CapSense® Sigma-Delta Plus ADC Datasheet CSDADC V 1.50

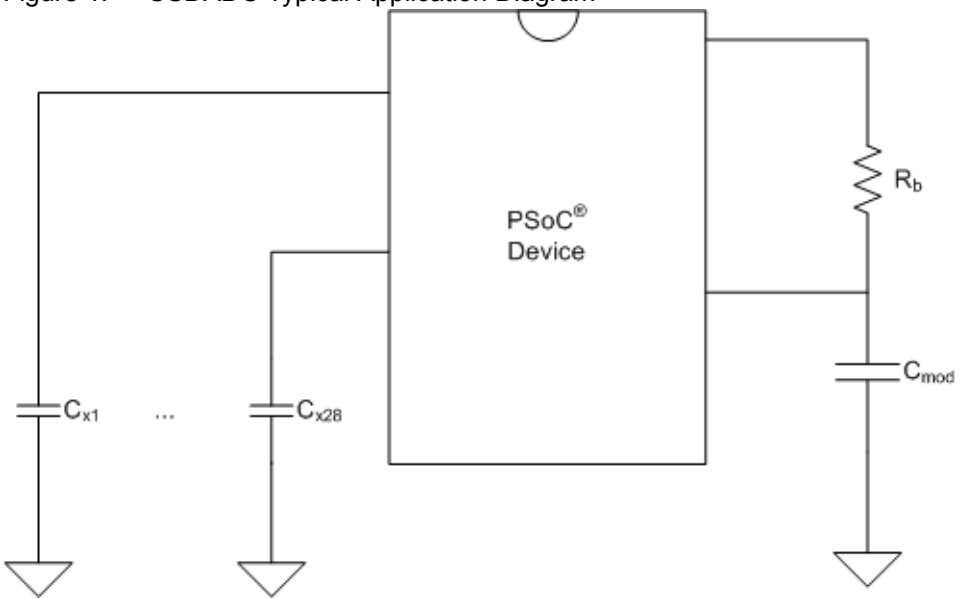
Copyright © 2008-2013 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes) Typical		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C21x34, CY8CLED04. Use of flash, RAM, and pins varies by the number of sensors and configuration.						
PRS16-based user module with 1 sensor	3	2	1	1491	36	2-5
PRS8 based user module with 1 sensor	2	2	1	1401	36	2-5
Prescaler based user module with 1 sensor	2	2	1	1426	36	2-5
VC2 clock source user module with 1 sensor	1	2	1	1351	36	2-5
Each additional CapSense® button	-	-	-	10	2	1
Static code and RAM increase when capacitive slider with five elements is used	-	-	-	79	583	5
Each additional slider element	-	-	-	10	2	1
Static code and RAM increase when slider diplexing is used	-	-	-	Sliders*2	0	-
Die Temperature measurement	-	-	-	380	2	-

Features and Overview

- Allows scanning CapSense sensors and measuring input voltages without using separate loadable configurations
- ADC features:
  - Absolute and ratiometric ADC input modes with runtime mode switching allows you to easily scan different sensor types
  - Single slope ADC for absolute voltage input mode
  - Built-in calibration mechanism for the single-slope ADC
  - Integrating incremental ADC for ratiometric input mode
  - Allows a coarse temperature measurement: range of -40 °C to +125 °C, accuracy of ± 40 °C with resolution ± 2 °C
- CapSense® features:
  - Based on the robust CSD method
  - Scans 1 to 28 capacitive sensors
  - Sensing possible with up to a 15-mm glass overlay
  - Proximity detection to 20 cm with a wire-based sensor
  - High immunity to AC mains noise, EMC noise, and power supply voltage changes
  - Supports different combinations of independent and slide capacitive sensors
  - Double slide sensor physical resolution using diplexing
  - Increase slide sensor resolution using interpolation
  - Touchpad support with two slide sensors
  - Sensing support through high-resistive conductive materials (ITO films for example)
  - Shield electrode support for reliable operation in the presence of water film or droplets
  - Guided sensor and pin assignments using the CSDADC Wizard
  - Integrated baseline update algorithm for handling temperature, humidity, and electrostatic discharge (ESD) events
  - Easily adjustable operational parameters
  - PC GUI application support for raw data monitoring and parameter optimization in real-time

Figure 1. CSDADC Typical Application Diagram



Quick Start

1. Select and place user modules that require dedicated pins (for example, I2C and LCD). Assign ports and pins as required.
2. Select and place the CSDADC User Module.
3. Right-click the CSDADC User Module to access the CSDADC Wizard.
4. Set the CapSense sensor count, configuration, and pin assignments.
5. Set pins and global parameters. Read all the parameter descriptions and follow the requirements and guidelines.
6. Generate the application and switch to the Application Editor.
7. Adapt the sample code as required to implement independent sensors, sliding sensors, or a touchpad.
8. Connect the RS232 level translator or USB-I^{2}C bridge to the target board, and optimize the parameters using a GUI.
9. Change the CSDADC parameters and rebuild the application.
10. Program the PSoC device and verify module operation. Tune the CSDADC parameters to achieve a 5:1 SNR requirement as discussed in the CY8C21x34/B CapSense Design Guide.

Functional Description

The CSDADC combines capacitance sensing with ADC functionality for voltage measurement without using separate loadable configurations. It saves code space by reusing modules common to both the CSD and ADC. You should use the CSDADC when you need both capacitance sensing and ADC functionality. Applications that need one or the other should use the CSD User Module or the ADC8 or ADC10.

CSDADC provides capacitance sensing by using the switched capacitor technique with a sigma-delta modulator (CSD) to convert the sensing switched capacitor current to a digital code. It is easy to interface with different sensor types because it can use both ratiometric and absolute input ADC modes and can switch modes at run time. For example, use the ratioi metric mode to measure the temperature with a thermistor and absolute voltage measurement when measuring a battery voltage.

The CSDADC User Module is implemented using a single slope ADC for absolute input voltage mode and an incremental ADC for ratiometric.

This user module datasheet provides basic information about the CSD and ADC. For a more detailed discussion of the theory and recommendations about setting parameters, use tips, and troubleshooting, please refer to the CSD and ADC10 datasheets and associated application notes at www.cypress.com. If this is your first time implementing either a capacitive sensing application or using an ADC you should read these documents before you begin.

Read the following documents before you use the CSDADC User Module for the first time:

- *CY8C21x34 PSoC Programmable System-on-ChipTechnical Reference Manual*, especially the following sections:
  - Two Column Limited Analog
  - Digital Clocks
  - I/O Analog Multiplexer
- CSD User Module Datasheet
- ADC10 User Module Datasheet

The following design guides are recommended after reading the CSD User Module datasheet. These documents are available on the Cypress Semiconductor website at 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

ADC Operation Description

The CSDADC supports two ADC types:

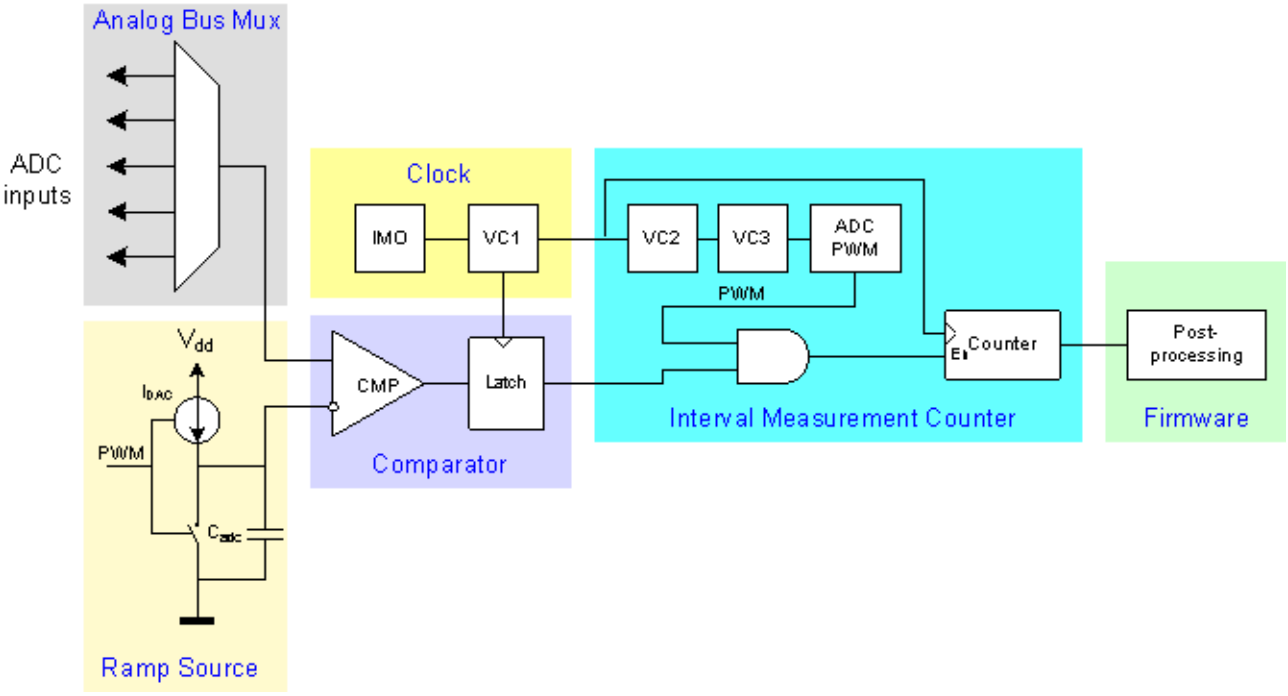
- Single slope. A single slope ADC supports an absolute input voltage mode, where the ADC reading is directly proportional to the input signal and does not depend on the PSoC power supply voltage.
- Incremental. An incremental ADC supports the ratiometric operation mode where ADC readings are directly proportional to the input voltage and are inversely proportional to the PSoC power supply voltage.

There are differences in the ADC sampling mechanisms. A single slope ADC converts the signal value at the moment it is sampled, where the ratiometric ADC converts the signal during the whole conversion interval due its integration nature. For this reason, you should use the incremental ADC if the input signals are noisy.

Single Slope ADC Operation

Configuration of the single slope ADC is the same as for the ADC10 User Module.

Figure 2. CSDADC Block Diagram, Absolute Input Voltage ADC Configuration



The Single Slope A/D Converter generates up to a 12-bit, full scale output (0 to 4095 count range). The sampling rate is determined by your configuration of the clock sources and selected parameters.

The ADC is constructed with these PSoC resources:

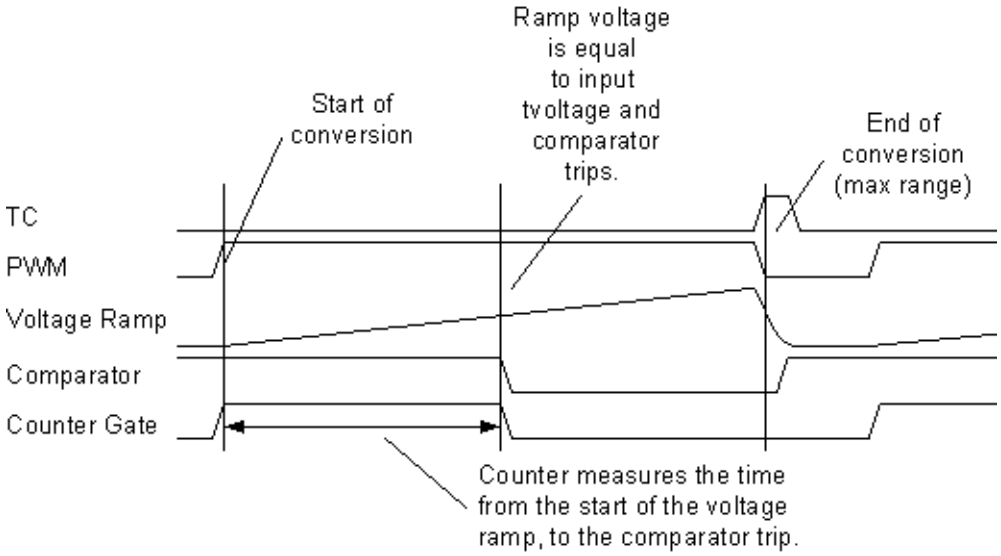
- An Analog CT Block, configured as a comparator.
- An Analog SC Block, configured as a ramp generator using internal capacitor and current source.
- A Digital Block, configured as an eight-bit counter.
- A dedicated PWM used for ADC timing.

A block diagram of the single slope ADC implementation is shown in Figure 2. The core of the conversion algorithm involves a current source, an integrating capacitor, and a comparator. When the current source is activated, a linear voltage ramp is generated on the capacitor. The capacitor voltage is one input to an analog comparator circuit; the other is the analog input voltage to be converted. The comparator is low until the ramp voltage exceeds the input voltage, then it transitions low. The counter keeps track of the time between the start of the ramp and the time when the comparator is tripped.

The basic conversion waveform is shown in Figure 3. In the case of a 12-bit sample, the high time of the PWM is set so that the counter reaches 4096. The low time of the PWM is designed to allow the capacitor to discharge and the answer to be processed. The terminal count of the PWM is used as an interrupt to

read the results of the conversion. Bit-7 of ADC_CR is set high if the ramp voltage never exceeds the input value. The PWM generator is clocked only from VC3 and has a limited selection of high and low times.

Figure 3. Single Slope ADC Conversion Timing Diagram



The ADC input is a sample hold of the comparator input. The sample and hold circuitry is controlled by the PWM. This guarantees that the signal remains constant while the conversion is taking place.

Calibration of Single-Slope ADC

You must calibrate the ADC before you use it. Each analog column has an ADC capacitor trim register for this purpose. This register controls the capacitor value that determines the slope of the ADC voltage ramp. The CAPVAL [7:0] bits of the ADC_TR register are used for calibration. The goal of this calibration process is to tune the ramp time (slope) so that a full-scale ADC input value results in a full scale ADC code. This is accomplished by matching the ramp time to the required full-scale conversion period. If the PSoC device operates from a power supply that is not well regulated, the calibration procedure should be run periodically because the DAC current depends on the power supply voltage.

