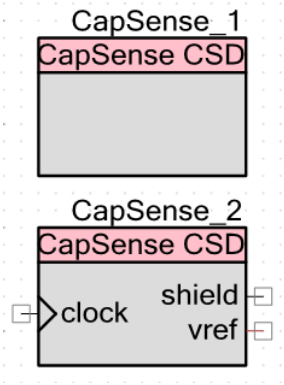
Capacitive Sensing (CapSense® CSD)

2.0



Features

- Supports different combinations of independent and slide capacitive sensors
- High immunity to AC mains noise, EMC noise, and power supply voltage changes
- Parallel (synchronized or asynchronous) and Serial Scanning Configuration
- 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

The 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, and proximity detection.

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

shield – Output *

Shield electrode signal is connected to this output. It is only available if shield electrode is enabled.

vref – Output *

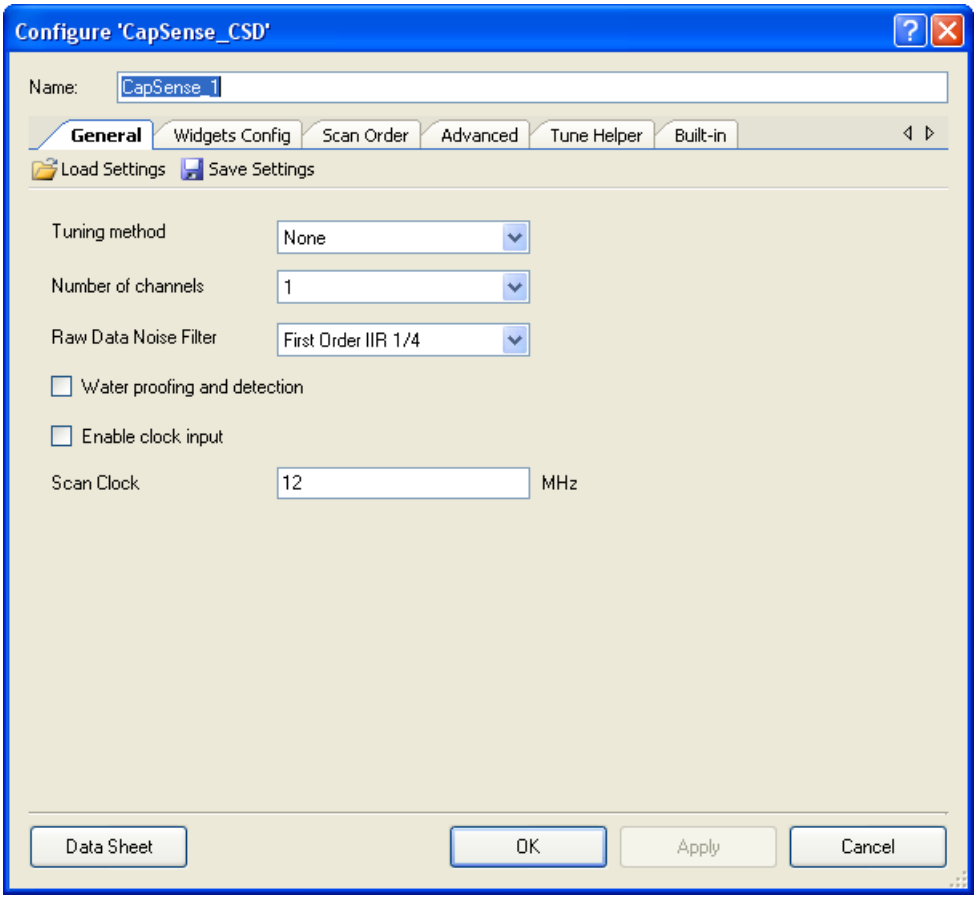
Analog reference voltage is connected to this output. It could be used to better adjust the shield signal amplitude. It is only available if the **Shield** option is enabled in **Source mode**.

Note The following section contains more details about using the Shield and Vref outputs.

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



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.

Tuning method

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

- Auto (SmartSense) – Allows tuning of the CapSense CSD component automatically. The average parameters, acceptable for all sensors, are selected by firmware. Additional RAM and CPU time are used in this mode.
- Manual – Allows you to tune 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 refer to the Capsense Tuner GUI User Guide section in this data sheet.
- None (Default) – Allows any tuning. All tuning parameters are stored in flash. This option could be used after all parameters of the CapSense component are tuned.

Note Selecting Manual or Auto tuning mode generates Tuner Helper APIs. Tuning requires an EZ I^2C communication component, which is specified on the **Tuner Helper** tab.

Number of channels

This parameter specifies the number of channels.

- 1 – 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.
  - 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 P12[X] and P15[X] have special connections to the AMUX bus Left and Right. 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.
- 2 – 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. Right and Left sensors are scanned two at time (one Right sensor and one Left sensor at a time) 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.
  - 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.
- ❑ Parallel scans run at the same scan rates.

Raw Data Noise Filter

This parameter selects the raw data filter. Only one filter may be selected and it applied to all sensors.

- None
- Median
- Averaging
- First Order IIR 1/2
- First Order IIR 1/4 – Default
- First Order IIR 1/8
- First Order IIR 1/16
- Jitter

Water proofing and detection

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

- Enables Shield output terminal
- Adds a Guard widget

Note The Guard widget can only be removed from **Advanced** tab.

Enable clock input

This parameter selects whether use internal clock or display input terminal for clock connection (Unchecked – Default).

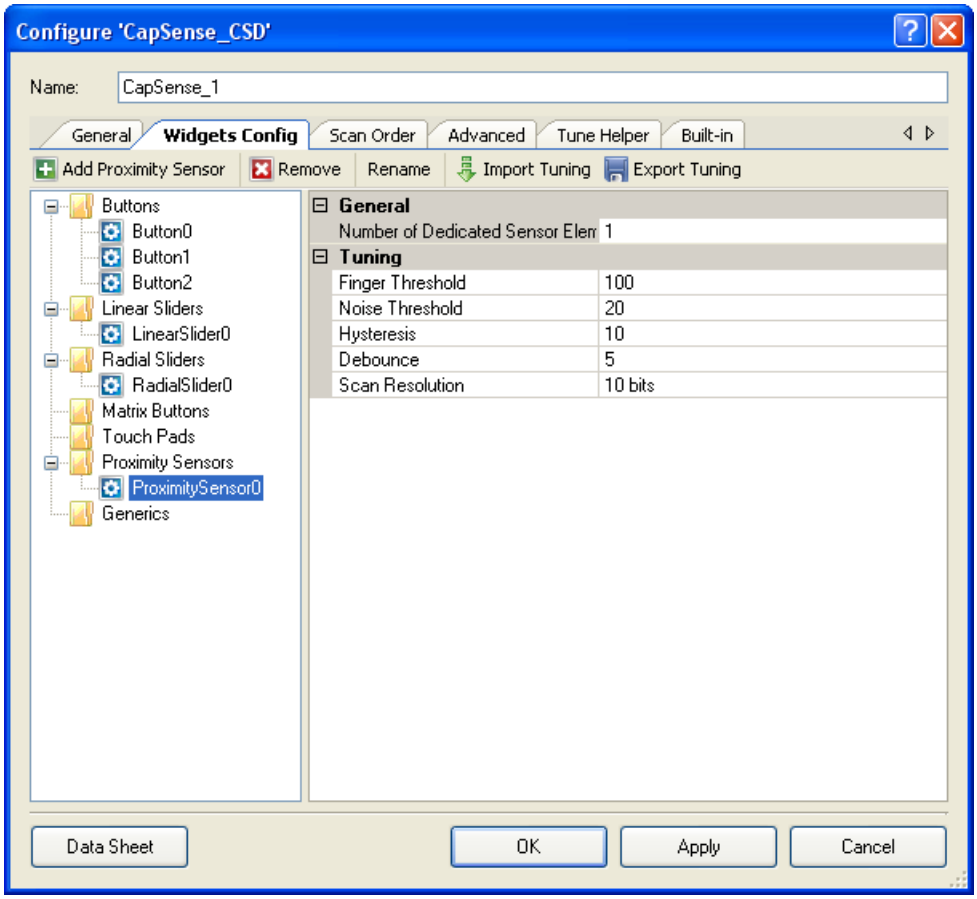
Note This option is disabled if the tuning method is Auto (SmartSense).

Scan Clock

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

Note The settings of **Analog Switch Drive Source** to "FF Timer (Default)" and/or **Digital Implementation** to "FF Timers," do not support the CapSense CSD clock equal to BUS_CLK. The **Enable clock input** should be selected and BUS_CLK provided in this case.

Widgets Config Tab



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

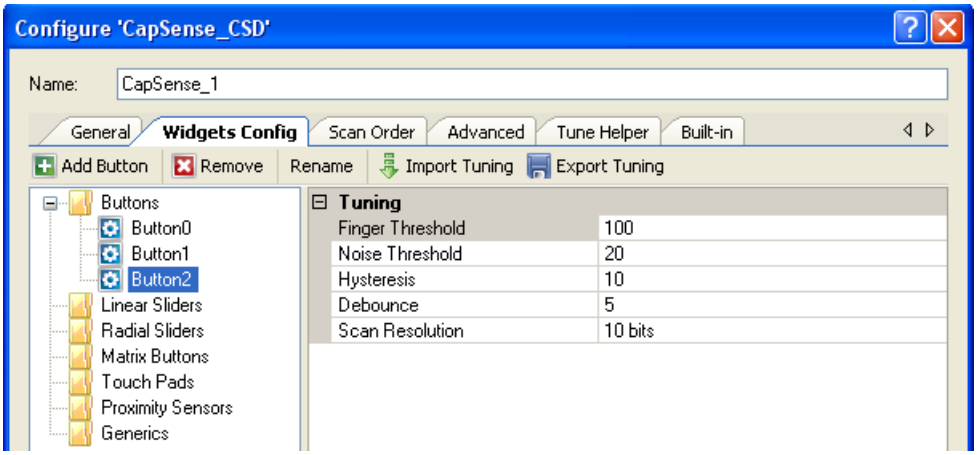
Toolbar

The toolbar contains the following commands:

- **Add widget** – Adds selected type of widget to the tree. The widget types are:
  - ❑ Button
  - ❑ Linear Slider
  - ❑ Radial Slider
  - ❑ Matrix Button
  - ❑ Touch Pad
  - ❑ Proximity Sensor
  - ❑ Generic Sensor
- **Remove widget** – Removes selected widget from the tree.

- **Rename** – Opens a dialog to change the widget name for a selected widget in the tree. You can also double-click a pin or press [F2] to open the dialog.
- **Import Tuning** – Imports tuning parameters from Tuner GUI to the component parameters.
- **Export Tuning** - Exports component parameters to the Tuner GUI.

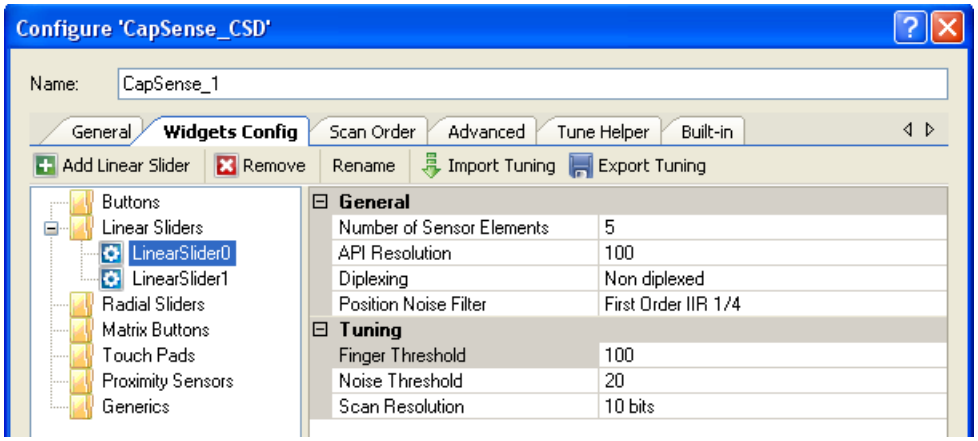
Buttons



Tuning:

- **Finger Threshold** – Defines sensor active threshold. Default value is 100. Valid range of values is [1…255].
- **Noise Threshold** – Defines sensor noise threshold. Count values above this threshold do not update the baseline. Default value is 20. Valid range of values 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. Default value is 10. Valid range of values is [1…255].
- **Debounce** – Adds debounce counter for 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. 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 SNR of touch detection. Default value is 10 bits.

