ModusToolbox™

CAPSENSE™ Configurator guide

About this document

Version

4.0

Scope and purpose

The CAPSENSE™ Configurator is used to create and configure CAPSENSE™ widgets, and generate code to control the application firmware.

Intended audience

This document helps application developers understand how to use the CAPSENSE™ Configurator as part of creating a ModusToolbox™ application.

Document conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, menus and sub-menus
Italics	Denotes file names and paths.
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
File > New	Indicates that a cascading sub-menu opens when you select a menu item

Abbreviations and definitions

The following define the abbreviations and terms used in this document:

- Application – One or more projects related to each other.
- CAPSENSE – capacitive sensing
- Configurator – A GUI-based tool used to configure a resource.
- CSD – self-capacitance sensing method
- CSD HW – CAPSENSE™ Sigma Delta hardware block
- CSX – CAPSENSE™ Transmit/Receive (CAPSENSE™ with two electrodes: Tx and Rx), the mutual capacitance sensing method
- MSC HW – multi sensing converter – A hardware block, the latest hardware block, which supports the CSX and CSD sensing methods.
- Peripheral – any external analog or digital device that provides an input and output for the computer

Reference documents

Refer to the following documents for more information as needed:

- Eclipse IDE for ModusToolbox™ user guide
- Device Configurator guide

- CAPSENSE™ Tuner guide
- CAPSENSE™ Middleware API reference guide
- Device datasheets
- Device technical reference manuals

Table of contents

1 Overview 4

1.1 Supported middleware 4

2 Launch the CAPSENSE™ Configurator 5

2.1 From the Device Configurator 5
2.2 make command 6
2.3 Eclipse IDE 6
2.4 Executable (GUI) 6
2.5 Executable (CLI) 7

3 Quick start 8

4 Code generation 9

5 GUI description 10

5.1 Menus 10
5.2 Notice List 11

6 Tabs 12

6.1 Basic tab 13
6.2 Advanced tab 18
6.3 Pins tab *[CSD HW]* 43
6.4 Scan Configuration tab *[MSC HW]* 43

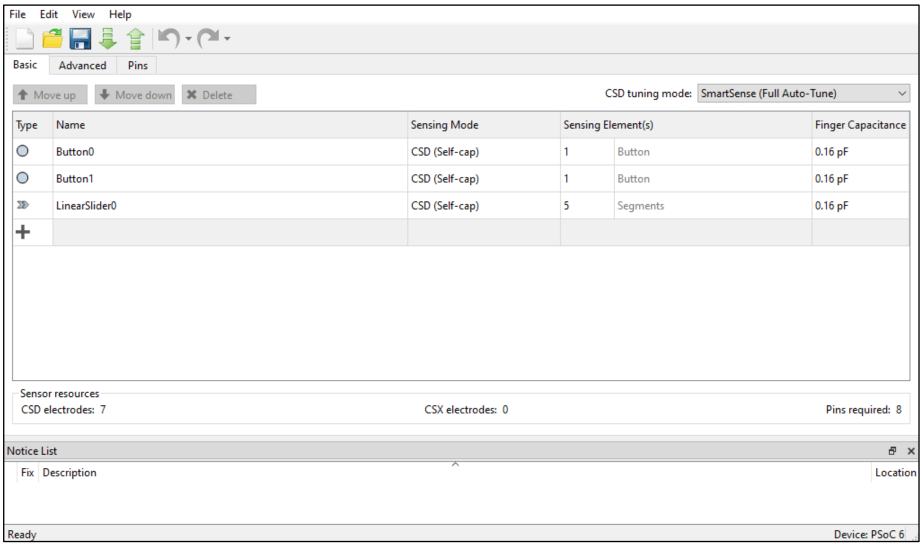
7 Version changes 46

1 Overview

CAPSENSE™ is our capacitive sensing solution used in a variety of applications and products where sleek human interfaces replace conventional mechanical buttons to transform the way users interact with electronic systems. These include home appliances, automotive, IoT, and industrial applications. CAPSENSE™ supports multiple interfaces (widgets) using both CSX and CSD sensing methods, with robust performance.

The CAPSENSE™ Configurator is part of a collection of tools included with ModusToolbox™. Use it to create and configure CAPSENSE™ widgets, and generate code to control the application firmware. There is a separate CAPSENSE™ Tuner application for tuning, testing, and debugging, for easy and smooth design of human interfaces on customer products.

The CAPSENSE™ Configurator supports all PSoC™ 6 and PSoC™ 4 families with the CSD hardware (HW) block, and PSoC™ 4 Max family with the MSC HW block. Both blocks – *[CSD HW]* and *[MSC HW]* sections respectively – are configured differently.



1.1 Supported middleware

Name	Version	Link
CAPSENSE™ Middleware Library	2.0, 2.10, 3.0	https://github.com/Infineon/capsense

2 Launch the CAPSENSE™ Configurator

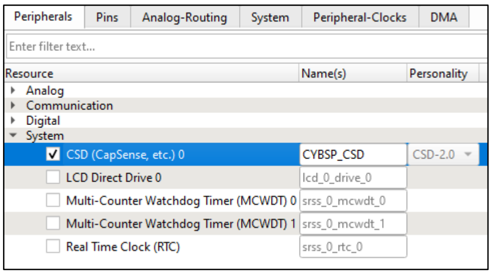
There are several ways to launch the CAPSENSE™ Configurator, and those ways depend on how you use the various tools in ModusToolbox™. However, the easiest way is to launch it using the Device Configurator because you can configure the rest of the parameters for your application right there. Refer to the Device Configurator Guide for more details.

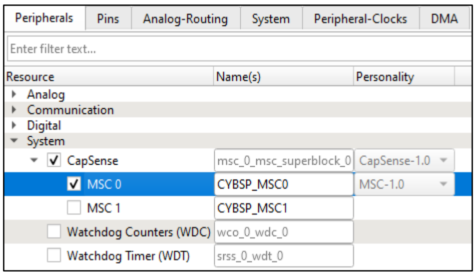
2.1 From the Device Configurator

You can launch the CAPSENSE™ Configurator by using the Device Configurator. The Device Configurator displays information based on the *design.modus* file. When you open the CAPSENSE™ Configurator from the Device Configurator, information about the device and the application is passed to the CAPSENSE™ Configurator. When you save changes in the CAPSENSE™ Configurator, it updates/generates a *design.cycapsense* configuration file in the same location as the *design.modus* file. For information about how to launch the Device Configurator, refer to the Device Configurator guide.

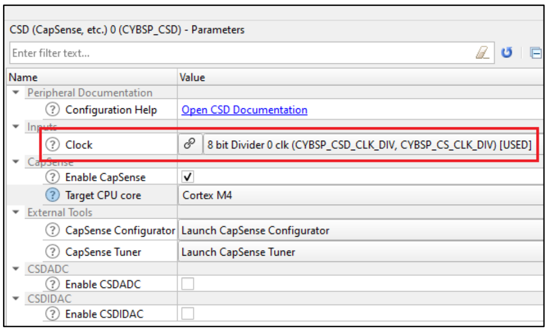
The process to launch the CAPSENSE™ Configurator from the Device Configurator differs slightly for device families with the CSD HW block and those with the MSC HW block. There is only one resource for the CSD HW block, while there is more than one resource for the MSC HW block.

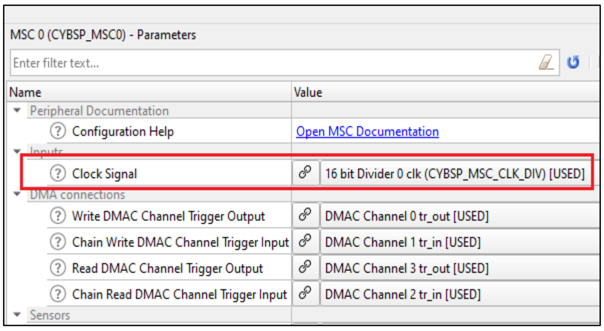
1. On the **Peripherals** tab, select the **CSD (CapSense)** resource or the **CapSense** and one or more **MSC** resources, as applicable for your device.



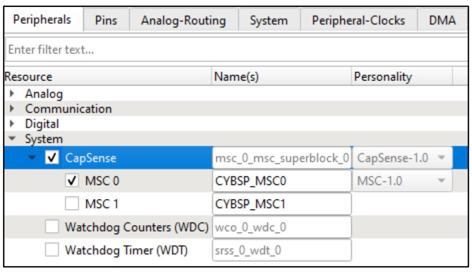


2. On the **Parameters** pane, select an appropriate input **Clock**.

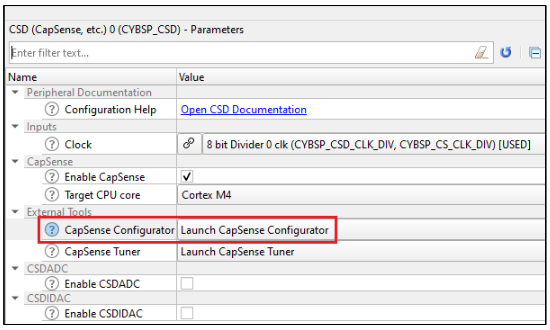


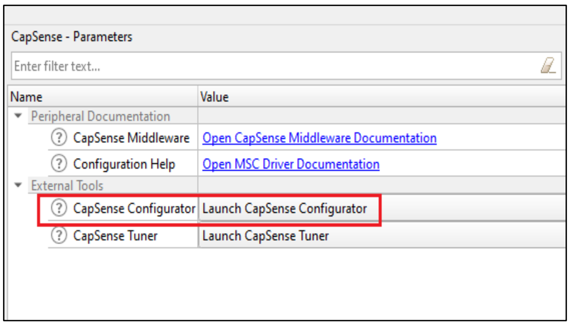


3. For the MSC HW block only, select the **CapSense** resource category on the **Peripherals** tab.



4. On the Parameters pane, click the **Launch CapSense Configurator** button.





2.2 make command

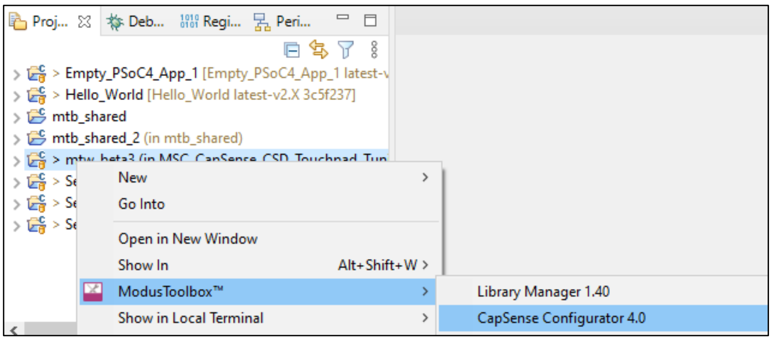
As described in the [ModusToolbox™ user guide](#) Build System chapter, you can run numerous make commands in the application directory, such as launching the CAPSENSE™ Configurator. After you have created a ModusToolbox™ application, navigate to the application directory and type the following command in the appropriate bash terminal window:

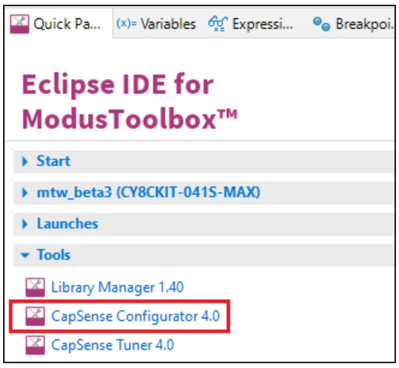
```
make open CY_OPEN_TYPE=capsense-configurator
```

This command opens the CAPSENSE™ Configurator GUI for the specific application in which you are working.

2.3 Eclipse IDE

If you use the Eclipse IDE for ModusToolbox™, you can launch the CAPSENSE™ Configurator for the selected application. In the Project Explorer, right-click on the project and select **ModusToolbox™ > CapSense Configurator <version>**. You can also click the CAPSENSE™ Configurator link in the IDE Quick Panel.





2.4 Executable (GUI)

If you don't have an application or if you just want to see what the configurator looks like, you can launch the CAPSENSE™ Configurator GUI by running its executable as appropriate for your operating system (for example, double-click it or select it using the Windows **Start** menu). By default, it is installed here:

<install_dir>/ModusToolbox/tools_<version>/capsense-configurator<version>

When opened this way, the CAPSENSE™ Configurator GUI opens without any information. You must open an existing **.cycapsense* file or create a new one for the application in which you want to configure CAPSENSE™.

Note: *Opening an existing or creating a new *.cycapsense file requires the design.modus file in the same directory.*

2.5 Executable (CLI)

You can run the capsense-configurator executable from the command line. There is also a capsense-configurator-cli executable, which re-generates source code based on the latest configuration settings from a command-line prompt or from within batch files or shell scripts. The exit code for the capsense-configurator-cli executable is zero if the operation is successful, or non-zero if the operation encounters an error. In order to use the capsense-configurator-cli executable, you must provide at least the `--config` argument with a path to the configuration file.

For details about command-line options, run the capsense-configurator or capsense-configurator-cli executable using the `-h` option.

