CapSense® Sigma-Delta Plus ADC Datasheet CSDADC V 1.50

Resources	PSoC® Blocks			API Memory (Bytes) Typical		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C24x94, CY8CLED0xD, CY8CLED0xG, CY8CLED04. Use of flash, RAM, and pins varies by the number of sensors and configuration.						
PRS16-based user module with 1 sensor	3	1	2	1257	34	2-5
PRS8-based user module with 1 sensor	1	1	2	1185	32	2-5
Prescaler clock source with 1 sensor	1	1	2	1185	32	2-5
VC2 clock source user module with 1 sensor	0	1	2	1170	1170	1
Each additional CapSense® button	-	-	-	2	10	1
Static code and RAM increase when capacitive slider with 5 elements is used	-	-	-	1197	79	5
Each additional slider element	-	-	-	2	10	1
Static code and RAM increase when slider diplexing is used	-	-	-	0	Sliders*2	-

Features and Overview

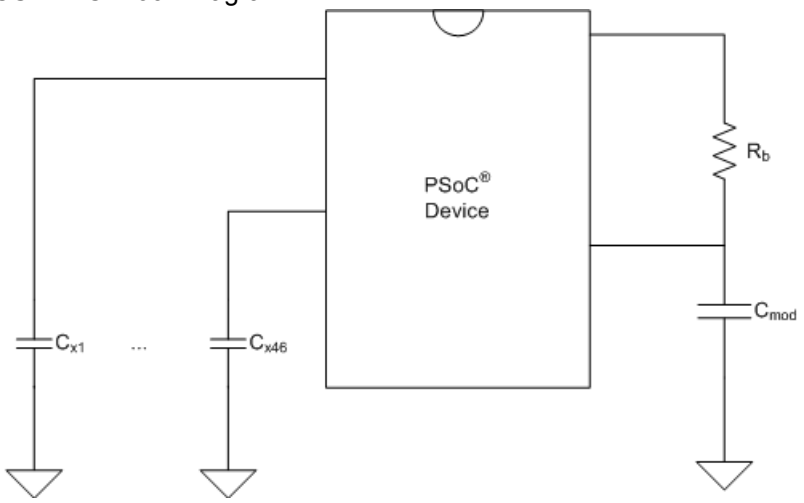
- Allows scanning CapSense sensors and measuring input voltages without using separate loadable configurations
- Uses a Sinc^N filter fully implemented in hardware to reduce CPU overhead and anti-alias requirements
- Supports a configuration that uses no digital blocks
- ADC features:
  - Sigma-delta ADC with second order modulator
  - Data in unsigned or signed 2’s complement formats
  - Dynamically changed resolution to 10,12, and 14 bits
  - Maximum sample rates of 31250 sps at 10-bit resolution, 7812 sps at 14-bit resolution
  - Input range defined by internal and external reference options

  - Built-in programmable gain amplifier with configurable gain and reference settings

- CapSense features:
  - Based on the robust CSD method
  - Second order modulator provides superior SNR performance
  - Scans 1 to 46 capacitive sensors
  - Sensing possible with up to a 25 mm glass overlay
  - Proximity detection to 30 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 (for example, ITO films)
  - 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 to support raw data monitoring and parameter optimization in real time

The CSDADC provides capacitance sensing using the switched capacitor technique with a sigma-delta modulator to convert the sensing switched capacitor current to digital code.

Figure 1. CSDADC Block 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 sensor count, configuration, and pin assignments.
5. Set pins and global parameters. Read all parameter descriptions and follow 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 I2C-USB 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 provides a combination of capacitance sensing with ADC functionality for voltage measurement without using a separate loadable configurations. It saves code space by reusing modules common to both the CSD and ADC. You must use the CSDADC when you need **both** capacitance sensing and ADC functionality. Applications that require one or the other must use the CSD or the DELSIG ADC.

A CSDADC implements capacitance sensing using the switched capacitor technique with a sigma-delta modulator (CSD) to convert the sensing switched capacitor current to a digital code. A CSDADC implements a sigma-delta ADC with a second order modulator and built in preamplifier. Using a second order modulator produces better SNR. This user module datasheet provides basic information about the CSD and ADC operation. For detailed information, recommendations about setting CSD parameters, CSD use tips, and CSD troubleshooting, see the CSD, DELSIG, and PGA datasheets and associated application notes. If this is the first time you are using the CSD and ADC User Modules, you must read these documents before attempting to implement a solution with them.

The following documents are recommended reading before you use the CSDADC User Module for the first time:

- *CY8C24x94 Series PSoC Programmable System-on-Chip Technical Reference Manual*, sections:
  - Two Column Limited Analog
  - Digital Clocks
  - IO Analog Multiplexer
- *Understanding Switched Capacitor Analog Blocks* – AN2041

The following design guides are recommended after reading the CSDADC User Module datasheet. These documents are available on the Cypress Semiconductor website at www.cypress.com:

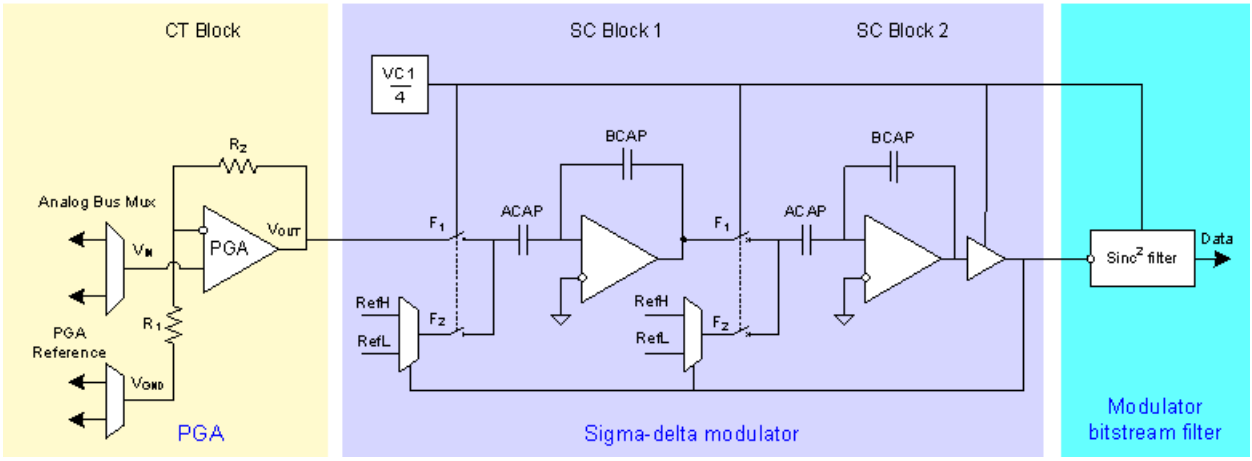
- Getting Started with CapSense
- CY8C20xx6A/H CapSense Design Guide
- CY8C21x34/B CapSense Design Guide

- CY8C20x34 CapSense Design Guide
- CY8CMBR2044 CapSense Design Guide

The ADC Operation

The CSDADC uses the DELSIG ADC with built-in PGA. The DELSIG User Module is an integrating converter, requiring from 64 to 256 integration cycles to generate a single output sample. Changing multiplexed inputs invalidates the first two samples following the change. The DELSIG ADC is composed of three primary modules: a PGA, a second order modulator, and a Sinc^2 Decimation Filter.

Figure 2. ADC Block Diagram



Programmable Gain Amplifier

The ADC is preceded by the programmable gain amplifier (PGA). This amplifier allows matching input signals range to the ADC signals range, giving better ADC dynamic range utilization. The PGA input comes from the analog bus, allowing it to be connected to external sources. It can also be reconfigured to support internal signal sources.

The PGA input signal can be referenced to the internal analog ground, V_{SS}, or other selected references. The gain of the programmable gain amplifier is set by programming the selectable tap in a resistor array and the feedback tap in a continuous time analog PSoC block. You can set the gain and reference in the device editor. The amplifier has the following transfer function:

Equation 1

$$V_{OUT} = (V_{IN} - V_{GND}) \cdot \left(1 + \frac{R_2}{R_1}\right) + V_{GND}$$

You can specify the reference as one of the following:

- A fixed value derived from an internal reference
- A value ratiometric to the supply voltage
- Analog ground
- An external input

The input and output voltage ranges of the amplifier do not extend to the power supplies (that is, they are not "rail-to-rail" opamps). The allowed input range is a combination of:

- Input limit

- Output limit
- Power supply voltage
- Analog ground value
- Selected gain

Modulator

The modulator is a 1-bit over-sampling circuit that represents the input voltage in terms of the density of 1's and 0's that it produces. The modulator output is reduced to the final sample rate by the low-pass decimation filter that converts multiple 1-bit samples into samples of higher resolution. In general, higher decimation rates (that is, higher oversample rates) can produce higher resolution results but other factors, such as the order of the modulator, also matter.

A key benefit of delta-sigma converters is the "noise shaping" provided by the modulator. Normally, the quantization noise inherent in sampling a signal is more or less evenly distributed ("white") in frequency between "DC" and one-half the sample frequency or Nyquist frequency. Simply put, the delta-sigma modulator shifts some of the quantization noise from lower into higher frequencies that are later attenuated by the decimation filter. A second order modulator that requires two switched-capacitor analog PSoC blocks does a better job of noise shaping than the first order modulator that only requires one analog PSoC block. At the highest decimation rate of 256X, a second order modulator accounts for a 3.5-bit increase in the effective resolution compared to a first order modulator. A second order modulator is constructed by feeding the analog output of a first order modulator into a similar PSoC block and modifying the feedback arrangement so that the 1-bit comparator output of the second block is back into both blocks as shown earlier.

Because the analog comparator buses run vertically in the columns of the analog PSoC block array, the blocks of a second order modulator must be positioned one above the other.

