Preliminary

SmartSense2X_EMC Datasheet SmartSense2X_EMC V 1.00

Copyright © 2013 Cypress Semiconductor Corporation. All Rights Reserved.

This datasheet contains Preliminary information

Resources	PSoC® Blocks				API Memory (Bytes)		Pins (in addition to sensors)
	CapSense®	I²C/SPI	Timer	Comparator	Flash	RAM	
CY8C21x45, CY8C22x45							
Single channel with IDAC	1	0	–	1	1006	29	1
Double channel with IDAC	2	–	–	2	1390	30	2
Each additional CapSense button	–	–	–	–	8	12	1
With auto-tune to noise	1	–	–	–	2937	52	2+1 optional
With auto-tune to sensitivity	1	–	–	–	2954	52	2+1 optional
Static code and RAM increase when using capacitive slider with 5 elements	–	–	–	–	613	90	5
Each additional slider segment	–	–	–	–	8	12	1
With auto-tune to noise	2	–	2	–	3442	52	2+1 optional
With auto-tune to sensitivity	2	–	–	–	2954	52	2+1 optional
Static code and RAM increase during slider diplexing (5 sensors)	–	–	–	–	10	–	–

Features and Overview

- Dual-channel SmartSense™.
- Manual and auto threshold setting for finger detection.
- Configurable sensitivity for each sensor.
- Background scan.
- Autoreset and button debounce.
- Buttons, sliders, and proximity sensors.
- Enables guided sensor and pin assignments using the SmartSense2X_EMC wizard.
- High immunity to AC mains noise, other EMI, and power supply noise.

Introduction

The SmartSense2X_EMC User Module is a SmartSense algorithm on top of the dual-channel CapSense® CSD2X User Module. The SmartSense2X_EMC User Module implements CapSense capacitive sensing. CapSense is a human interface technology that operates by detecting the capacitance of the human body. This is done using sensors that consist of a conductive surface, usually a pad etched on the PCB. Because CapSense detects body capacitance, it can sense through insulating layers such as plastic or glass overlays. These overlays usually constitute the external enclosure of the device. These attributes make CapSense an attractive alternative to mechanical input devices such as push buttons and potentiometers.

The SmartSense2X_EMC User Module has background scanning that enables the CPU to do other tasks while the scan is in progress in hardware. This is an attractive feature when using a CapSense Plus application such as LED fading or motor driving.

Modulator Capacitor Pin

SmartSense2X_EMC requires an external modulation capacitor, Cmod, connected from Vss to one of the two dedicated PSoC pins, P0[5] or P0[7]. The SmartSense2X_EMC dual-channel configuration requires two external modulation capacitors that must be connected to P0[5] and P0[7] pins (Modulator Capacitor Pin property is not available and pins P0[5] and P0[7] are locked).

The selected pins must not be used for any other purpose. The recommended value for the external modulation capacitor is 2.2 nF. A ceramic capacitor must be used. The temperature capacitance coefficient is not important. Cypress recommends using a 560-ohm series resistor on all CapSense sensor traces for RF interference suppression. This resistor must be placed as close to the PSoC device as possible.

PCB Level

Figure 1 and Figure 2 show a schematic of the SmartSense2X_EMC User Module for single and double channels. The physical sensor is typically a conductive pattern constructed on a PCB connected to a PSoC I/O pin with an insulating overlay. See the design guide Getting Started with CapSense for more information on PCB-level CapSense implementation.

Figure 1. SmartSense2X_EMC Single-channel Schematic

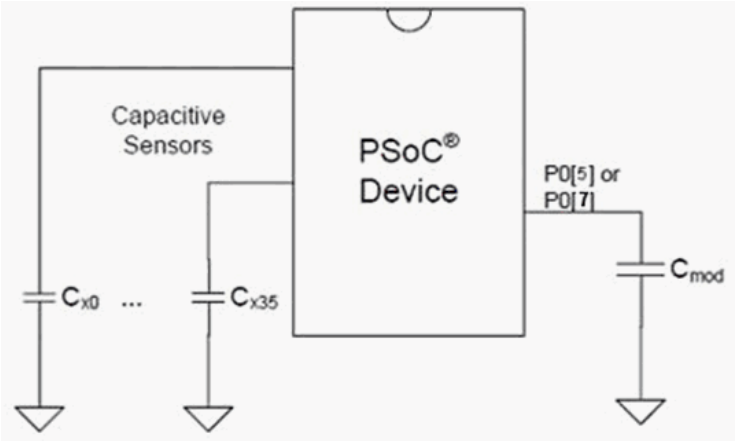
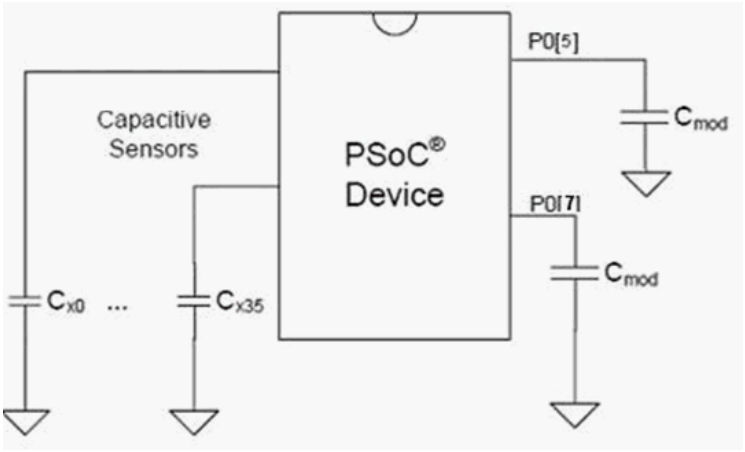


Figure 2. SmartSense2X Double-Channel Schematic

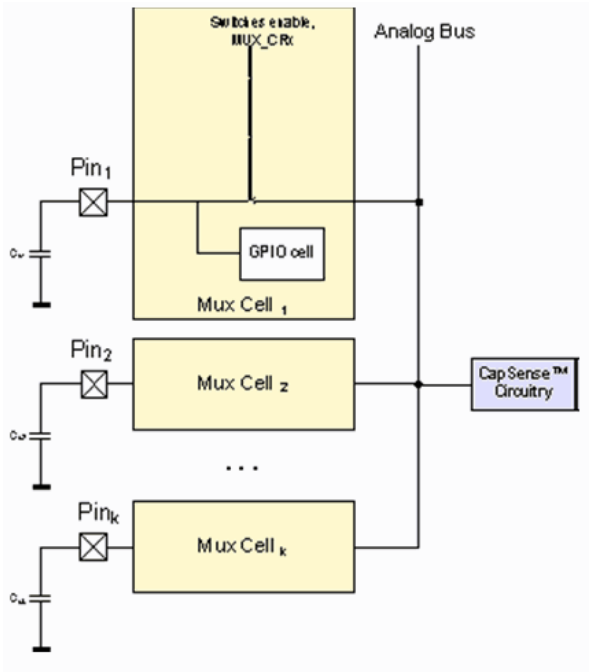


IC Level

The SmartSense2X_EMC User Module enables capacitive sensing algorithm using a sigma-delta modulator with many of the CSD parameters configured automatically and dynamically during runtime or power-on-reset (POR). The SmartSense2X_EMC User Module's hardware use is similar to that of CSD2X, except for the addition of a software layer, which includes the SmartSense algorithm with background scanning and immunity to the noise improvement algorithm.

The CY8C22x45 device has an analog mux (AMUX) bus that allows connecting capacitive sensing analog circuitry to any PSoC pin. The SmartSense2X_EMC User Module connects the active sensor to the AMUX bus, enabling the CapSense circuitry to measure its capacitance and translate that capacitance into a digital code. The firmware serially scans the sensors by sequentially setting corresponding bits in the MUX_CRx registers. This is represented in Figure 3.

Figure 3. AMUX Block Diagram



Software

The SmartSense2X_EMC software component has the following attributes:

- Auto-tuning algorithms configure the analog capacitive sensing circuitry in runtime for optimal performance. These algorithms take into account physical sensor characteristics, IC characteristics, and the Sensor Sensitivity user module parameter.
- The raw count value from the capacitance converter circuitry is analyzed in runtime by API functions to make sensor state decisions and to compensate for environmental changes.
- Background scanning algorithms that enable the CPU to perform other tasks while the scan is in progress in hardware.
- In the case of consecutive, dependent sensors (for example, sliders and touchpads), API functions are given to interpolate a position with greater resolution than the physical pitch of the sensors.
- High-level software functions accommodate slider diplexing so that one I/O pin can be routed to two physical sensors. This reduces by half the number of I/O pins consumed for a specific number of slider elements.
- Immunity to external noise improvement algorithms.

Recommended Reading

Cypress recommends reading the following documents before implementing a CapSense design using the SmartSense2X_EMC User Module. These documents are available at the Cypress website: www.cypress.com.

- Getting Started with CapSense
- CY8C20xx6A/H CapSense Design Guide
- CY8C21x34/B CapSense Design Guide
- CY8C20x34 CapSense Design Guide
- CY8CMBR2044 CapSense Design Guide

Chip Level View

The CY8C22x45 family of devices has a built-in analog bus. It allows capacitive sensor connections to any PSoC pin. The SmartSense2X_EMC User Module uses internal precharge switches to charge active sensors at clock signal phase Ph1, and connects the analog bus to the sensor at phase Ph2. The sigma-delta modulation capacitor and comparator inputs are connected to the analog bus permanently. The firmware performs sensor scanning in series by setting corresponding bits in the MUX_CRx registers.

The following pins are used for SmartSense2X_EMC:

- Sensor pins
- External modulation capacitor (Cmod) pin P0[5] or P0[7] (single channel only)
- Shielding electrode pins

Pins and Routing

The dual-channel configuration for the CY8C22x45 device has Left Shield Electrode Output and Right Shield Electrode Output properties for the left and right channels respectively. The shield electrode, if enabled, is routed to P1[6] or P3[6] for the left channel, or P1[7] or P3[7] for the right channel in the SmartSense2X_EMC Wizard. On devices with fewer pins, P3[6] and P3[7] may be unavailable. For a single channel, the connection depends on whether the right or left CapSense channel is used. The drive mode for the selected pin should be set to Strong.

Placement

The SmartSense2X_EMC Multi User Module (MUM) has two selection windows:

- First selection window:

  - Single-channel with IDAC configuration: This configuration uses one analog/compare column block. The analog mux bus left and the analog mux bus right are connected. No external Rb is required. IDAC is used for charging Cmod and Cp.

  - Dual-channel with IDAC configuration: This configuration uses two analog/compare column blocks. The analog mux bus left and the analog mux bus right scan the sensors respectively. No external Rb is required.

- Second selection window (after selecting first):

  - CapSense0 block: This configuration uses one analog/compare column block ACE02 and a Capsense0 block.

  - CapSense1 block: This configuration uses one analog/compare column block ACE03 and a Capsense1 block.

Figure 4. SmartSense2X_EMC MUM: First Selection Window

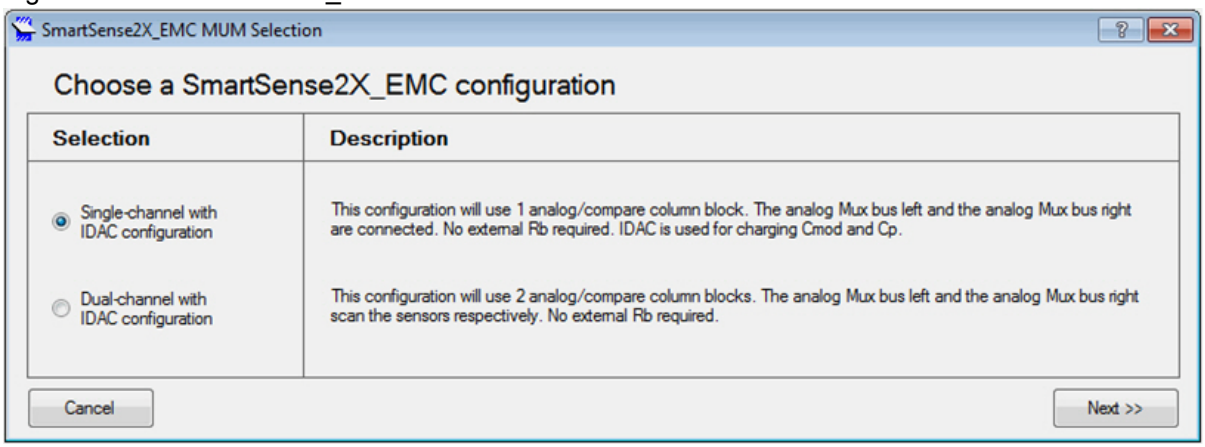
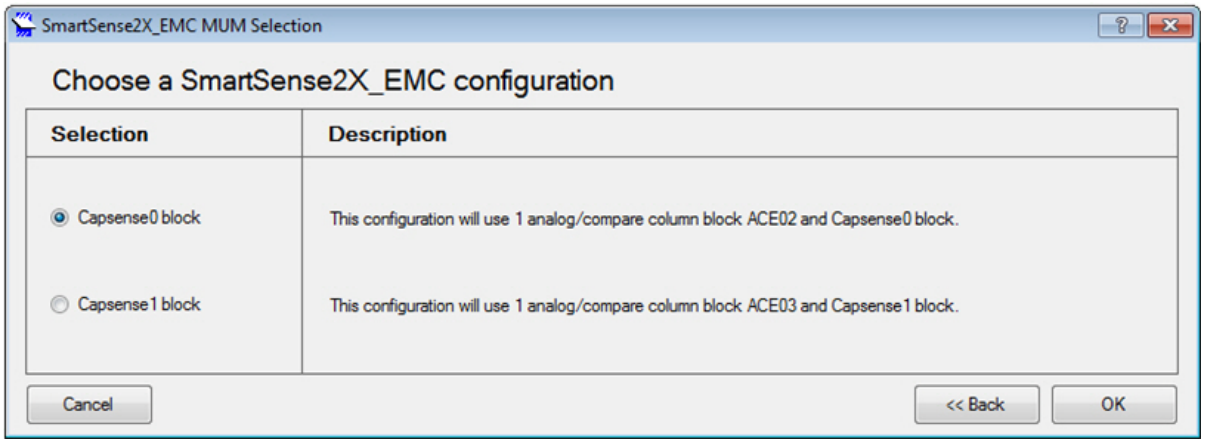


Figure 5. SmartSense2X_EMC MUM: Second Selection Window



The CY8C22x45 SmartSense2X_EMC MUM has two configurations. The following table lists the abbreviations for these configurations.

