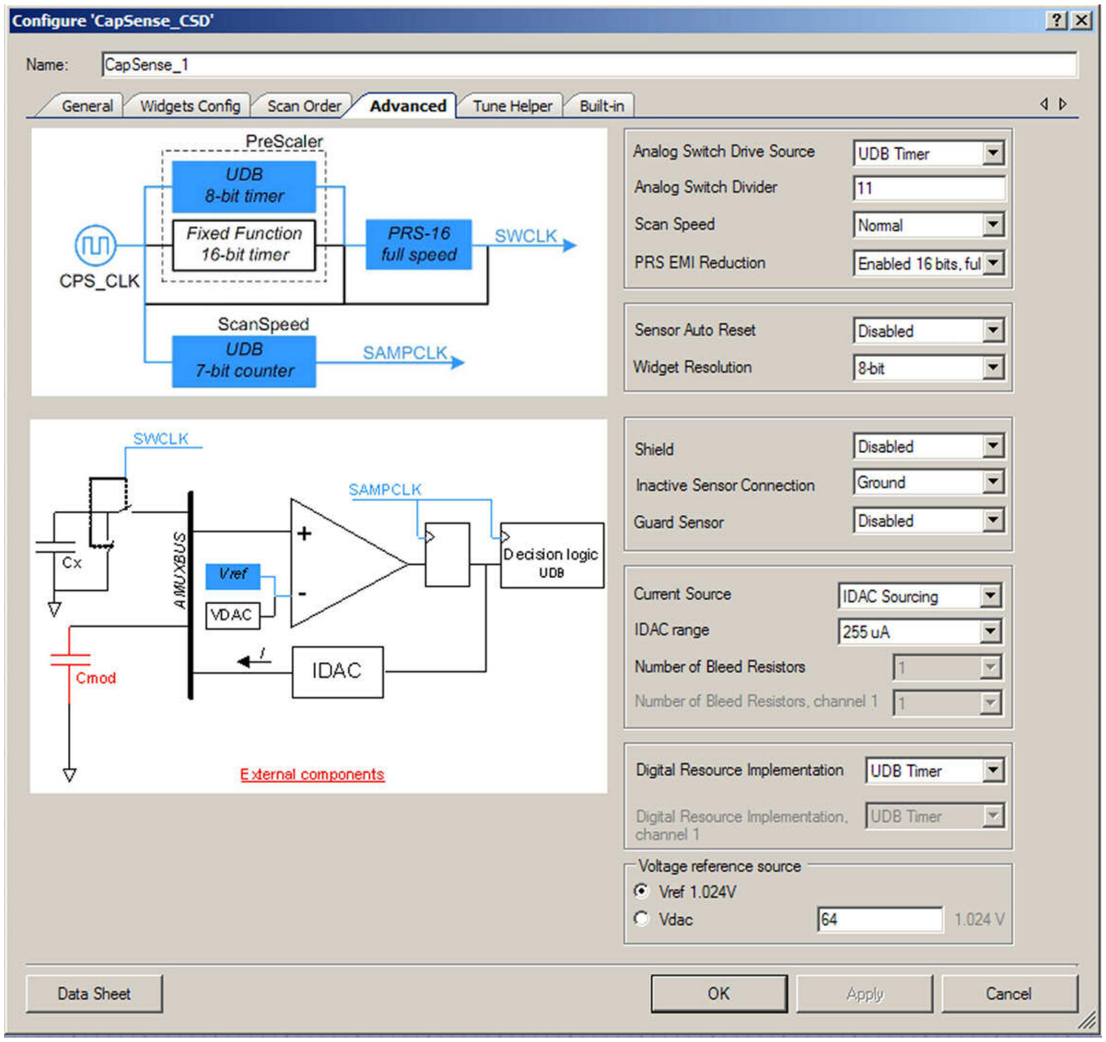
Widget List

Widgets are listed in alternating gray and orange rows in the table. All sensors associated with a widget share the same color to highlight different widget elements.

Proximity scan sensors can use dedicated proximity sensors, or they can detect proximity from a combination of dedicated sensors and/or other sensors. For example, the board may have a trace that goes all the way around an array of buttons and the proximity sensor may be made up of the trace and all of the buttons in the array. All of these sensors are scanned at the same time to detect proximity. A drop down is provided on proximity scan sensors to choose one or more sensors to scan to detect proximity.

Like proximity sensors, generic sensors can consist of multiple sensors as well. A generic sensor can get data from a dedicated sensor, any other existing sensor, or from multiple sensors. Select the sensors with the drop down provided.

Advanced Tab



Name: CapSense_1

Analog Switch Drive Source	UDB Timer
Analog Switch Divider	11
Scan Speed	Normal
PRS EMI Reduction	Enabled 16 bits, ful
Sensor Auto Reset	Disabled
Widget Resolution	8-bit
Shield	Disabled
Inactive Sensor Connection	Ground
Guard Sensor	Disabled
Current Source	IDAC Sourcing
IDAC range	255 uA
Number of Bleed Resistors	1
Number of Bleed Resistors, channel 1	1
Digital Resource Implementation	UDB Timer
Digital Resource Implementation, channel 1	UDB Timer

Voltage reference source
(● Vref 1.024V) ○ Vdac 64 1.024 V

Analog Switch Drive Source

This parameter specifies the source of the analog switch divider which determines the rate at which the sensors are switched to and from the modulation capacitor Cmod. Implementing the timer in the Fixed Function Timer blocks (FF Timer) results in minimal resources used while still

- Direct – Does not use FF Timer or UDB resources but limits device maximum clock rate to the same as the analog switch rate. Not recommended in most designs.
- UDB Timer – Uses UDB resources
- FF Timer – Default – Does not use UDB resources

Analog Switch Divider

This parameter specifies the value of the analog switch divider and determines the pre-charge switch output frequency. Valid range of values is [1…255]. Default value is 11.

This feature is disabled if **Analog Switch Drive Source** is set to Direct.

- The sensors are continuously switched to and from the modulation capacitor Cmod at the speed of the precharge clock. The Analog Switch Divider divides the CapSense CSD clock to generate the precharge clock. When the divider value is decreased, the sensors are switched faster and the raw counts increase and vice versa.

Scan Speed

This parameter specifies the CapSense CSD component digital logic clock frequency which determines the scan time of sensors. Slower scanning speeds take longer but provide the advantages of improved SNR, and better immunity to power supply and temperature changes,

- Slow – Divides the component input clock by 16
- Normal – Divides the component input clock by 8 – Default
- Fast – Divides the component input clock by 4
- Very Fast – Divides the component input clock by 2

Table 1. Scanning Time in µs vs Scan Speed and Resolution

Resolution, bits	Scanning speed			
	Very Fast	Fast	Normal	Slow
8	58	80	122	208
9	80	122	208	377
10	122	208	377	718
11	208	377	718	1400
12	377	718	1400	2770
13	718	1400	2770	5500
14	1400	2770	5500	10950
15	2770	5500	10950	21880

Resolution, bits	Scanning speed			
	Very Fast	Fast	Normal	Slow
16	5500	10950	21880	43720

Note Table 1 scan time is an estimate based on the following settings. Master Clock and CPU Clock = 48 MHz, CapSense CSD clock = 24 MHz, number of channels = 1. Scanning time was measured as time interval of one sensor scan. This time includes sensor setup time, sample conversion interval, and data processing time. These values can be used to estimate scanning speed for other clock rates and additional sensors by scaling the provided values linearly.

PRS EMI Reduction

This parameter specifies if the Psuedo Random Sequence (PRS) generator will be used to generate the analog pre-charge clock. Use of the PRS is recommended as it spreads the spectrum of the CapSense analog switching frequency reducing EMI emissions and sensitivity. The PRS clock source is provided by the Analog Switch Divider settings. If PRS EMI reduction is not enabled a single frequency will be used resulting in increased emissions of the fundamental frequency and harmonics:

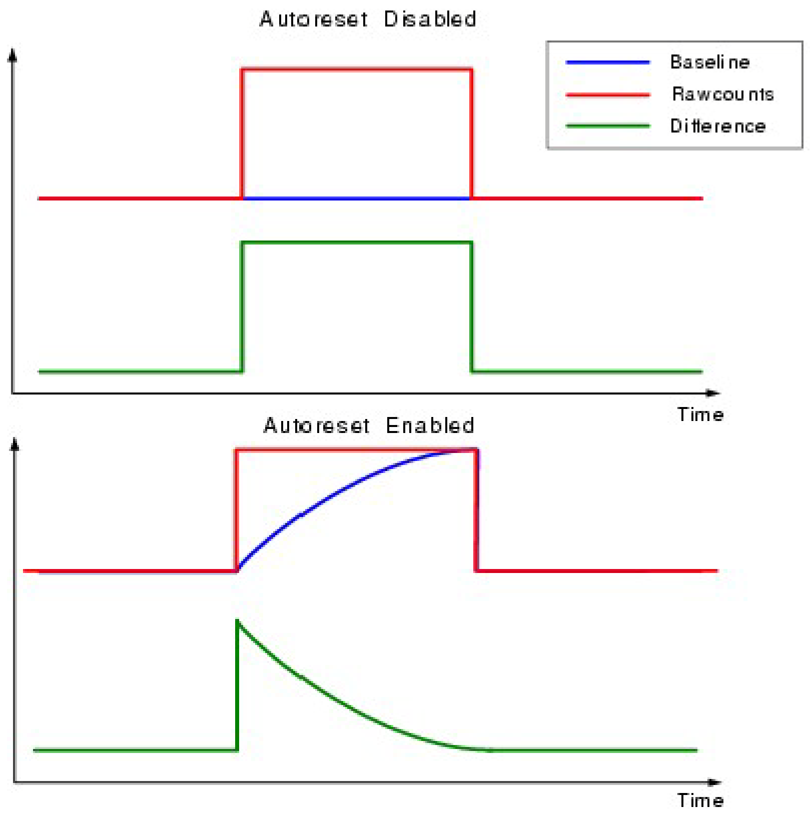
- Disabled
- Enabled 8 bits – 8 bit provides better SNR but the shorter repeat period increases EMI.
- Enabled 16 bits, full speed – Default – 16 bit provides a lower SNR but superior EMI reduction
- Enabled 16 bits, ¼ full speed - Requires a 4 times faster clock to obtain the same PRS clock output as “Enable 16 bits, full speed”

Sensor Auto Reset

This parameter enables auto reset which causes the baseline to always update regardless of whether the difference counts are above or below the noise threshold. When auto reset is disabled, the baseline only updates when difference counts are within the plus/minus noise threshold (the noise threshold is mirrored). You should leave this parameter Disabled unless you have problems with sensors permanently turning on when the raw count suddenly rises without anything touching the sensor.

- Enable – Auto reset ensures that the baseline is always updated avoiding missed button presses and stuck buttons but limits the maximum length of time a button will report as pressed. This setting limits the maximum time duration of the sensor (typical values are 5 – 10s), but it prevents the sensors from permanently turning on when the raw count suddenly rises without anything touching the sensor. This sudden rise can be caused by a large power supply voltage fluctuation, a high energy RF noise source, or a very quick temperature change.

- Disable – Default – Abnormal system conditions can cause the baseline to stop updating by continuously exceeding the noise threshold resulting in missed button presses or stuck buttons. The benefit is that a button can continue to report its pressed state indefinitely. The designer may need to provide an application dependant method of determining stuck or unresponsive buttons.



Widget Resolution

This parameter specifies the Signal resolution that the Widget reports. 8 bits (1 byte) is the default option and should be used for the vast majority of applications. If widget values exceed the 8 bit range the system is too sensitive and should be tuned to move the nominal value to approximately mid range (~128). Slider and Touchpad widgets that require high accuracy can benefit from 16 bit resolution. 16 bit resolution increases linearity by avoiding rounding errors possible with 8 bits but at the expense of additional SRAM usage of two bytes per sensor.

- 8 bits (1 byte) – Default
- 16 bits (2 bytes)

Shield electrode

This parameter specifies if the shield electrode output, which is used to remove the effects of water droplets and water films, is enabled or disabled. For more information about shield electrode usage refer to the **Shield electrode usage and Restrictions** section.

- Disable – Default
- Enable

Inactive Sensor Connection

This parameter defines the default sensor connection for all sensors not being actively scanned

- Ground – Default - Should be used for the vast majority of applications as it reduces noise on the actively scanned sensors.
- Hi-Z Analog - Leaves the inactive sensors at Hi-Z.
- Shield - Provides the shield waveform to all unscanned sensors, the amplitude of the shield signal is equal to Vddio. Provides increased water proofing and lower noise when used with the Shield Electrode.

Guard Sensor

This parameter enables the guard sensor, which helps detect water drops in an application that requires water proofing. This feature is enabled automatically if **Water Proofing and detection** (under the **General** tab) is checked. For more information about the Guard sensor, refer to "Guard Sensor Implementation" in the Functional Description section of this data sheet.

- Disable – Default
- Enable

Current Source

CapSense CSD requires a precision current source for detecting touch on the sensors. IDAC Sink and IDAC Source require the use of a hardware IDAC on the PSoC device. External Resistor uses a user supplied resistor on the PCB rather than an IDAC and is useful in IDAC constrained applications.

- IDAC Sourcing – Default - The IDAC sources the current into the modulation capacitor CMOD. The analog switches are configured to alternate between the modulation capacitor CMOD and GND providing a sink for the current. IDAC Sourcing is recommended for most designs because it provides the greatest Signal To Noise Ratio of the three methods but may require an additional VDAC resource to set the Vref level that the other modes do not require.
- IDAC Sinking - The IDAC sinks current from the modulation capacitor CMOD. The analog switches are configured to alternate between Vdd and the modulation capacitor CMOD

providing a source for the current. Works well in most designs although SNR is generally not as high as the IDAC Sourcing mode.

- External Resistor - This functions the same as the IDAC sinking configuration except the IDAC is replaced with a bleed resistor to ground, Rb. The bleed resistor is connected between the modulation capacitor, Cmod and a GPIO. The GPIO is configured to Open-Drain Drives Low drive mode allowing Cmod to be discharged through Rb. This mode requires the least analog resources and should only be used when resource constraints require. Because this mode does not require an IDAC or VDAC it can result in the lowest power configuration of the component if power is critical system consideration.

IDAC Range

This parameter specifies the IDAC Range of the Current Source. This parameter is disabled if **Current Source** is set to External Resistor. The Default is the best choices for almost all CapSense designs. The lower and higher current ranges are generally only used with non touch capacitive based sensors.

- 32uA
- 255uA – Default
- 2.04mA

Number of Bleed Resistors, channel 0/channel 1

This parameter specifies the number of bleed resistors. The maximum number of bleed resistors is three per channel. This feature is disabled if the **Current Source** is set to IDAC Source or IDAC Sink. Multiple bleed resistors are supported to allow different currents for up to three groups of sensors which aids in system tuning. Most designs with a similar sensor size require only one bleed resistor.

Digital Resource Implementation, channel 0/channel 1

This parameter specifies the type of resources to be used for implementing the digital portion of CapSense which includes a timer and a counter. For most designs this parameter should not be changed as it is designed to provide maximum implementation flexibility.

- UDB Timer – Default – Most flexible implementation but uses valuable UDB resources
- FF Timer – FF Timer implementation frees UDB resources but does not support Scan Speed = Very Fast.