Linear Sliders



General:

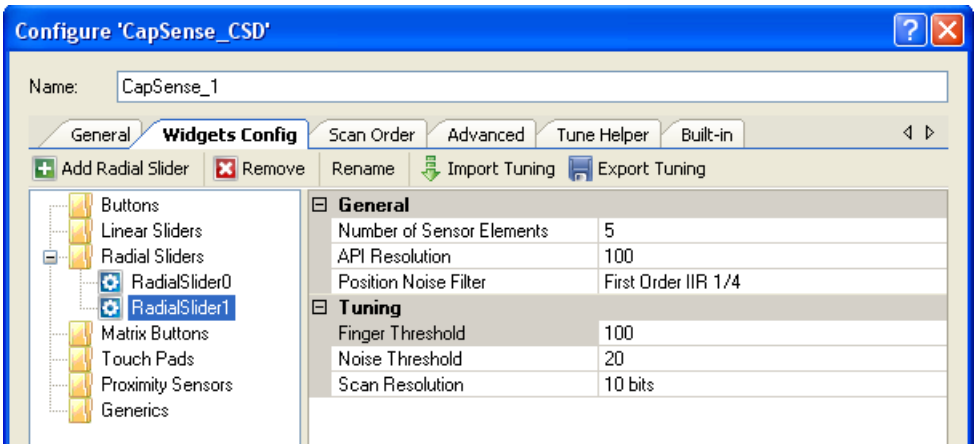
- **Numbers of Sensors Elements** – Defines the number of elements within the slider. Valid range of values is [2…32]. Default value is 5 elements.
- **API Resolution** – Defines the slider resolution. The position value will be changed within this range. Valid range of values is [1…255].
- **Diplexing –** Non diplexed (Default) or Diplexed.
- **Position Noise Filter** – Adds noise filter to position calculations. Only one filter may be applied for a selected widget.
  - 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 for slider elements. 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. 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. Default value is 10 bits.

Radial Slider



General:

- **Numbers of Sensors Elements** – Defines the number of elements within the slider. 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** – Adds noise filter to position calculations. Only one filter may be applied for a selected widget.
  - None
  - Median Filter
  - Averaging Filter
  - First Order IIR 1/2
  - First Order IIR 1/4 – Default
  - Jitter Filter

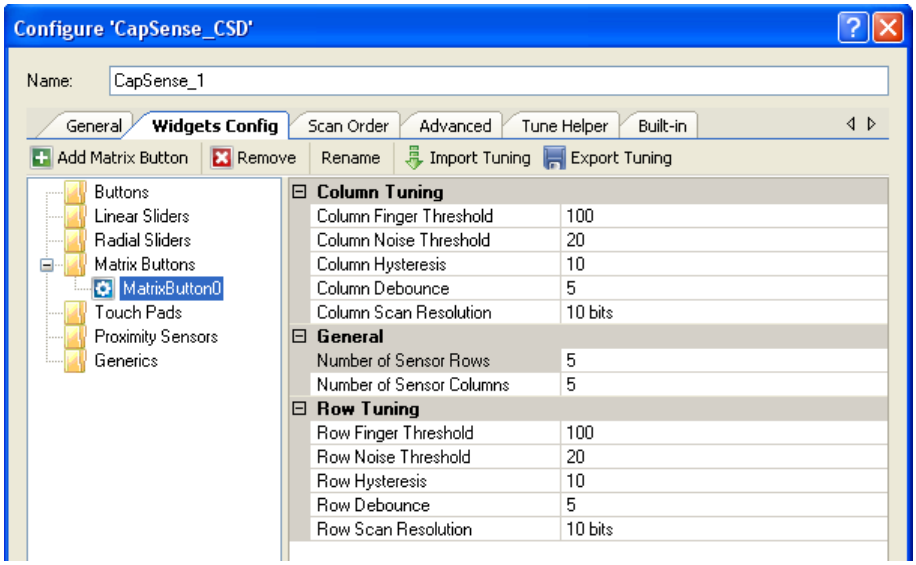
Tuning:

- **Finger Threshold** – Defines the sensor active threshold for slider elements. 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. 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. Default value is 10 bits.

Matrix Buttons



Tuning

- **Column/Row Finger Threshold** – Defines the sensor active threshold for matrix buttons column/row. Default value is 100. Valid range of values is [1…255].

- **Column/Row Noise Threshold** – Defines the sensor noise threshold for matrix buttons column/row. Count values above this threshold do not update the baseline. Default value is 20. Valid range of values is [1…255].

- **Column/Row Hysteresis** – Adds the differential hysteresis for sensor active state transition for matrix buttons column/row. 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. Default value is 10. Valid range of values is [1…255].

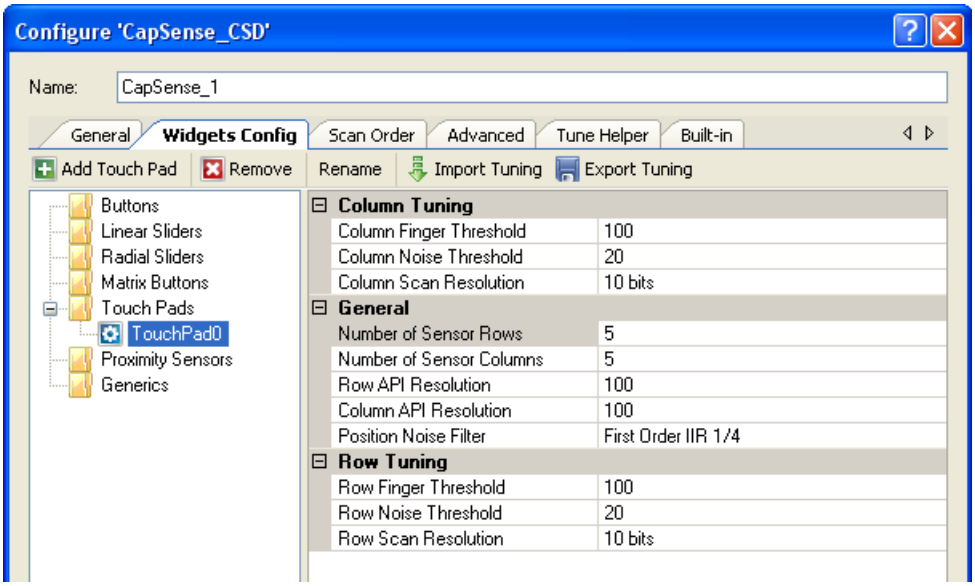
- **Column/Row Debounce** – Adds a debounce counter for the sensor active state transition for matrix buttons column/row. In order for the sensor to transition from inactive to active, the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified. Default value is 5. Valid range of values is [1…255].

- **Column/Row Scan Resolution** – Defines the scanning resolution of matrix buttons column/row. 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. The Column and Row scanning resolution should be equal to get the same sensitivity level. Default value is 10 bits.

General

- **Number of Sensor Rows/Columns** – Defines the number of elements within the matrix button column/row. Valid range of values is [2…32]. Default value is 5 elements within Colum/Row.

Touch Pads



Tuning

- **Column/Row Finger Threshold** – Defines the sensor active threshold for touchpad column/row. Default value is 100. Valid range of values is [1…255].

- **Column/Row Noise Threshold** – Defines the sensor noise threshold for touchpad column/row. Count values above this threshold do not update the baseline. 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].

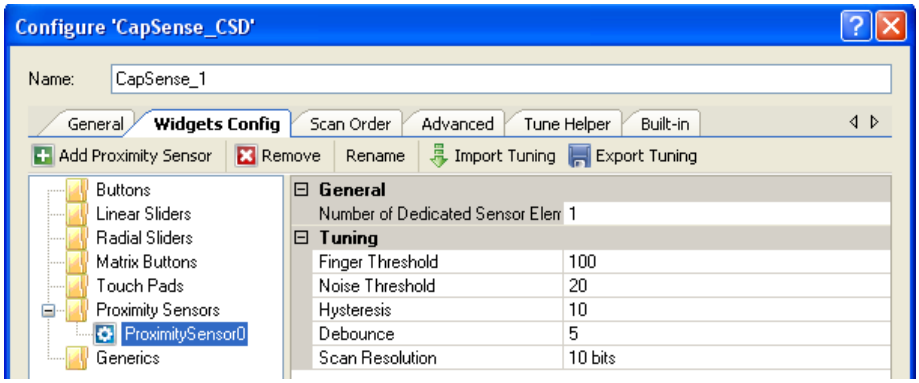
- **Column/Row Scan Resolution** – Defines the scanning resolution of touchpad column/row. This parameter has an effect on the scanning time of all sensors within a column/row of a touchpad widget. The maximum raw count for scanning resolution for N bits is 2^N-1. Increasing the resolution improves sensitivity and the SNR of touch detection. 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 elements within the touch pad column/row. Valid range of values is [2…32]. Default value is 5 elements within a column/row.
- **API Resolution Colum/Row** – Defines the resolution of touch pad column/row. The position values will be changed 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.
  - None
  - Median Filter
  - Averaging Filter
  - First Order IIR 1/2
  - First Order IIR 1/4 – Default
  - Jitter Filter

Proximity Sensors



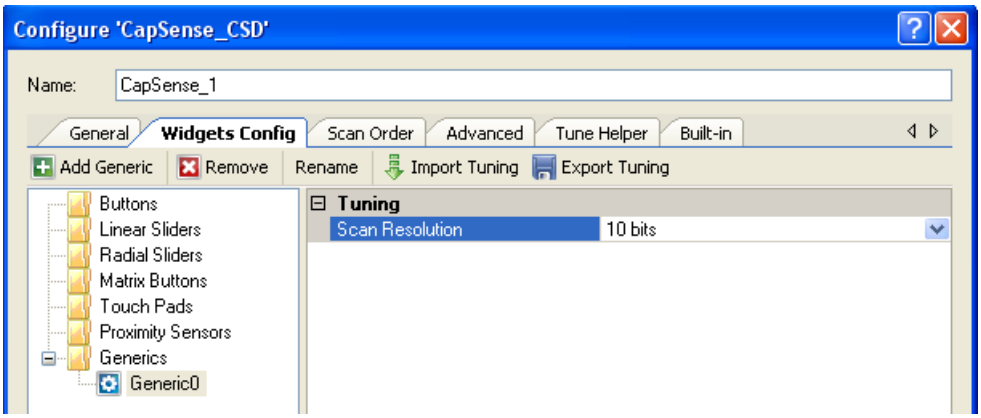
General

- **Numbers of Dedicated Sensors elements** – Selects the number of proximity dedicated sensors:
  - ❑ 0 – The proximity sensor will scan one or more existing sensors to determine proximity. No terminals are allocated for this sensor.
  - ❑ 1 – Number of dedicated proximity sensors in the system. – Default.

Tuning

- **Finger Threshold** – Defines the sensor active threshold. Valid range is [1…255].
- **Noise Threshold** – Defines the sensor noise threshold. Count values above this threshold do not update the baseline. Valid range of values is [1…255].
- **Hysteresis** – Adds the 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. Valid range of values is [1…255].
- **Debounce** – Adds a debounce counter for 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. 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. Use a high resolution for proximity detection. Default value is 10 bits.

Generics



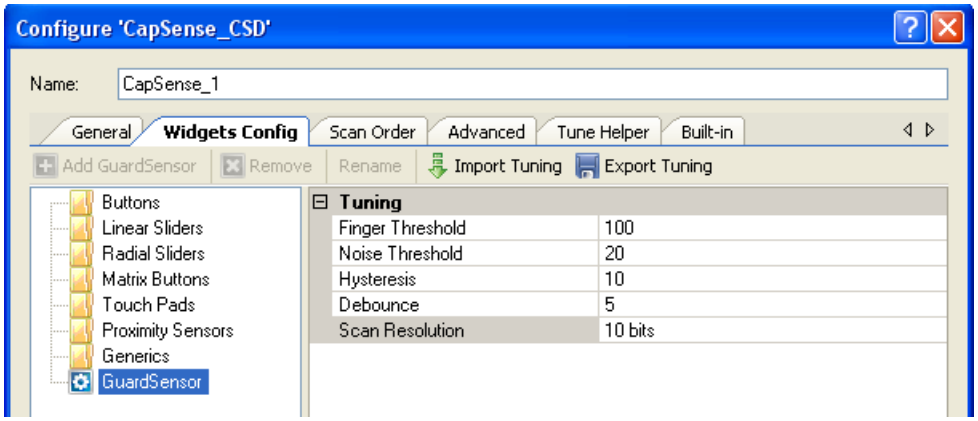
Tuning

- **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. Default value is 10 bits.

One tuning option is available for a generic widget, because the High Level handling is added by the customer for such kind of widgets.

Guard Sensor

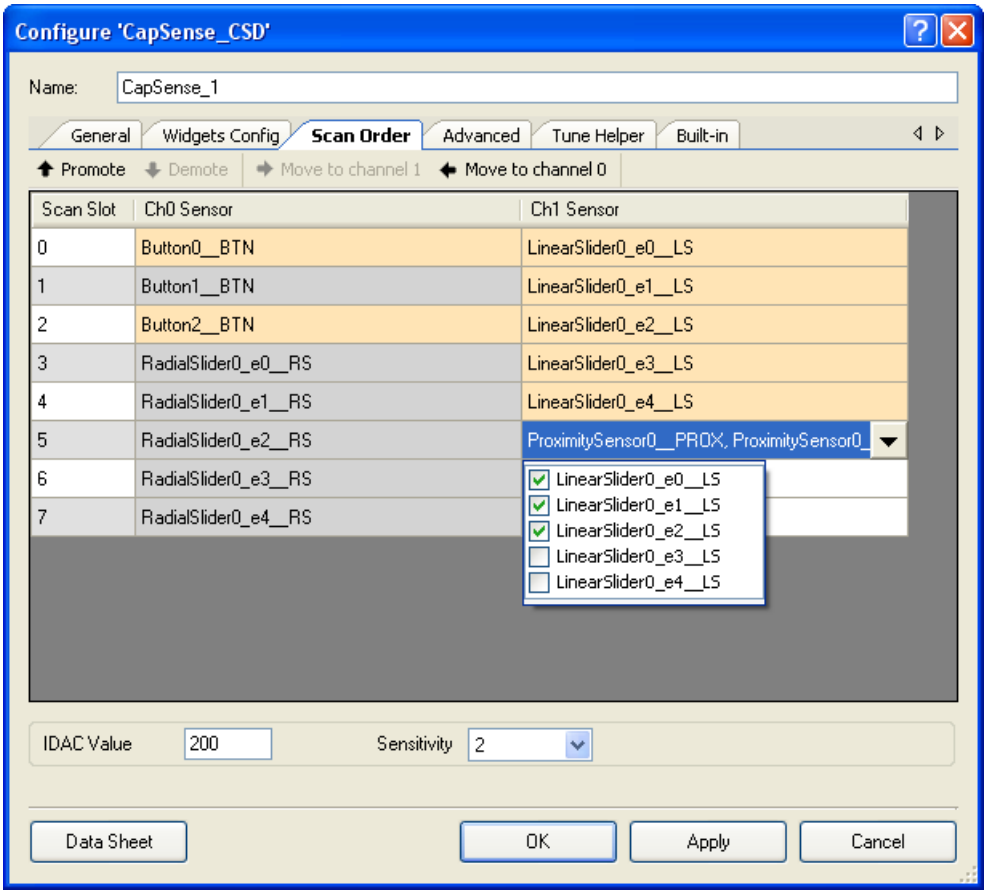
This special sensor is added/removed using the **Advanced Tab**. For more information about this sensor type, refer to "Guard Sensor Implementation" in the Functional Description section of this data sheet.