Configuration	Abbreviation
Single Channel with IDAC configuration on Capsense0	SmartSense2X_EMC_SGLI22
Single-Channel with IDAC configuration on Capsense1	SmartSense2X_EMC_SGRI22
Dual Channel with IDAC	SmartSense2X_EMC_DBLI22

Block Resources

The blocks for the user module are automatically placed when the user module is instantiated; alternate placements are available for single-channel configurations only. The SmartSense2X_EMC User Module consumes the CapSense block and one Comparator block.

User modules that require specific pin resources, including the LCD and I2CHW, must be placed before starting the SmartSense2X_EMC Wizard to establish pin connections for the SmartSense2X_EMC User Module. Avoid P1[0] and P1[1] when placing capacitive sensor connections. These pins are used for programming the part and may have excess routing capacitance affecting sensor sensitivity and noise. The following figures illustrate the block resources for single and dual channel respectively.

Figure 6. Block Resources - SmartSense2X_EMC Single Channel

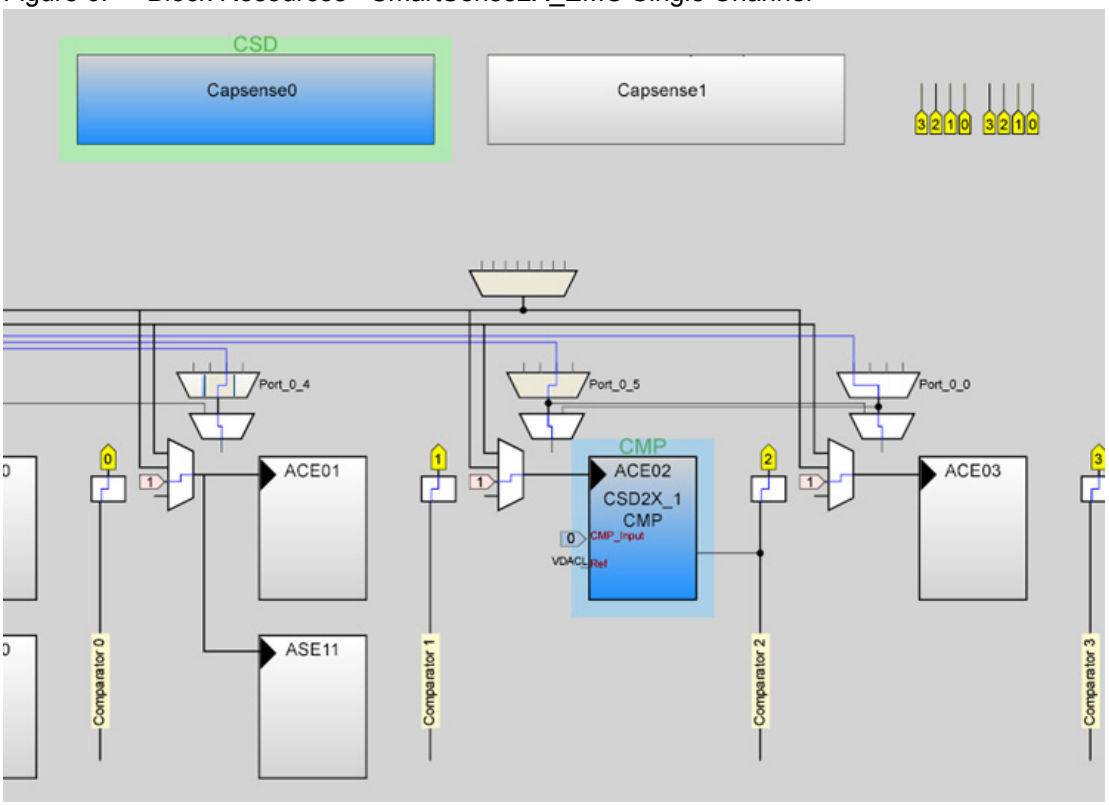
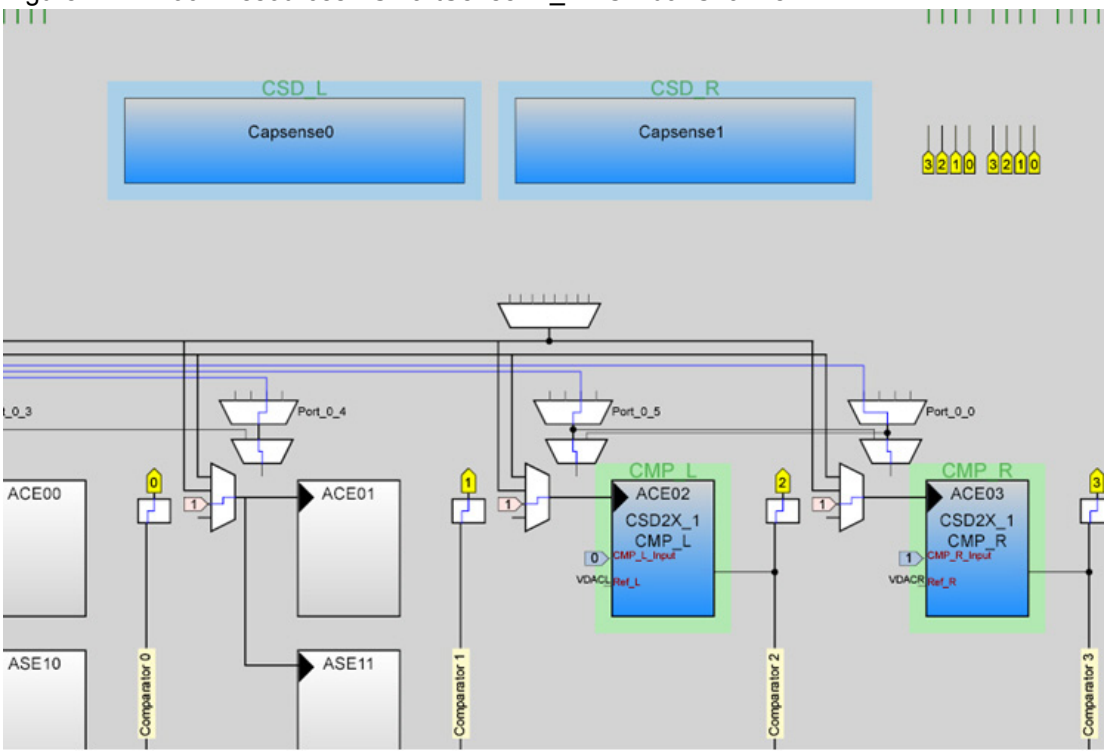


Figure 7. Block Resources - SmartSense2X_EMC Dual Channel



SmartSense2X_EMC Wizard

The following diagrams show the GUI of the SmartSense2X_EMC User Module wizard.

Figure 8. SmartSense2X_EMC Single-Channel Configuration Wizard

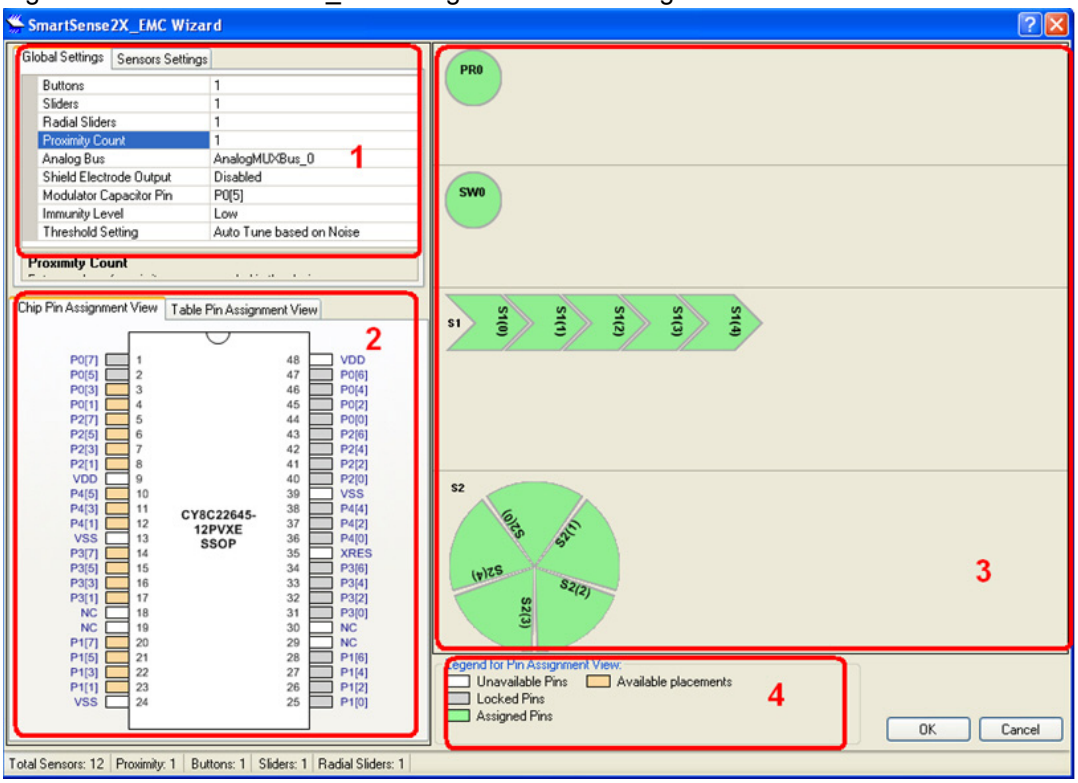
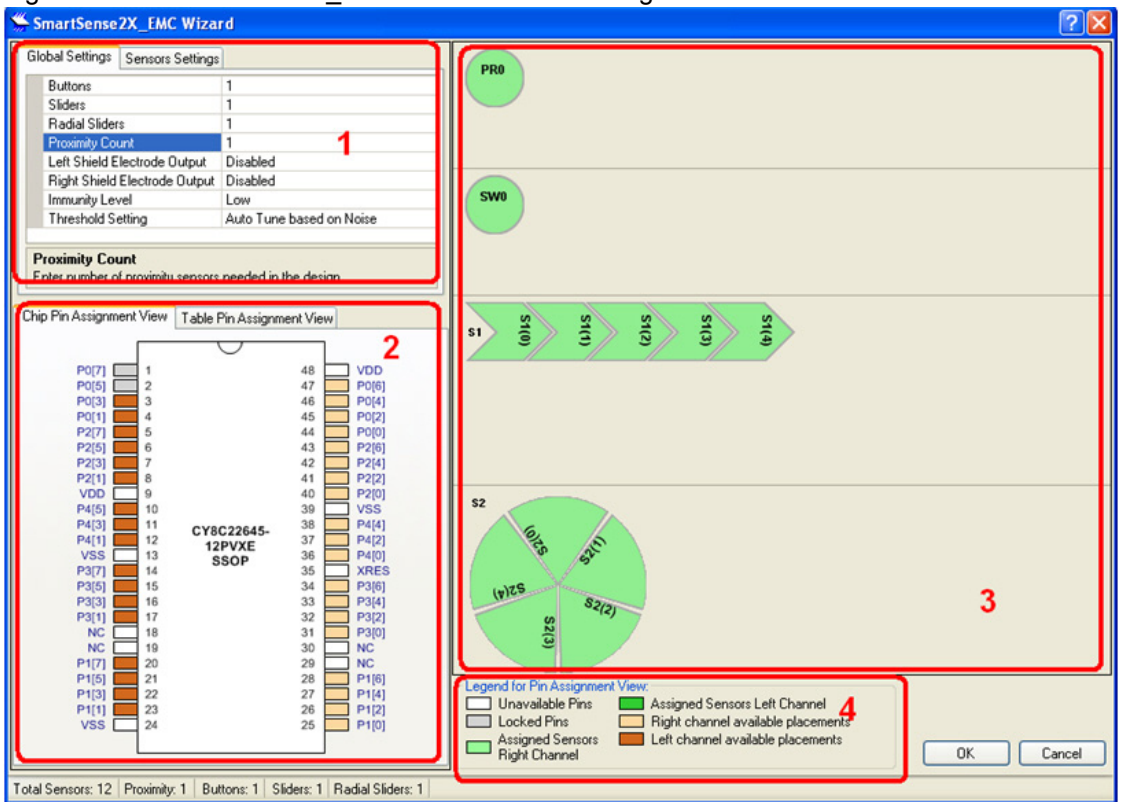


Figure 9. SmartSense2X_EMC Dual Channel Configuration Wizard



Wizard Parameters Description

Wizard parameters are described in the Global Settings Tab and Sensor Settings Tab sections.