The range of the DelSig ADC is established by $\pm V_{Ref}$. You must set V_{Ref} in the Global Resources window in PSoC Designer™. For a fixed scale, V_{Ref} is set to $\pm V_{Bandgap}$ or for $\pm 1.6\ V_{Bandgap}$. For an adjustable scale, V_{Ref} is set to ±Port 2[6]. For a supply ratiometric scale, V_{Ref} is set to $\pm V_{DD}/2$. The complete list of options is given in the following table:

Table 1. Input Voltage Ranges for the Ref Mux Global Parameter Setting

RefMux Setting	V_{DD} = 5 V	V_{DD} = 3.3 V
$(V_{DD}/2) \pm$ BandGap	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
$(V_{DD}/2) \pm (V_{DD}/2)$	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
BandGap ± BandGap	$0 < V_{in} < 2.6$	$0 < V_{in} < 2.6$
(1.6*BandGap) ± (1.6*BandGap)	$0 < V_{in} < 4.16$	NA
(2*BandGap) ± BandGap	$1.3 < V_{in} < 3.9$	NA
(2*BandGap) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	NA
P2[4] ± BandGap	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

Sinc^2 Decimation Filter

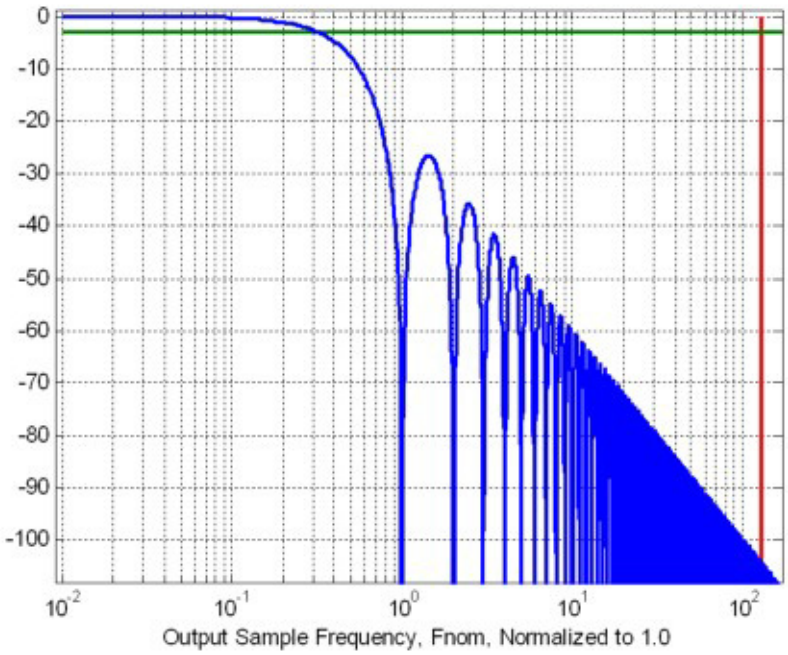
The response of the decimation filter is given by the following z-domain relation:

Equation 2

$$H(z) = \left[\frac{1 - z^{-n}}{1 - z^{-1}}\right]^2, \text{ where n is the decimation level.}$$

The frequency domain transfer function plotted in Figure 3 normalizes the frequency so the output sample rate, F_{nom}, equals 1.0. The -3 dB point occurs just above $0.318 \times F_{nom}$ and the zeros of the function occur at each integer multiple of F_{nom}. Since the 1-bit sample rate is 64 to 256 higher than the nominal output rate, the Nyquist limit is 5 to 7 octaves above F_{nom}, significantly reducing the requirements for an anti-alias filter. The 1-bit Nyquist frequency for a decimation rate of 256 is shown by the heavy vertical line at the right of the graph. Though higher decimation rates are possible, they contribute little additional benefit because of the noise floor of the device. In the case of the 12-bit topology, a second order modulator with a decimation rate of 256, the resolution is limited by the signal-to-noise ratio.

Figure 3. Sinc^2 Decimation Filter Magnitude Response, with -3 dB Point and Nyquist Frequency



The decimator operates in self-counting mode, without using additional timer block to form a decimation rate. The decimator implements the denominator of the transfer function by a double integrator operating at the 1-bit sample rate. The numerator is implemented by a double differentiator (second difference operator) that runs at the nominal output sample rate.

Capacitance Measurement Operation

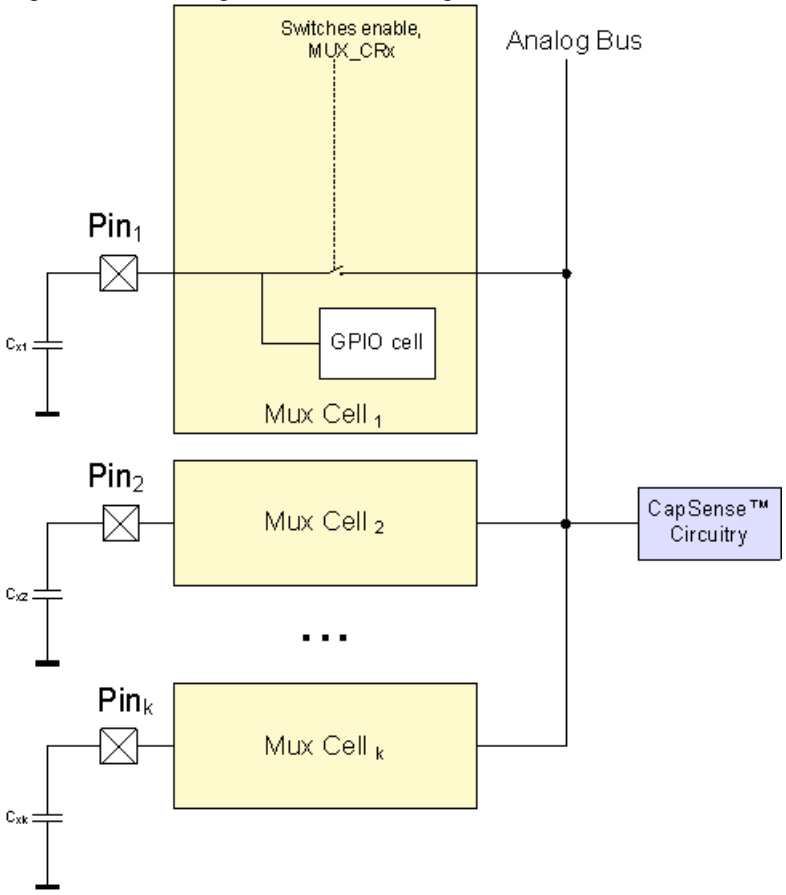
The decision logic is implemented in firmware. The firmware analyzes capacitance measurement, tracks the slow capacitance change due to environmental factors, and runs decision logic to detect button touches and calculate the slider position.

Scanning an Array of Sensors

The CY8C24x94 family of devices have two built-in analog buses. The two analog buses are connected together to provide the possibility of scanning sensors connected to any pins. The CSDADC User Module uses internal precharge switches to charge active sensors at clock signal phase Ph_1 and connects the Analog Buses 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.

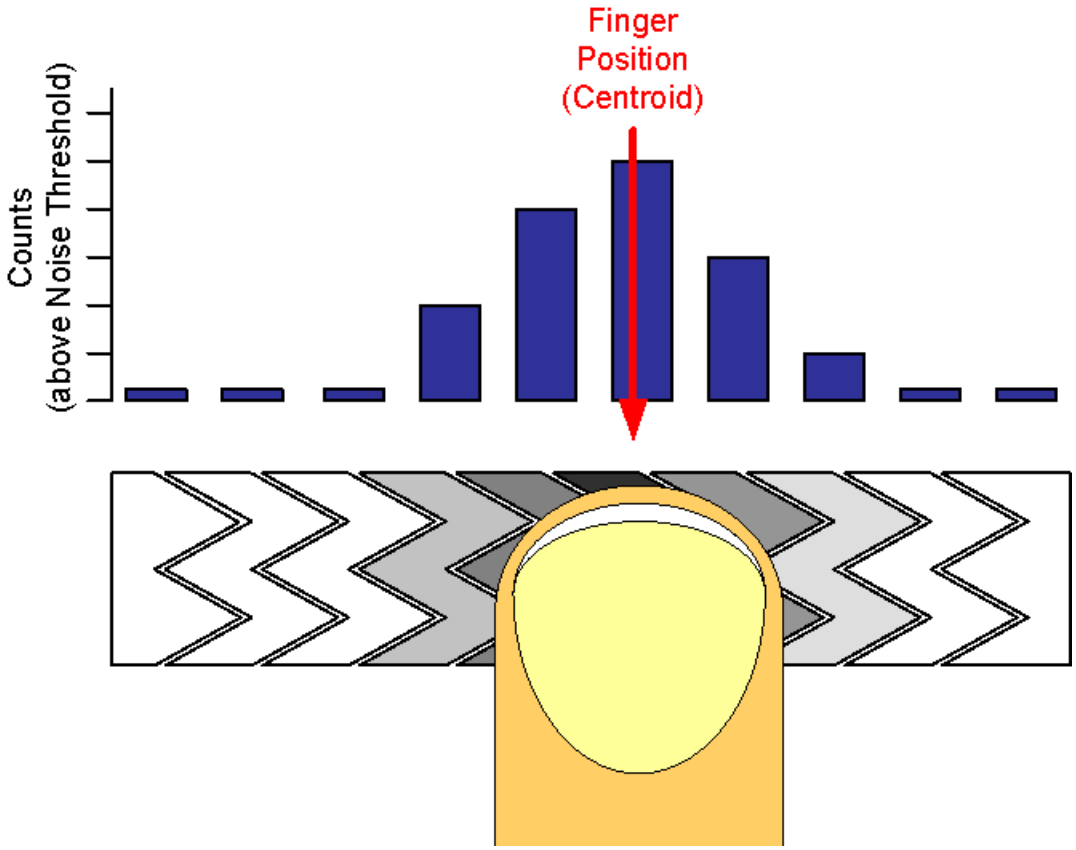
Figure 4. Analog Bus with Precharge Switches



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 number for sensors slider is five, the upper limit is simply the number of sensor positions available on the PSoC device selected.