Tuning:

- **Finger Threshold** – Defines the sensor active threshold. Default value is 100. Valid range of values is [1…255].
- **Noise Threshold** – Defines the sensor noise threshold. Count values above this threshold do not update the baseline. 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. Default value is 10. Valid range of values is [1…255].
- **Debounce** – Adds a debounce counter for 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. 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. Default value is 10 bits.

Scan Order Tab



Toolbar

The toolbar contains the following commands:

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

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

Note The Proximity sensor is excluded from scanning process by default.

IDAC Value

Specifies the IDAC value of selected sensors (0 – 255). The IDAC range should take to account when specifies this value. This option is active only when IDAC is selected as **Current Source** (under the **Advanced** tab). Default value is 200.

Sensitivity

Sensitivity is a 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.

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

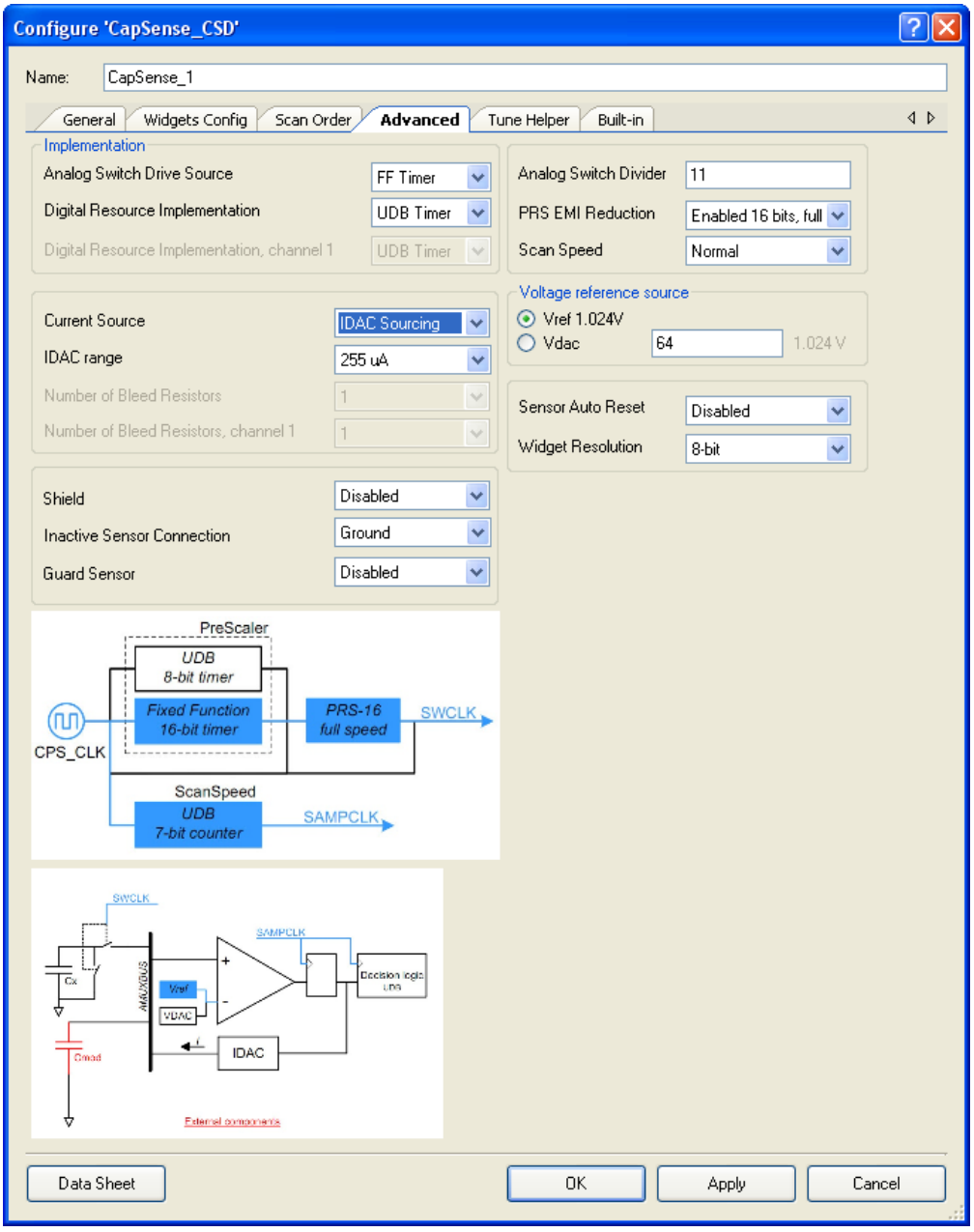
Widget List

Widgets are listed in alternating gray and orange rows in the table. All terminals are associated with a widget (for example, all sensors within a slider widget) share the same color.

Proximity scan sensors can use dedicated proximity sensors, or they can detect proximity from a combination of dedicated sensors and 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 the 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



Analog Switch Drive Source

This parameter specifies the source of analog switch divider. Any additional system resources are consumed if **Direct** is specified.

- Direct
- UDB Timer

- FF Timer – Default

Digital Resource Implementation, channel 0/channel 1

This parameter specifies the type of resources to be consumed for implementation of digital part of CapSense.

- UDB Timer – Default
- FF Timer

Note FF Timers do not support Scan Speed set to Very Fast. Also, the FF Timers option is not supported in PSoC 3 ES3 silicon.

Current Source

This parameter specifies one of CapSense CSD modes of operations.

- IDAC Source - the switch stage is configured to alternate between GND and the AMUX bus. The IDAC sources the current – Default
- IDAC Sink - the switch stage is configured to alternate between Vdd and the AMUX bus. The IDAC sinks the current.
- External Resistor - this is 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 Cmod and a GPIO. The GPIO is configured to Open-Drain Drives Low drive mode. This drive mode allows the Cmod to be discharged through Rb.

IDAC Range

This parameter specifies the IDAC Range. This feature is disabled if **Current Source** is set to External Resistor.

- 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.

Shield electrode

This parameter specifies if shield output 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 unscanning sensor connection. Ground is the default and should be used for the vast majority of application. Shield provides shield signal to all unscanning sensors, the amplitude of signal is equal to Vddio.

- Ground – Default
- Hi-Z Analog
- Shield

Guard Sensor

This parameter enables the guard sensor, which helps detect water drops in the water proof application. 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

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.

PRS EMI Reduction

This parameter specifies if the PRS will be fed pre-charge clock. The clock source for PRS depends on analog switch divider settings:

- Disabled
- Enabled 8 bits
- Enabled 16 bits, full speed – Default

- Enabled 16 bits, ¼ full speed

Note The "Enabled 16 bits, ¼ full speed" implementation requires a 4 times faster clock to obtain the same output as "Enabled 16 bits, full speed."

Scan Speed

This parameter specifies the digital clock frequency and has an effect on scan time of sensors.

- 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	61	85	130	220
9	85	130	220	403
10	130	220	403	766
11	220	403	766	1500
12	403	766	1500	2950
13	766	1500	2950	5870
14	1500	2950	5870	11700
15	2950	5870	11700	23360
16	5870	11700	23360	46690

Note Master Clock 67 MHz operation and CapSense CSD clock 24 MHz. Scanning time was measured as time interval of one sensor scan. This time includes sensor setup time, sample conversion interval and data pre-proceessing time.

Voltage Reference Source

This parameter specifies the type and level of reference source voltage. It is better to have a much as possible bigger value of reference source for IDAC Source mode and smaller for IDAC Sink or External Resistor Current Sources.

- 1.024V – Default
- VDAC

Note The reference source VDAC is only available when **Current Source** is set to IDAC Source.

Sensor Auto Reset

This parameter enables auto reset. This 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).

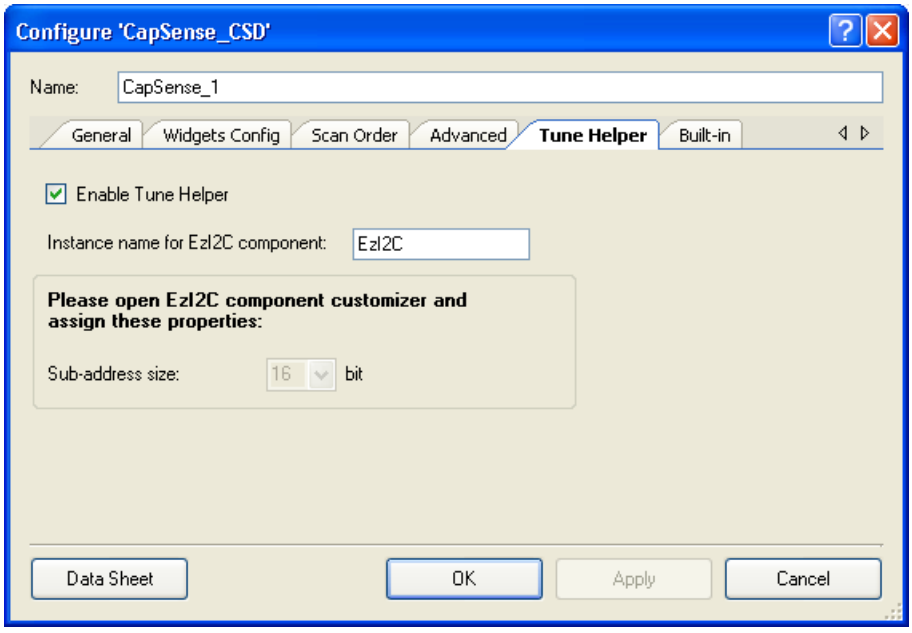
- Enable
- Disable – Default

Widget Resolution

This parameter specifies the Signal resolution. 8 bits (1 byte) is the default option and should be used for the vast majority of applications.

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

Tune Helper Tab



Enable Tune Helper

This parameter adds code to functions to support communication with the Tuner GUI. This feature should be checked if the Tuner GUI will be used. If this option is not checked, the communication functions do nothing. Therefore, you may not remove these functions when the tuning method will be changed. Default – Checked.

Instance name for EzI2C component

This parameter defines the instance name for the EZ I^2C component in your design to be used for communication with the Tuner GUI.

Note There is no validation to ensure the actual instance name matches the instance name entered here. You must make sure they match.

