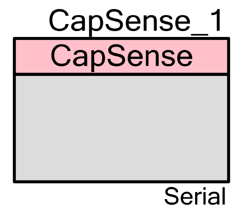
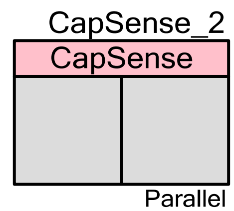
Capacitive Sensing (CapSense®)

1.20





Features

- Supports different combinations of independent and slide capacitive sensors
- High immunity to AC mains noise, EMC noise, and power supply voltage changes
- Parallel (synchronized or asynchronous) and Serial Scanning Configuration
- Shield electrode support for reliable operation in the presence of water film or droplets
- Guided slot and terminal assignments using the CapSense customizer

General Description

The Capacitive Sensing (CapSense) component provides a versatile and efficient means for measuring capacitance in applications such as touch sense buttons, sliders, and proximity detection.

When to use a CapSense Component

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

CapSense Component Quick Start

The following steps walk you through creating a CapSense project that senses two CapSense buttons and displays the status on an LCD. This section assumes that you are familiar with PSoC Creator and describes the basics of configuring the CapSense component in a project targeted to work on the CY8CKIT-001 DVK. If you are not familiar with PSoC Creator you may want to learn the basics before continuing.

This quick start section describes the process of placing and configuring the CapSense component, assigning the CapSense signals to physical PSoC pins and adding application level calls to CapSense component APIs to scan the sensors and act on the sensor status values. Parameter values used in these steps are suggested starting values assuming use of the DVK. If

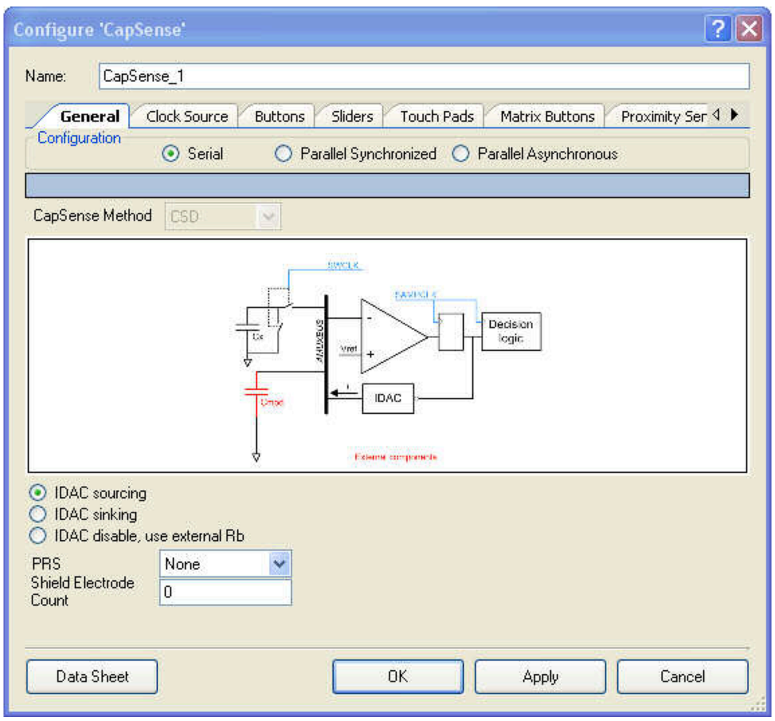
PRELIMINARY

you were creating an actual application, you would have to tune the parameters on your target hardware for optimal CapSense functionality.

1. Open a new PSoC Creator project. This project uses the default PSoC Creator clock resource configuration.

   Locate the CapSense component in the PSoC Creator Component Catalog. Select the component icon and drag it onto the TopDesign schematic view.

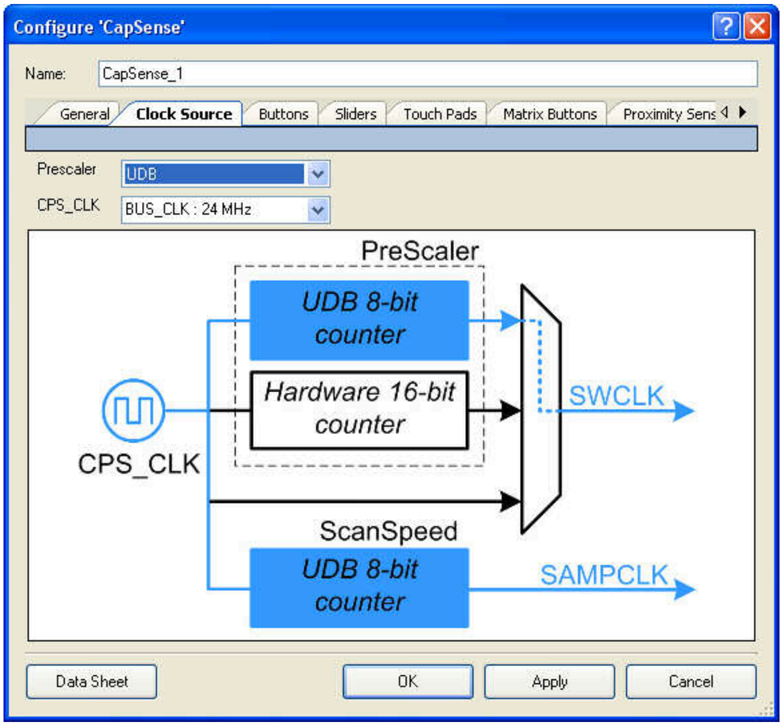
2. Double-click the CapSense component to open the Configure dialog. The dialog contains several tabs to configure the CapSense sensing method and various sensor types. Refer to the Parameters section for more details.



3. Under the **General** tab, choose the following settings:
   - Configuration: **Serial**
   - CapSense Method: **CSD**
   - Select : IDAC sourcing
   - PRS: **None**

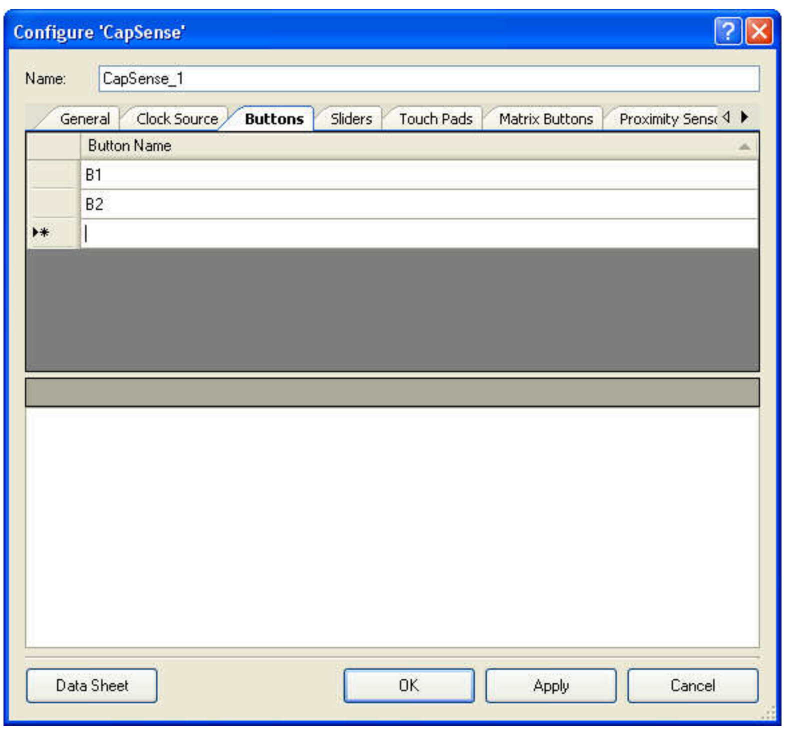
PRELIMINARY

4. Select the **Clock Source** tab and choose the following settings:



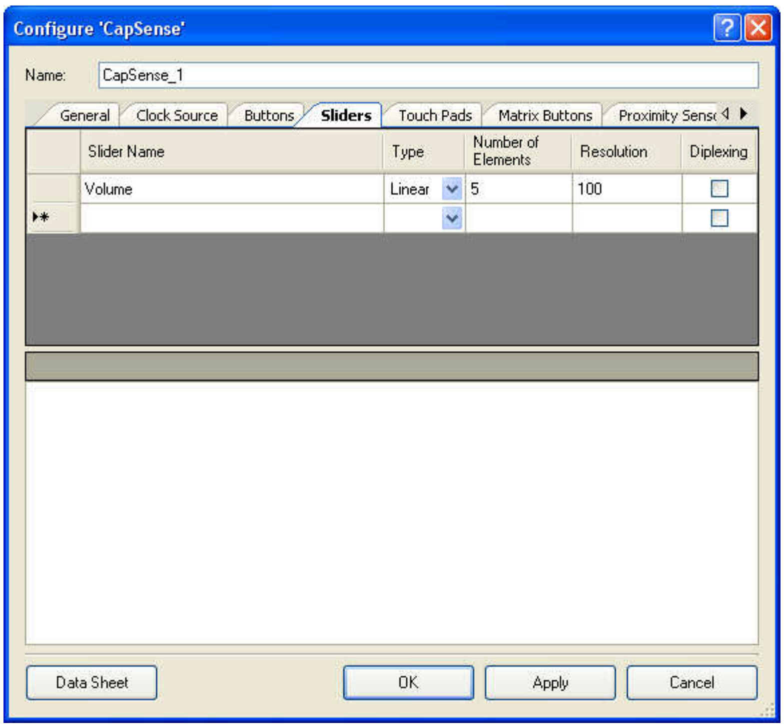
- Prescaler **UDB**
- CPS_CLK BUS_CLK: 24MHz

5. Select the **Buttons** tab.



- Click in the first row of the button definition table with the "*" symbol in first column. Type "B1" for the **Button Name**; enter "B2" as the name for the second button.
- The sensor specific CapSense parameters for the selected button are displayed in the lower half of the dialog. Make the following settings for both buttons:
  - Finger Threshold **75**
  - Noise Threshold **10**
  - Debounce **5**
  - Hysteresis **5**

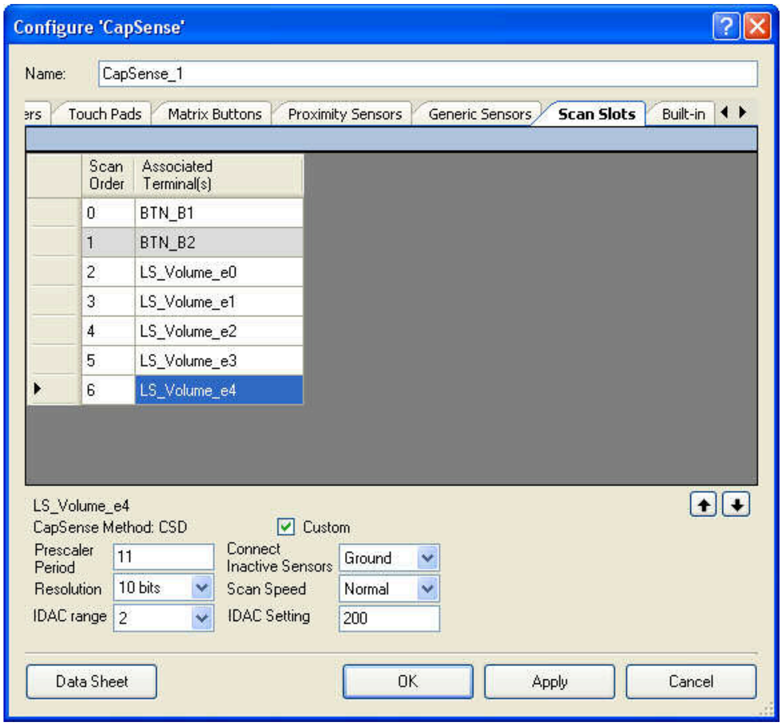
6. Select the **Sliders** tab.



- Click in the first row of the button definition table with the "*" symbol in first column. Type "Volume" for the **Slider Name**
- Specify **Type** of Slider as Linear, **Number of Elements** is 5, **Resolution** is 100.
- The sensor specific CapSense parameters for the selected button are displayed in the lower half of the dialog. Make the following settings for both buttons:
  - Finger Threshold 30
  - Noise Threshold 10

PRELIMINARY

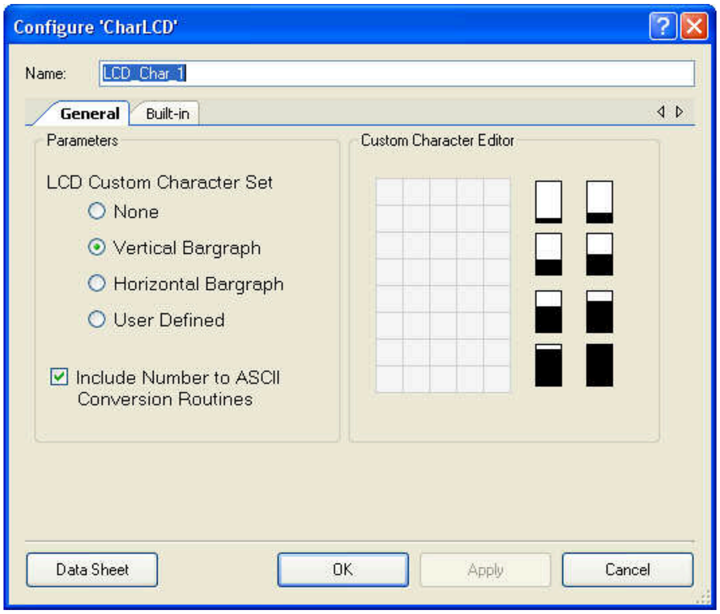
7. Select the **Scan Slots** tab (if the Scan Slots tab is not visible, press the right arrow control to scroll the Scan Slots tab into view). This tab allows you to configure the scanning order for the sensor slots set the sensing algorithm parameters individually for each scan slot.