Figure 5. Ordering Physical Sensor Locations



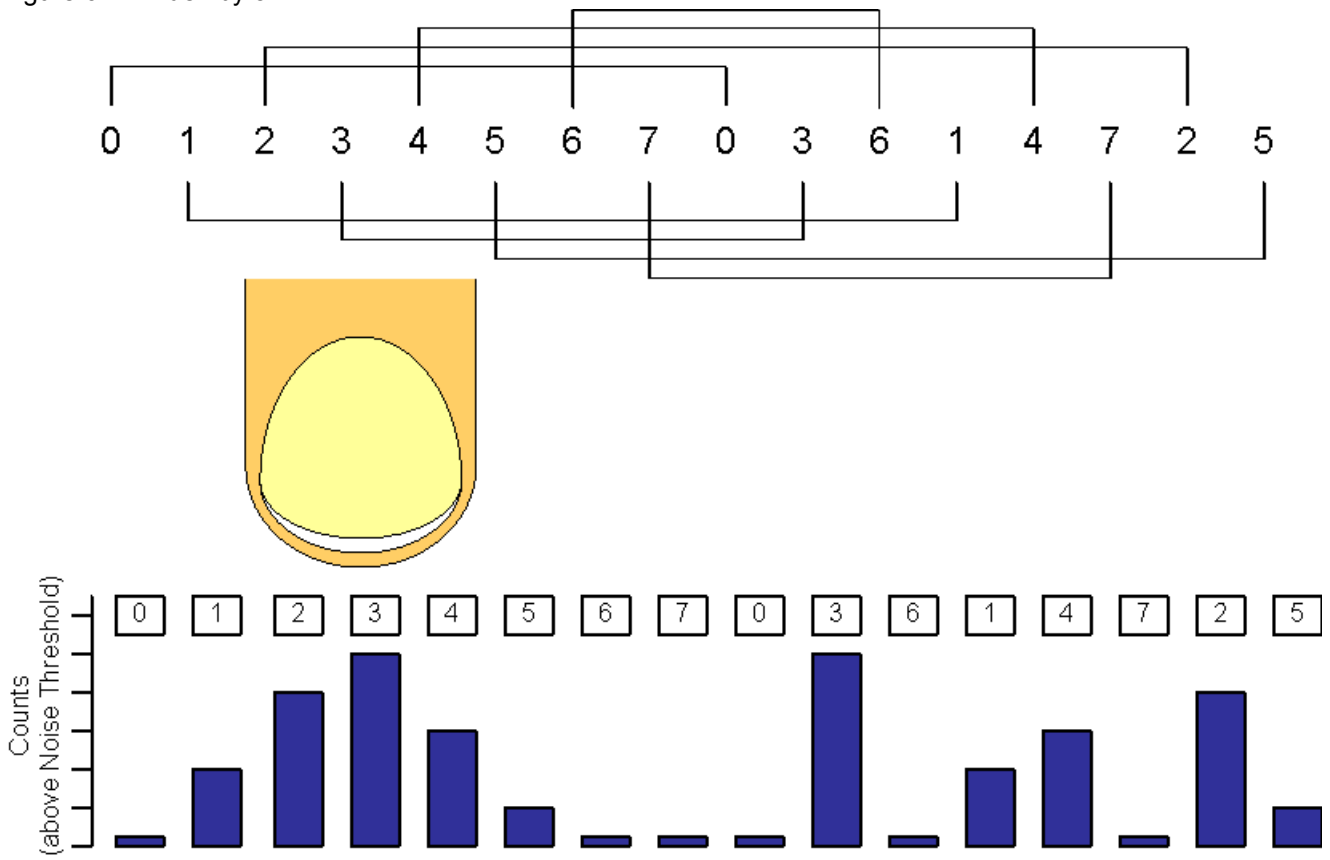
The close proximity of strong signals in one half of the slider results in the same levels aliased into the upper half, but the results are scattered. The sensing algorithms search for strong adjacent sets of signals to declare the resolved slider position.

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 method 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 6. Index by 3



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. This table illustrates the diplexing sequences for different slider segments count:

Table 2. 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
58	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,0,3,6,9,12,15,18,21,24,27,1,4,7,10,13,16,19,22,25,28,2,5,8,11,14,17,20,23,26
60	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,0,3,6,9,12,15,18,21,24,27,1,4,7,10,13,16,19,22,25,28,2,5,8,11,14,17,20,23,26,29
62	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,0,3,6,9,12,15,18,21,24,27,30,1,4,7,10,13,16,19,22,25,28,2,5,8,11,14,17,20,23,26,29
64	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,0,3,6,9,12,15,18,21,24,27,30,1,4,7,10,13,16,19,22,25,28,31,2,5,8,11,14,17,20,23,26,29

Total Slider Segment Count	Segment Sequence
66	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,0,3,6,9,12,15,18,21,24,27,30,1,4,7,10,13,16,19,22,25,28,31,2,5,8,11,14,17,20,23,26,29,32
68	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,0,3,6,9,12,15,18,21,24,27,30,33,1,4,7,10,13,16,19,22,25,28,31,2,5,8,11,14,17,20,23,26,29,32
70	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,0,3,6,9,12,15,18,21,24,27,30,33,1,4,7,10,13,16,19,22,25,28,31,34,2,5,8,11,14,17,20,23,26,29,32
72	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,0,3,6,9,12,15,18,21,24,27,30,33,1,4,7,10,13,16,19,22,25,28,31,34,2,5,8,11,14,17,20,23,26,29,32,35
74	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,0,3,6,9,12,15,18,21,24,27,30,33,36,1,4,7,10,13,16,19,22,25,28,31,34,2,5,8,11,14,17,20,23,26,29,32,35
76	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,0,3,6,9,12,15,18,21,24,27,30,33,36,1,4,7,10,13,16,19,22,25,28,31,34,37,2,5,8,11,14,17,20,23,26,29,32,35
78	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,0,3,6,9,12,15,18,21,24,27,30,33,36,1,4,7,10,13,16,19,22,25,28,31,34,37,2,5,8,11,14,17,20,23,26,29,32,35,38
80	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,0,3,6,9,12,15,18,21,24,27,30,33,36,39,1,4,7,10,13,16,19,22,25,28,31,34,37,2,5,8,11,14,17,20,23,26,29,32,35,38
82	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,0,3,6,9,12,15,18,21,24,27,30,33,36,39,1,4,7,10,13,16,19,22,25,28,31,34,37,40,2,5,8,11,14,17,20,23,26,29,32,35,38
84	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,0,3,6,9,12,15,18,21,24,27,30,33,36,39,1,4,7,10,13,16,19,22,25,28,31,34,37,40,2,5,8,11,14,17,20,23,26,29,32,35,38,41
86	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,1,4,7,10,13,16,19,22,25,28,31,34,37,40,2,5,8,11,14,17,20,23,26,29,32,35,38,41
88	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,2,5,8,11,14,17,20,23,26,29,32,35,38,41
90	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,2,5,8,11,14,17,20,23,26,29,32,35,38,41,44
92	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,2,5,8,11,14,17,20,23,26,29,32,35,38,41,44

Slider Segment Selection Guidelines for the Diplex Slider

Selecting the number of segments needed for a slider mainly 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.

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 3

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

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 required 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

The resolution is found by using this equation:

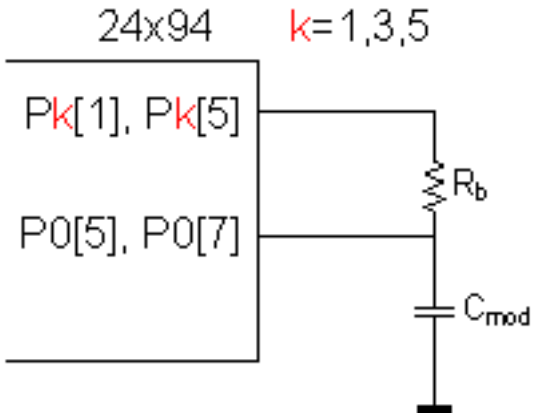
Resolution = (Number of Sensors – 1) x Multiplier

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

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[5], P0[7] port pins, and Vss ground. The feedback resistor R_b can be connected to the port pins P1[1], P1[5], P3[1], P3[5], P5[1], P5[5] and the capacitor pin. The pins are selected with the user module parameter setting. Do not use pins selected for modulator component connection 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 shown in the Wizard when it is opened.

Figure 7. External Component Connections



The recommended value for the modulation capacitor is 4.7 – 47 nF. The optimal capacitance can be selected by experiment to get maximum SNR. A value of 5.6 – 10 nF gives good results in most cases. **Note** If configuration with prescaler has been used, a larger modulator capacitor is required. A good quality 100 nF capacitor is sufficient in most cases. You can experiment with several capacitor values to get the best SNR after selecting the feedback resistor. A ceramic capacitor must be used. The temperature capacitance coefficient is not important. The resistor values depend on the total sensor capacitance C_s. The resistor value must be selected in the following method:

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

Typical values are 500 Ω – 10 kΩ depending on sensor capacitance and precharge switch operation frequency. You can start with 2.0 kΩ if you are using the CY3214 evaluation board.