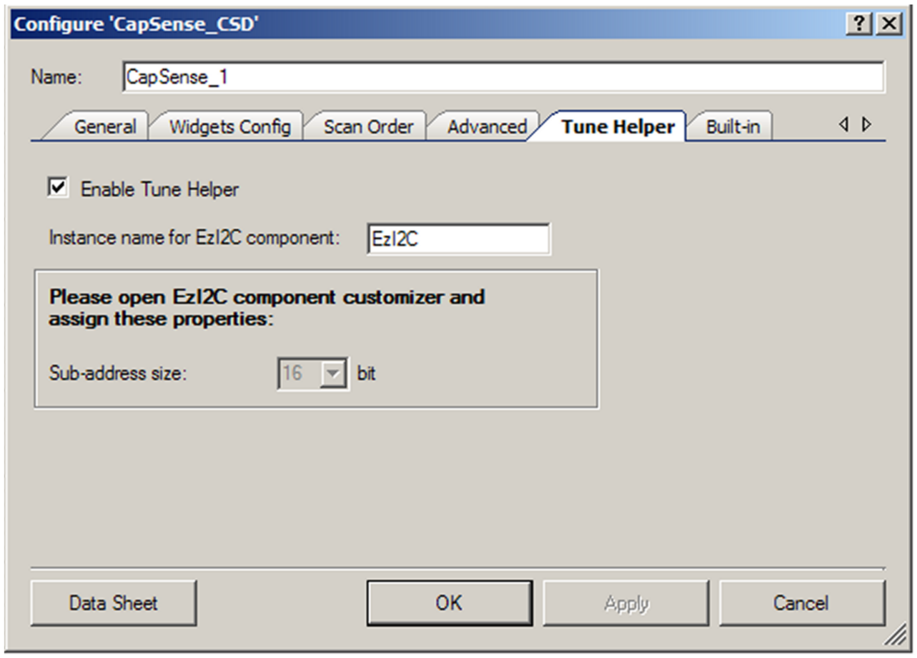
Voltage Reference Source

This parameter specifies the type and level of the reference source voltage. It is best to have as high a reference voltage for IDAC Source mode and as low a reference voltage for IDAC Sink or External Resistor Current Source modes.

- 1.024V – Default – Best for IDAC sink mode
- VDAC – Best for IDAC source mode. Allows adjusting the reference voltage using a Voltage DAC to maximize the available range. Reference source VDAC is only available when **Current Source** is set to IDAC Source and requires a VDAC device resource. As the reference voltage is increased so is sensitivity but the influence on the shield electrode is decreased.

When VDAC is selected the CapSense buffer isn't used, as it is designed for low voltage. This causes Cmod to be charged to Vref from VDAC on start up. The amount of time required to charge Cmod to Vref, may cause baseline initialization fails. Typically, double baseline initialization will solve the problem.

Tune Helper Tab



Name: CapSense_1

☑ Enable Tune Helper

Instance name for EzI2C component: EzI2C

Please open EzI2C component customizer and assign these properties:

Sub-address size: 16 bit

Enable Tune Helper

This parameter adds functions to support communication with the Tuner GUI with minimal user effort. This feature should be checked if the Tuner GUI will be used. If this option is not checked, the communication functions are still provided but do nothing. Therefore, when tuning is complete or the tuning method is changed you do not need to remove these functions. Default – Checked.

Instance name for EZI2C component

This parameter defines the instance name for the EZI2C component in your design to be used for communication with the Tuner GUI.

Note There is no real time Design Rule Check to ensure the actual instance name matches the instance name entered here. You must make sure they match. If the names do not match, build errors will be generate during the project build do to misnamed APIs.

For more information about how to use Tuner GUI, refer to the Tuner GUI User Guide section of this data sheet.

Placement

Not applicable

Resources

CapSense CSD analog and pins resources.

Resources	Analog resources			Pins (per External I/O)
	VIDAC	**Comparator**	**CapSense Buffers**	
Channels: 1 Current Mode: External resistor	0	1	1	2 + Shield + SensorsNumber
Channels: 2 Current Mode: External resistor	0	2	2	4 + Shield + SensorsNumber
Channels: 1 Current Mode: IDAC sinking	1	1	1	1 + Shield + SensorsNumber
Channels: 2 Current Mode: IDAC sinking	2	2	2	2 + Shield + SensorsNumber
Channels: 1 Current Mode: IDAC sorcing Vref: VDAC	2	1	1	1 + Shield + SensorsNumber
Channels: 2 Current Mode: IDAC sorcing Vref: VDAC	4	2	2	2 + Shield + SensorsNumber

CapSense CSD digital resources (only scanning and sleep APIs are included to Flash and RAM usage).

	Digital resources						API Memory (Bytes)	
Description	Data-paths	Macro cells	Status Registers	Control Registers	Counter 7	Interrupt	Flash	RAM
Channels: 1 Current Mode: External resistor	4	19	0	1	1	1	1270	11
Channels: 2 Current Mode: External resistor	6	31	0	1	1	2	2262	17
Channels: 1 Current Mode: IDAC sinking	4	19	0	1	1	1	1345	10
Channels: 2 Current Mode: IDAC sinking	6	31	0	1	1	2	2446	15
Channels: 1 Current Mode: IDAC sorcing Vref: VDAC	4	18	0	1	1	1	1452	11
Channels: 2 Current Mode: IDAC sorcing Vref: VDAC	6	30	0	1	1	2	2656	17

CapSense CSD high level API resources

	API Memory (Bytes)	
Project description	Flash	RAM
Widgets type: Buttons	1197	22

Project description	API Memory (Bytes)	
	Flash	RAM
Count: 4		
Widgets type: Non-diplexed linear slider Size: 5 sensors	1866	25
Widgets type: Diplexed linear slider Size: 5 sensors	2304	25
Widgets type: Matrix buttons Size: 5x5 sensors	1526	55
Widgets type: Radial slider Size: 5 sensors	1704	25
Widgets type: Touch-pad Size: 5x5 sensors	2289	48

Tuner GUI User Guide

This section provides instructions and information that will help you use the CapSense Tuner.

The CapSense Tuner assists in tuning the CapSense component to the specific environment of the system when in manual tuning mode. It is also capable of displaying the tuning values (read only) and performance when the component is in SmartSense mode. No tuning is supported when the component is in no tuning mode as all parameters are stored in Flash and are read only for minimum SRAM usage.

CapSense Tuning Process

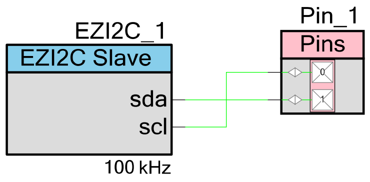
The following is the typical process for using and tuning a CapSense component:

Create a design in PSoC Creator

Refer to the PSoC Creator Help as needed.

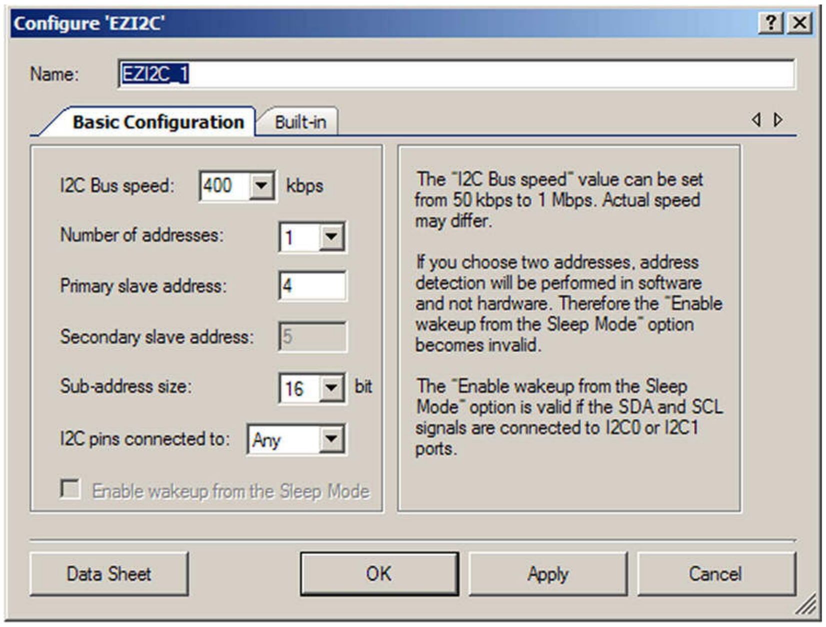
Place and Configure EZI2C Component

1. Drag an EZI2C component from the Component Catalog onto your design.



Double-click it to open the Configure dialog.

Change the parameters as follows, and click **OK** to close the dialog.



Name: EZI2C_1

I2C Bus speed: 400 kbps

Number of addresses: 1

Primary slave address: 4

Secondary slave address: 5

Sub-address size: 16 bit

I2C pins connected to: Any

☐ Enable wakeup from the Sleep Mode

The "I2C Bus speed" value can be set from 50 kbps to 1 Mbps. Actual speed may differ.

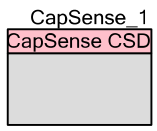
If you choose two addresses, address detection will be performed in software and not hardware. Therefore the "Enable wakeup from the Sleep Mode" option becomes invalid.

The "Enable wakeup from the Sleep Mode" option is valid if the SDA and SCL signals are connected to I2C0 or I2C1 ports.

- Sub-address size must be "16 bit."
- The instance Name must match the name used on the CapSense CSD Configure dialog, under **Tune Helper** tab in order for the generated APIs to function

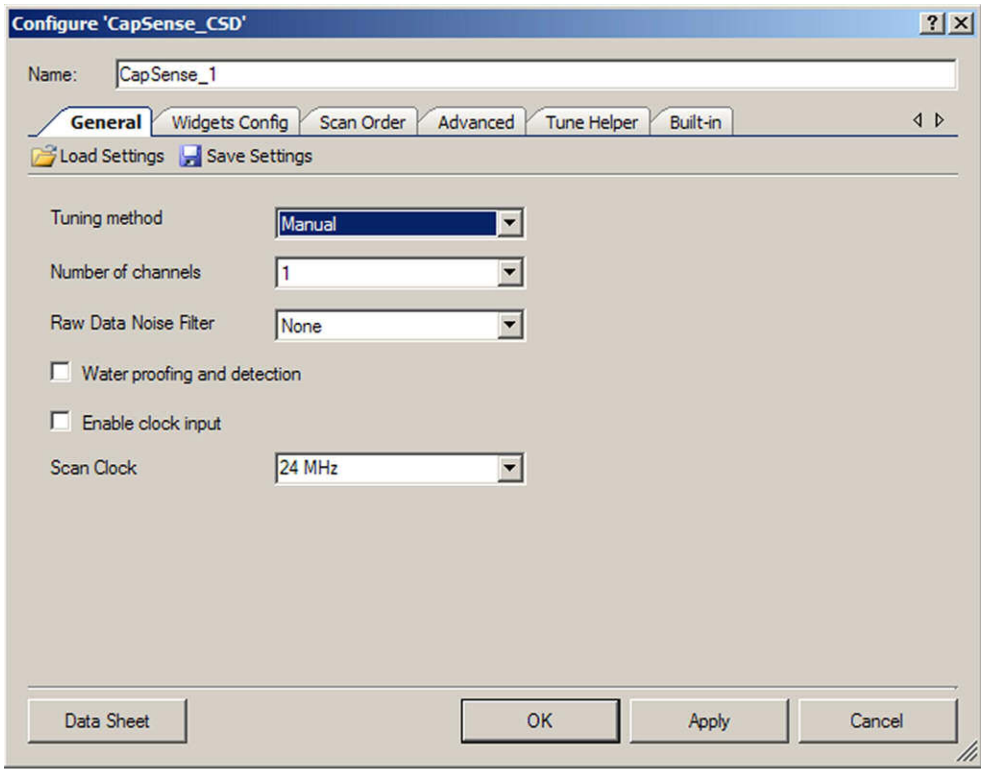
Place and Configure the CapSense Component

1. Drag a CapSense_CSD component from the Component Catalog onto your design.



Double-click it to open the Configure dialog.

Change CapSense CSD parameters as required for your application. Select **Tuning method** as Manual or Auto (SmartSense). Click **OK** to close the dialog and save the selected parameters.



Tuning method: Manual

Number of channels: 1

Raw Data Noise Filter: None

☐ Water proofing and detection

☐ Enable clock input

Scan Clock: 24 MHz

Selecting Auto (SmartSense)

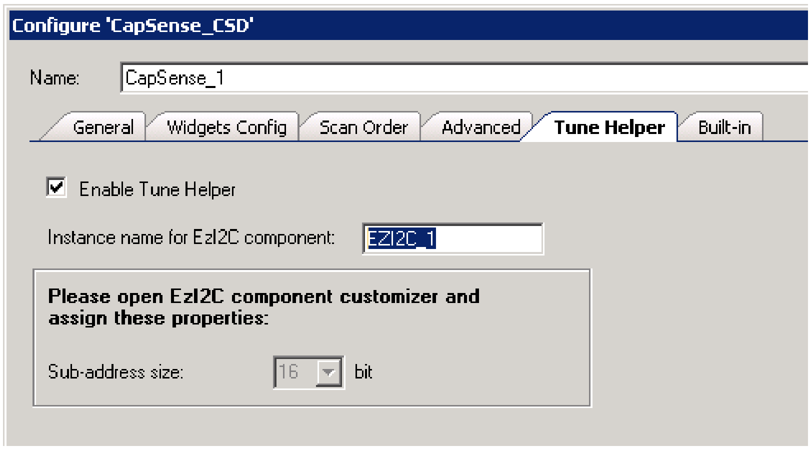
Auto (SmartSense) allows tuning of the CapSense CSD component to the specifics of the system automatically. CapSense CSD parameters are computed at runtime by firmware. Additional RAM and CPU time are used in this mode. Auto (SmartSense) eliminates the error prone and repetitive process of manually tuning the CapSense CSD component parameters to ensure proper system operation. Selecting Auto (SmartSense) tunes the following CSD parameters:

Parameter	Calculation
Finger Threshold	Calculated continuously during sensor scanning.
Noise Threshold	Calculated continuously during sensor scanning.
IDAC Value	Calculated once on CapSense CSD start up.
Analog Switch Divider	Calculated once on CapSense CSD start up.
Scan Resolution	Calculated once on CapSense CSD start up.