3 Quick start

This section provides a simple workflow for how to use the CAPSENSE™ Configurator.

1. Create a CAPSENSE™ application from the Eclipse IDE, which provides simple CAPSENSE™ Buttons and Slider [CSD HW] and MSC CAPSENSE™ Button Tuning [MSC HW] examples to get started.
2. Launch the Device Configurator. Refer to the Device Configurator guide.
3. Enable and configure a communication peripheral. The examples use an SCB configured as EZI2C.
4. Launch the CAPSENSE™ Configurator.
5. Add and configure widgets on the Basic tab.
6. Configure parameters on the Advanced tab.
7. [CSD HW] Assign pins on the Pins tab.
8. [MSC HW] Assign channels, pins, and slots on the Scan Configuration tab.
9. Save to generate code.
10. Include *cycfg_capsense.h* in the *main.c* file.
11. Build the application in the Eclipse IDE, and program the device.
12. Launch the CAPSENSE™ Tuner.

4 Code generation

The CAPSENSE™ Configurator generates .c and .h files into directory *GeneratedSource* next to the **. cycapsense* file, which contains the user configuration. These files – *cycfg_capsense.h, cycfg_capsense.c, cycfg_capsense_defines.h, cycfg_capsense_tuner_regmap.h* – contain relevant firmware used for the CAPSENSE™ Middleware configuration and operation. The directory contains the necessary source (.c) and header (.h) files for the generated firmware, which uses the relevant driver APIs to configure the hardware.

The file *cycfg_capsense_defines.h* is required for the CAPSENSE™ Middleware to build successfully. It should be located in the *GeneratedSource* directory when the CAPSENSE™ Middleware is included in the application.

The tool generates code every time you save the configuration file. Code can be generated during the build of an application if it is missing or out of date.

5 GUI description

5.1 Menus

5.1.1 File

- **New** – Creates a new file with new configuration.
- **Open…** – Opens a specified <file_name>.*cycapsense* configuration file. The current file, if any, will be closed.
- **Close** – Closes the current file.
- **Save** – Saves the current configuration file and generates CAPSENSE™ middleware configuration code. If there are errors in the application, a dialog will indicate such.
- **Open in System Explorer** – This opens your computer's file explorer tool to the folder that contains the *.cycapsense file.
- **Import…** – Imports a specified configuration file.
- **Export…** – Exports the current configuration file into a specified file.
- **Export Register Map to PDF** – Exports the current configuration register map in PDF format.
- **Recent files** – Shows recent files that you can open directly.
- **Exit** – Closes the configurator. You will be prompted to save any pending changes.

5.1.2 Edit

- **Undo** – Undoes the last action or sequence of actions.
- **Redo** – Redoes the last undone action or sequence of undone actions.

5.1.3 View

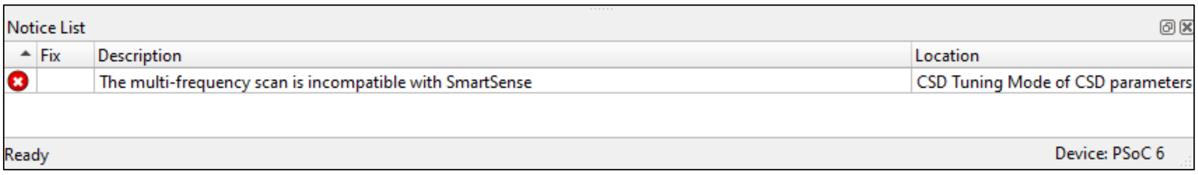
- **Notice List** – Hides or shows the Notice List pane. The pane is shown by default.
- **Toolbar** – Hides or shows the Toolbar.
- **Reset View** – Resets the view to the default.

5.1.4 Help

- **View Help** – Opens this document.
- **About CapSense Configurator** – Opens the About box for version information.

5.2 Notice List

The Notice List pane combines notices (errors, warnings, tasks, and notes) from many places in the configuration into a centralized list. If a notice shows a location, you can double-click the entry to show the error or warning.



The Notice List pane contains the following columns:

- **Icon** – Displays the icons for the error, warning, task, or note.
- **Fix** – This may display a wrench icon, which can be used to automatically address the required notice.
- **Description** – Displays a brief description of the notice.
- **Location** – Displays the specific tab of the message, when applicable.

6 Tabs

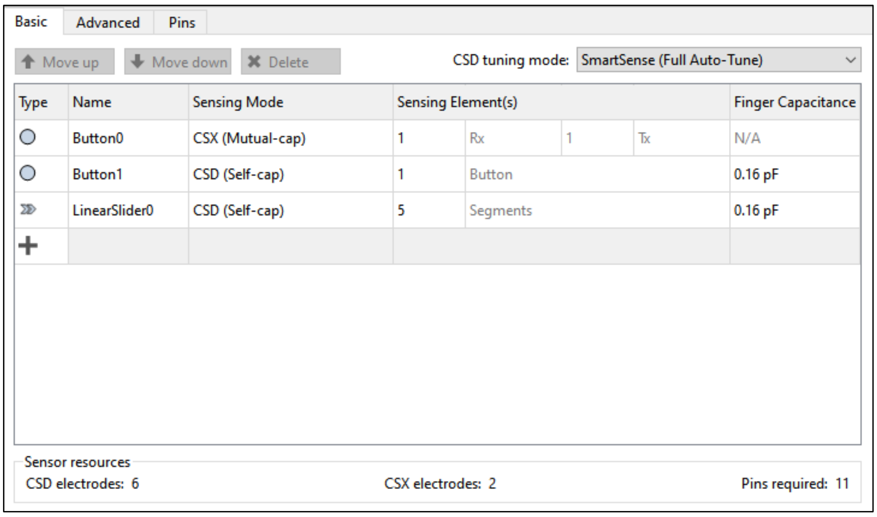
The CAPSENSE™ Configurator contains the following tabs, each of which provides access to specific parameters. Separate sections in this document provide more descriptions of these tabs.

- Basic tab
- Advanced tab
- Pins tab [CSD HW]
- Scan configuration tab [MSC HW]

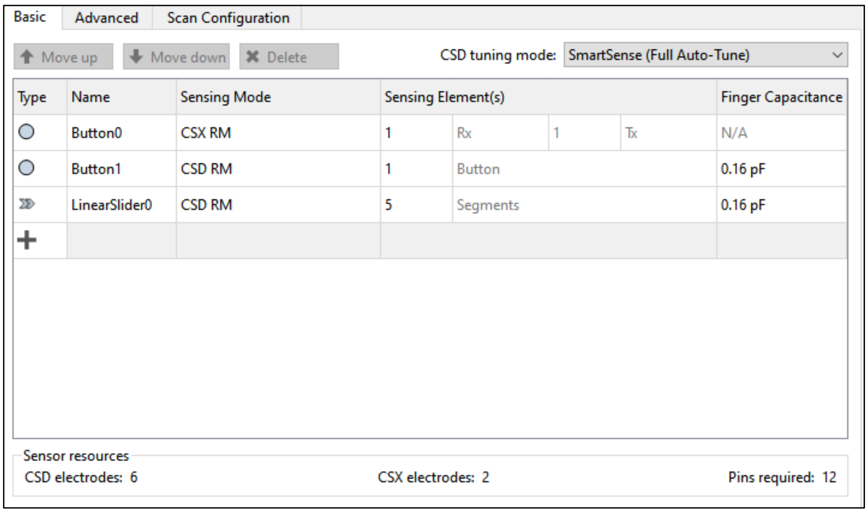
6.1 Basic tab

The **Basic** tab defines the high-level middleware configuration. Use this tab to add various *Widget Type* and assign *Sensing mode*, *Widget Sensing Element(s)* and *Finger capacitance* for each widget.

CSD HW:



MSC HW:



The following table contains descriptions of the various **Basic** tab parameters:

Name	Description
CSD tuning mode *[CSD HW]*	Tuning is a process of finding appropriate values for configurable parameters (Hardware parameters and Threshold parameters) for proper functionality and optimized performance of the CAPSENSE™ system. The SmartSense Auto-tuning is an algorithm embedded in the CAPSENSE™ middleware. The algorithm automatically finds optimum values for configurable parameters basing on the hardware properties of capacitive sensors. This allows the user to avoid the manual-tuning process. Configurable parameters that affect the operation of the sensing hardware are called Hardware parameters. Parameters that affect the operation of the touch-detection firmware algorithm are called Threshold parameters. This parameter is a drop-down box to select the tuning mode for the CSD widgets. **SmartSense (Full Auto-Tune)** – This is the quickest way to tune a design. Most hardware and threshold parameters are automatically tuned by the middleware and the configurator GUI displays them as *Set by SmartSense* mode. In this mode, the following parameters are automatically tuned: *CSD* Settings tab: *Enable IDAC auto-calibration* *Widget* Details tab: The CSD-related parameters of the groups *Widget hardware parameters* and *Widget threshold parameters* *Widget* Details tab: the *Compensation IDAC value* parameter if *Enable compensation IDAC* is set. **SmartSense (Hardware parameters only)** – The Hardware parameters are automatically set by the middleware. The Threshold parameters are set manually by the user. This mode consumes less memory and less CPU processing time, this leads to consuming lower average power. In this mode, the following parameters are automatically tuned: *CSD* Settings subtab: *Enable IDAC auto-calibration*. *Widget* Details subtab: The CSD-related parameters of the *Widget hardware parameters* group*;* *Compensation IDAC* value parameter if *Enable compensation IDAC* is set. **Manual** – The SmartSense auto-tuning is disabled, the *Widget hardware parameters* and *Widget threshold parameters* are tuned manually. The lowest memory and CPU process-time consumption. The SmartSense Auto-tuning (both Full Auto-Tune and Hardware parameters only) supports the *IDAC Sourcing* configuration only. *SmartSense Auto-tuning* requires Modulator clock frequency set to 6000 kHz or higher. SmartSense operating conditions: Sensor capacitance Cp range 5 pF to 61 pF Maximum external series resistance on a sensor Rext < 1.1 kOhm.

Name	Description
CSD tuning mode *[MSC HW]*	This parameter is a drop-down box to select the tuning mode for the CSD widgets. **SmartSense (Full Auto-Tune)** – This is the quickest way to tune a design. Most hardware and threshold parameters are tuned automatically by the middleware and the configurator GUI displays them as *Set by SmartSense* mode. In this mode, the following parameters are tuned automatically: *CSD* Settings tab: *Enable CDAC auto-calibration* *Widget* Details subtab: The CSD-related parameters of the groups *Widget hardware parameters* and *Widget threshold parameters* *Widget* Details subtab: the *Compensation CDAC value* parameter if *Enable compensation CDAC* is set. **SmartSense (Hardware parameters only)** – The Hardware parameters are automatically set by the middleware. The Threshold parameters are set manually by the user. This mode consumes less memory and less CPU processing time, this leads to consuming lower average power. In this mode, the following parameters are automatically tuned: *CSD* Settings subtab: *Enable CDAC auto-calibration.* *Widget* Details subtab: The CSD-related parameters of the *Widget hardware parameters* group*;* *Compensation CDAC value* parameter if *Enable compensation CDAC* is set. **Manual** – The SmartSense auto-tuning is disabled, the *Widget hardware parameters* and *Widget threshold parameters* are tuned manually. The lowest memory and CPU process-time consumption.

Name	Description
Widget Type	A widget is one sensor or a group of sensors that perform a specific user-interface functionality. The following widgets types consist: **Button** – One or more sensors. Each sensor in the widget can detect the presence or absence (i.e. only two states) of a finger on the sensor. **Linear Slider** – More than one sensor arranged in the specific order to detect the presence and movement of a finger on a linear axis. If a finger is present, Linear Slider detects the physical position (single axis position) of the finger. **Radial Slider** – More than one sensor arranged in the circular order to detect the presence and radial movement of a finger. If a finger is present, the Radial Slider detects the physical position of the finger. **Matrix Buttons** – Two or more sensors arranged in the specific horizontal and vertical order to detect the presence or absence of a finger on the intersections of vertically and horizontally arranged sensors. If M and N are the numbers of the sensors in the horizontal and vertical axis respectively, the total of the M x N intersection positions can detect a finger touch. When using the *CSD sensing method*, a simultaneous finger touch on more than one intersection is invalid and produces invalid results. This limitation does not apply when using the *CSX sensing method* and all intersections can detect a valid touch simultaneously. **Touchpad** – Multiple sensors arranged in the specific horizontal and vertical order to detect the presence or absence of a human finger. If a finger is present, the widget will detect the physical position (both X and Y axis position) of the touch. The *CSD sensing method* supports detection of up to 2 simultaneous touches (when Advanced Centroid is enabled). The *CSX sensing method* supports detection of up to 3 simultaneous finger touches. **Proximity Sensor** – One or more sensors. Each sensor in the widget can detect the proximity of conductive objects, such as a human hand or finger to the sensors. The proximity sensor has two thresholds: *Proximity threshold* – To detect an approaching hand or finger. *Touch threshold* – To detect a finger touch on the sensor.
Widget Name	A widget name can be defined to aid in referring to a specific widget in a design. A widget name does not affect functionality or performance. A widget name is used throughout source code to generate macro definitions. A maximum of 255 alphanumeric characters (the first letter must be an alphabetic character) is acceptable for a widget name.
Sensing mode	The parameter to select the sensing mode for each widget: **CSD** sensing method (Capacitive Sigma Delta) – A Cypress patented method of performing self-capacitance measurement; supported by all widget types. **CSX** sensing method – A Cypress patented method of performing mutual-capacitance measurement; supported by the Button, Linear Slider, Matrix Buttons, and Touchpad widgets types. Note that the CSX Linear Slider is supported only by CAPSENSE™ Middleware 3.0 and later.