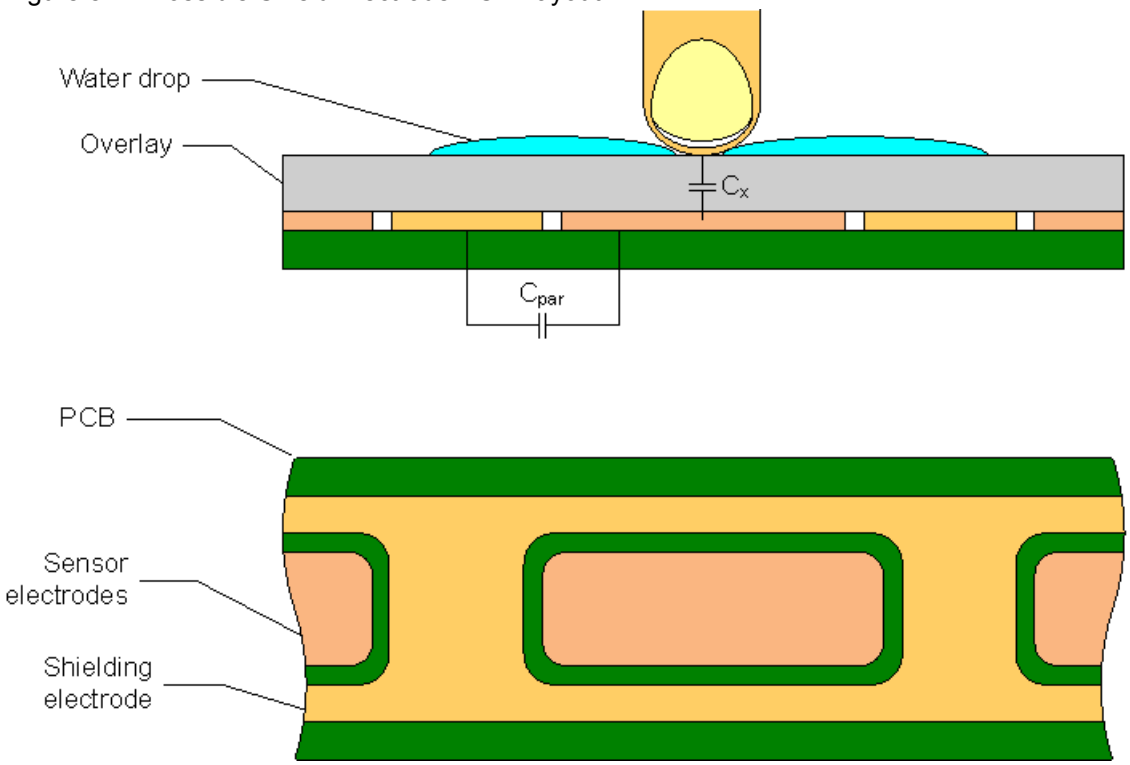
Shielding Electrode

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 located on the device insulation overlay surface, the coupling between the shielding and sensing

electrodes is increased. The shielding electrode allows you to reduce the influence of parasitic capacitance, which gives you more dynamic range for processing sense capacitance changes.

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

Figure 8. Possible Shield Electrode PCB Layout



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

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

When water drops are located between the shielding and sensing electrodes, the C_{par} is increased and the modulator current can be reduced. In practical tests, the modulator reference voltage can be increased by the API so that the raw count increase from water drops must be close to zero or slightly negative. You can achieve this by selecting the appropriate modulator reference.

The shield electrode can be connected to any PSoC pins to which it can be routed. Set the drive mode to **Strong Slow** to reduce ground noise and radiated emissions. In addition, the 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:

This table compares the four 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, use a PRS16 configuration.
Prescaler	fixed, IMO/(Period+1)	1	Device is sensitive for EMC signals at operation frequency and its harmonics. Recommended only for operation using high resistance materials.
VC2	fixed, IMO/(VC_1xVC_2)	0	Device is sensitive for EMC signals at operation frequency and its harmonics. Recommended only when no certification EMC/EMI tests are planned.

DC and AC Electrical Characteristics

Table 3. Power Supply Voltage

Parameter	Min	Typical	Max	Unit	Test Conditions and Comments
Value	3.0	5.0	5.25	V	

Table 4. Typical Noise

Parameter[a]	Fast	Normal	Slow	Unit	Test Conditions (V_{DD} = 5 V, SysClk = 24 MHz, CPU Clock = 12 MHz, Baseline >= 70% of Resolution Max Count) Gain = 4
Noise Counts, peak-peak	1	1	1	peak-to-peak	Resolution = 14
Noise Counts, peak-peak	1	2	2	peak-to-peak	Resolution = 12
Noise Counts, peak-peak	3	3	4	peak-to-peak	Resolution = 10

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

Table 5. Power Consumption

Supply Voltage	Min	Typ	Max	Unit	Test Conditions and Comments
Active Current		10		mA	Average current during scan, 8 sensors
Standby Current		250		µA	Scanning Speed = Ultra Fast, Resolution = 9 100 ms report rate, 8 sensors
		1.6		mA	Scanning Speed = Fast, Resolution = 12 100 ms report rate, 8 sensors
Sleep/Wake Current		10		µA	1s report rate, 1 sensor

Table 6. 5.0 V PGA DC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Gain Deviation from Nominal				
G=48.00	3.0	--	%	
G=24.00	2.2	--	%	
G=16.00	1.5	--	%	
G=4.00	0.7	--	%	
G=1.0	0.5	--	%	
Input				
Input Offset Voltage	4.5	--	mV	
Input Voltage Range	--	Vss to V_{DD}	V	
Leakage[1]	1	--	nA	
Input Capacitance[1]	3	--	pF	
Output Swing	0.05 to V_{DD}-0.05	--	V	
PSRR	73	--	dB	

Table 7. 5.0 V PGA AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Slew Rate(20% to 80%)[2]				
Low Power	0.6	--	V/µs	
Med Power	2.5	--	V/µs	
High Power	9.5	--	V/µs	
Settling Time				

Parameter	Typical	Limit	Units	Conditions and Notes
Low Power	13	--	µs	
Med Power	4	--	µs	
High Power	1	--	µs	
Noise[2]				Referred to input
Med Power	110		nV/√Hz	OpAmp bias low except at High Power. Reference input set to AGND.
High Power	100		nV/√Hz	

Table 8. 3.3 V PGA DC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Gain Deviation from Nominal				
G=48.00	4.0	--	%	
G=24.00	2.2	--	%	
G=16.00	1.2	--	%	
G=4.00	0.6	--	%	
G=1.0	0.3	--	%	
Input				
Input Offset Voltage	3.5	--	mV	
Input Voltage Range	--	Vss to V_{DD}	V	
Leakage[1]	1	--	nA	
Input Capacitance[1]	3	--	pF	
Output Swing	0.05 to V_{DD}-0.05	--	V	
PSRR	68	--	dB	
Operating Current				
Low Power	130	--	µA	
Med Power	520	--	µA	
High Power	2000	--	µA	

Table 9. 5.0 V ADC modulator DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to V_{DD}	V	Ref Mux = $V_{DD}/2 \pm V_{DD}/2$
Input Capacitance	3	---	pF	Includes I/O pin.
Input Impedance	1/(C*clk)	---	W	
Effective Resolution				
Decimate by 64	---	10	Bits	
Decimate by 128		12		
Decimate by 256		14		
Sample Rate				
Decimate by 64	---	31,250	sps	Data Clock 8 MHz
Decimate by 128		15625		
Decimate by 256		7812		
DC Accuracy				
DNL				
Decimate by 64	<1	---	LSB	Source Clock 1.5 MHz
Decimate by 128	<1			
Decimate by 256	0.6			
Offset Error	13	---	mV	
Gain Error	2		% FSR	Including Reference Gain Error
Data Clock	---	0.032 to 8.0	MHz	Input to digital blocks and analog column clock

Table 10. 3.3 V ADC modulator DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd	V	Ref Mux = $V_{DD}/2 \pm V_{DD}/2$
Input Capacitance	3	---	pF	Includes I/O pin.
Input Impedance	1/(C*clk)	---	W	
Effective Resolution				

Parameter	Typical	Limit	Units	Conditions and Notes
Decimate by 64	---	10	Bits	
Decimate by 128		12		
Decimate by 256		14		
Sample Rate				
Decimate by 64	---	31,250	sps	Data Clock 8 MHz
Decimate by 128		15625		
Decimate by 256		7812		
DC Accuracy				
DNL				
Decimate by 64	<1	---	LSB	Data Clock 1.5 MHz
Decimate by 128	<1			
Decimate by 256	0.5			
Offset Error	13	---	mV	
Gain Error	2		% FSR	Including Reference Gain Error
Data Clock	---	0.032 to 8.0	MHz	Input to digital blocks and analog column clock

Placement

The blocks for the user module are automatically placed when the user module is instantiated, alternate placements are not available. The modulator comparator is located at ACB01 continuous time block and ASD11/ASD21 switching capacitor blocks. Different UM configurations use 0-3 digital blocks.

This table summarizes the digital resources used.

Configuration	Digital Blocks Used
PRS16	3
PRS8	2
Prescaler	1
VC2 clock source	0

The unused analog and digital blocks are available for your own purposes. All UM configurations use the hardware decimator.

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.

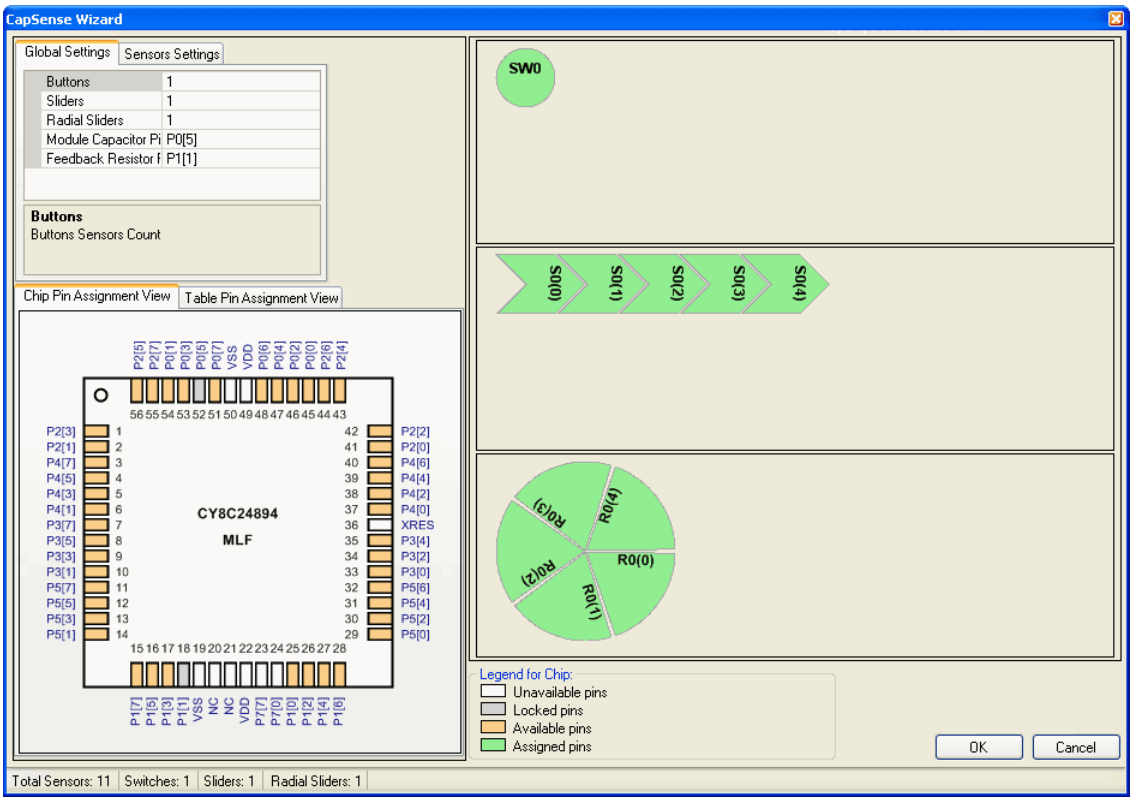
Avoid P1[0] and P1[1] when placing capacitive sensor connections. These pins are used for programming the part and may have excess routing capacitance affecting sensor sensitivity and noise.