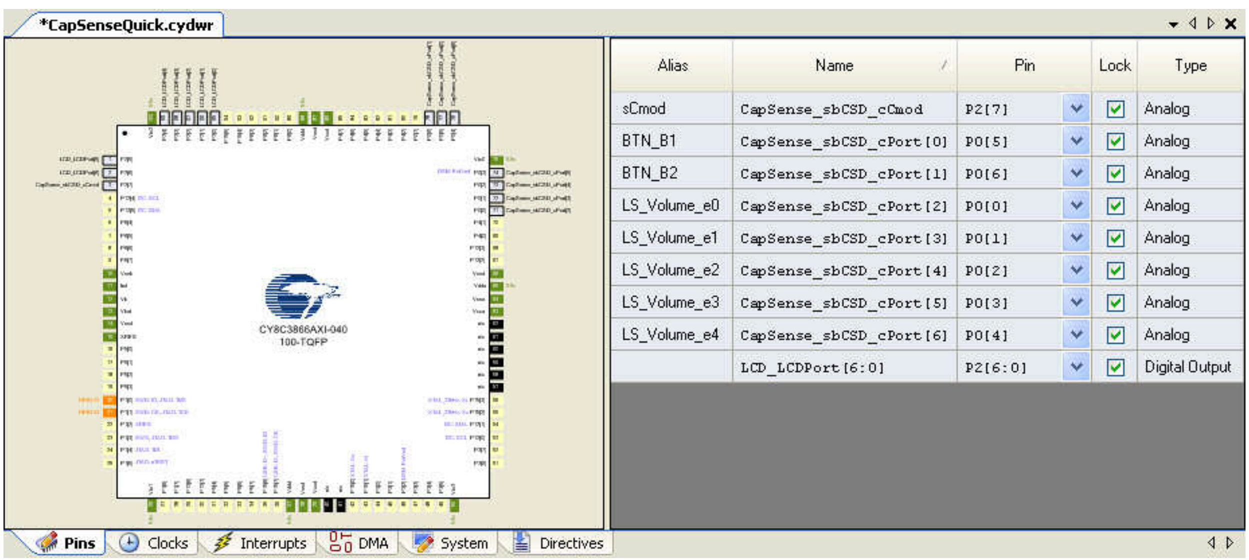
- Select button "BTN_B1" and "BTN_B2" by clicking in the button alias in the Associated Terminal column.
- Select the **Custom** check box. This allows you to set the scanning method parameters for this slot. Make the following settings for each scan slot:
  - Prescaler Period 11
  - Resolution 10 bits
  - IDAC Range 2
  - Scan Speed Normal
  - IDAC Setting 127
- Select all LS_Volume elements. Set the above settings except:
  - IDAC Setting 200

8. Click **OK** to close the dialog.
9. Locate the Character LCD component in the Component Catalog (in the "Display" folder). Select the component icon and drag it onto the TopDesign schematic view.

10. Double click the Character LCD component to open the configure dialog. Select Vertical Bargraph and include ASCII to Number Conversion Routines so that it works with the sample code provided.



11. Click **OK** to exit the dialog
12. Click the Save button to save the project.
13. The sensor signals (as well as the LCD signals) must be assigned to physical pins using the Design Wide Resources Pin Editor.



Each of the external signals associated with the project will be displayed in the pin assignment table on the right side of the dialog. Assign the CapSense signal pins as follows to use this project on the CY8CKIT-001 DVK board:

- sCmod P2[7]

PRELIMINARY

- BTN_B1 P0[5]
- BTN_B2 P0[6]
- LS_Volume_e0 – e4 P0[0] – P0[4]

Note Cmod 2.2nF implemented on board starts from Rev 5.

Note Cmod is connected between pin and GND.

General guidelines for assigning CapSense signals to physical pins are provided in the Pin Assignments Section.

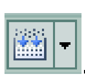
14. Assign the LCD control signals (LCD_LCDPort[6:0]) to P2[6:0] to use this project on the CY8CKIT-001 DVK board.

15. Copy the following code into the project *main.c* file. The generated API prototypes and available #defines used in this code are located in the *CapSense.h* file.

    The code fragment below demonstrates the initialization of the CapSense component and a loop to scan the two buttons and slider. The LCD displays the buttons status, position on slider and move bargraph accordingly slider position.

```
#include <device.h>

extern const uint8 LCD_Char_1_customFonts[];

void main()
{
        uint8 CurPos, OldPos = 0;
        CYGlobalIntEnable;

        /* Initialize LCD */
        LCD_Char_1_Start();
        LCD_Char_1_LoadCustomFonts(LCD_Char_1_customFonts);
        LCD_Char_1_Position(0, 0);
        LCD_Char_1_PrintString("TEST");

        /* Initialize the CapSense component */
        CapSense_1_Start();
        CapSense_1_CSHL_InitializeAllBaselines();

        /* Sensor Scanning Loop */
        while(1)
        {
              /* Scan all sensors and update baseline */
              CapSense_1_ScanAllSlots();
              CapSense_1_CSHL_UpdateAllBaselines();

              LCD_Char_1_Position(0, 6);

              /* Left button pressed */
              if (CapSense_1_CSHL_CheckIsSlotActive(CapSense_1_SCANSLOT_BTN_B1))
              {
                    LCD_Char_1_PrintString("ON ");
              }
```

```
        else
        {
            LCD_Char_1_PrintString("OFF");
        }

        LCD_Char_1_Position(0, 10);

        /* Right button pressed */
        if (CapSense_1_CSHL_CheckIsSlotActive(CapSense_1_SCANSLOT_BTN_B2))
        {
            LCD_Char_1_PrintString("ON ");
        }
        else
        {
            LCD_Char_1_PrintString("OFF");
        }

        /* Find Slider Position */
        CurPos = CapSense_1_CSHL_GetCentroidPos(CapSense_1_CSHL_LS_VOLUME);

        /* Reset position */
        if(CurPos == 0xFF)
        {
            CurPos = 0;
        }

        /* Move bargraph */
        if (CurPos != OldPos)
        {
            LCD_Char_1_DrawVerticalBG(1, OldPos/6, 1, 0);
            OldPos = CurPos;
            if (CurPos != 0)
            {
                LCD_Char_1_DrawVerticalBG(1, CurPos/6, 1, 7);
            }

            LCD_Char_1_Position(0, 14);
            LCD_Char_1_PrintInt8(CurPos);
        }
    }
}
```

16. Click Build Project . This will generate and compile the source code files that implement the CapSense solution.
17. Program the project on to your DVK board.

Input/Output Connections

No schematic connections are required for the CapSense component. The component encompasses the analog and digital functional blocks and required interconnections that implement the capacitive sensing algorithms.

PRELIMINARY

Parameters and Setup

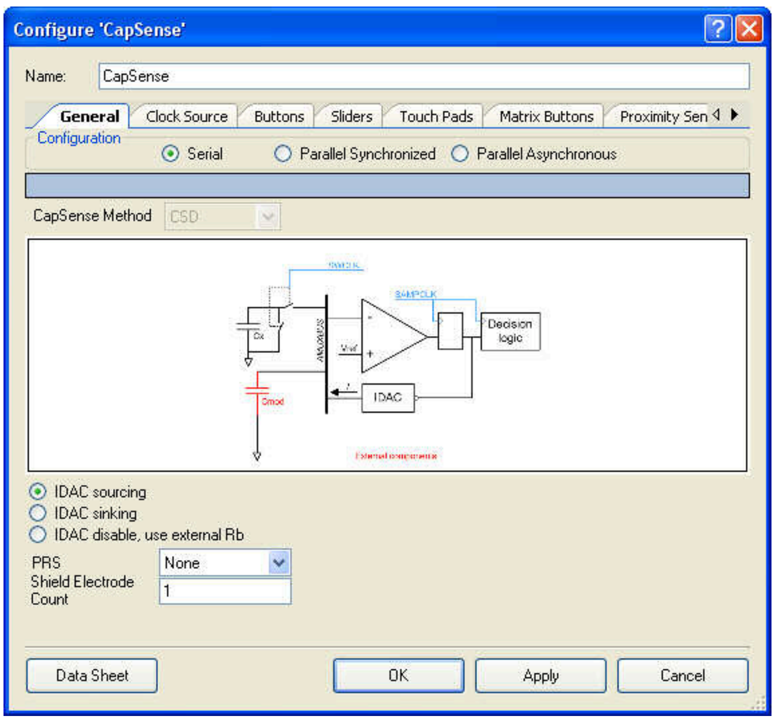
Drag a CapSense component onto your design and double-click it to open the Configure dialog. This dialog has several tabs to guide you through the process of setting up the CapSense component.

General Tab

Configuration

There are three Configuration options:

- **Serial –** The component is capable of performing 1 capacitive scan at a time. Multiple sensors are scanned one at a time in succession.

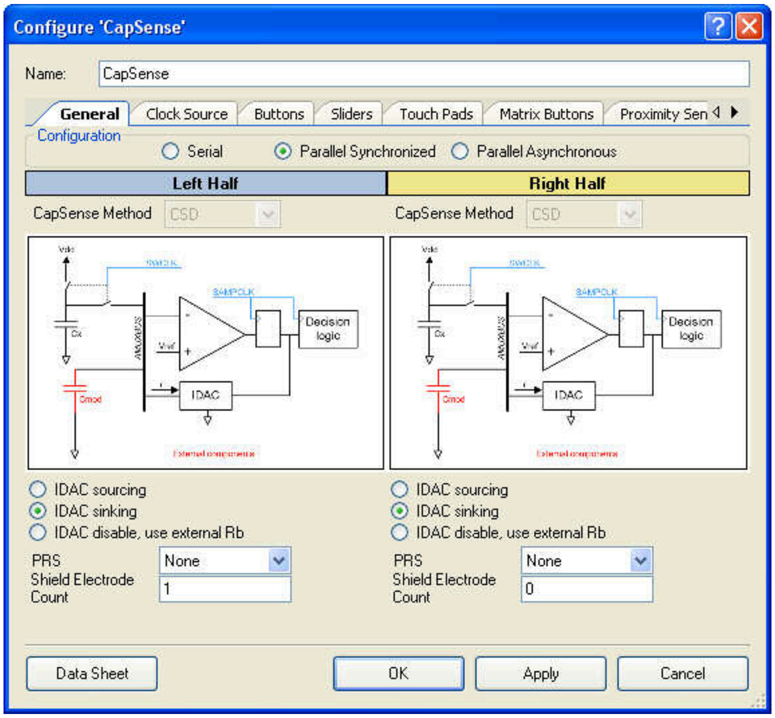


  - The AMUX buses are tied together.

    Note If all capacitive sensors are allocated on one side of the chip Left (0 – (#EVEN_PORT_GPIO – 1) or Right (0– (#ODD_PORT_GPIO – 1) the AMUX buses don't tie together, the one half of AMUX bus is used.

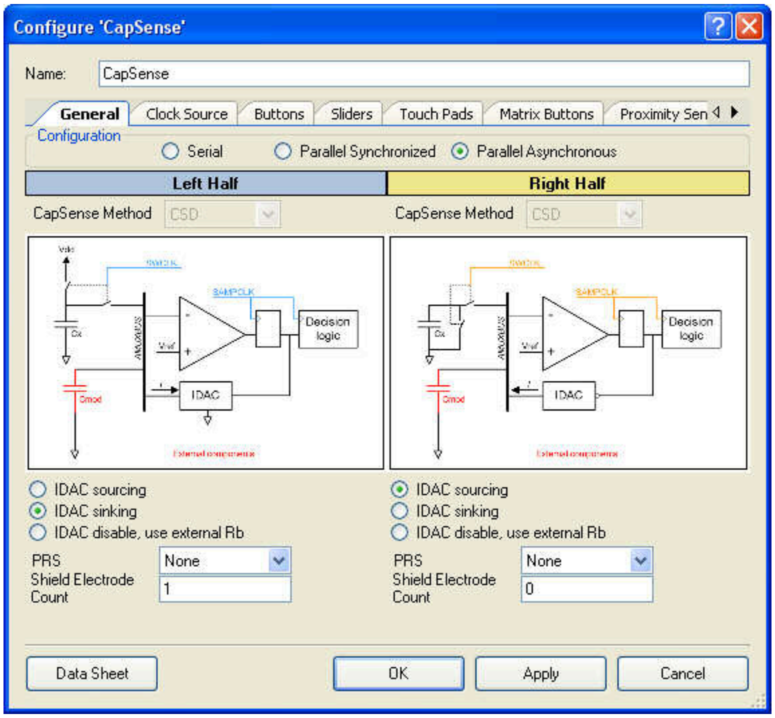
  - The component is capable of scanning 1 to (#GPIO – 1) capacitive sensors.
  - The tied AMUX bus can use only one sensing method.
  - One Cmod external capacitor is required.

- **Parallel Synchronized –** The component is capable of performing 2 simultaneous capacitive scans. Both the left and right AMUX buses are used. Right and left sensors are scanned two at time (one right sensor and one left sensor at a time) in succession. If one channel has more sensors than the other, the channel with the greater number of sensors will finish scanning the remaining sensors in its array one at a time until done.



  - The left AMUX bus is capable of scanning 1 to (#EVEN_PORT_GPIO – 1) capacitive sensors.
  - The right AMUX bus is capable of scanning 1 to (#ODD_PORT_GPIO – 1) capacitive sensors.
  - Each AMUX bus can use a different sensing method.
  - Two Cmod external capacitors are required.
  - Parallel scans run at the same scan rate.

- **Parallel Asynchronous –** The component is capable of performing 2 simultaneous capacitive scans. The left and right AMUX bus are used. The right and left channels are scanned independently of each other.



  - The left AMUX bus is capable of scanning 1 to (#EVEN_PORT_GPIO – 1) capacitive sensors.
  - The right AMUX bus is capable of scanning 1 to (#ODD_PORT_GPIO – 1) capacitive sensors.
  - Each AMUX bus can use a different sensing method.
  - Two Cmod external capacitors are required.
  - Parallel scans run at the different scan rates.