Name	**Description**
Widget Sensing Element(s)	A sensing element refers to the sensing terminals assigned to port pins to connect to physical sensors on a user-interface panel (such as a pad or layer on a PCB, ITO, or FPCB). The following element numbers are supported by the *CSD sensing method*: *Button* – Supports 1 to 64 sensors within a widget. *Linear Slider* – Supports 3 (5 for diplexed) to 64 segments within a widget. *Radial Slider* – Supports 3 to 64 segments within a widget. *Matrix Buttons* – Support 2 to 64 rows and columns. *Touchpad* – Supports 3 to 64 rows and columns. *Proximity* –Supports 1 to 64 sensors within a widget. The following element numbers are supported by the *CSX sensing method*: *Button* – Supports 1 to 64 Rx electrodes (for 1 to 64 sensors) and Tx is fixed to 1. *Linear Slider* – Supports 3 (5 for diplexed) to 64 Rx electrodes and Tx is fixed to 1. *Matrix Buttons* – Supports 2 to 64 Tx and Rx. *Touchpad* – Supports 3 to 64 Tx and Rx. The total intersections (node) number is equal to Tx × Rx.
Finger capacitance	Finger capacitance is defined as capacitance introduced by a user touch on the sensors. This parameter is used to indicate how a sensitive CSD widget is tuned by the *SmartSense Auto-tuning* algorithm. The supported Finger capacitance range: *SmartSense (Full Auto-Tune)* mode – 0.1 pF to 1 pF. *SmartSense (Hardware parameters only)* mode – 0.02 pF to 20.48 pF on the exponential scale. CAPSENSE™ sensor sensitivity is inversely proportional to a finger capacitance value. A smaller value of finger capacitance provides higher sensitivity for a sensor. To detect a user touch on a thick overlay (4-mm plastic overlay), finger capacitance is set to a small value, e.g. 0.1 pF. For a sensor with a thin overlay or no overlay, the 0.1 pF finger capacitance setting makes the sensor too sensitive and may cause false touches. For the robust operation, it is important to set the appropriate finger capacitance value by considering the sensor size and overlay thickness of the design. Refer to the *CapSense design guide* for more information.
Move up / Move down	Moves the selected widget up or down by one on the list. It defines the widget scanning order.
Delete	Deletes the selected widget from the list.
CSD electrodes	Indicates the total number of electrodes (port pins) used by the CSD widgets.
CSX electrodes	Indicates the total number of electrodes (port pins) used by the CSX widgets.
Pins required	Indicates the total number of port pins required for the design. This does not include port pins used by other peripherals in the application or SWD pins in Debug mode. Pins required includes the number of CSD and CSX electrodes, Cmod, Csh, Shield, CintA and CintB electrodes.

6.2 Advanced tab

The **Advanced** tab provides advanced configuration parameters. In *SmartSense Auto-tuning,* most of the advanced parameters are automatically tuned by the algorithm and the user does not need to set values for these parameters by the *Manual* tuning process. When Manual tuning mode is selected, the **Advanced** tab allows the user to control and configure the CAPSENSE™ middleware parameters.

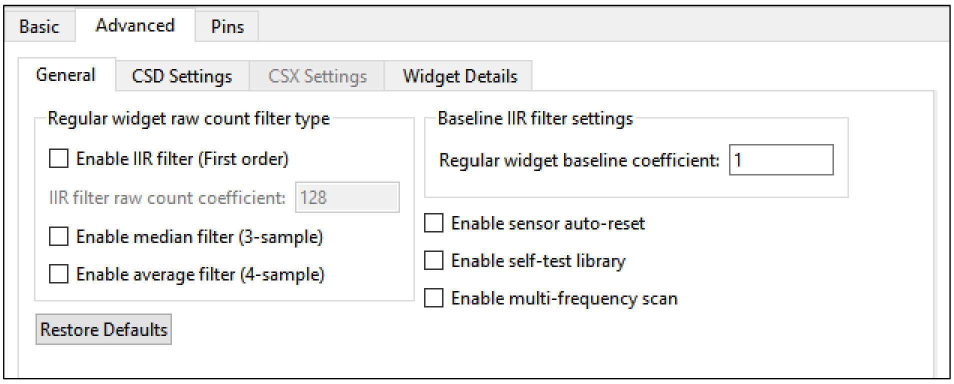
The parameters in the **Advanced** tab are systematically arranged in the following sub-tabs.

- *General* – Contains the parameters common for all widgets respective of the sensing method used for the widgets.
- *CSD* Settings – Contains the parameters common for all widgets using the CSD sensing method. This tab is relevant only if one or more widgets use the CSD sensing method.
- *CSX Settings* – Contains the parameters common for all widgets using the CSX sensing method. This tab is relevant only if one or more widgets use the CSX sensing method.
- *Widget* Details – Contains parameters specific to widgets and/or sensors.

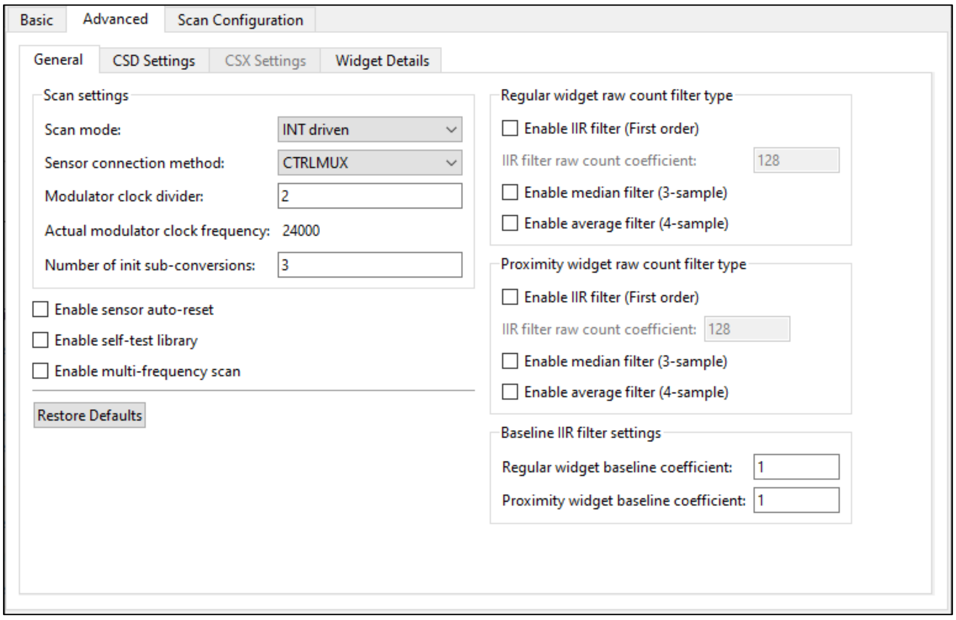
6.2.1 General subtab

Contains the parameters common for all widgets respective of *Sensing mode* used for widgets.

CSD HW:



MSC HW:



The **General** sub-tab contains the following sections:

6.2.1.1 Scan settings [MSC HW]

Name	**Description**
Scan mode *[MSC HW]*	Selects a sensor sequencing method. **INT driven** (default) – In Interrupt driven mode, the CPU extracts the results and programs the MSC HW for the next scan in the scope of the End of Scan interrupt servicing routine. **CS-DMA** – In Chained Scan DMA mode, the DMA functionality extracts the results and programs the MSC HW for the next scan. The CPU does not need to intervene between scans.
Sensor connection method *[MSC HW]*	Selects the method how to connect a sensor to the CAPSENSE™ HW block. **AMUXBUS** – In this mode, the CSD sensors, shield electrodes, and Rx electrodes are connected to the MSC HW using the analog bus and can be assigned to any GPIO which supports a connection with the analog bus for the particular device. **CTRLMUX** (default) – In this mode, the CSD sensors, shield electrodes, and Rx electrodes are connected to the MSC HW using the direct connection and can be assigned to the dedicated pads only.
Modulator clock divider *[MSC HW]*	Selects the modulator clock divider used for both CSD and CSX sensing methods **CSD** sensing method. This divider defines the operating frequency of the CSD and CSX blocks.
Actual modulator clock frequency (kHz) *[MSC HW]*	The modulator clock frequency depends on the MSC peripheral clock frequency and the modulator clock divider.
Number of init sub-conversions *[MSC HW]*	Selects the number of initialization sub-conversions at the start of the scan. This part of scan is intended to ensure proper initialization of CAPSENSE™ HW and does not perform the raw count measurement.

6.2.1.2 General settings

The general settings are applicable to the whole CAPSENSE™ middleware behavior.

Name	Description
Enable sensor auto-reset	When enabled, the baseline is always updated and when disabled, the baseline is updated only when the difference between the baseline and raw count is less than the noise threshold. When enabled, the feature prevents the sensors from permanently turning on when the raw count accidentally rises due to a large power supply voltage fluctuation or other spurious conditions.
Enable self-test library	The CAPSENSE™ middleware provides the **B**uilt-**I**n **S**elf-**T**est (BIST) library to support the design compliant with the safety-integrity level of Class B (IEC-60730) white goods and automotive, and design for manufacturing testing. The library includes a set of tests for board validation, middleware configuration, and operation. The feature includes safety functions to reduce the risk, validate boards at manufacturing, and verify the middleware operation at run-time. The BIST tests are classified into two categories: 1. Hardware Tests – To confirm the CSD HW (MSC HW) block and sensor hardware (external to chip) function correctly: Chip analog-routing verification Pin faults checking PCB-trace opens / shorts checking External capacitors and sensors capacitance measurement VDDA measurement. 2. FW Tests – To confirm the integrity of data used for decision-making on the sensor status: Global and widget specific configuration verification Sensor baseline duplication Sensor raw count and baseline are in the specified range. The middleware is responsible for running each test at start and run-time as required by the product requirements. *Note:* *If SmartSense (Full Auto-Tune) is enabled, the self-test library cannot be enabled.* *This option is supported by CAPSENSE™ Middleware 2.1 and later.*

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Enable multi-frequency scan (MFS)</td><td>
The MFS provides superior immunity against external noises and is suitable for applications subjected to harsh environments.
MFS implementation for CSD HW:
When the MFS is enabled, each sensor is scanned three times with three different sensor frequencies. The base frequency F0 (zero channel) is the nominal sensor frequency. The second F1 and the third F2 frequencies are obtained by increasing the sense clock-divider by 1 and by 2 correspondingly:
$$F_N = \frac{ModClk}{\frac{ModClk}{SnsClk} + N}$$
where:
N is a frequency channel 0, 1, or 2;
F_N is a scanning frequency;
ModClk is the CSD Modulator clock frequency or CSX Modulator clock frequency;
SnsClk is the CSD Sense clock frequency or CSX Tx clock frequency.
The SmartSense (Full Auto-Tune), SmartSense (Hardware parameters only) and the MFS features are mutually exclusive. If the SmartSense is enabled, MFS cannot be enabled.
Note: Enabling the MFS increases RAM usage by three times approximately.
Enabling the MFS increases the sensor scan duration by three times.
MFS implementation for MSC HW:
When MFS is enabled, the Configurator creates two supplementary widgets for each existing widget with the same properties but different CSD Sense clock divider or CSX Tx clock divider. The supplementary widgets have “_F1” and “F2” suffixes in their names. Their sensors are ganged with the main widget’s sensors. The slot assignment should be performed manually on the Scan Configuration tab [MSC HW].
The Enable multi-frequency scan checkbox is shown as checked when the MFS is enabled for all widgets, and partially checked when the MFS is enabled only for some widgets. You can enable or disable the MFS for a particular widget on the Widget Details subtab.
The SmartSense (Full Auto-Tune), SmartSense (Hardware parameters only) and the MFS features are mutually exclusive. If the SmartSense is enabled, MFS cannot be enabled.
</td></tr>
</table>

6.2.1.3 Regular widget raw count filter type

The Regular widget raw count filter type applies to raw counts of sensors belonging to non-proximity widgets. These parameters can be enabled only when one or more non-proximity widgets are added to the **Basic** tab. The filter algorithm is executed when any processing function is called by the application layer. When enabled, each filter consumes RAM to store a previous raw count (filter history). If multiple filters are enabled, the total filter history correspondingly increases so that the size of the total filter history is equal to a sum of all enabled filter histories.