Global Settings Tab

The Global Settings tab consists of the following parameters: Buttons, Sliders, Radial Sliders, Proximity Count, Modulator Capacitor Pin, Immunity Level, Threshold Setting, and Analog Bus.

Figure 10. Single Channel Configuration Global Settings

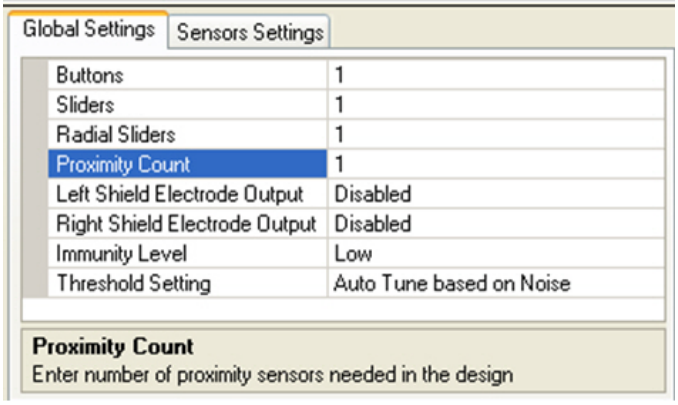
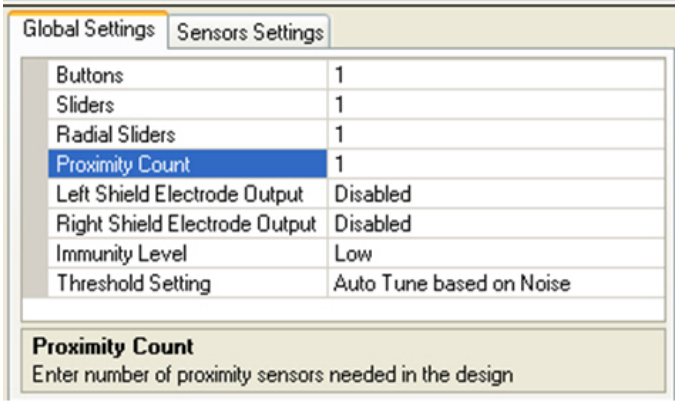


Figure 11. Dual Channel Configuration Global Settings

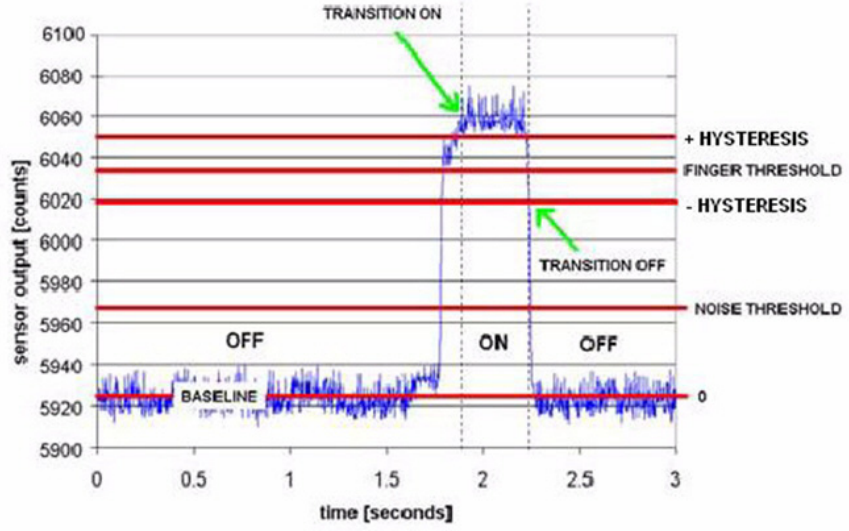


Buttons

The Buttons parameter defines the number of single sensors. The maximum number is 37 capacitive sensors. It is restricted by the number of available device port pins. CapSense buttons are analogous to mechanical push-buttons. They are used for discreet controls such as on/off switches, function keys, menu keys, and so on. API functions monitor the capacitance signals from each sensor and compare them to threshold levels, calculated by the SmartSense2X_EMC auto-tuning algorithms. When a sensor is touched, its capacitance signal increases; if the increase is enough as determined by the SmartSense2X_EMC decision logic, then that sensor is activated. Figure 12 shows a typical signal (blue line) from a sensor when it is activated. SmartSense2X_EMC automatically sets the

thresholds (red lines) based on the user input to provide the desired system behavior. In addition, the Manual Mode is accessible, which enables setting the Finger Threshold for each sensor.

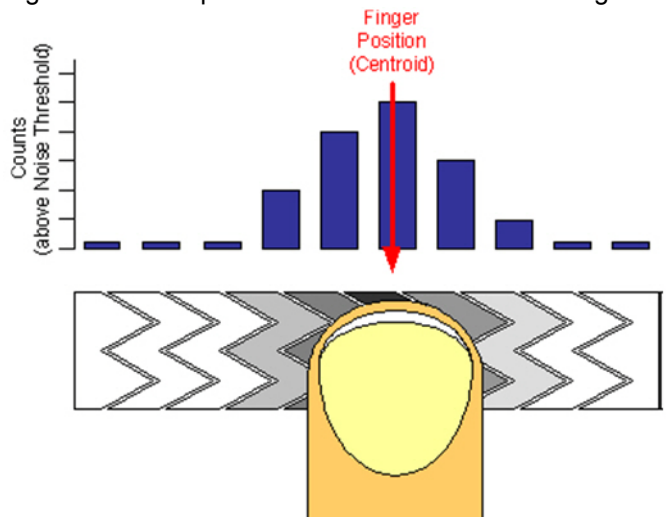
Figure 12. Capacitance Signal from a Sensor when it is Activated



Sliders

The Sliders parameter defines the number of linear sliders. The maximum number of sliders is defined by the maximum number of sensors. In addition, it is restricted by the available number of device port pins. A CapSense slider is implemented with an array of adjacent sensors. When a slider is actuated by a finger, several adjacent sensors register an increase in capacitance signal (see the following figure). The exact position of the touch is found by computing the centroid location of the set of activated sensors. The minimum number of sensors in a slider is restricted to two but the practical minimum number of sensors in a slider is five (default value), and the maximum is limited only by the number of available I/O pins on the PSoC device.

Figure 13. Interpolated Centroid Position of a Finger on a Slider

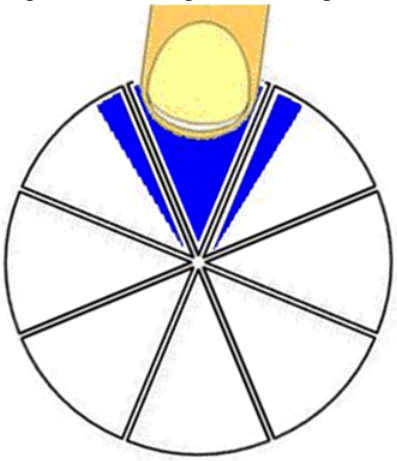


Radial Sliders

The Radial Sliders parameter defines the number of radial sliders. The maximum number of sliders is defined by the maximum number of sensors. It is also restricted by the available number of device port pins. The maximum slider count is equal to the maximum available pins on the selected part

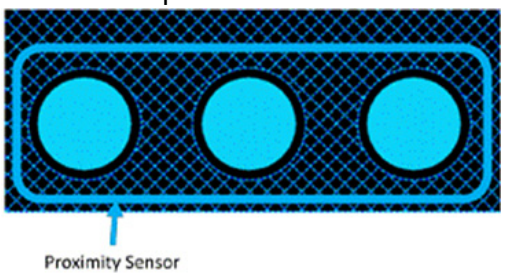
divided by 2 (2 is the minimum value of sensors, which can be set for one slider). SmartSense2X_EMC supports two slider types: linear and radial. Linear sliders have a beginning and an end, whereas radial sliders do not. In either case, when a touch occurs, the centroid algorithm takes into account signal from sensors adjacent to the sensor with the largest signal to interpolate the exact position of the touch. Radial sliders are not diplexed. The resolution is limited to (number of pins used for sensors - 1) x 2^8 - 1 or (2 x pins used for sensors - 1) x 2^8 - 1 for diplexed sliders.

Figure 14. Finger Touching Radial Slider



Proximity Count

A proximity sensor detects the presence of a hand or another conductive object before it makes contact with the touch surface. For example, imagine a hand stretched out to operate a car audio system in the dark. The proximity sensor causes the buttons of the audio system to glow using back-light LEDs when your hand is near. One implementation of a proximity sensor consists of a long trace on the perimeter of the user interface, as shown in the following figure.



Shield Electrode Output

This parameter enables or disables the algorithm that supports the optional shield electrode. The dual-channel configuration has Left Shield Electrode Output and Right Shield Electrode Output properties for left and right channels accordingly. The shield electrode, if enabled, is routed to P1[6] or P3[6] for the left channel, or P1[7] or P3[7] for right channel in the SmartSense2X_EMC Wizard. On devices with fewer pins, P3[6] and P3[7] may be unavailable. For a single channel, the connection depends on whether the right or left CapSense channel is used. The drive mode for the selected pin should be set to Strong.

Table 1. Single Channel Configuration Settings

Type	Enum
Range	Disabled, GOO[6] to P1[6], GOO[6] to P3[6]
Default	Disabled

Table 2. Dual Channel Configuration Settings

Type	Enum
Left Shield Electrode Output	
Range	Disabled, GOO[6] to P1[6], GOO[6] to P3[6]
Default	Disabled
Right Shield Electrode Output	
Range	Disabled, GOO[7] to P1[7], GOO[7] to P3[7]
Default	Disabled

Modulator Capacitor Pin

SmartSense2X_EMC requires an external modulation capacitor, Cmod, connected from Vss to one of the two dedicated PSoC pins, P0[5] or P0[7]. The SmartSense2X_EMC dual-channel configuration requires two external modulation capacitors that must be connected to P0[5] and P0[7] pins (Modulator Capacitor Pin property is not available and pins P0[5] and P0[7] are locked).

The selected pins must not be used for any other purpose. The recommended value for the external modulation capacitor is 2.2 nF. A ceramic capacitor must be used. The temperature capacitance coefficient is not important. Cypress recommends using a 560-ohm series resistor on all CapSense sensor traces for RF interference suppression. This resistor must be placed as close to the PSoC device as possible.

Threshold Setting

Selects between automatic and manual threshold setting. The options for threshold setting are:

- Manual
  - All the thresholds are set manually with values ranging from 1 to 255. In this case, the Finger Threshold parameter should be adjusted for each sensor in the Sensor Settings Tab. The Hysteresis, NoiseThreshold, NegativeNoiseThreshold, BaselineUpdateThreshold, and LowBaselineReset values are fixed and calculated using the Finger Threshold parameter.

- Auto Tune based on Sensitivity
  - In this option the user module sets the finger threshold to the sensitivity candidate computed during SmartSense initialization. The sensitivity candidate is calculated in the same manner as the CY8C20xx6 SmartSense.

- Auto Tune based on Noise
  - Full auto-tuning ensures a minimum of 5:1 SNR – the user module implements a full-fledged SmartSense algorithm to update finger threshold during run time. The Auto Tune based on

Noise option is available for selection only if the Immunity Level parameter is selected as Low. The default setting is Auto Tune based on Noise.

Analog Bus

This parameter defines one or two analog mux uses that are used. If AnalogMUXbus_0 is selected, then only sensors that are connected to the odd pins (with the exception of P0[7]) are scanned. The default setting is to use AnalogMUXbus_0 analog bus. If AnalogMUXbus_0 is selected, then two analog mux buses are not connected. This gives an opportunity to the other user modules to use AnalogMUXbus_1 without conflicts.

Table 3. Analog Bus Parameters

Type	API
Range	AnalogMUXbus_0 Both
Default	AnalogMUXbus_0

1. Available for single channel configurations only.

2. This parameter is introduced to enable using the SmartSense2X_EMC User Module with other UMs that require analog mux bus.

User modules (such as AMuxN) that occupy the same analog mux bus may have a conflict with the SmartSense2X_EMC User Module when they are used at the same time. If a simultaneous operation is needed, then the user modules should use the pins that belong to different analog mux buses.

Immunity Level

This parameter defines Immunity level. The selectable options are Low, Medium, and High. A higher immunity level improves the performance in a high-noise environment but increases the memory use and, therefore, decreases the maximum number of supported sensors. The sliders are disabled if the Immunity level is set to Medium.

Sensor Settings Tab

The Sensor Settings Tab consists of the following parameters: Diplex, Finger Threshold, Resolution, Sensitivity Level, and Sensors Count.