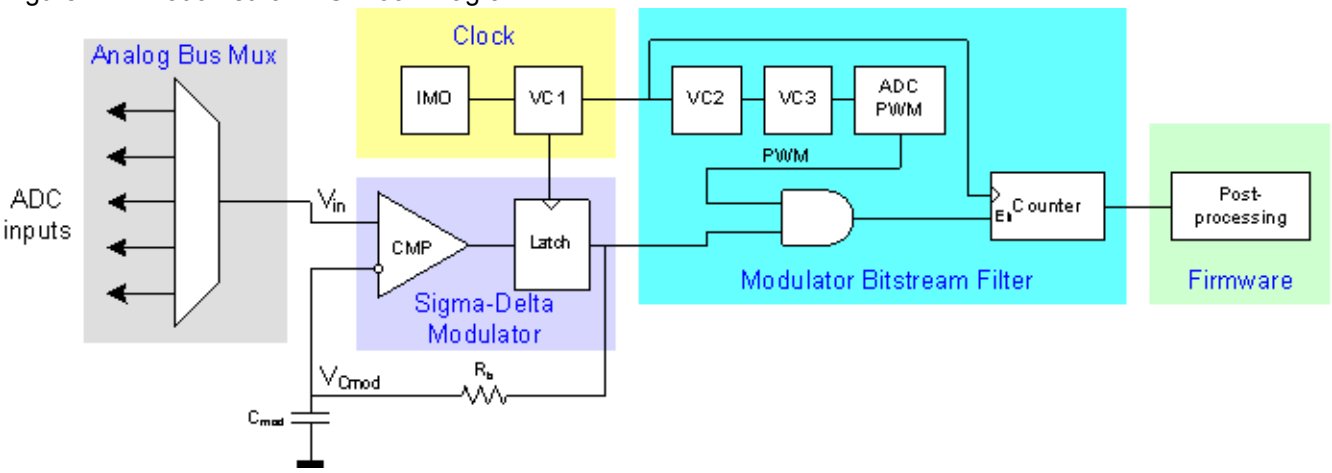
The user module API includes a routine called CSDADC_iCal that executes a calibration algorithm to trim the ADC_TR register based upon a calibration voltage and calibration count value.

The return value of the CSDADC_iCal routine is the result of running the ADC on the calibration signal. Ideally, it should be the expected calibration count value. If it is not, software methods can be used to guarantee accuracy.

Incremental ADC

The incremental ADC operation is based on a combination of a first-order sigma-delta modulator and counter-based modulator bitstream digital filter. A block diagram of this ADC type is shown in Figure 4:

Figure 4. Ratiometric ADC Block Diagram



This ADC consists of the following parts:

- Sigma-delta modulator
- Modulator bit stream filter
- Clock source
- Resolution
- Sample rate

The modulator is formed with a comparator, a comparator latch, a modulation capacitor, C_{mod}, and a feedback resistor, R_b. If the modulation capacitor voltage is below the input voltage V_{in}, the comparator output is the high V_{dd} level, charging the modulator capacitor. When the modulator capacitor voltage crosses the input voltage, the comparator output switches to the low Vss level. This causes the modulator capacitor voltage to drop. Once the modulator voltage crosses a input voltage again, the modulator capacitor voltage starts rising again, repeating the modulation capacitor charge/discharge cycles. The latch makes the comparator operation synchronize to the clock VC1 signal and limits the minimum capacitor charge/discharge intervals. The sigma-delta modulator keeps the voltage V_{Cmod} close to the input voltage V_{in} in average by alternatively charging/discharging a modulator capacitor C_{mod}.

The modulator math is simple. Suppose a modulator bit steam duty cycle is d_{mod}. The modulator reference voltage is:

Equation 1

$$V_{Cmod} = V_{dd} \cdot d_{mod}$$

Taking into account that $V_{Cmod} = V_{in}$ on average, the modulator duty cycle can be calculated using the following equation:

Equation 2

$$d_{mod} = \frac{V_{in}}{V_{dd}}$$

The resolution of the ADC in CSDADC is fixed to 12-bit if the mode = CSDADC_ABSOLUTE. The resolution of the ADC in CSDADC is configurable (Resolution parameter sets it) if the mode = CSDADC_RATIOMETRIC.

The sample rate depends on the Resolution and Scanning speed parameters. There is no equation for it in this case because these parameters are fixed: Resolution = 12 (for CSDADC_ABSOLUTE) or Resolution = 9 to 16 (for CSDADC_RATIOMETRIC); and Scanning speed = Ultra Fast, Fast, Normal, and Slow. Refer to Table 10 and Table 11 in this datasheet where the Scanning Time is given. The Sample Rate is inversely proportional to Scanning Time. For example, if Scanning Time = 1010 µsec (Resolution = 12, Scanning speed = Normal), then the Sample Rate = 1/1010 = 990 samples/sec.

ADC Modulator Bit Stream Filter and Clock Source

The modulator converts the input voltage to the output bit stream duty cycle. The duty cycle is measured using a digital filter. The implementation on the CY8C21x34 devices uses a counter-based filter, because the CY8C21x34 devices have no dedicated decimator. The filter consists of the 8-bit hardware counter and a measurement interval forming circuit. The counter is extended to 16 bits by handling the terminal count overflows in firmware to save hardware resources. The counter uses the common VC1 clock signal with a modulator. The counter preserves state when the modulator output is low and decrements by one when modulator output is high.

The measurement interval forming circuit uses periodic interrupts to read the counter and gates the counter enable input, to prevent the counter value from changing during read. The duty cycle measurement interval N_m in the internal main oscillator (IMO) is determined by the VC1, VC2, VC3, and ADCPWM values by the following equation:

Equation 3

$$N_m = VC1 \cdot VC2 \cdot VC3 \cdot N_{ADCPWM}$$

VC1, VC2, VC3, N_{ADCPWM} – divider values.

The digital filter forms one sample each N_m IMO cycle, and the filter sample value is linearly proportional to the total sensing capacitance C_s. The maximum sample value is determined by N_{max}:

Equation 4

$$N_{max} = VC2 \cdot VC3 \cdot N_{ADCPWM}$$

Equation 5

$$N_s = d_{mod} N_{max}$$

Equation 6

$$N = \frac{V_{in}}{V_{dd}} \cdot VC2 \cdot VC3 \cdot N_{ADCPWM}$$

The user module uses two interrupts: one counter overflow interrupt to extend the 8-bit hardware counter maximum count, and the ADCPWM interrupt. The ADCPWM interrupt is routed to the Comparator 0 bus interrupt. Two comparator buses are used, even though only one comparator is turned on.

Die Temperature Measurement

The ADC provides measurement of the on-chip die temperature circuit output signal. The signal is available as PMux analog input of the continuous time block ACE01. This signal is approximately 1.0 V nominally and ranges are approximately between 0.7 V and 1.3 V. The output voltage is a function of the die temperature.

Capacitance Measurement Operation

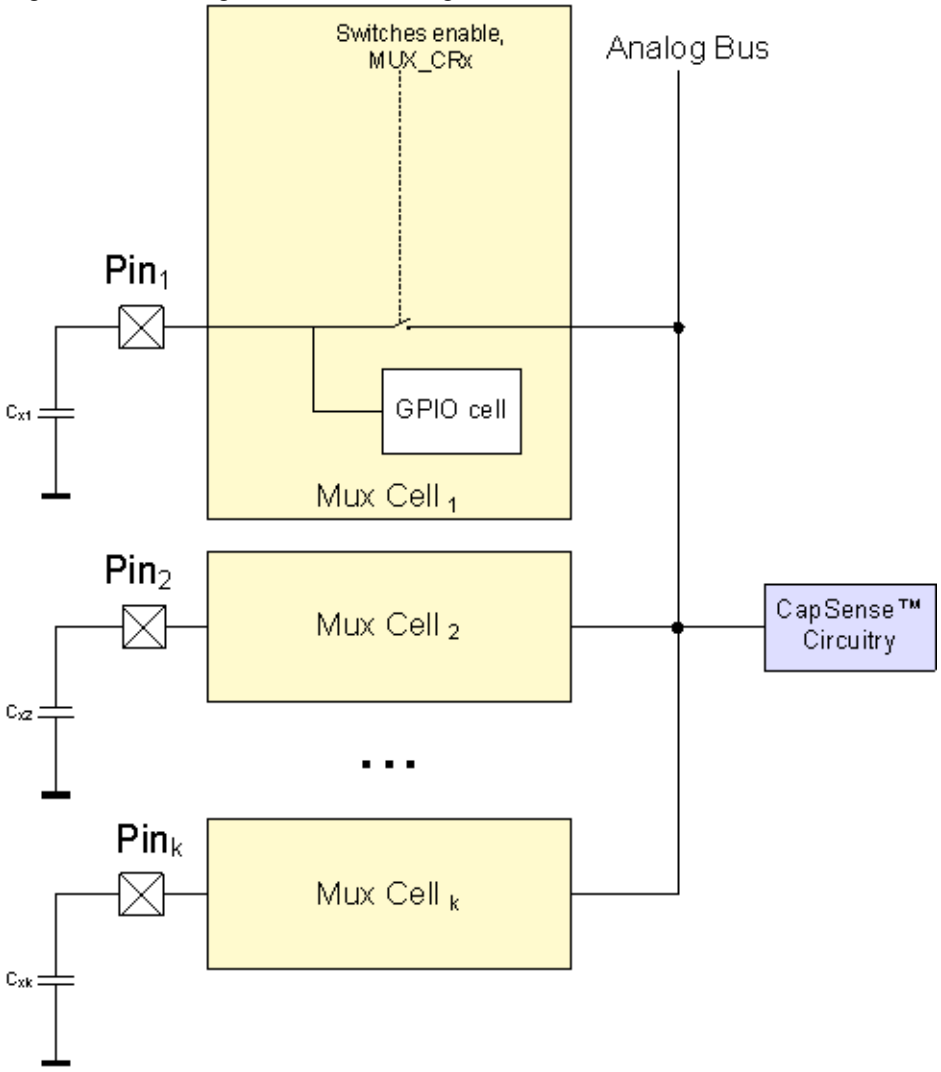
The decision logic is implemented in the firmware. The firmware analyzes the capacitance measurement, tracks the slow capacitance change due to environmental factors, and runs decision logic to detect button touches and calculate the slider position.

Scanning an Array of Sensors

The CY8C21x34 family of devices have a built-in analog bus. It allows capacitive sensor connections to any PSoC pin. The CSDADC User Module uses internal precharge switches to charge active sensors at clock signal phase Ph_1 and connects the Analog Bus to the sensor at phase Ph_2. The sigma-delta modulator 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.

Figure 5. Analog Bus with Precharge Switches

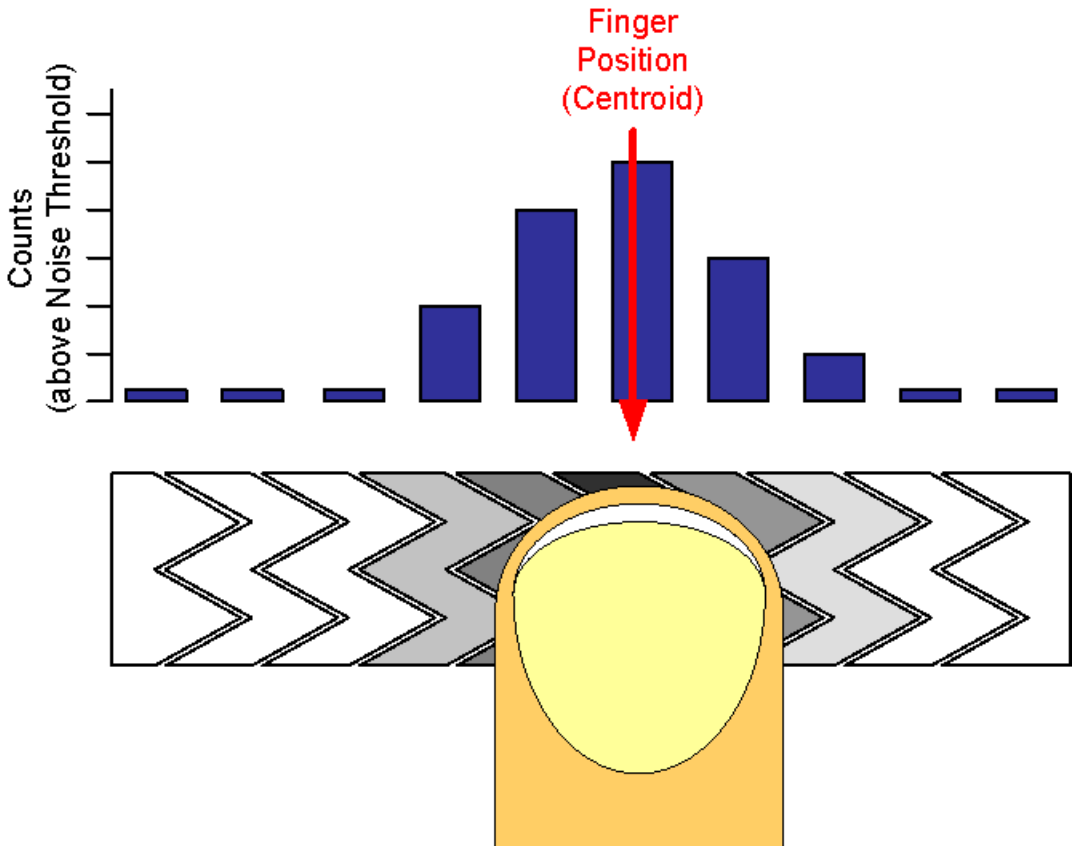


Sliders

Sliders are used for controls that require gradual adjustments. Examples include a lighting control (dimmer), volume control, graphic equalizer, and speed control. These sensors are mechanically adjacent to one another. Actuation of one sensor results in partial actuation of physically adjacent sensors. The actual position in the slider is found by computing the centroid location of the set of activated sensors. Sliders are accommodated in the CSDADC Wizard, by establishing groups in which each group of sliders

has a specific order. The practical lower limit for the number slider sensors is five, the upper limit is simply the number of sensor positions available on the PSoC device selected.