Restrictions of hardware parameters for Auto(SmartSense) tuning method:

Parameter	Required Setting
Scan Clock	Clock must be internal ("Enable clock input" in General tab set to false).
Current Source	IDAC Sourcing.
PRS EMI Reduction	Enabled 16 bits.
Scan Speed	Normal
Vref	1.024 V

In the Tune Helper tab: The EzI2C component instance name must be entered and Enable Tune Helper checkbox must be checked



Name: CapSense_1

☑ Enable Tune Helper

Instance name for EzI2C component: EZI2C_1

Please open EzI2C component customizer and assign these properties:

Sub-address size: 16 bit

Add Code

Add tuner initialization and communication code to the projects *main.c* file. Example *main.c* file:

```
void  main()
{
    CYGlobalIntEnable;
    CapSense_1_TunerStart();
    while(1)
    {
        CapSense_1_TunerComm();
    }
}
```

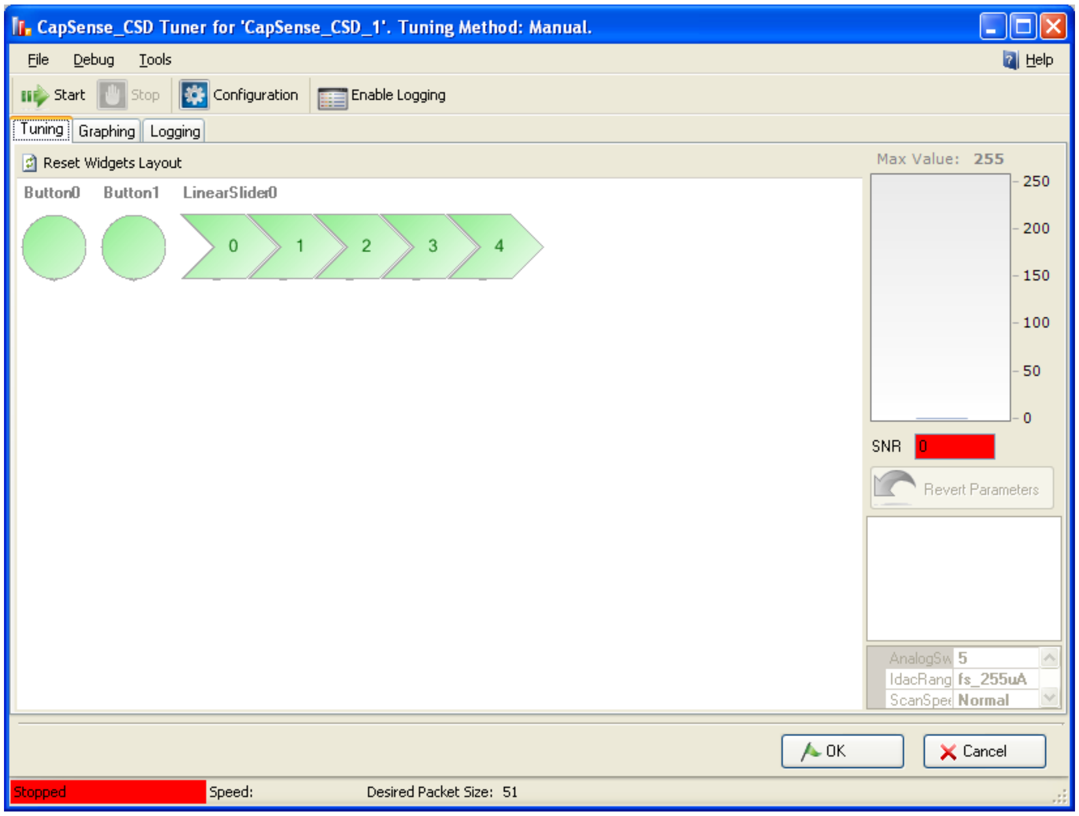
Build the design and program the PSoC device

Refer to PSoC Creator Help as needed.

Launch the Tuner application

Right-click the CapSense CSD component icon and select **Launch Tuner** from the context menu.

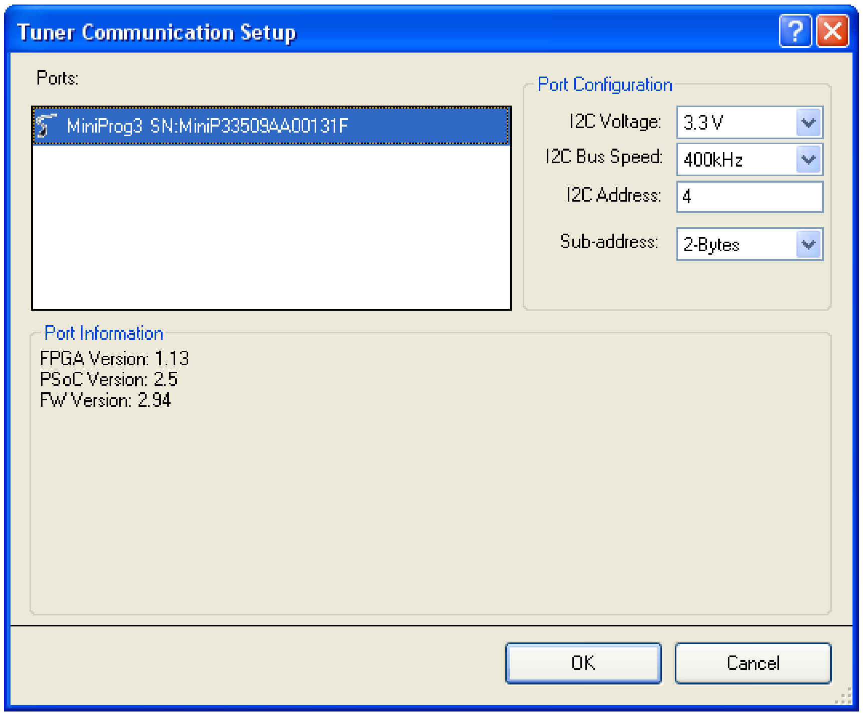
The Tuner application opens.



AnalogSw 5
IdacRang fs_255uA
ScanSpee Normal

Configure communication parameters

1. Click Configuration to open the Tuner Communication dialog.



MiniProg3 SN:MiniP33509AA00131F

I2C Voltage: 3.3 V
I2C Bus Speed: 400kHz
I2C Address: 4
Sub-address: 2-Bytes

FPGA Version: 1.13
PSoC Version: 2.5
FW Version: 2.94

Set the communication parameters and press OK.

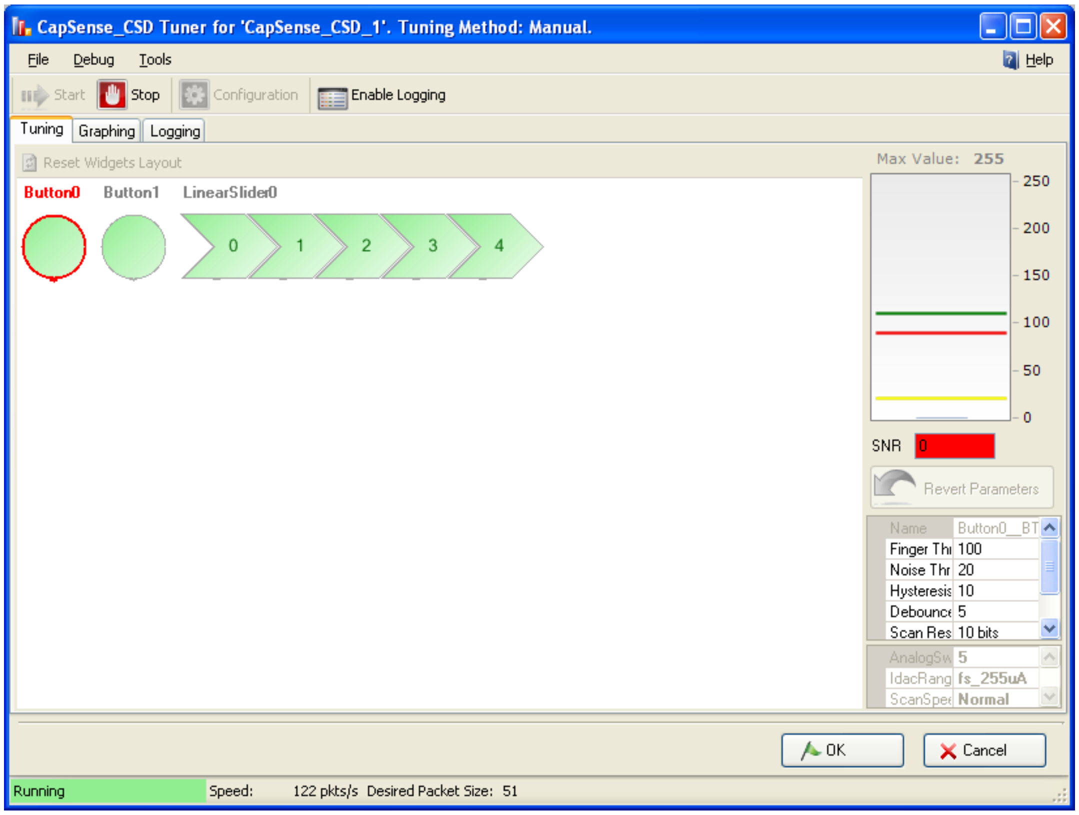
Important: Properties must be identical to those in the EZ I^{2}C component: **I2C Bus Speed**, **I2C Address**, **Sub-address** = 2-Bytes.

Start tuning

Click **Start** on the tuning GUI. All of the CapSense elements start showing their values.

Edit CapSense parameter values

Edit a parameter value for one of the elements and it is automatically applied after pressing "Enter" key or moving to another option. The GUI continues to show the scanning data but it is now altered based on the application of the updated parameter. Refer to the Tuner GUI Interface section later in this data sheet.



Name	Button0__BT
Finger Thi	100
Noise Thr	20
Hysteresis	10
Debounce	5
Scan Res	10 bits

AnalogSw	5
IdacRang	fs_255uA
ScanSpee	Normal

Running Speed: 122 pkts/s Desired Packet Size: 51

Repeat as needed

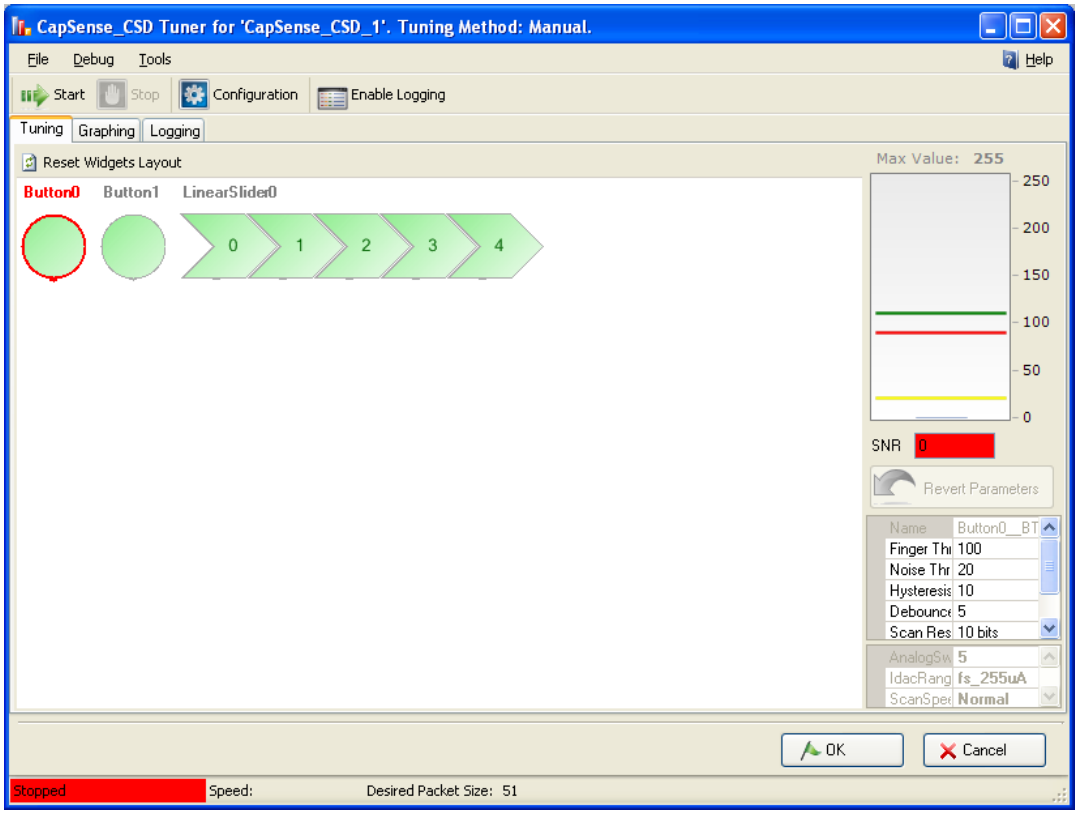
Repeats steps as needed until tuning is complete and the CapSense component provides reliable touch sensor results.

Close the Tuner application

Click the **OK** button and the parameters are written back to the CapSense_CSD instance and the Tuner application dialog closes.

Tuner GUI Interface

General Interface



Max Value: 255

SNR 0

Name	Button0__BT
Finger Thr	100
Noise Thr	20
Hysteresis	10
Debounce	5
Scan Res	10 bits
AnalogSw	5
IdacRang	fs_255uA
ScanSpe	Normal

Stopped Speed: Desired Packet Size: 51

Top panel buttons:

- **Start** (or main menu item **Debug > Start**) – Starts reading and displaying data from the chip. Also starts graphing and logging if configured.
- **Stop** (or main menu item **Debug > Stop**) – Stops reading and displaying data from the chip.
- **Configuration** (or main menu item **Debug > Configuration**) – Opens the Communication Configuration dialog.
- **Enable Logging** (or main menu item **Debug > Start**) – Enables logging of data received from device to a log file.

Main Menu:

- **File > Settings > Load Settings from File** – Imports settings from a XML tuning file and loads all data into the tuner.
- **File > Help** – Opens help file.
  - Other items duplicate functionality of top and bottom panel buttons.

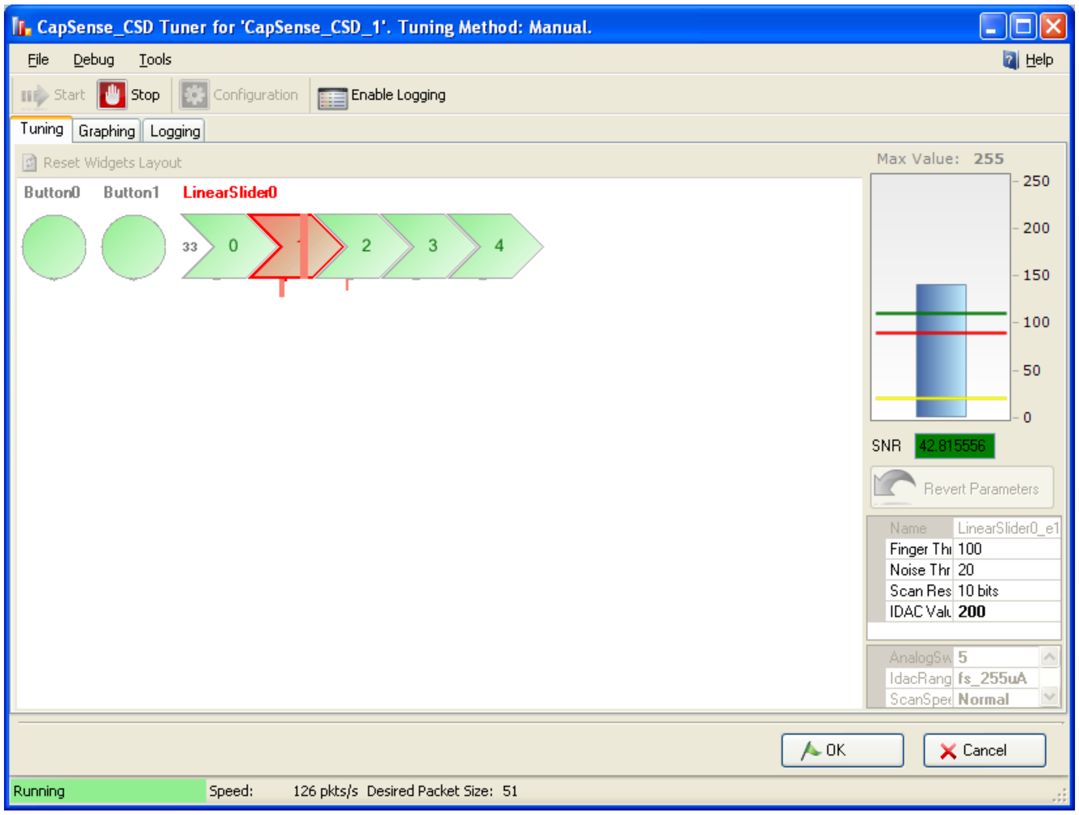
Tabs

- **Tuning Tab** – Displays all of the component widgets as configured on a workspace. This allows you to arrange the widgets similar to how they appear on the physical PCB or enclosure. This tab is used for tuning widget parameters and visualization widgets data and states.
- **Graphing Tab** – Displays detailed individual widget data on charts.
- **Logging Tab** – Provides logging data functionality and debugging features.