CapSense Method

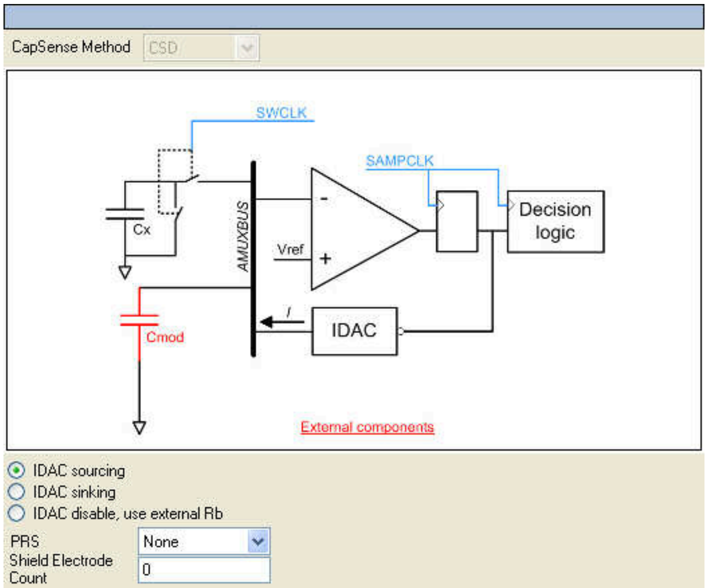
Choose a CapSense method:

- **CSD –** The CSD (Capacitive Sensing using a Delta-Sigma Modulator) provides capacitance sensing using the switched capacitor technique with a delta-sigma modulator to convert the sensing switched capacitor current to a digital code. It allows implementation of buttons, sliders, touch pads and matrix buttons using arrays of conductive sensors. High level software routines allow for enhancement of slider resolution using diplexing, and compensation for physical and environmental sensor variation.

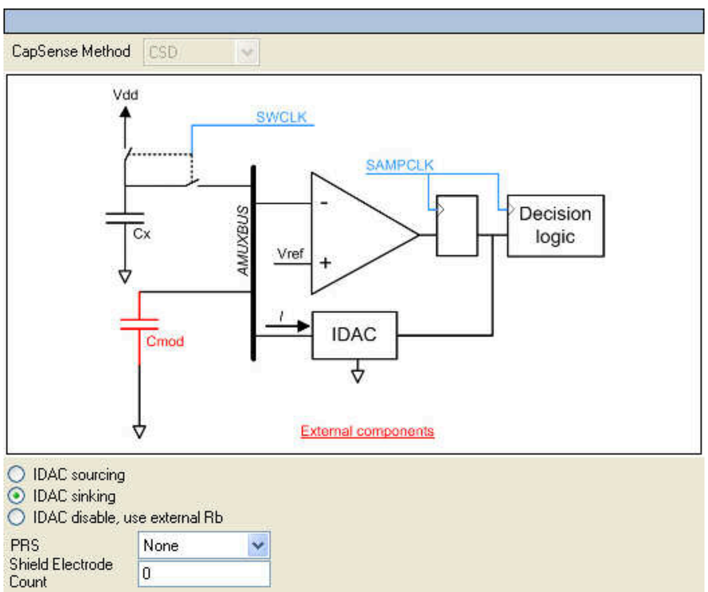
CSD Method Variants

There are three CSD method variants:

- **IDAC sourcing –** The switch stage is configured to alternate between GND and the AMUX bus. The IDAC sources the current.

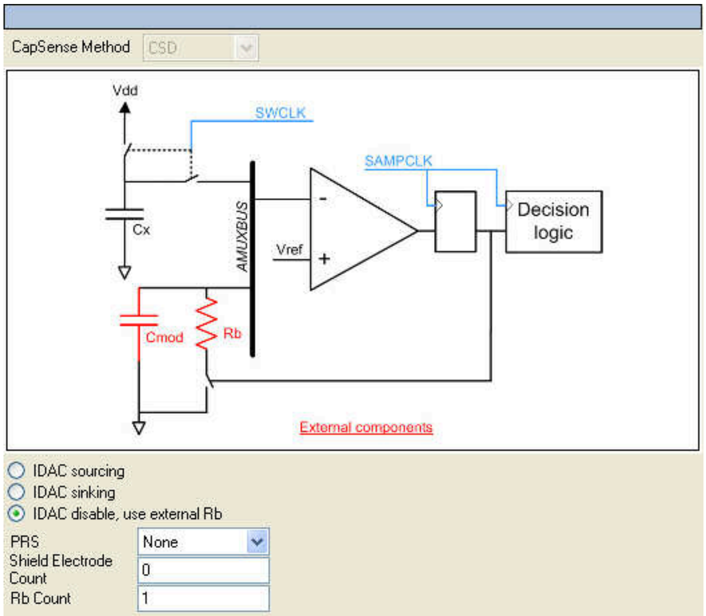


- **IDAC sinking –** The switch stage is configured to alternate between Vdd and the AMUX bus. The IDAC sinks the current.



PRELIMINARY

- **IDAC disable, use external Rb –** This is the same as the IDAC sinking configuration except the IDAC is replaced with a bleed resistor to ground, Rb. The bleed resistor is connected between the Cmod and a GPIO. The GPIO is configured to Open-Drain Drives Low drive mode. This drive mode allows the Cmod to be discharged through Rb.



PRS

The Pseudo Random Sequence (PRS) generator drives the switching clock, SW_CLK. The options include:

- PRS 16 – 16-bit pseudo-random sequence generator is used.
- PRS 8 – 8-bit pseudo-random sequence generator is used.
- None – no pseudo-random sequence generator is used.

Note When using PRS 16, the SW_CLK should be two times faster.

Shield Electrode Count

This determines the number of shield electrodes. This option creates a number of special terminals for shield signals. Shield signals are shown in the Pin Editor but are not shown in the Scan Slots tab.

Rb Count

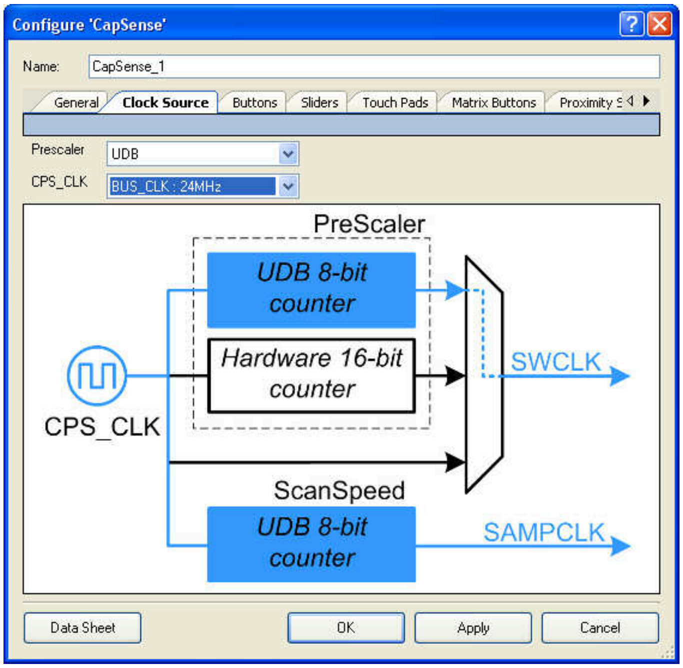
This determines the number of bleed resistors. The maximum number of bleed resistors is 3. Bleed resistor terminals are shown in the Pin Editor but are not shown in the Scan Slots tab.

Note The maximum quantity of sensors, shield-electrodes, and bleed resistors is 62.

Clock Sources Tab

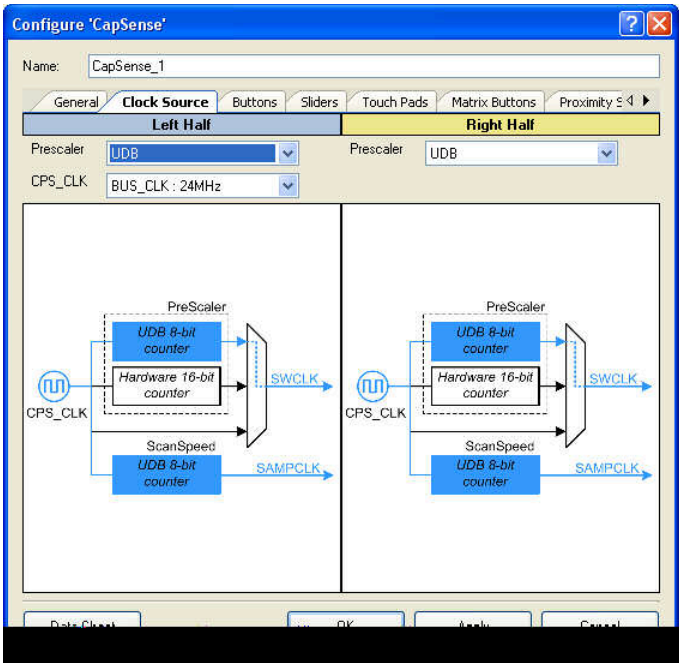
Serial Mode

In this mode one clock source for all CapSense system is needed. Choose a CPS_CLK from the available sources.



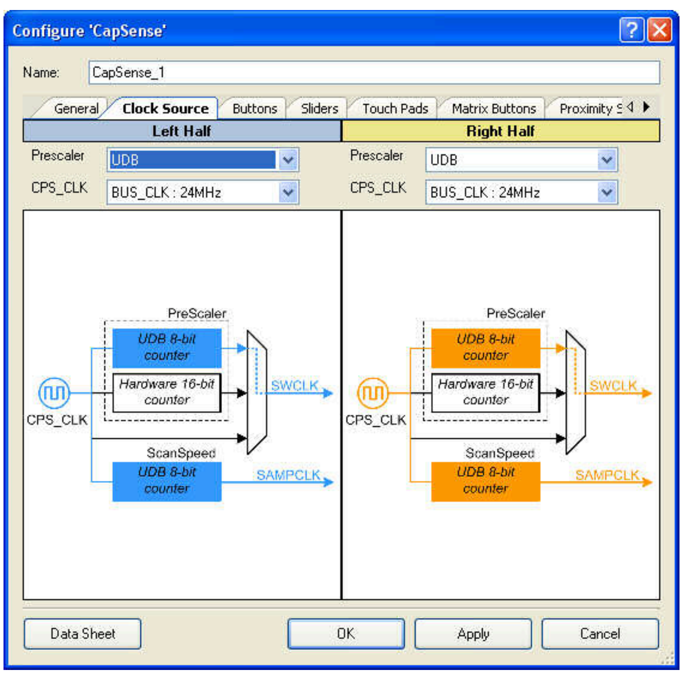
Parallel Synchronized

In this mode the same clock is provided to both prescalers. Choose one CPS_CLK for both channels from the available sources.



Parallel Asynchronous

In this mode the different clocks are provided to the left and right prescalers. Choose a CPS_CLK for each channel from the available sources.



Prescaler

This is used to select resources that are used for prescaler:

- UDB – a UDB-based 8-bit counter is used as the prescaler.
- HW – the fixed function 16-bit timer is used as the prescaler.
- None – the prescaler functionality is not used.

Note When using HW prescaler to run at a frequency equal to the bus clock, the bus clock must be used.

CPS_CLK

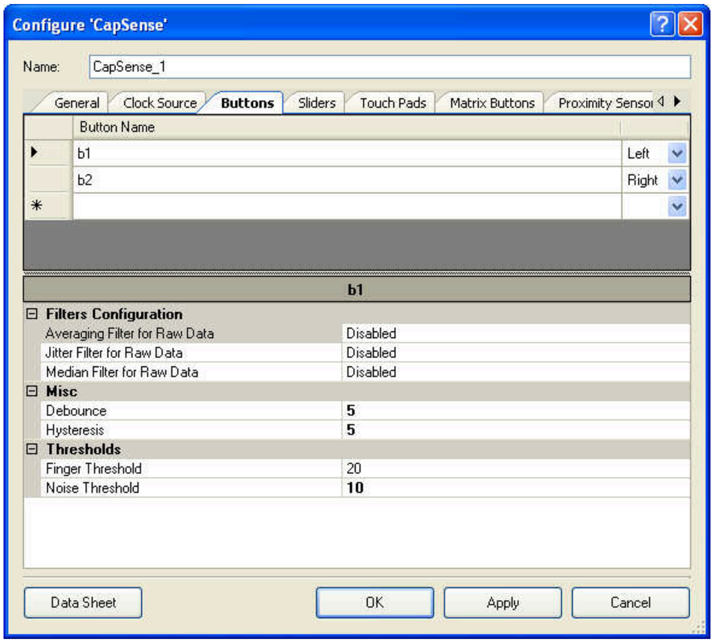
Select the clock source for the prescaler. Available choices are:

- Several clocks are specially created for the CapSense component. If you choose one of these clocks, one clock divider from digital clock tree will be utilized. The special CapSense clock sources are:
  - 12 MHz
  - 24 MHz
  - 48 MHz
  - 92 MHz
- CapSense clock could be select as direct clock that goes directly from one of the system clocks.Now the only available choose is BUS_CLK. Choosing direct clock will not consume any additional system resources.
- The custom CapSense clock could be created, putting the desire clock frequency in MHz into the edit box. If you create custom clock, one clock divider from digital clock tree will be utilized.

Note The CapSense clock cannot be greater than the fastest clock in the system (MASTER_CLK).

Buttons Tab

The buttons tab varies slightly depending on whether the configuration is serial or parallel. The serial configuration does not have the choice of assigning buttons to the left or right side.