Figure 15. Sensor Settings for Buttons -Threshold Setting = Auto Tune based on Noise

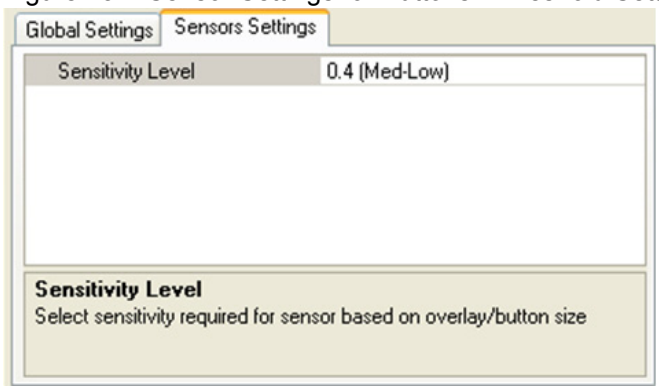


Figure 16. Sensor Settings for Buttons - Threshold Setting = Auto Tune based on Sensitivity

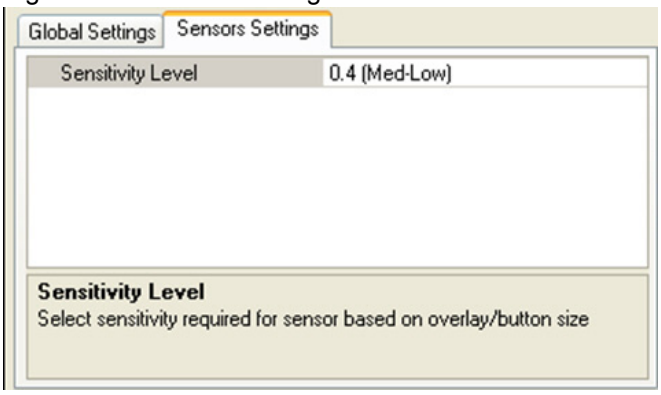


Figure 17. Sensor Settings for Buttons - Threshold Setting = Manual

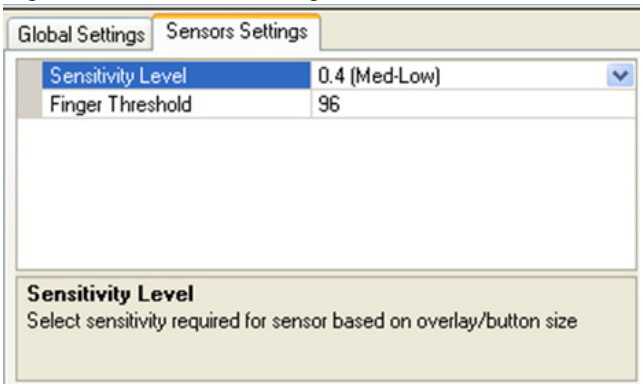


Figure 18. Sensor Settings for Proximity Sensors - Threshold Setting = Auto Tune based on Noise

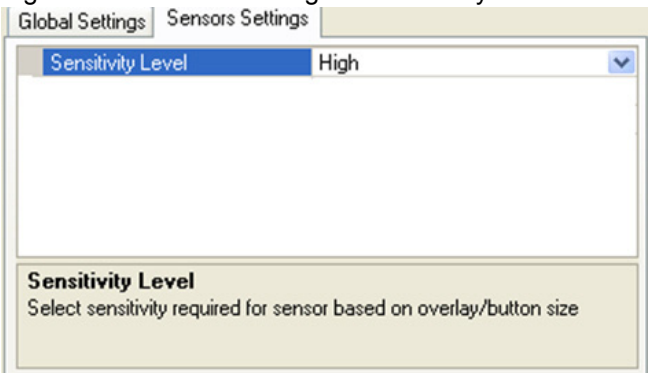


Figure 19. Sensor Settings for proximity sensors -Threshold Setting = Auto Tune based on Sensitivity

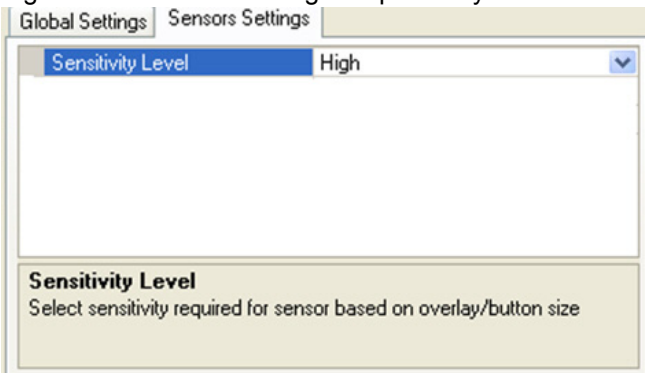


Figure 20. Sensor Settings for proximity sensors -Threshold Setting = Manual

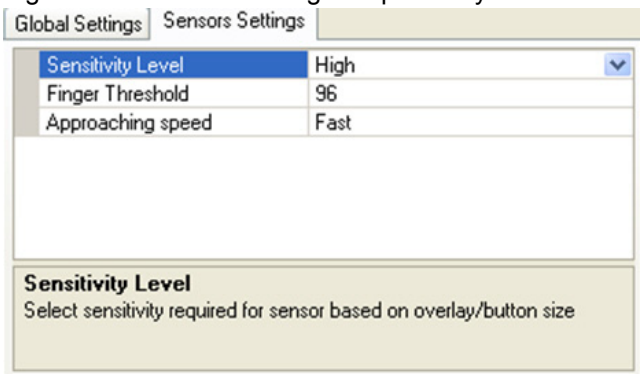


Figure 21. Sensor Settings for linear sliders -Threshold Setting = Auto Tune based on Noise

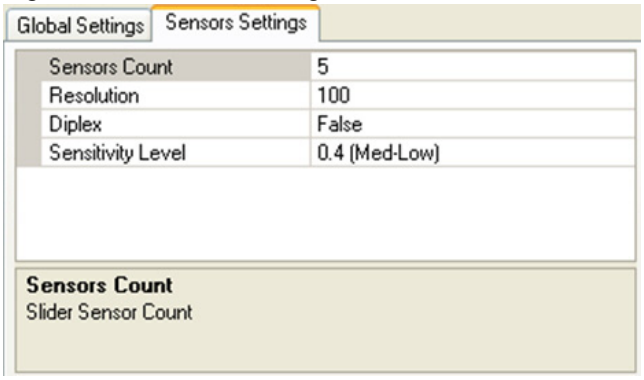


Figure 22. Sensor Settings for linear sliders -Threshold Setting = Auto Tune based on Sensitivity

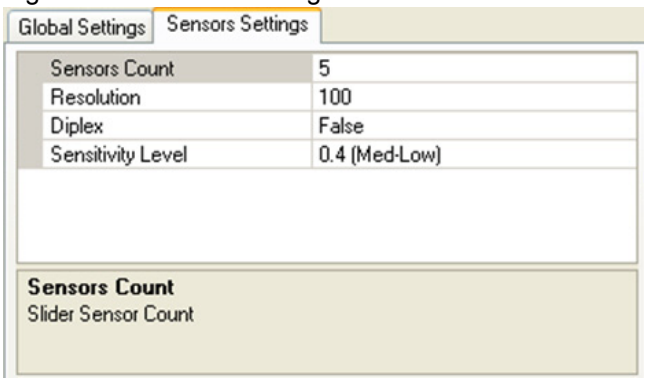


Figure 23. Sensor Settings for linear sliders -Threshold Setting = Manual

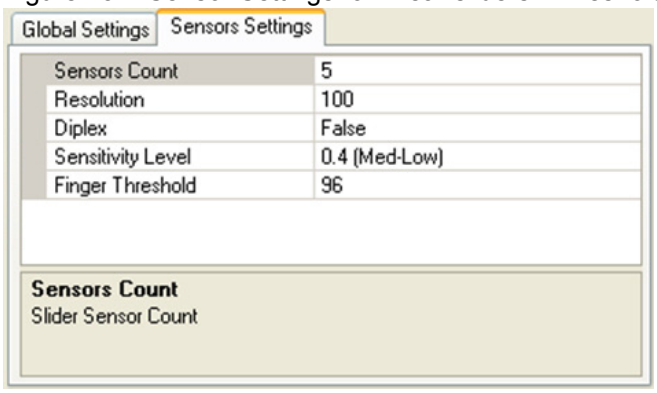


Figure 24. Sensor Settings for Radial Sliders - Threshold Setting = Auto Tune based on Noise

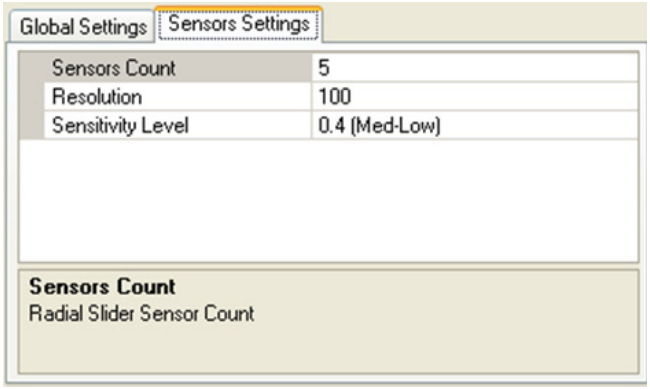


Figure 25. Sensor Settings for Radial Sliders - Threshold Setting = Auto Tune based on Sensitivity

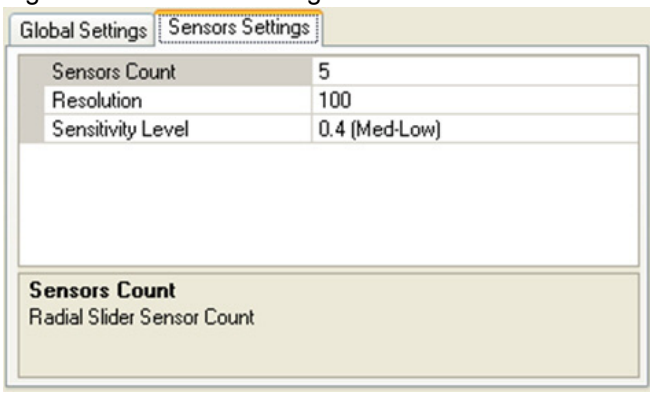
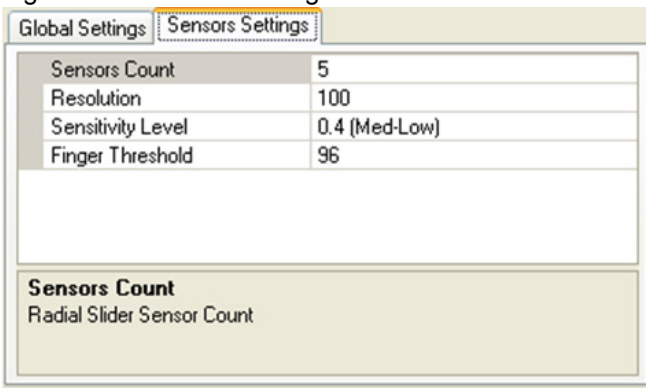


Figure 26. Sensor Settings for Radial Sliders - Threshold Setting = Manual



Finger Threshold

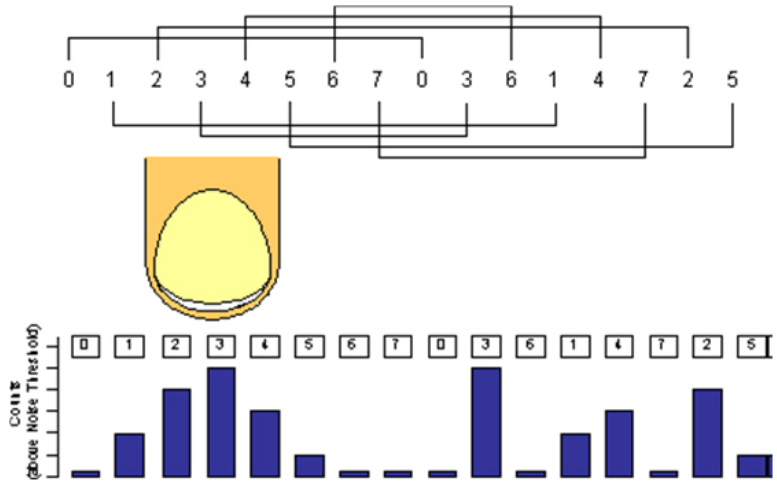
This threshold is used to determine the state of each button sensor. If any sensor is active, the bIsAnySensorActive() function returns a 1. If all sensors are off, the bIsAnySensorActive() function returns a 0. Possible values range from 1 to 255. The default value is 80. The Finger Threshold is visible only if the 'manual' mode is selected for the Threshold Setting global parameter.

The Finger Threshold property is disabled when "Threshold Setting" is set to "Auto Tune based on Sensitivity" or "Auto Tune based on Noise".

Diplex

This option enables or disables the slider diplex. This parameter is available for linear sliders only. When diplexing is used, each pin on the PSoC device that is designated as a slider element is mapped to two physical locations in the array of slider sensors. The first (or numerically lower) half of the physical locations is mapped according to the port pin assigned in the SmartSense2X_EMC Wizard. The second (or upper) half of the physical sensor locations are automatically mapped using the pattern shown in the following figure.