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

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
	1	TBD	1	1	0	TBD	TBD	TBD
	2	TBD	1	1	0	TBD	TBD	TBD

Tuner GUI User Guide

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

The

Capacitive Sensing (CapSense® CSD) 1
Features 1
General Description 1
When to use a CapSense Component 1
Input/Output Connections 1
clock – Input * 1
shield – Output * 2
vref – Output * 2
Parameters and Setup 2
General Tab 2
Load Settings/Save Settings 3
Tuning method 3
Number of channels 3
Raw Data Noise Filter 4
Water proofing and detection 4
Enable clock input 4
Scan Clock 4
Widgets Config Tab 5

Toolbar 5
Buttons 6
Linear Sliders 7
Radial Slider 8
Matrix Buttons 9
Touch Pads 10
Proximity Sensors 11
Generics 12
Guard Sensor 13
Scan Order Tab 14
Toolbar 14
IDAC Value 14
Sensitivity 15
Widget List 15
Advanced Tab 16
Analog Switch Drive Source 16
Digital Resource Implementation, channel 0/channel 1 17
Current Source 17
IDAC Range 17
Number of Bleed Resistors, channel 0/channel 1 17
Shield electrode 18
Inactive Sensor Connection 18
Guard Sensor 18
Analog Switch Divider 18
PRS EMI Reduction 18
Scan Speed 19
Voltage Reference Source 20
Sensor Auto Reset 20
Widget Resolution 20
Tune Helper Tab 21
Enable Tune Helper 21
Instance name for EzI2C component 21

Placement 21
Resources 22
Tuner GUI User Guide 22
CapSense Tuning Process 31
Create a design in PSoC Creator 31
Place and Configure EZ I²C Component 31
Place and Configure CapSense Component 32
Selecting Auto (SmartSense) 33
Add Code 34
Build the design and program the PSoC device 35
Launch the Tuner application 35
Configure communication parameters 36
Start tuning 36
Edit CapSense parameter values 37
Repeat as needed 37
Close the Tuner application 37
Tuner GUI Interface 38
General Interface 38
Tuning Tab 39
Graphing Tab 40
Logging Tab 41
Application Programming Interface 42
General APIs 42
void CapSense_Start (void) 43
Description: 43
Parameters: 43
Return Value: 43
Side Effects: 43
void CapSense_Stop (void) 43
Description: 43
Parameters: 43
Return Value: 43

Side Effects: 43
void CapSense_Sleep(void) 43
Description: 43
Parameters: 43
Return Value: 43
Side Effects: 43
void CapSense_WakeUp(void) 44
Description: 44
Parameters: 44
Return Value: 44
Side Effects: 44
void CapSense_Init(void) 44
Description: 44
Parameters: 44
Return Value: 44
Side Effects: 44
void CapSense_Enable(void) 44
Description: 44
Parameters: 44
Return Value: 44
Side Effects: 44
void CapSense_SaveConfig(void) 44
Description: 44
Parameters: 44
Return Value: 44
Side Effects: 44
void CapSense_RestoreConfig(void) 45
Description: 45
Parameters: 45
Return Value: 45
Side Effects: 45
Scanning Specific APIs 45

void CapSense_ScanSensor (uint8 sensor) 46
Description: 46
Parameters: 46
Return Value: 46
Side Effects: 46
void CapSense_ScanEnabledWidgets (void) 46
Description: 46
Parameters: 46
Return Value: 46
Side Effects: 46
uint8 CapSense_IsBusy (void) 47
Description: 47
Parameters: 47
Return Value: 47
Side Effects: 47
void CapSense_SetScanSlotSettings (uint8 slot) 47
Description: 47
Parameters: 47
Return Value: 47
Side Effects: 47
void CapSense_ClearSensors (void) 47
Description: 47
Parameters: 47
Return Value: 47
Side Effects: 47
void CapSense_EnableSensor(uint8 sensor) 48
Description: 48
Parameters: 48
Return Value: 48
Side Effects: 48
void CapSense_DisableSensor(uint8 sensor) 48
Description: 48

Parameters: 48
Return Value: 48
Side Effects: 48
uint16 CapSense_ReadSensorRaw (uint8 sensor) 48
Description: 48
Parameters: 48
Return Value: 48
Side Effects: 48
void CapSense_SetRBleed(uint8 rbleed) 49
Description: 49
Parameters: 49
Return Value: 49
Side Effects: 49
High Level APIs 49
void CapSense_InitializeSensorBaseline (uint8 sensor) 50
Description: 50
Parameters: 50
Return Value: 50
Side Effects: 50
void CapSense_InitializeAllBaselines(void) 51
Description: 51
Parameters: 51
Return Value: 51
Side Effects: 51
void CapSense_UpdateSensorBaseline (uint8 sensor) 51
Description: 51
Parameters: 51
Return Value: 51
Side Effects: 51
void CapSense_UpdateEnabledBaselines(void) 51
Description: 51
Parameters: 51

Return Value: 51
Side Effects: 51
void CapSense_EnableWidget (uint8 widget) 52
Description: 52
Parameters: 52
Return Value: 52
Side Effects: 52
void CapSense_DisableWidget (uint8 widget) 52
Description: 52
Parameters: 52
Return Value: 52
Side Effects: 52
uint8 CapSense_CheckIsWidgetActive(uint8 widget) 53
Description: 53
Parameters: 53
Return Value: 53
Side Effects: 53
uint8 CapSense_CheckIsAnyWidgetActive(void) 53
Description: 53
Parameters: 53
Return Value: 53
Side Effects: 53
uint16 CapSense_GetCentroidPos(uint8 widget) 54
Description: 54
Parameters: 54
Return Value: 54
Side Effects: 54
uint16 CapSense_GetRadialCentroidPos(uint8 widget) 55
Description: 55
Parameters: 55
Return Value: 55
Side Effects: 55

uint8 CapSense_GetTouchCentroidPos (uint8 widget) 56
Description: 56
Parameters: 56
Return Value: 56
Side Effects: 56
Tuner Helper APIs 56
void CapSense_TunerStart (void) 57
Description: 57
Parameters: 57
Return Value: 57
Side Effects: 57
void CapSense_TunerComm (void) 57
Description: 57
Parameters: 57
Return Value: 57
Side Effects: 57
Data Structures 57
CapSense_SensorRaw [ ] 58
CapSense_SensorEnableMask [ ] 58
CapSense_SensorEnableMask [0] 58
CapSense_portTable[ ] and CapSense_maskTable[ ] 58
CapSense_SensorBaselineLow[ ] 59
CapSense_SensorBaseline[ ] 59
CapSense_SensorSignal[ ] 59
CapSense_SensorOnMask[ ] 59
CapSense_SensorOnMask[0] 59
Constants 59
Sensor Constants 60
Widget Constants 61
Sample Firmware Source Code 61
Pin Assignments 63
Sides 63

Sensor Pins – CapSense_cPort – Pin Assignment 63
CapSense_cCmod_Port – Pin Assignment 63
CapSense_cRb_Ports – Pin Assignment 64
Interrupt Service Routines 65
Functional Description 65
Definitions 65
Sensor 65
Scan Sensor 65
CapSense Widget 65
FingerThreshold 65
NoiseThreshold 66
Debounce 66
Hysteresis 66
API Resolution – Interpolation and Scaling 66
Diplexing 67
Diplexing Sequence for Different Slider Segment Counts 68
Filters 70
Position Median Filter 70
Position Averaging Filter 70
Position First Order IIR Filter 70
Position Jitter Filter 71
Raw Data Median Filter 71
Raw Data Averaging Filter 71
Raw First Order IIR Filter 71
Raw Data Jitter Filter 71
Water Influence on CapSense System 71
Water proofing and detection 72
Shielding Electrode 72
Shield Electrode Usage and Restrictions 73
Current Mode IDAC Source 73
Current Mode Sink and External Resistor 74
Guard Sensor Implementation 74

Block Diagram and Configuration 75
IDAC sourcing 75
IDAC Sinking 76
IDAC disable, use external R_b 76
DC and AC Electrical Characteristics 77
5.0V/3.3V DC and AC Electrical Characteristics 77
Component Changes 77

CapSense Tuner is capable of tuning the CapSense component when in manual tuning mode. It is capable of displaying the tuning values (read only) and performance when the component is in smart tune mode. No tuning will be supported when the component is in no tuning mode as all parameters are stored in Flash and read only.

CapSense Tuning Process

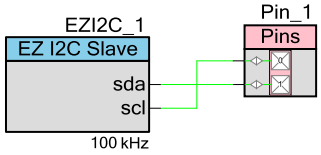
The following is the typical usage for tuning a CapSense component:

Create a design in PSoC Creator

Refer to the PSoC Creator Help as needed.

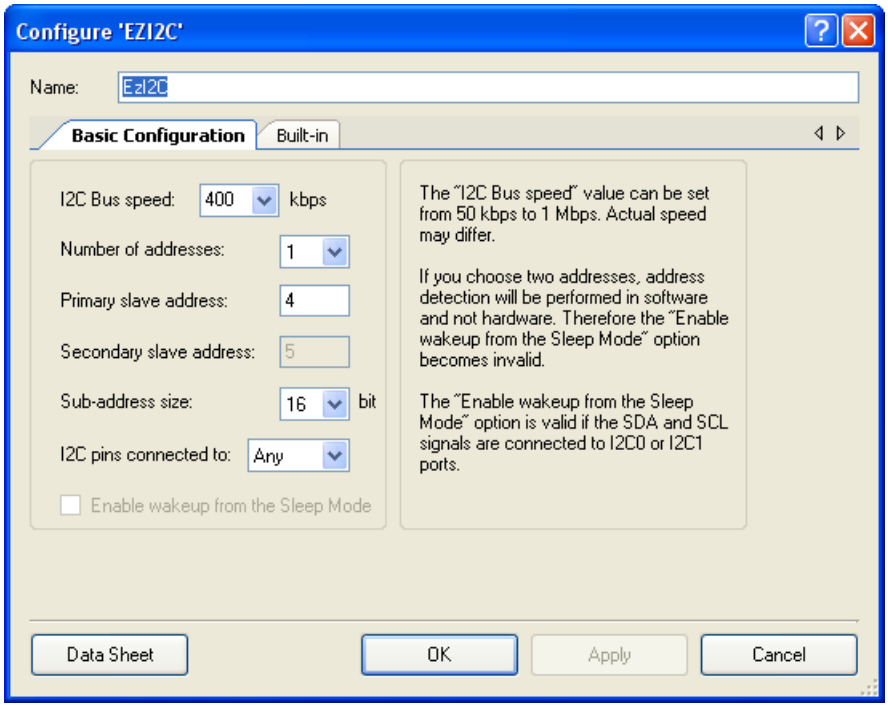
Place and Configure EZ I²C Component

Drag an EZ I²C component from the Component Catalog onto your design.



Double-click it to open the Configure dialog.

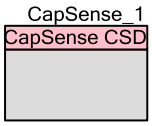
Change the parameters as follows, and click **OK** to close the dialog.



- 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.

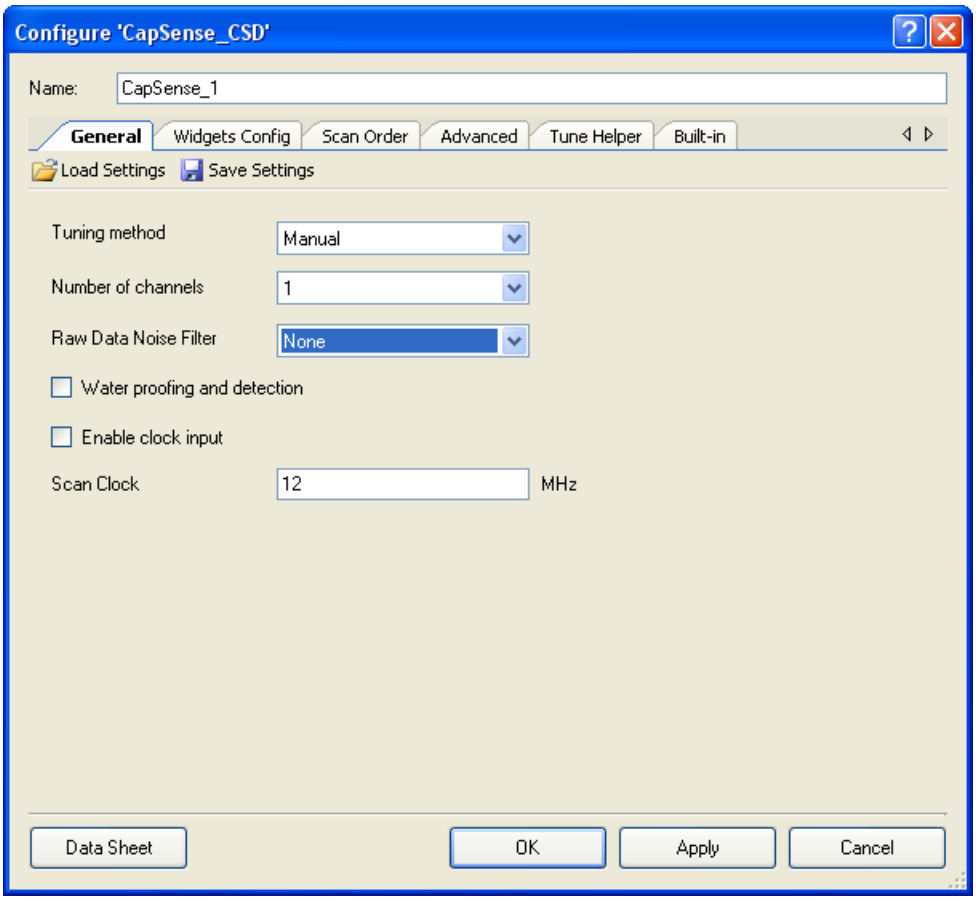
Place and Configure CapSense Component

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 selected parameters.



Selecting Auto (SmartSense)

Auto (SmartSense) allows the tuning of the CapSense CSD component automatically. The average parameters, acceptable for all sensors, are selected by firmware. Additional RAM and CPU time are used in this mode. Auto (SmartSense) eliminates the error prone and tedious process of manually tuning the CapSense CSD component parameters to ensure proper system operation. Selecting Auto (SmartSense) tunes the following CSD parameters:

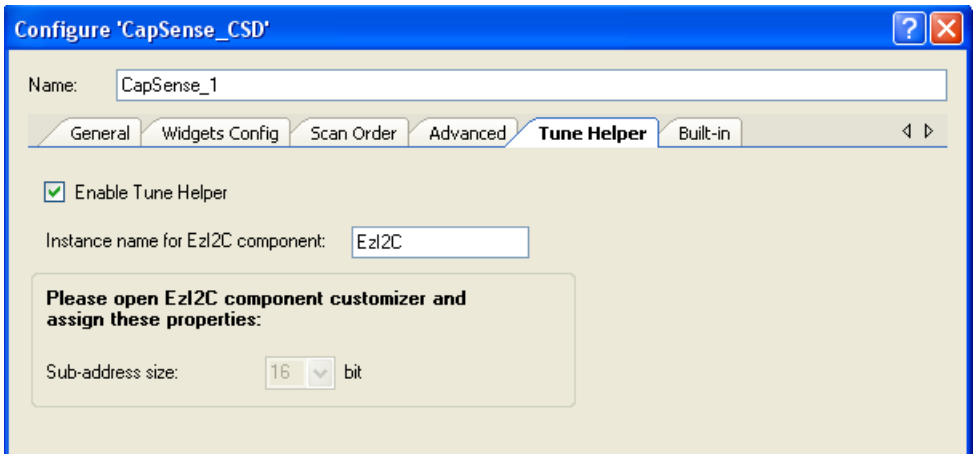
Parameter	Calculation
Finger Threshold	Calculated continuously during sensors scanning.
Noise Threshold	Calculated continuously during sensors scanning.
IDAC Value	Calculated once on CapSense CSD start up.
Analog Switch Divider	Calculated once on CapSense CSD start up.