Name	Description
Enable IIR filter (First order)	Enables the IIR filter (See equation below) with a step response similar to an RC low-pass filter, thereby passing the low-frequency signals (finger touch responses). $Output = \frac{N}{K} \times input + \frac{(K-N)}{K} \times previousOutput$ where: *K* is always 256. *N* is the IIR filter raw count coefficient selectable from 1 to 128 in the configurator. A lower N (set in the *IIR filter raw count coefficient* parameter) results in lower noise, but slows down the response. This filter eliminates high-frequency noise. Consumes 2 bytes of RAM per each sensor to store a previous raw count (filter history).
IIR filter raw count coefficient	The coefficient (N) of IIR filter for raw counts is explained in the *Enable IIR filter (First order)* parameter. The range of valid values: 1-128.
Enable median filter (3-sample)	Enables a non-linear filter that takes three of most recent samples and computes the median value. This filter eliminates spike noise typically caused by motors and switching power supplies. Consumes 4 bytes of RAM per each sensor to store a previous raw count (filter history).
Enable average filter (4-sample)	The finite-impulse response filter (no feedback) with equally weighted coefficients. It takes four of most recent samples and computes their average. Eliminates periodic noise (e.g. noise from AC mains). Consumes 6 bytes of RAM per each sensor to store a previous raw count (filter history).

Note: If multiple filters are enabled, the execution order is the following:
1. Median filter
2. IIR filter
3. Average filter

6.2.1.4 Proximity widget raw count filter type

The proximity widget raw count filter applies to raw counts of sensors belonging to the proximity widgets, these parameters can be enabled only when one or more proximity widgets are added on the *Basic tab*.

Parameter Name	Description
Enable IIR filter (First order)	The design of these parameters is the same as the parameters. The *Proximity* sensors require high-noise reduction. These dedicated parameters allow for setting the proximity filter configuration and behavior differently compared to other widgets.
IIR filter raw count coefficient	
Enable median filter (3-sample)	
Enable average filter (4-sample)	

6.2.1.5 Baseline filter settings

Baseline filter settings are applied to all sensors' baselines. But, filter coefficients for the proximity and regular widgets can be controlled independently from each other.

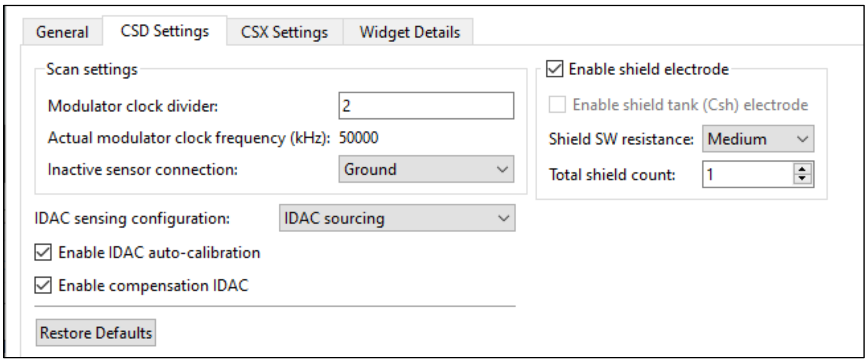
The design baseline IIR filter is the same as the raw count *Enable IIR filter (First order)* parameter. But, filter coefficients can be separate for both baseline filter and raw count filters to produce a different roll-off. The baseline filter is applied to a filtered raw count (if the widget raw count filters are enabled).

Name	Description
Regular widget baseline coefficient	Baseline IIR filter coefficient selection for sensors in non-proximity widgets. The range of valid values: 1-255.
Proximity widget baseline coefficient	The design of these parameters is the same as the *Regular widget baseline* coefficient, but with a dedicated parameter allows controlling the baseline update-rate of the proximity sensors differently compared to other widgets.

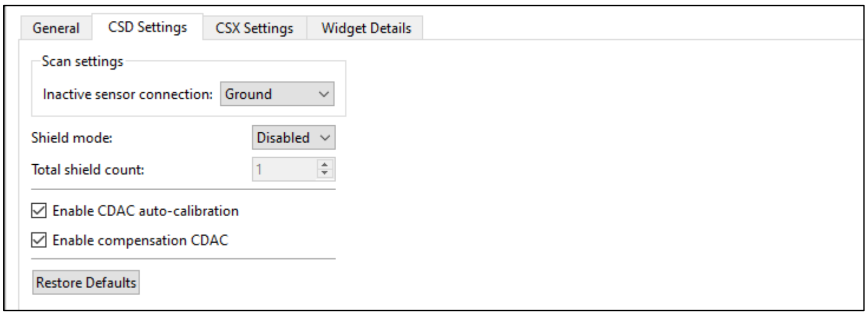
6.2.2 CSD Settings subtab

Contains the parameters common for all widgets using the *CSD sensing method*, is relevant only if at least one widget uses the CSD sensing method.

CSD *HW*:



MSC *HW*:



The **CSD Settings** subtab contains the following parameters:

Name	Description
Modulator clock divider *[CSD HW]*	Selects the modulator clock divider used for the *CSD sensing method.* It defines the operating frequency of the CSD block.

Name	Description
Actual modulator clock frequency (kHz) *[CSD HW]*	The modulator clock frequency depends on the CSD peripheral clock frequency and the modulator clock divider. The read-only value is displayed only when the CAPSENSE™ Configurator is launched from the Device Configurator.
Inactive sensor connection	Selects the state of the sensor when it is not scanned. **Ground (default)** – Inactive sensors are connected to the ground. **High-Z** – Inactive sensors are floating (not connected to GND or Shield). **Shield** - Inactive sensors are connected to Shield. This option is available only if the *Enable shield electrode* check box is set. **Ground** is the recommended selection for this parameter when water tolerance is not required for the design. Select Shield when the design needs water tolerance or to reduce the sensor parasitic capacitance in the design.
IDAC sensing configuration *[CSD HW]*	Selects the type of IDAC switching: **IDAC Sourcing (default)** – Sources current into the modulator capacitor (Cmod). The analog switches are configured to alternate between the Cmod and GND. IDAC Sourcing is recommended for most designs because of the better signal-to-noise ratio **IDAC sinking** – Sinks current from the modulator capacitor (Cmod). The analog switches are configured to alternate between V_{DD} and Cmod.
Enable IDAC auto-calibration *[CSD HW]*	When enabled, values of the CSD widget IDACs are automatically set by the middleware. Select the Enable IDAC Auto-calibration parameter for robust operation. The *SmartSense Auto-tuning* parameter can be enabled only when the Enable IDAC auto-calibration is selected.
Enable compensation IDAC *[CSD HW]*	The compensation IDAC is used to compensate for sensor parasitic capacitance to improve performance. Enabling the compensation IDAC is recommended unless one IDAC is required for general purpose (other than CAPSENSE™) in the application.
Enable CDAC auto-calibration *[MSC HW]*	When enabled, the values of the CSD widget CDACs are automatically set by the middleware. Select the Enable CDAC Auto-calibration parameter for robust operation.
Enable compensation CDAC *[MSC HW]*	The compensation CDAC is used to compensate for sensor parasitic capacitance to improve the performance. Select this parameter unless one CDAC is required for general purposes (other than CAPSENSE™) in the application.
Enable shield electrode *[CSD HW]*	The shield electrode is used to reduce the sensor parasitic capacitance, enable water-tolerant CAPSENSE™ designs and enhance the detection range for the *Proximity* sensors. When the shield electrode is disabled, configurable parameters associated with the shield electrode are hidden.
Enable shield tank (Csh) capacitor *[CSD HW]*	The shield tank capacitor is used to increase the drive capacity of the shield electrode driver. It should be enabled when the shield electrode capacitance is higher than 100 pF. The recommended value for a shield tank capacitor is 10nF/5V/X7R or an NP0 capacitor. The shield tank capacitor is not supported in configuration which includes both CSD and CSX sensing-based widgets.
Shield SW resistance *[CSD HW]*	Selects the resistance of switches used to drive the shield electrode. The four options: **Low**; **Medium** (default); **High**; **Low EMI**
Shield mode *[MSC HW]*	Selects the shield drive. The options: **Disabled** (default); **Active**; **Passive** When Shield mode is disabled, configurable parameters associated with it are not applicable.

Name	Description
Total shield count	Selects the number of shield electrodes required in the design. Most designs work with one dedicated shield electrode but, some designs require multiple dedicated shield electrodes to ease the PCB layout routing or to minimize the PCB area used for the shield layer. The minimum value is 0 (i.e. shield signal could be routed to sensors using the *Inactive sensor connection* parameter) and the maximum value is equal to the total number of CAPSENSE™-enabled port pins available for the selected device.

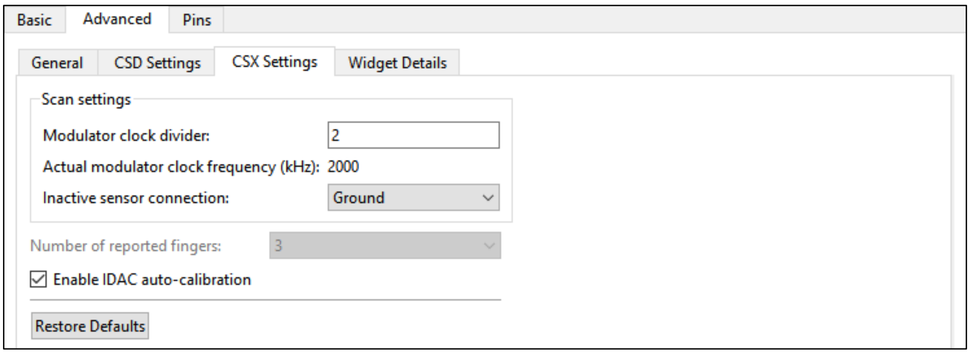
Commands:

- **Restore Defaults** – restores parameters values on the current tab to their default values.

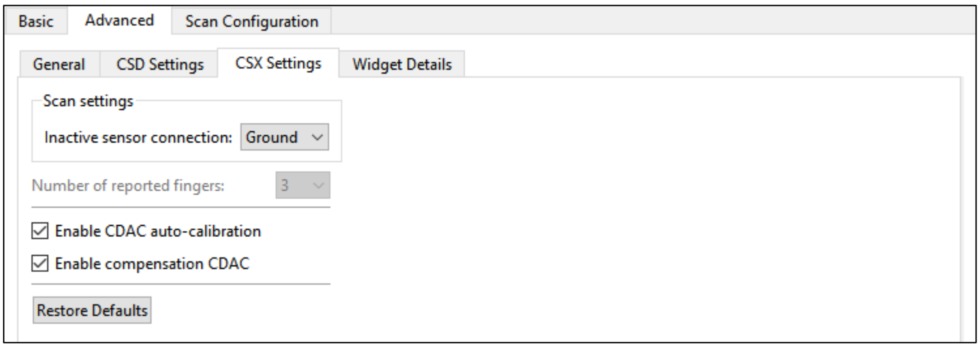
6.2.3 CSX Settings subtab

The parameters in this sub-tab apply to all widgets that use the *CSX sensing method*, is relevant only if at least one widget uses the CSX sensing method.

CSD HW:



MSC HW:



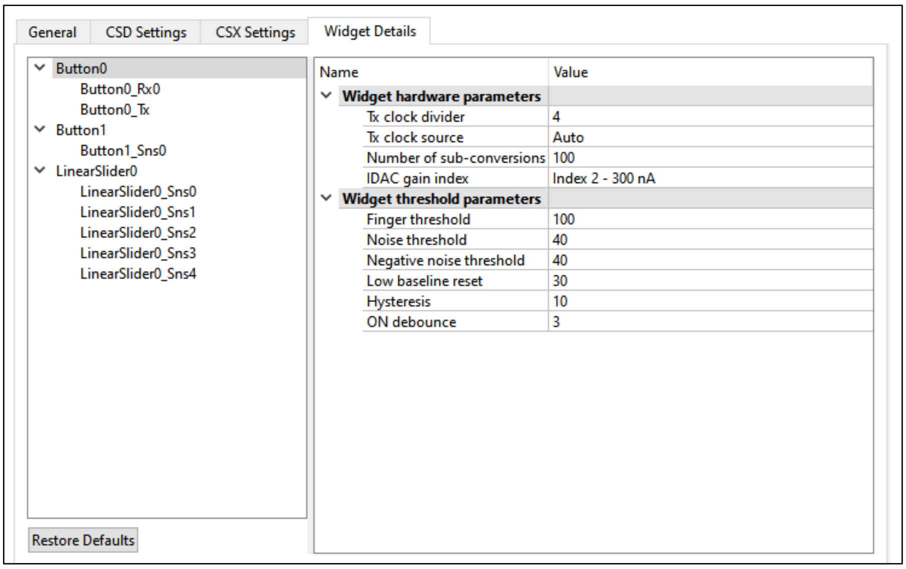
The **CSX Settings** subtab contains the following parameters:

Name	Description
Modulator clock divider [CSD HW]	Selects the modulator clock divider used for the *CSX sensing method*. It defines the operating frequency of the CSD block. A higher modulator clock frequency reduces the sensor scan time, results in lower power, and reduces the noise in raw counts, so use the highest possible frequency.
Actual modulator clock frequency (kHz) [CSD HW]	The modulator clock frequency depends on the CSX peripheral clock frequency and the modulator clock divider. The read-only value is displayed only when the CAPSENSE™ Configurator is launched from the Device Configurator.