Figure 27. Indexing of Diplexed Slider Array by SmartSense2X_EMC



The close proximity of strong signals in the lower half of the slider results in the same levels aliased into the upper half. However, in the upper half, the results are scattered and non-contiguous. The centroid algorithm searches for strong adjacent sets of signals to declare the resolved slider position. The pattern used for mapping the upper half sensors ensures that a valid signal pattern in one half does not result in a valid signal pattern in the other half, as shown in Figure 27.

Take care to ensure the mapping of sensors to pins on the PCB matches the ‘Index by 3’ sequence used by the diplexing algorithm. The capacitance of sensor pairs in a diplexed slider must be reasonably well-matched (within 10 pF). The diplex sensor index table is automatically generated by the SmartSense2X_EMC Wizard when you select diplexing. The following table shows the diplexing sequences for up to 56 slider segments diplexed into 28 PSoC I/O pins.

Table 4. Diplexing Sequence for Different Slider Segment Counts

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

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

Resolution

This option sets the sensor resolution in the range from 5 to (number of pins used for sensors – 1) × 2^8 – 1 or (2 × number of pins used for sensors – 1) × 2^8 – 1 for diplexed sliders. This parameter is available for linear and radial sliders only.

Sensors Count

Sensors Count is the number of physical sensors in the slider or radial slider. This parameter is available for linear and radial sliders only. The default value is 5 and the minimum value is 2. The maximum value is defined by maximum available sensors in the device.

Sensitivity Level

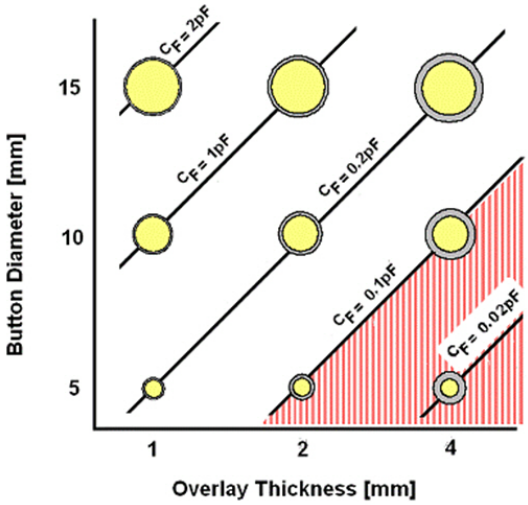
Sensitivity Level sets the capacitance signal change (sensor response) in pF that is needed to activate a button sensor. The available settings are 0.1 (High), 0.2 (Med-High), 0.3 (Medium), 0.4 pF (Med-Low), and 0.5 pF (Low). The default setting is 0.5 pF. 0.1 pF corresponds to the highest sensitivity offered and 0.5 pF corresponds to the lowest sensitivity.

Sensitivity Level for proximity sensors has Low and High values, and the default is set to High.

The following figure shows the relationship between the button size, overlay thickness (acrylic plastic), and sensor response (CF), and can be used as a guide for setting Sensor Sensitivity. The Sensor Sensitivity should always be set at or below the sensor response indicated in the following figure. This

ensures robust operation. Note that the area shaded in red must be avoided, because the sensor response is below the minimum 0.1 that can be detected by SmartSense2X_EMC.

Figure 28. Relationship Between Button Size, Overlay Thickness, and Sensor Response in pF



Approaching Speed

This parameter decides the rate at which the baseline of the proximity sensor is updated. It has the following values: slow, medium, and fast, and the default setting is Fast. This parameter is available if the Threshold Setting parameter is set to Manual.

Pin Assignment View

The Pin Assignment view allows assigning switches or sensors to pins by dragging the switch or sensor to the pin in the device. You can choose to drag switches or sensors to pins in the Chip Pin Assignment View or the Table Pin Assignment View. The port pin becomes green after it is assigned and is no longer available. Change sensor assignments by dragging the port pin back to the uncommitted table. Make sure to avoid selecting pins already committed to other user modules.

To change the pin assignment, place your cursor on the assigned pin, click the pin, and drag and drop it outside the switches box. The pin is now unassigned and you can reassign it.

When you right-click the wizard "Chip Pin Assignment View" and "Table Pin Assignment View", the "Clear All Pins" option appears. This option unassigns all chip pins.

Sensor Elements Area

The Sensor Elements area consists of Buttons View, Sliders View, and Radial Sliders View.

Buttons View

This view of the SmartSense2X_EMC Wizard shows the CapSense buttons sensors. These sensors can be assigned to pins by dragging the sensor to the pin in the Pin Assignment View.

Sliders View

This view of the SmartSense2X_EMC Wizard shows the linear sliders. Each slider sensor can be assigned to a pin by dragging the sensor to the pin in the Pin Assignment View.

Radial Sliders View

This view of the SmartSense2X_EMC Wizard shows the radial sliders. Each slider sensor can be assigned to a pin by dragging the sensor to the pin in the Pin Assignment View.

Pin Legend Area

The legend explains the colors on the pins.

White – The pin cannot be used as a CapSense input.

Gray – The pin is locked. There are two possible causes for this. The first possibility is that another user module, such as the LCD or I^2C, has claimed the pin. The second possibility is that the name of the pin was changed from its default. To return the pin name to its default, expand the pin in the Pinout view, and select **Default** from the **Select** menu. The pin is now available for assignment in the Wizard.

Orange – The pin is available for assignment.

Green – The pin has been assigned as a CapSense input.

Status Bar

The Status bar shows the following information: total sensors count, number of buttons, number of sliders, and number of radial sliders.

OK Button

The OK button stores the SmartSense2X_EMC configuration. Based on your entries for sensor count, pin assignment, diplexing, and resolution, a set of tables is generated. The tables are located in SmartSense2X_EMC.asm and SmartSense2X_EMCHL.asm.

Sensor Table

The Sensor table is located in the *SmartSense2X_EMC.asm* and consists of a 2-byte entry for each sensor. The first byte is the port number and the second byte is the bit mask for the bit (not the bit number). An example for a table with six sensors is as follows:

```
SmartSense2X_EMC_Sensor_Table:
_ SmartSense2X_EMC_Sensor_Table
dw 0x0140 // Port 1 Bit 6
dw 0x0301 // Port 3 Bit 0
dw 0x0304 // Port 3 Bit 2
dw 0x0308 // Port 3 Bit 3
dw 0x0302 // Port 3 Bit 1
dw 0x0108 // Port 1 Bit 3
```

Group Table

The Group table defines each of the groups of button sensors or sliders. There is one entry for each slider and one for the independent button sensors. The first entry is always the independent buttons. Each entry is six bytes. The first byte is the index in the Sensor Table where the group starts. The second byte is the number of sensors in that group. The third byte signifies whether the slider is diplexed or not (4 is diplexed, 0 is not diplexed). The fourth, fifth, and sixth bytes are the fixed point multipliers by which the slider's centroid is scaled to achieve the resolution specified in the SmartSense2X_EMC wizard.

```
SmartSense2X_EMC_Group_Table:
_ SmartSense2X_EMC_Group_Table:
; Group Table:
```

```
; Origin Count Diplex? DivBtwSw(wholeMSB, wholeLSB, fractByte)
db 0x0, 0x3,    0x00,    0x00, 0x00, 0x00 ; Buttons
db 0x3, 0x8,    0x4,      0x0,   0x0,   0x44 ; Slider 1
```

Diplex Table

The Diplex table scan order data is produced for a group that is a slider and with diplexing enabled. Otherwise, a label is created, but no data is placed. The table consists of two parts: sensor mapping for each slider, and a reference for each separate slider to its table. A typical example for an eight-sensor slider is shown here:

```
DiplexTable_0:
; This group is not a diplexed slider
DiplexTable_1:
 db 0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5// 8 switch slider

 SmartSense2X_EMC_Diplex_Table:
_SmartSense2X_EMC_Diplex_Table:
 db >DiplexTable_0, <DiplexTable_0
 db >DiplexTable_1, <DiplexTable_1
```

Order Table

The Order Table consists of one byte for each sensor. This byte is the left sensor order in the raw counts result array without channel dependence. The table is generated by the wizard and reflects the sensor order.

```
SmartSense2X_EMC_Order_Table_Left:
_SmartSense2X_EMC_Order_Table_Left:
DB 0x01 // Position 1
SmartSense2X_EMC_Order_Table_Right:
_SmartSense2X_EMC_Order_Table_Right:
DB 0x00,0x02 // Position 0 and 2
```

Finger Threshold Table

The Finger Threshold table defines the normalized finger threshold for each sensor. A typical example for a one-button and five-sensor slider is shown here.

```
SmartSense2X_EMC_Finger_Threshold_Table:
_ SmartSense2X_EMC_Finger_Threshold_Table:
db 255, 80, 80, 80, 80, 80
```

Sensitivity Level Table

The Sensitivity Level Table defines the sensor sensitivity for each sensor. A typical example for a one-button and five-sensor slider is shown here.

```
_Sensitivity_Level_Table:
SmartSense2X_EMC_Sensitivity_Level_Table:
_ SmartSense2X_EMC_Sensitivity_Level_Table:
db, 4, 4, 4, 4, 4, 4
```

Table 5. Sensitivity Level Table Values for Buttons and Sliders

Parameter Value	Table Value
0.1 (High)	1
0.2 (Med-High)	2
0.3 (Medium)	3
0.4 pF (Med-Low)	4
0.5 pF (Low)	5

Table 6. Sensitivity Level Table Values for Proximity Sensors

Parameter Value	Table Value
High	1
Low	5

Approaching Speeds Table

The Approaching Speeds Table defines the approaching speeds for each proximity sensor. This table also has optional default values (equal to 2) for buttons and sliders that are intended to simplify the ASM code. A typical example is shown here:

```
SmartSense2X_EMC_Approaching_Speeds:
_ SmartSense2X_EMC_Approaching_Speeds:
db 2, 2, 1, 0, 2
```

Table 7. Approaching Speeds Table Values for Buttons and Proximity Sensors

Parameter Value	Table Value	Notes
Slow	1	
Medium	0	
Fast	2	Buttons and slides have this value too.

Parameters and Resources

After completing configuration and I/O pin assignment in the SmartSense wizard, set the user module parameters. Note that for any user module parameter change to take affect, the project must be regenerated.

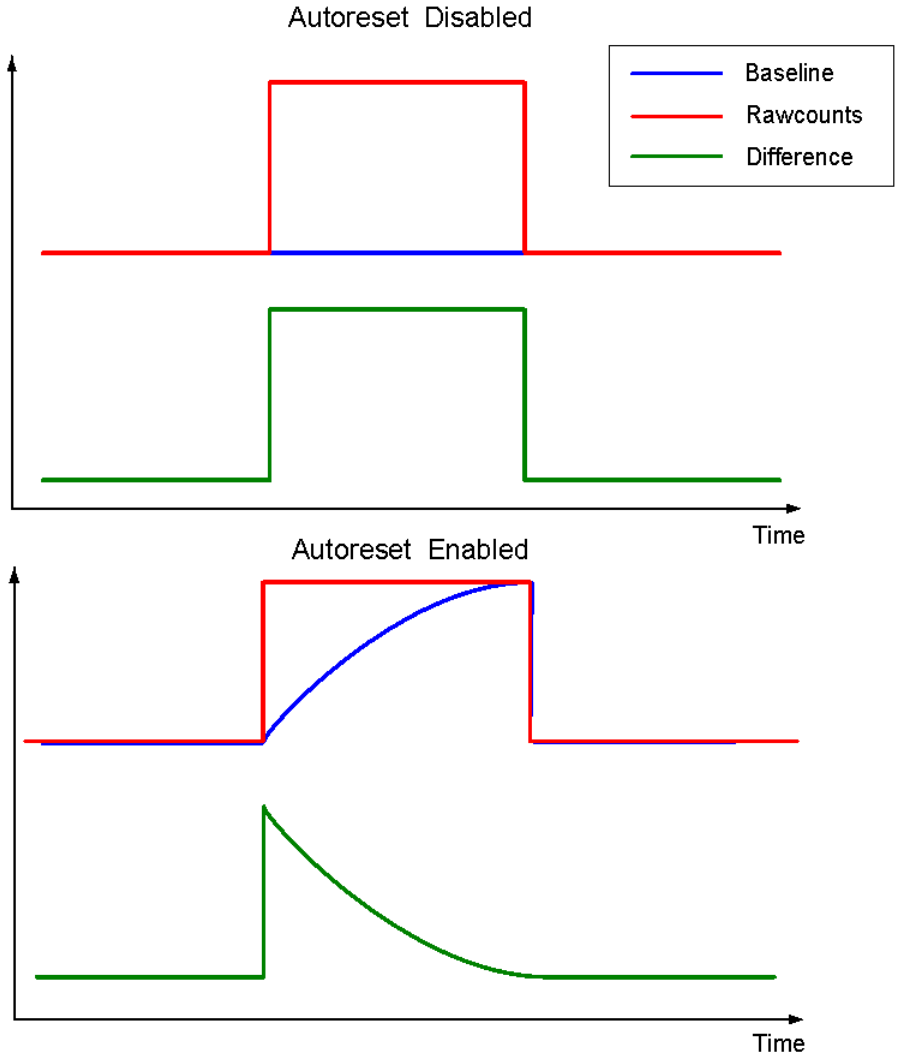
Sensors Autoreset

This parameter determines whether the baseline is updated at all times or only when the signal difference is below the Noise Threshold. The default value for this parameter is **Disabled**, that is, the baseline is updated only when the difference between the raw count and the baseline is below the Noise Threshold. Figure 27 illustrates this parameter's effect on baseline update. When Sensors Autoreset is set to **Enabled**, the baseline is always updated without regard to Noise Threshold. This limits the maximum activated time of sensors (typically to 5 - 10 s). However, this provides the benefit of preventing sensors from getting stuck due to sudden rises in raw counts that are not caused by a

touch. Such sudden rises can be caused by a large power supply voltage fluctuation, a high energy RF noise source, or rapid temperature change.