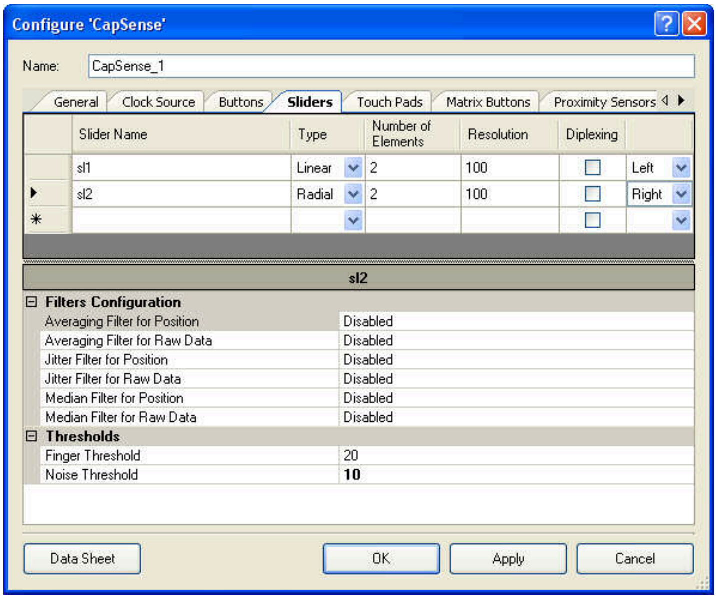


Definitions of the parameters are in the Functional Description section.

- Finger Threshold and Noise Threshold (simple ON/OFF result)
- Hysteresis for Finger Threshold
- Debounce support
- Raw Data Filtering:
    - Jitter Filter
    - Median Filter
    - Averaging Filter
    - IIR Filter

Sliders Tab

The sliders tab varies slightly depending on whether the configuration is serial or parallel. The serial configuration does not have the choice of assigning sliders to the left or right side.



Linear Slider:

Definitions of the parameters are in the Functional Description section.

- Interpolated position (resolution)
- Diplexing
- Raw Data Filtering:
  - Jitter Filter
  - Median Filter
  - Averaging Filter
  - IIR Filter
- Position Filtering:
  - Jitter Filter
  - Median Filter
  - Averaging Filter
  - IIR Filter

PRELIMINARY

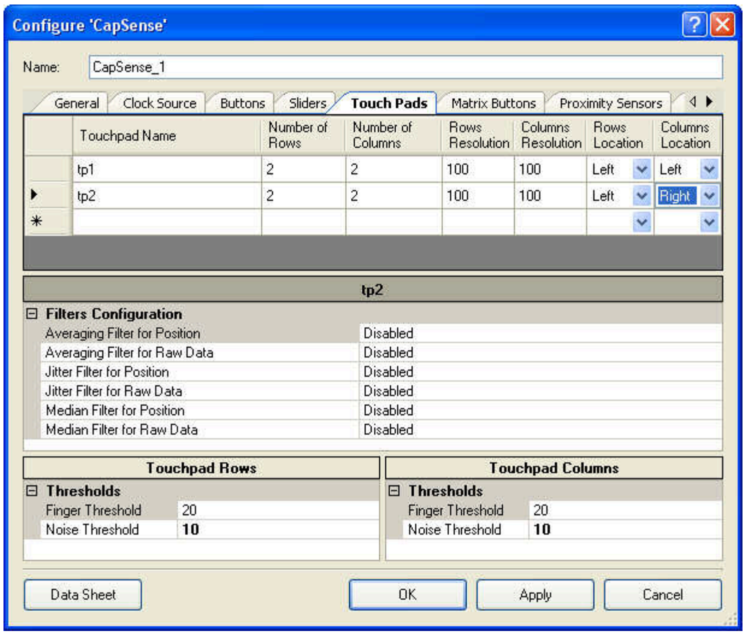
Radial Slider:

Definitions of the parameters are in the Functional Description section.

- Interpolated position (resolution)
- Diplexing
- Raw Data Filtering:
  - Jitter Filter
  - Median Filter
  - Averaging Filter
  - IIR Filter
- Position Filtering:
  - Jitter Filter
  - Median Filter
  - Averaging Filter
  - IIR Filter

Touch Pads Tab

The touch pads tab varies slightly depending on whether the configuration is serial or parallel. In the **serial** configuration there is no choice of left or right side. In the **parallel synchronized** configuration the column and row can be assigned to the left side or right side separately. For example, the left side can scan rows while the right side scans columns. In the **parallel asynchronous** configuration the entire touch pad, both rows and columns, should be assigned to the same side. For example, the right side scans the touch pad while the left side scans all other CapSense controls.



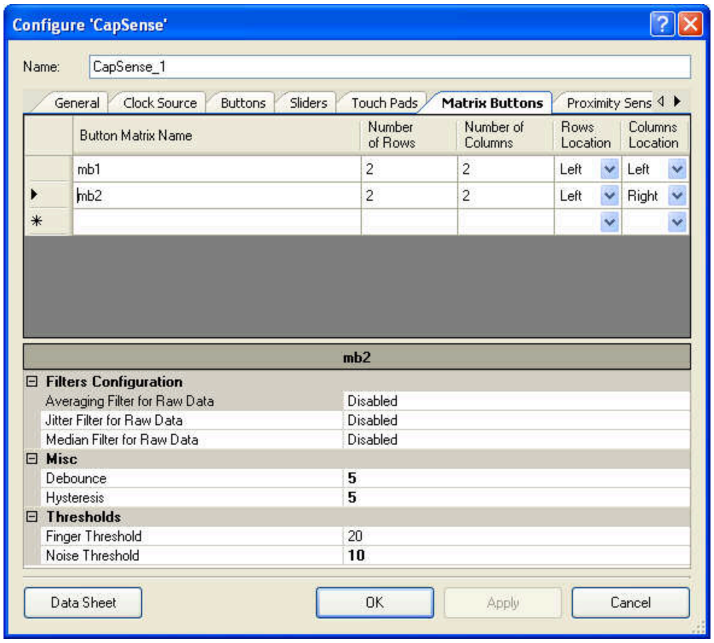
Definitions of the parameters are in the Functional Description section.

- Interpolated position (resolution) for X and Y
- Raw Data Filtering:
  - Jitter Filter
  - Median Filter
  - Averaging Filter
  - IIR Filter
- Position Filtering (X and Y):
  - Jitter Filter
  - Median Filter
  - Averaging Filter
  - IIR Filter

PRELIMINARY

Matrix Buttons Tab

The matrix buttons tab varies slightly depending on whether the configuration is serial or parallel. In the serial configuration there is no choice of left or right side. In the parallel synchronized configuration the column and row can be assigned to the left side or right side separately. For example, the left side can scan rows while the right side scans columns. In the parallel asynchronous configuration all matrixed buttons, both rows and columns, should be assigned to the same side. For example, the right side scans the button matrix while the left side scans all other CapSense controls.

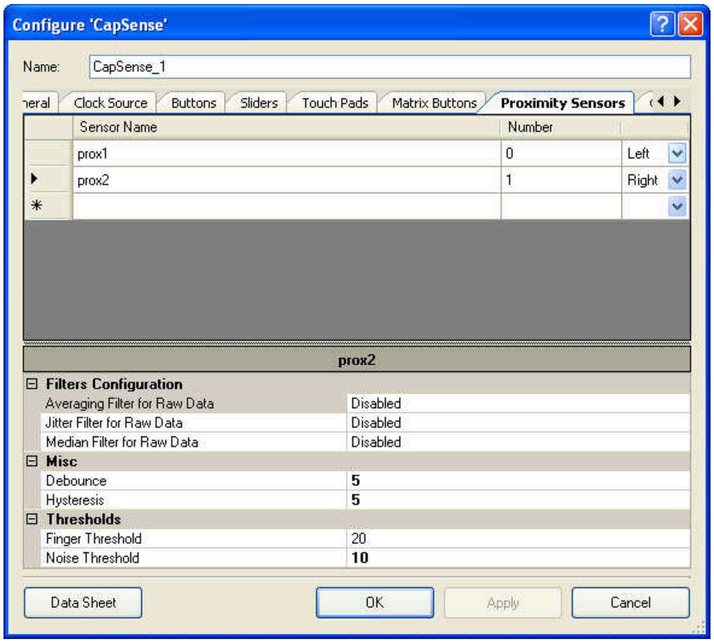


Definitions of the parameters are in the Functional Description section.

- Finger Threshold and Noise Threshold (simple ON/OFF result for each button)
- Hysteresis for Finger Threshold
- Debounce Support
- Raw Data Filtering:
    - Jitter Filter
    - Median Filter
    - Averaging Filter
    - IIR Filter

Proximity Sensors Tab

The proximity sensors tab varies slightly depending on whether the configuration is serial or parallel. The serial configuration does not have the choice of assigning proximity sensors to the left or right side.



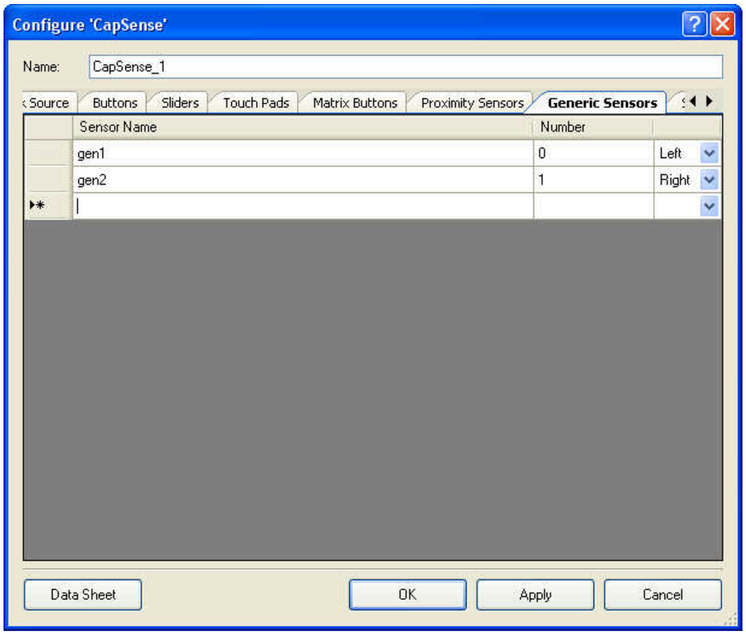
Definitions of the parameters are in the Functional Description section.

- Finger Threshold and Noise Threshold (simple ON/OFF result for the proximity sensor)
- Hystersis for Proximity Threshold
- Debounce Support
- Number – selects the number of proximity sensors:
  - 0 – The proximity sensor will scan one or more existing sensors to determine proximity. No terminals are allocated for this sensor.
  - 1 to N – Number of dedicated proximity sensors in the system.
- Raw Data Filtering:
  - Jitter Filter
  - Median Filter
  - Averaging Filter
  - IIR Filter

PRELIMINARY

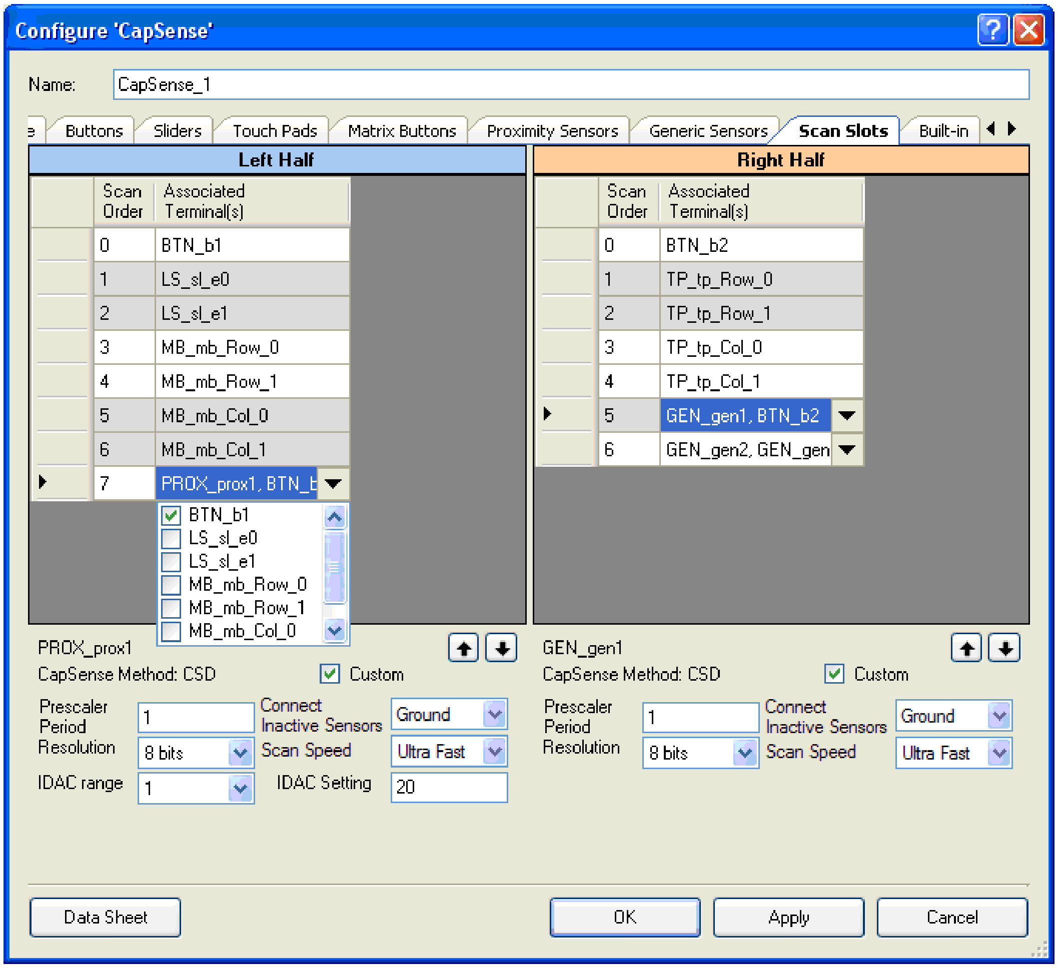
Generic Sensors Tab

The generic sensors tab varies slightly depending on whether the configuration is serial or parallel. The serial configuration does not have the choice of assigning generic sensors to the left or right side.



- Generic Sensors – No high-level support is provided. Raw sensor data is provided by the component and any high-level functionality required by the application will be developed by the user.
- Number – selects the number of generic sensors:
    - 0 – Use this setting to get raw data from another sensor or sensors. No terminals are allocated for this sensor.
    - 1-N – number of generic sensors. 1-N generic terminals are allocated.