Name	Description
Number of reported fingers	Sets the number of reported fingers for a CSX Touchpad widget only. The available options are from 1 to 3.
Inactive sensor connection	Selects the sensor state when it is not scanned. **Ground (default)** – Inactive sensors are connected to the ground. **High-Z** – Inactive sensors are floating (not connected to GND or Shield). **VDDA/2** – Inactive sensors are connected to the VDDA/2 voltage level source. This option is available for MSC HW only.
Enable IDAC auto-calibration *[CSD HW]*	When enabled, IDAC values are automatically set by the middleware. It is recommended to select the Enable IDAC auto-calibration for robust operation.
Enable CDAC auto-calibration *[MSC HW]*	When enabled, the CDAC values are automatically set by the middleware. Select the Enable CDAC auto-calibration for robust operation.
Enable compensation CDAC *[MSC HW]*	The compensation CDAC is used to compensate for sensor parasitic capacitance to improve the performance. Select the compensation CDAC unless one CDAC is required for general purposes (other than CAPSENSE™) in the application.

6.2.4 Widget Details subtab

This sub-tab contains parameters specific to each widget and sensor. These parameters must be set when *SmartSense Auto-tuning* is not enabled. The parameters are unique for each widget type.



Commands:

- **Restore Defaults** – restores parameters values on the current tab to their default values.

The **Widget Details** subtab contains the following parameters:

Name	Description
Widget general parameters	
Diplexing	Enabling Diplexing allows doubling the slider physical touch sensing area by using the specific duplexing sensor pattern and without using additional port pins and sensors.

Name	Description
Maximum position	Represents the maximum Centroid position for the slider. A touch on the slider would produce a position value from 0 to the maximum position-value set. No Touch would produce 0x0000.
Maximum X-axis position	Represents the maximum column (X-axis) Centroid position and row (Y-axis) Centroid positions for a touchpad. A touch on the touchpad would produce a position value from 0 to the maximum position set. No Touch would produce 0x0000.
Maximum Y-axis position	
Enable multi-frequency scan *[MSC HW]*	Enables the multi-frequency scan for the current widget. Refer to *Enable multi-frequency scan* for details.

Widget hardware parameters

Note: *All the Widget Hardware parameters for the CSD widgets are automatically set when SmartSense Auto-tuning is selected in the CSD tuning mode.*

Sense clock divider	Sets the CSD Sense clock divider. When SmartSense is selected in *CSD tuning mode*, the Sense Clock divider is automatically set by the middleware to an optimal value by following the 2*5*R*C rule (refer to *CapSense design guide* for more information on this rule) and this control is not editable.
Row sense clock divider	Sets the CSD Sense clock divider for row and column sensors of the *Matrix Buttons* and *Touchpad* widgets.
Column sense clock divider	
Tx clock divider	Sets the Tx Clock divider for the CSX widgets.

Name	Description
Sense clock source *[CSD HW]*	The Sense clock frequency is derived from the Modulator clock frequency using a clock-divider and is used to sample the sensor. Both the clock source and clock divider are configurable. The Spread Spectrum Clock (SSC) provides a dithering clock source with a center frequency equal to the Sense clock frequency. The PRS clock source spreads the clock using the pseudo-random sequencer and the Direct source disables both SSC and PRS sources and uses a fixed-frequency clock. Both PRS and SSC reduce the radiated noise by spreading the clock and improve the immunity against external noise. Using a higher number of bits of SSC and PRS lowers the radiation and increases the immunity against external noise. The following sources are available: *Direct* – PRS and SSC are disabled and a fixed clock is used. *PRS8* – The clock spreads using PRS to Modulator Clock / 256. *PRS12* – The clock spreads using PRS to Modulator Clock / 4096. *SSC6, SSC7, SSC9* and *SSC10* – The clock spreads using from 6 to 10 bits of the sense-clock divider respectively. *Auto* – The middleware automatically selects optimal SSC, PRS or Direct sources individually for each widget. The Auto is the recommended sense clock source selection. The following rules and recommendations for the SSC selection: The ratio between the Modulator clock frequency and Sense clock frequency must be greater than or equal to 20. 20% of the ratio between the Modulator clock frequency and Sense clock frequency should be greater or equal to the SSC frequency range = 32. It allows varying the ratio between the Modulator and Sense clock frequencies to 32 different clocks evenly spaced over +/- 10% from the center frequency. $160 \le \frac{ModClk}{SnsClk}$ Where *ModClk* is the Modulator clock frequency and *SnsClk* is Sense clock frequency. It is recommended that at least one full-spread spectrum polynomial should end during the scan time: $\frac{2^N - 1}{ModClk} \ge \frac{2^{SSCN} - 1}{SnsClk}$ where *N* is the *Scan resolution*, *SSCN* is the number of bits used for SSC (6, 7, 9 and 10), *ModClk* is Modulator clock frequency and *SnsClk* is Sense clock frequency. It is recommended that the number of sub-conversions for the widget should be an integer multiple of the SSC polynomial selected. For example, if SSC6 is selected, the number of the sub-conversion should be multiple of (2^{SSC6}-1) = 63. The recommendation for the PRS selection: At least one full PRS polynomial should finish during the scan time: $\frac{2^N - 1}{ModClk} \ge \frac{2^{PRSN} - 1}{SnsClk}$ where *N* is the *Scan resolution*, *PRSN* is the number of bits used for PRS (8 and 12), *ModClk* is the Modulator clock frequency and *SnsClk* is the average Sense clock frequency.

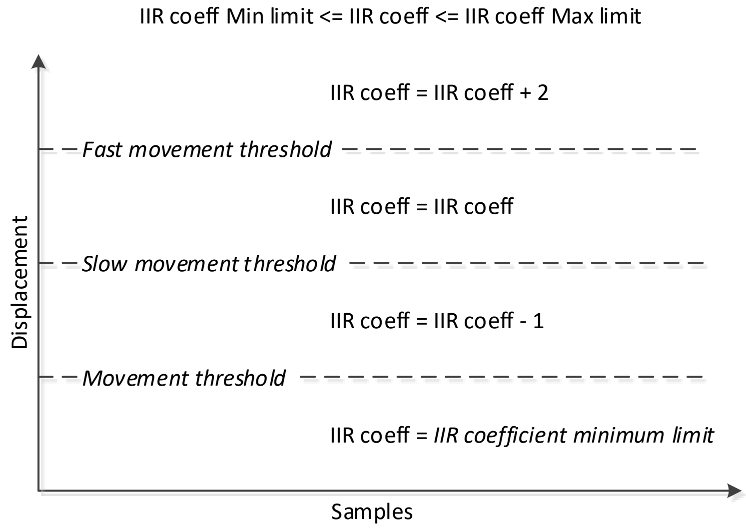
Name	Description
Tx clock source *[CSD HW]*	The Tx clock frequency derives from the Modulator clock frequency using a clock-divider and is used to sample the sensor. Both the clock source and clock divider are configurable. The Spread Spectrum Clock (SSC) provides a dithering clock source with a center frequency equal to the Tx clock frequency and the Direct source disables the SSC source and uses a fixed frequency clock. The SSC reduces the radiated noise by spreading the clock and improves the immunity against external noise. Using a higher number of bits of SSC lowers the radiation and increases the immunity against external noise. The following clock sources are available: *Direct* – SSC is disabled and a fixed clock is used. *SSC6, SSC7, SSC9* and *SSC10* – The clock spreads using from 6 to 10 bits of the sense-clock divider respectively. *Auto* – The middleware automatically selects optimal SSC or Direct sources individually for each widget. Auto is the recommended Sense clock source selection. The rules and recommendations for the SSC selection: The ratio between the Modulator clock frequency and Tx clock frequency must be greater than or equal to 20. 20% of the ratio between the Modulator clock frequency and Tx clock frequency should be greater or equal to the SSC frequency range = 32. It allows varying the ratio between the Modulator and Tx clock frequencies to 32 different clocks evenly spaced over +/- 10% from the center frequency. $160 \le \frac{ModClk}{TxClk}$ where *ModClk* is the Modulator clock frequency and *TxClk* is Tx clock frequency. It is recommended that at least one full-spread spectrum polynomial should end during the scan time. $\frac{N_{Sub}}{ModClk} \ge \frac{2^{SSCN} - 1}{TxClk}$ where N_{Sub} is the *Number of sub-conversions*, *SSCN* is the number of bits used for SSC (6, 7, 9 and 10), *ModClk* is the Modulator clock frequency and *TxClk* is the Tx clock frequency. It is recommended that *Number of sub-conversions* for the widget should be an integer multiple of the SSC polynomial selected. For example, if SSC6 is selected, the number of sub-conversions should be multiple of (2^{SSC6}-1) = 63.
Clock source *[MSC HW]*	The clock frequency is used to sample the sensor. It is derived from the modulator clock frequency using a clock divider. Both the clock source and clock divider are configurable. The Spread Spectrum Clock (SSC) provides a dithering clock source with a center frequency equal to the Sense clock frequency. The PRS clock source spreads the clock using the pseudo-random sequencer and the Direct source disables both SSC and PRS sources and uses a fixed-frequency clock. Both PRS and SSC reduce the radiated noise by spreading the clock and improve the immunity against external noise. Using a higher number of bits of SSC and PRS lowers the radiation and increases the immunity against external noise. The sources: *Direct* – Disable PRS and SSC and use a fixed clock. *SSC* – The clock spreads the by variation of sense-clock divider in the [-16,15] range. *PRS* – The clock spreads using PRS to Sensor Clock. *SSC Auto* – The middleware automatically selects optimal SSC or Direct sources individually for each widget. *PRS Auto* – The middleware automatically selects optimal PRS or Direct sources individually for each widget.

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>LFSR range
[MSC HW]</td><td>For CSD widgets, sets the Sense Clock Divider deviation range. For CSX widgets, sets the Tx Clock Divider deviation range.
For example, if the clock divider is set to 16 and the LFSR range is set to [-2; 1], the MSC HW block will vary the clock divider in the range from 14(16 - 2) to 17 (16 + 1) during the scan.
This parameter is editable when the Clock Source is SSC or SSC Auto.</td></tr>
<tr><td>Scan resolution
[CSD HW]</td><td>Selects the scan resolution of the CSD widgets (Resolution of capacitance to digital conversion). Acceptable values are from 6 to 16 bits.</td></tr>
<tr><td>Number of sub-conversions</td><td>Selects the number of sub-conversions.
For the CSD block, applicable to the *CSX sensing method*.</td></tr>
<tr><td>Modulator IDAC
[CSD HW]</td><td>Sets the modulator IDAC value for the CSD Button, Slider, or Proximity widget.
The value of this parameter is automatically set when *Enable IDAC auto-calibration* is selected in the *CSD* Settings tab.</td></tr>
<tr><td>Row modulator IDAC
[CSD HW]</td><td rowspan="2">Sets a separate modulator IDAC value for the row and column sensors of the CSD *Matrix Buttons* and *Touchpad* widget.
These parameters values are automatically set when *Enable IDAC auto-calibration* is checked in the *CSD* Settings tab.</td></tr>
<tr><td>Column modulator IDAC
[CSD HW]</td></tr>
<tr><td>IDAC gain index
[CSD HW]</td><td>Sets the IDAC gain index. Options include:
Index 0 – 37.5 nA
Index 1 - 75 nA
Index 2 - 300 nA (default for CSX widgets)
Index 3 - 600 nA
Index 4 - 2400 nA (default for CSD widgets)
Index 5 - 4800 nA
The value of this parameter is automatically set when Enable IDAC auto-calibration is selected.</td></tr>
<tr><td>Reference CDAC value
[MSC HW]</td><td>Sets the reference CDAC value for the Button, Slider, or Proximity widget.
Values are set automatically when *Enable CDAC auto-calibration* is selected in the *CSD* Settings tab.</td></tr>
<tr><td>Row reference CDAC value
[MSC HW]</td><td rowspan="2">Sets separate CDAC values for the row and column sensors of the *Matrix Buttons* and *Touchpad* widgets.
These parameters values are automatically set when *Enable CDAC auto-calibration* is checked in the *CSD* Settings tab or *CSX Settings* tab depending on a widget sensing mode.</td></tr>
<tr><td>Column reference CDAC value
[MSC HW]</td></tr>
<tr><td>Enable CDAC dither
[MSC HW]</td><td>Enables the CDAC dithering.</td></tr>
<tr><td>Compensation CDAC divider *[MSC HW]*</td><td>The number of times the DAC switches in the sense clock period.</td></tr>
</table>