Figure 6. Ordering Physical Sensor Locations



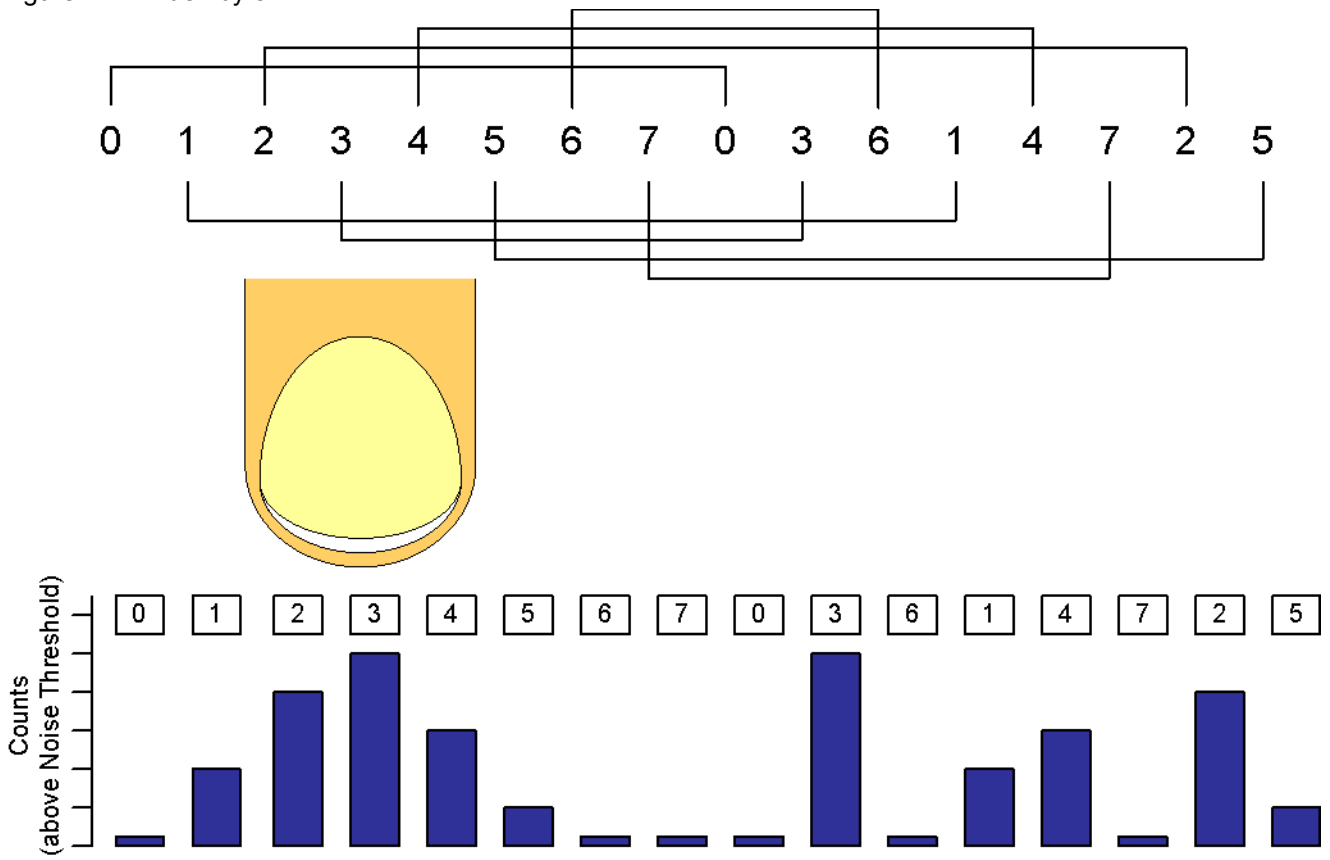
The close proximity of strong signals in one half of the slider results in the same levels aliased into the upper half, but the results are scattered. The sensing algorithms search for strong adjacent sets of signals to declare the resolved slider position.

Diplexing

Each PSoC sensor connection in a 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 CSDADC Wizard. The second (or upper) half of the physical sensor locations is automatically mapped by an algorithm in the Wizard 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. Take care to determine this order and map it onto the printed circuit board.

There are many 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 user module is to index by three.

Figure 7. Index by 3



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 CSDADC Wizard when you select diplexing. The following table illustrates the diplexing sequences for different slider segments count.

Table 1. 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

Total Slider Segment Count	Segment Sequence
24	0,1,2,3,4,5,6,7,8,9,10,11,0,3,6,9,1,4,7,10,2,5,8,11
26	0,1,2,3,4,5,6,7,8,9,10,11,12,0,3,6,9,12,1,4,7,10,2,5,8,11
28	0,1,2,3,4,5,6,7,8,9,10,11,12,13,0,3,6,9,12,1,4,7,10,13,2,5,8,11
30	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0,3,6,9,12,1,4,7,10,13,2,5,8,11,14
32	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,3,6,9,12,15,1,4,7,10,13,2,5,8,11,14
34	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14
36	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14,17
38	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,0,3,6,9,12,15,18,1,4,7,10,13,16,2,5,8,11,14,17
40	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11,14,17
42	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11,14,17,20
44	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,2,5,8,11,14,17,20
46	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20
48	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20,23
50	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20,23
52	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,25,2,5,8,11,14,17,20,23
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

Slider Segment Selection Guidelines for the Diplex Slider

Selecting the number of segments needed for a slider depends on the physical length of the slider. However, special care must be taken when you decide the number of segments for a diplexing slider.

In a diplexing slider design, one sensor is used as two different physical slider segments to increase the physical length of slider. The number of segments that are completely covered by a finger touch must be less than the number of sensors between two segments derived from the same sensor. This ensures the proper working of the diplex slider.

For example, in the case of a 10-segment slider (5 sensors), two slider segments derived from sensor 3 are separated by only two sensors (sensor 4 and 0). In this case, a finger touch must not completely cover more than two sensor segments to ensure the proper working of the slider.

For a 12-segment slider, one finger touch must not cover more than 3 segments. Similarly, for a 18-segment slider, one finger touch must not completely cover more than 4 segments.

Interpolation and Scaling

In applications for sliding sensors and touchpads it is often necessary to determine the 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:

Equation 7

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

The calculated value is typically fractional. 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 CSDADC 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

You can calculate the resolution using this equation:

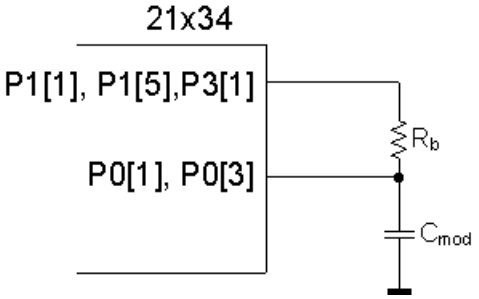
Resolution = (Number of Sensors -1) x Multiplier

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

Feedback Component Selection Guidelines

The user module requires an external modulation capacitor C_{mod} and a modulator feedback resistor R_b. The capacitor can be connected to the P0[1], P0[3] port pins, and Vss ground. The feedback resistor R_b can be connected to port pins P1[1], P1[5], P3[1] and the capacitor pin. The pins are selected with a user module parameter setting. Do not use pins selected for connecting modulator components for any other purposes. User modules that consume specific pin resources, including the LCD and I2CHW, must be placed before establishing port pin connections for the CSDADC User Module. The configuration selections are reflected in the Wizard when it is opened.

Figure 8. External Components Connection



The recommended value for the modulation capacitor is 4.7 to 47 nF. The optimal capacitance can be selected by experiment to get maximum SNR. A value of 5.6 to 10 nF gives good results in the most cases for the PRS16 and PRS8 configuration. If a configuration with a prescaler is selected, the recommended value of the integration capacitor is 22 to 47 nF. You can experiment with several capacitor values to get the best SNR after selecting the feedback resistor. A ceramic capacitor should be used. The temperature capacitance coefficient is not important. The resistor values depend on the total sensor capacitance C_s. The resistor value should be selected using the following steps:

- 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 increase when resistor values increase.

Typical values are 500 Ω to 10 kΩ depending on the sensor capacitance. You can start with 2 kΩ if you are using the CY3213 evaluation board.

Shielding Electrode

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

In some applications it is useful to select the shielding electrode signal and its placement relative to the sensing electrode such that increasing the coupling between these electrodes causes the opposite of the touch change of the sensing electrode capacitance measurement. This simplifies the high-level software API work. The CSDADC User Module supports separate output for the shielding electrode.

Figure 9. Possible Shield Electrode PCB Layout

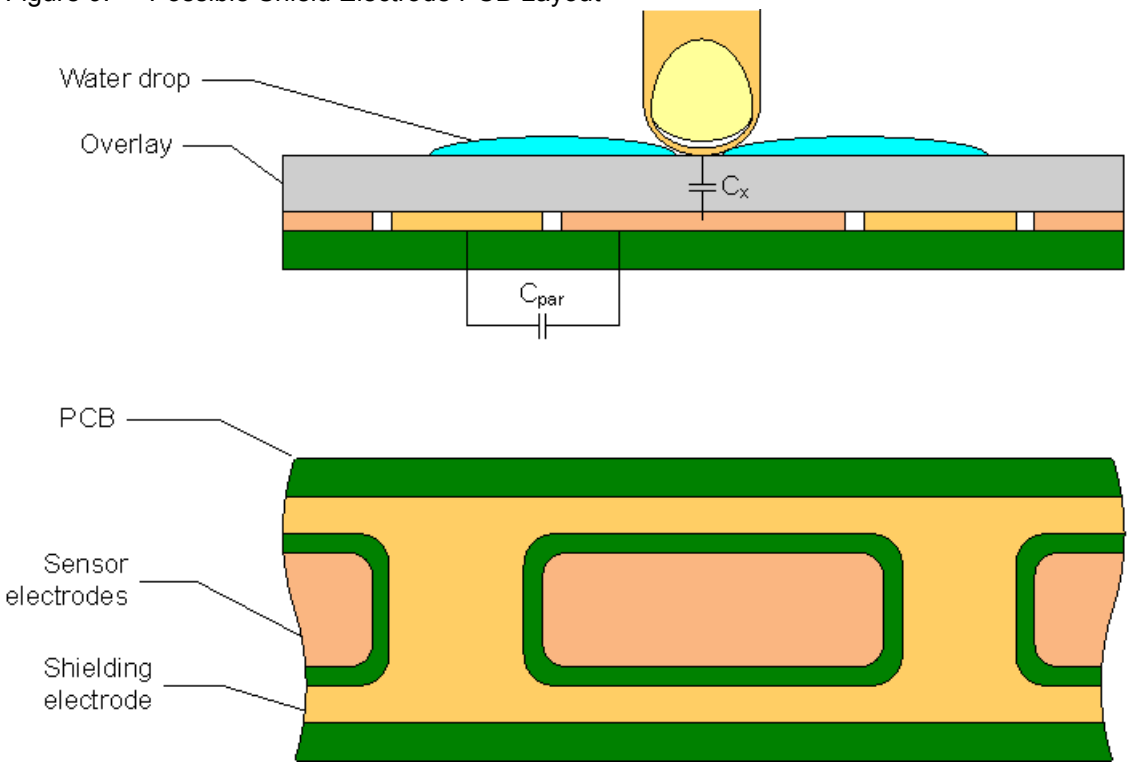


Figure 9 illustrates one possible layout configuration for the button's shield electrode. The shield electrode is especially useful for transparent ITO touchpad devices, where it blocks the LCD drive electrode's noise influence and reduces stray capacitance at the same time.

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

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

Clock Source

The clock source is used to control the switches on the sensing capacitor. The user module supports four selection options as the clock source for the precharge switches. The following table compares the configurations:

Configuration	Operation Frequency	Digital Blocks Used	EMC Noise Immunity
PRS16	Spread-spectrum, average is $F_{IMO}/4$, peak is $F_{IMO}/2$	3	High. Sensitive points are multiples of the PRS sequence repeat period and PRS fundamental frequency F_{IMO}.
PRS8	Adjustable spread spectrum, $F_{IMO}/4 – F_{IMO}/512$	2	Moderate. Sensitive at more points due to the shorter PRS repeat period. If good EMI immunity is required, please use a PRS16 configuration.
Prescaler	fixed, IMO/(Period+1)	2	Device is sensitive for EMC signals at operation frequency and its harmonics. Recommended only for operation via high-resistance materials.
VC_2	fixed, IMO/(VC_1xVC_2)	1	Device is sensitive for EMC signals at operation frequency and its harmonics. Recommended only when no certification EMC/EMI tests are planned.

Modulator Reference Source

The comparator reference source is used to form the comparator reference voltage. The reference voltage value determines the sensitivity.

The following table summarizes the reference selection options:

External components	UM selection	When to use
none	VBG	Readings are proportional to the power supply voltage. Use only when power supply is well regulated
none	ASE11	Recommended for most applications. Try to start testing from this option.
2	AnalogColumn Input Select	Readings are less dependent on power supply. Recommended R1 = 10k; R2 = 3.6k
2	AnalogColumn Input Select	If other reference selections have too much noise.

It is likely that you will use only the VBG and ASE11 reference selections. The other two can be useful in special applications.

DC and AC Electrical Characteristics

Table 2. Power Supply Voltage

Parameter	Min	Typical	Max	Unit	Test Conditions and Comments
Value	2.7	5.0	5.25	V	

Table 3. Capacitance Sensing Noise

Parameter[a]	Min	Typical	Max	Unit	Test Conditions (Vdd = 5 V, SysClk =24 MHz, CPU Clock = 12MHz, Baseline >= 70% of Resolution Max Count), PRS16
Noise Counts		4		peak-to-peak	Resolution = 14
Noise Counts		7		peak-to-peak	Resolution = 12
Noise Counts		12		peak-to peak	Resolution = 10

a. SNR increases as the Scan Speed slows and the Baseline counts increase.

Table 4. Capacitance Sensing Noise

Parameter[a]	Min	Typical	Max	Unit	Test Conditions (Vdd = 3.3 V, SysClk =12 MHz, CPU Clock = 6 MHz, Baseline >= 70% of Resolution Max Count)
Noise Counts, peak-peak		0.2		%, (noise counts)/ (baseline counts)	Resolution >= 14
Noise Counts, peak-peak		1		%, (noise counts)/ (baseline counts)	Resolution = 12
Noise Counts, peak-peak		10		%, (noise counts)/ (baseline counts)	Resolution = 10

a. SNR increases as the Scan Speed slows and the Baseline counts increase.