Bottom panel buttons:

- **OK** (or main menu item **File > Apply Changes and Close**) – Commits the current values of parameters to the CapSense component instance and exits the GUI.
- **Cancel** (or main menu item **File > Exit**) – Exits the GUI without committing the values of parameters to the component instance.

Tuning Tab



Max Value: 255

SNR 42.815556

Name	LinearSlider0_e1
Finger Thr	100
Noise Thr	20
Scan Res	10 bits
IDAC Valu	200

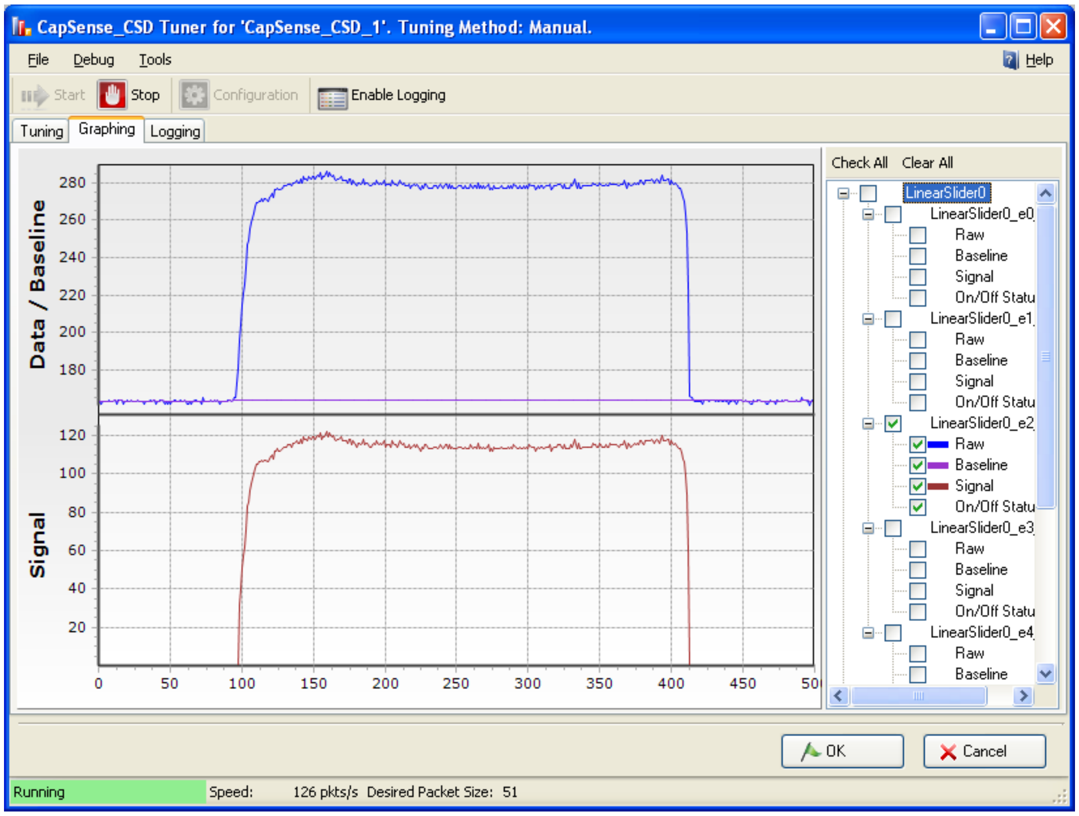
AnalogSw	5
IdacRang	fs_255uA
ScanSpe	Normal

Running Speed: 126 pkts/s Desired Packet Size: 51

- **Widgets schematic** – contains a graphical representation of all of the configured widgets. If a widget is composed of more than one sensor the individual sensors may be selected for detailed analysis.

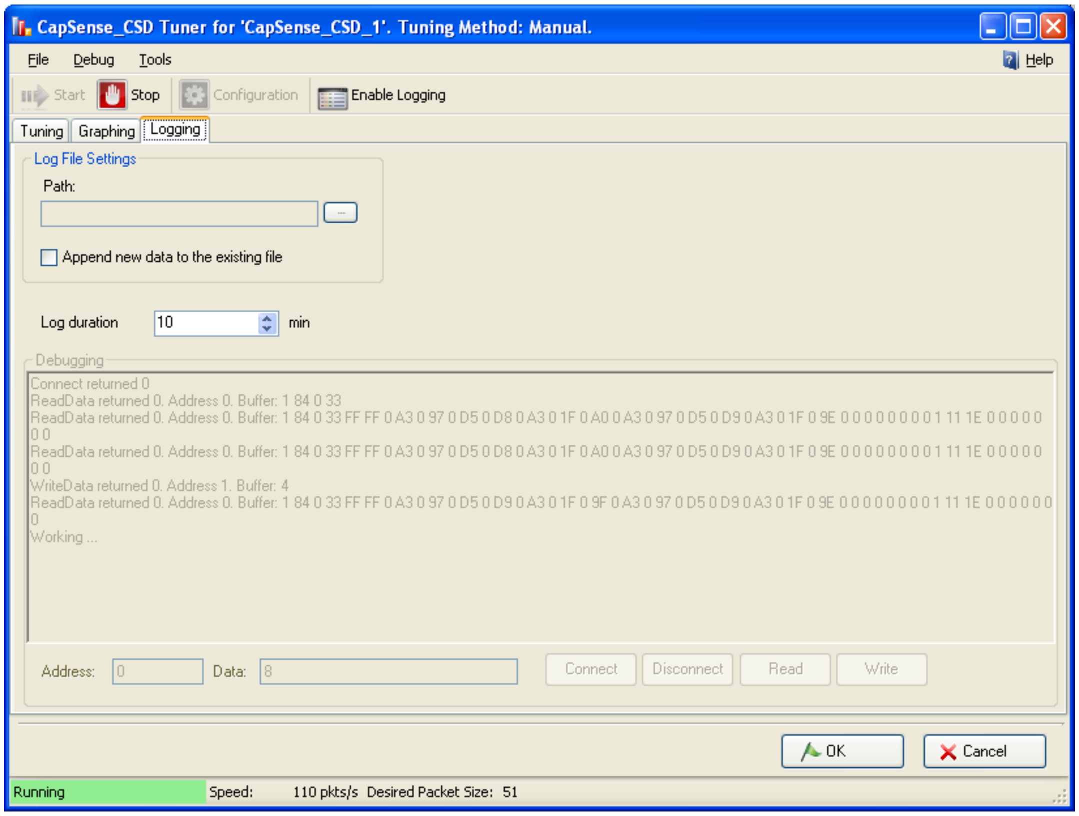
- **Bar graph** – Displays signal values for the selected sensor.
  - The maximum scale of the detailed view bar graph can be adjusted by double clicking on Max Value label. Valid range is between 1 and 255, default is 255.
  - The current finger turn on threshold is displayed as a **green line** across the bar graph.
  - The current finger turn off threshold is displayed as a **red line** across the bar graph.
  - The current noise threshold is displayed as a **yellow line** across the bar graph.
- **SNR** – The signal to noise ratio is computed in real time for the selected sensor. SNR values below 5 are poor and colored red, 5 to 10 are marginal and yellow, and greater than 10 is good and colored green. SNR value is calculated based on previously received data.
- **Revert Parameters** button – Resets the parameters to their initial values and sends those values to the chip. Initial values are what were displayed when the GUI was launched
- **Sensor properties** – Displays the properties for the selected sensor based on the widget type. It is located on the right side panel.
- **General CapSense properties** (Read Only) – Displays global properties for the CapSense CSD component that cannot be changed at run time. These are for reference only. It is located on the bottom of the right side panel
- **Widget controls context menu** (this functionality applies only to the layout of widget controls in GUI):
  - Send To Back – Sends widget control to the back of the view.
  - Bring To Front – Brings widget control to the front of the view.
  - Rotate Clockwise 90 – Rotates widget control 90 degrees clockwise. (Only for Linear Sliders)
  - Rotate Counter Clockwise 90 – Rotates widget control 90 degrees counter clockwise. (Only for Linear Sliders)
  - Flip Sensors – Reverses the order of the sensors. (Only for Linear and Radial Sliders)
  - Flip Columns Sensors – Reverses the order of the Columns sensors. (Only for Touch Pads and Matrix Buttons)
  - Flip Row Sensors – Reverses the order of the Row sensors. (Only for Touch Pads and Matrix Buttons)
  - Exchange Columns and Rows – Columns sensors become rows and rows sensors become columns. (Only for Touch Pads and Matrix Buttons)

Graphing Tab



- **Chart area** – Displays charts for selected items from the tree view.
  - ❑ Chart area right click context menu item **Export to .jpg** makes a screenshot of the chart area and saves it as a .jpg picture.
- **Tree view** – Provides all combinations of data for widgets and sensors which can be shown on the chart and logged to a file if the logging feature is enabled. The On/Off Status data value can only be logged, it can’t be show on a chart.

Logging Tab



Append new data to the existing file

Log duration 10 min

Debugging

```
Connect returned 0
ReadData returned 0. Address 0. Buffer: 1 84 0 33
ReadData returned 0. Address 0. Buffer: 1 84 0 33 FF FF 0 A3 0 97 0 D5 0 D8 0 A3 0 1F 0 A0 0 A3 0 97 0 D5 0 D9 0 A3 0 1F 0 9E 0 0 0 0 0 0 0 0 1 11 1E 0 0 0 0 0
0 0
ReadData returned 0. Address 0. Buffer: 1 84 0 33 FF FF 0 A3 0 97 0 D5 0 D8 0 A3 0 1F 0 A0 0 A3 0 97 0 D5 0 D9 0 A3 0 1F 0 9E 0 0 0 0 0 0 0 0 1 11 1E 0 0 0 0 0
0 0
WriteData returned 0. Address 1. Buffer: 4
ReadData returned 0. Address 0. Buffer: 1 84 0 33 FF FF 0 A3 0 97 0 D5 0 D9 0 A3 0 1F 0 9F 0 A3 0 97 0 D5 0 D9 0 A3 0 1F 0 9E 0 0 0 0 0 0 0 0 1 11 1E 0 0 0 0 0 0
0
Working ...
```

Address: 0 Data: 8

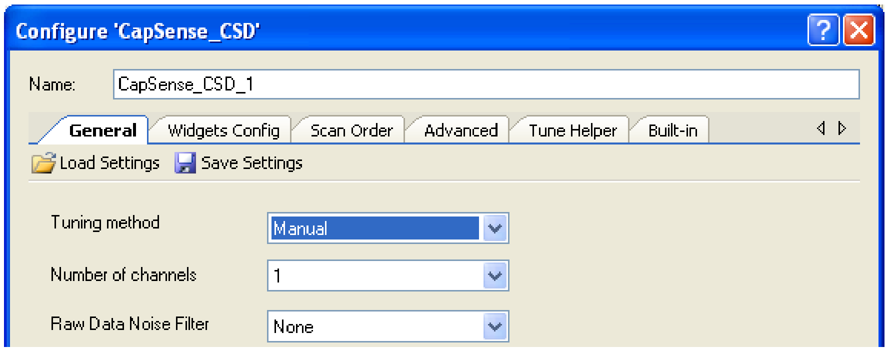
Running Speed: 110 pkts/s Desired Packet Size: 51

- Data that will be logged is selected by checking checkboxes on the Tree View of the Graphing tab.
- Path – Defines log file path (file extension is .csv).
- Append new data to existing file checkbox – If checked, new data is appended to an existing file. If not checked, old data will be erased from the file and replaced with the new data.
- Log duration – Defines log duration in minutes.
- Connect – Connects to the PSoC device.
- Disconnect – Disconnects from the PSoC device.
- Read – Reads data from the PSoC device. The address field defines the address in the buffer. Data field defines number of bytes to read.
- Write – Writes data to the PSoC device. The address field defines the address in the buffer. Data field defines the data to write.

Save/Load Settings feature

The tuner GUI also can be opened as standalone application. In this case the user has to use the Save and Load Settings feature of the CapSense CSD component Tuner GUI.

1. Click the Save Settings button in the customizer



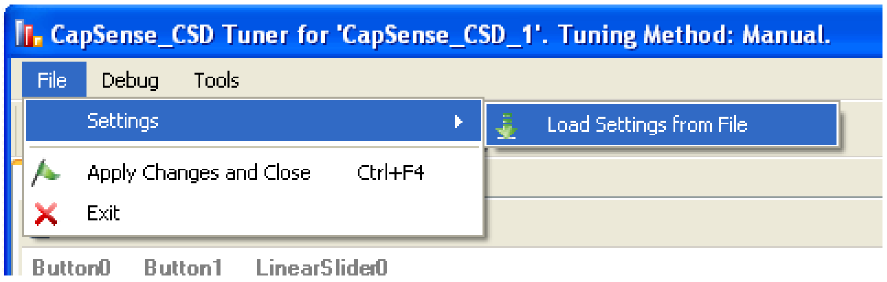
Name: CapSense_CSD_1

Tuning method: Manual

Number of channels: 1

Raw Data Noise Filter: None

2. In Save File Dialog window specify name of the file and location where it will be saved.
3. Open Tuner window and click File > Settings > Load Settings from File


CapSense_CSD Tuner for 'CapSense_CSD_1'. Tuning Method: Manual.
File
Debug
Tools
Settings
Load Settings from File
Apply Changes and Close Ctrl+F4
Exit
Button0
Button1
LinearSlider0

4. In File Open Dialog point to the previously saved file with the component settings. Settings will automatically load into the Tuner.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table provides an overview of each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "CapSense_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "CapSense".