Scan Slots Tab



For serial scanning configurations one table lists the scan slots for the Analog Mux Bus (two Analog Mux Buses tied together*). For parallel scanning configurations two tables list the scan slots for each Analog Mux Bus. Widgets are listed in alternating gray and white rows in the table. All terminals are associated with a widget (for example, a touch pad rows or columns) share the same color.

You cannot use the scan slot parameters tab to move a scan slot from the right half to the left half, but you can change the scan order of the slots. If you move one member of a widget (for example the first sensor in a linear slider) all other sensors belonging to the widget will be moved at the same time.

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

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

If all capacitive sensors are allocated on one side of the chip Left –(0 – (#EVEN_PORT_GPIO – 1) or Right(0 – (#ODD_PORT_GPIO – 1) the AMUX buses don't tie together, the one half of AMUX bus is used.

CSD Scan Slot Parameters

Scan slots within a particular widget, such as all the sensors in a linear slider, should have identical parameters except IDAC range and IDAC Setting. If you change any of the other parameters for one sensor in a widget you should change all the others to be the same.

Custom – If Custom is checked, each scan slot has different settings. Select each scan slot in turn to define its settings. If Custom is unchecked, all scan slots share the same settings.

IDAC Sourcing and IDAC Sinking Configurations:

- Scan Speed – Defines the scan speed as Ultra Fast, Fast, Normal, or Slow.
- Resolution – Defines the scanning resolution of the PWM. Choices are from 8 to16 bits.
- IDAC Range – Multiplies the IDAC current by the number selected (1 – 3).
- IDAC Settings – Selects the IDAC scanning value (0 – 255).
- PreScaler Period – The switch clock (SWCLK) is the CapSense clock (CPS_CLK) divided by this number to obtain the switch frequency for the break-before-make logic. (2 – 255).
- Connect Inactive Sensors – Defines the unscanning sensor connection as GND, High-Z and Shield electrode*.

Configurations without an IDAC:

- Scan Speed – Defines the scan speed as Ultra Fast, Fast, Normal, or Slow.
- Resolution – Defines the scanning resolution of the PWM. Choices are 8 and 16 bits.
- PreScaler Period – The switch clock (SWCLK) is the CapSense clock (CPS_CLK) divided by this number to obtain the switch frequency for the break-before-make logic. (2 – 255).
- Connect Inactive Sensors – Defines the unscanning sensor connection as GND, High-Z and Shield electrode[1]

[1] Shield option is available only if shield electrode count is greater than zero.

Clock Selection

Select the clock source for the prescaler. Available choices are:

- Several clocks are specially created for the CapSense component. If you choose one of these clocks, one clock divider from digital clock tree will be utilized. The special CapSense clock sources are:
  - 12 MHz
  - 24 MHz
  - 48 MHz
  - 92 MHz
- CapSense clock could be select as direct clock that goes directly from one of the system clocks.Now the only available choose is BUS_CLK. Choosing direct clock will not consume any additional system resources.
- The custom CapSense clock could be created, putting the desire clock frequency in MHz into the edit box. If you create custom clock, one clock divider from digital clock tree will be utilized.

Note The CapSense clock cannot be greater than the fastest clock in the system (MASTER_CLK).

Placement

Not applicable

Resources

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

PRELIMINARY

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

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

High Level APIs

The CSHL is prefix for all High Level APIs. These APIs get raw data from scan slots and convert it to on/off for buttons, position for sliders, or X and Y coordinates for touch pads.

Some High Level API functions are appended with “Left” and “Right” in Parallel mode. APIs appended with “Left” work only with the left side of CapSense system. Those appended with “Right” work only with the right side.

Function	Description
CapSense_CSHL_InitializeSlotBaseline	Loads the CapSense_CSHL_SlotBaseline[slot] array element with an initial value by scanning the selected slot. The raw count value is copied into the baseline array for each slot. The raw data filters are initialized if enabled.
CapSense_CSHL_InitializeAllBaselines	Uses the CapSense_CSHL_InitializeSlotBaseline function to loads the CapSense_CSHL_SlotBaseline[ ] array with an initial values by scanning all slots. The raw count values are copied into the baseline array for all slots. The raw data filters are initialized if enabled..
CapSense_CSHL_UpdateSlotBaseline	Updates the CapSense_CSHL_SlotBaseline[ ] array using the LP filter with k = 256. The signal calculates the difference of count by subtracting the previous baseline and noise threshold from the current raw count value. The baseline stops updating if signal is greater that zero. Raw data filters are applied to the values if enabled.
CapSense_CSHL_UpdateAllBaselines	Uses the CapSense_CSHL_UpdateSlotBaseline function to update the baselines for all slots. Raw data filters are applied to the values if enabled.
CapSense_CSHL_CheckIsSlotActive	Compares the selected slot of the CapSense_CSHL_Signal[ ] array to its finger threshold. Hysteresis is taken into account. The Hysteresis value is added or subtracted from the finger threshold based on whether the slot is currently active. If the slot is active, the threshold is lowered by the hysteresis amount. If it is inactive, the threshold is raised by the hysteresis amount. This function also updates the slot's bit in the CapSense_CSHL_SlotOnMask[ ] array.

Function	Description
CapSense_CSHL_CheckIsAnySlotActive	Compares all slots of the CapSense_CSHL_Signal[ ] array to their finger threshold. Calls Capsense_CSHL_CheckIsSlotActive() for each slot so the CapSense_CSHL_SlotOnMask[ ] array is up to date after calling this function.
CapSense_CSHL_GetCentroidPos	Checks the CapSense_CSHL_Signal[ ] array for a centroid. The centroid position is calculated to the resolution specified in the CapSense customizer. The position filters are applied to the result if enabled. This function is available only if a linear slider is defined by the CapSense customizer.
CapSense_CSHL_GetRadialCentroidPos	Checks the CapSense_CSHL_Signal[ ] array for a centroid. The centroid position is calculated to the resolution specified in the CapSense Customizer. The position filters are applied to the result if enabled. This function is available only if a radial slider is defined by the CapSense customizer.
CapSense_CSHL_GetDoubleCentroidPos	If a finger is present, this function calculates the X and Y position of the finger by calculating the centroids. The X and Y positions are calculated to the resolutions set in the CapSense customizer. Returns a '1' if a finger is on the touchpad. The position filters are applied to the result if enabled. This function is available only if a touch pad is defined by the CapSense customizer.

void CapSense_CSHL_InitializeSlotBaseline(uint8 slot)

Description: Loads the CapSense_CSHL_SlotBaseline[slot] array element with an initial value by scanning the selected slot. The raw count value is copied into the baseline array for each slot. The raw data filters are initialized if enabled.

Parameters: slot: uint8 – Scan slot number

Return Value: None

Side Effects: None

void CapSense_CSHL_InitializeAllBaselines

Description: Uses the CapSense_CSHL_InitializeSlotBaseline function to loads the CapSense_CSHL_SlotBaseline[ ] array with an initial values by scanning all slots. The raw count values are copied into the baseline array for all slots. The raw data filters are initialized if enabled..

Parameters: None

Return Value: None

Side Effects: None

void CapSense_CSHL_UpdateSlotBaseline(uint8 slot)

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

Parameters: slot: uint8 – Scan slot number

Return Value: None

Side Effects: None

void CapSense_CSHL_UpdateAllBaselines

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

Parameters: None

Return Value: None

Side Effects: None

uint8 CapSense_CSHL_CheckIsSlotActive(uint8 slot)

Description: Compares the selected slot of the CapSense_CSHL_Signal[ ] array to its finger threshold. Hysteresis is taken into account. The Hysteresis value is added or subtracted from the finger threshold based on whether the slot is currently active. If the slot is active, the threshold is lowered by the hysteresis amount. If it is inactive, the threshold is raised by the hysteresis amount. This function also updates the slot's bit in the CapSense_CSHL_SlotOnMask[ ] array.

Parameters: slot: uint8 – Scan slot number

Return Value: uint8: Scan slot state 1 if active, 0 if inactive

Side Effects: None

uint8 CapSense_CSHL_CheckIsAnySlotActive

Description: Compares all slots of the CapSense_CSHL_Signal[ ] array to their finger threshold. Calls Capsense_CSHL_CheckIsSlotActive() for each slot so the CapSense_CSHL_SlotOnMask[ ] array is up to date after calling this function.

Parameters: None

Return Value: uint8: 1 if any scan slot is active, 0 none of scan slots are active

Side Effects: None

uint16 CapSense_CSHL_GetCentroidPos(uint8 widget)

Description: Checks the CapSense_CSHL_Signal[ ] array for a centroid. The centroid position is calculated to the resolution specified in the CapSense customizer. The position filters are applied to the result if enabled. This function is available only if a linear slider is defined by the CapSense customizer.

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

```
#define CapSense_CSHL_LS_"widget_name"            5
```

Return Value: uint16: Position value of the slider

Side Effects: If any slider slot is active, the function returns values from zero to the resolution value set in the CapSense customizer. If no sensors are active, the function returns -1. If an error occurs during execution of the centroid/diplexing algorithm, the function returns -1.
You can use the CSHL_CheckIsSlotActive() routine to determine which slider segments are touched, if required.

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

uint16 CapSense_CSHL_GetRadialCentroidPos(uint8 widget)

Description: Checks the CapSense_CSHL_Signal[ ] array for a centroid. The centroid position is calculated to the resolution specified in the CapSense Customizer. The position filters are applied to the result if enabled. This function is available only if a radial slider is defined by the CapSense customizer.

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

```
#define CapSense_CSHL_RS_"widget_name"            5
```

Return Value: uint16: Position value of the slider.

Side Effects: If any radial slider slot is active, the function returns values from zero to the resolution value set in the CapSense customizer. If no slots are active, the function returns -1. If an error occurs during execution of the centroid/diplexing algorithm, the function returns -1.
You can use the CSHL_CheckIsSlotActive() routine to determine which slider segments are touched, if required.

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

uint8 CapSense_CSHL_GetDoubleCentroidPos(uint8 widget)

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

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

```
#define CapSense_CSHL_TP_"widget_name"            5
```

Return Value: uint8: 1 if finger is on the touchspad, 0 if not.

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

```
CapSense_CSHL_TPCol_"widget_name"_Results - position of X
CapSense_CSHL_TPRow_"widget_name"_Results - position of Y
```

Scan Control APIs

These APIs start and stop the component and control scanning functions.

Function	Description
CapSense_Start	Initializes registers for each module and starts the component. This function calls functions automatically depending on modules selected in the customizer. This function should be called prior to calling any other component functions.
CapSense_Stop	Stops the slot scanner, disables interrupts, and resets all slots to an inactive state.
CapSense_ScanSlot	Calls the function <METHOD_NAME>_ScanSlot where <METHOD_NAME> is method name for this scan slot in CapSense customizer.
CapSense_ScanAllSlots	**Parallel mode**: Scans slots from both sides in parallel. One slot from the right side and one slot from the left side are scanned at a time with or without synchronization depends of CapSense customizer choose. If one side has more slots than the other then the remaining slots on the side with more are scanned singly. The scanning ends when all slots are scanned. **Serial mode**: Scans all slots by calling <METHOD_NAME>_ScanAllSlots() function for each method selected in customizer.

void CapSense_Start

Description: Initializes registers for each module and starts the component. This function calls functions automatically depending on modules selected in the customizer. This function should be called prior to calling any other component functions.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_Stop

Description: Stops the slot scanner, disables interrupts, and resets all slots to an inactive state

Parameters: None

Return Value: None

Side Effects: None

void CapSense_ScanSlot(uint8 slot)

Description: Calls the function <METHOD_NAME>_ScanSlot where <METHOD_NAME> is the defined module name for this scan slot in the CapSense customizer.

Parameters: slot: uint8 – Scan slot number

Return Value: None

Side Effects: None

void CapSense_ScanAllSlots

Description: **Parallel mode**: Scans slots from both sides in parallel. One slot from the right side and one slot from the left side are scanned at a time with or without synchronization depends of CapSense customizer choose. If one side has more slots than the other then the remaining slots on the side with more are scanned singly. The scanning ends when all slots are scanned.

Serial mode: Scans all slots by calling <METHOD_NAME>_ScanAllSlots() function for method selected in customizer.

Parameters: None

Return Value: None

Side Effects: None

Method Specific APIs

These API functions depend on the CapSense method selected in CapSense customizer. For example, if the method chosen is CSD, the instance name is "CapSense_CSD".

Some Method Specific API functions are appended with "Left" and "Right" in Parallel mode. APIs appended with "Left" work only with the left side of CapSense system. Those appended with "Right" work only with the right side.

Function	Description
CapSense_CSD_Start	Initializes registers and starts the CSD method of CapSense component.
CapSense_CSD_Stop	Stops the slot scanner, disables internal interrupts, and calls CSD_ClearSlots() to reset all slots to an inactive state.

PRELIMINARY

Function	Description
CapSense_CSD_ScanSlot	Sets scan settings and scans the selected scan slot.
CapSense_CSD_ScanAllSlots	Scans all scan slots by calling CSD_ScanSlot() for each slot index.
CapSense_CSD_SetSlotSettings	Sets the scan settings of selected scan slot.
CapSense_CSD_SetRBleed	Sets the pin to use for the bleed resistor (Rb) connection. This function can be called at runtime to select the current Rb pin setting from those defined customizer. The function overwrites the component parameter setting. This function is available only if Rb configuration is defined by the CapSense customizer.
CapSense_CSD_ClearSlots	Clears all slots to the non-sampling state by sequentially disconnecting all slots from Analog MUX Bus and connecting them to GND.
CapSense_CSD_EnableSensor	Configures the selected sensor to measure during the next measurement cycle. The corresponding pins are set to Analog High-Z mode and connected to the Analog Mux Bus. This also enables the comparator function.
CapSense_CSD_DisableSensor	Disables the selected sensor. The corresponding pin is disconnected from the Analog Mux Bus and connected to GND, High_Z or Shield electrode.
CapSense_CSD_ReadSlot	Returns scan slot raw data from the SlotResult[] array. Each scan slot has a unique number within the slot array. This number is assigned by the CapSense customizer in sequence.

void CapSense_CSD_Start

Description: Initializes registers and starts the CSD method of CapSense component.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_CSD_Stop

Description: Stops the slot scanner, disables internal interrupts, and calls CSD_ClearSlots() to reset all slots to an inactive state

Parameters: None

Return Value: None

Side Effects: None

void CapSense_CSD_ScanSlot(uint8 slot)

Description: Sets scan settings and scans the selected scan slot. Each scan slot has a unique number within the slot array. This number is assigned by the CapSense customizer in sequence.