Table 5. Capacitance Sensing Noise

Parameter[a]	Min	Typical	Max	Unit	Test Conditions (Vdd = 2.7 V, SysClk =12 MHz, CPU Clock = 6 MHz, Baseline >= 70% of Resolution Max Count)
Noise Counts		9		peak-to-peak	Resolution = 12

a. SNR increases as the Scan Speed slows and the Baseline counts increase.

Table 6. Power Consumption During Capacitance Scanning

Parameter	Min	Typical	Max	Unit	Test Conditions (Vdd = 2.7 V, SysClk = CPU Clock = 6 MHz)
Active Current		1.43		mA	Average current during scan, 8 sensors
Standby Current		40		µA	Scanning Speed = Ultra Fast, Resolution = 9, 100ms report rate, 8 sensors
		250		µA	Scanning Speed = Fast, Resolution = 12 100 ms report rate, 8 sensors
Sleep/Wake Current		5		µA	1s report rate, 1 sensor

Table 7. Power Consumption During Capacitance Scanning

Parameter	Min	Typical	Max	Unit	Test Conditions (Vdd = 3.3 V, SysClk = CPU Clock = 6 MHz)
Active Current		1.9		mA	Average current during scan, 8 sensors
Standby Current		50		µA	Scanning Speed = Ultra Fast, Resolution = 9, 100ms report rate, 8 sensors
		300		µA	Scanning Speed = Fast, Resolution = 12 100 ms report rate, 8 sensors
Sleep/Wake Current		6		µA	1s report rate, 1 sensor

Table 8. Power Consumption During Capacitance Scanning

Parameter	Min	Typical	Max	Unit	Test Conditions (Vdd = 5.0 V, SysClk = CPU Clock = 6 MHz)
Active Current		3.18		mA	Average current during scan, 8 sensors
Standby Current		90		µA	Scanning Speed = Ultra Fast, Resolution = 9, 100ms report rate, 8 sensors
		550		µA	Scanning Speed = Fast, Resolution = 12 100 ms report rate, 8 sensors
Sleep/Wake Current		7		µA	1s report rate, 1 sensor

Table 9. Incremental ADC Characteristics

Parameter	Typical	Limit	Unit	Conditions and Notes
Operating Current		3.18	mA	5-V power supply. CPU 6 MHz
Noise, ratiometric		3	LSB	
Resolution, ratiometric		9 to 16	bits	
Resolution, absolute		12	bits	
Sample Rate, ratiometric		0.04 to 16.6	Ksps	Depends on resolution and scanning speed

Parameter	Typical	Limit	Unit	Conditions and Notes
Sample Rate, absolute		0.58 to 4.6	Ksps	
Input				
Input Voltage Range	-	Vss+0.3 to Vdd-0.3	V	
Input Capacitance		3	pF	

Placement

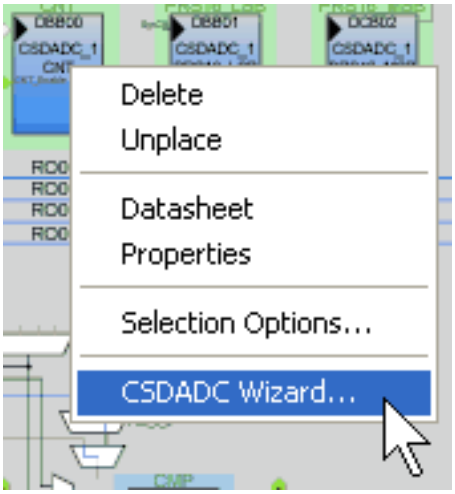
The blocks for the user module are automatically placed when the user module is instantiated; alternate placements are not available.The CSDADC consumes from 1 to 3 digital blocks depending on the configuration. The analog part of the CSDADC is placed in ACE00, ACE01, and ASE11; no alternate placements are available.

User modules that consume specific pin resources, including the LCD and I2CHW, must be placed before establishing port pin connections for the CSDADC User Module. The configuration selections are reflected in the Wizard when it is opened.

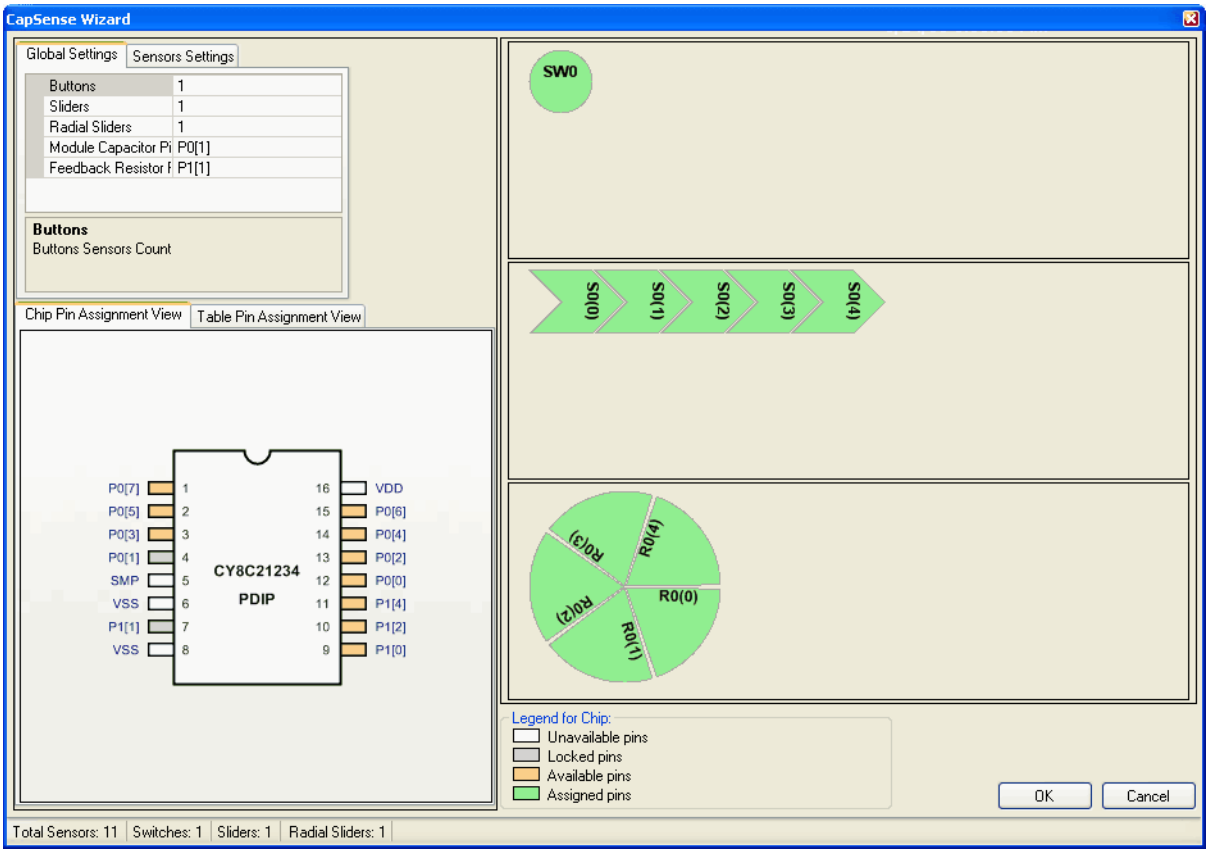
Avoid P1[0] and P1[1] when placing capacitive sensor connections. These pins are used for programming the part and may have excess routing capacitance affecting sensor sensitivity and noise.

Wizard Access

1. To access the Wizard, right-click any block of the CSDADC in the Device Editor Interconnect View, then select the CSDADC Wizard.



2. The Wizard opens showing the numeric entry boxes for the number of sensors and the number of slider sensors.



Wizard Pin Legend

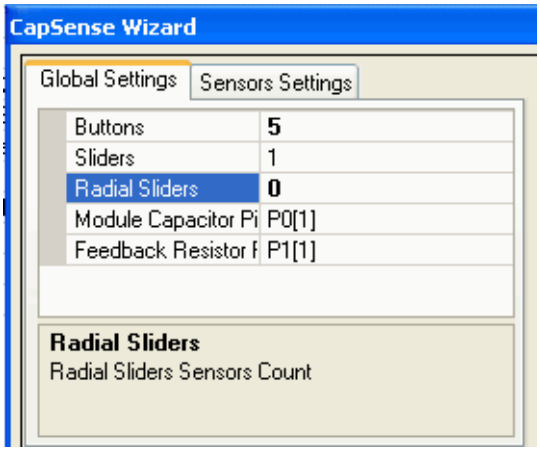
White – The pin cannot be used as a CapSense input.

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 has been 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.

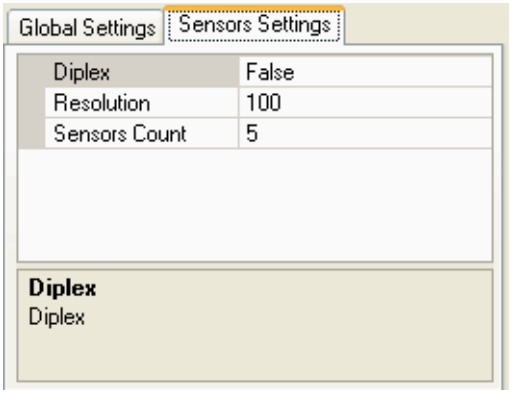
Orange – The pin is available for assignment.

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

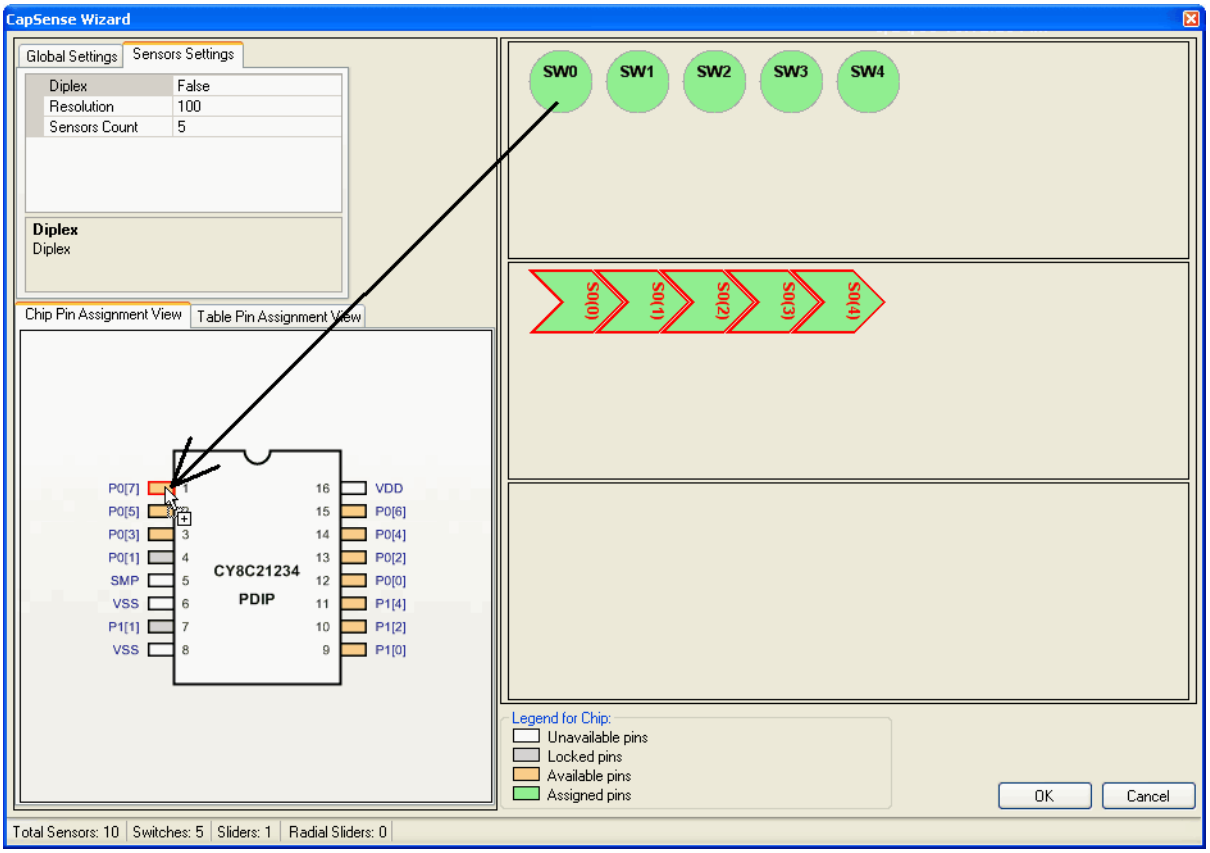
3. Type the number of independent buttons, sliders, and radial sliders. X-Y touch pads require two sliders but one is selected. The number of sensors is limited to the number of pins available.