General APIs

These are the general CapSense API functions that place the component into operation or halt operation:

Function	Description
CapSense_Start	Preferred method to start the component. Initializes registers and enables active mode power template bits of the sub components used within CapSense
CapSense_Stop	Disables component interrupts, and calls CapSense_ClearSensors() to reset all sensors to an inactive state.
CapSense_Sleep	Prepares the component for the device entering a low power mode. Disables Active mode power template bits of the sub components used within CapSense, saves non-retention registers and resets all sensors to an inactive state.
CapSense_WakeUp	Restores CapSense configuration and non-retention register values after the device wake from a low power mode sleep mode.
CapSense_Init	Initializes the default CapSense configuration provided with the customizer
CapSense_Enable	Enables the Active mode power template bits of the sub components used within CapSense.
CapSense_SaveConfig	Saves the configuration of CapSense non-retention registers. Resets all sensors to an inactive state.
CapSense_RestoreConfig	Restores CapSense configuration and non-retention register values.

void CapSense_Start (void)

Description:	This is the preferred method to begin component operation. CapSense_Start() calls the CapSense_Init() function, and then calls the CapSense_Enable() function. Initializes registers and starts the CSD method of the CapSense component. Resets all sensors to an inactive state. Enables interrupts for sensors scanning. When SmartSense tuning mode is selected the tuning procedure is applied for all sensors. The CapSense_Start() routine must be called before any other API routines.
Parameters:	None
Return Value:	None
Side Effects:	None

void CapSense_Stop (void)

Description:	Stops the sensor scanning, disables component interrupts, and resets all sensors to an inactive state. Disables Active mode power template bits for the subcomponents used within CapSense.
Parameters:	None
Return Value:	None
Side Effects:	This function should be called after all scanning is completed.

void CapSense_Sleep(void)

Description:	This is the preferred method to prepare the component for device low power modes. Disables Active mode power template bits for the sub components used within CapSense. Calls CapSense_SaveConfig() function to save customer configuration of CapSense non-retention registers and resets all sensors to an inactive state.
Parameters:	None
Return Value:	None
Side Effects:	This function should be called after scans will be completed. This function does not put pins used by CapSense component into lowest power consumption state. To change a pin’s drive mode, use the functions described in the “Pins APIs” section.

void CapSense_WakeUp(void)

Description: Restores the CapSense configuration and non-retention register values. Restores the enabled state of the component by setting Active mode power template bits for the subcomponents used within CapSense.

Parameters: None

Return Value: None

Side Effects: This function does not restore pins used by the CapSense component to the state it was before.

void CapSense_Init(void)

Description: Initializes the default CapSense configuration provided by the customizer that defines component operation and resets all sensors to an inactive state.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_Enable(void)

Description: Enables Active mode power template bits for the subcomponents used within CapSense.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_SaveConfig(void)

Description: Saves the configuration of CapSense non-retention registers. Resets all sensors to an inactive state.

Parameters: None

Return Value: None

Side Effects: This function should be called after scanning is complete.

This function does not put pins used by CapSense component into lowest power consumption state. To change a pin's drive mode, use the functions described in the "Pins APIs" section

void CapSense_RestoreConfig(void)

Description: Restores CapSense configuration and non-retention registers.

Parameters: None

Return Value: None

Side Effects: This function should be called after scanning is complete.

This function does not restore pins used by the CapSense component to the state it was before.

Scanning Specific APIs

These API functions are used to implement CapSense sensor scanning.

Function	Description
CapSense_ScanSensor	Sets scan settings and starts scanning a sensor or group of combined sensors on each channel.
CapSense_ScanEnabledWidgets	The preferred scanning method. Scans all of the enabled widgets.
CapSense_IsBusy	Returns status of sensor scanning.
CapSense_SetScanSlotSettings	Sets the scan settings of the selected scan slot (sensor or pair of sensors).
CapSense_ClearSensors	Resets all sensors to the non-sampling state.
CapSense_EnableSensor	Configures the selected sensor to be scanned during the next scanning cycle.
CapSense_DisableSensor	Disables the selected sensor so it is not scanned in the next scanning cycle.
CapSense_ReadSensorRaw	Returns sensor raw data from the CapSense_SensorResult[ ] array.
CapSense_SetRBleed	Sets the pin to use for the bleed resistor (Rb) connection if multiple bleed resistors are used.

void CapSense_ScanSensor (uint8 sensor)

Description:	Sets scan settings and starts scanning a sensor or pair of sensors on each channel. If two channels are configured, two sensors may be scanned at the same time. After scanning is complete the isr copies the measured sensor raw data to the global raw sensor array. Use of the isr ensures this function is non-blocking. Each sensor has a unique number within the sensor array. This number is assigned by the CapSense customizer in sequence.
Parameters:	(uint8) sensor: Sensor number
Return Value:	None
Side Effects:	None

void CapSense_ScanEnabledWidgets (void)

Description:	This is the preferred method to scan all of the enabled widgets. Starts scanning a sensor or pair of sensors within the enabled widgets. The isr continues scanning sensors until all enabled widgets are scanned. Use of the isr ensures this function is non-blocking. All widgets are enabled by default except proximity widgets. Proximity widgets must be manually enabled as their long scan time is incompatible with the fast response desired of other widget types.
Parameters:	None
Return Value:	None
Side Effects:	If no widgets are enabled the function call has no effect.

uint8 CapSense_IsBusy (void)

Description:	Returns status of sensor scanning.
Parameters:	None
Return Value:	(uint8) Returns the state of scanning. ‘1’ – scanning in progress, ‘0’ – scanning completed.
Side Effects:	None

void CapSense_SetScanSlotSettings (uint8 slot)

Description:	Sets the scan settings provided in the customizer or wizard of the selected scan slot (sensor or pair of sensors for a two channel design). The scan settings provide an IDAC value (for IDAC configurations) for every sensor as well as resolution. The resolution is the same for all sensors within a widget.
Parameters:	(uint8) slot: Scan slot number
Return Value:	None
Side Effects:	None

void CapSense_ClearSensors (void)

Description:	Resets all sensors to the non-sampling state by sequentially disconnecting all sensors from the Analog MUX Bus and connecting them to the inactive state.
Parameters:	None
Return Value:	None
Side Effects:	None

void CapSense_EnableSensor(uint8 sensor)

Description:	Configures the selected sensor to be scanned during the next measurement cycle. The corresponding pins are set to Analog High-Z mode and connected to the Analog Mux Bus. This also affects the comparator output.
Parameters:	(uint8) sensor: Sensor number
Return Value:	None
Side Effects:	None

void CapSense_DisableSensor(uint8 sensor)

Description:	Disables the selected sensor. The corresponding pins are disconnected from the Analog Mux Bus and put into the inactive state.
Parameters:	(uint8) sensor: Sensor number
Return Value:	None
Side Effects:	None

uint16 CapSense_ReadSensorRaw (uint8 sensor)

Description:	Returns sensor raw data from the global CapSense_SensorResult[ ] array. Each scan sensor has a unique number within the sensor array. This number is assigned by the CapSense customizer in sequence. Raw data may be used to perform calculations outside of the CapSense provided framework.
Parameters:	(uint8) sensor: Sensor number
Return Value:	(uint16) Current Raw data value
Side Effects:	None

void CapSense_SetRBleed(uint8 rbleed)

Description: Sets the pin to use for the bleed resistor (Rb) connection. This function can be called at runtime to select the current Rb pin setting from those defined in the customizer. The function overwrites the component parameter setting. This function is available only if Current Source is set to External Resistor.

This function is effective when some sensors need to be scanned with different bleed resistor values. For example, regular buttons can be scanned with a lower value of bleed resistor. The proximity detector can be scanned less often with a larger bleed resistor to maximize proximity detection distance. This function can be used in conjunction with the CapSense_ScanSensor() function.

Parameters: (uint8) rbleed: Ordered number for bleed resistor defined in CapSense customizer.

Return Value: None

Side Effects: The number of bleed resistors is restricted by 3. The function does not check for an out of range number.

High Level APIs

These API functions are used to work with raw data for sensor widgets. The raw data is retrieved from scanned sensors and converted to on/off for buttons, position for sliders, or X and Y coordinates for touch pads.

Function	Description
CapSense_InitializeSensorBaseline	Loads the CapSense_SensorBaseline[sensor] array element with an initial value by scanning the selected sensor.
CapSense_InitializeAllBaselines	Loads the CapSense_SensorBaseline[] array with initial values by scanning all sensors.
CapSense_UpdateSensorBaseline	The historical count value, calculated independently for each sensor, is called the sensor's baseline. This baseline updated uses a low pass filter with k = 256.
CapSense_UpdateEnabledBaselines	Checks the CapSense_SensorEnableMask[]array and calls the CapSense_UpdateSensorBaseline function to update the baselines for enabled sensors.
CapSense_EnableWidget	Enables all sensor elements in a widget for the scanning process.
CapSense_DisableWidget	Disables all sensor elements in a widget from the scanning process.

Function	Description
CapSense_CheckIsWidgetActive	Compares the selected of widget to the CapSense_Signal[] array to determine if it has a finger press.
CapSense_CheckIsAnyWidgetActiv e	Uses the CapSense_CheckIsWidgetActive() function to find if any widget of the CapSense CSD component is in active state.
CapSense_GetCentroidPos	Checks the CapSense_SensorSignal[] array for a finger press in a linear slider and returns the position.
CapSense_GetRadialCentroidPos	Checks the CapSense_SensorSignal[] array for a finger press in a radial slider widget and returns the position.
CapSense_GetTouchCentroidPos	If a finger is present, this function calculates the X and Y position of the finger by calculating the centroids within the touch pad.

void CapSense_InitializeSensorBaseline (uint8 sensor)

Description: Loads the CapSense_SensorBaseline[sensor] array element with an initial value by scanning the selected sensor (one channel design) or pair of sensors (two channels design). The raw count value is copied into the baseline array for each sensor. The raw data filters are initialized if enabled.

Parameters: (uint8) sensor: Sensor number

Return Value: None

Side Effects: None

void CapSense_InitializeAllBaselines(void)

Description:	Uses the CapSense_InitializeSensorBaseline function to load the CapSense_SensorBaseline[ ] array with initial values by scanning all sensors. The raw count values are copied into the baseline array for all sensors. The raw data filters are initialized if enabled.
Parameters:	None
Return Value:	None
Side Effects:	None

void CapSense_UpdateSensorBaseline (uint8 sensor)

Description:	The sensor's baseline is a historical count value, calculated independently for each sensor. Updates the CapSense_SensorBaseline[sensor] array element using a low pass filter with k = 256. The function calculates the difference count by subtracting the previous baseline from the current raw count value and stores it in CapSense_SensorSignal[sensor]. If auto reset option is enabled the baseline updates independent of the noise threshold. If the auto reset option is disabled the baseline stops updating if the signal is greater than the noise threshold and resets the baseline when signal is less than the minus noise threshold. Raw data filters are applied to the values if enabled before baseline calculation.
Parameters:	(uint8) sensor: Sensor number
Return Value:	None
Side Effects:	None

void CapSense_UpdateEnabledBaselines(void)

Description:	Checks the CapSense_SensorEnableMask[] array and calls the CapSense_UpdateSensorBaseline() function to update the baselines for all enabled sensors.
Parameters:	None
Return Value:	None
Side Effects:	None

void CapSense_EnableWidget (uint8 widget)

Description: Enables the selected widget sensors to be part of the scanning process.

Parameters: (uint8) widget: Widget number. For every widget there are defines in this format:

```
#define CapSense_"widget_name"__"widget type"            5
```

Example:

```
#define CapSense_MY_VOLUME1__LS                          5
#define CapSense_MY_UP__BNT                              6
```

All widget names are upper case.

Return Value: None

Side Effects: None

void CapSense_DisableWidget (uint8 widget)

Description: Disables the selected widget sensors from the scanning process.

Parameters: (uint8) widget: Widget number. For every widget there are defines in this format:

```
#define CapSense_"widget_name"__"widget type"            5
```

Example:

```
#define CapSense_MY_VOLUME1__RS                          5
#define CapSense_MY_UP__MB                               6
```

All widget names are upper case.

Return Value: None

Side Effects: None

uint8 CapSense_CheckIsWidgetActive(uint8 widget)

Description: Compares the selected sensor CapSense_Signal[ ] array value to its finger threshold. Hysteresis and Debounce are taken into account. If the sensor is active, the threshold is lowered by the hysteresis amount. If it is inactive, the threshold is increased by the hysteresis amount. If the active threshold is met, the Debounce counter increments by one until reaching the sensor active transition at which point this API sets the Widget as active. This function also updates the sensor's bit in the CapSense_SensorOnMask[ ] array.

The touch pad and matrix buttons widgets need to have active sensor within col and row to return widget active status.

Parameters: (uint8) widget: Widget number. For every widget there are defines in this format:

```
#define CapSense_"widget_name"__"widget type"            5
```

Example:

```
#define CapSense_MY_VOLUME1__LS                          5
```

All widget names are upper case.

Return Value: (uint8) Widget sensor state. 1 if one or more sensors within widget are active, 0 if all sensors within widget are inactive.

Side Effects: This function also updates values in CapSense_SensorOnMask[ ] for all sensors belonging to the widget. The debounce counter is also modified on every call when there is a transition to the active state.

uint8 CapSense_CheckIsAnyWidgetActive(void)

Description: Compares all sensors of the CapSense_Signal[ ] array to their finger threshold. Calls Capsense_CheckIsWidgetActive() for each widget so the CapSense_SensorOnMask[ ] array is up to date after calling this function.

Parameters: None

Return Value: (uint8) 1 if any widget is active, 0 no widgets are active.

Side Effects: Has the same side effects as the CapSense_CheckIsWidgetActive() function but for all sensors.

uint16 CapSense_GetCentroidPos(uint8 widget)

Description: Checks the CapSense_Signal[ ] array for a finger press within a linear slider. The finger position is calculated to the API resolution specified in the CapSense customizer. A position filter is applied to the result if enabled. This function is available only if a linear slider widget is defined by the CapSense customizer.

Parameters: (uint8) widget: Widget number. For every linear slider widget there are defines in this format:

```
#define CapSense_"widget_name"__LS              5
```

Example:

```
#define CapSense_MY_VOLUME1__LS                 5
```

All widget names are upper case.

Return Value: (uint16) Position value of the linear slider

Side Effects: If any sensors within the slider widget are active, the function returns values from zero to the API resolution value set in the CapSense customizer. If no sensors are active, the function returns 0xFFFF. If an error occurs during execution of the centroid/diplexing algorithm, the function returns 0xFFFF.

There are no checks of widget argument provided to this function. An incorrect widget value provided will cause unexpected position calculations.

Note If noise counts on the slider segments are greater than the noise threshold, this subroutine may generate a false finger press result. The noise threshold should be set carefully (high enough above the noise level) so that noise will not generate a false finger press.

uint16 CapSense_GetRadialCentroidPos(uint8 widget)

Description: Checks the CapSense_Signal[ ] array for a finger press within a radial slider. The finger position is calculated to the API resolution specified in the CapSense customizer. A position filter is applied to the result if enabled. This function is available only if a radial slider widget is defined by the CapSense customizer.

Parameters: (uint8) widget: Widget number. For every radial slider widget there are defines in this format:

```
#define CapSense_"widget_name"__RS              5
```

Example:

```
#define CapSense_MY_VOLUME2__RS                 5
```

All widget names are upper case.

Return Value: (uint16) Position value of the radial slider.

Side Effects: If any sensors within the slider widget are active, the function returns values from zero to the API resolution value set in the CapSense customizer. If no sensors are active, the function returns 0xFFFF.

There are no checks of widget type argument provided to this function. An incorrect widget value provided will cause unexpected position calculations.

Note If noise counts on the slider segments are greater than the noise threshold, this subroutine may generate a false finger press result. The noise threshold should be set carefully (high enough above the noise level) so that noise will not generate a false finger press.

uint8 CapSense_GetTouchCentroidPos (uint8 widget)

Description: If a finger is present on touch pad, this function calculates the X and Y position of the finger by calculating the centroids within the touch pad sensors. The X and Y positions are calculated to the API resolutions set in the CapSense customizer. Returns a '1' if a finger is on the touchpad. A position filter is applied to the result if enabled. This function is available only if a touchpad is defined by the CapSense customizer.