When Sensors Autoreset is Disabled, the baseline is updated only when the difference between raw count and baseline is below the Noise Threshold. This parameter should generally be left in its default "Disabled" state. See the Appendix section for additional explanation of this parameter.

Figure 29. Effect of the Sensor Autoreset Parameter on Baseline Update



Type	API
Range	Disabled, Enabled
Default	Disabled

Debounce

This parameter adds a debounce counter to the sensor active transition. For a sensor to transition from inactive to active, the difference count value must stay above finger threshold plus hysteresis for the number of samples specified by this parameter. The debounce counter is incremented by the bIsSensorActive or bIsAnySensorActive API functions.

Possible values are 1 to 255. A setting of '1' has no debounce, but gives the fastest response. The default setting is 3.

Type	API
Range	1-255
Default	3

Background Scanning

This parameter enables and disables background scan. The default setting is ‘Disable’. If it is set to Enable, then background scanning is implemented.

Type	API
Range	Enable/Disable
Default	Disable

FMEA_Shorts_Test

This parameter enables/disables the FMEA GND, VDD, and Sensor short tests.

Type	ENUM
Range	Enable/Disable
Default	Disable

FMEA_Cmod_Test

This feature will enable the test for Cmod and Rb calculation.

Type	ENUM
Range	Enable/Disable
Default	Disable

Application Programming Interface

The Application Programming Interface (API) functions are provided as part of the user module to enable you to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the SmartSense_1 to the first instance of this user module in a particular project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable, and constant symbol. In the following descriptions the instance name has been shortened to SmartSense for simplicity.

Note ** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must also ensure their code observes the policy. Though some user module API functions may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Entry Points are supplied to initialize the SmartSense, start it sampling, and stop the SmartSense. In all cases, the instance name of the module replaces the SmartSense prefix shown in the following entry points. Failure to use the correct instance name is a common cause of syntax errors.

API functions use different global arrays. Do not alter these arrays manually. However, you can inspect these values for debugging purposes. For example, you can use a charting tool to display the contents of the arrays. Here are some of the global arrays:

- SmartSense2X_EMC_waSnsBaseline[]
- SmartSense2X_EMC_waSnsResult[]
- SmartSense2X_EMC_waSnsDiff[]
- SmartSense2X_EMC_baSnsOnMask[]
- SmartSense2X_EMC_baDAC[]
- SmartSense2X_EMC_baCompensationDAC[]
- SmartSense2X_EMC_baDACCodeBaseline[]
- SmartSense2X_EMC_bScanComplete
- SmartSense2X_EMC_baSnSSignal[]

Note <Instance_Name> is SmartSense2X_EMC for all APIs and variables descriptions in this document.

SmartSense2X_EMC_waSnsBaseline[] – This is an integer array that contains the baseline data of each sensor. The array size is equal to the sensor count. The SmartSense_EMC_waSnsBaseline[] array is updated by these functions:

- SmartSense2X_EMC_UpdateAllBaselines()
- SmartSense2X_EMC_UpdateSensorBaseline()
- SmartSense2X_EMC_InitializeBaselines()

SmartSense2X_EMC_waSnsResult[] – This is an integer array that contains the raw data of each sensor. The array size is equal to the sensor count. The SmartSense2X_EMC_waSnsResult[] data is updated by these functions:

- SmartSense2X_EMC_ScanSensor()
- SmartSense2X_EMC_ScanAllSensors().

SmartSense_EMC_waSnsDiff [] – This is an integer array that contains the difference between the raw data and the baseline data of each sensor. The array size is equal to the sensor count.The SmartSense2X_EMC_waSnsDiff [] data is updated by these functions:

- SmartSense2X_EMC_UpdateAllBaselines()
- SmartSense2X_EMC_UpdateSensorBaseline()

SmartSense_EMC_baSnsOnMask[] – This is a byte array that holds the sensor on or off state (for buttons or sliders). SmartSense_EMC_baSnsOnMask[0] contains the masked bits for sensors 0 through 7 (sensor 0 is bit 0, sensor 1 is bit 1). SmartSense_EMC_baSnsOnMask[1] contains the masked bits for sensors 8 through 15 (if they are needed), and so on. This byte array contains as many elements as are necessary to contain all the placed sensors. The value of a bit is 1 if the button is on and 0 if the button is off. The SmartSense2X_EMC_baSnsOnMask[] data is updated by these functions:

- SmartSense2X_EMC_bIsSensorActive()
- SmartSense2X_EMC_bIsAnySensorActive()

SmartSense2X_EMC_baDAC [] – This is a byte array that contains the IDAC Register value for each sensor. The values from this array are copied to the SmartSense2X_EMC_bIdacValue variable before scan.

Smartsense2X_EMC_baCompensationDAC [] – This is a byte array that contains the Compensation IDAC Register value for each sensor. The values from this array are copied to the SmartSense2X_EMC_bCompensationIdacValue variable before scan.

SmartSense2X_EMC_baDACCodeBaseline [] – This table contains calibrated current values for each sensor, which can be changed in runtime. This can be useful in complex projects.

For double-channel configuration:

```
SmartSense2X_EMC_baDACCodeBaselineL:
_SmartSense2X_EMC_baDACCodeBaselineL:
BLK SmartSense2X_EMC_TotalLeftSensorCount
SmartSense2X_EMC_baDACCodeBaselineR:
_SmartSense2X_EMC_baDACCodeBaselineR:
BLK SmartSense2X_EMC_TotalRightSensorCount
```

For single-channel configuration:

```
SmartSense2X_EMC_baDACCodeBaselineL:
_SmartSense2X_EMC_baDACCodeBaselineL:
BLK SmartSense2X_EMC_TotalSensorCount
SmartSense2X_EMC_baDACCodeBaselineR:
_SmartSense2X_EMC_baDACCodeBaselineR:
BLK SmartSense2X_EMC_TotalSensorCount
```

SmartSense2X_EMC_bScanComplete [] – This variable is valid only when the background scanning feature is enabled. This variable should be set whenever sensor scanning is completed.

SmartSense2X_EMC_baSnSSignal[] – This is an integer array that contains the normalized signal for each sensor.

SmartSense2X_EMC_Start

Description:

Initializes registers and starts the user module. This function must be called before any other user module functions.

C Prototype:

```
void  SmartSense2X_EMC_Start()
```

Assembly:

```
lcall  SmartSense2X_EMC_Start
```

Parameters:

None

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_Stop

Description:

Restores the SmartSense2X_EMC block to its idle default configuration, releases the AMUX bus for other purposes, disables internal interrupts, and calls SmartSense2X_EMC_ClearSensors() to reset all sensors to their inactive state.

C Prototype:

```
void  SmartSense2X_EMC_Stop()
```

Assembly:

```
lcall  SmartSense2X_EMC_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_Resume

Description:

Resumes the user module operation after the SmartSense2X_EMC_Stop call.

C Prototype:

```
void SmartSense2X_EMC_Resume()
```

Assembly:

```
lcall SmartSense2X_EMC_Resume
```

Parameters:

None

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_ScanSensor(BYTE bSensor)

SmartSense2X_EMC_ScanSensor(BYTE bLeftSensor, BYTE bRightSensor)

Description:

Scans the selected sensors. Each sensor has a unique number within the sensor array. For single-channel configurations, this number is assigned by the CSD2X Wizard in sequence. For example, Sw0 is sensor 0, Sw1 is sensor 1, and so on.

For two-channel configurations, the sensor number is a value from 0 to Maximum Channel Sensor Number. For example, if there are two sensors on the left channel, their values are 0 and 1, respectively. If there are two sensors on the right channel too, their values are also 0 and 1, respectively. If a value of 0xFF is passed into this function as a sensor number, no sensor is scanned for that channel.

C Prototype:

In Singe Channel Configuration:

```
void SmartSense2X_EMC_ScanSensor(BYTE bSensor);
```

In Double Channel Configuration:

```
void  SmartSense2X_EMC_ScanSensor(BYTE bSensorLeft, BYTE bSensorRight);
```

Assembly:

In Singe Channel Configuration:

```
mov A,  bSensor
lcall SmartSense2X_EMC_ScanSensor
```

In Double Channel Configuration:

```
mov   A,  bSensorLeft
mov   X,  bSensorRight
lcall SmartSense2X_EMC_ScanSensor
```

Parameters:

In Singe Channel Configuration:

A => Sensor Number

In Double Channel Configuration:

A => Sensor Number Left

X => Sensor Number Right

Return Value:

None

Side Effects

**

SmartSense2X_EMC_ScanAllSensors

Description:

Scans all of the configured sensors by calling SmartSense2X_EMC_ScanSensor() for each sensor.

C Prototype:

```
void SmartSense2X_EMC_ScanAllSensors()
```

Assembly:

```
lcall  SmartSense2X_EMC_ScanAllSensors
```

Parameter:

None

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_ClearSensors

Description:

Clears all sensors to the non-sampling state by sequentially calling SmartSense2X_EMC_wGetPortPin() and SmartSense2X_EMC_DisableSensor() for each of the sensors.

C Prototype:

```
void SmartSense2X_EMC_ClearSensors()
```

Assembly:

```
lcall  SmartSense2X_EMC_ClearSensors
```

Parameter:

None

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_wReadSensor(BYTE bSensor)

Description:

Returns the key raw scan value in A (LSB) and X (MSB).

C Prototype:

```
WORD  SmartSense2X_EMC_wReadSensor(BYTE bSensor);
```

Assembly:

```
mov  A,  bSensor
lcall SmartSense2X_EMC_wReadSensor
```

Parameters:

A => Sensor Number

Return Value:

Scan value of sensor, LSB in A and MSB in X.

Side Effects:

**

SmartSense2X_EMC_wGetPortPin(BYTE bSensor)

Description:

Returns the port number and pin mask for a particular sensor. The passed parameter indexes and selects the data from the SmartSense2X_EMC_Sensor_Table[]. The return value can be passed to SmartSense2X_EMC_EnableSensor() and SmartSense2X_EMC_DisableSensor(). This function is only available in single-channel configurations.

C Prototype:

```
WORD  SmartSense2X_EMC_wGetPortPin(BYTE bSensor);
```

Assembly:

```
mov   A,  bSensor
lcall  SmartSense2X_EMC_wGetPortPin
```

Parameters:

bSensor - Sensor Number, the range is 0 to (n - 1) where 'n' is the total of the number of sensors set in the SmartSense2X_EMC Wizard plus the number of sensors included in sliders. The sensor number is used by SmartSense2X_EMC_wGetPortPin() to determine port and bit mask for the selected active sensor.

Return Value:

A => Sensor Bitmap

X => Port Number

Side Effects:

**

SmartSense2X_EMC_wGetPortPinLeft - Dual Channel

Description

Returns the port number and pin mask for a given sensor. The passed parameter indexes and selects the data from the SmartSense2X_EMC_Sensor_Table_Left[].The return value can be passed to the SmartSense2X_EMC_EnableSensor() and SmartSense2X_EMC_DisableSensor(). This function is available in the dual channel configurations only.

C Prototype

```
WORD SmartSense2X_EMC_wGetPortPinLeft(BYTE bSensor)
```

Assembly:

```
mov A, bSensor
lcall SmartSense2X_EMC_wGetPortPinLeft
```

Parameters:

bSensor – Sensor Number; the range is 0 to (n – 1) where 'n' is the total of the number of sensors set in the SmartSense2X_EMC Wizard for left channel (which includes number of slider segments connected to the left channel).The sensor number is used by SmartSense2X_EMC_wGetPortPinLeft() to determine the port and bit mask for the selected active sensor.

Return Value:

A => Sensor BitmapX => Port Number

Side Effects:

**

SmartSense2X_EMC_wGetPortPinRight - Dual Channel

Description

Returns the port number and pin mask for a given sensor that is connected to right channel.The passed parameter indexes and selects the data from SmartSense2X_EMC_Sensor_Table_Right[]. The return value can be passed to the SmartSense2X_EMC_EnableSensor() and SmartSense2X_EMC_DisableSensor(). This function is available in the dual channel configurations only.

C Prototype

```
WORD SmartSense2X_EMC_wGetPortPinRight(BYTE bSensor)
```

Assembly:

```
mov A, bSensor
lcall SmartSense2X_EMC_wGetPortPinRight
```

Parameters:

bSensor – Sensor Number; the range is 0 to (n – 1) where 'n' is the total of the number of sensors set in the SmartSense2X_EMC Wizard for the right channel (which includes number of slider segments connected to right channel).The sensor number is used by SmartSense2X_EMC_wGetPortPinRight() to determine the port and bit mask for the selected active sensor.

Return Value:

A => Sensor BitmapX => Port Number

Side Effects:

**

SmartSense2X_EMC_EnableSensor(BYTE bMask, BYTE bPort)

Description:

Returns the port number and pin mask for a given sensor. The passed parameter indexes and selects the data from the SmartSense2X_EMC_Sensor_Table[]. The return value can be passed to the SmartSense2X_EMC_EnableSensor(), SmartSense2X_EMC_DisableSensor(). This function is available in single-channel configurations only.

C Prototype:

```
void  SmartSense2X_EMC_EnableSensor(BYTE bMask, BYTE bPort);
```

Assembly:

```
mov   X,  bPort
mov   A,  bMask
lcall  SmartSense2X_EMC_EnableSensor
```

Parameters:

A => Sensor Bitmap

X => Port Number

Return Value:

None.

Side Effects:

**

SmartSense2X_EMC_DisableSensor(BYTE bMask, BYTE bPort)

Description:

Disables the sensor selected by the SmartSense2X_EMC_wGetPortPin() function. The drive mode is changed to Strong (001). This effectively grounds the sensor. The connection from the port pin to the AnalogMuxBus is turned off. The SmartSense2X_EMC_wGetPortPin() function returns the functional parameters.

C Prototype:

```
void  SmartSense2X_EMC_DisableSensor(BYTE bMask, BYTE bPort);
```

Assembly:

```
mov   X,  bPort
mov   A,  bMask
lcall  SmartSense2X_EMC_EnableSensor
```

Parameters:

A => Sensor Bitmap

X => Port Number

Return Value:

None.

Side Effects:

**

SmartSense2X_EMC_UpdateSensorBaseline(BYTE bSensorNum)

Description:

The historical count value, calculated independently for each sensor, is called the sensor's baseline.

This baseline is updated using the "Bucket Method".

The Bucket Method uses the following algorithm:

- Each time the SmartSense2X_EMC_UpdateSensorBaseline() is called, a difference count is calculated by subtracting the previous baseline from the raw count value. This difference is stored in the SmartSense2X_EMC_waSnsDiff[] array.
- If Sensors Autoreset is disabled, then each time the SmartSense2X_EMC_UpdateSensorBaseline() is called, the difference count is compared to the noise threshold. If the difference is below the noise threshold, it is accumulated into a virtual bucket. If the difference is above the noise threshold, the bucket is not updated. If Sensors Autoreset is enabled, the difference is accumulated into a virtual bucket regardless of the noise threshold parameter.
- After the accumulated difference counts in the virtual bucket reach the BaselineUpdateThreshold, the baseline is incremented by one and the bucket is reset to 0.

- If the difference count is below the noise threshold, the value held in the waSnsDiff[] array is reset to 0. As a result, this array does not contain elements with values greater than 0, but below the Noise-Threshold.

C Prototype:

```
void  SmartSense2X_EMC_UpdateSensorBaseline(BYTE bSensorNum);
```

Assembly:

```
mov   A,  bSensorNum
lcall   SmartSense2X_EMC_UpdateSensorBaseline
```

Parameters:

A => Sensor Number

Return Value:

None.

Side Effects:

**

SmartSense2X_EMC_UpdateAllBaselines

Description:

Uses the SmartSense2X_EMC_bUpdateSensorBaseline() function to update the baselines for all sensors.

C Prototype:

```
void  SmartSense2X_EMC_UpdateAllBaselines();
```

Assembly:

```
lcall  SmartSense2X_EMC_UpdateAllBaselines
```

Parameters:

None

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_bIsSensorActive(BYTE bSensorNum)

Description:

Checks the difference count array for a particular sensor in comparison to its finger threshold. The hysteresis value is taken into account. The hysteresis value is added or subtracted from the finger threshold based on whether the sensor is currently on. If it is active, the threshold is lowered. If it is inactive, the threshold is raised. This function also updates the sensor's bit in the SmartSense2X_EMC_baSnsOnMask[] array.

C Prototype:

```
BYTE SmartSense2X_EMC_bIsSensorActive(BYTE bSensorNum);
```

Assembly:

```
mov   A,  bSensorNum
lcall SmartSense2X_EMC_bIsSensorActive
```

Parameters:

bSensorNum A => Sensor Number

Return Value:

Returns value of 1 if active, 0 if not active

A => 1 - Selected sensor is active, 0 - Selected sensor is not active

Side Effects:

**

SmartSense2X_EMC_bIsAnySensorActive

Description:

Checks the difference count array for all sensors compared to their finger threshold. Calls SmartSense2X_EMC_bIsSensorActive() for each sensor so that the SmartSense2X_EMC_baSnsOnMask[] array is up to date after calling this function.

C Prototype:

```
BYTE  SmartSense2X_EMC_bIsAnySensorActive();
```

Assembly:

```
lcall  SmartSense2X_EMC_bIsAnySensorActive
```

Parameters:

None

Return Value:

Returns value of 1 if active, 0 if not active

A => 1 - One or more sensors are active, 0 - No sensors are active

Side Effects:

**

SmartSense2X_EMC_InitializeSensorBaseline(BYTE bSensorNum)

Description:

Loads the SmartSense2X_EMC_waSnsBaseline[bSensor] array element with an initial value by scanning the selected sensor. The raw count value is copied into the baseline array element for the selected sensor. This function can be used to reset the baseline of an individual sensor.

C Prototype:

```
void SmartSense2X_EMC_InitializeSensorBaseline(BYTE bSensorNum);
```

Assembly:

```
mov A, bSensor
lcall SmartSense2X_EMC_InitializeSensorBaseline
```

Parameters:

A => Sensor Number

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_InitializeBaselines

Description:

Loads the SmartSense2X_EMC_waSnsBaseline[] array with initial values by scanning each sensor. The raw count values are copied to the baseline array for each sensor.

C Prototype:

```
void  SmartSense2X_EMC_InitializeBaselines();
```

Assembly:

```
lcall  SmartSense2X_EMC_InitializeBaselines
```

Parameters:

None.

Return Value:

None

Side Effects:

**

SmartSense2X_EMC_wGetCentroidPos(BYTE bSnsGroup)

Description:

Checks a linear slider array for a centroid. If there is a centroid, the offset and length are stored in temporary variables and the centroid position is calculated to the resolution specified in the SmartSense2X_EMC Wizard. This function is available only if the slider is defined by the SmartSense2X_EMC Wizard.

C Prototype:

```
WORD  SmartSense2X_EMC_wGetCentroidPos(BYTE bSnsGroup);
```

Assembly:

```
mov    A,  bSnsGroup
lcall    SmartSense2X_EMC_wGetCentroidPos
```

Parameters:

bSnsGroup A => Group Number

This parameter is the Group Number of the slider. Group 0 is always the independent buttons. Sliders are contained in group 1 and higher.

Return Value:

Position value of the slider; LSB in A and MSB in X.

Side Effects:

This routine modifies the difference counts by subtracting the noise threshold value and should only be called once after each scan to avoid getting negative difference values. If your application monitors difference count signals, call this routine after difference count data transmission.

If any slider sensor is active, the function returns values from zero to the Resolution value set in the Wizard. If no sensors are active, the function returns -1 (FFFFh). If an error occurs during execution of the centroid/ uplexing algorithm, the function returns -1 (FFFFh). You can use the SmartSense2X_EMC_blsSensorActive() routine to determine which slider segments are touched.

SmartSense2X_EMC_wGetRadialPos(BYTE bSnsGroup)

Description:

Checks a radial slider array for a centroid. If there is a centroid, the centroid position is calculated to the resolution specified in the SmartSense2X_EMC Wizard.

C Prototype:

```
WORD  SmartSense2X_EMC_wGetRadialPos(BYTE bSnsGroup);
```

Assembly:

```
mov   A,  bSnsGroup
lcall  SmartSense2X_EMC_wGetRadialPos
```

Parameters:

bSnsGroup A => Group Number

This parameter is the Group Number of the radial slider. You can get its number through the SmartSense2X_EMC Wizard on the left side of radial slider representation (for example, "s2" means the radial slider Group Number is 2).

Return Value:

Position value of the radial slider; LSB in A and MSB in X.

Side Effects:

This routine modifies the difference counts by subtracting the noise threshold value and should only be called once after each scan to avoid getting negative difference values and baseline update. If your application monitors difference count signals, call this routine after difference count data transmission.

If any slider sensor is active, the function returns values from zero to the Resolution value set in the Wizard. If no sensors are active, the function returns -1 (FFFFh). If an error occurs during execution of the centroid algorithm, the function returns -1 (FFFFh). You can use the SmartSense2X_EMC_blsSensorActive() routine to determine which slider segments are touched.

SmartSense2X_EMC_wGetRadialInc(BYTE bSnsGroup)

Description:

Returns actual finger shift, the difference between current and previous finger positions. This function works in conjunction with SmartSense2X_EMC_wGetRadialPos().

C Prototype:

```
WORD  SmartSense2X_EMC_wGetRadialInc(BYTE bSnsGroup);
```

Assembly:

```
mov   A,  bSnsGroup
```

```
lcall   SmartSense2X_EMC_wGetRadialInc
```

Parameters:

bSnsGroup A => Group Number

This parameter is the Group Number of the radial slider. You can get its number through the SmartSense2X_EMC Wizard on the left side of the radial slider representation (for example, "s2" means the radial slider Group Number is 2).

Return Value:

Finger shift value, positive if clockwise and negative if anti-clockwise, LSB in A and MSB in X.

Finger shift value is the difference between current and previous finger positions.

Side Effects:

The routine must be called only after calling SmartSense2X_EMC_wGetRadialPos() because it uses internal data stored in global variables by the latter.

BOOL SmartSense2X_EMC_bIsScanComplete

Description:

Checks the scan complete flag in the bScanComplete variable and returns TRUE or FASLE. Also this API should reset this flag after call. This API is available if "BackgroundScanning" property is set to "Enable".

C Prototype

```
BOOL SmartSense2X_EMC_bIsScanComplete()
```

Assembly

```
lcall  SmartSense2X_EMC_bIsScanComplete
```

Parameters:

None

Return Value:

Returns value of 1 if active, 0 if not active

A => 1 - scan is complete, 0 - scan is incomplete

Side Effects:

**

SmartSense_EMC_UpdateSensorSignal(BYTE bSensorNumber)

Description:

Updates the SmartSense_EMC_baSnSSignal[] array with normalized difference count.

Parameters:

A => Sensor Number.

Return Value:

A => Normalized difference count.

Side Effects:

**

SmartSense2X_EMC_bFMEA_CheckGndShort(void)

Description

This API checks if any of the sensors is shorted to ground. The SmartSense2X_EMC_bFMEA_CheckVddShort() function updates the SmartSense2X_EMC_baSensorShortGnd[] array, which is valid only when the FMEA feature is enabled. SmartSense2X_EMC_baSensorShortGnd[0] contains the masked bits for sensors 0 through 7 (sensor 0 is bit 0, sensor 1 is bit 1). SmartSense2X_EMC_baSensorShortGnd[1] contains the masked bits for sensors 8 through 15 (if they are needed), and so on. This byte array contains as many elements as are necessary to contain all the placed sensors.

An additional array, SmartSense2X_EMC_baSnsShortChk[], has the same structure as the SmartSense2X_EMC_baSensorShortGnd[]. This SmartSense2X_EMC_baSnsShortChk[] is updated by all bFMEA_CheckSensorShort, bFMEA_CheckVddShort, bFMEA_CheckGndShort, bFMEA_Cmod_Check (or, bFMEA_Left_Cmod_Check, bFMEA_Right_Cmod_Check) APIs. The SmartSense2X_EMC_baShieldShort values for single and dual configurations are listed in the following tables.

Constant (Single Configuration)	Value	Comments
SmartSense2X_EMC_SHIELD_NO_SHORT	0x00	Shield Electrode is not shorted to GND
SmartSense2X_EMC_SHIELD_GND_SHORT	0x02	Shield Electrode is shorted to GND

Constant (Single Configuration)	Value	Comments
SmartSense2X_EMC_LEFT_SHIELD_NO_SHORT	0x00	Left Shield Electrode is not shorted to GND
SmartSense2X_EMC_LEFT_SHIELD_GND_SHORT	0x02	Left Shield Electrode is shorted to GND
SmartSense2X_EMC_RIGHT_SHIELD_NO_SHORT	0x00	Right Shield Electrode is not shorted to GND
SmartSense2X_EMC_RIGHT_SHIELD_GND_SHORT	0x20	Right Shield Electrode is shorted to GND

C Prototype