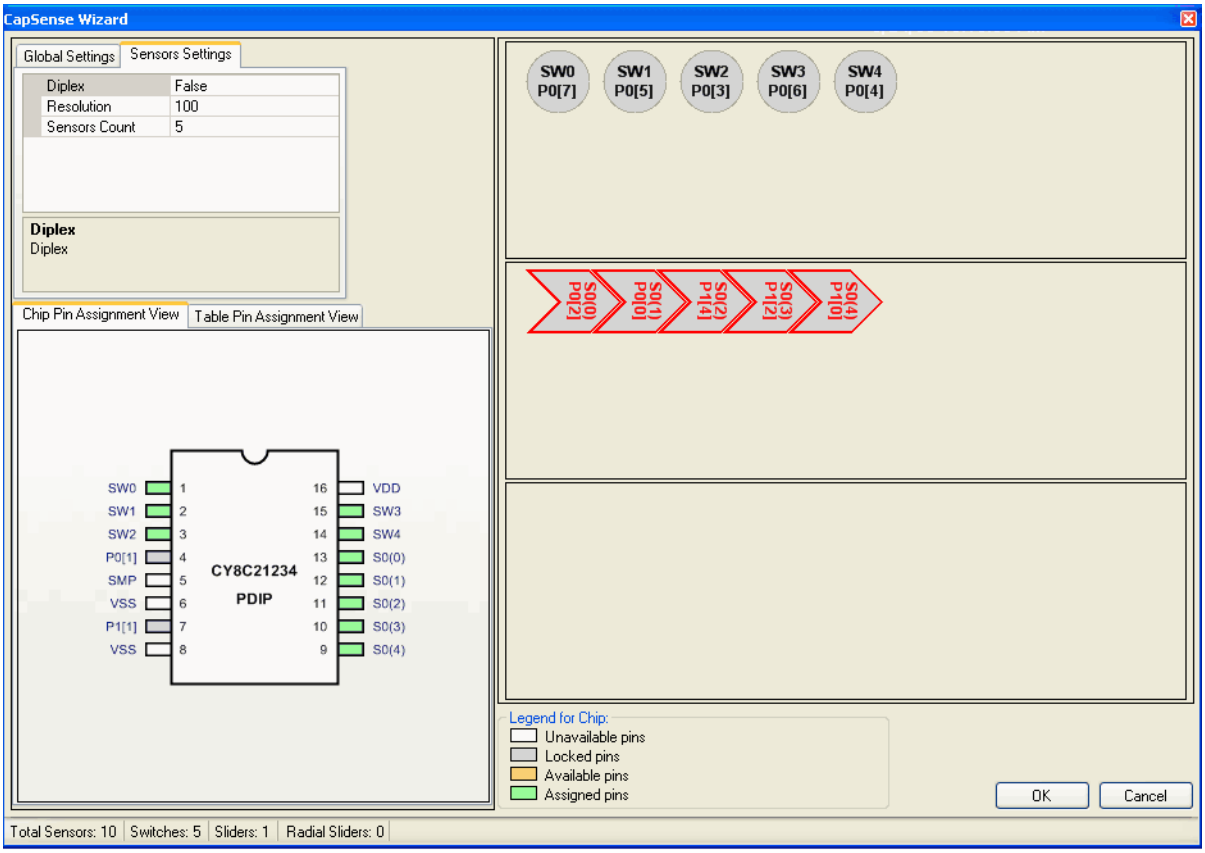
4. Select the Sensor Settings tab to specify settings for sliders and radial sliders. Type the number of sensor elements in each slider. The practical minimum number of sensors in a slider sensor is five, the maximum is limited by pin count.



5. Type the output resolution. The minimum value is 5. The maximum value is (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.
6. Select Diplex, if needed. This maps the number of pins selected for sensors to twice as many sensor locations on the board. Only the first half of the diplex sensors is shown; the second half is automatically mapped as outlined in the previous section on Diplexing. See the Diplexing section to find Diplexing tables for pin connections.
7. Click a button and drag it onto any available pin. The port pin is green after selection and is no longer available. Change sensor assignments by dragging the sensor off the port pin.



8. Repeat for the remainder of independent buttons.
9. Mapping of individual slider sensors to physical port pins is the same as for buttons.



10. Click **OK** to accept data and return to PSoC Designer.

Sensor placement is now complete. Right-click in the Device Editor window and select **Refresh** to update the pin connections.

Set user module parameters and generate application. You can adapt a sample project now, if you want.

When entering the numerical values in the CSDADC Wizard, please delete the old value first, then enter the new value. The cursor is not shown in the edit box.

If you want change pin assignment, place your cursor on the assigned pin, click the pin, and drag and drop it outside the switches box. The pin will are unassigned and you can then re-assign it.

After completing the Wizard, click Generate Application. Based on your entries for sensor count, pin assignment, diplexing, and resolution, a set of tables will be generated. The tables are located in CSDADC_Table.asm

Wizard Properties

Set the following properties in the Properties section of the Wizard.

Buttons

This is the number of physical buttons sensors.

Sliders

This is the number of physical linear sliders sensors.

Radial Sliders

This is the number of physical radial sliders sensors.

Modulator Capacitor Pin

This parameter sets the pin to connect the external modulator capacitor (Cmod). Choose from the available pins, P0[1] or P0[3]. The default value is P0[1]. You usually get better SNR with an external capacitor.

Feedback Resistor Pin

This parameter sets the pin to connect the external feedback resistor (Rb). Choose from the available pins: P1[1], P1[5], P3[1]. Some pins are not available on some device packages. Note that if some of these pins are used for other purposes (for example, allocated for sensor connection), they are not available for selection in the user module parameter list. Future versions of the CSD User Module may allow additional pins to be used for connecting the feedback resistor. This allows the use of a second I2C port on packages that have no P3 port. Use pins P1[5] or P3[1] to avoid programming problems.

Sensor Settings

Make the following settings in the Sensors Settings section of the Wizard.

Diplex

This option enables/disables diplex for slider. See the "Diplexing" section for a detailed description.

Resolution

This option sets the sensor resolution for slider. The minimum value is 5. The maximum value is limited to (number of pins used for sensors - 1) x 2^8 - 1 for non-diplexed sliders and radial sliders, or (2 x number of pins used for sensors - 1) x 2^8 - 1 for diplexed sliders.

Sensors Count

This is the number of physical sensors in slider or radial slider.

Pin Assignment

Assign switches or sensors to pins by dragging the switch or sensor to the pin in the Pin Assignment View. 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 is green after selection 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.

Clear All Pins

The "Clear All Pins" option is available by right-clicking on the wizard "Chip Pin Assignment View" and "Table Pin Assignment View". This option unassigns all chip pins.

Sensor Table

The Sensor Table 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). The table includes all independent sensors, then each sensor in order. An example for a table with six sensors is:

```
CSDADC_Sensor_Table:
_CSDADC_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
```

This table is used by CSDADC_wGetPortPin() routine.

Group Table

The Group Table defines each of the groups of button sensors or sliders. There is one entry for each slider plus one for the free button sensors. The first entry is always the free sensor. 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 multiplier that the slider's calculated centroid will be multiplied by to achieve the required resolution in the CSDADC wizard.

```
CSDADC_Group_Table:
_CSDADC_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

Diplex table scan order data is produced for a group when it is a slider and is also diplexed. 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:

```
DiplexTable_0:
; This group is not a diplexed slider
DiplexTable_1:
db0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5// 8 switch slider