Parameters: (uint8) widget: Widget number. For every touchpad widget there are defines in this format:

```
#define CapSense_"widget_name"__TP             5
```

Example:

```
#define CapSense_MY_TOUCH1__TP                 5
```

All widget names are upper case.

Return Value: (uint8) 1 if finger is on the touchpad, 0 if not.

Side Effects: The result of calculation of X and Y position are stored in global array. The array name and position are:

```
CapSense_position[widget]      - position of X
CapSense_position[widget + 1]  - position of Y
```

There are no checks of widget value argument provided to this function. An incorrect widget value will cause unexpected position calculations.

Tuner Helper APIs

These API functions are used to work with Tuner GUI.

Function	Description
CapSense_TunerStart	Initializes CapSense CSD and EZI2C components, initializes baselines and starts the sensor scanning loop.
CapSense_TunerComm	Execute communication between the Tuner GUI.

void CapSense_TunerStart (void)

Description: Initialize the CapSense CSD component and EZI2C component. Also initialize baselines and starts the sensor scanning loop with the currently enabled sensors.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_TunerComm (void)

Description: Execute communication functions with Tuner GUI.

- Manual mode: Transfer sensor scanning and widget processing results to the Tuner GUI from the CapSense CSD component. Reads new parameters from Tuner GUI and apply them to the CapSense CSD component.
- Auto(SmartSense): Execute communication functions with Tuner GUI. Transfer sensor scanning and widget processing results to Tuner GUI. The auto tuning parameters also transfer to Tuner GUI. Tuner GUI parameters are not transferred back to the CapSense CSD component.

This function is blocking and waits while the Tuner GUI modifies CapSense CSD component buffers to allow new data.

Parameters: None

Return Value: None

Side Effects: None

Pins APIs

These API functions are used to change drive mode of pins used by CapSense component. These APIs are most often used to place CapSense CSD component pins into the Strong drive mode to minimize leakage while the device is in a low power mode.

Function	Description
CapSense_SetAllSensorsDriveMode	Sets the drive mode for the all pins used by capacitive sensors within the CapSense component.
CapSense_SetAllCmodsDriveMode	Sets the drive mode for the all pins used by Cmod capacitors within the

	CapSense component.
CapSense_SetAllRbsDriveMode	Sets the drive mode for the all pins used by bleed resistors (Rb) within the CapSense component. Only available when Current Source is set to external resistor.

void CapSense_SetAllSensorsDriveMode(uint8 mode)

Description:	Sets the drive mode for the all pins used by capacitive sensors within the CapSense component.
Parameters:	(uint8) mode: Desired drive mode. See the Pins component datasheet for information on drive modes.
Return Value:	None
Side Effects:	None

void CapSense_SetAllCmodsDriveMode(uint8 mode)

Description:	Sets the drive mode for the all pins used by Cmod capacitors within the CapSense component.
Parameters:	(uint8) mode: Desired drive mode. See the Pins component datasheet for information on drive modes.
Return Value:	None
Side Effects:	None

void CapSense_SetAllRbsDriveMode(uint8 mode)

Description:	Sets the drive mode for the all pins used by bleed resistors (Rb) within the CapSense component. Only available when Current Source is set to external resistor.
Parameters:	(uint8) mode: Desired drive mode. See the Pins component datasheet for information on drive modes.
Return Value:	None
Side Effects:	None

Data Structures

The API functions use several global arrays for processing sensor and widget data. You should not alter these arrays manually. These values can be viewed for debugging and tuning purposes. For example, you can use a charting tool to display the contents of the arrays. The global arrays are:

- CapSense_SensorRaw [ ]
- CapSense_SensorEnableMask [ ]
- CapSense_portTable[ ] and CapSense_maskTable[ ]
- CapSense_SensorBaseline [ ]
- CapSense_SensorBaselineLow[ ]
- CapSense_SensorSignal [ ]
- CapSense_SensorOnMask[ ]

CapSense_SensorRaw [ ]

This array contains the raw data for each sensor. The array size is equal to the total number of sensors (CapSense_TOTAL_SENSOR_COUNT). The CapSense_SensorRaw[ ] data is updated by these functions:

- CapSense_ScanSensor()
- CapSense_ScanEnabledWidgets()
- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorEnableMask [ ]

This is a byte array that holds the sensor scanning state CapSense_SensorEnableMask[0**]** contains the masked bits for sensors 0 through 7 (sensor 0 is bit 0, sensor 1 is bit 1). CapSense_SensorEnableMask[1] contains the masked bits for sensors 8 through 15 (if needed), and so on. This byte array holds as many elements as are necessary to contain the total number of sensors. The value of a bit specifies if a sensor is scanned by the CapSense_ScanEnabledWidgets() function call: 1 – sensor is scanned , 0 – sensor is not scanned. The CapSense_SensorEnableMask[ ] data is changed by functions:

- CapSense_EnabledWidget()
- CapSense_DisableWidget()

The CapSense_SensorEnableMask[ ] data is used by function:

- CapSense_ScanEnabledWidgets()

CapSense_portTable[ ] and CapSense_maskTable[ ]

These arrays contain port and pin masks for every sensor to specify what pin the sensor is connected to.

- Port – Defines the port number that pin belongs to.
- Mask – Defines pin number within the port.

CapSense_SensorBaselineLow[ ]

This array holds the fractional byte of baseline data of each sensor used in the low pass filter for baseline update. The arrays size is equal to the total number of sensors. The CapSense_SensorBaselineLow[ ] array is updated by these functions:

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorBaseline[ ]

This array holds the baseline data of each sensor. The arrays size is equal to the total number of sensors. The CapSense_SensorBaseline[array is updated by these functions:

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorSignal[ ]

This array holds the sensor signal count computed by subtracting the previous baseline from the current raw count of each sensor. The array size is equal to the total number of sensors. The **Widget Resolution** parameter defines the resolution of this array as **1 byte** or **2 bytes**. The CapSense_SensorSignal[ ] array is updated by these functions:

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorOnMask[ ]

This is a byte array that holds the sensors on/off state.

CapSense_SensorOnMask[0] contains the masked bits for sensors 0 through 7 (sensor 0 is bit 0, sensor 1 is bit 1). CapSense_SensorOnMask[1] contains the masked bits for sensors 8 through 15 (if needed), and so on. This byte array holds as many elements as are necessary to contain the total number of sensors. The value of a bit is 1 if the sensor is on (active) and 0 if the sensor is off (inactive). The CapSense_SensorOnMask[ ] data is updated by functions:

- CapSense_CheckIsWidgetActive()
- CapSense_CheckIsAnyWidgetActive()

Constants

The following constants are defined. Some of the constants are defined conditionally and will only be present if necessary for the current configuration.

- CapSense_TOTAL_SENSOR_COUNT – Defines the total number of sensors within the CapSense CSD component.

For two channels designs the number of sensor belongs to a channel is defined as:

- CapSense_TOTAL_SENSOR_COUNT__CH0 – Defines total number of sensors belonging to channel 0.
- CapSense_TOTAL_SENSOR_COUNT__CH1 – Defines total number of sensors belonging to channel 1.
- CapSense_CSD_TOTAL_SCANSLOT_COUNT – Defines the maximum sensor count in either channel 0 or channel 1.

Sensor Constants

A constant is provided for each sensor. These constants can be used as parameters in the following functions:

- CapSense_EnableSensor
- CapSense_DisableSensor

The constant names consist of:

Instance name + "_SENSOR" + Widget Name + element + "#element number" + "__" + Widget Type

For example:

```
#define CapSense_SENSOR_TP1_ROW0__TP     0
#define CapSense_SENSOR_TP1_ROW1__TP     1
#define CapSense_SENSOR_TP1_COL0__TP     2
#define CapSense_SENSOR_TP1_COL0__TP     3
#define CapSense_SENSOR_LS0_E0__LS       5
#define CapSense_SENSOR_LS0_E1__LS       6
#define CapSense_SENSOR_PROX1__PROX      7
```

- Widget Name – The user-defined name of the widget (must be a valid C style identifier). The widget name must be unique within the CapSense CSD component. All Widget Names are upper case.
- Element Number – The element number only exists for widgets that have multiple elements, such as radial sliders. For touch pads and matrix buttons the element number consists of the word 'Col' or 'Row' and its number (for example: Col0, Col1, Row0, Row1). For linear and radial sliders, the element number consists of the character 'e' and its number (for example: e0, e1, e2, e3).
- Widget Type – There are several widget types:

Alias	Description
BTN	Buttons
LS	Linear Sliders
RS	Radial Sliders
TP	Touch Pads
MB	Matrix Buttons
PROX	Proximity Sensors
GEN	Generic Sensors
GRD	Guard Sensor

Widget Constants

A constant is provided for each widget. These constants can be used as parameters in the following functions:

- CapSense_CheckIsWidgetActive()
- CapSense_EnableWidget() and CapSense_DisableWidget()
- CapSense_GetCentroidPos()
- CapSense_GetRadialCentroidPos()
- CapSense_TouchPos()

The constants consist of:

Instance name + Widget Name + Widget Type

For example:

```
#define CapSense_UP__BTN         0
#define CapSense_DOWN__BTN       1
#define CapSense_VOLUME__SL      2
#define CapSense_TOUCHPAD__TP    3
```

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

Pin Assignments

The CapSense customizer generates a pin alias name for each of the CapSense sensors and support signals. These aliases are used to assign sensors and signals to physical pins on the device. Assign CapSense CSD component sensors and signals to pins in the Pin Editor tab of the Design Wide Resources file view.

Sides

The analog routing matrix within the PSoC device is divided into two halves – left and right. Even port number pins are on the left side of the device and odd port number pins are on the right side.

For serial sensing applications, sensor pins can be assigned to either side of the device. If the application uses a small number of sensors, assigning all sensor signals to one side of the device makes routing of analog resources more efficient and frees analog resource for other components.

In parallel sensing applications the CapSense component is capable of performing two simultaneous scans on two independent sets of hardware. Each of the two parallel circuits has a separate Cmod and Rb (as applicable), and its own set of sensor pins. One set will occupy the right side and the other will occupy the left side of the device. The signal name alias indicates which side the signal is associated with.

Sensor Pins – CapSense_cPort – Pin Assignment

Aliases are provided to associate sensor names with widgets types and widgets names in the CapSense customizer.

The aliases for sensors are:

Widget Name + Element Number + "__" + Widget Type

Note In two-channel designs, widget elements that belong to a channel can only be connected to the same side of the chip as that channels Cmod. **The Pin Editor does not verify correct pin assignment with a design rule check**. Pin placement errors will be flagged during the build process.

Note The Opamp outputs P0[0], P0[1], P3[6] and P3[7] have greater parasitic capacitance than other pins. This causes less finger response from P0[0], P0[1], P3[6] and P3[7] in CapSense applications and they should be avoided if possible. If they must be used, they should be used for individual buttons where the capacitive difference will not translate into position errors for sliders and touchpads.

CapSense_cCmod_Port – Pin Assignment

One side of the external modulator capacitor (Cmod) should be connected to a physical pin and the other to GND. Two-channel designs require two Cmod capacitors, one for the left side and one for the right side of the device. The Cmod can be connected to **any pin**, but for most efficient analog routing the following pins allow for a direct connection:

- Left side: P2[0], P2[4], P6[0], P6[4], P15[4]
- Right side: P1[0], P1[4], P5[0], P5[4]

The aliases for the Cmod capacitors are:

Alias	Description
CmodCH0	Cmod for channel 0
CmodCH1	Cmod for channel 1. Only available in two-channel designs.

The recommended value for the modulator capacitor is 4.7 – 47 nF. The optimal capacitance can be selected by experiment to get the maximum SNR for the application. A value of 5.6 – 10 nF gives good results in most cases.

A ceramic capacitor should be used. The capacitors temperature coefficient is not important.

When **Current Source** is set to External Resistor, the external Rb feedback resistor value should be selected before determining the optimal Cmod value.

CapSense_cRb_Ports – Pin Assignment

An external bleed resistor (Rb) is required when **Current Source** is set to External Resistor. The external bleed resistor (Rb) should be connected to a physical pin and to the ungrounded connection of the modulator capacitor (Cmod).

Up to three bleed resistors are supported per channel. The three pins can be allocated for bleed resistors: cRb0, cRb1 and cRb2.

The aliases for external bleed resistors are:

Alias	Description
Rb0CH0, Rb1CH0, Rb2CH0	External resisters for channel 0
Rb0CH1, Rb1CH1, Rb2CH1	External resisters for channel 1. Only available in two-channel designs.

The resistor values depend on the total sensor capacitance. The resistor value should be selected as follows:

- Monitor the raw counts for different sensor touches.
- Select a resistance value that provides maximum readings about 30% less than the full scale readings at the selected scanning resolution. The raw count value is increased when the resistor values increase.

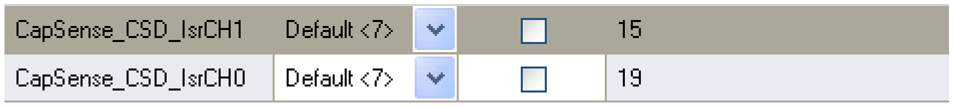
Typical bleed resistor values are 500 Ω - 10 kΩ depending on sensor capacitance.

Interrupt Service Routines

The CapSense component uses an interrupt that triggers after the end of each sensor scan. Stub routines are provided where you can add your own code if required. The stub routines are generated in the *CapSense_INT.c* file the first time the project is built. The number of interrupts depends on CapSense mode selection depending on the Number of Channels, one per channel. Your code must be added between the provided comment tags in order to be preserved between builds.

Two Channel Mode ISR Priority Set

The ISRs routines of CapSense CSD component are not reentrant. This causes a restriction on the ISR priority set for two channels designs. To prevent the channel ISR routines from becoming reentrant the ISR priority of the two channels must be the same.



CapSense_CSD_IsrCH1	Default <7>	☐	15
CapSense_CSD_IsrCH0	Default <7>	☐	19

Functional Description

Definitions

Sensor

One CapSense element connected to PSoC via one pin. A sensor is a conductive element on a substrate. Examples of sensors include: Copper on FR4, Copper on Flex, Silver ink on PET, ITO on glass.

Scan Time

A scan time is a period of time that the CapSense module is scanning one or more capacitive sensors. Multiple sensors can be combined in a given scan sensor to enable modes such as proximity sensing.

CapSense Widget

A CapSense widget is built from one or more scan sensors to provide higher level functionality. Some examples of CapSense Widgets include buttons, sliders, radial sliders, touch pads, matrix buttons, and proximity sensors.

FingerThreshold

This value is used to determine if a finger is present on the sensor or not.

NoiseThreshold

Determines the level of noise in the capacitive scan. The baseline algorithm filters the noise in order to track voltage and temperature variations in the sensor baseline value.

Debounce

Adds a debounce counter to the sensor active transition. In order for the sensor to transition from inactive to active the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified in order to filter out high amplitude and frequency noise.

Hysteresis

Sets the hysteresis value used with the finger threshold. If hysteresis is desired, the sensor will not be considered "On" or "Active" until the count value exceeds the finger threshold plus the hysteresis value. The sensor will not be considered "Off" or "Inactive" until the measured count value drops below the finger threshold minus the hysteresis value.