Parameter	Calculation
Scan Resolution	Calculated once on CapSense CSD start up.

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

Setting	Restriction
Scan Clock	Internal Clock equal 24MHz.
Current Source	IDAC Sourcing.
Analog Switch Source	UDB/FF Timer.
PRS EMI Reduction	Enabled 16 bits.
Scan Speed	Fast.
Vref	1.024 V
Widget Resolution	8 bits

In Tune Helper tab: EzI2C component instance name must be filled and Enable Tune Helper checkbox must be checked



Add Code

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

```
void  main()
```

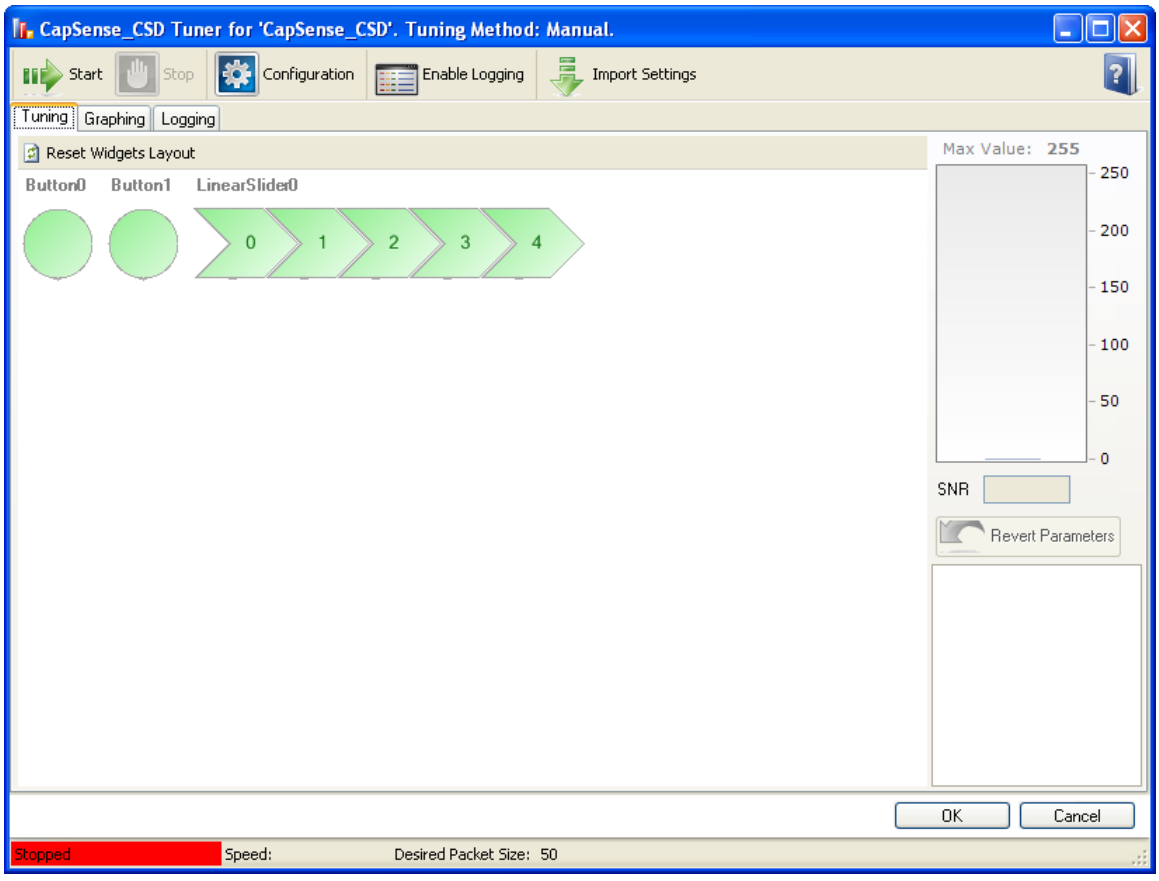
```
{
    CYGlobalIntEnable;
    CapSense_1_TunerStart();
    while(1)
    {
        CapSense_1_TunerComm();
    }
}
```

Build the design and program the PSoC device

Refer to the PSoC Creator Help as needed.

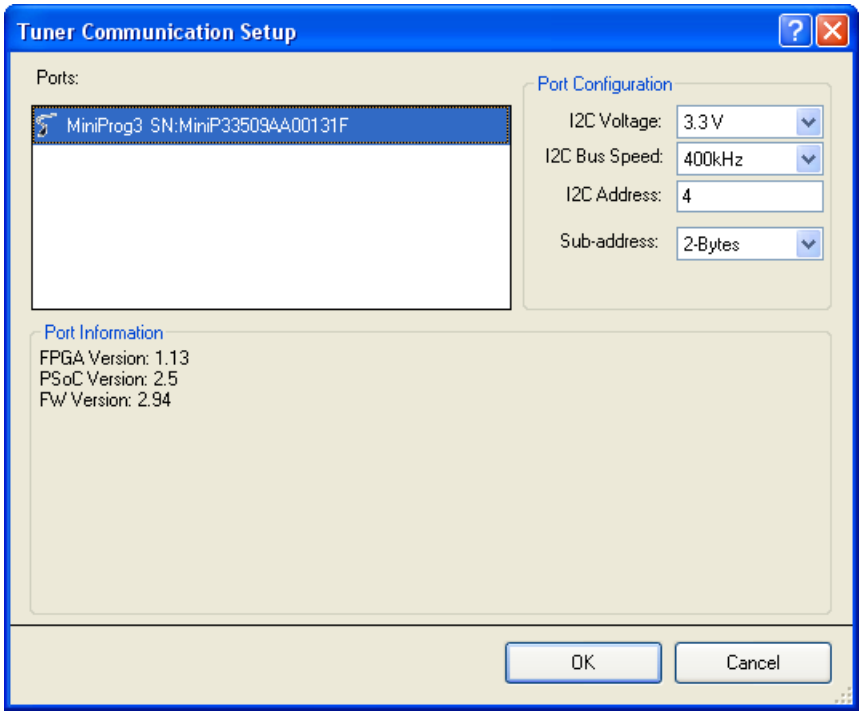
Launch the Tuner application

Right-click and select **Launch Tuner** from the CapSense_CSD instance context menu. The Tuner application opens.



Configure communication parameters

1. Click Configuration to open the Tuner Communication dialog.



Sets the communication parameters.

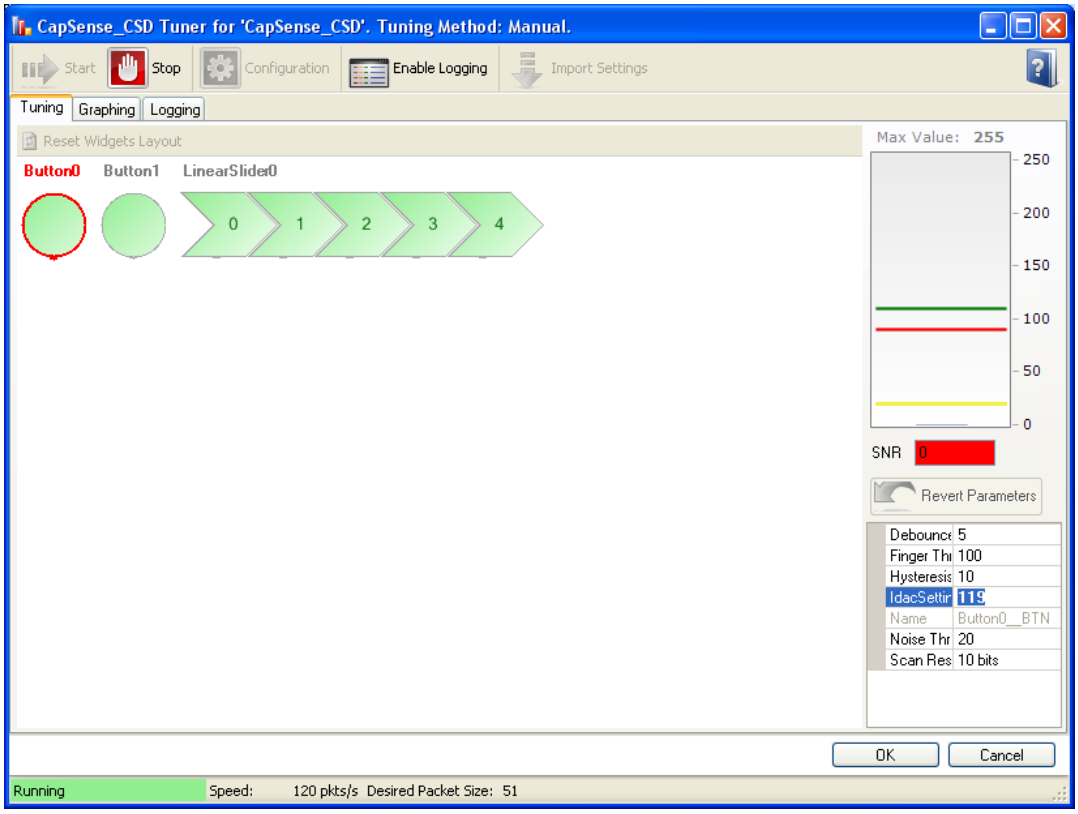
Important: Properties must be identical to EZ I^{2}C component: **I2C Bus Speed**, **I2C Address**, **Sub-address** = 2-Bytes.

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. 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.



Repeat as needed

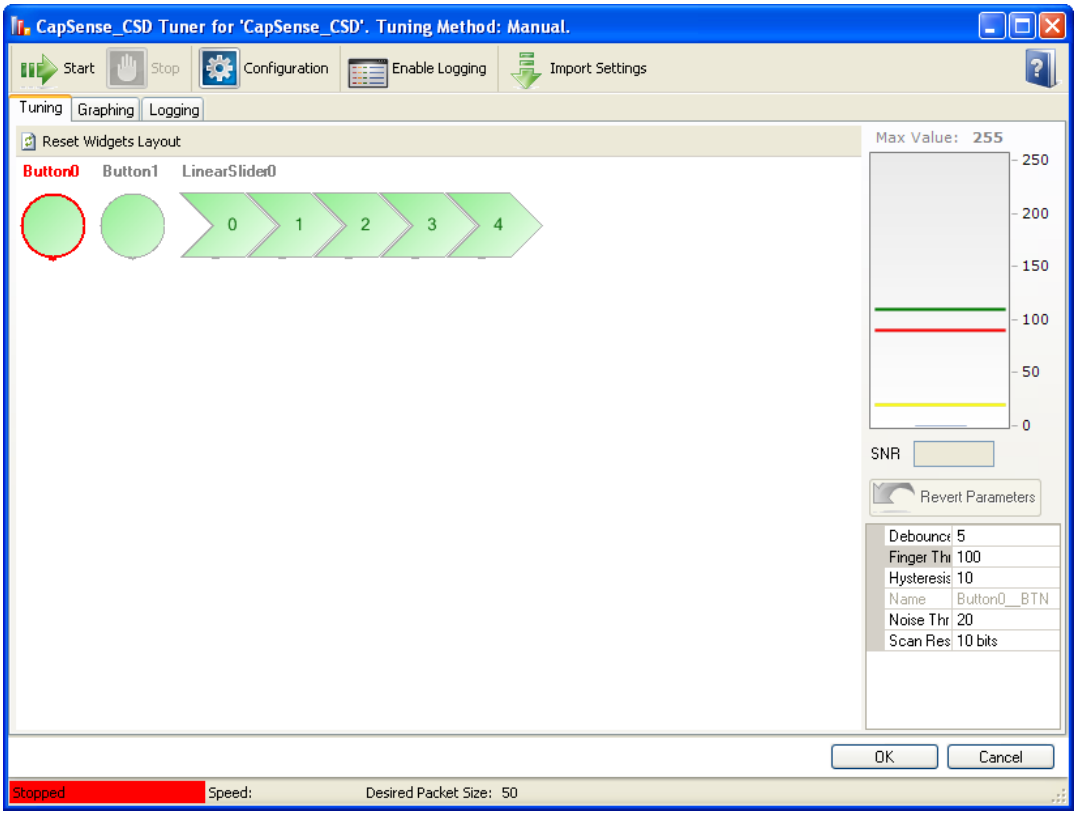
Repeats steps as needed until tuning is complete.

Close the Tuner application

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

Tuner GUI Interface

General Interface



Top panel buttons:

- **Start** – Starts reading and displaying data from the chip. Also starts graphing and logging if configured.
- **Stop** – Stops reading and displaying data from the chip.
- **Configuration** – Opens the Communication Configuration dialog.
- **Enable Logging** – Enables logging data received from device.
- **Import Settings** – Imports settings from XML tuning file and reloads all data in tuner.
- **Help** – Opens help file.