 CSDADC_Diplex_Table:
_CSDADC_Diplex_Table:
db >DiplexTable_0, <DiplexTable_0
db >DiplexTable_1, <DiplexTable_1
```

Parameters and Resources

ADCEnabled

The ADCEnabled parameter can have two values:

- Enabled (default value)
- Disabled

When this parameter is set to Enabled, the ADC routines are included in the compilable code and can be called from user code. When this parameter is set to Disabled, the ADC routines are not included. This setting can be useful if the ADC is no longer required by your design. When Disabled, the CSDADC User Module behaves like the CSD User Module.

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’.

The finger detection threshold values apply to all sensors and sliders. For individual sensors (not contained in a slider group), these thresholds are variable and provided in the baBtnFThreshold[] array. The SetDefaultFingerThresholds() function may be used to set the thresholds to the default value set in the Device Editor. To adjust the sensitivity for individual sensors, change the baBtnFThreshold[] value for each sensor. (The size of this byte array is equal to the count of implemented individual sensors.)

Possible values range from 5 to 255. The default value is 40.

Die Temp Measurement

Enables or disables the existing ADC capability to measure the VTEMP voltage and convert the voltage into a corresponding temperature value. The parameter can be set to Enable only when the ADCEnabled parameter is set to Enabled. By default, this parameter is disabled.

Noise Threshold

For individual sensors, this parameter sets the count value above which the baseline is not updated. For slider sensors, it sets the count value below which the results are not counted in the calculation of the centroid. Possible values are 5 to 255. The default value is 20.

BaselineUpdate Threshold

When the new raw count value is above the current baseline and the difference is below the noise threshold (with the Sensors Autoreset parameter set to Disabled), the difference between the current baseline and the raw count is accumulated into what could be thought of as a bucket. When the bucket fills, the baseline is incremented by some value and the bucket is emptied. This parameter sets the threshold that the bucket must reach for the baseline to increment. Possible values are 0 to 255. Larger parameter values yield slower baseline update speeds. If you need more frequent baseline updates, decreases this parameter. The default value is 200.

LowBaselineReset

The LowBaselineReset parameter works together with the NegativeNoiseThreshold parameter. If the sample count values are below the baseline minus the NegativeNoiseThreshold for the specified number of samples, the baseline is set to the new raw count value. It essentially counts the number of abnormally low samples required to reset the baseline. It is generally used to correct for the finger-on-at-startup condition. Possible values are 0 to 255. The default value is 50.

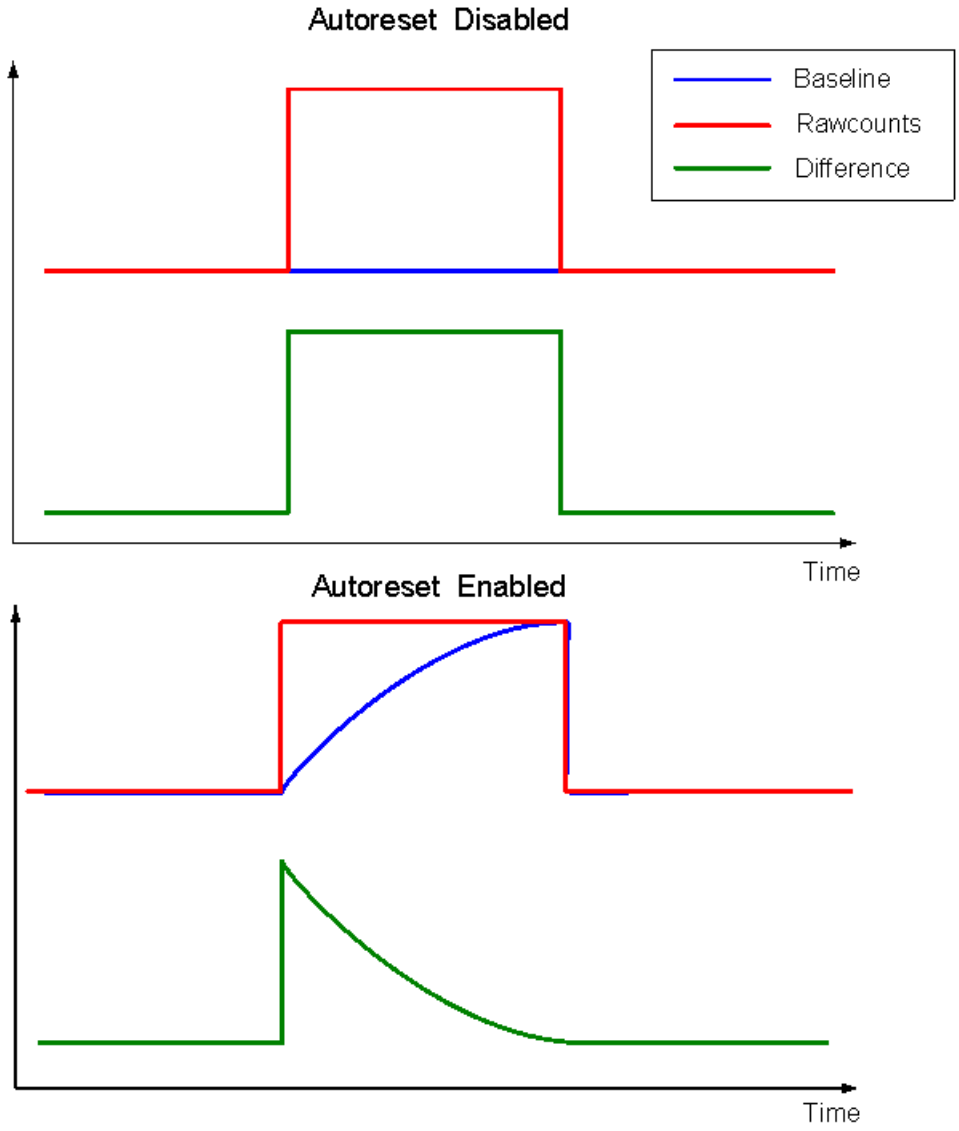
Sensors Autoreset

This parameter determines whether the baseline is updated at all times or only when the signal difference is below the Noise Threshold. When set to **Enabled** the baseline is updated constantly. This setting limits the maximum time duration of the sensor (typical values are 5 – 10 s), but it prevents the sensors from permanently turning on when the raw count suddenly rises without anything touching the sensor. This sudden rise can be caused by a large power supply voltage fluctuation, a high-energy RF noise source, or a very quick temperature change. This parameter is enabled by default.

When the parameter is set to **Disabled,** the baseline is updated only when raw count and baseline difference is below the Noise Threshold parameter. You should leave this parameter Enabled unless it is necessary to keep the sensors in the on state for a long time.

Figure 10 illustrates this parameter's influence on the baseline update.

Figure 10. Sensor Autoreset Parameter



Hysteresis

The Hysteresis parameter adds or subtracts from the finger threshold depending on whether the sensor is currently active or inactive. If the sensor is inactive, the difference count must overcome the finger threshold plus hysteresis. If the sensor is active, the difference count must go below the finger threshold minus hysteresis. It is used to add debouncing and stickiness to the finger detection algorithm. The threshold with hysteresis is evaluated when bIsSensorActive() or bIsAnySensorActive() is called. The sensor state can be monitored with the return value of bIsSensorActive() or the baSnsOnMask[] array. Possible values are 0 to 255, but must be lower than the Finger Threshold parameter setting. The default value is 10.

The proper selection of high-level decision logic parameters allows you to effectively compensate for environmental factors (temperature, humidity changes, and so on), suppress noisy signals (ESD and power supply spikes), and provide reliable touch detection under various conditions.

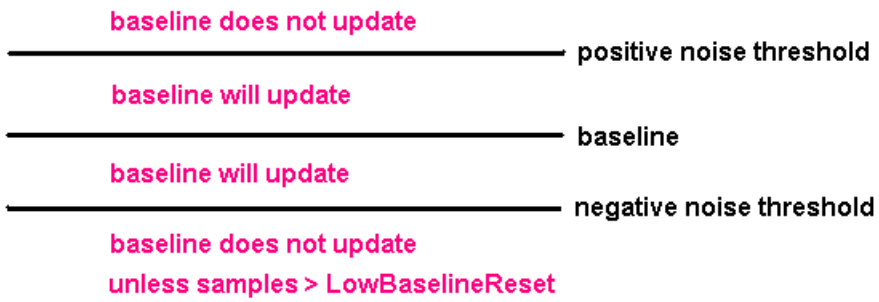
Debounce

The Debounce parameter adds a debounce counter to the sensor active transition. 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. The debounce counter is incremented by the bIsSensorActive or bIsAnySensorActive API functions.

Possible values are 1 to 255. A setting of 1 provides no debouncing. The default value is 3.

NegativeNoiseThreshold

The NegativeNoiseThreshold parameter adds a negative difference count threshold. If the current raw count is below the baseline and the difference between them is greater than this threshold, the baseline will not be updated. However, if the current raw count stays in the low state (difference greater than threshold) for the number of samples specified by the LowBaselineReset parameter, the baseline will be reset. Possible values are 0 to 255. The default value is 20.



LowBaselineReset

The LowBaselineReset parameter works together with the NegativeNoiseThreshold parameter. If the sample count values are below the baseline minus the NegativeNoiseThreshold for the specified number of samples, the baseline is set to the new raw count value. It essentially counts the number of abnormally low samples required to reset the baseline. It is generally used to correct for the finger-on-at-startup condition. Possible values are 0 to 255. The default value is 50.

Scanning Speed

This parameter affects the sensors' scanning speed. The available selections are: **Ultra Fast, Fast, Normal, and Slow**. The default setting is Normal. Slower scanning speeds provide the following advantages:

- Improved SNR
- Better immunity to power supply and temperature changes
- Less demand for system interrupt latency; you can handle longer interrupts.

See the warnings section for more about interrupt latency.

Resolution

This parameter determines the scanning resolution in bits. The sensors can be scanned with resolutions ranging from 9 to 16 bits. The default resolution is 12 bits.The maximum raw count for scanning resolution for N bits is 2^N-1.

Increasing the resolution improves sensitivity and the SNR of touch detection. Use a high resolution for proximity detection. A 16-bit resolution, slow scanning mode, and a 20 cm wire allows you to detect a hand at 20 cm or more.

The scanning speed and resolution affect the VC1, VC2, VC3, and ADCPWM dividers in the following ways:

Scanning Speed	VC1
Ultra fast	1
Fast	2
Normal	4
Slow	8

Resolution, bits	VC2	VC3	ADCPWM
9	8	16	4
10	8	32	4
11	8	64	4
12	8	128	4
13	8	256	4
14	8	256	8
15	8	256	16
16	8	256	32

The VC1 divider depends on the scanning speed only. The VC2, VC3, and ADCPWM depend on the resolution only.

Table 10. Scanning Time in µs vs. Scanning Speed and Resolution for 24 MHz IMO Operation, PRS16 Configuration

Resolution, bits	Scanning speed			
	Ultra Fast	Fast	Normal	Slow
9	75	110	170	300
10	110	170	300	510
11	170	300	510	1010
12	300	510	1010	2030

Resolution, bits	Scanning speed			
	Ultra Fast	Fast	Normal	Slow
13	510	1010	2030	4060
14	850	1690	3380	6760
15	1520	3040	6080	12200
16	2880	5720	11500	23200

Table 11. Scanning Time in µs vs. Scanning Speed and Resolution for 24 MHz IMO Operation, PRS8 with Prescaler Configuration

Resolution, bits	Scanning speed			
	Ultra Fast	Fast	Normal	Slow
9	60	85	150	255
10	85	150	255	510
11	150	255	510	1020
12	255	510	1020	2040
13	510	1020	2040	4080
14	845	1700	3380	6760
15	1530	3060	6120	12100
16	2880	5800	11500	23000

Note The scanning time is measured as the time interval between two sensor scans. This time includes the sensor setup time, modulator stabilization delay, sample conversion interval, and data preprocessing time.

Modulator Capacitor Pin

This parameter sets the pin to connect the external modulator capacitor (C_{mod}). Choose from the available pins P0[1], P0[3]. The default pin is P0[1].

Feedback Resistor Pin

This parameter sets the pin to connect the external feedback resistor (R_b). Choose from the available pins: P1[1], P1[5], P3[1]. Some pins are not available on some device packages. Tip: if some of these pins are used for other purposes (for example, allocated for sensor connection), they are not available for selection in user module parameter list. Future versions of the CSDADC User Module may allow additional pins to be used for connecting the feedback resistor. This will allow the use of a second I^2C port on packages that have no P3 port. Use pins P1[5] or P3[1] to avoid programming problems. The default pin is P1[1].

Reference

This parameter sets the source of the comparator reference. See the following Ref Value parameter description for more information. By default the reference comes from the bandgap (VBG).

Ref Value

This parameter sets the comparator reference value, when the comparator reference comes from an analog modulator (ASE11) or an externally filtered PWM/PRSPWM signal (from AnalogColumn_InputSelect_1 with RC filter). This value has no effect when the reference comes from the bandgap (VBG) or from the external voltage divider (from AnalogColumn_InputSelect_1 with resistive voltage divider). The default RefValue is 0.

The ‘0’ corresponds to the minimum reference (1/4 Vdd). The ‘8’ corresponds to the maximum value (3/4Vdd). When reference increases the sensitivity is decreased, but influence on the shielding electrode is increased.

If the design has sensors with noticeable capacitance differences (for example, sensors with different sized squares), you can balance raw counts by setting a higher reference for the sensors with larger capacitance using an API function.

Note This parameter is not available in CSDADC with VC2 clock source configuration.

Prescaler Period

This parameter sets the prescaler period register and determines the precharge switch output frequency. This parameter is available for configuration with prescaler only. The prescaler period values can range from 1 to 255. The default Prescaler Period is 7.

The recommended values are 2^n-1 to obtain the maximum signal to noise ratio (SNR):

- 1
- 3
- 7
- 15
- 31
- 63
- 127
- 255

Other values can result in more noise, especially at low resolution and high scan speed.

ShieldElectrodeOut

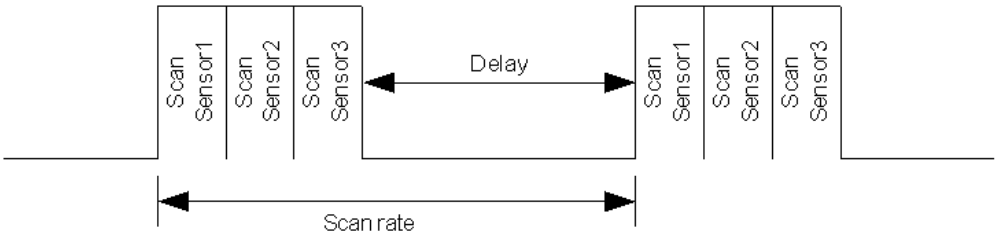
The shielding electrode signal source can be selected from one of the free digital row buses (Row_0_Output_0 - Row_0_Output_3). Each row output can be routed to one of three pins. Set the Row LUT Function to A. The default setting is None.

Note This parameter is available only in CSDADC with PRS8/PRS16 clock source configurations. In CSDADC with PWM8 clock source configuration, the shielding electrode signal is permanently connected to Row_0_Output_0.

Sensor Scan Rate Selection Guidelines

Scan rate is the rate at which sensors are scanned. An example of a 3-button design is shown in the following figure. All sensors in the design are scanned sequentially and there is a delay before the next sensor scan is initiated.

Figure 11. Typical Sensor Scan



To ensure proper working of the baseline, it is recommended to maintain a scan rate of 15 ms or more in a design. This indicates that a design with less number of sensors must add a delay to make the sensor scan rate equal to or greater than 15 ms. A design with more number of sensors may not need any delay as scanning all sensors itself may consume 15 ms. A good design may put the CapSense controller in sleep mode, instead of the firmware delay routine, to create a low power design.

Application Programming Interface

The Application Programming Interface (API) functions are provided as part of the user module to allow 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 CSDADC_1 to the first instance of this user module in a given 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 CSDADC 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 will 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 CSDADC, start it sampling, and stop the CSDADC. In all cases the instance name of the module replaces the CSDADC prefix shown in the following entry points. Failure to use the correct instance name is a common cause of syntax errors.

Most of CSDADC API functions were inherited from the CSD User Module and ADC10 API. If you have existing code from a design that used both user modules, the code changes are minimal. Most of the functions have the same function names as the two separate user modules. As a side effect to leaving the function names the same, there are some functions with very similar names that perform different

functions in the CapSense and ADC modules. Read the function descriptions carefully to avoid coding mistakes. Several new API routines were added to support ADC functionality.

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 several global arrays:

- CSDADC_waSnsBaseline[]
- CSDADC_waSnsResult[]
- CSDADC_waSnsDiff[]
- CSDADC_baSnsOnMask[]

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

- CSDADC_UpdateAllBaselines();
- CSDADC_UpdateSensorBaseline();
- CSDADC_InitializeBaselines().

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

- CSDADC_ScanSensor();
- CSDADC_ScanAllSensors().

CSDADC_waSnsDiff [] – This is an integer array that contains the difference between the raw data and the baseline data of each capacitive sensor. The array size is equal to the sensor count.

CSDADC_baSnsOnMask[] – This is a byte array that holds the capacitive sensor on or off state (for buttons or sliders). CSDADC_baSnsOnMask[0] contains the masked bits for sensors 0 through 7 (sensor 0 is bit 0, sensor 1 is bit 1). CSDADC_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 CSDADC_baSnsOnMask[] data is updated by CSDADC_bIsSensorActive(BYTE bSensor) function or CSDADC_bIsAnySensorActive() routines.

CSDADC_Start

Description:

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

C Prototype:

```
void  CSDADC_Start()
```

Assembly:

```
lcall  CSDADC_Start
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_Stop

Description:

Stops the sensor scanner, disables internal interrupts, and calls CSDADC_ClearSensors() to reset all sensors to an inactive state.

C Prototype:

```
void  CSDADC_Stop()
```

Assembly:

```
lcall  CSDADC_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

**

CSDADC_Resume

Description:

Resumes the user module operation after CSDADC_Stop call.

C Prototype:

```
void  CSDADC_Resume()
```

Assembly:

```
lcall  CSDADC_Resume
```

Parameters:

None

Return Value:

None

Side Effects:

**

CSDADC_EnableADC

Description:

Activates the ADC function of the user module. Once CSDADC_ABSOLUTE mode is selected, the single slope ADC mode is activated and ADC needs to be calibrated. See the CSDADC_wCal()

routine for details. If CSDADC_RATIOMETRIC is selected then CSDADC does not require calibration.

All ADC-related APIs should be called only after CSDADC_EnableADC called.

This function is available only if ADC is enabled in user module parameters.

Assembly:

```
mov A, Mode
lcall CSDADC_EnableADC
```

C Prototype:

```
void  CSDADC_EnableADC (BYTE Mode);
```

Parameters:

Mode – determines the ADC operation mode. Two possible options are supported for this parameter:

CSDADC_ABSOLUTE – configures the ADC in the absolute voltage mode;

CSDADC_RATIOMETRIC – configures the ADC in the ratiometric measurement mode.

Return Value:

None

Side Effects:

**

CSDADC_EnableCapsense

Description:

This routine enables the CapSense operation, setting the hardware configuration back to CapSense functions.

This function is available only if ADC is enabled in user module parameters.

Assembly:

```
lcall CSDADC_EnableCapsense
```

C Prototype:

```
void  CSDADC_EnableCapsense(void);
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_EnableInput

Description:

Connects the input port to the ADC. The connection is done using the analog bus, so all available ports can serve ADC input functions. Before switching to the CSD mode, it is important to disconnect

the pin from analog bus using CSDADC_DisableInput(). In the same way, disconnect the pin from the analog bus before setting other pin for ADC using the CSDADC_EnableInput().

This function is available only if ADC is enabled in user module parameters.

Assembly:

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

C prototype:

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

Parameters:

bPort - sets the ADC input port

bMask - port bit.

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_DisableInput

Description:

Disconnects the input port from ADC by disconnecting a selected port pin from the analog bus.

This function is available only if ADC is enabled in user module parameters.

Assembly:

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

C prototype:

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

Parameters:

bPort - ADC input port

bMask - port bit.

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_StartADC

Description:

Turns on the ADC and runs it continuously.

This function is available only if ADC is enabled in user module parameters.

Assembly:

```
lcall CSDADC_StartADC
```

C Prototype:

```
void CSDADC_StartADC(void)
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_StopADC

Description:

Turns off the ADC and stops the conversion process immediately.

This function is available only if ADC is enabled in user module parameters.

Assembly:

```
lcall CSDADC_StopADC
```

C Prototype:

```
void CSDADC_StopADC(void)
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_fIsDataAvailable

Description:

Checks the status of the ADC.

This function is available only if ADC is enabled in user module parameters.

Assembly:

```
lcall CSDADC_fIsDataAvailable
; Return value will be in A
```

C Prototype:

```
BYTE CSDADC_fIsDataAvailable(void)
```

Parameters:

None

Return Value:

Returns a non-zero value if data has been converted and is ready to read.

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_wGetData

Description:

Returns converted data. CSDADC_fIsDataAvailable() should be called before verifying that the data sample is ready.

This function is available only if ADC is enabled in user module parameters.

Assembly:

```
lcall CSDADC_wGetData
;Data will be in A (LSB) and X(MSB) upon return
```

C Prototype:

```
WORD CSDADC_wGetData(void)
```

Parameters:

None

Return Value:

Returns the converted data sample in unsigned integer format through A and X registers.

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_wGetDataClearFlag

Description:

Gets data and resets the data-available flag.

This function is available only if ADC is enabled in user module parameters.

Assembly:

```
lcall CSDADC_wGetDataClearFlag
;Data will be in A (LSB) and X(MSB) upon return
```

C Prototype:

```
WORD CSDADC_wGetDataClearFlag(void)
```

Parameters:

None

Return Value:

Returns the converted data sample in unsigned integer format through A and X registers.

Side Effect:

See Note ** at the beginning of the API section.

CSDADC_wCal

Description:

Calibrates the ADC by trimming the Capacitor in the SC block to, in turn, vary the slope of the reference ramp used in the conversion. The ADC input must be set to a user selected fixed reference voltage to which to calibrate. This function is called after calling CSDADC_Start and before calling CSDADC_StartADC.

This function is available only if ADC is enabled in user module parameters.