Wizard

The CSDADC Wizard is used to set up the pinout for your CapSense buttons, sliders, and proximity sensors. You choose the configuration you want and assign the buttons and segments using a drag and drop interface.

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

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.

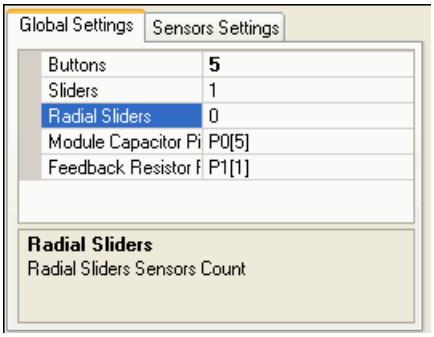
Gray – The pin is locked. There are two possible causes for this. The first possibility is that another user module such as the LCD or I^2C has claimed the pin. The second possibility is

that the name of the pin has been changed from its default.To return the pin name to its default, in the Pinout view expand the pin, from the **Select** menu, select **Default**. The pin is now available for assignment in the wizard.

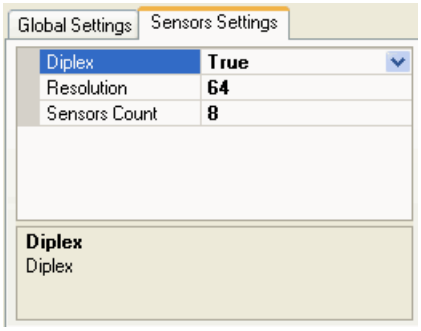
Orange – The pin is available for assignment.

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

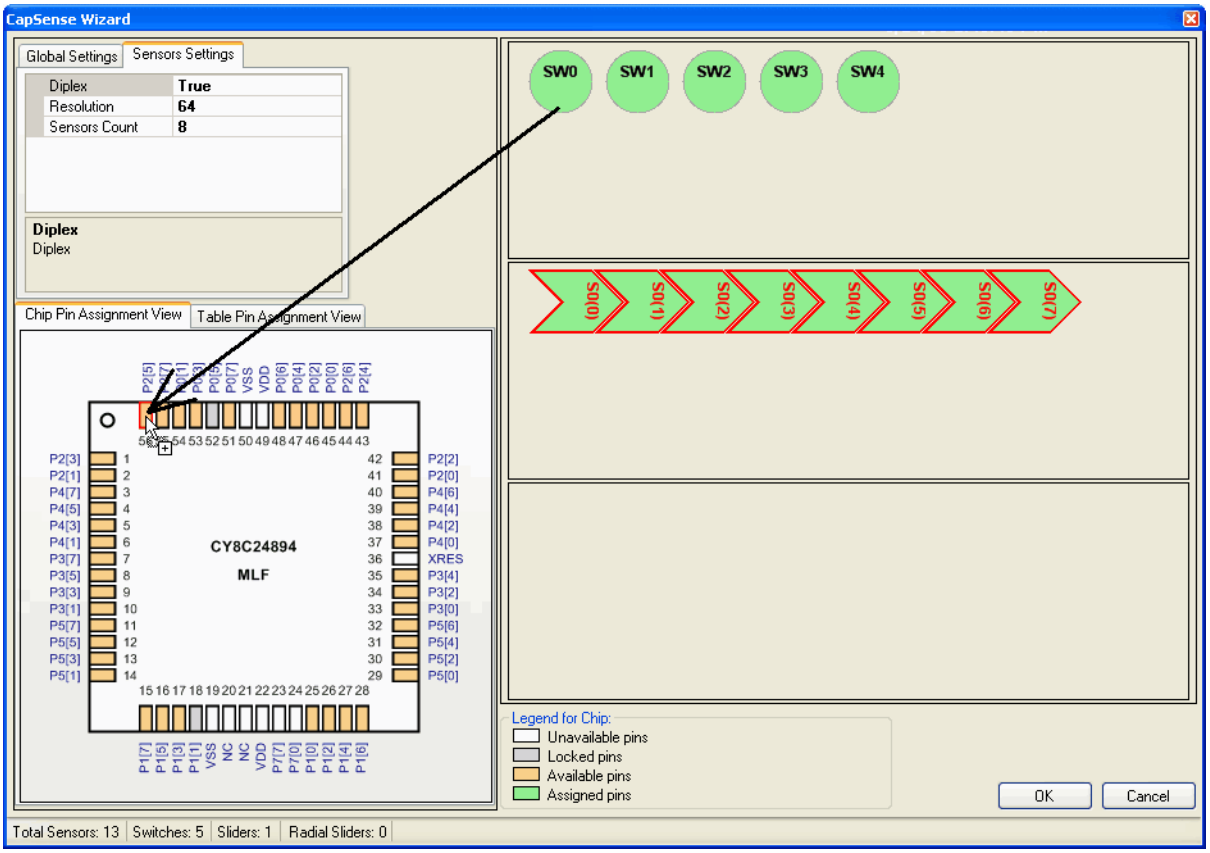
3. Type the number of independent buttons, sliders, and radial sliders. The total number of sensors is limited to the number of pins available. X-Y touchpads require two sliders but one is selected.



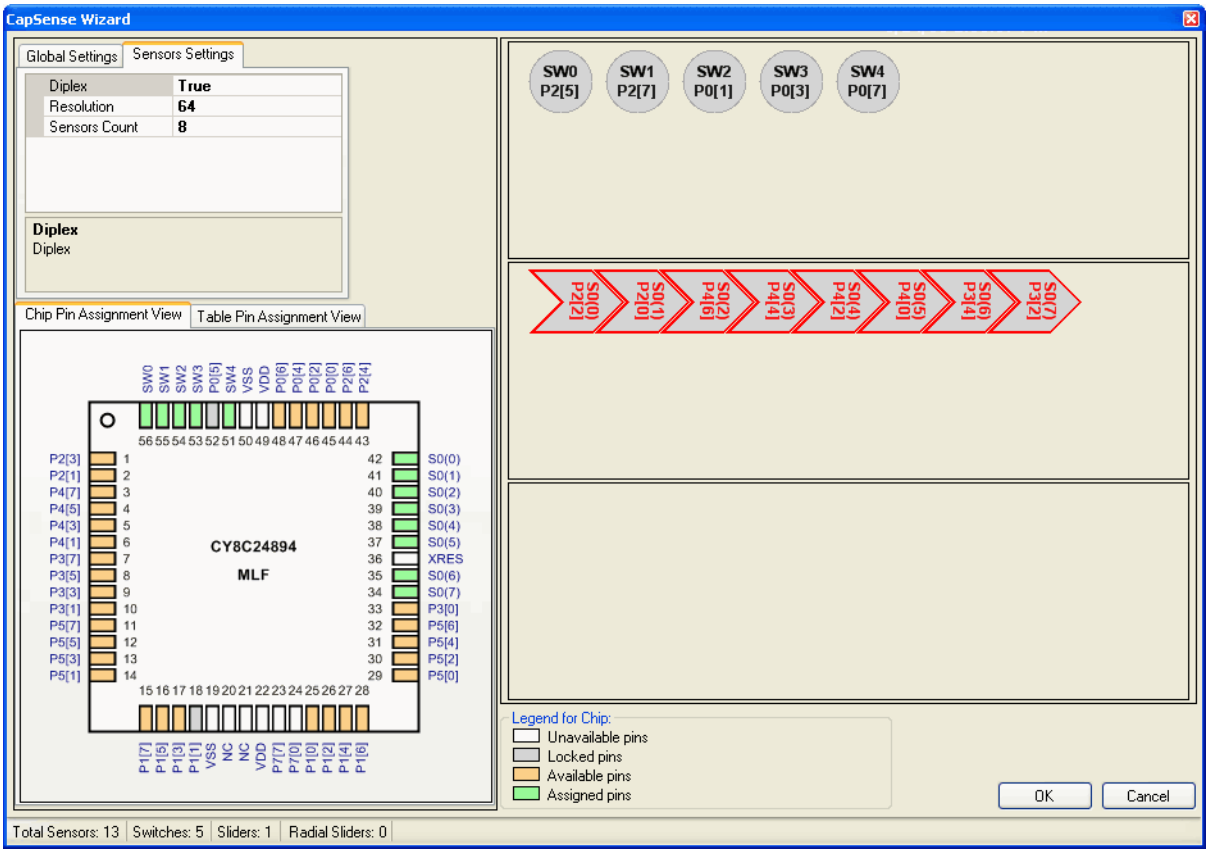
4. Select the **Sensor Settings** tab to set the sensor count and other settings for your sliders and radial sliders. Type the number of sensor elements in each slider. The practical minimum number of sensors in a slider sensor is five, the maximum is limited by pin count.



5. Type the output resolution. The minimum value is five. 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 desired. 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. Left 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 button off of the port pin.



8. Repeat for the remainder of the buttons.
9. Mapping of individual slider sensors onto 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 wish.

When entering the numerical values in the CSDADC Wizard, 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 is unassigned and you can then reassign it.