Parameters: slot: uint8 – Scan slot number

Return Value: None

Side Effects: None

void CapSense_CSD_ScanAllSlots

Description: Scans all scan slots by calling CSD_ScanSlot() for each slot index.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_CSD_SetSlotSettings(uint8 slot)

Description: Sets the scan settings of selected scan slot.. Each setting has a unique number within the settings array and is connected to corresponding scan slot. This number is assigned by the CapSense customizer in sequence.

Parameters: slot: uint8 – Scan slot number

Return Value: None

Side Effects: None

void CapSense_CSD_SetRBleed(uint8 rbleed)

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

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

Parameters: rbleed: uint8 – Ordering number for bleed resistor terminal defined in CapSense customizer.

Return Value: None

Side Effects: None

void CapSense_CSD_ClearSlots

Description: Clears all slots to the non-sampling state by sequentially disconnecting all slots from Analog MUX Bus and connecting them to GND.

Parameters: None

Return Value: None

Side Effects: None

void CapSense_CSD_EnableSensor(uint8 sensor)

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

Parameters: sensor: uint8 – Sensor Number

Return Value: None

Side Effects: None

void CapSense_CSD_DisableSensor(uint8 sensor, uint8 state)

Description: Disables the selected sensor. The corresponding pin is disconnected from the Analog Mux Bus and connected to GND, High_Z or Shield electrode.

Parameters: sensor: uint8 – Sensor Number

state: uint8 – State of sensor when disabled, Possible states include:

- CapSense_DISABLE_STATE_GND
- CapSense_DISABLE_STATE_HIGHZ
- CapSense_DISABLE_STATE_SHIELD

Return Value: None

Side Effects: None

uint16 CapSense_CSD_ReadSlot(uint8 slot)

Description: Returns scan slot raw data from the SlotResult[] array. Each scan slot has a unique number within the slot array. This number is assigned by the CapSense customizer in sequence.

Parameters: slot: uint8 – Scan Slot Number

Return Value: uint16: Current Raw data value

Side Effects: None

Data Structures

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

In Parallel mode all data structures appended with “Left” and “Right.” Data structures appended with “Left” contain values for the left side of CapSense system. Those appended with “Right” contain values for the right side. In Serial mode the structures are not divided left and right. The data structures documented here assume serial mode for simplicity.

CapSense_CSHL_SlotBaseline[]

This array holds the baseline data of each slot.

CapSense_CSHL_SlotBaselineLow[]

This array holds the fractional byte of baseline data of each slot. The arrays size is equal to the total number of scan slots minus the number of generic scan slots. The CapSense_CSHL_SlotBaseline[] and CapSense_CSHL_SlotBaselineLow[] array is updated by these functions:

- CapSense_CSHL_InitializeAllBaselines()
- CapSense_CSHL_InitializeSlotBaseline()
- CapSense_CSHL_UpdateSlotBaseline()
- CapSense_CSHL_UpdateAllBaselines()

CapSense_CSHL_SlotSignal[]

This array holds the difference of count by subtracting the previous baseline and noise threshold from the current raw count value of each slot. The array size is equal to the total number of scan slots minus the number of generic scan slots.

CapSense_CSHL_SlotOnMask[]

This is a byte array that holds the slots on or off state. Generic slots are excluded from it.

CapSense_CSHL_SlotOnMask[0]

This array contains the masked bits for slots 0 through 7 (slot 0 is bit 0, slot 1 is bit 1). CapSense_CSHL_SlotOnMask[1] contains the masked bits for slots 8 through 15 (if needed), and so on. This byte array holds as many elements as are necessary to contain the total number of scan slots minus number of generic scan slots. The value of a bit is 1 if the button is on and 0 if the button is off. The CapSense_CSHL_SlotOnMask[] data is updated by functions:

- CapSense_CSHL_CheckIsSlotActive()

- CapSense_CSHL_CheckIsAnySlotActive()

CapSense_SlotResult[]

This array contains the raw data of each scan slot. The array size is equal to the total number of scan slots (CapSense_TOTAL_SCANSLOT_COUNT). The CapSense_SlotResult[] data is updated by these functions:

- CapSense_ScanSlot()
- CapSense_ScanAllSlots()

CapSense_PortShiftTable[]

This array contains a port and mask for every sensor. The PortMask data structure contains fields:

- Port – Defines the port number of the sensor.
- Shift – Defines pin within the port.

CapSense_ScanSlotTable[]

This array contains all scan slots in the CapSense system. The ScanSlot data structure contains fields:

- RawIndex – Contains the the place in the SlotResult[] array where the raw data is placed after scanning. This field should not be edited manually for proper CapSense component operation.
- IndexOffset – Contains offset for gung proximity and generic widgets. This field should not be edited manually for proper CapSense component operation.
- SnsCnt – This field contains the number of sensors in this scan slot. This field should not be edited manually for proper CapSense component operation.
- WidgetNumber – This field contains the widget that the sensor belongs to. If this field contains 0xFF the current scan slot has no widget. Generic scan slots use 0xFF as their widget number. This field should not be edited manually for proper CapSense component operation.
- DebounceCount – Contains the debounce counter to sensor active transition. This field should not be edited manually for proper CapSense component operation.

CapSense_SettingsTable[]

This array contains the settings of every scan slot in the CapSense system. The CSD_Settings data structure contains the following fields:

- IdacRange – Contains the range selection parameter for the IDAC. This parameter is only available in configurations with an IDAC. The following constants are provided:

```
CapSense_IDAC_RANGE_32uA
CapSense_IDAC_RANGE_255uA
CapSense_IDAC_RANGE_2mA
```

- IdacSettings – Contains the IDAC value. The capacitance measurement range depends on this parameter. A higher value indicates a wider range. Possible values are 1 to 255. This parameter is only available in configurations with an IDAC.
- PrescalerPeriod – Contains the value of the prescaler period register and determines the precharge switch output frequency. This parameter is only available on configurations with a prescaler. The prescaler period values can range from 2 to 255.
- Resolution – Contains the scanning resolution in bits. The sensors can be scanned with resolutions ranging from 8 to 16 bits. The maximum raw count for a scanning resolution for N bits is 2N-1. The following constants are provided.

```
CapSense_PWM_RESOLUTION_8_BITS
CapSense_PWM_RESOLUTION_9_BITS
CapSense_PWM_RESOLUTION_10_BITS
CapSense_PWM_RESOLUTION_11_BITS
CapSense_PWM_RESOLUTION_12_BITS
CapSense_PWM_RESOLUTION_13_BITS
CapSense_PWM_RESOLUTION_14_BITS
CapSense_PWM_RESOLUTION_15_BITS
CapSense_PWM_RESOLUTION_16_BITS
```

- ScanSpeed – This field affects the sensors' scanning speed. The available selections are: Ultra Fast, Fast, Normal, Slow. The following constants are provided:

```
CapSense_SCAN_SPEED_ULTRA_FAST
CapSense_SCAN_SPEED_FAST
CapSense_SCAN_SPEED_NORMAL
CapSense_SCAN_SPEED_SLOW
```

- PrsPolynomial – Contains the polynomial value for PRS. This parameter is available for configuration with PRS only.
- Disable State – Defines the unscanning sensor connection as GND, High-Z and Shield electrode. The following constants are provided:

```
CapSense_DISABLE_STATE_GND
CapSense_DISABLE_STATE_HIGHZ
CapSense_DISABLE_STATE_SHIELD
```

CapSense_CSHL_WidgetTable[]

This array contains all widgets in CapSense system. The WidgetTable data structure contains:

- Type – Contains the type of widget. The types of widgets are: Buttons, Sliders (Linear and Radial), TouchPads, Matrixbuttons, Proximity. Generic sensors do not have a High Level API so there are no instances for them in the widget table. This field should not be edited manually for proper CapSense component operation.
- RawOffset – Contains the start position in the SlotResult[] array. This field should not be edited manually for proper CapSense component operation.

- ScanSlotCount – Contains the number of scan slots within the widget. This field should not be edited manually for proper CapSense component operation.
- FingerThreshold – Contains the software value that is used to determine if a finger is present on the sensor.
- NoiseThreshold – Contains a value that indicates the level of noise in the capacitive scan. The baseline algorithm tracks and filters the noise, but if the measured count value exceeds the noise threshold, the baseline algorithm stops updating.
- Debounce – Contains a debounce counter. In order for the sensor to transition from inactive to active the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified.
- Hysteresis – Contains the hysteresis value for the widget. If hysteresis is desired, the slot will not be considered on or active until the count value exceeds the finger threshold plus the hysteresis value. The slot will not be considered off or inactive until the measured count value drops below the finger threshold minus the hysteresis value.
- Filters – Contians the bit filed with raw counts and position filters. This field should not be edited manually for proper CapSense component operation.
- AdvancedSettings – Contains pointer to structure with advanced settings fot widget. Every widget have different structure of advanced settings. This field should not be edited manually for proper CapSense component operation.

Constants

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

- CapSense_TOTAL_SCANSLOT_COUNT – The total number of scan slots in the CapSense component.
- CapSense_TOTAL_SCANSLOT_COUNT_LEFT – The total number of scan slots in the left half of the CapSense component (exists only in parallel mode).
- CapSense_TOTAL_SCANSLOT_COUNT_RIGHT – The total number of scan slots in the right half of the CapSense component (exists only in parallel mode).
- CapSense_TOTAL_GENERIC_SCANSLOT_COUNT – The total number of Generic scan slots in the CapSense component.
- CapSense_TOTAL_GENERIC_SCANSLOT_COUNT_LEFT – The total number of scan slots in the left half of Capsense Component (exists only in parallel mode).
- CapSense_TOTAL_GENERIC_SCANSLOT_COUNT_RIGHT - Total number of scan slots in the right half of the CapSense component (exists only in parallel mode).

Sensor Constants

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

- CapSense_CSD_EnableSensor
- CapSense_CSD_DisableSensor

The constant names consist of:

Instance name + _SENSOR + *Widget Type* + *Widget Name* + *element* + *Side*

For example:

```
/* Parallel mode */
#define CapSense_SENSOR_TP_TP1_ROW_0_LEFT     0
#define CapSense_SENSOR_TP_TP1_ROW_1_LEFT     1
#define CapSense_SENSOR_TP_TP1_COL_0_LEFT     2
#define CapSense_SENSOR_PROX_PROX1_RIGHT      0

/* Serial mode */
#define CapSense_SENSOR_TP_TP1_ROW_0    0
#define CapSense_SENSOR_TP_TP1_ROW_1    1
#define CapSense_SENSOR_TP_TP1_COL_0    2
#define CapSense_SENSOR_BTN_UP          3
```

Scan Slot Contants

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

- CapSense_CSD_SetSlotSettings()
- CapSense_CSD_ScanSlot()
- CapSense_CSD_ReadSlot()
- CapSense_CSHL_InitializeSlotBaseline()
- CapSense_CSHL_UpdateSlotBaseline()
- CapSense_CSHL_CheckIsSlotActive()

The constant names consist of:

Instance name + _SCANSLOT + *Widget Type* + *Widget Name* + *element* + *Side*

Widget Type – There are several widget types:

Alias	Description
BTN	Buttons
LS	Linear Sliders
RS	Radial Sliders

PRELIMINARY

Alias	Description
TP	Touch Pads
MB	Matrix Buttons
PROX	Proximity Sensors
GEN	Generic Sensors

- **Widget Name** – The user-defined name of the widget (must be a valid C style indentifier). The widget name must be unique within the widget type. For example, you can have a BTN_MyName and a PROX_MyName but you may not have two aliases BTN_MyName.
- **Element Number** – The element number only exists for widgets that have multiple elements, such as radial sliders. For touch pads and matrix buttons the element number consists of the word 'Col' or 'Row' and its number (for example: Col_0, Col_1, Row_0, Row_1). For linear and radial sliders, the element number consists of the character 'e' and its number (for example: e_0, e_1, e_2, e_3).
- **Side** – In parallel mode sides are Left and Right. Sides do not exist in serial mode.