```
BYTE SmartSense2X_EMC_bFMEA_CheckGndShort(void);
```

Assembly

```
lcall  SmartSense2X_EMC_bFMEA_CheckGndShort
```

Parameters

None

Return Value

Returns value of 1 if any sensor is shorted to GND

Returns a value of 0 if none of the sensors is shorted to GND.

Side Effects

**

SmartSense2X_EMC_bFMEA_CheckVddShort(void)

Description

This API checks if any of the sensors is shorted to Vdd.

The SmartSense2X_EMC_bFMEA_CheckVddShort() function updates the SmartSense2X_EMC_baSensorShortVdd[] array, which is valid only when the FMEA feature is enabled. SmartSense2X_EMC_baSensorShortVdd[0] contains the masked bits for sensors 0 through 7 (sensor 0 is bit 0, sensor 1 is bit 1). SmartSense2X_EMC_baSensorShortVdd[1] contains the masked bits for sensors 8 through 15 (if they are needed), and so on. This byte array contains as many elements as are necessary to contain all the placed sensors.An additional array, SmartSense2X_EMC_baSnsShortChk[], has the same structure as the SmartSense2X_EMC_baSensorShortGnd[]. This SmartSense2X_EMC_baSnsShortChk[] is updated by all bFMEA_CheckSensorShort, bFMEA_CheckVddShort, bFMEA_CheckGndShort, and bFMEA_Cmod_Check (or bFMEA_Left_Cmod_Check, bFMEA_Right_Cmod_Check) APIs. The SmartSense2X_EMC_baShieldShort values for single and dual configurations are listed in the following tables.

Constant (Single Configuration)	Value	Comments
SmartSense2X_EMC_SHIELD_NO_SHORT	0x00	Shield Electrode is not shorted to VDD
SmartSense2X_EMC_SHIELD_VDD_SHORT	0x01	Shield Electrode is shorted to VDD

Constant (Single Configuration)	Value	Comments
SmartSense2X_EMC_LEFT_SHIELD_NO_SHORT	0x00	Left Shield Electrode is not shorted to VDD
SmartSense2X_EMC_LEFT_SHIELD_VDD_SHORT	0x01	Left Shield Electrode is shorted to VDD
SmartSense2X_EMC_RIGHT_SHIELD_NO_SHORT	0x00	Right Shield Electrode is not shorted to VDD
SmartSense2X_EMC_RIGHT_SHIELD_VDD_SHORT	0x10	Right Shield Electrode is shorted to VDD

C Prototype

```
BYTE SmartSense2X_EMC_bFMEA_CheckVddShort(void)
```

Assembly

```
lcall  SmartSense2X_EMC_bFMEA_CheckVddShort
```

Parameters

None

Return Value

Returns a value of 1 if any sensor is shorted to VDD and a value of 0 if none of the sensors is shorted to VDD.

SmartSense2X_EMC_bFMEA_CheckSensorShort(void)

Description

This API checks if any of the sensors is shorted to another sensor.An additional array, SmartSense2X_EMC_baSnsShortChk[], has the same structure as the SmartSense2X_EMC_baSensorShortGnd[]. This SmartSense2X_EMC_baSnsShortChk[] is updated by all bFMEA_CheckSensorShort, bFMEA_CheckVddShort, bFMEA_CheckGndShort, and bFMEA_Cmod_Check (or, bFMEA_Left_Cmod_Check, bFMEA_Right_Cmod_Check) APIs.The optional SmartSense2X_EMC_baShieldShort variable contains the Shield Electrode status. The SmartSense2X_EMC_baShieldShort values for single and dual configurations are listed in the following tables.

Constant (Single Configuration)	Value	Comments
SmartSense2X_EMC_SHIELD_NO_SHORT	0x00	Shield Electrode is not shorted to any sensor
SmartSense2X_EMC_SHIELD_SENS_SHORT	0x04	Shield Electrode is shorted to a sensor

Constant (Single Configuration)	Value	Comments
SmartSense2X_EMC_LEFT_SHIELD_NO_SHORT	0x00	Left Shield Electrode is not shorted to any sensor
SmartSense2X_EMC_LEFT_SHIELD_SENS_SHORT	0x04	Left Shield Electrode is shorted to a sensor
SmartSense2X_EMC_RIGHT_SHIELD_NO_SHORT	0x00	Right Shield Electrode is not shorted to any sensor
SmartSense2X_EMC_RIGHT_SHIELD_SENS_SHORT	0x40	Right Shield Electrode is shorted to a sensor
SmartSense2X_EMC_SHIELD_SHIELD_SHORT	0x80	Right Shield Electrode is shorted to Left Shield Electrode

C Prototype

```
BYTE SmartSense2X_EMC_bFMEA_CheckSensorShort(void)
```

Assembly

```
lcall  SmartSense2X_EMC_bFMEA_CheckSensorShort
```

Parameters

None

Return Value

Returns a value of 1 if any sensor is shorted to another sensor and a value of 0 if no sensor is shorted to another sensor.

SmartSense2X_EMC_bFMEA_Cmod_Check(void) - Single Channel

SmartSense2X_EMC_bFMEA_Left_Cmod_Check(void) - Dual Channel

SmartSense2X_EMC_bFMEA_Right_Cmod_Check(void) - Dual Channel

Description

This API is available only if the IDAC method is selected and if the FMEA feature is enabled. This checks Cmod for short test and also whether it is within the valid range or not. The valid range is defined by the recommended minimum and maximum Cmod values, with an error of 20%. The minimum Cmod value is 3.7 nF and the maximum Cmod value is 56.4 nF.The SmartSense2X_EMC_bFMEA_Cmod_Check() function updates SmartSense2X_EMC_wCmod_Val (SmartSense2X_EMC_wCmod_L_Val and SmartSense2X_EMC_wCmod_R_Val for Dual Channel Configurations) WORD variable. This variable contains the value of Cmod in nF scaled to 100. This value is valid only if the SmartSense2X_EMC_bFMEA_Cmod_Check() API returns bRetVal.4. An additional array, SmartSense2X_EMC_baSnsShortChk[], has the same structure as the SmartSense2X_EMC_baSensorShortGnd[]. This SmartSense2X_EMC_baSnsShortChk[] is updated by all bFMEA_CheckSensorShort, bFMEA_CheckVddShort, bFMEA_CheckGndShort, and bFMEA_Cmod_Check (or, bFMEA_Left_Cmod_Check, bFMEA_Right_Cmod_Check) APIs.

C Prototype

Single Channel:

```
BYTE SmartSense2X_EMC_bFMEA_Cmod_Check(void);
```

Dual Channel:

```
BYTE SmartSense2X_EMC_bFMEA_Left_Cmod_Check(void);
BYTE SmartSense2X_EMC_bFMEA_Right_Cmod_Check(void);
```

Assembly

```
lcall  SmartSense2X_EMC_bFMEA_Cmod_Check
```

Parameters

None

Return Value

bRetVal.0: Cmod shorted to GND

bRetVal.1: Cmod shorted to VDD

bRetVal.4: Cmod within +20%

bRetVal.8: Cmod is low (under the valid range) or Cmod is disconnected.

bRetVal.16: Cmod is high (over the valid range)

Single channel:

Constants	Value	Description
SMARTSENSE2X_EMC_CMOD_SHORTED_TO_GND	0	Cmod shorted to GND
SMARTSENSE2X_EMC_CMOD_SHORTED_TO_VDD	1	Cmod shorted to Vdd
SMARTSENSE2X_EMC_CMOD_WITHIN_20	4	Cmod within ±20%
SMARTSENSE2X_EMC_CMOD_IS_LOW	8	Cmod is out of range
SMARTSENSE2X_EMC_CMOD_IS_HIGH	16	Cmod is out of range

Dual channel:

Constants	Value	Description
SMARTSENSE2X_EMC_LEFT_CMOD_SHORTED_TO_GND	0	Left Cmod shorted to GND
SMARTSENSE2X_EMC_LEFT_CMOD_SHORTED_TO_VDD	1	Left Cmod shorted to Vdd
SMARTSENSE2X_EMC_LEFT_CMOD_WITHIN_20	4	Left Cmod within ±20%
SMARTSENSE2X_EMC_LEFT_CMOD_IS_LOW	8	Left Cmod is out of range
SMARTSENSE2X_EMC_LEFT_CMOD_IS_HIGH	16	Left Cmod is out of range
SMARTSENSE2X_EMC_RIGHT_CMOD_SHORTED_TO_GND	0	Right Cmod shorted to GND
SMARTSENSE2X_EMC_RIGHT_CMOD_SHORTED_TO_VDD	1	Right Cmod shorted to Vdd
SMARTSENSE2X_EMC_RIGHT_CMOD_WITHIN_20	4	Right Cmod within ±20%
SMARTSENSE2X_EMC_RIGHT_CMOD_IS_LOW	8	Right Cmod is out of range
SMARTSENSE2X_EMC_RIGHT_CMOD_IS_HIGH	16	Right Cmod is out of range

DC and AC Electrical Characteristics

Table 8. DC and AC Electrical Characteristics

Parameters	Description	Conditions	Min	Typ	Max	Units
Cp (max)	Supported sensor parasitic capacitance max value	Over –40 to +80 °C	45	–	–	pF
Cp (min)	Supported sensor parasitic capacitance min value	Over –40 to +80 °C	–	–	5	pF

Sample Firmware Source Code

Example 1. This code starts the user module and continuously scans the sensors. The communication section can be used to communicate values to a PC charting tool.

```
//-------------------------------------------------------------------------
// Sample C code for the SmartSense2X_EMC module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
    M8C_EnableGInt;
    SmartSense2X_EMC_Start();
    SmartSense2X_EMC_InitializeBaselines() ; //scan all sensors first time, init
    //baseline
    // Loop Forever

    while (1)
    {
        SmartSense2X_EMC_ScanAllSensors(); //scan all sensors in array (buttons and
        //sliders)
        SmartSense2X_EMC_UpdateAllBaselines(); //Update all baseline levels;
        //detect if any sensor is pressed
        if(SmartSense2X_EMC_bIsAnySensorActive())
        {
        // Add user code here to proceed with sensor touching
        }
        // now we are ready to send all status variables to chart program
        // communication here
        // OUTPUT SmartSense2X_EMC_waSnsResult[x] <- Raw Counts
        // OUTPUT SmartSense2X_EMC_waSnsDiff[x] <- Difference
        // OUTPUT SmartSense2X_EMC_waSnsBaseline[x] <- Baseline
        // OUTPUT SmartSense2X_EMC_baSnsOnMask[x] <- Sensor On/Off
    }
}
```

Example 2. This code starts the user module and continuously scans the sensors; however, unlike Example 1, looping through the sensors is done in the user code. The communication section can be used to communicate values to a PC charting tool.

```
//-------------------------------------------------------------------------
// Sample C code for the SmartSense2X_EMC module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

void main(void)
{
    BYTE bIndex;
    M8C_EnableGInt;
    SmartSense2X_EMC_Start();
    SmartSense2X_EMC_InitializeBaselines() ; //Scan all sensors first time, init
    //baseline
    while (1) //Loop forever
    {
        for(bIndex=0; bIndex < SmartSense2X_EMC_TotalSensorCount; bIndex++)
        //Loop through all sensors
        {
            SmartSense2X_EMC_ScanSensor(bIndex);           // Scan sensors
            SmartSense2X_EMC_UpdateSensorBaseline(bIndex); // Run baseline filter
            if(SmartSense2X_EMC_bIsSensorActive(bIndex))
            {
            // Add user code here to process the sensor touching
            }
        }
        // detect if any sensor is pressed
        // now we are ready to send all status variables to chart program
        // communication here
        //
        // OUTPUT SmartSense2X_EMC_waSnsResult[x] <- Raw Counts
        // OUTPUT SmartSense2X_EMC_waSnsDiff[x] <- Difference
        // OUTPUT SmartSense2X_EMC_waSnsBaseline[x] <- Baseline
        // OUTPUT SmartSense2X_EMC_baSnsOnMask[x] <- Sensor On/Off
    }
}
```

Example 3. This code starts the user module and continuously scans the sensors when the background scanning feature is enabled. The communication section can be used to communicate values to a PC charting tool.

```
//-------------------------------------------------------------------------
// Sample C code for the SmartSense2X_EMC module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
```

```
#include "PSoCAPI.h"    // PSoC API definitions for all user modules
void main(void)
{
    M8C_EnableGInt;
    SmartSense2X_EMC_Start();
    SmartSense2X_EMC_InitializeBaselines() ; //scan all sensors first time, init
    //baseline
    //
    // Loop Forever
    //
    while (1)
    {
        SmartSense2X_EMC_ScanAllSensors(); //scan all sensors in array (buttons and
        //sliders)
        while(!SmartSense2X_EMC_bIsScanComplete()); // wait until scanned

            SmartSense2X_EMC_UpdateAllBaselines(); //Update all baseline levels;
            // now we are ready to send all status variables to chart program
            // communication here
            //
            // OUTPUT SmartSense2X_EMC_waSnsResult[x] <- Raw Counts
            // OUTPUT SmartSense2X_EMC_waSnsDiff[x] <- Difference
            // OUTPUT SmartSense2X_EMC_waSnsBaseline[x] <- Baseline
            // OUTPUT SmartSense2X_EMC_baSnsOnMask[x] <- Sensor On/Off

        //detect if any sensor is pressed
        if(SmartSense2X_EMC_bIsAnySensorActive())
        {
            // Add user code here to proceed the sensor touching
        }
        // Do other tasks
    }
}
```

Version History

Version	Originator	Description
1.00	HPHA	Initial version.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2013 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.