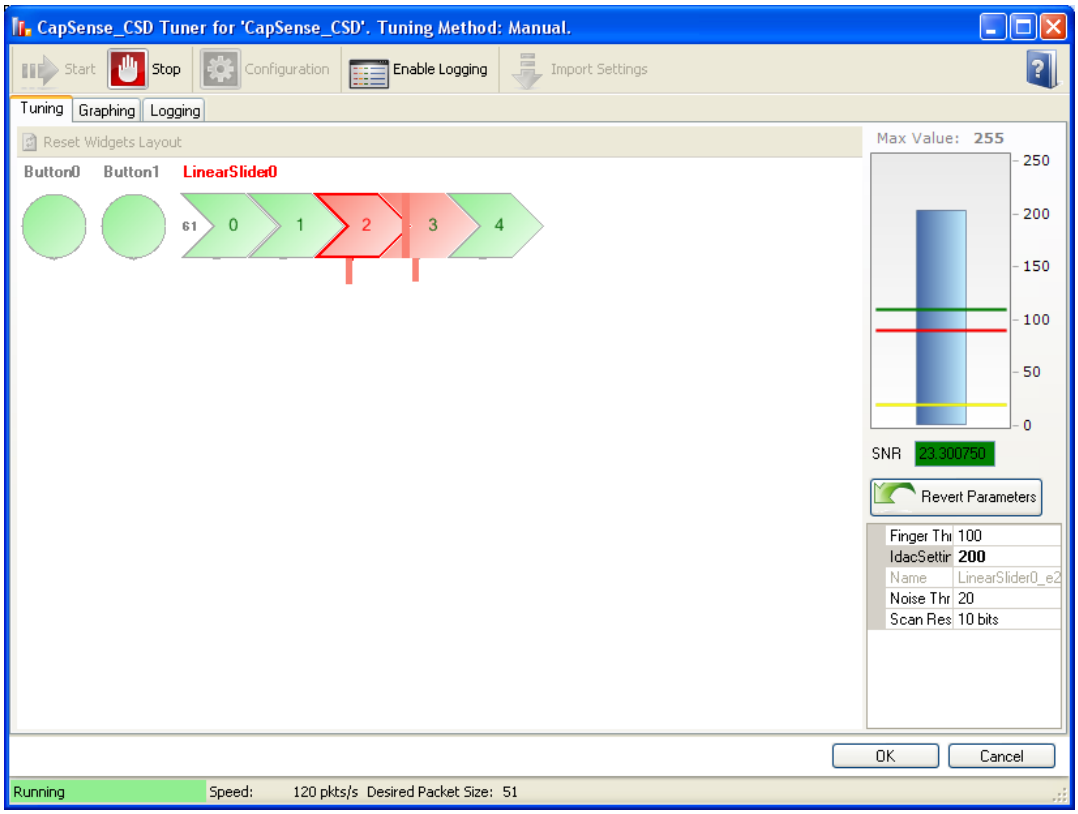
Tabs

- **Tunning Tab** – Displays all of the component configured widgets in the large workspace. This allows you to arrange the widgets similar to how they appear on the PCB. It used for tuning widget parameters and visualization widgets states.
- **Graphing Tab** – Displays widget data on charts.
- **Logging Tab** – Provides logging data functionality and debugging features.

Bottom panel buttons:

- **OK**– Commits the current values of parameters to the CapSense component instance and exit the GUI.
- **Cancel** – Exits the GUI without committing the values of parameters to the instance.

Tuning Tab

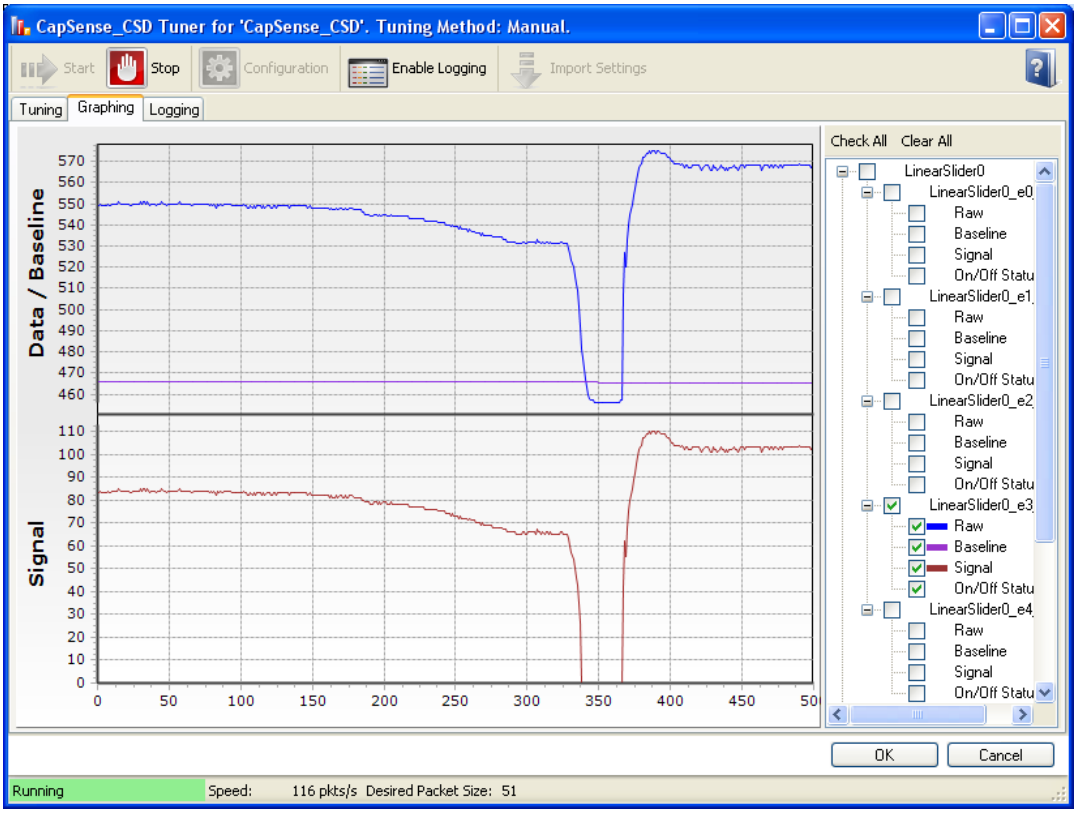


- **Widgets schematic** – contains graphic widgets representation.
- **Revert Parameters** button – Resets the parameters to their initial values (initial value is displayed when the GUI was launched) and sends those values to the chip.
- **Bar graph** – Displays signal value for selected sensor.

- ❑ The maximum scale of the detailed view bar graph can be adjusted by double clicking on Max Value label (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.

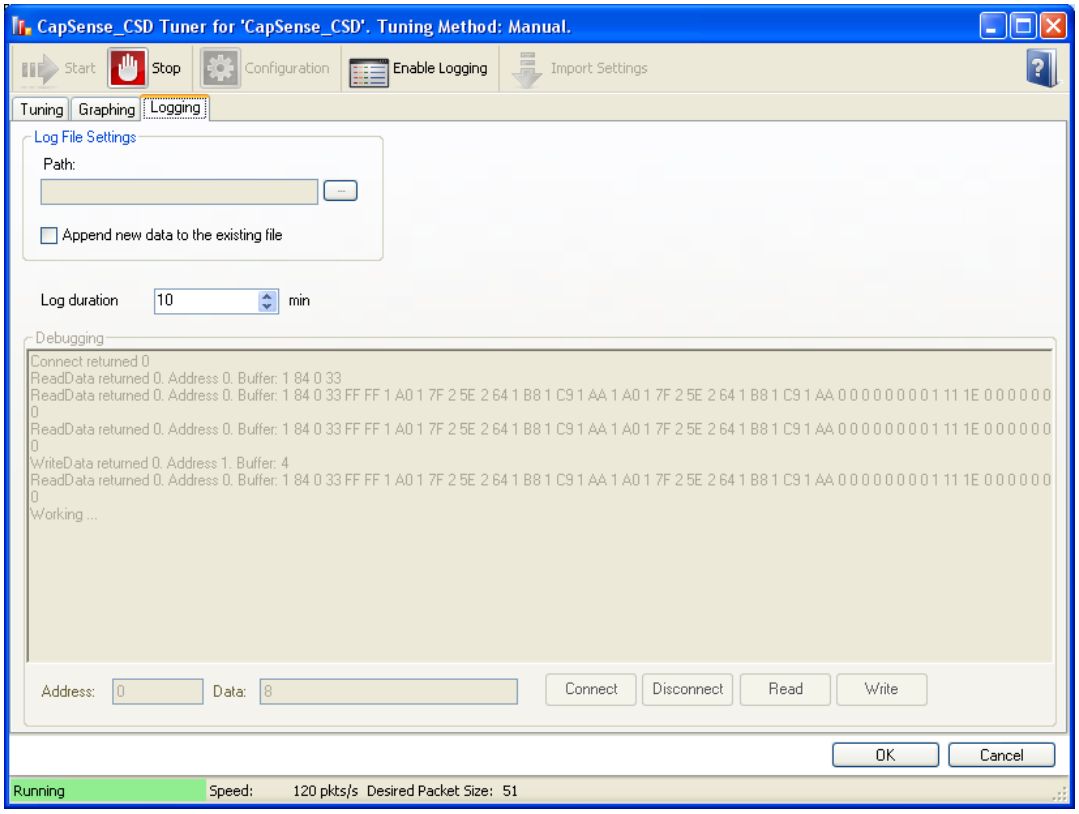
- SNR – the signal to noise ratio is computed in real time for selected sensor. SNR value below 5 is colored red, 5 to 10 yellow, greater than 10 is colored green.
- **Sensor properties** – Displays properties for selected sensor.
- **General CapSense properties** (Read Only) – Displays properties for CapSense.

Graphing Tab



- **Charts area** – Displays charts for selected items.
- **Tree view** – Provides all combinations of data for widgets and sensors which can be shown on chart and be logged to file (if logging feature is enabled).

Logging Tab



- User defines data which will be logged by checking checkboxes on the Graphing tab (TreeView on Graphing tab contains all possible data that can be logged).
- Path – Defines logging file path (file extension is .csv).
- Append new data to existing file checkbox – If checked, new data is appended to existing file. If not, old data will be erased from the file.
- Log duration – Defines log duration in minutes.
- Connect – Connects to device.
- Disconnect – Disconnects from device.
- Read – Reads data from device. Address field defines the address in buffers. Data field defines count of bytes to read.
- Write – Writes data to device. Address field defines address in buffers. Data field defines buffer to write.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to 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:

Function	Description
CapSense_Init	Initializes default CapSense configuration provided with customizer
CapSense_Enable	Enables Active mode power template bits for number of component used within CapSense.
CapSense_Start	Initializes registers and starts the CSD method of CapSense component.
CapSense_Stop	Stops the sensors scanner, disables internal interrupts, and calls CapSense_ClearSensors() to reset all sensors to an inactive state.
CapSense_Sleep	Disables Active mode power template bits and resets all sensors to an inactive state.
CapSense_WakeUp	Restores CapSense configuration and non-retention register values.
CapSense_SaveConfig	Reset all sensors to an inactive state.
CapSense_RestoreConfig	Restores CapSense configuration and non-retention register values.

void CapSense_Start (void)

Description: Initializes registers and starts the CSD method of CapSense component. Reset all sensors to an inactive state. Enables interrupts for sensors scanning.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_Stop (void)

Description: Stops the sensors scanner, disables internal interrupts, and resets all sensors to an inactive state. Disables Active mode power template bits for number of component used within CapSense.

Parameters: None

Return Value: None

Side Effects: This function should be called after scans will be completed.

void CapSense_Sleep(void)

Description: Disables Active mode power template bits 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.

void CapSense_WakeUp(void)

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

Parameters: None

Return Value: None

Side Effects: None

void CapSense_Init(void)

Description: Inits default CapSense configuration provided with customizer that defines mode of component operations 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 number of component used within CapSense.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_SaveConfig(void)

Description: Reset all sensors to an inactive state.

Parameters: None

Return Value: None

Side Effects: This function should be called after scans will be completed.

void CapSense_RestoreConfig(void)

Description: Restores CapSense configuration and non-retention register values.

Parameters: None

Return Value: None

Side Effects: This function should be called after scans will be completed.

Scanning Specific APIs

These API functions are used to implement scanning loop.

Function	Description
CapSense_ScanSensor	Sets scan settings and starts scanning a sensor or group of combined sensors on each channel.
CapSense_ScanEnabledWidgets	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 measure during the next measurement cycle.
CapSense_DisableSensor	Disables the selected sensor.
CapSense_ReadSensorRaw	Returns sensor raw data from the CapSense_SensorResult[ ] array.
CapSense_SetRBleed	Sets the pin to use for the bleed resistor (Rb) connection.

void CapSense_ScanSensor (uint8 sensor)

Description: This function sets scan settings and starts scanning a sensor or pair of combined 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 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: Scans all of the enabled widgets. Starts scanning a sensor or pair of sensors within enabled widget. The isr proceeding scanning next sensor or pair till all enabled widgets will be 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 fast response desired of other widget types.