After completing the Wizard, click Generate Application. Based on your entries for sensor count, pin assignment, diplexing, and resolution, a set of tables is 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 ten sensors is:

```
CSDADC_Sensor_Table:
_CSDADC_Sensor_Table:
    dw    0x0001    // Port 0 Bit 0
    dw    0x0002    // Port 0 Bit 1
    dw    0x0004    // Port 0 Bit 2
    dw    0x0008    // Port 0 Bit 3
    dw    0x0010    // Port 0 Bit 4
    dw    0x0101    // Port 1 Bit 0
    dw    0x0102    // Port 1 Bit 1
```

```
    dw    0x0104    // Port 1 Bit 2
    dw    0x0108    // Port 1 Bit 3
    dw    0x0110    // Port 1 Bit 4
```

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 sensors. Each entry is six bytes. The first byte is the index in the Sensor Table where the group starts. The second byte is how many sensors are 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 is multiplied by to achieve the resolution desired in the CSDADC wizard.

```
CSDADC_1_Group_Table:
_CSDADC_1_Group_Table:
; Group Table:
;    Origin    Count    Diplex    DivBtwSw(wholeMSB, wholeLSB, fractByte)
 db    0x0,      0x5,     0x00,     0x00,     0x00,     0x00 ; Buttons
 db    0x5,      0x5,     0x3,      0x0,      0x0,      0x71 ; 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 five sensor slider is shown here.

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


 CSDADC_1_Diplex_Table:
_CSDADC_1_Diplex_Table:
    db >DiplexTable_0, <DiplexTable_0
    db >DiplexTable_1, <DiplexTable_1
```

Parameters and Resources

ADCEnabled

The ADCEnabled parameter can have 2 values:

- Enabled (default value)
- Disabled

When parameter is set to Enabled, the ADC routines are included in the user module compilable code and can be called from user code. When this parameter is set to Disabled, the ADC routines are not included by using conditional compilation directives. Setting this parameter to Disabled can be useful if no ADC is required longer by design, in this way a CSDADC allows using second order modulator for getting SNR improvement over standard CY8C24x94 CSD UM.

PGAGain

Sets the PGA gain for the ADC. Gain ranges are 1 to 48.00, using PSoC Designer or the CSDADC_SetGain routine provided in the API. Gain settings less than 1 are not supported. The default value is 1.00.

PGAReference

The gain of the PGA is referenced to a "ground" value that you select. Choices include AGND (on-chip analog ground), V_{SS}, and adjacent continuous time (CT) and switched capacitor (SC) blocks. CT and SC block connections allow for adjustable offsets as a controlled reference voltage. The CSDADC_SetRef API can be used to change the reference at run-time. The default PGA reference is AGND.

AnalogBus

The PGA block output may be routed through the analog PSoC block array's network of local interconnections and/or through an analog output bus. Setting the PGA User Module AnalogBus parameter to Disable, the default value, restricts the set of possible connections to the local network. If the PGA AnalogBus output is enabled onto the bus, care must be exercised to ensure that no other user module drives this same bus. This parameter is used by both the CSD and ADC portions of the user module. The default setting is Empty.

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.

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.

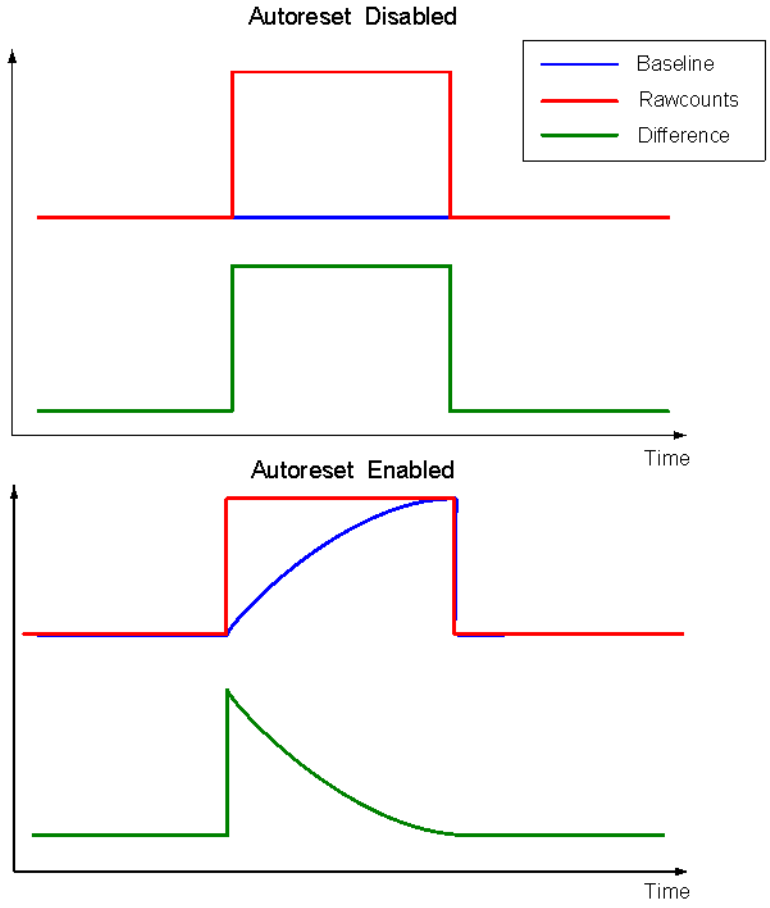
Sensors Autoreset

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 – 10s), 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. By default it is enabled.

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 Disabled unless you have problems with sensors permanently turning on when the raw count suddenly rises without anything touching the sensor.

The following figure illustrates this parameter's influence on the baseline update.

Figure 9. The 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.

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

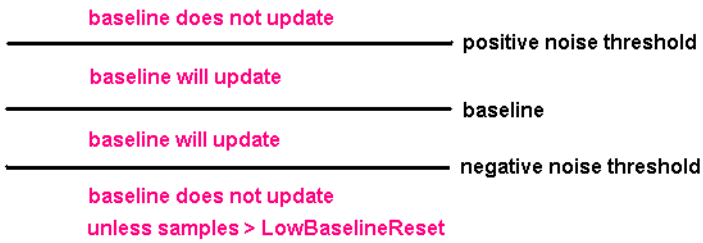
Debounce

The Debounce parameter adds a debounce counter to the sensor active transition. In order for the sensor to transition from inactive to active the difference count value must stay above the finger threshold plus hysteresis for the number of samples specified. 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 is not 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 is reset. Possible values are 0 to 255. The default value is 20.



Scanning Speed

This parameter affects the sensors' scanning speed. The available selections are **Fast**, **Normal**, and **Slow**. The default value 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

Resolution

This parameter determines the scanning resolution in bits. The PRS16 configuration allows scanning sensors with resolutions ranging from 9 to 16 bits. The default resolution is 12 bits. The PRS8, prescaler, VC2 configuration allows scanning sensors at 10, 12, and 14 bits only. If you need higher resolution with these configurations, use oversampling and average several samples. The maximum raw count for scanning resolution for N bits is 2^N-1. Same parameter value is used for ADC resolution setting.

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.

Table 11. Scanning Time in µs vs. Scanning Speed and Resolution for 24 MHz IMO Operation, PRS16 Configuration

Resolution, bits	Scanning speed		
	Fast	Normal	Slow
9	424	548	990
10	680	890	1670
11	1200	1580	3040
12	2220	2880	5800
13	4280	5680	11200
14	8400	11100	22300
15	16600	22100	44200
16	33000	44000	88000

Table 12. Scanning Time in µs vs. Scanning Speed and Resolution for 24 MHz IMO Operation, PRS8, Prescaler and VC2 Configurations

Resolution, bits	Scanning speed		
	Fast	Normal	Slow
10	124	136	296
12	220	220	548
14	428	552	1060

Note. The scanning time was measured as the time interval between 2 sensor scans. This time includes the sensor setup time, modulator stabilization delay, sample conversion interval and data pre-processing time.

Modulator Capacitor Pin

This parameter sets the pin to connect the external modulator capacitor (C_{mod}). Choose from the available pins. The default pin is P0[5].

Feedback Resistor Pin

This parameter sets the pin to connect the external feedback resistor (R_b). Choose from the available pins. Using P1[1] for the feedback resistor connection is not recommended due to possible problems with ISSP programming. Tip: If some of these pins are used for other purposes (for example, allocated for sensor connection), they are not available for selection in the UM parameter list. The default pin is P1[1].

Prescaler Period

This parameter sets the prescaler period register and determines the precharge switch output frequency. This parameter is available for PRS8 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.

PRS Polynomial

This parameter sets the PRS polynomial in the PRS8 configuration. There are two selection options:

- Short – The short polynomial setting yields better SNR, but due to the shorter repeat period, the end device can be more susceptible to external noise sources.
- Long – The long polynomial setting yields worse SNR, but the device is more robust against noise signals. It is default value.

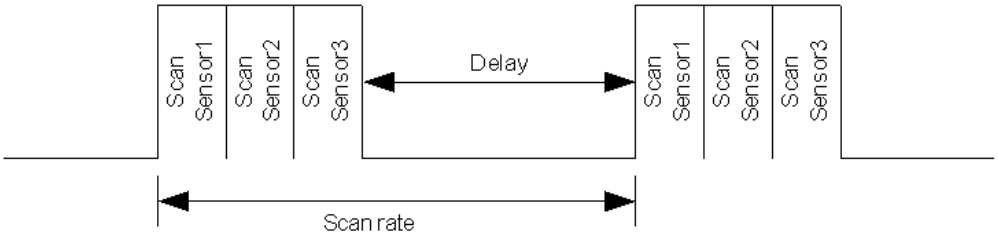
ShieldElectrodeOut

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 value is None.

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 10. 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.

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

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

Entry Points are supplied to initialize the 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.

API functions use different global arrays. You must 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 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 sensor. The array size is equal to the sensor count. The CSDADC_waSnsResult[] data is updated by these functions:

- CSDADC_ScanSensor();
- CSDADC_ScanAllSensors().
- CSDADC_ScanSensorsAveraging()

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

CSDADC_baSnsOnMask[] – This is a byte array that holds the 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_blsSensorActive(BYTE bSensor) function or CSDADC_bIsAnySensorActive() routines.