If you use the CSDADC_ABSOLUTE mode, make sure to calibrate the ADC. This can be an internal or external reference. For example, you can calibrate the ADC input to Vbg = 1.3 V. It is calculated in this example when the device has a supply voltage of 5 V:The Full Scale count at a Resolution of 12 is 4095. You get this count when 5 V is applied to the ADC input. In this case we should calculate the Counts that correspond to Vbg = 1.3 V. Use following formula:

wVal = 4095*1.3/5 = 1064 or 0x428

The example code to calibrate the ADC is as follows:// Calibrate the user module using the 1.3 V internal BandGap reference sourceACE01CR1 &= ~0x04; // Connect Vref to PMux input of ACE01 Analog BlockCSDADC_wCal(0x428); // Calibrate ADCACE01CR1 |= 0x04; // Restore PMux connection to Analog MUX Bus

C Prototype:

```
int   CSDADC_wCal(WORD wVal)
```

Assembly:

```
mov   A, [wVal+1]
mov   X, [wVal]
lcall  CSDADC_wCal
;Data will be in A (LSB) and X(MSB) upon return
```

Parameters:

wVal: This number is the expected value given the reference voltage to which the input is set. The CSDADC_wCal() routine trims the capacitor in the SC block until the result is equal to or as close to as the wVal as possible.

Return Value:

Returns an unsigned integer indicating the nearest result to the wVal that was possible.

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_StartTempMeasurement

Note This API is available only when the "Die Temp Measurement" parameter is set to "Enabled".

Description:

Connects the VTemp sensor output to the PMux input. Then starts the ADC measurement. For an example of the API usage, see the "Die Temperature Measurement" section.

C Prototype:

```
void CSDADC_StartTempMeasurement (void)
```

Assembly:

```
lcall  CSDADC_StartTempMeasurement
```

Parameters:

None

Return Value:

None

Side Effects

See Note ** at the beginning of the API section.

CSDADC_GetTemperature

Note This API is available only when the "Die Temp Measurement" parameter is set to "Enabled".

Description:

Stops the ADC measurement, calculates Die Temperature value, restores the ADC input and resets the data-available flag. The CSDADC_fIsDataAvailable API must be called before the API to verify that the data sample is ready. For an example of the API usage, see the "Die Temperature Measurement" section.

The Die Temperature is calculated using the following equation:

DieTemp = M * wADCCount * wCalibrationVoltage / wCalibrationCount - B

In this equation:

wADCCount - result of the ADC VTemp voltage measurement;

wCalibrationVoltage - the API parameter;

wCalibrationCount - internally stored value of the wVal parameter of the CSDADC_wCal(WORD wVal) API function;

M, B - coefficients that are stored in the flash.

C Prototype:

```
CHAR CSDADC_GetTemperature(WORD wCalibrationVoltage)
```

Assembly:

```
mov A, [wCalibrationVoltage+1] ; LSB of API parameter
mov X, [wCalibrationVoltage] ; MSB of API parameter
lcall  CSDADC_GetTemperature
```

Parameters:

WORD wCalibrationVoltage: Calibration voltage (measured in mV) used for the ADC calibration. This voltage corresponds to the wVal parameter (counts) of the CSDADC_wCal API function.

Return Value:

‘A’ contains the Die Temperature in Celsius degrees.

Side Effects

See Note ** at the beginning of the API section.

CSDADC_ScanSensor

Description:

Scans the selected capacitive sensor. Each sensor has a unique number within the sensor array. This number is assigned by the CSDADC Wizard in sequence. Sw0 is sensor 0, Sw1 is sensor 1, and so on.

C Prototype:

```
void CSDADC_ScanSensor(BYTE bSensor);
```

Assembly:

```
mov A,  bSensor
lcall CSDADC_ScanSensor
```

Parameters:

Register A contains the Sensor Number before call

Return Value:

None

Side Effects

See Note ** at the beginning of the API section.

CSDADC_ScanAllSensors

Description:

Scans all of the configured capacitive sensors by calling CSDADC_ScanSensor() for each sensor index.

C Prototype:

```
void CSDADC_ScanAllSensors();
```

Assembly:

```
lcall CSDADC_ScanAllSensors
```

Parameters:

None

Return Value:

None

Side Effects

See Note ** at the beginning of the API section.

CSDADC_UpdateSensorBaseline

Description:

The historical count value, calculated independently for each capacitive sensor, is called the sensor's baseline. This baseline is updated using the Bucket Method.

The Bucket Method uses the following algorithm.

1.Each time CSDADC_UpdateSensorBaseline() is called, a difference count is calculated by subtracting the previous baseline from the raw count value. This difference is stored in the CSDADC_waSnsDiff[] array and is provided to you.

2.If Sensors Autoreset is disabled, each time CSDADC_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.

3.Once the accumulated difference counts in the virtual bucket has reached the BaselineUpdateThreshold, the baseline is incremented by one and the bucket is reset to 0.

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

C Prototype:

```
void CSDADC_UpdateSensorBaseline(BYTE bSensor)
```

Assembly:

```
mov   A,   bSensor
lcall  CSDADC_UpdateSensorBaseline
```

Parameter:

A => Sensor Number

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_UpdateAllBaselines

Description:

Uses the CSDADC_bUpdateSensorBaseline() function to update the baselines for all capacitive sensors

C Prototype:

```
void  CSDADC_UpdateAllBaselines()
```

Assembly:

```
lcall CSDADC_UpdateAllBaselines
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_bIsSensorActive

Description:

Checks the difference count array for the given sensor compared to its finger threshold. Hysteresis 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 CSDADC_baSnsOnMask[] array.

C Prototype:

```
BYTE  CSDADC_bIsSensorActive(BYTE bSensor)
```

Assembly:

```
mov  A,  bSensor
lcall CSDADC_bIsSensorActive
```

Parameters:

bSensor A => Sensor Number

Return Value:

Return value of '1' if active, '0' if not active

A => '1' – Selected sensor is active, '0' – Selected sensor is not active.

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_bIsAnySensorActive

Description:

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

C Prototype:

```
BYTE  CSDADC_bIsAnySensorActive()
```

Assembly:

```
lcall CSDADC_bIsAnySensorActive
```

Parameters:

None

Return Value:

Return value of 1 if active, 0 if not active

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

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_wGetCentroidPos

Description:

Checks a difference array for a centroid. If one exists, the offset and length are stored in temporary variables and the centroid position is calculated to the resolution specified in the CSDADC Wizard.

This function is available only if slider is defined by the CSDADC Wizard.

C Prototype:

```
WORD  CSDADC_wGetCentroidPos(BYTE bSnsGroup)
```

Assembly:

```
mov  A,  bSnsGroup
lcall CSDADC_wGetCentroidPos
```

Parameters:

bSnsGroup A => Group Number

This parameter is a reference to a specific group of sensors used as a slider. Group 0 is for 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. The routine should be called only 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 CSDADC Wizard. If no sensors are active, the function returns –1 (FFFFh). If an error occurs during execution of the centroid/diplexing algorithm, the function returns –1 (FFFFh). You can use the CSDADC_blsSensorActive() routine to determine which slider segments are touched, if required.

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.

CSDADC_InitializeSensorBaseline

Description:

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

C Prototype:

```
void CSDADC_InitializeSensorBaseline(BYTE bSensor)
```

Assembly:

```
mov A, bSensor
lcall CSDADC_InitializeSensorBaseline
```

Parameters:

A => Sensor Number

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_InitializeBaselines

Description:

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

C Prototype:

```
void CSDADC_InitializeBaselines()
```

Assembly:

```
lcall CSDADC_InitializeBaselines
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_SetDefaultFingerThresholds

Description:

Loads the CSDADC_baBtnFThreshold[] array with the FingerThreshold parameter value. This function must be called before scanning if the CSDADC_baBtnFThreshold[] array is not manually loaded with custom values.

C Prototype:

```
void CSDADC_SetDefaultFingerThresholds()
```

Assembly:

```
lcall CSDADC_SetDefaultFingerThresholds
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_SetScanMode

Description:

Sets scanning speed and resolution for capacitive sensors and scanning speed and resolution for incremental ADC. This function can be called at runtime to change the scanning speed and resolution. The function overwrites the user module parameter settings. This function is effective when some sensors need to be scanned with different scanning speed and resolution, for example, regular buttons and a proximity detector. The regular buttons can be scanned with 9-bit resolution and 300 µs scan time. The proximity detector can be scanned less often with 16-bit resolution and scanning time of more than 12 ms for long-range detection. This function can be used in conjunction with CSDADC_ScanSensor() function.

C Prototype:

```
void CSDADC_SetScanMode(BYTE bSpeed, BYTE bResolution);
```

Assembly:

```
mov     A, bSpeed
mov     X, bResolution
```

```
lcall    CSDADC_SetScanMode
```

Parameters:

bSpeed: Scanning Speed

The following constants are given for the bSpeed parameter:

Constant	Value
CSDADC_ULTRA_FAST_SPEED	0x00
CSDADC_FAST_SPEED	0x01
CSDADC_NORMAL_SPEED	0x02
CSDADC_SLOW_SPEED	0x03

bResolution: Scanning Resolution. Set this value to the required number of bits of resolution. This parameter value must not be lower than 9 or greater than 16.

The following possible constants are given for the bResolution parameter:

Constant	Value
CSDADC_9_BIT_RESOLUTION	9
CSDADC_10_BIT_RESOLUTION	10
CSDADC_11_BIT_RESOLUTION	11
CSDADC_12_BIT_RESOLUTION	12
CSDADC_13_BIT_RESOLUTION	13
CSDADC_14_BIT_RESOLUTION	14
CSDADC_15_BIT_RESOLUTION	15
CSDADC_16_BIT_RESOLUTION	16

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_SetRefValue

Description:

Sets capacitive sensors scanning reference value. Valid only when the reference is supplied from the analog modulator (ASE11 in the Reference parameter) or from externally filtered PWM/PRS signals. Accepted values are 0..8. Value 0 corresponds to the minimum reference voltage that provides the maximum sensitivity. The value 8 sets the maximum reference voltage and results in lower sensitivity. This function can be used in conjunction with CSDADC_ScanSensor().

This function is not available in CSDADC with VC2 clock source configuration.

C Prototype:

```
void CSDADC_SetRefValue(BYTE bRefValue);
```

Assembly:

```
mov     A, bRefValue
lcall    CSDADC_SetRefValue
```

Parameters:

bRefValue - Sets the scanning reference vale. Accepted values are 0..8.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_ClearSensors

Description:

Clears all capacitive sensors to the non-sampling state by sequentially calling CSDADC_wGetPortPin() and CSDADC_DisableSensor() for each of the sensors.

C Prototype:

```
void  CSDADC_ClearSensors()
```

Assembly:

```
lcall  CSDADC_ClearSensors
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_wReadSensor

Description:

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

C Prototype:

```
WORD  CSDADC_wReadSensor(BYTE bSensor)
```

Assembly:

```
mov A, bSensor
lcall CSDADC_wReadSensor
```

Parameters:

A => Sensor Number

Return Value:

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

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_wGetPortPin

Description:

Returns the port number and pin mask for a given sensor. The passed parameter indexes and selects the data from the CSDADC_Sensor_Table[]. The return value can be passed to the CSDADC_EnableSensor(), CSDADC_DisableSensor().

C Prototype:

```
WORD  CSDADC_wGetPortPin(BYTE bSensorNum)
```

Assembly:

```
mov  A,  bSensorNumber
lcall     CSDADC_wGetPortPin
;Data will be in A (LSB, Sensor Bitmap) and X(MSB, Port Number) upon return
```

Parameters:

bSensorNumber – The range is 0 to (n – 1) where n is the total of the number of sensors set in the CSDADC Wizard plus the number of sensors included in sliders. The sensor number is used by CSDADC_wGetPortPin() to determine port and bit mask for the selected active sensor.

Return Value:

A => Sensor Bitmap

X => Port Number

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_EnableSensor

Description:

Configures the selected sensor to measure during the next measurement cycle. The port and sensor can be selected using the CSDADC_wGetPortPin() function, with the port number and sensor bitmask loaded into X and A, respectively. Drive modes are modified to place the selected port and pin into Analog High-Z mode and to enable the correct Analog Mux Bus input. This also enables the comparator function.

C Prototype:

```
void  CSDADC_EnableSensor(BYTE bMask, BYTE bPort)
```

Assembly:

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

Parameters:

A => Sensor Bitmap

X => Port Number

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

CSDADC_DisableSensor

Description:

Disables the sensor selected by the CSDADC_wGetPortPin() function. The drive mode is changed to Strong (001). This effectively grounds the sensor. The connection from the port pin to the Analog-MuxBus is turned off. The function parameters are returned by CSDADC_wGetPortPin() function.

C Prototype:

```
void  CSDADC_DisableSensor(BYTE bMask, BYTE bPort)
```

Assembly:

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

Parameters:

A => Sensor Bitmap

X => Port Number

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

Die Temperature Measurement

The Die Temperature can be measured only when the "Die Temp Measurement" parameter is set to "Enabled".