Parameters: None

Return Value: None

Side Effects: If no enabled widgets the function call has no effect.
Proximity widgets scanning is disabled by default.

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 of the selected scan slot (sensor or pair of sensors). The scan settings incorporate IDAC value (for IDAC configurations) for every sensor and resolution. The resolution is the same for all sensors within 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 Analog MUX Bus and putting them to inactive state.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_EnableSensor(uint8 sensor)

Description: Configures the selected sensor to measure during the next measurement cycle. The corresponding pins are set to Analog High-Z mode and connected to the Analog Mux Bus. This also enables the comparator function.

Parameters: (uint8) sensor: Sensor number

Return Value: None

Side Effects: None

void CapSense_DisableSensor(uint8 sensor)

Description: Disables the selected sensor. The corresponding pin is disconnected from the Analog Mux Bus and putting them to 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 CapSense_SensorResult[ ] array. Each scan 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: (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 customizer. The function overwrites the component parameter setting. This function is available only if Current Source is 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: Ordering number for bleed resistor defined in CapSense customizer.

Return Value: None

Side Effects: The number of bleed resistors is restricted by 3. The function do not check the out or 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 each sensor.
CapSense_UpdateSensorBaseline	The historical count value, calculated independently for each sensor, is called the sensor's baseline.
CapSense_UpdateEnabledBaselines	Checks CapSense_SensorEnableMask[] and calls the CapSense_UpdateSensorBaseline function to update the baselines for enabled sensors.
CapSense_EnableWidget	Enable all widget elements (sensors) to scanning process.
CapSense_DisableWidget	Disable all widget elements (sensors) from scanning process.

Function	Description
CapSense_CheckIsWidgetActive	Compares the selected sensor of widget to the CapSense_Signal[] array to its finger threshold.
CapSense_CheckIsAnyWidgetActive	Uses the CapSense_CheckIsWidgetActive() function to find if any widget is in active state.
CapSense_GetCentroidPos	Checks the CapSense_ SensorSignal[] array for a centroid within slider specified range.
CapSense_GetRadialCentroidPos	Checks the CapSense_ SensorSignal [] array for a centroid within slider specified range.
CapSense_GetTouchCentroidPos	If a finger is present, this function calculates the X and Y position of the finger by calculating the centroids within touch pad specified range.

void CapSense_InitializeSensorBaseline (uint8 sensor)

Description: Loads the CapSense_SensorBaseline[sensor] array element with an initial value by scanning the selected sensor. 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 loads the CapSense_SensorBaseline[ ] array with an 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: Updates the CapSense_SensorBaseline[ ] array using the LP filter with k = 256. The signal calculates the difference of count by subtracting the previous baseline and noise threshold from the current raw count value. The baseline stops updating if signal is greater that zero. Raw data filters are applied to the values if enabled.

Parameters: (uint8) sensor: Sensor number

Return Value: None

Side Effects: None

void CapSense_UpdateEnabledBaselines(void)

Description: Uses the CapSense_UpdateSensorBaseline function to update the baselines for all sensors. Raw data filters are applied to the values if enabled.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_EnableWidget (uint8 widget)

Description: Enable all widget elements (sensors) to 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: Disable all widget elements from 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 of the CapSense_Signal[ ] array to its finger threshold. Hysteresis and Debounce are taken into account. The Hysteresis value is added or subtracted from the finger threshold based on whether the sensor is currently active. If the sensor is active, the threshold is lowered by the hysteresis amount. If it is inactive, the threshold is raised by the hysteresis amount. The Debounce counter added to the sensor active transition. This function also updates the sensor's bit in the CapSense_SensorOnMask[ ] array.

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 is/are active, 0 if all sensors within widget are inactive.

Side Effects: This function update values in CapSense_SensorOnMask[ ] for all sensors belong to widget. The debounce counter also modifies on every call when there is a transition to 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 none of widgets are active.

Side Effects: Use function CapSense_CheckIsWidgetActive and have the same side effects but for all sensors.

uint16 CapSense_GetCentroidPos(uint8 widget)

Description: Checks the CapSense_Signal[ ] array for a centroid within slider specified range. The centroid position is calculated to the API resolution specified in the CapSense customizer. The position filters are applied to the result if enabled. This function is available only if a linear slider 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 sensor within the slider widget is 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 type argument provided to this function. The unproper widget type 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 centroid result. The noise threshold should be set carefully (high enough above the noise level) so that noise will not generate a false centroid.

uint16 CapSense_GetRadialCentroidPos(uint8 widget)

Description: Checks the CapSense_Signal[ ] array for a centroid within slider specified range. The centroid position is calculated to the API resolution specified in the CapSense customizer. The position filter is applied to the result if enabled. This function is available only if a radial slider 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 sensor within the slider widget is 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 type argument provided to this function. The unproper widget type 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 centroid result. The noise threshold should be set carefully (high enough above the noise level) so that noise will not generate a false centroid.

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 touch pad specified range. 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. The position filter is applied to the result if enabled. This function is available only if a touch pad 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 touchspad, 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 type argument provided to this function. The unproper widget type provided will cause unexpected position calculations.

Tuner Helper APIs

These API functions are used to work with Tuner GUI.

Function	Description
CapSense_TunerStart	Initialize CapSense CSD component and EzI2C component, initialize baselines and start the sensor scanning loop.
CapSense_TunerComm	Execute communication functions with Tuner GUI.

void CapSense_TunerStart (void)

Description: Initialize the CapSense CSD component and EZ I²C component. Also initialize baselines and start the sensor scanning loop.

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 Tuner GUI. Read new parameters from Tuner GUI and apply it to 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.

Parameters: None

Return Value: None

Side Effects: None

Data Structures

API functions use different global arrays. You should not alter these arrays manually. You can inspect these values for debugging purposes, however. For example, you can use a charting tool to display the contents of the arrays. There are several global arrays:

- CapSense_SensorRaw [ ]
- CapSense_SensorBaseline [ ]
- CapSense_SensorBaselineLow[ ]
- CapSense_SensorSignal [ ]

CapSense_SensorRaw [ ]

This array contains the raw data of 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]

This array 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 sensor will be scanned on next function call CapSense_ScanEnabledWidgets(), 1 – will be scanned, 0 – not. 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 mask for every sensor.

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

CapSense_SensorBaselineLow[ ]

This array holds the fractional byte of baseline data of each sensor. The arrays size is equal to the total number of sensors.

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[ ], CapSense_SensorBaselineLow[ ] and CapSense_SensorSignal[ ] arrays are updated by these functions:

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

CapSense_SensorSignal[ ]

This array holds the difference of count by subtracting the previous baseline and noise threshold from the current raw count value of each sensor. The array size is equal to the total number of sensors. The **Widget Resolution** parameter defines the resolution of this array a 1 byte or 2 bytes.

CapSense_SensorOnMask[ ]

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

CapSense_SensorOnMask[0]

This array 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 scan sensors. The value of a bit is 1 if the button is on and 0 if the button is off. 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 total number of sensors within CapSense CSD component.

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

- CapSense_TOTAL_SENSOR_COUNT__CH0 – Defines total number of sensors belongs to channel 0.
- CapSense_TOTAL_SENSOR_COUNT__CH1 – Defines total number of sensors belongs to channel 1.
- CapSense_CSD_TOTAL_SCANSLOT_COUNT – Defines the bigger sensors count in channel 0 and 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 indentifier). The widget name must be unique within 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

Alias	Description
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_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

The following is a C language example demonstrating the basic functionality of the CapSense component. This example assumes the component has been placed in a design with the default name "CapSense_1" with two button widgets given default names.

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
#include <device.h>
```

```
void LCD_DisplayState(void);

void main()
{
    CYGlobalIntEnable;

    /* Start CapSense and baselines */
    CapSense_1_Start();

    /* Initialize LCD */
    LCD_Char_1_Start();
    LCD_Char_1_Position(0u, 0u);
    LCD_Char_1_PrintString("BUTTONS TEST");

    /* Initialize UART TX only */
    CapSense_1_InitializeAllBaselines();

    /* Starts scan all sensors */
    CapSense_1_ScanEnabledWidgets();

    /* Sensor Scanning Loop */
    while(1)
    {
        if (CapSense_1_IsBusy() == 0u)
        {
            CapSense_1_UpdateEnabledBaselines();

            LCD_DisplayState ();

            /* Start scan all sensors */
            CapSense_1_ScanEnabledWidgets();
        }
    }
}


/*******************************************************
* Display on LCD Buttons state
*******************************************************/
void LCD_DisplayState(void)
{
    LCD_Char_1_Position(1u, 0u);
    if (CapSense_1_CheckIsWidgetActive(CapSense_1_BUTTON0__BTN)) {
        LCD_Char_1_PrintString("ON "); }
    else {
        LCD_Char_1_PrintString("OFF"); }