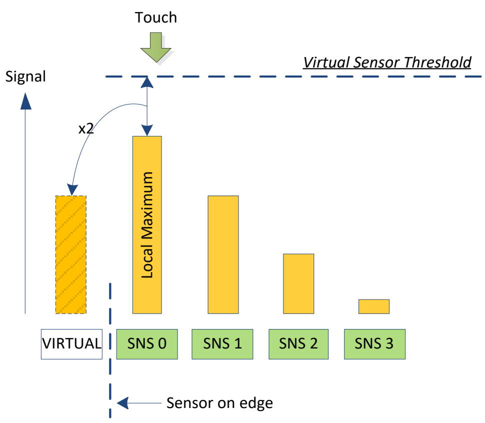
Name	Description
Widget threshold parameters	
Note:	*All the threshold parameters for the CSD widgets are automatically set when SmartSense (Full Auto-Tune) is selected in the CSD tuning mode parameter.*
Finger threshold	The finger threshold parameter is used along with the hysteresis parameter to determine the sensor state as follows: ON – Signal > (Finger Threshold + Hysteresis) OFF – Signal ≤ (Finger Threshold – Hysteresis). Note that "Signal" in the above equations refers to: Difference Count = Raw Count – Baseline. It is recommended to set the Finger threshold parameter value equal to the 80% of the touch signal. The Finger Threshold parameter is not available for the *Proximity* widget. Instead, Proximity has two thresholds: *Proximity threshold* *Touch threshold*
Noise threshold	Sets a raw count limit below which a raw count is considered as noise. When a raw count is above the Noise Threshold, a difference count is produced and the baseline is updated only if *Enable sensor auto-reset* is selected (In other words, the baseline remains constant as long as the raw count is above the baseline + noise threshold. This prevents the baseline from following raw counts during a finger touch detection event). It is recommended to set the noise threshold parameter value equal to 2x noise in the raw count or the 40% of the signal.
Negative noise threshold	Sets a raw count limit below which the baseline is not updated for the number of samples specified by the *Low baseline reset* parameter. The negative noise threshold ensures that the baseline does not fall low because of any high-amplitude repeated negative-noise spikes on a raw count caused by different noise sources such as ESD events. It is recommended to set the negative noise threshold parameter value equal to the *Noise threshold* parameter value.
Low baseline reset	This parameter is used along with the *Negative noise threshold* parameter. It counts the number of abnormally low raw counts required to reset the baseline. If a finger is placed on the sensor during a device startup, the baseline gets initialized to a high raw count value at a startup. When the finger is removed, the raw count falls to a lower value. In this case, the baseline should track low raw counts. The Low Baseline Reset parameter helps handle this event. It resets the baseline to a low raw count value when the number of low samples reaches the low-baseline reset number. *Note:* *After a finger is removed from the sensor, the sensor will not respond to finger touches for low baseline-reset time.* The recommended value is 30 which works for most designs.
Hysteresis	The hysteresis parameter is used along with the *Finger threshold* parameter (*Proximity threshold* and *Touch threshold* for Proximity sensor) to determine the sensor state. The hysteresis provides immunity against noisy transitions of the sensor state. See the description of the *Finger threshold* parameter for details. The recommend value for the hysteresis is the 10% *Finger threshold*.

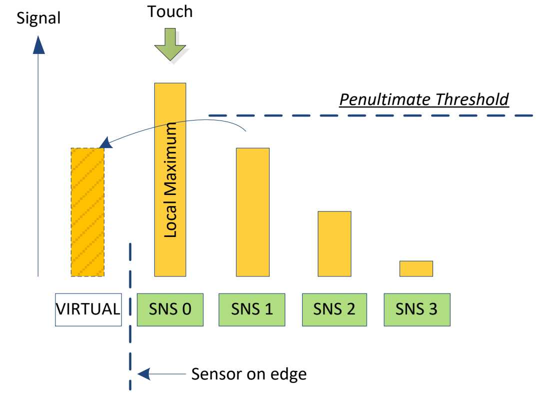
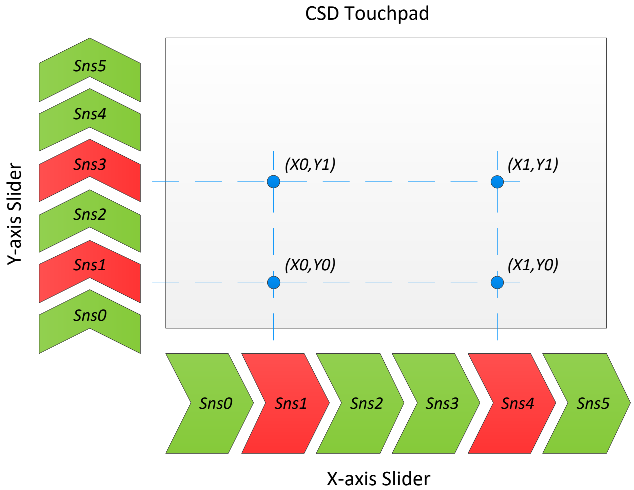
Name	Description
ON debounce	Selects a number of consecutive CAPSENSE™ scans during which a sensor must be active to generate an ON state from the middleware. The Debounce ensures that high-frequency, high-amplitude noise does not cause false detection Buttons/Matrix buttons/Proximity – An ON status is reported only when the sensor is touched for a consecutive debounce number of samples. Sliders/Touchpads – The position status is reported only when any of the sensors is touched for a consecutive debounce number of samples. The recommended value for the Debounce parameter is 3 for reliable sensor status detection.
Proximity threshold Touch threshold	The design of these parameters is the same as for the *Finger threshold* parameters. The proximity sensor requires a higher noise reduction, and supports two levels of detection: The proximity level to detect an approaching hand or finger. The touch level to detect a finger touch on the sensor similarly to other *Widget Type* sensors. Note that for valid operation, the Proximity threshold must be higher than the Touch threshold. The threshold parameters such as *Hysteresis* and *ON debounce* are applicable to both detection levels.
Velocity	Defines the maximum speed of a finger movement in terms of the squared distance of the touchpad resolution. The parameter is applicable for a multi-touch touchpad (CSX Touchpad) only. If the detected position of the next scan is further than the defined squared distance, then this touch is considered as a separate touch with a new touch ID.
Position filter parameters These parameters enable firmware filters on a centroid position to reduce noise. These filters are available for Slider and Touchpad widgets only. If multiple filters are enabled, the execution order corresponds to the listed below and the total RAM consumption increases so that the size of the total filter history is equal to a sum of all enabled filter histories.	
Infinite-impulse response (IIR) filter	Enables the IIR filter (see equation below) with a step response. $Output = \frac{N}{K} \times Input + \frac{(K-N)}{K} \times prevOutput$ where: *K* is always 256; *N* is the IIR filter raw count coefficient selectable from 1 to 255 in the configurator. A lower N (set in the *IIR filter coefficient* parameter) results in lower noise, but slows down the response. This filter eliminates high-frequency noise. Consumes 2 bytes of RAM per each position (filter history).
IIR filter coefficient	The coefficient (N) of the IIR filter for a position as explained in the *Infinite-impulse response* (IIR) filter parameter. The range of valid values: 1-255.
Median filter	Enables a non-linear filter that takes three of most recent samples and computes the median value. This filter eliminates the spikes noise typically caused by motors and switching power supplies. Consumes 4 bytes of RAM per each position (filter history).
Average filter	Enables the finite-impulse response filter (no feedback) with equally weighted coefficients. It takes two of most recent samples and computes their average. Eliminates periodic noise (e.g. noise from AC mains). Consumes 2 bytes of RAM per each position (filter history).
Jitter filter	This filter eliminates the noise in the position data that toggles between the two most recent values. If the most recent position value is greater than the previous one, the current position is decremented by 1; if it is less, the current position is incremented by 1. The filter is most effective at low noise. Consumes 2 bytes of RAM per each position (filter history).

Name	Description
Adaptive IIR filter parameters	

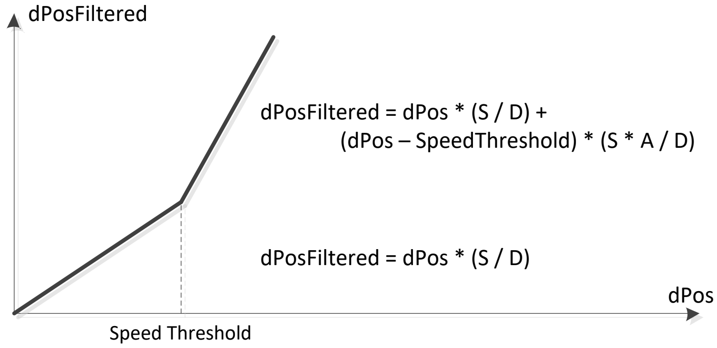


Name	Description
Adaptive IIR filter	Enables the Adaptive IIR filter. It is the IIR filter that changes its IIR coefficient according to the speed of the finger movement. This is done to smooth the fast movement of the finger and at the same time control properly the position movement. The filter coefficients are automatically adjusted by the adaptive algorithm with the speed of the finger movement. If the finger moves slowly, the IIR coefficient decreases; if the finger moves fast, the IIR coefficient increases from the existing value. Consumes 3 bytes of RAM per each position (filter history).
Position movement threshold	Defines the position threshold below which a position displacement is ignored or considered as no movement.
Position slow movement threshold	Defines the position threshold below which (and above Position movement threshold) a position displacement (the difference between the current and previous position) is considered as slow movement. If the position displacement is within the threshold limits, the IIR filter coefficient decreases during each new scan. So, the filter impact on the position becomes less intensive.
Position fast movement threshold	Defines the position threshold above which a position displacement is considered as fast movement. If the position displacement is above the threshold limit, the IIR filter impact on the position becomes more intensive during each new scan as the filter coefficient increases.
IIR coefficient maximum limit	Defines the maximum limit of the IIR coefficient when the finger moves fast. The fast movement event is defined by the Position fast movement threshold.
IIR coefficient minimum limit	Defines the minimum limit of the IIR coefficient when the finger moves slowly. The slow movement event is defined by the Position slow movement threshold.
IIR coefficient divisor	This parameter acts as the scale factor for the filter IIR coefficient. $Output = \frac{Coeff}{Divisor} \times Input + \frac{Divisor - Coeff}{Divisor} \times previousOutput$ where: *Input*, *Output*, and *Previous Output* are the touch positions; *Coeff* is the automatically adjusted IIR filter coefficient; *Divisor* is the IIR coefficient divisor (this parameter).
Centroid parameters These parameters are available for the CSD Touchpad widgets only.	
Centroid type	Selects a sensor matrix size for centroid calculation. The 5x5 centroid (also known as Advanced Centroid) provides benefits such as Two-finger detection, Edge correction and improved accuracy. If Advanced Centroid is selected, the below parameters are configured as well.

Name	**Description**
Cross-coupling position threshold	Defines the cross-coupling threshold. This value is subtracted from the sensor signal used for centroid position calculation to improve the accuracy. The threshold should be equal to a sensor signal when a finger is near the sensor but is not touching the sensor. This can be determined by slowly dragging the finger across the panel and finding the inflection point of the difference counts at the base of the curve. The difference value at this point is the Cross-coupling threshold. The default value is 5.
Edge correction	This feature is available if the Centroid Type is configured to 5x5. When enabled, a matrix of centroid calculation is updated with virtual sensors on the edges of a touchpad. It improves the accuracy of the reported position on the edges. When enabled, two more parameters must be configured: Virtual Sensor threshold and Penultimate threshold.
Virtual sensor threshold	This parameter is applicable only if Edge correction is enabled and it is used to calculate a signal (difference count) for a virtual sensor used for the edge correction algorithm. A touch position on a slider or touchpad is calculated using a signal from the local-maxima sensor and its neighboring sensors. A touch on the edge sensor of a slider or touchpad does not accurately report a position because the edge sensor lacks signal from one side of neighboring sensors of the local-maxima sensor.  If the Edge correction is enabled, the algorithm adds a virtual neighbor sensor to correct the deviation in the reported position. The Virtual sensor signal is defined by the Virtual sensor threshold: $DiffCount_{VIRTUAL} = (Threshold_{VIRTUAL} - DiffCount_{SNS0}) \times 2$ where: $DiffCount_{VIRTUAL}$ is the virtual sensor difference count; $Threshold_{VIRTUAL}$ is the virtual sensor threshold; $DiffCount_{SNS0}$ is the sensor 0 difference count. The conditions for a virtual sensor (and Edge correction algorithm) to be applied: Local-maxima detected on the edge sensor Difference count from the penultimate sensor less than the Penultimate threshold.

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Penultimate threshold</td><td>This parameter is applicable only if the Edge correction is enabled and it works along with the Virtual sensor threshold parameter.
This parameter defines the threshold of penultimate sensor signal. If the signal from penultimate sensor is below the Penultimate threshold, the edge correction algorithm is applied to the centroid calculation.
The conditions for the edge correction to be applied:
Local-maxima detected on the edge sensor
The difference count of the penultimate sensor (SNS 1 in the figure below) less than the Penultimate threshold.

</td></tr>
<tr><td>Two-finger detection</td><td>Enables the detection of the second finger on a CSD touchpad.
In general, a CSD touchpad can detect only one true touch position. A CSD touchpad widget consists of two Linear Sliders and each slider reports the X and Y coordinates of a finger touch. If there are two touches on the touchpad, there are four possible touch positions as shown in the figure below. The two of these touches are real touches and two are known as “ghost” touches. There is no possibility to differentiate between ghost and real touches in a CSD widget (to get true multi-touch performance, use the CSX Touchpad widget).