Sample Code:

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

void main(void)
{
    CHAR cTemp;
    WORD wVoltage;

    M8C_EnableGInt;
    CSDADC_Start(); // Start the User Module
    CSDADC_EnableADC (CSDADC_ABSOLUTE);

// Calibrate the user module using the 1.3 V internal BandGap reference source
ACE01CR1 &= ~0x04;           // Connect Vref to PMux input of ACE01 Analog Block
```

```
    CSDADC_wCal(0x428);        // Calibrate ADC
    ACE01CR1 |= 0x04;               // Restore PMux connection to Analog MUX Bus

    while (1)
    {
        CSDADC_EnableInput(0x01, 0x00); // use P0[0]
        CSDADC_StartADC();
        while (!CSDADC_fIsDataAvailable());
        wVoltage = CSDADC_wGetDataClearFlag();
        CSDADC_StopADC();
        CSDADC_DisableInput(0x01, 0x00); // required for normal CSD operation

CSDADC_StartTempMeasurement();
        while (!CSDADC_fIsDataAvailable());
        cTemp = CSDADC_GetTemperature(1300);

/* Add user code here to display the results of
        voltage and temperature measurement */
    }
}
```

Note

1. The ADC must be calibrated before calling the CSDADC_StartTempMeasurement API.
2. The CSDADC_StartTempMeasurement API calls the CSDADC_StartADC API; as a result, it is not necessary to call the API before starting temperature measurement.
3. The CSDADC_GetTemperature API stops the ADC converting. In case you need to measure a voltage after the Die Temperature measurement, the CSDADC_StartADC API must be called.

Sample Firmware Source Code

This code starts the user module and continuously scans the sensors.

It is easy to interface with different sensor types supporting both ratiometric and absolute input ADC modes and possibility modes switching in run time.

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

//#define ACD_KIND  CSDADC_RATIOMETRIC
#define ACD_KIND  CSDADC_ABSOLUTE

WORD wCal;
WORD wResult;

void main(void)
{
  M8C_EnableGInt;
  CSDADC_Start();
  CSDADC_SetDefaultFingerThresholds();
CSDADC_InitializeBaselines();

#if (ACD_KIND==CSDADC_ABSOLUTE)
  CSDADC_EnableADC(ACD_KIND);
  wCal = CSDADC_wCal(1000);
```

```
#endif

  while (1) {
    CSDADC_EnableCapsense();
    CSDADC_ScanAllSensors();
    CSDADC_UpdateAllBaselines();

CSDADC_EnableADC(ACD_KIND);
    CSDADC_EnableInput(0x01, 0x02); // use P2[0]

CSDADC_StartADC();

while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();

CSDADC_StopADC();

CSDADC_DisableInput(0x01, 0x02); // required for normal CSD operation
  }
}
```

The same project is Assembly is:

```
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

; ACD_KIND: EQU  CSDADC_RATIOMETRIC
  ACD_KIND: EQU  CSDADC_ABSOLUTE

_main:

    M8C_EnableGInt

lcall  CSDADC_Start
lcall  CSDADC_SetDefaultFingerThresholds
    lcall  CSDADC_InitializeBaselines

IF (ACD_KIND & CSDADC_ABSOLUTE)
    mov    A, CSDADC_ABSOLUTE
    lcall  CSDADC_EnableADC

mov    A, <1000
    mov    X, >1000
    lcall  CSDADC_wCal
    ;calibration data are located in A (LSB) and X(MSB)
ENDIF

loop:
    lcall  CSDADC_EnableCapsense
    lcall  CSDADC_ScanAllSensors
    lcall  CSDADC_UpdateAllBaselines
```

```
mov    A, ACD_KIND
    lcall  CSDADC_EnableADC

mov    A, 0x01
    mov    X, 0x02
lcall  CSDADC_EnableInput  ; use P2[0]

lcall  CSDADC_StartADC

.scan:
lcall  CSDADC_fIsDataAvailable
    cmp    A, 0
    jz     .scan
    lcall  CSDADC_wGetDataClearFlag

; The ADC rezult is stored in A (LSB) and X(MSB)

lcall  CSDADC_StopADC

    mov    A, 0x01
    mov    X, 0x02
    lcall  CSDADC_DisableInput  ; required for normal CSD operation

    jmp    loop
```

Configuration Registers

CSDADC with PRS16 clock source Configuration Registers

Table 12. Block CMP, Register: ACE_CONTROL1 (ACE01CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	1	Reference			1	1	1

Reference is maintained by the Reference parameter.

Table 13. Block CMP, Register: ACE_CONTROL2 (ACE01CR2), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	Power	

Power is maintained by CSDADC_Start() and CSDADC_Stop() APIs.

Table 14. Block CMP_REF, Register: ASE_CONTROL (ASE11CR0), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 15. Block CMP_REF, Register: ADC_CONTROL (ADC1_CR), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CMPST	1	1	0	0	Auto	0	ADCEN

The CMPST is read-only and is used by ADC-related API. Auto bit is maintained over ADC-related API. ADCEN enables ADC operations and is maintained by the CSDADC API.

Table 16. Block CMP_REF, Register: ADC_TRIM (ADC1_TR), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	ADCTrimValue							

ADCTrimValue is over CSDADC API control. Contain ADC trim value for Single Slope ADC and zero value for CSD operation.

Table 17. Block CNT, Register: Function (DxBxxFN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	1

Table 18. Block CNT, Register: Input (DxBxxIN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 19. Block CNT, Register: Output (DxBxxOU), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	1	0	0	0	0	0	0

Table 20. Block CNT, Register: Control (DxBxxCR0), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

The Enable bit enables Counter operations, and is maintained by CSDADC API.

Table 21. Block CNT, Register: Period (DxBxxDR0), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CounterValue no direct readable or writable							

Table 22. Block CNT, Register: Period (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	1	1	1

Table 23. Block CNT, Register: Compare (DxBxxDR2), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 24. Block PRS, Register: Function (DxBxxFN), Bank 1

Bit	7	6	5	4	3	2	1	0
LSB	0	0	0	0	1	0	1	0
MSB	0	1	1	0	1	0	1	0

Table 25. Block PRS, Register: Input (DxBxxIN), Bank 1

Bit	7	6	5	4	3	2	1	0
LSB	0	0	0	0	0	0	0	0
MSB	0	0	1	1	0	0	0	0

Table 26. Block PRS, Register: Output (DxBxxOU), Bank 1

Bit	7	6	5	4	3	2	1	0
LSB	1	1	0	0	0	0	0	0
MSB	1	1	1	0	0	ShieldElectrodeOut		

The ShieldElectrodeOut enables output signal for shield electrode to Row Output. It is controlled by user module parameter with the same name.

Table 27. Block PRS, Register: Control (DxBxxCR0), Bank 1

Bit	7	6	5	4	3	2	1	0
LSB	0	0	0	0	0	0	0	Enable
MSB	0	0	0	0	0	0	0	0

Enable bit enables PRS block, and is maintained by the CSDADC API

Table 28. Block PRS, Register: Shift (DxBxxDR0), Bank 0

Bit	7	6	5	4	3	2	1	0
LSB	PRS Shift Register (LSB) - no direct access							
MSB	PRS Shift Register (MSB) - no direct access							

Table 29. Block PRS, Register: Polynomial (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
LSB	PRS Polynomial (LSB)							
MSB	PRS Polynomial (MSB)							

The PRS Polynomial is maintained by CSDADC API depending on ScanSpeed and Resolution parameters.

Table 30. Block PRS, Register: Seed (DxBxxDR2), Bank 0

Bit	7	6	5	4	3	2	1	0
LSB	PRS Seed/Compare register(LSB)							
MSB	PRS Seed/Compare register (MSB)							

The value of this register is maintained by API depending on almost all parameter values.

CSDADC with PWM8 Clock Source Configuration Registers

Table 31. Block CMP, Register: ACE_CONTROL1 (ACE01CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	1	Reference			1	1	1

Reference is maintained by user module parameter with the same name.

Table 32. Block CMP, Register: ACE_CONTROL2 (ACE01CR2), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	Power	

Power is maintained by CSDADC_Start() and CSDADC_Stop() APIs.

Table 33. Block CMP_REF, Register: ASE_CONTROL (ASE11CR0), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 34. Block CMP_REF, Register: ADC_CONTROL (ADC1_CR), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CMPST	1	1	0	0	Auto	0	ADCEN

The CMPST is read-only and is used by ADC-related API. Auto bit is maintained over ADC-related API. ADCEN enables ADC operations and is maintained by CSDADC API.

Table 35. Block CMP_REF, Register: ADC_TRIM (ADC1_TR), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	ADCTrimValue							

ADCTrimValue is over CSDADC API control. Contain ADC trim value for Single Slope ADC and zero value for CSD operation.

Table 36. Block CNT, Register: Function (DxBxxFN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	1

Table 37. Block CNT, Register: Input (DxBxxIN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 38. Block CNT, Register: Output (DxBxxOU), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	1	0	0	0	0	0	0

Table 39. Block CNT, Register: Control (DxBxxCR0), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

The Enable bit enables Counter operations, and is maintained by the CSDADC API.

Table 40. Block CNT, Register: Period (DxBxxDR0), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CounterValue not direct readable or writable							

Table 41. Block CNT, Register: Period (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	1	1	1

Table 42. Block CNT, Register: Compare (DxBxxDR2), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 43. Block PWM, Register: Function (DxBxxFN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	1	1	0	0	0	0	1

Table 44. Block PWM, Register: Input (DxBxxIN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	0	0	0

Table 45. Block PWM, Register: Output (DxBxxOU), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	1	1	0	0	0	1	0	0

Table 46. Block PWM, Register: Control (DxBxxCR0), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

Enable bit enables PRS block, and is maintained by the CSDADC API

Table 47. Block PWM, Register: Count (DxBxxDR0), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	Count register- no direct access							

Table 48. Block PWM, Register: Period (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	PrescalerPeriod							

The PrescalerPeriod is controlled by user module parameter with the same name.

Table 49. Block PWM, Register: Compare (DxBxxDR2), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	Compare							

The value of this register is maintained by API depending on Reference user module parameter.

CSDADC with PRS8 clock source Configuration Registers

Table 50. Block CMP, Register: ACE_CONTROL1 (ACE01CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	1	Reference			1	1	1

Reference is maintained by user module parameter with the same name.

Table 51. Block CMP, Register: ACE_CONTROL2 (ACE01CR2), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	Power	

Power is maintained by CSDADC_Start() and CSDADC_Stop() APIs.

Table 52. Block CMP_REF, Register: ASE_CONTROL (ASE11CR0), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 53. Block CMP_REF, Register: ADC_CONTROL (ADC1_CR), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CMPST	1	1	0	0	Auto	0	ADCEN

The CMPST is read-only and is used by ADC-related API. Auto bit is maintained over ADC-related API. ADCEN enables ADC operations and is maintained by CSDADC API.

Table 54. Block CMP_REF, Register: ADC_TRIM (ADC1_TR), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	ADCTrimValue							

ADCTrimValue is over CSDADC API control. Contain ADC trim value for Single Slope ADC and zero value for CSD operation.

Table 55. Block CNT, Register: Function (DxBxxFN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	1

Table 56. Block CNT, Register: Input (DxBxxIN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 57. Block CNT, Register: Output (DxBxxOU), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	1	0	0	0	0	0	0

Table 58. Block CNT, Register: Control (DxBxxCR0), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

The Enable bit enables Counter operations, and is maintained by the CSDADC API.

Table 59. Block CNT, Register: Period (DxBxxDR0), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CounterValue no direct readable or writable							

Table 60. Block CNT, Register: Period (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	1	1	1

Table 61. Block CNT, Register: Compare (DxBxxDR2), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 62. Block PRS, Register: Function (DxBxxFN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	1	1	0	1	0	1	0

Table 63. Block PRS, Register: Input (DxBxxIN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 64. Block PRS, Register: Output (DxBxxOU), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	0	0	ShieldElectrodeOut		

The ShieldElectrodeOut enables output signal for shield electrode to Row Output. It is controlled by the user module parameter with the same name.

Table 65. Block PRS, Register: Control (DxBxxCR0), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

Enable bit enables PRS block, and is maintained by the CSDADC API.

Table 66. Block PRS, Register: Shift (DxBxxDR0), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	PRS Shift Register - no direct access							

Table 67. Block PRS, Register: Polynomial (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	PRS Polynomial							

The PRS Polynomial is maintained by the CSDADC API, depending on ScanSpeed and Resolution parameters.

Table 68. Block PRS, Register: Seed (DxBxxDR2), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	PRS Seed/Compare register							

The value of this register is maintained by API depending on almost all parameter values.

CSDADC with VC2 clock source Configuration Registers.

Table 69. Block CMP, Register: ACE_CONTROL1 (ACE01CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	1	Reference			1	1	1

Reference is maintained by the user module parameter with the same name.

Table 70. Block CMP, Register: ACE_CONTROL2 (ACE01CR2), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	Power	

Power is maintained by the CSDADC_Start() and CSDADC_Stop() APIs.

Table 71. Block CMP_REF, Register: ASE_CONTROL (ASE11CR0), Bank x

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 72. Block CMP_REF, Register: ADC_CONTROL (ADC1_CR), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CMPST	1	1	0	0	Auto	0	ADCEN

The CMPST is read-only and is used by ADC-related API. Auto bit is maintained over ADC-related API. ADCEN enables ADC operations and is maintained by the CSDADC API.

Table 73. Block CMP_REF, Register: ADC_TRIM (ADC1_TR), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	ADCTrimValue							

ADCTrimValue is over CSDADC API control. Contain ADC trim value for Single Slope ADC and zero value for CSD operation.

Table 74. Block CNT, Register: Function (DxBxxFN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	1

Table 75. Block CNT, Register: Input (DxBxxIN), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 76. Block CNT, Register: Output (DxBxxOU), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	1	0	0	0	0	0	0

Table 77. Block CNT, Register: Control (DxBxxCR0), Bank 1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

The Enable bit enables Counter operations, is maintained by CSDADC API.

Table 78. Block CNT, Register: Period (DxBxxDR0), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	CounterValue, no direct readable or writable							

Table 79. Block CNT, Register: Period (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	1	1	1

Table 80. Block CNT, Register: Compare (DxBxxDR2), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Version History

Version	Originator	Description
1.1	DHA	Updated error messages.
1.20	DHA	1. Moved DisableInt macro call from polling loop into ISR (PRS16 CSD mode). 2. The connection between AnalogComparatorColumn and GlobalOutput line is displayed in the Interconnect view.
1.20.b	DHA	Added Help file to Wizard.
1.30	DHA	1. Transferred the DiplexTable from "AREA UserModules" to "AREA lit". 2. Set the default "DiplexTable" parameter value to 0x0112. 3. Added the "DiplexUsed" parameter to improve code compression.

Version	Originator	Description
1.40	DHA	1. Updated area declarations to support Imagecraft optimization. 2. Added symbolic names for the Resolution parameter in this user module datasheet. 3. Added Die temp measurement function description and updated the function description for SetScanMode() API. 4. Updated the CSDADC User Module to implement the CSDADC_StartTempMeasurement and CSDADC_GetTemperature functions. 5. Updated the resolution range calculation for Slider and Radial Slider in the user module wizard. 6. Updated the user module wizard help. Added a description of the slider resolution parameter min/max values.
1.40.b	DHA	Corrected resolution value calculation in UM wizard to address the error after change in diplexing.
1.40.c	DHA	1. Changed default "Reference" value from VBG to ASE11. 2. Updated user module block diagram in user module datasheet.
1.50	MYKZ	1. Added Resume() function to User Module API. 2. Fixed problem with saving information for sliders. 3. Updated baseline algorithm to check for negative difference counts. 4. Added build error message when user attempts to build project without first calling the user module wizard. 5. Updated UM Wizard pin assignment algorithm to take into account free pins. 6. Returned value for the CY8C24094 family was limited to (2^Resolution)-1 in interrupt service routine. 7. Optimized Start User Module function code. 8. Removed default value for feedback resistor pin and fixed feedback pin handling in User Module wizard.

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 © 2008-2013 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.