API Resolution – Interpolation and Scaling

In slider sensors and touch pads it is often necessary to determine finger (or other capacitive object) position to more resolution than the native pitch of the individual sensors. The contact area of a finger on a sliding sensor or a touchpad is often larger than any single sensor.

In order to calculate the interpolated position using a centroid calculation, the array is first scanned to verify that a given sensor location is valid. The requirement is for some number of adjacent sensor signals to be above the noise threshold. When the strongest signal is found, that signal and adjacent contiguous signals larger than the noise threshold are used to compute a centroid. As few as two and as many as eight sensors are used to calculate the centroid.

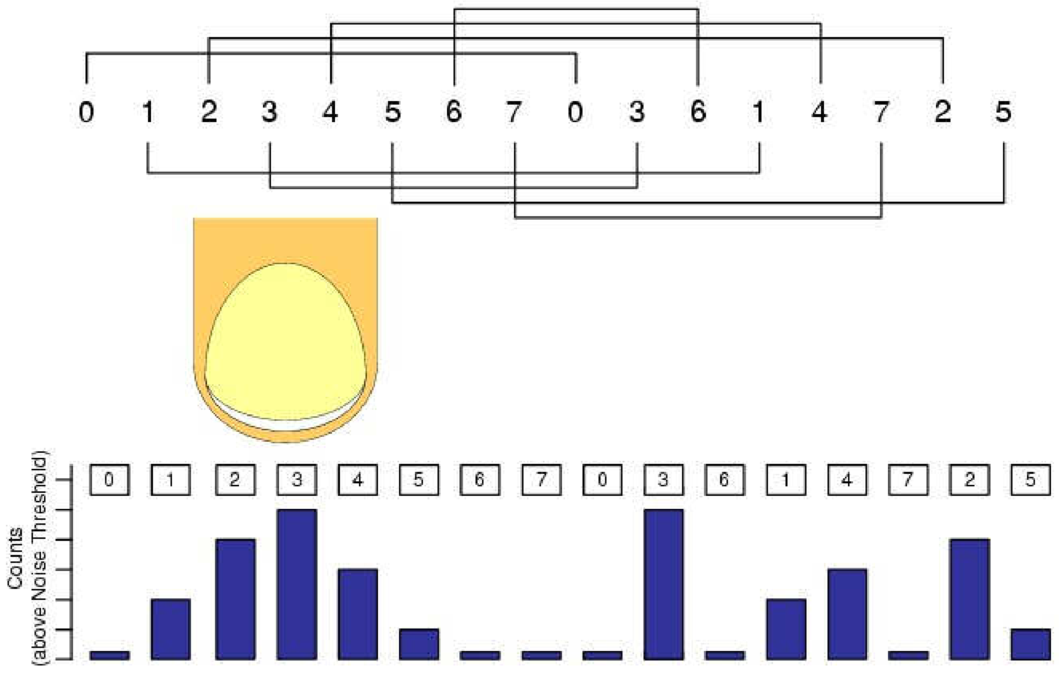
$$N_{Cent} = \frac{n_{i-1}(i-1) + n_i i + n_{i+1}(i+1)}{n_{i-1} + n_i + n_{i+1}}$$

The calculated value is typically fractional. In order to report the centroid to a specific resolution, for example a range of 0 to 100 for 12 sensors, the centroid value is multiplied by a scalar. It is more efficient to combine the interpolation and scaling operations into a single calculation and report this result directly in the desired scale. This is handled in the high-level APIs. Slider sensor count and resolution are set in the CapSense CSD customizer.

Diplexing

In a diplexed slider, each PSoC sensor connection in the slider is mapped to two physical locations in the array of slider sensors. The first (or numerically lower) half of the physical locations is mapped sequentially to the base assigned sensors, with the port pin assigned by the designer using the CapSense Customizer. The second (or upper) half of the physical sensor locations is automatically mapped by an algorithm in the Customizer and listed in an include file. The order is established so that adjacent sensor actuation in one half does not result in adjacent sensor actuation in the other half. Exercise care to determine this order and map it onto the printed circuit board.

Figure 1. Diplexing



You should balance sensor capacitance in the slider. Depending on sensor or PCB layouts, there may be longer routes for some of the sensor pairs. The diplex sensor index table is automatically generated by the CapSense customizer when you select diplexing and is included below for your reference.

Diplexing Sequence for Different Slider Segment Counts

Total Slider Segment Count	Segment Sequence
10	0,1,2,3,4,0,3,1,4,2

Total Slider Segment Count	**Segment Sequence**
12	0,1,2,3,4,5,0,3,1,4,2,5
14	0,1,2,3,4,5,6,0,3,6,1,4,2,5
16	0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5
18	0,1,2,3,4,5,6,7,8,0,3,6,1,4,7,2,5,8
20	0,1,2,3,4,5,6,7,8,9,0,3,6,9,1,4,7,2,5,8
22	0,1,2,3,4,5,6,7,8,9,10,0,3,6,9,1,4,7,10,2,5,8
24	0,1,2,3,4,5,6,7,8,9,10,11,0,3,6,9,1,4,7,10,2,5,8,11
26	0,1,2,3,4,5,6,7,8,9,10,11,12,0,3,6,9,12,1,4,7,10,2,5,8,11
28	0,1,2,3,4,5,6,7,8,9,10,11,12,13,0,3,6,9,12,1,4,7,10,13,2,5,8,11
30	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0,3,6,9,12,1,4,7,10,13,2,5,8,11,14
32	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,3,6,9,12,15,1,4,7,10,13,2,5,8,11,14
34	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14
36	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14,17
38	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,0,3,6,9,12,15,18,1,4,7,10,13,16,2,5,8,11,14,17
40	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11,14,17
42	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11,14,17,20
44	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,2,5,8,11,14,17,20
46	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20
48	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20,23

Total Slider Segment Count	Segment Sequence
50	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20,23
52	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,25,2,5,8,11,14,17,20,23
54	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,25,2,5,8,11,14,17,20,23,26
56	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,0,3,6,9,12,15,18,21,24,27,1,4,7,10,13,16,19,22,25,2,5,8,11,14,17,20,23,26

Filters

Several filters are provided in the CapSense component: median, averaging, first order IIR and jitter. The filters can be used with both raw sensor data to reduce sensor noise and with position data of sliders and touchpad to reduce position noise.

Median Filter

The median filter looks at the three most recent samples and reports the median value. The median is calculated by sorting the three samples and taking the middle value. This filter is used to remove short noise spikes and generates a delay of one sample. This filter is generally not recommended because of the delay and RAM use. Enabling this filter consumes 4 bytes of RAM for each sensor(raw) and Widget(position). It is disabled by default.

Averaging Filter

The averaging filter looks at the three most recent samples of position and reports the simple average value. It is used to remove short noise spikes and generates a delay of one sample. This filter is generally not recommended because of the delay and RAM use. Enabling this filter consumes 4 bytes of RAM for each sensor(raw) and Widget(position). It is disabled by default.

First Order IIR Filter

The first order IIR filter is the recommended filter for both raw and sensor filters because it requires the smallest amount of SRAM and provides a fast response. The IIR filter scales the most recent sensor or position data and adds it to a scaled version of the previous filter output. Enabling this filter consumes and 2 bytes of RAM for each sensor(raw) and Widget(position). The IIR1/4 is enabled by default for both raw and position filters.

1st-Order IIR filters:

$$IIR1/2 = 1/2\,previous + 1/2\,current$$

$$IIR1/4 = 3/4\,previous + 1/4\,current$$

$$IIR1/8 = 7/8\,previous + 1/8\,current$$

$$IIR1/16 = 15/16\,previous + 1/16\,current$$

Jitter Filter

This filter eliminates noise in the raw sensor or position data that toggles between two values (jitter). If the most current sensor value is greater than the last sensor value then the previous filter value is incremented by 1, if it is less then it is decremented. It is most effective when applied to data that contains noise of four LSBs peak-to-peak or less and when a slow response is acceptable which is useful for some position sensors. Enabling this filter consumes 2 bytes of RAM for each sensor(raw) and Widget(position). It is disabled by default.

Water Influence on CapSense System

The water drop and finger influence on CapSense are similar. However, water drop influence on the whole surface of the sensing area differs from a finger influence.

There are several variants of water influence on the CapSense surface:

- Forming of thin stripes or streams of water on the device surface.
- Separate drops of water.
- Stream of water cover all or a large portion of the device surface, when the device is being washed or dipped.

Salts or minerals that the water contains make it conductive. Moreover, the greater their concentration is the more conductive the water is. Soapy water, sea water, and mineral water are liquids that influence the CapSense poorly. These liquids emulate a finger touch on the device surface, which can cause the faulty device performance.

Waterproofing and Detection

- This feature configures the CapSense CSD component to suppress water influence on the CapSense system. This feature sets the following parameters:
- Enables a Shield electrode to be used to compensate for the water drops influence on the sensor at the hardware level.
- Adds a Guard sensor. The guard sensor should surround all sensors such that the guard sensor placement ensures that it will be covered by water if any of the actual sensing

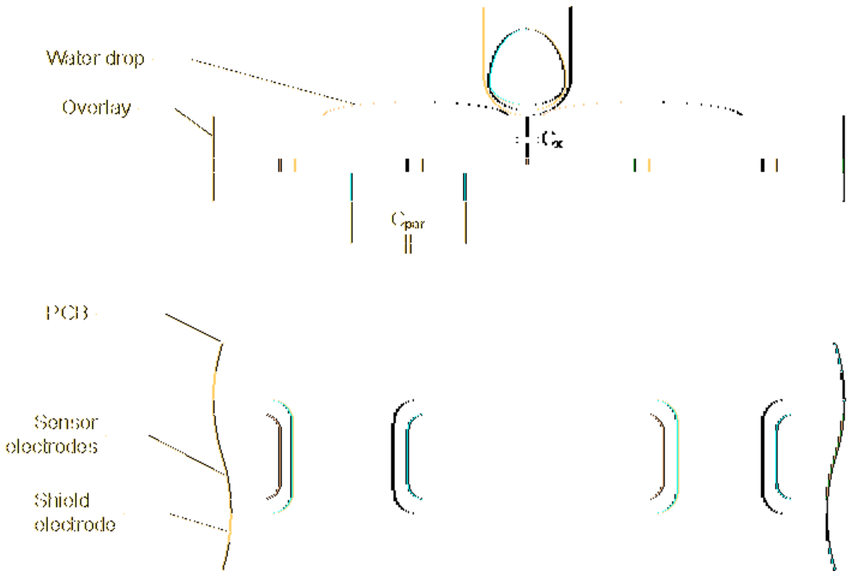
Widgets are covered. CapSense output of widget status should be blocked programmatically when the Guard sensor triggers.

Shield Electrode

Some applications require reliable operation in the presence of water film or droplets. White goods, automotive applications, various industrial applications, and others need capacitive sensors that do not provide false triggering because of water, ice, and humidity changes that cause condensation. In this case a separate shielding electrode can be used. This electrode is located behind or around the sensing electrodes. When water film is present on the device overlay surface, the coupling between the shield and sensing electrodes is increased. The shield electrode allows you to reduce the influence of parasitic capacitance, which gives you more dynamic range for processing sense capacitance changes.

In some applications it is useful to select the shield electrode signal and its placement relative to the sensing electrodes such that increasing the coupling between these electrodes due to moisture causes a negative touch change of the sensing electrode capacitance measurement. This simplifies the high level software API work by suppresses false touches due to moisture. The CapSense CSD component supports separate outputs for the shield electrode to simplify PCB routing.

Figure 2. Possible Shield Electrode PCB Layout



The previous figure illustrates one possible layout configuration for the button's shield electrode. The shield electrode is especially useful for transparent ITO touchpad devices, where it blocks the LCD drive electrode's noise and reduces stray capacitance at the same time.

In this example, the button is covered by a shielding electrode plane. As an alternative, the shielding electrode can be located on the opposite PCB layer, including the plane under the button. A hatch pattern is recommended in this case, with a fill ratio of about 30 to 40%. No additional ground plane is required in this case.

When water drops are located between the shield and sensing electrodes, the parasitic capacitance (Cpar) is increased and modulator current can be reduced.

The shield electrode can be connected to any pins. Set the drive mode to Strong Slow to reduce ground noise and radiated emissions. Also, a slew limiting resistor can be connected between the PSoC device and the shielding electrode.

Shield Electrode Usage and Restrictions

The CapSense CSD component provides the following modes for shield electrode usage.

Current Mode IDAC Source

This mode has some restrictions, because the sensors alternate between GND and Vref = 1.024 V. The shield electrode signal alternates between GND and Vddio (typically equal to power supply). The difference is significant and the shield signal could completely offset the signal from the sensors. The possible solutions are:

- Use a high Vref to eliminate the difference to a minimal value. The VDAC as reference could be used for this purpose.
- Use SIO pins as the shield to provide output equal to Vref. The CapSense CSD output Vref terminal could be used to route Vref to the SIO pins. This is the preferred method. The Sensor Connection to Shield should not be used in this mode because it provides ouput equal to Vddio. The Vref = 1.024 V setting has routing limitations and can not be routed to pins.

Current Mode IDAC Sink and External Resistor

These modes have no restriction on Shield and Inactive Sensor mode usage, because the sensor alternates between Vddio and Vref = 1.024 V. The shield electrode signal alternates between GND and Vddio (typically equal to power supply). The difference is not significant enough in this case to cause issues.

Guard Sensor Implementation

The Guard sensor is commonly used in water proof applications to detect water on the surface.

The Advanced Tab option is provided to add a guard sensor. This sensor has to have a special layout, typically located around the perimeter of the sensing area surface. When water is on the surface of the Guard sensor, the widget becomes active. The widget active detection firmware CapSense_1_IsWidgetActive() is available to define the state of the Guard sensor.

The detection of CapSense widgets should block programmatically in user code for a certain period of time when the Guard sensor triggers. If the guard sensor triggers, water is present and the other sensors can not be reliably sensed.

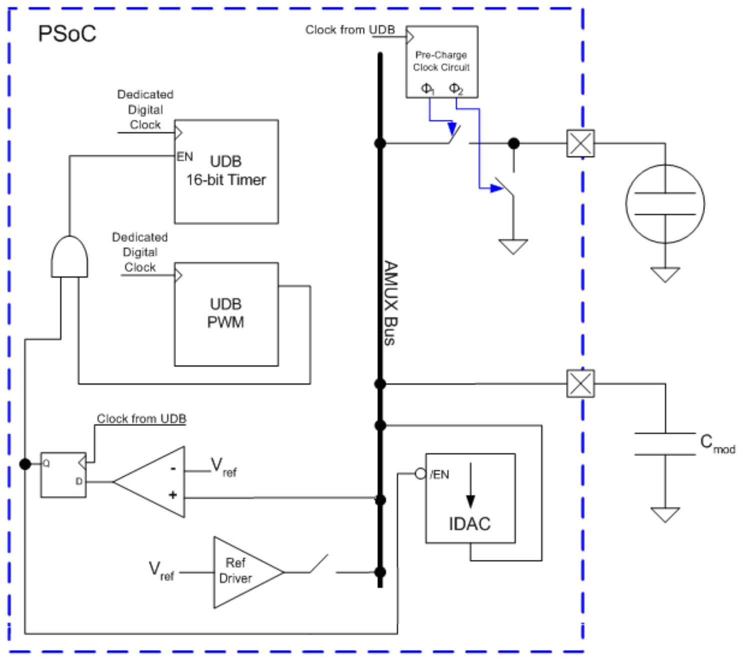
Taking into consideration the Guard sensor's size, its signal will differ from other sensors' signals. This means a larger amount of water may be present on its surface than on a standard sensor's surface. Therefore, the signal received with the presence of water drops will be much stronger than the signal caused by a finger touch. This allows setting the trigger threshold and filter so that a finger's touch on the Guard sensor has no effect. The Guard sensor scans without any special options. The shield electrode is not disabled while the Guard sensor scanning. The Guard sensor in two channel designs always scans last and by itself.