    LCD_Char_1_Position(1u, 4u);
    if (CapSense_1_CheckIsWidgetActive(CapSense_1_BUTTON1__BTN)) {
        LCD_Char_1_PrintString("ON "); }
    else {
        LCD_Char_1_PrintString("OFF"); }
}
```

Pin Assignments

The CapSense customizer generates a pin alias names for each of the CapSense sensors and support signals. These aliases are used to assign sensors and signals to physical pins on the PSoC chip. Assign 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 chip is divided into two halves – left and right. Even port number pins are on the left side of the chip and odd port number pins are on the right side.

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

In parallel sensing applications the CapSense component is capable of performing two simultaneous scans on two 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 chip. 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 obly be connected to the same side of the chip as Cmod. **The Pin Editor does not verify correct pin assignment**.

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.

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 chip. The Cmod can be connected to **any pin**, but for direct connection (do not overuse routing resources) use pins:

- 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 maximum SNR. A value of 5.6 – 10 nF gives good results in the most cases.

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

When **Current Source** is set to External Resistor, the external Rb feedback resistor value should be selected before experimenting to determine 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 modulator capacitor (Cmod).

Up to three bleed resistors are supported. 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 counts are increased when resistor values increase.

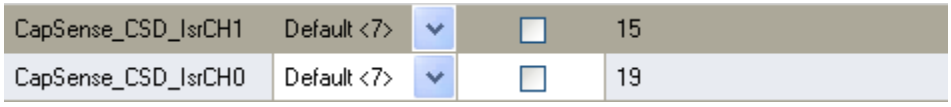
Typical values are 500Ω – 10 kΩ depending on sensor capacitance.

Interrupt Service Routines

The CapSense component uses an interrupt that triggers after end of sensor scan. Stub routines are provided where you can add your own code. 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: one or two, depends on Number of Channels. Your code must be added between the provided comment tags.

Two Channel mode ISR priority set

The ISRs routines of CapSense CSD component are not reentrant. This cause restriction of ISR priority set for two channels designs. To prevent ISR routines reentrant the ISR priority of two channels should be same.



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, and Silver ink on PET.

Scan Sensor

A scan sensor 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. Some examples of CapSense Widgets include buttons, sliders, radial sliders, touch pads and 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 tracks and filters the noise. If the measured count value exceeds the noise threshold, the baseline algorithm stops updating.

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.

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 applications for sliding 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, 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 a noise threshold. When the strongest signal is found, this signal and those contiguous signals larger than the noise threshold are used to compute a centroid. As few as two and as many as (typically) eight sensors are used to calculate the centroid in the form of:

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

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 calculated 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 CSD Wizard. A scaling value is calculated by the wizard and stored as fractional values.

The multiplier for the centroid resolution is contained in three bytes with these bit definitions:

Resolution Multiplier MSB								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Multiplier	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
Resolution Multiplier ISB								
Multiplier	128	64	32	18	16	8	4	2
Resolution Multiplier LSB								
Multiplier	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256

The resolution is found by using this equation:

Resolution = (Number of Sensors - 1) x Multiplier

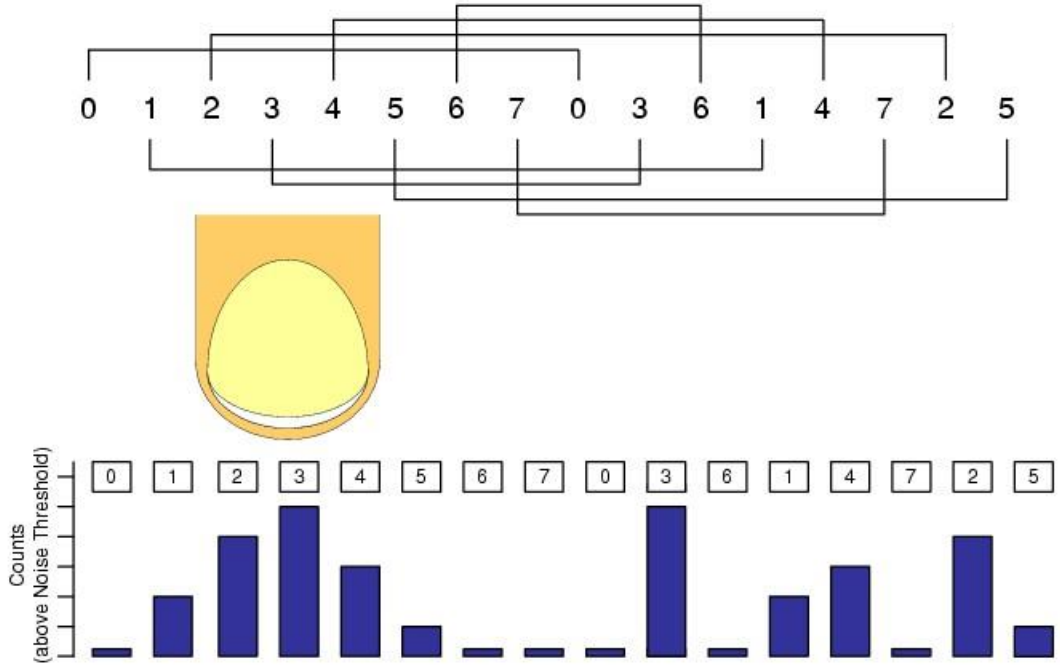
The centroid is held in a 24-bit unsigned integer and its resolution is a function of the number of sensors and the multiplier.

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.

There are a number of methods to order the second half of the physical sensor locations. The simplest is to index the sensors in the upper half, all of the even sensors, followed by all of the odd sensors. Other methods include indexing by other values. The method selected for this component is to index by three.

Figure 1 Diplexing, Index by Three



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.

Diplexing Sequence for Different Slider Segment Counts

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

Total Slider Segment Count	Segment Sequence
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
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

Total Slider Segment Count	Segment Sequence
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 are divided into two categories: raw data and position.

Position Median Filter

The median filter looks at the three most recent samples of position and reports the median value. It is used to remove short noise spikes. This filter generates a delay of one sample. This filter is generally not recommended because of the delay and RAM use. Enabling this filter consumes TBD Flash and 4 bytes of RAM. It is disabled by default.

Position Averaging Filter

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

Position First Order IIR Filter

The first oder IIR filter looks at the two most recent samples of position and add them with defined coefficients. Enabling this filter consumes TBD Flash and 2 bytes of RAM. The IIR1/4 is enabled by default.

1st-Order IIR filters:

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

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

Position Jitter Filter

This filter eliminates noise in the position that toggles between two values (jitter). It is most effective when applied to data that contains noise of four LSBs peak-to-peak or less. Enabling this filter consumes TBD Flash and 2 byte of RAM. It is disabled by default.

Raw Data Median Filter

The median filter looks at the three most recent samples from a sensor and reports the median value. It is used to remove short noise spikes. This filter generates a delay of one sample. This filter is generally not recommended because of the delay and RAM use. Enabling this filter consumes TBD Flash and (Number of Sensors × 4) bytes of RAM. It is disabled by default.

Raw Data Averaging Filter

The averaging filter looks at the three most recent samples from a sensor and reports the averaging value. It is used to remove short noise spikes. This filter generates a delay of one sample. This filter is generally not recommended because of the delay and RAM use. Enabling this filter consumes TBD Flash and (Number of Sensors × 4) bytes of RAM. It is disabled by default.

Raw First Order IIR Filter

The first oder IIR filter looks at the two most recent samples from a sensor and add them with defined coefficients. Enabling this filter consumes TBD Flash and (Number of Sensors × 2) bytes of RAM. The IIR1/4 is enabled by default.

1st-Order IIR filters:

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

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

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

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

Raw Data Jitter Filter

This filter eliminates noise in the the raw data that toggles between two values (jitter). It is most effective when applied to data that contains noise of four LSBs peak-to-peak or less. Enabling this filter consumes TBD Flash and (Number of Sensors × 2) bytes of RAM. It is disabled by default.

Water Influence on CapSense System

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

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

- Forming of thin stripes of water on the device surface.
- Separate drops of water.
- Stream of water, when the device is being washed or dipped.

Salts or minerals that such water contains make it conductive. Moreover, the more their concentration is the more conductive the water is. Soap water, sea water, mineral water can be referred to the liquids that influence the CapSense badly. Such liquid emulates a finger touch on the surface, which cause the device faulty performance.

Water proofing and detection

This feature configures the CapSense CSD to suppress water influence on CapSense system. This feature sets the following parameters:

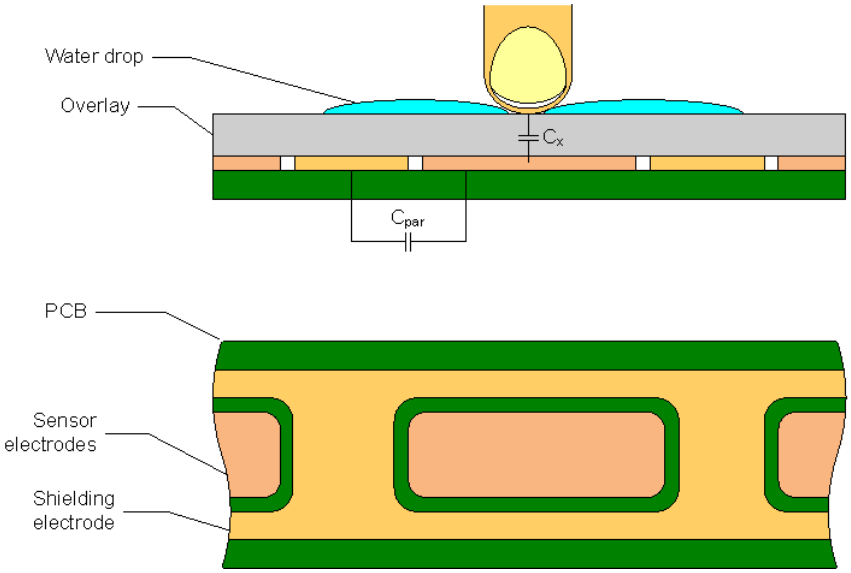
- Enables Shield electrode to be used to compensate the water drops influence on the sensor at the hardware level.
- Adds Guard sensor. The special shape should be designed for this sensor. The CapSense performance should be blocked programmatically when Guard sensor triggers.

Shielding Electrode

Some applications require reliable operation in the presence of water films 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. In this case a separate shielding electrode can be used. This electrode is located behind or outside the sensing electrode. When water films are located on the device insulation overlay surface, the coupling between the shielding and sensing electrodes is increased. The shielding 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 shielding electrode signal and its placement relative to the sensing electrode such that increasing the coupling between these electrodes causes the opposite of the touch change of the sensing electrode capacitance measurement. This simplifies the high level software API work. The CapSense CSD supports separate output for the shielding electrode.

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 influence 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 shielding and sensing electrodes, the Cpar is increased and modulator current can be reduced. In practical tests, the modulator reference voltage can be increased by the API so that the raw count increase from water drops should be close to zero or be slightly negative.

The shield electrode can be connected to any pins. Set the drive mode to Strong Slow to reduce ground noise and radiated emissions. Also, the 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 sensor alternates 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 eliminate signal from sensor. The possible solutions are:

- Use high Vref to eliminate difference to minimal value. The VDAC as reference could be used for this purposes.
- Use SIO pin as shield to provide output equal to Vref. The CapSense CSD output Vref terminal could be used.

Note The Inactive Sensor Connection to Shield should not be used in this mode because it provides the ouput equal to Vddio.

Note The Vref = 1.024 V has routing limitation and could not be routed to pin explicit.

Current Mode 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 so significant in this case.

Guard Sensor Implementation

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

The option on Advanced Tab is provided to add such type of sensor. This sensor has to have the special layout, typically located around the perimeter of 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 CapSense performance should blocked programmatically for a certain period of time when Guard sensor triggers.

Taking into consideration the Guard sensor's size, its signal will differ from other sensors' signals. This means a bigger amount of water may come on its surface than on a usual sensor's surface. Therefore, the signal received at the water drops presence will be much stronger than the signal caused by a finger touch. This allows setting the trigger threshold and filtering a finger's touch on the Guard sensor. So, its triggering will signal that a large amount of water has come on the sensing area surface.

Note The Guard sensor scans without any special options. The shield electrode (which typically used in water proof designs) doesn't disable while Guard sensor scanning.

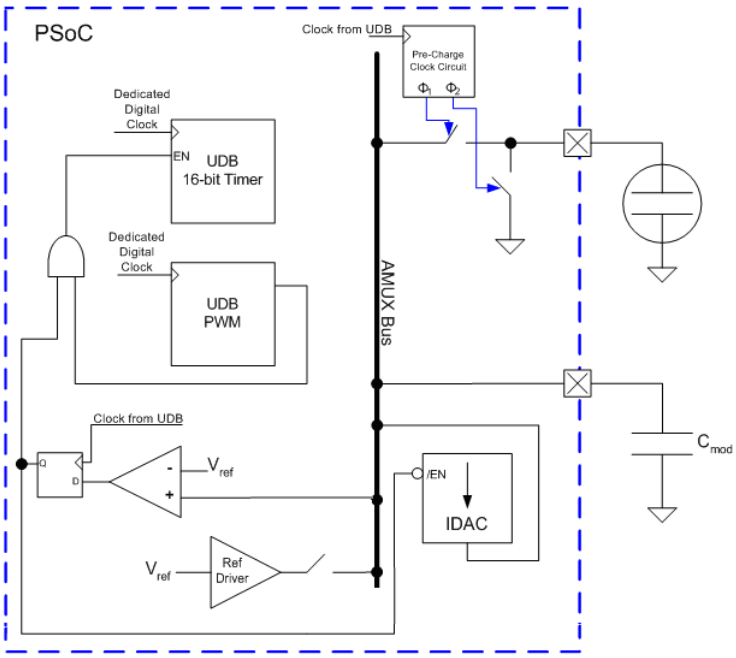
Note The Guard sensor in two channel designs always scans last and alone (without pair).

Block Diagram and Configuration

The CSD (Capacitive Sensing using a Delta-Sigma Modulator) provides capacitance sensing using the switched capacitor technique with a delta-sigma modulator to convert the sensing switched capacitor current to digital code. It allows implementation of buttons, sliders, 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.

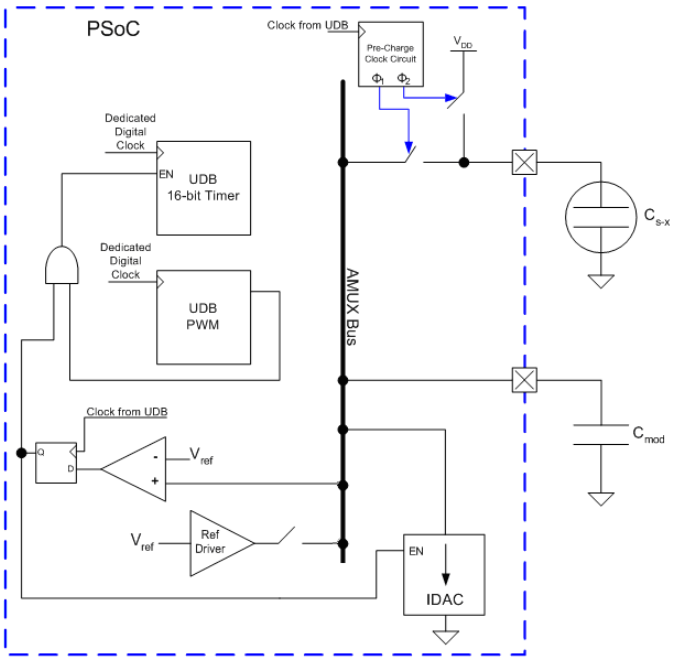
IDAC sourcing

The switch stage is reconfigured to alternate between GND and AMUX bus. In this configuration, the IDAC is configured to source current to AMUX bus.



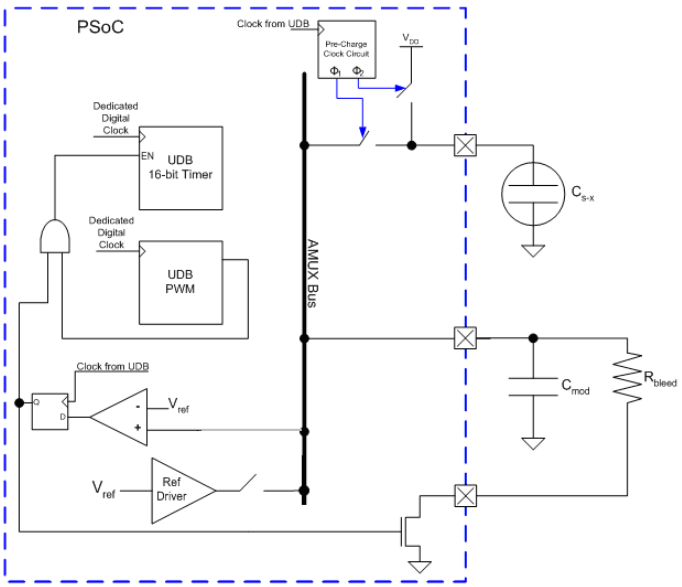
IDAC Sinking

The switch stage is reconfigured to alternate between Vdd and AMUX bus. In this configuration, the IDAC is configured to sink current instead of sourcing current.



IDAC disable, use external R_b

Same as the IDAC (Sinking) configuration except the IDAC is replaced by 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 drive mode allows C_{mod} to be discharged through R_b.



DC and AC Electrical Characteristics

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		67	MHz	

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
2.0.b	Minor datasheet edits.	
2.0.a	Minor datasheet edits.	
2.0	The CapSense CSD component version 2.0 is a completely re-designed implementation of the CapSense component version 1.30.	Version 2.0 offers many improvements over version 1.30. However, the interfaces and implementations are very different between these versions. Therefore, you cannot use the Component Update Tool to update existing designs to version 2.0. Instead, you must manually replace the component and reconfigure your design.

© Cypress Semiconductor Corporation, 2009-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

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

PRELIMINARY