CSDADC_Start

Description:

Initializes registers and starts the user module. This function must 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:

**

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_SetRefMux

Description:

This routine sets a AGND, RefHi, RefLo references in system, overwriting a value, set in the Device Editor of PSoC Designer

C Prototype:

```
void  CSDADC_SetRefMux(BYTE bReference)
```

Assembly:

```
mov A, [Ref]
lcall  CSDADC_SetRefMux
```

Parameters:

Reference, can accept one of the following values:

Symbolic Name	Value	Description
REF_HALFVDD_BANDGAP	0x00	(Vdd/2) +/- Bandgap
REF_P24_P26	0x08	P2[4] +/- P2[6]
REF_HALFVDD_HALFVDD	0x10	(Vdd/2) +/- (Vdd/2)
REF_2BANDGAP_BANDGAP	0x18	2*BandGap +/- BandGap
REF_2BANDGAP_P26	0x20	2Bandgap +/- P2[6]
REF_P24_BANDGAP	0x28	P2[4] +/- BandGap
REF_BANDGAP_BANDGAP	0x30	BandGap +/- BandGap
REF_16BANDGAP_16BANDGAP	0x38	1.6BandGap +/- 1.6BandGap

Return Value:

None

Side Effects:

**

CSDADC_SetPGAGain

Description:

Sets the gain for the PGA block, overwriting value set in the Device Editor.

C Prototype:

```
void  CSDADC_SetPGAGain(BYTE bGainSetting)
```

Assembly:

```
mov   A, bGainSetting
lcall  CSDADC_SetPGAGain
```

Parameters:

bGainSetting: symbolic names provided in C and assembly include files, and their associated values, are given in the following table. PGA gain can be set from 1 to 48, settings below 1 are not supported. This function is common for ADC and CSD modes. In the first case it sets a ADC preamplifier gain, in the second it sets a modulator gain.

Symbolic Name	Value
PGA_G48_0	0x0C
PGA_G24_0	0x1C
PGA_G16_0	0x08
PGA_G8_00	0x18
PGA_G5_33	0x28
PGA_G4_00	0x38
PGA_G3_20	0x48
PGA_G2_67	0x58
PGA_G2_27	0x68
PGA_G2_00	0x78
PGA_G1_78	0x88
PGA_G1_60	0x98
PGA_G1_46	0xA8
PGA_G1_33	0xB8
PGA_G1_23	0xC8
PGA_G1_14	0xD8
PGA_G1_06	0xE8
PGA_G1_00	0xF8

Return Value:

None

Side Effects:

**

CSDADC_SetPGARef

Description:

Sets the reference for PGA block. The gain of the PGA is referenced to a "ground" value selected by the user. Choices include AGND (on-chip analog ground), V_{SS}, and adjacent continuous time (CT) and switched capacitor (SC) blocks. CT and SC block connections allow for adjustable offsets as a controlled reference voltage.

C Prototype:

```
void  CSDADC_SetPGARef(BYTE bRefSetting)
```

Assembly:

```
mov   A, bRefSetting
lcall  CSDADC_SetPGARef
```

Parameters:

bRefSetting: symbolic names provided in C and assembly, and their associated values, are given in this table.

Symbolic Name	Value
ADC00	0x00
AGND	0x01
VSS	0x02
ASC10	0x03

Return Value:

None

Side Effects:

**

CSDADC_EnableADC

Description:

Activates the ADC function of UM. It is allowed to call ADC related routines only after this routine call.

C Prototype:

```
void  CSDADC_EnableADC (void);
```

Assembly:

```
lcall CSDADC_EnableADC
```

Parameters:

None

Return Value:

None

Side Effects:

**

CSDADC_EnableCapsense

Description:

This routine enables the CapSense operation, setting the hardware configuration back to CapSense functions. It is allowed to call CapSense related routines after this routine call now.

C Prototype:

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

Assembly:

```
lcall CSDADC_EnableCapsense
```

Parameters:

None

Return Value:

None

Side Effects:

**

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.

C prototype:

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

Assembly:

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

Parameters:

bPort - sets the ADC input port

bMask - port bit.

Side Effects:

**

CSDADC_DisableInput

Description:

Disconnects the input port from ADC by disconnection a selected port pin from analog bus. The routines CSDADC_ EnableInput and CSDADC_ DisableInput must be called in pairs.

C prototype:

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

Assembly:

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

Parameters:

bPort - sets the ADC input port

bMask - port bit.

Side Effects:

**

CSDADC_StartADC

Description:

Turns on the ADC and runs it continuously. Global interrupts must be enabled for the ADC functioning.

C Prototype:

```
void CSDADC_StartADC(void)
```

Assembly:

```
lcall CSDADC_StartADC
```

Parameters:

None

Return Value:

None

Side Effects:

**

CSDADC_StopADC

Description:

Turns off the ADC and stops the conversion process immediately.

C Prototype:

```
void CSDADC_StopADC(void)
```

Assembly:

```
lcall CSDADC_StopADC
```

Parameters:

None

Return Value:

None

Side Effects:

**

CSDADC_fIsDataAvailable

Description:

Checks the status of the ADC.

C Prototype:

```
BYTE CSDADC_fIsDataAvailable(void)
```

Assembly:

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

Parameters:

None

Return Value:

Returns a nonzero value if data has been converted and is ready to read.

Side Effects:

**

CSDADC_wGetData

Description:

Returns converted data. CSDADC_fIsDataAvailable() must be called to verify that the data sample is ready.

C Prototype:

```
WORD CSDADC_wGetData(void)
```

Assembly:

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

Parameters:

None

Return Value:

Returns the converted data sample in unsigned integer format

Side Effects:

**

CSDADC_wGetDataClearFlag

Description:

Get data and resets the data-available flag.

C Prototype:

```
WORD CSDADC_wGetDataClearFlag(void)
```

Assembly:

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

Parameters:

None

Return Value:

Returns a ADC conversion sample in the unsigned integer format

Side Effect:

**

CSDADC_ScanSensor

Description:

Scans the capacitive selected 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:

A => Sensor Number

Return Value:

None

Side Effects

**

CSDADC_ScanSensorAveraging

Description:

Scans the selected sensor 2^{bPower} times and averages the result. The output data resolution is kept because the output data is shifted to the right by bPower bits. Averaging multiple scan results allows a higher signal to noise ratio (SNR). This function is faster than calling the CSDADC_ScanSensor() function multiple times when using the PRS8, prescaler, or VC2 precharge clock configurations because the $Sinc^2$ filter requires the CSDADC_ScanSensor() function to skip the first two samples.

If T is the single sensor scanning time using CSDADC_ScanSensor() function for the given resolution and scanning speed then the speed difference is:

- ScanSensorAveraging – $(T/3)*2^{bPower} + 2T/3$
- Multiple ScanSensor calls – $T*2^{bPower}$

You should use this function only when you are not capable of getting good SNR results from your application even after carefully tuning the results as described in the CY8C21x34/B CapSense Design Guide, for example when scanning through thick overlays, when using CapSense for proximity detection, or when you must operate at low frequencies such as when using ITO film.

C Prototype:

```
void CSDADC_ScanSensorAveraging(BYTE bSensor, BYTE bPower);
```

Assembly:

```
mov A,  bSensor
mov A,  bPower
lcall CSDADC_ScanSensorAveraging
```

Parameters:

bSensor: sensor number to be scanned.

bPower: Defines the number of times the sensor is scanned. The sensor is scanned 2^{bPower} times. Allowed values are in the range 0..8.

Return Value:

None

Side Effects

**

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

**

CSDADC_UpdateSensorBaseline

Description:

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

The Bucket Method uses the following algorithm.

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 NoiseThreshold.

C Prototype:

```
void CSDADC_UpdateSensorBaseline(BYTE bSensorNum)
```

Assembly:

```
mov   A,   bSensorNum
lcall  CSDADC_UpdateSensorBaseline
```

Parameter:

A => Sensor Number

Return Value:

None

Side Effects:

**

CSDADC_UpdateAllBaselines

Description:

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

C Prototype:

```
void  CSDADC_UpdateAllBaselines()
```

Assembly:

```
lcall CSDADC_UpdateAllBaselines
```

Parameters:

None

Return Value:

None

Side Effects:

**

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 bSensorNum)
```

Assembly:

```
mov  A,  bSensorNum
lcall CSDADC_bIsSensorActive
```

Parameters:

bSensor A => Sensor Number

Return Value:

Return value of 1 if active, 0 if not active

Side Effects:

**

CSDADC_bIsAnySensorActive

Description:

Checks the difference count array for all 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

Side Effects:

**

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 must 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.

Note If noise counts on the slider segments are greater than the noise threshold, this subroutine may generate a false centroid result. The noise threshold must be set carefully (high enough above the noise level) so that noise does 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 for resetting the baseline of an individual sensor.

C Prototype:

```
void CSDADC_InitializeSensorBaseline(BYTE bSensorNum)
```

Assembly:

```
mov A, bSensorNum
lcall CSDADC_InitializeSensorBaseline
```

Parameters:

A => Sensor Number

Return Value:

None

Side Effects:

**

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:

**

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:

**

CSDADC_SetScanMode

Description:

Sets scanning speed and resolution. 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_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. The value range depends on the user module configuration.

The following possible constants are given for the bResolution parameter for PRS16 Configuration:

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

The following possible constants are given for the bResolution parameter for bResolution parameter for PRS8, Prescaler, and VC2 Configurations:

Constant	Value
CSDADC_10_BIT_RESOLUTION	10
CSDADC_12_BIT_RESOLUTION	12
CSDADC_14_BIT_RESOLUTION	14

Return Value:

None

Side Effects:

**

CSDADC_ClearSensors

Description:

Clears all capacitive sensors to the nonsampling 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:

**

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:

**

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:

**

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:

**

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:

**

Sample Firmware Source Code

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

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

WORD wResult;

void main(void) {
   M8C_EnableGInt;

   CSDADC_SetRefMux(CSDADC_REF_BANDGAP_BANDGAP); //BandGap +/- BandGap
   CSDADC_Start();
   CSDADC_SetDefaultFingerThresholds();
   CSDADC_InitializeBaselines();

   while (1) {
     CSDADC_SetRefMux(CSDADC_REF_BANDGAP_BANDGAP); //BandGap +/- BandGap
     CSDADC_EnableCapsense();
```

```
    CSDADC_ScanAllSensors();
    CSDADC_UpdateAllBaselines();

    //Here you can insert a code for processing of CapSence data

    CSDADC_SetRefMux(CSDADC_REF_HALFVDD_HALFVDD);//(Vdd/2) +/-(Vdd/2)
    CSDADC_SetPGAGain(CSDADC_G1_00);
    CSDADC_EnableADC();
    CSDADC_EnableInput(0x01,0x02); // use P2[0]
    CSDADC_StartADC();

    // Skip three first samples
    while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();
    while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();
    while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();

    //Here you can insert a code for processing of ADC data

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

The same project in 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

area bss(RAM)          ;inform assembler that variables follow
bCounter:  blk 1

area text(ROM,REL)     ;inform assembler that program code follows

_main:

    M8C_EnableGInt

mov    A, CSDADC_REF_BANDGAP_BANDGAP
    lcall  CSDADC_SetRefMux       ; BandGap +/- BandGap

lcall  CSDADC_Start
    lcall  CSDADC_SetDefaultFingerThresholds
    lcall  CSDADC_InitializeBaselines

loop:
    mov    A, CSDADC_REF_BANDGAP_BANDGAP
    lcall  CSDADC_SetRefMux       ; BandGap +/- BandGap
```

```
    lcall  CSDADC_EnableCapsense
    lcall  CSDADC_ScanAllSensors
 lcall  CSDADC_UpdateAllBaselines

    ;Here you can insert a code for processing of CapSence data

    mov    A, CSDADC_REF_HALFVDD_HALFVDD
    lcall  CSDADC_SetRefMux        ; (Vdd/2) +/-(Vdd/2)

    mov    A, CSDADC_G1_00
    lcall  CSDADC_SetPGAGain

    lcall  CSDADC_EnableADC

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

    lcall  CSDADC_StartADC

    ; Skip three first samples

mov   [bCounter], 3
.scan:
lcall  CSDADC_fIsDataAvailable
    cmp    A, 0
    jz     .scan
    lcall  CSDADC_wGetDataClearFlag
    dec    [bCounter]
    jnz    .scan

; The ADC rezult is stored in A (LSB) and X(MSB)
    ; Here you can insert a code for processing of ADC data

lcall  CSDADC_StopADC

    mov    A, 0x01
    mov    X, 0x02
    lcall  CSDADC_DisableInput

    jmp    loop
```

Configuration Registers

Table 13. Block PGA, Register: ACB_CONTROL0_REG (ACB01CR0), Bank x

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

Gain is maintained by user module parameter with the PGA Gain name and ADC related APIs. Reference is maintained by PGA Ref User Module parameter.

Table 14. Block PGA, Register: ACB_CONTROL1 (ACB01CR1), Bank x

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

AnalogBus is maintained by user module parameter with the same name. Power is maintained by CSDADC_Start() and CSDADC_Stop() APIs.

Table 15. Block PGA, Register: ACB_CONTROL2 (ACB01CR2), Bank x

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

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

Table 16. Block PGA, Register: ACB_CONTROL3 (ACB01CR3), Bank x

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

The ExGain bit is set when ever a PGA Gain of 24 or 48 is selected.

Table 17. Block ADC1, Register: AtoD1cr0 (ASD11CR0), Bank x

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

Table 18. Block ADC1, Register: AtoD1cr1 (ASD11CR1), Bank x

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

Table 19. Block ADC1, Register: AtoD1cr2 (ASD11CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 20. Block ADC1, Register: AtoD1cr3 (ASD11CR3), Bank x

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

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

Table 21. Block ADC2, Register: AtoD2cr0 (ASC21CR0), Bank x

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

Table 22. Block ADC2, Register: AtoD2cr1 (ASC21CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	00	0	0	0	0	0	0	0

Table 23. Block ADC2, Register: AtoD2cr2 (ASC21CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 24. Block ADC3, Register: AtoD2cr3 (ASC21CR3), Bank x

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

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

Table 25. 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 26. 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 27. 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 28. 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, is maintained by CSDADC API

Table 29. 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 30. 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 31. 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 PRS8 Clock Source Configuration Registers

Table 32. Block PGA, Register: ACB_CONTROL0_REG (ACB01CR0), Bank x

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

Gain is maintained by a user module parameter with the PGA Gain name and ADC related APIs. Reference is maintained by the PGA Ref User Module parameter.

Table 33. Block PGA, Register: ACB_CONTROL1 (ACB01CR1), Bank x

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

AnalogBus is maintained by a user module parameter with the same name. Power is maintained by the CSDADC_Start() and CSDADC_Stop() APIs.

Table 34. Block PGA, Register: ACB_CONTROL2 (ACB01CR2), Bank x

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

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

Table 35. Block PGA, Register: ACB_CONTROL3 (ACB01CR3), Bank x

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

The ExGain bit is set when ever a PGA Gain of 24 or 48 is selected.

Table 36. Block ADC1, Register: AtoD1cr0 (ASD11CR0), Bank x

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

Table 37. Block ADC1, Register: AtoD1cr1 (ASD11CR1), Bank x

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

Table 38. Block ADC1, Register: AtoD1cr2 (ASD11CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 39. Block ADC1, Register: AtoD1cr3 (ASD11CR3), Bank x

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

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

Table 40. Block ADC2, Register: AtoD2cr0 (ASC21CR0), Bank x

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

Table 41. Block ADC2, Register: AtoD2cr1 (ASC21CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	00	0	0	0	0	0	0	0

Table 42. Block ADC2, Register: AtoD2cr2 (ASC21CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 43. Block ADC3, Register: AtoD2cr3 (ASC21CR3), Bank x

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

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

Table 44. 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 45. 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 46. 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 a user module parameter with the same name.

Table 47. 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, is maintained by CSDADC API

Table 48. Block PRS, Register: Shift (DxBxxDR0), Bank 0

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

Table 49. 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 CSDADC API depending on ScanSpeed and Resolution parameters.

Table 50. 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 PWM8 Clock Source Configuration Registers

Table 51. Block PGA, Register: ACB_CONTROL0_REG (ACB01CR0), Bank x

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

Gain is maintained by the user module parameter with the PGA Gain name and ADC related APIs. Reference is maintained by the PGA Ref User Module parameter.

Table 52. Block PGA, Register: ACB_CONTROL1 (ACB01CR1), Bank x

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

AnalogBus is maintained by user module parameter with the same name. Power is maintained by CSDADC_Start() and CSDADC_Stop() APIs.

Table 53. Block PGA, Register: ACB_CONTROL2 (ACB01CR2), Bank x

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

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

Table 54. Block PGA, Register: ACB_CONTROL3 (ACB01CR3), Bank x

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

The ExGain bit is set when ever a PGA Gain of 24 or 48 is selected.

Table 55. Block ADC1, Register: AtoD1cr0 (ASD11CR0), Bank x

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

Table 56. Block ADC1, Register: AtoD1cr1 (ASD11CR1), Bank x

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

Table 57. Block ADC1, Register: AtoD1cr2 (ASD11CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 58. Block ADC1, Register: AtoD1cr3 (ASD11CR3), Bank x

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

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

Table 59. Block ADC2, Register: AtoD2cr0 (ASC21CR0), Bank x

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

Table 60. Block ADC2, Register: AtoD2cr1 (ASC21CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	00	0	0	0	0	0	0	0

Table 61. Block ADC2, Register: AtoD2cr2 (ASC21CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 62. Block ADC3, Register: AtoD2cr3 (ASC21CR3), Bank x

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

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

Table 63. 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 64. 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 65. 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 66. 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, is maintained by CSDADC API

Table 67. Block PWM, Register: Count (DxBxxDR0), Bank 0

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

Table 68. Block PWM, Register: Period (DxBxxDR1), Bank 0

Bit	7	6	5	4	3	2	1	0
Value	PrescalerPeriod							

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

Table 69. 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 the API depending on Reference User Module parameter.

CSDADC with VC2 Clock Source Configuration Registers

Table 70. Block PGA, Register: ACB_CONTROL0_REG (ACB01CR0), Bank x

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

Gain is maintained by the user module parameter with the PGA Gain name and ADC related APIs. Reference is maintained by the PGA Ref User Module parameter.

Table 71. Block PGA, Register: ACB_CONTROL1 (ACB01CR1), Bank x

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

AnalogBus is maintained by the user module parameter with the same name. Power is maintained by CSDADC_Start() and CSDADC_Stop() APIs.

Table 72. Block PGA, Register: ACB_CONTROL2 (ACB01CR2), Bank x

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

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

Table 73. Block PGA, Register: ACB_CONTROL3 (ACB01CR3), Bank x

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

The ExGain bit is set when ever a PGA Gain of 24 or 48 is selected.

Table 74. Block ADC1, Register: AtoD1cr0 (ASD11CR0), Bank x

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

Table 75. Block ADC1, Register: AtoD1cr1 (ASD11CR1), Bank x

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

Table 76. Block ADC1, Register: AtoD1cr2 (ASD11CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 77. Block ADC1, Register: AtoD1cr3 (ASD11CR3), Bank x

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

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

Table 78. Block ADC2, Register: AtoD2cr0 (ASC21CR0), Bank x

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

Table 79. Block ADC2, Register: AtoD2cr1 (ASC21CR1), Bank x

Bit	7	6	5	4	3	2	1	0
Value	00	0	0	0	0	0	0	0

Table 80. Block ADC2, Register: AtoD2cr2 (ASC21CR2), Bank x

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

The CompBus bit is controlled over CSDADC APIs.

Table 81. Block ADC3, Register: AtoD2cr3 (ASC21CR3), Bank x

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

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

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.
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.

Version	Originator	Description
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.