But, if this feature is enabled, the CSD touchpad can report up to two touches, mainly to be used in conjunction with two-finger gestures where real and ghost touches do not need to be fully differentiated. It is available for the CSD touchpad only when the Centroid type is configured to 5x5.
The Advanced centroid (Centroid type is 5x5) uses the 3x3 centroid matrix when detects two touches.</td></tr>
</table>

Name	Description
Ballistic multiplier parameters	
These parameters are available for the CSD Touchpad widgets only.	
Ballistic multiplier	Enables the Ballistic multiplier filter used to provide better user experience of the pointer movement. Fast movement will move the cursor by more pixels. Consumes 16 bytes of RAM when enabled. The simplified diagram of the Ballistic Multiplier filter operation:  where, *dPos* is an input position displacement either in the X axis or Y axis, *dPosFiltered* is the filtered displacement; *SpeedThreshold* is either the X-axis speed threshold or Y-axis speed threshold; *A* is the Acceleration coefficient; *S* is the Speed coefficient; *D* is the Divisor value.
Acceleration coefficient	Defines the value at which the position movement needs to be interpolated when the movement is classified as fast movement. The reported position displacement is multiplied by this parameter.
Speed coefficient	Defines the value at which the position movement is interpolated when the movement is classified as slow movement. The reported position displacement is multiplied by this parameter.
Divisor value	Defines the divisor value used to create a fraction for the acceleration and speed coefficients. The interpolated position coordinates are divided by the value of this parameter.
X-axis speed threshold	Defines the threshold to distinguish fast and slow movement on the X axis. If the X-axis position displacement reported between two consecutive scans exceeds this threshold, then it is considered as fast movement, otherwise as slow movement.
Y-axis speed threshold	Defines the threshold to distinguish fast and slow movement on the Y axis. If the Y-axis position displacement reported between two consecutive scans exceeds this threshold, then it is considered as fast movement, otherwise as slow movement.
Gesture parameters	
Enable gestures	Master enable for gestures feature. Each gesture consists of a sequence of Touchdown and Lift Off events. A simple touch on a widget is reported as a Touchdown event. Removal of a finger from a widget reported as a Lift Off event. If the Lift Off event triggers another higher-level Gesture, then the Lift Off event is not reported.

Name	Description
Enable one-finger single click gestures	One-finger single click gesture is a combination of a Touchdown and Lift Off events with the conditions to be met: A touchdown event is followed by a Lift Off event. The touch duration (duration between touchdown and lift off) must be greater than Minimum click timeout and less than Maximum click timeout. Position displacements between the Touchdown and Lift Off events must be within the Maximum click distance.
Enable one-finger double click gestures	A One-finger double click gesture is a combination of two sequential one-finger single click gestures under specific conditions: Both clicks in the sequence must meet one-finger single click conditions. The touch duration between the two touchdown events must be within the Minimum second click interval and Maximum second click interval timeout limits. The distance between two clicks must not exceed the Maximum second click distance
Enable one-finger click & drag gestures	This gesture is a one-finger click and then a hold, followed by a drag. A typical use case is while moving items on the screen from one point to another. It is triggered when the finger movement follows this sequence: Touchdown → Lift Off → Touchdown → Drag Gesture triggering condition: A one-finger click gesture and a subsequent touchdown were detected within the Minimum click timeout and Maximum click timeout limits and within Maximum second click distance. Then the finger exceeds the Maximum click distance from a drag touchdown.
Enable two-finger single click gestures	A Two-finger single click gesture is a combination of a Touchdown and Lift Off events with under specific conditions: Two simultaneous finger touches (touchdown and lift off) should be detected. The duration between the second finger touchdown and lift off events of both fingers must be within the Minimum second click interval and Maximum second click interval timeout limits. The duration counting starts when the settling time elapsed for the second finger touchdown event. A position displacement between the touchdown and lift off events must be less than the Maximum second click distance.
Enable one-finger scroll gestures	A One-finger Scroll gesture is a combination of a touchdown followed by a displacement in a specific direction under specific conditions: For a slider, the position displacement between two consecutive scans must exceed the Minimum scroll distance. The Scroll debounce number of a scroll gesture in the same direction is already detected.
Enable two-finger scroll gestures	The design of a two-finger scroll gesture is the same as of a one-finger scroll gesture, except for the conditions below. The conditions of a one-finger scroll are met. There must be two simultaneous finger touches detected on a widget for a scroll to be considered as a two-finger scroll. The displacement of both finger touches must be on same direction for a two-finger scroll to be valid.
Enable one-finger edge swipe gestures	An edge swipe gesture is a combination of a touchdown on an edge followed by a displacement towards the center. The conditions for an edge swipe gesture: A touchdown event must occur in the edge area defined by the Edge size. A finger displacement must occur from the edge towards the center within the Maximum edge angle. The displacement must exceed the Minimum edge distance within the Maximum edge timeout duration.

Name	Description
Enable one-finger flick gestures	A flick gesture is a combination of a touchdown followed by a high-speed displacement and a lift off event. A flick gesture starts at a touchdown and ends and reported at a lift off event. The conditions for a flick gesture. The displacement must exceed the Minimum flick distance. The duration between a touchdown and lift off events must be less than the Maximum flick timeout. *Note:* *The flick gesture is detected in 8 directions:* *Up; Down; Left; Right; Up-Right; Down-Left; Up-Left; Down-Right*
Enable one-finger rotate gestures	A one-finger rotate gesture is reported when a circular displacement is detected. The decoding algorithm uses four directions to identify a circular displacement. A displacement in all four directions must be in the succession order to report a rotate gesture. The rotation direction can be clockwise or counter-clockwise.
Enable two-finger zoom gestures	A two-finger zoom gesture is reported when two touches move towards each other (Zoom Out) or move away from each other (Zoom In). The conditions for a zoom gesture: An increase or decrease in distance between two-finger touch positions must exceed the Minimum zoom distance. The zoom debounce number of a Zoom In or Zoom Out gesture must be sequentially detected for a Zoom gesture to be reported. A scroll to the zoom debounce number of a zoom gestures must be sequentially detected for a Zoom gesture to be reported. If a Zoom gesture occurred after a scroll, the gesture is reported and there was no lift off event between the scroll and Zoom gestures.
Enable gesture filtering	Enables filtering of the detected gestures. The gesture priority is defined as follows (starting from the most important): Two-finger zoom; Two-finger scroll; One-finger rotate; One-finger edge swipe; One-finger flick; One-finger scroll; Two-finger single click; One-finger click and drag; One-finger double click; One-finger single click; Touchdown; Liftoff
Maximum click timeout	Defines the maximum duration between a touchdown and lift off events of a click event. This parameter is used in all click-based gestures.
Minimum click timeout	Defines the minimum duration between a touchdown and lift off events of a click event. This parameter is used in all click-based gestures.
Maximum click distance	Defines the maximum displacement between a touchdown and lift off events of a click event. This parameter is used in all click-based gestures.
Maximum second click interval	Defines the maximum displacement between a touchdown and lift off events of a click event. This parameter is used in all click-based gestures.
Minimum second click interval	This parameter defines the minimum duration between the first lift off and the second touchdown events. If the second click occurs early this limit, the double click and click&drag gestures are not reported.
Maximum second click distance	Defines the maximum distance between the first lift off event and the second touchdown event. If the second click occurs outside this limit, the double click and click&drag gestures are not reported.
Scroll debounce	Defines the minimum number sequential scroll steps in the same direction to be detected prior to the scroll is considered valid. A widget must detect scroll steps, at the minimum of Debounce times in the same direction to be considered as a scroll in that direction.
Minimum scroll distance	Defines the minimum displacement to recognize a single scroll step. A scroll step is calculated between two consecutive scans.

Name	Description
Rotate debounce	Defines the maximum number of sequential rotate steps in the same direction to deem a rotate gesture invalid. For example, if the Debounce value is set to 5, then the touch cannot continue in the same direction for 5 rotate steps and still have a valid rotate gesture. After this threshold, the reported gesture stops being a rotate gesture. If this parameter is set to 0, then the Debounce is disabled.
Minimum rotate distance	Defines the minimum displacement to recognize a single rotate step.
Zoom debounce	Defines the minimum number of zoom steps in a particular direction (in or out) to report a zoom gesture.
Minimum zoom distance	Defines the minimum displacement to recognize a single zoom step.
Maximum flick timeout	Defines the maximum duration of how long a flick gesture is searched after a touchdown event. A position displacement and lift off event must happen within the duration defined by this parameter for a flick to be valid.
Minimum flick distance	Defines the minimum displacement to be detected for a one-finger flick to be valid.
Edge size	Defines the maximum edge area where a touchdown must be detected for an edge swipe to be reported.
Minimum edge distance	Defines the minimum displacement to be detected from an edge to the center for an edge swipe to be reported.
Maximum edge timeout	Defines the maximum duration, within which an edge swipe must occur to be reported. The displacement must exceed the displacement threshold within the duration defined by this parameter for the edge swipe to be reported.
Maximum edge angle	To report this gesture, a finger movement starts from an edge and moves in the center direction. This is the ideal line. These parameters define the maximum angle deviation (in degree) from this ideal line for the edge swipe to be valid. Degree 1 means that the user can do gestures only on a single ideal line.
Sensing parameters	
Compensation IDAC value *[CSD HW]*	Sets the Compensation IDAC value for each CSD sensor when *Enable compensation IDAC* is selected on the *CSD* Settings tab. If the *CSD tuning mode* is set to *SmartSense Auto-tuning* or *Enable IDAC auto-calibration* is selected on the *CSD* Settings tab, the value of this parameter is set equal to the Modulator IDAC value at a device power-up for the maximum performance from the sensor. Select the *Enable IDAC auto-calibration* for robust operation.
Compensation IDAC values *[CSD HW]*	Sets the IDAC value for each CSX sensor/node, a lower IDAC value without saturating raw counts provides better performance for sensor/nodes. When *Enable IDAC auto-calibration* is selected on the *CSX Settings* tab, the value of this parameter is automatically set to the lowest possible value at a device power-up for better performance. It is recommended to select *Enable IDAC auto-calibration* for robust operation.
Compensation CDAC value *[MSC HW]*	Sets the Compensation CDAC value for each CSD sensor when *Enable compensation CDAC* is selected on the *CSD* Settings tab. Select the *Enable CDAC auto-calibration* for robust operation.

Compensation CDAC value(s) *[MSC HW]*	Sets the CDAC value for each CSX sensor/node. A lower CDAC value without saturating raw counts provides for better performance for the sensor/nodes. When *Enable CDAC auto-calibration* is selected on the *CSX Settings* tab, the value of this parameter is automatically set to the lowest possible value at a device power-up for better performance. Select the *Enable CDAC auto-calibration* for robust operation.
Selected pins *[CSD HW]*	Selects a port pin for the sensor (CSD sensing) and electrode (CSX sensing). The available options use a dedicated pin for a sensor or re-use one or more pins from any other sensor. Re-using the pins of any other sensor from any widgets helps create a ganged sensor.

The following table shows which Widget / Sensor parameters belong to a given widget type:

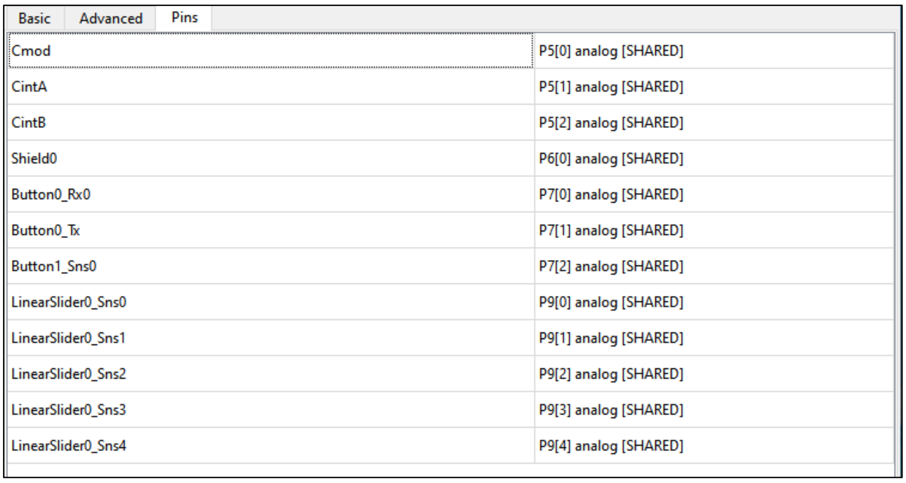
Parameters	**CSD HW**	**MSC HW**	**Widget type**									
			CSD widget						**CSX widget**			
			Button	**Linear Slider**	**Radial Slider**	**Matrix Buttons**	**Touchpad**	**Proximity**	**Button**	**Linear Slider**	**Matrix Buttons**	**Touchpad**
Widget general												
Diplexing	√	√		√						√		
Maximum position	√	√		√	√					√		
Maximum X-axis position	√	√					√					√
Maximum Y-axis position	√	√					√					√
Enable multi-frequency scan		√	√	√	√	√	√	√	√	√	√	√
Widget hardware												
Sense clock divider	√	√	√	√	√			√				
Column sense clock divider	√	√				√	√					
Row sense clock divider	√	√				√	√					
Sense clock source	√		√	√	√	√	√	√				
Tx clock divider	√	√							√	√	√	√
Tx clock source	√								√	√	√	√
Clock source		√	√	√	√	√	√	√	√	√	√	√
LFSR range		√	√	√	√	√	√	√	√	√	√	√
Scan resolution	√		√	√	√	√	√	√				
Number of sub-conversions	√								√	√	√	√
Number of sub-conversions		√	√	√	√	√	√	√	√	√	√	√
Modulator IDAC	√		√	√	√			√				
Column modulator IDAC	√					√	√					
Row modulator IDAC	√					√	√					
IDAC gain index	√		√	√	√	√	√	√	√	√	√	√
Reference CDAC value		√	√	√	√			√	√	√	√	√
Column reference CDAC value		√				√	√					
Row reference CDAC value		√				√	√					
Enable CDAC dither		√	√	√	√	√	√	√	√	√	√	√
Compensation CDAC divider		√	√	√	√	√	√	√	√	√	√	√
Widget threshold												
Proximity threshold	√	√						√				
Touch threshold	√	√						√				
Finger threshold	√	√	√	√	√	√	√		√	√	√	√