Block Diagram and Configuration

Capacitive sensing using a Sigma-Delta (CSD) modulator) provides capacitance sensing using a switched capacitor analog technique and a digital delta-sigma modulator to convert the sensed switched capacitor current into a digital code. It allows implementation of buttons, sliders, proximity detectors, touch pads, and touchscreens using arrays of conductive sensors. High level software routines allow for enhancement of slider resolution using diplexing, and compensation for physical and environmental sensor variation. There are three analog hardware variations possible on the basic CSD method, they are detailed below.

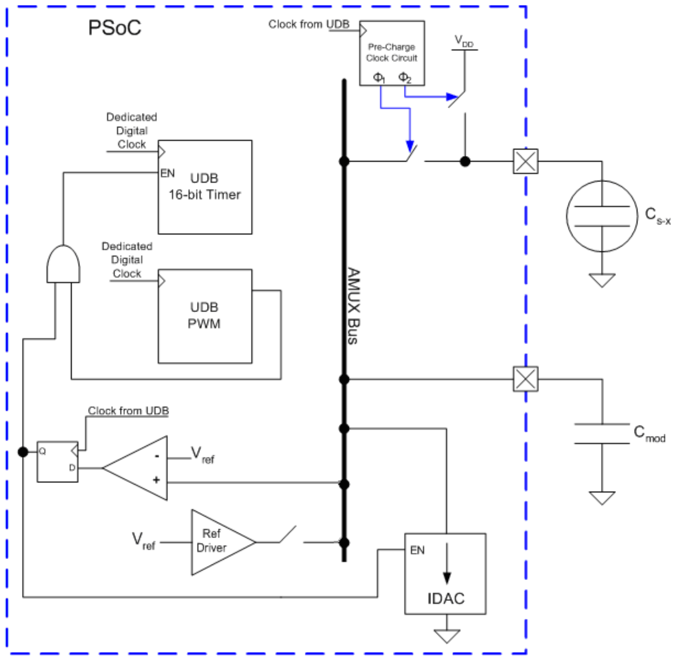
IDAC Sourcing

The sensor switch stage is configured to alternate between GND and the AMUX bus which connects to the modulation capacitor. In this configuration, the IDAC is configured to source current to the sensor.



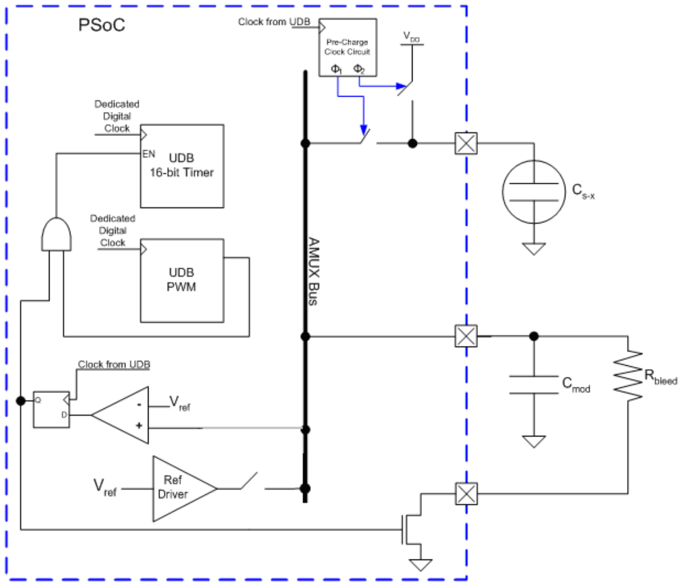
IDAC Sinking

The sensor switch stage is configured to alternate between Vdd and AMUX bus which connects to the modulation capacitor. In this configuration, the IDAC is configured to sink current from the sensor.



IDAC Disabled, Use External R_b

Using an external bleed resistor Rb functions the same as the IDAC Sinking configuration except the IDAC is replaced by a resistor to ground, R_b. The bleed resistor is physically connected between C_{mod} and a GPIO. The GPIO is configured in the "Open-Drain Drives Low" drive mode. This mode allows C_{mod} to be discharged through R_b.



DC and AC Electrical Characteristics

5.0V/3.3V DC and AC Electrical Characteristics

Power Supply Voltage

Parameter	Test Conditions and Comments	Min	Typ	Max	Units
Value	--	2.7	5.0	5.5	V

Noise

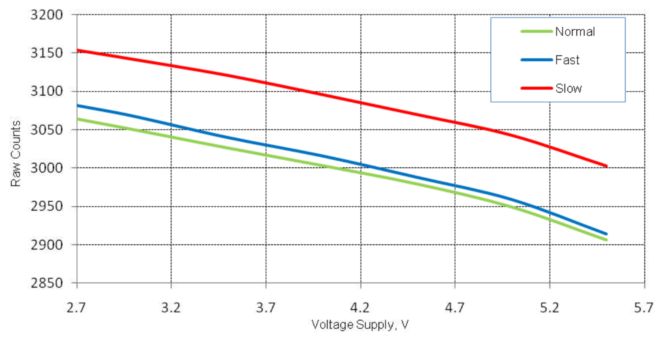
Parameter	Test Conditions and Comments	Min	Typ	Max	Units
Noise Counts, peak-peak (noise counts/(baseline counts)	Resolution = 16 (noise counts/(baseline counts)	--	0.2	--	%
	Resolution = 14	--	0.3	--	%
	Resolution = 10	--	0.4	--	%

Power Consumption

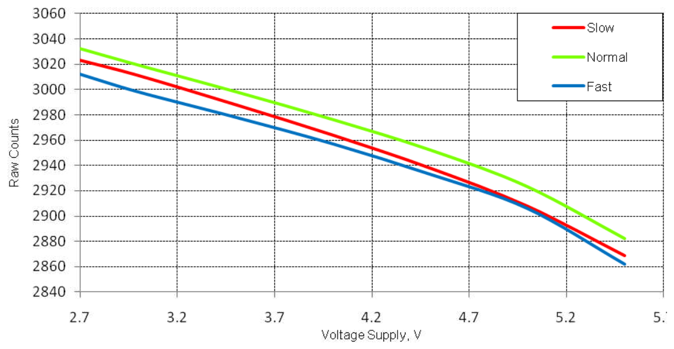
Parameter	Test Conditions and Comments	Min	Typ	Max	Units
Active Current	Vdd=3.3 V, CPU Clock= 24 MHz, CapSense Scan Clock=24 MHz, Average current during scan, 8 sensors	--	9.42	--	mA
Standby Current	Vdd=3.3 V, CPU Clock= 24 MHz, CapSense Scan Clock=24 MHz,, Scanning Speed = Ultra Fast, Resolution = 9 100 ms report rate, 8 sensors	--	92	--	uA
	Vdd=3.3 V, CPU Clock= 24 MHz, CapSense Scan Clock=24 MHz, Scanning Speed = Fast, Resolution = 12 100 ms report rate, 8 sensors	--	574	--	uA
Sleep/Wake Current	Vdd=3.3 V, CPU Clock= 24 MHz, CapSense Scan Clock=24 MHz, 1-s report rate, Scanning Speed = Fast, Resolution= 12, 1 sensor	--	8	--	uA

Figures

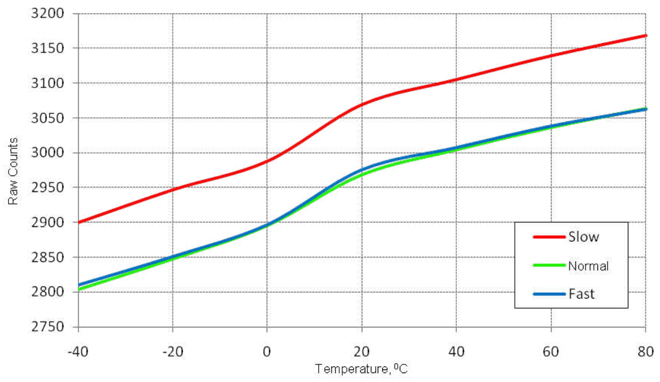
Rawcount vs Supply Voltage at Different Scan Speeds, PRS 16 full speed



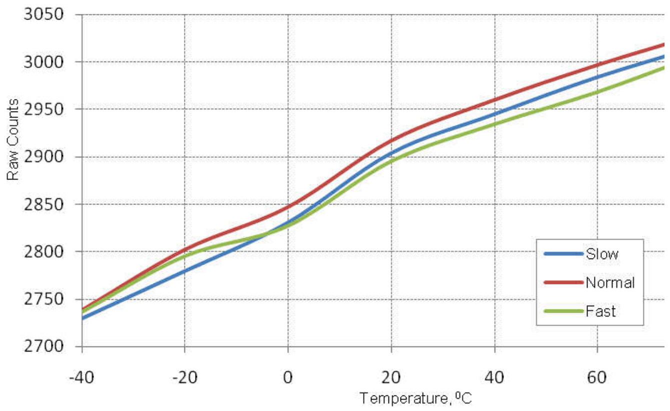
Rawcount vs Supply Voltage at Different Scan Speeds, PRS 8



Rawcount vs Temperature at Different Scan Speeds, PRS 16 full speed



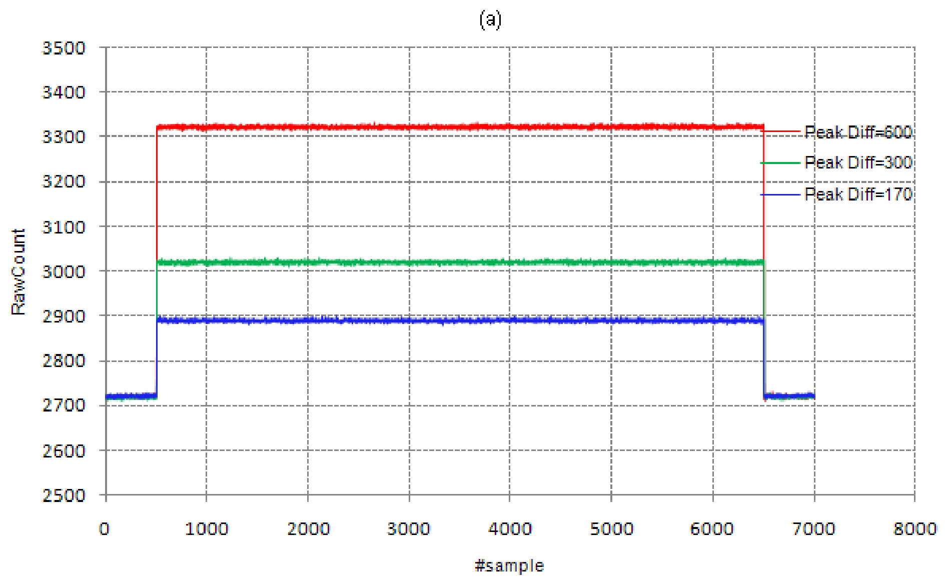
Raw Count vs Temperature at Different Speeds, PRS 8

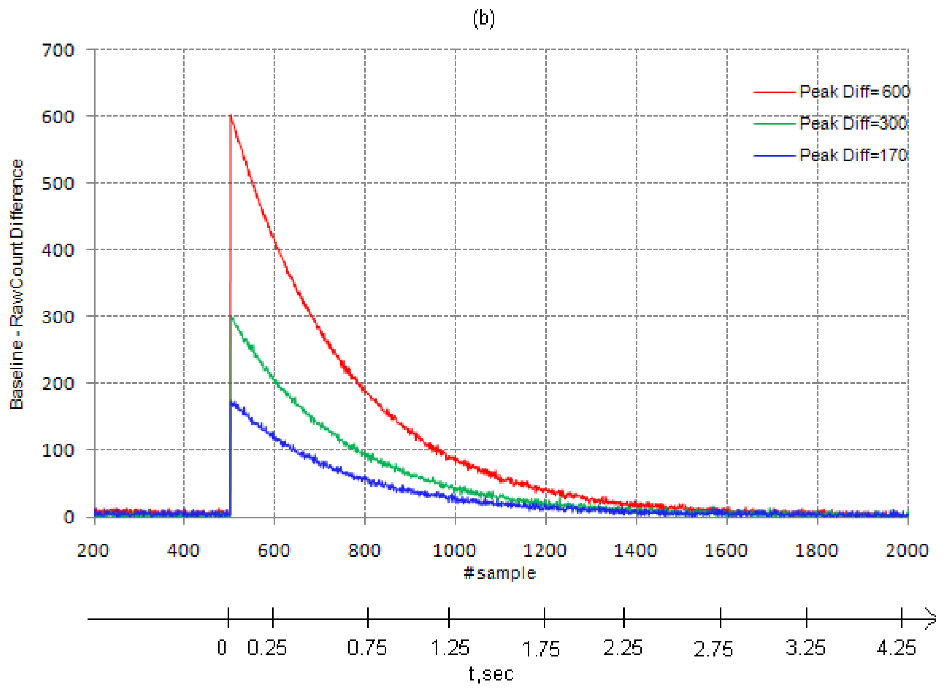


Variation of Baseline with time for different raw count step change values

(a) RawCounts Step Change for different step values

(b) Difference between Raw Count.





Component Changes

Version	Description of Changes	Reason for Changes / Impact
2.10	When Vdac is used as Reference the CapSense Buffer is removed. This action has an effect on designs where 1.024V is used as Reference.	The transient process on Cmod is predictable when Vdac is used as Reference.
	The Tuner GUI communication and scanning are separated.	EZI2C communication interrupt has no effect on scanning process.
	The user section is added to CapSenseCSD_PreScan() function.	Makes it possible to disable and enable interrupts at start of scanning if they affect scanning results.
	Add to CapSense_InitializeEnabledBaselines() to customer interface.	Allows initializing baselines only for enabled widgets.
	Add vector based images in customizer.	Improve image quality in customizer
	Redesign Advanced Tab	Improve controls layout
	Optimize packet size in AutoSense mode.	Reduce packet size and increase communication speed
	Redesign Import/Export mechanism between customizer and tuner	Make it more user friendly
	Update Tuner GUI: Added main menu, update color scheme.	Make it more user friendly
	Remove hysteresis related lines in sensor detailed view for centroid widgets	Clarify sensor detailed view for centroid widgets
	Improve SNR calculation speed in slow designs.	
	Add hysteresis parameter change visualization during AutoSense procedure in Tuner.	Allows the customer to review current hysteresis value
	Fix Tuner “Log duration” feature.	
	Implement mechanism that reads sensor ScanResolution, IdacValue and Prescaler only during the first read in AutoSense mode, because that data doesn’t change after the first transaction	Reduce data size that are transferred to Tuner GUI
	Added characterization data to datasheet	

Version	Description of Changes	Reason for Changes / Impact
	Total datasheet rewrite	

The CapSense CSD component version 2.10 is improved implementation of the CapSense CSD component version 2.00.