Examples:

```
/* Parallel mode */
#define CapSense_SCANSLOT_TP_TP1_ROW_0_LEFT  0
#define CapSense_SCANSLOT_TP_TP1_ROW_1_LEFT  1
#define CapSense_SCANSLOT_TP_TP1_COL_0_LEFT  2
#define CapSense_SCANSLOT_PROX_PROX1_RIGHT   0

/* Serial mode */
#define CapSense_SCANSLOT_TP_TP1_ROW_0 0
#define CapSense_SCANSLOT_TP_TP1_ROW_1 1
#define CapSense_SCANSLOT_TP_TP1_COL_0 2
#define CapSense_SCANSLOT_BTN_UP       3
```

Widget Constants

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

- CapSense_CSHL_GetCentroidPos()
- CapSense_CSHL_GetRadialCentroidPos()
- CapSense_CSHL_GetDoubleCentroidPos()

The constants consist of:

Instance name + _CSHL + *Widget Type* + *Widget Name*

For example:

```
#define CapSense_CSHL_BTN_B1            0
#define CapSense_CSHL_BTN_B2            1
#define CapSense_CSHL_RS_SL2            7
```

For widgets that have columns and rows (TouchPad and Matrix buttons) to Widget Type add ROW or COL . The constants look like these examples:

```
#define CapSense_CSHL_TPCOL_TP1            0
#define CapSense_CSHL_TPROW_TP1            2
#define CapSense_CSHL_MBCOL_MB1            1
#define CapSense_CSHL_MBROW_MB1            3
```

The TouchPad has separate define to provide as argument to function CapSense_CSHL_GetDoubleCentroidPos():

```
#define CapSense_CSHL_TP_TP1               0
```

Sample Firmware Source Code

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

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

```
#include <device.h>

void main()
{
    CYGlobalIntEnable;

    /* Initialize LCD */
    LCD_Char_1_Start();
    LCD_Char_1_Position(0,0);
    LCD_Char_1_PrintString("B1     B2");

    /* Initialize the CapSense component */
    CapSense_1_Start();
    CapSense_1_CSHL_InitializeAllBaselines();

    /* Sensor Scanning Loop */
    while(1)
    {
        CapSense_1_CSD_ScanAllSlots();
        CapSense_1_CSHL_UpdateAllBaselines();

        /* Position LCD pointer to update button 1 status */
        LCD_Char_1_Position(1,0);

        /* Left button pressed */
        if (CapSense_1_CSHL_CheckIsSlotActive(CapSense_1_SCANSLOT_BTN_B1))
        {
            /* Action for button 1 active */
            LCD_Char_1_PrintString("On ");
```

PRELIMINARY

```
        }
        else
        {
             /* Action for button 1 inactive */
             LCD_Char_1_PrintString("Off");
        }

        /* Position LCD pointer to update button 2 status */
        LCD_Char_1_Position(1,6);

        /* Check if B2 is pressed */
        if (CapSense_1_CSHL_CheckIsSlotActive(CapSense_1_SCANSLOT_BTN_B2))
        {
            /* Action for button 2 active */
            LCD_Char_1_PrintString("On ");
        }
        else
        {
            /* Action for button 2 inactive */
            LCD_Char_1_PrintString("Off");
        }
    }
}
```

Pin Assignments

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

Sides

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

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

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

Sensor Pins – CapSense_cPort – Pin Assignment

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

The aliases for sensors are:

Widget Type + Widget Name + Element Number

Widget Type

There are several widget types:

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

Widget Name

The user-defined name of the widget (must be a valid C style indentifier). The widget name must be unique within the widget type. For example, you can have a BTN_MyName and a PROX_MyName but you may not have two aliases BTN_MyName.

Element Number

The element number only exists for widgets that have multiple elements, such as radial sliders. For touch pads and matrix buttons the element number consists of the word ‘Col’ or ‘Row’ and its number (for example: Col_0, Col_1, Row_0, Row_1). For linear and radial sliders, the element number consists of the character ‘e’ and its number (for example: e_0, e_1, e_2, e_3).

Note In parallel sensing applications, widget elements that belong to left side of can connect only to even (left side) ports. Widget elements that belong to the right side can connect only to odd (right side) ports. The Pin Editor does not verify correct pin assignment.

Note The Opamp outputs P0[0], P0[1], P3[6] and P3[7] have greater parasitic capacitance than other pins. This causes less finger response from P0[0], P0[1], P3[6] and P3[7] in CapSense applications.

CapSense_cCmod_Port – Pin Assignment

One side of the external modulator capacitor (Cmod) should be connected to a physical pin and the other to GND. Parallel configurations require two Cmod capacitors, one for the left side and one for the right side. The Cmod can be connected to **any pin**, but for direct connection (do not overuse routing resources) use pins:

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

The aliases for the Cmod capacitors are:

Alias	Description
Cmod	Cmod for serial mode applications.
lCmod	Left Cmod in parallel mode applications.
rCmod	Right Cmod in parallel mode applications.

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

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

When the *CSD, IDAC disable, use external Rb* method is being used, the external Rb feedback resistor value should be selected before experimenting to determine the optimal Cmod value.

CapSense_cRb_Ports – Pin Assignment

An external bleed resistor (Rb) is required when the *CSD, IDAC disable, use external Rb* configuration is selected. The Rb should be connected to a physical pin and to the modulator capacitor (Cmod).

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

The aliases for external bleed resistors are:

Side + ***Rb*** + *Number*

Alias	Description
Side	Either ‘l’ for the left side or ‘r’ for the right side. The side prefix is not used in serial applications.
Number	Multiple bleed resistors on the same side are given a sequence number, 0, 1, or 2. For example, rRb0, rRb1.

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

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

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

CapSense_cShield_Port – Pin Editor

Shield electrodes are available only in the *CSD, IDAC sinking* and *CSD, IDAC disable, use external Rb* configurations. Shield electrodes (Shield) should be connected to a physical pin and a shield electrode layer on the board. The maximum number of shield electrodes is three.

The aliases for shield electrodes are:

Side + **Shield** + *Number*

Alias	Description
Side	Either ‘l’ for the left side or ‘r’ for the right side. The side prefix is not used in serial applications.
Number	Multiple shield electrodes on the same side are given a sequence number, 0, 1, or 2. For example, rShield0, rShield1.

Note The maximum quantity of sensors, shield-electrodes, and bleed resistors is 62.

Interrupt Service Routines

The CapSense Component uses interrupt that triggers after end of sensor scan. Stub routine is provided where you can add your own code. The routine stubs is generated in the CapSense_INT.c file the first time the project is built. The number of interrupts depends on CapSense mode selection: one for Serial and two for Parallel. The conditional compilation is used to provide one stub routine for Serial and Parallel modes. Your code must be added between the provided comment tags as follows.

For Serial mode

```
CY_ISR(CapSense_1_ISR)
{
         /*  Place your Interrupt code here. */
         /* `#START CapSense_1_ISR` */

         /* `#END` */
         CapSense_1_status &= ~CapSense_1_START_CAPSENSING;

         #if(CYDEV_CHIP_DIE_EXPECT == CYDEV_CHIP_DIE_LEOPARD)
               #if defined(CapSense_1_CSD_METHOD)
                     #if((CYDEV_CHIP_REV_EXPECT <= CYDEV_CHIP_REV_LEOPARD_ES2) &&
(CapSense_1_sbCSD_cISR__ES2_PATCH))
                           CapSense_1_ISR_PATCH();
                     #endif
```

PRELIMINARY

```
                #endif
                #if defined(CapSense_1_CSA_METHOD)
                        #if((CYDEV_CHIP_REV_EXPECT <= CYDEV_CHIP_REV_LEOPARD_ES2) &&
(CapSense_1_sbCSA_cISR__ES2_PATCH))
                                CapSense_1_ISR_PATCH();
                        #endif
                #endif
        #endif
}
```

For parallel mode(Left and Right sides)

```
CY_ISR(CapSense_1_ISRLeft)
{
        /*  Place your Interrupt code here. */
        /* `#START CapSense_1_ISRLeft` */

        /* `#END` */
        CapSense_1_statusLeft &= ~CapSense_1_START_CAPSENSING;

        #if(CYDEV_CHIP_DIE_EXPECT == CYDEV_CHIP_DIE_LEOPARD)
                #if defined(CapSense_1_CSD_METHOD_LEFT)
                        #if((CYDEV_CHIP_REV_EXPECT <= CYDEV_CHIP_REV_LEOPARD_ES2) &&
(CapSense_1_lbCSD_cISR__ES2_PATCH))
                                CapSense_1_ISR_PATCH();
                        #endif
                #endif
                #if defined(CapSense_1_CSA_METHOD_LEFT)
                        #if((CYDEV_CHIP_REV_EXPECT <= CYDEV_CHIP_REV_LEOPARD_ES2) &&
(CapSense_1_lbCSA_cISR__ES2_PATCH))
                                CapSense_1_ISR_PATCH();
                        #endif
                #endif
        #endif
}

CY_ISR(CapSense_1_ISRRight)
{
        /*  Place your Interrupt code here. */
        /* `#START CapSense_1_ISRRight` */

        /* `#END` */
        CapSense_1_statusRight &= ~CapSense_1_START_CAPSENSING;

        #if(CYDEV_CHIP_DIE_EXPECT == CYDEV_CHIP_DIE_LEOPARD)
                #if defined(CapSense_1_CSD_METHOD_RIGHT)
                        #if((CYDEV_CHIP_REV_EXPECT <= CYDEV_CHIP_REV_LEOPARD_ES2) &&
(CapSense_1_rbCSD_cISR__ES2_PATCH))
                                CapSense_1_ISR_PATCH();
                        #endif
                #endif
                #if defined(CapSense_1_CSA_METHOD_RIGHT)
                        #if((CYDEV_CHIP_REV_EXPECT <= CYDEV_CHIP_REV_LEOPARD_ES2) &&
(CapSense_1_rbCSA_cISR__ES2_PATCH))
                                CapSense_1_ISR_PATCH();
```

```
                    #endif
                #endif
            #endif
    }
```

Functional Description

Definitions

Sensor

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

Scan Slot

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

CapSense Widget

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

FingerThreshold

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

NoiseThreshold

Determines the level of noise in the capacitive scan. The baseline algorithm tracks and filters the noise. If the measured count value exceeds the noise threshold, the baseline algorithm stops updating.

Debounce

Adds a debounce counter to the sensor active transition. In order for the sensor to transition from inactive to active the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified.

Hysteresis

Sets the hysteresis value used with the finger threshold. If hysteresis is desired, the slot will not be considered "ON" or "Active" until the count value exceeds the finger threshold PLUS the hysteresis value. The slot will not be considered "OFF" or "Inactive" until the measured count value drops below the finger threshold MINUS the hysteresis value.

Resolution – Interpolation and Scaling

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

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

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

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

Slider sensor count and resolution are set in the CSD Wizard. A scaling value is calculated by the wizard and stored as fractional values.

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

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

The resolution is found by using this equation:

Resolution = (Number of Sensors - 1) x Multiplier

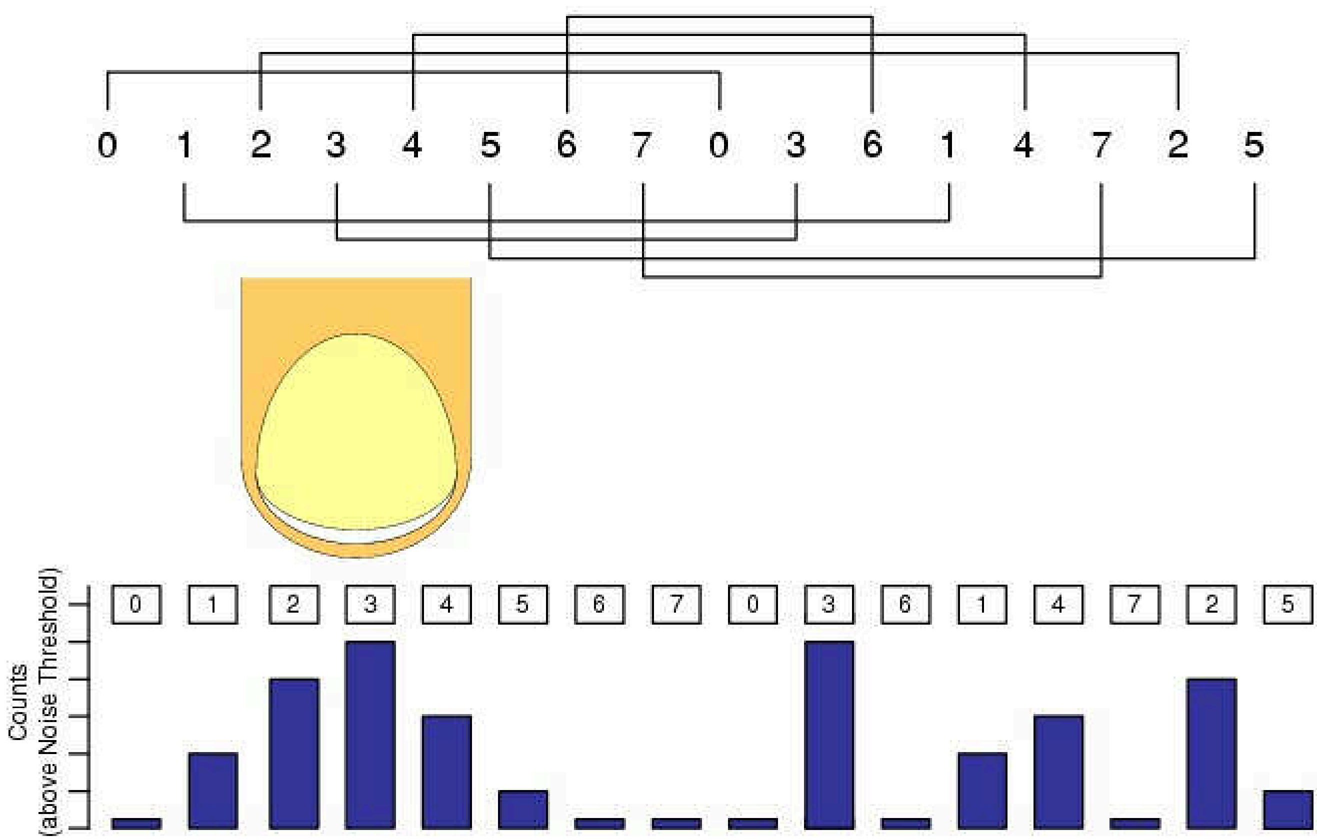
The centroid is held in a 24-bit unsigned integer and its resolution is a function of the number of sensors and the multiplier.

Diplexing

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

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

Figure 1 Diplexing, Index by Three



PRELIMINARY

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

Diplexing Sequence for Different Slider Segment Counts

Total Slider Segment Count	Segment Sequence
10	0,1,2,3,4,0,3,1,4,2
12	0,1,2,3,4,5,0,3,1,4,2,5
14	0,1,2,3,4,5,6,0,3,6,1,4,2,5
16	0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5
18	0,1,2,3,4,5,6,7,8,0,3,6,1,4,7,2,5,8
20	0,1,2,3,4,5,6,7,8,9,0,3,6,9,1,4,7,2,5,8
22	0,1,2,3,4,5,6,7,8,9,10,0,3,6,9,1,4,7,10,2,5,8
24	0,1,2,3,4,5,6,7,8,9,10,11,0,3,6,9,1,4,7,10,2,5,8,11
26	0,1,2,3,4,5,6,7,8,9,10,11,12,0,3,6,9,12,1,4,7,10,2,5,8,11
28	0,1,2,3,4,5,6,7,8,9,10,11,12,13,0,3,6,9,12,1,4,7,10,13,2,5,8,11
30	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0,3,6,9,12,1,4,7,10,13,2,5,8,11,14
32	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,3,6,9,12,15,1,4,7,10,13,2,5,8,11,14
34	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14
36	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14,17
38	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,0,3,6,9,12,15,18,1,4,7,10,13,16,2,5,8,11,14,17
40	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11,14,17
42	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11,14,17,20
44	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,2,5,8,11,14,17,20
46	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20
48	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20,23
50	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,0,3,6,9,12,15,18,21,24,1,4,7,10,13,1

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

Filters

Several filters are provided in the CapSense component: median, averaging, first order IIR and jitter. The filters are divided into two categories: raw data and position.