Parameters	**CSD HW**	**MSC HW**	**Widget type**									
			CSD widget						**CSX widget**			
			Button	**Linear Slider**	**Radial Slider**	**Matrix Buttons**	**Touchpad**	**Proximity**	**Button**	**Linear Slider**	**Matrix Buttons**	**Touchpad**
Noise threshold	√	√	√	√	√	√	√	√	√	√	√	√
Negative noise threshold	√	√	√	√	√	√	√	√	√	√	√	√
Low baseline reset	√	√	√	√	√	√	√	√	√	√	√	√
Hysteresis	√	√	√	√	√	√	√	√	√	√	√	√
ON debounce	√	√	√	√	√	√	√	√	√	√	√	√
Velocity	√	√										√
Sensing parameters												
Compensation IDAC value	√		√	√	√	√	√	√		√		
IDAC values	√								√		√	√
Selected pins	√		√	√	√	√	√	√	√	√	√	√
Compensation CDAC value		√	√	√	√	√	√	√				
Compensation CDAC values		√							√	√	√	√
Position filter parameters												
IIR filter	√	√		√	√		√			√		√
IIR filter coefficient	√	√		√	√		√			√		√
Median filter	√	√		√	√		√			√		√
Average filter	√	√		√	√		√			√		√
Jitter filter	√	√		√	√		√			√		√
Adaptive IIR filter parameters												
Adaptive IIR filter	√	√		√	√		√			√		√
Position movement threshold	√	√		√	√		√			√		√
Position slow movement threshold	√	√		√	√		√			√		√
Position fast movement threshold	√	√		√	√		√			√		√
IIR coefficient maximum limit	√	√		√	√		√			√		√
IIR coefficient minimum limit	√	√		√	√		√			√		√
IIR coefficient divisor	√	√		√	√		√			√		√
Centroid parameters												
Centroid type	√	√					√					
Cross-coupling position threshold	√	√					√					
Edge correction	√	√					√					
Virtual sensor threshold	√	√					√					
Penultimate threshold	√	√					√					
Two-finger detection	√	√					√					
Ballistic multiplier parameters												
Ballistic multiplier	√	√					√					
Acceleration coefficient	√	√					√					
Speed coefficient	√	√					√					
Divisor value	√	√					√					
X-axis speed threshold	√	√					√					
Y-axis speed threshold	√	√					√					
Gesture parameters												
Enable gestures	√	√		√			√			√		√

Parameters	CSD HW	MSC HW	Widget type									
			CSD widget						CSX widget			
			Button	Linear Slider	Radial Slider	Matrix Buttons	Touchpad	Proximity	Button	Linear Slider	Matrix Buttons	Touchpad
Enable one-finger single click gestures	√	√		√			√			√		√
Enable one-finger double click gestures	√	√		√			√			√		√
Enable one-finger click & drag gestures	√	√		√			√			√		√
Enable two-finger single click gestures	√	√		√			√			√		√
Enable one-finger scroll gestures	√	√		√			√			√		√
Enable two-finger scroll gestures	√	√		√			√			√		√
Enable one-finger edge swipe gestures	√	√		√			√			√		√
Enable one-finger flick gestures	√	√		√			√			√		√
Enable one-finger rotate gestures	√	√		√			√			√		√
Enable one-finger zoom gestures	√	√		√			√			√		√
Enable gesture filtering	√	√		√			√			√		√
Maximum click timeout	√	√		√			√			√		√
Minimum click timeout	√	√		√			√			√		√
Maximum click distance	√	√		√			√			√		√
Maximum second click interval	√	√		√			√			√		√
Maximum second click interval	√	√		√			√			√		√
Maximum second click distance	√	√		√			√			√		√
Scroll debounce	√	√		√			√			√		√
Minimum scroll distance	√	√		√			√			√		√
Rotate debounce	√	√		√			√			√		√
Minimum rotate distance	√	√		√			√			√		√
Zoom debounce	√	√		√			√			√		√
Minimum zoom distance	√	√		√			√			√		√
Maximum flick timeout	√	√		√			√			√		√
Maximum flick distance	√	√		√			√			√		√
Edge size	√	√		√			√			√		√
Minimum edge distance	√	√		√			√			√		√
Maximum edge timeout	√	√		√			√			√		√
Maximum edge angle	√	√		√			√			√		√

6.3 Pins tab *[CSD HW]*

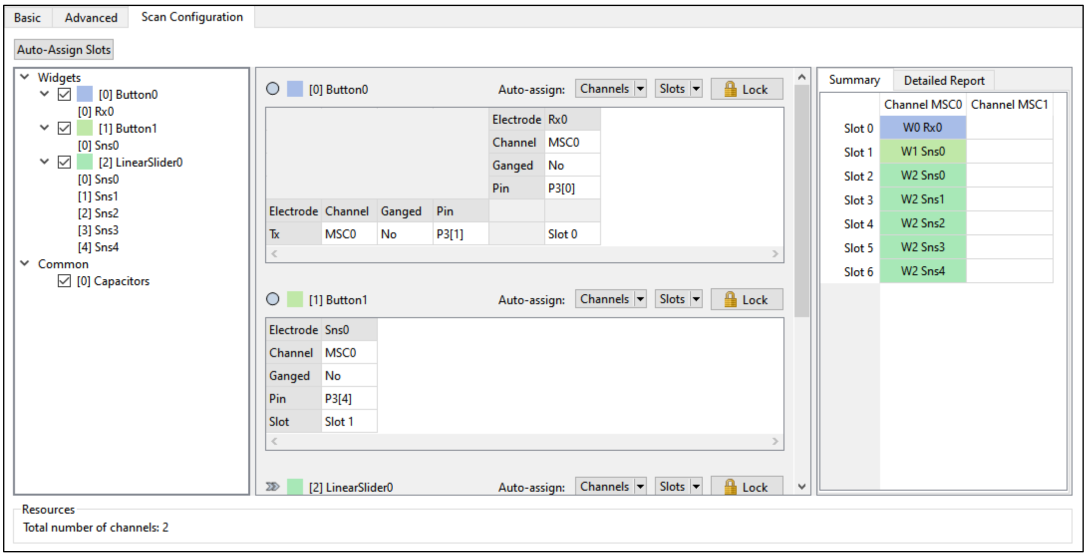
Use the **Pins** tab to assign pins to each sensor. Select the appropriate signal from the pull-down menu.



6.4 Scan Configuration tab *[MSC HW]*

Use the **Scan Configuration** tab to distribute the electrodes among the channels, make ganged connection, assign pins, and scan the slots to each sensor.

MSCv3:



Commands:

- **Auto-Assign Slots** – automatically reassigns all slots for sensors based on a widget and sensor order depending on the assigned channel. Locked widgets are not modified.

6.4.1 Widgets Explorer

The **Widgets Explorer** is used for toggling widgets, capacitors, and shields in the **Widgets configuration** pane.

6.4.2 Widgets configuration pane

The **Widgets** configuration pane contains tables for configuring channels, pins, and slots for each instance from the **Widget Explorer**.

Parameter	Description
Channel	Selects a channel in the multi-channel solutions. The available channels correspond to the enabled MSC resources.
Ganged	Selects a port pin for the sensor (CSD sensing) and electrode (CSX sensing). The available options use a dedicated pin for a sensor or re-use one or more pins from any other sensor. The latter helps create a ganged sensor.
Pin	Assigns pins for sensors. Select the appropriate signal from the pull-down menu.
Slot	Selects scan slots for sensors. In Multi-channel mode, a scan slot represents a group of sensors scanned together. In Single-channel mode, one sensor is scanned per scanning slot.

6.4.2.1 Commands

- **Auto-Assign Channels** – automatically assigns channels for widget electrodes. Multiple options are available:
  - Assign all electrodes to one channel.
  - Assign electrodes to different channels sequentially.
  - Assign electrodes to different channels alternately.
  - Assign only columns or rows.
- **Auto-Assign Slots** – incrementally assigns the slots for all widget sensors based on:
  - The slot of the first electrode.
  - The slots of the first electrodes on each channel.

  This command does not reassign channels.
- **Lock** – prevents the widget scan configuration from editing and slot reassignment.

6.4.3 Summary Table

The **Summary table** visually represents scan slot configuration. The cell color indicates the state of a scan slot:

- white – not occupied
- red – occupied by more than one sensor
- gray – reserved
- other – corresponds to the color of the widget, which occupies the slot; green-color shades for CSD widgets and blue-color shades for CSX widgets.

The red color in the index column cell indicates an error in this slot. The tooltip provides a description of the error.

6.4.4 Detailed Report

The **Detailed report** provides all relevant information about slot assignment.

Column name	Description
Widget	The index of the widget, which sensor is assigned on that slot.
Node/Sns	The index of the node or sensor assigned to that slot.
Eltd config	The electrode configuration of the node.
Sensor clock, kHz	The CSD Sense clock frequency or CSX Tx clock frequency of the widget assigned to that slot.
Number of conversions	The *Number of sub-conversions* of the widget assigned to that slot.
Scan time, us	Time needed to perform a scan of the slot.
Status	Displays errors if any. The text of the error is shown in the cell’s tooltip.

7 Version changes

This section lists and describes the changes for each version of this tool.

Version	Description
1.0	New tool.
1.1	Added the Notice List.
	Added more configuration parameters validation.
	Fixed minor issues.
2.0	Added "IDAC gain index" parameter.
	Changed the data storage location from the header (.h) file to XML-based file with the .cycapsense extension. For backward compatibility, the configurator is still able to load the header (.h) file that contains the legacy format configuration. But, if the legacy header (.h) with the configuration is passed via a command-line parameter, a message appears saying that the .h file is not supported.
	Added the **Import** and **Export** options to the **File** menu that enable importing and exporting the configuration file from and into the external file.
	Added the **Reset View** command to the **View** menu that resets the view to the default.
	Changed the Widget / Sensor parameters and Widget Type table to align with the actual CapSense Configurator widget parameters and types. Changed the name of Section “Sensor Parameters” to “Sensing Parameters” to align with the tool.
	Changed generation of the middleware initialization structure according to the changes in CapSense v2.0 middleware (adding fields for flash memory optimization, fixed the defect with the rawcount filters config, IDAC gain index, etc.)
	Added verification if the provided MPNs match the contents of the design.modus/xml config file.
	Added the warning about opening a broken configuration file.
	Added highlighting bold of modified properties in the property grid.
	Added handling of invalid command-line arguments.
	Fixed the pin assignment issues.
3.0	Added the self-test library support.
	Added the Undo / Redo feature.
	Improved configuration validation. Added new validation rules.
3.10	Updated versioning to support patches.
	Added Copy feature to the Notice List.
	Fixed the error visualization for the Enable shield electrode parameter.
	Removed duplicated gesture defines from the generated code.
	Fixed the xxx_PARAM_ID define value with the correct widget id.
3.11	Updated versioning to support the updated backend, for detail, see Device Configurator User Guide.
3.15	Added support of PSoC 4 devices.
	Prohibited saving configurations with errors.
	Removed the command-line generate options: -g and --generate.
4.0	Added support for the PSoC 4100S Max family.
	Added support for the CAPSENSE™ Middleware Library 3.0.
	Added support of the CSX Linear Slider.
	Added two more generated files: cycfg_capsense_defines.h and cycfg_capsense_tuner_regmap.h
	Added Undo/Redo support for pins selection.

Version	Description
	Removed: the migration of configuration to the current XML format – configuration saved in the comments in generated HEADER files (the old method).

Revision history

Revision	Date	Description
**	11/26/2018	New document.
*A	12/05/2018	Documents were updated with changes from business unit.
*B	02/26/2019	Updated to version 1.1.
*C	10/16/2019	Updated to version 2.0.
*D	03/27/2020	Updated to version 3.0.
*E	09/01/2020	Updated to version 3.10.
*F	12/14/2020	Updated to version 3.11.
*G	03/15/2021	Updated to version 3.15.
*H	09/27/2021	Updated to version 4.0.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 12-07-2021

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-24372 Rev. *H