Position Median Filter

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

Position Averaging Filter

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

Position First Order IIR FILTER

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

1st-Order IIR filter with selectable coefficients:

IIR = ½ previous + ½ current (Default)

IIR = ¾ previous + ¼ current

Position Jitter Filter

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

PRELIMINARY

Raw Data Median Filter

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

Raw Data Averaging Filter

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

Raw First Order IIR FILTER

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

1st-Order IIR filter with selectable coefficients:

IIR = ½ previous + ½ current (Default)

IIR = ¾ previous + ¼ current

Raw Data Jitter Filter

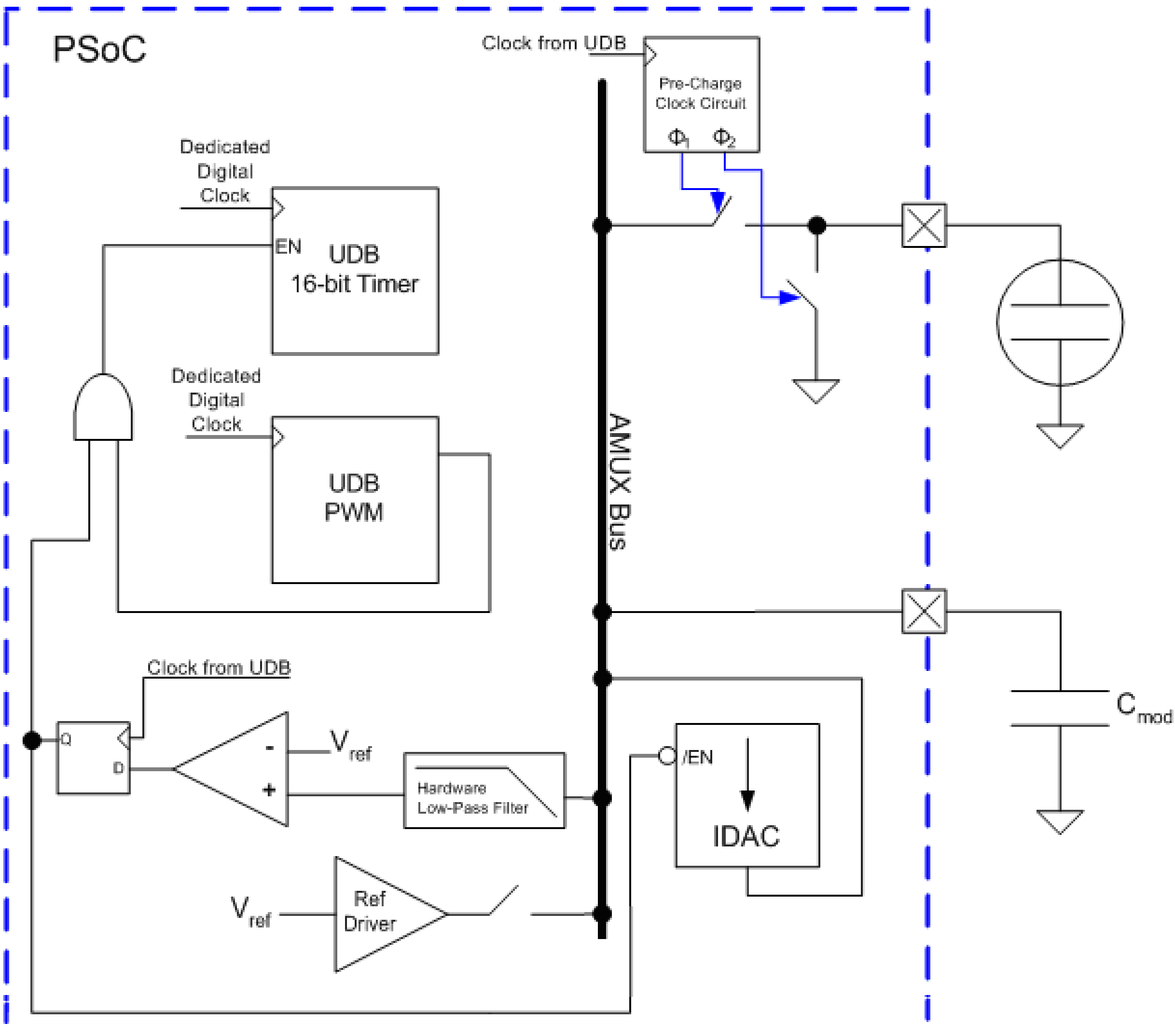
This filter eliminates noise in the the raw data that toggles between two values (jitter). It is most effective when applied to data that contains noise of four LSBs peak-to-peak or less. Enabling this filter consumes TBD Flash and (Number of Sensors × 2) bytes of RAM. It is disabled by default.

Block Diagram and Configuration

The CSD (Capacitive Sensing using a Delta-Sigma Modulator) provides capacitance sensing using the switched capacitor technique with a delta-sigma modulator to convert the sensing switched capacitor current to digital code. It allows implementation of buttons, sliders, touchpads and touchscreens using arrays of conductive sensors. High level software routines allow for enhancement of slider resolution using diplexing, and compensation for physical and environmental sensor variation.

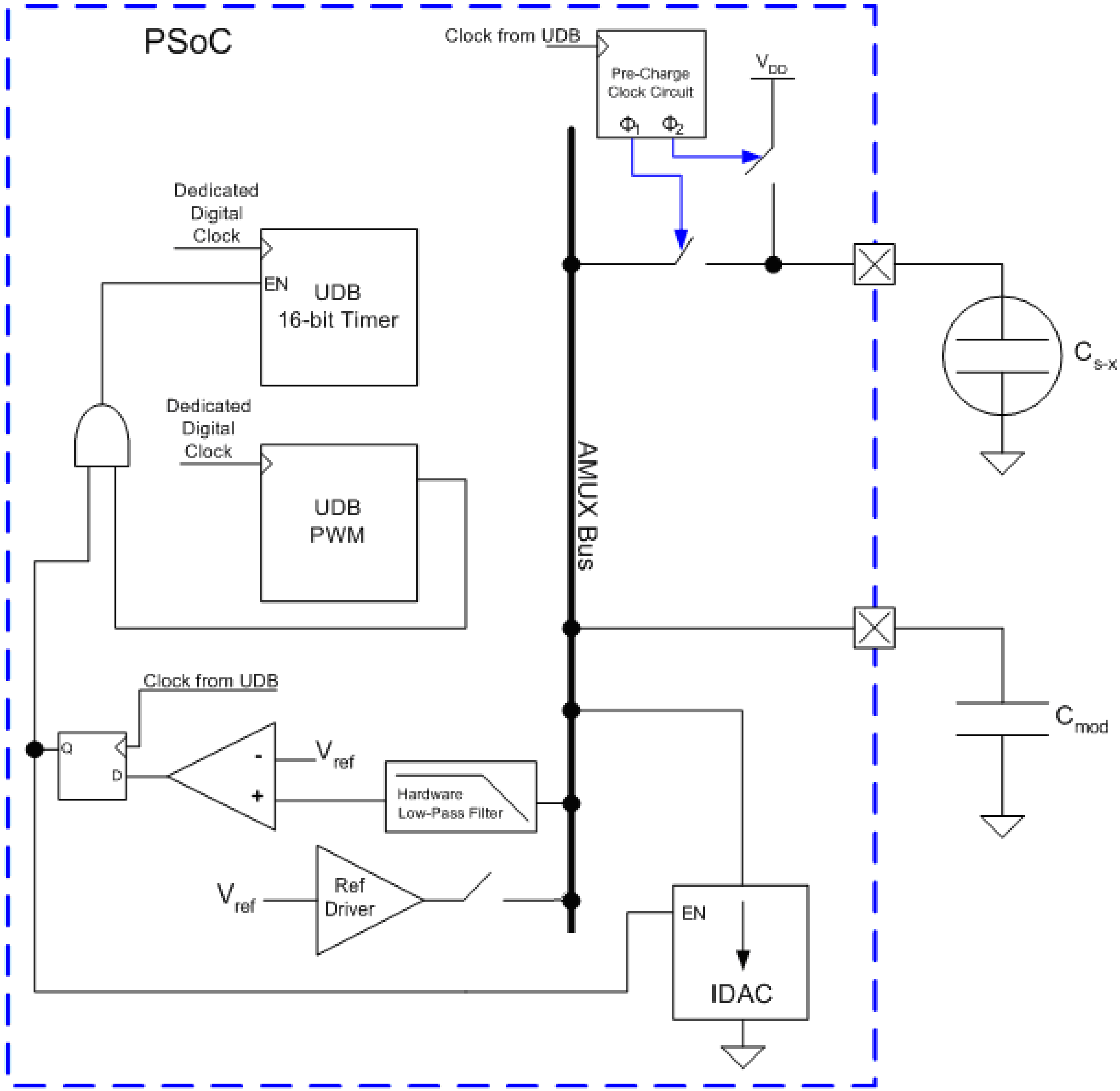
IDAC sourcing

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



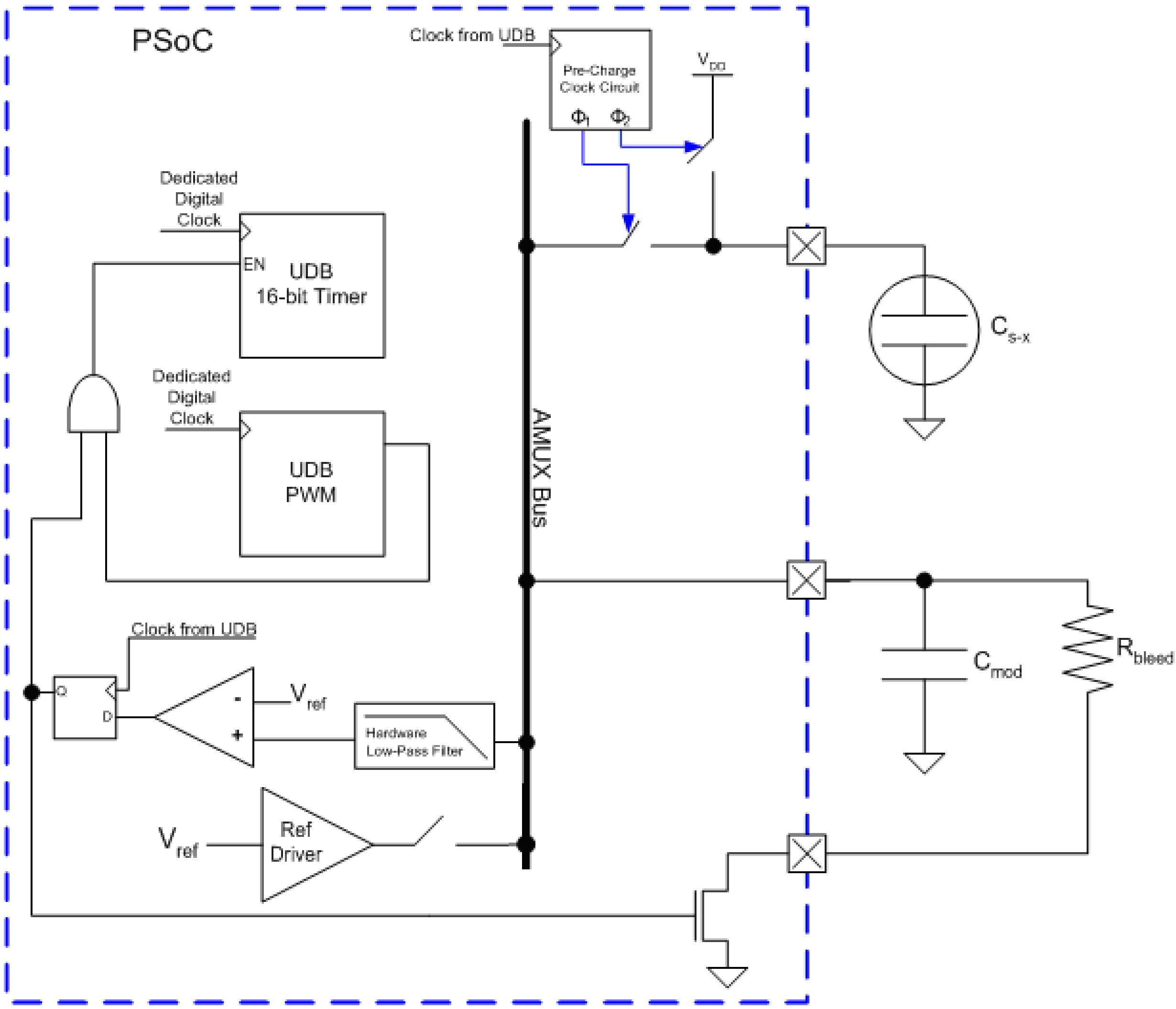
IDAC sinking

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



IDAC disable, use external R_b

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



PRELIMINARY

DC and AC Electrical Characteristics

5.0V/3.3V DC and AC Electrical Characteristics

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

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.20.b	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
1.20.a	Moved local parameters to formal parameter list.	To address a defect that existed in PSoC Creator v1.0 Beta 4.1 and earlier, the component was updated so that it could continue to be used in newer versions of the tool. This component used local parameters, which are not exposed to the user, to do background calculations on user input. These parameters have been changed to formal parameters which are visible, but un-editable. There are no functional changes to the component but the affected parameters are now visible in the “expression view” of the customizer dialog.