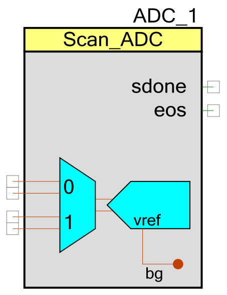
PSoC 6 Scanning SAR ADC (Scan_ADC)

2.10

Features



- 12-bit resolution
- Interleaved or channel-sequential averaging in hardware
- Up to 16-bit resolution with averaging
- Aggregate sample rate up to 1 Msps
- Single-ended and differential input modes
- Scheduler optimizes settling time and clock to fit scan rate
- Scan up to sixteen analog signals automatically
- Four run-time selectable configurations

General Description

The Scanning SAR ADC Component gives configuration-, schematic-, and firmware-level support for the version of the Successive Approximation Register (SAR) ADC present on the PSoC 6 family. Up to sixteen analog channels (from sources dependent on the specific device) can be automatically scanned, either on demand or continuously, with the results placed in individual result registers. The scan scheduler adjusts internal sampling behavior and clock to accommodate specific settling time and overall scan rate requirements. Averaging can be applied to any channel in a scan.

When to Use a Scanning SAR ADC

The Scanning SAR ADC is the Component used to access the ADC functionality in members of the PSoC 6 family. It is flexible and versatile in both high sample rate continuous-sampling applications (timed entirely in hardware), and lower-rate ad-hoc triggered scan applications.

The offset and span of the ADC depend on the parameters configured for the Component. Regardless of these settings, the analog signals connected to the PSoC's pins must be between V_{SSA} and V_{DDA}. For some settings, 'rail-to-rail' conversion is possible.

Input/Output Connections

This section describes the various input and output connections for the Scanning SAR ADC that may appear as terminals on the Component symbol. An asterisk (*) after the terminal name indicates that the terminal may not be present on the symbol under certain conditions.

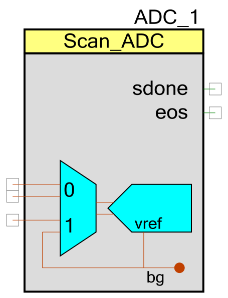
Note Throughout this document when signal connections are abbreviated, 's/e' means single-ended, 'diff' means differential.

Note During the sampling time for a given channel, its +Input, -Input, and/or vneg input signals connect directly to the input capacitor of the ADC core, and must charge that capacitor up before the actual conversion. A minimum input settling time value can be entered into each channel's parameter selections to allow for that channel's source impedance.

+Input – Analog Input

This input (not marked; it is always the upper terminal of a differential input pair on the symbol) is the 'positive' (also called non-inverting) analog signal input to the ADC. There are always the same number of 'positive' analog signal input terminals as there are channels selected, whether they are specified as differential or single-ended.

The following symbol has two channels, with channel zero configured as a single-ended channel using vref as the inverting input connection.



–Input – Analog Input*

This input (not marked; it is always the lower terminal of a differential input pair on the symbol) is the 'negative' (also called inverting) analog signal input to the ADC. It is only present for channels that have been declared as differential. On all channels declared as single-ended channels, the inverting input of the ADC is connected to the Vneg signal as described in the Vneg for S/E connection. There are always the same number of 'negative' analog signal input terminals as there are differential channels selected.

vneg – Analog Input*

This is a common negative input reference. This terminal is present only if one or more analog channels are declared as a single-ended input and the **Vneg for S/E** parameter is set to **External**.

soc – Digital Input *

This terminal is present if the “Use signal on soc terminal” box is checked in the Common tab. See the **Sample Mode** section for a description of how the soc terminal is used by the Component.

PSoC Creator Components can be stopped and started with firmware API calls. To allow for circuit stabilization, the first **soc** rising edge should be generated at least 10 µs after the Component is started.

aclk – Clock Input *

This terminal allows a PSoC clock to be connected to the Component. This mode is used when it is important that the clock used by the ADC is identical to that used by another Component on the schematic.

You can add this optional terminal if you check the **‘Show analog clock (aclk) terminal’** selection, otherwise, the terminal is hidden. Without this terminal, the Component will auto-select the ADC clock frequency, which may allow closer matching of user-specified sample rate.

sdone – Digital Output

This signal goes high for two ADC clock cycles to indicate that the ADC has sampled the current input channel. Internally, this signal is used to advance the signal multiplexer onto the next enabled channel.

eos – Digital Output

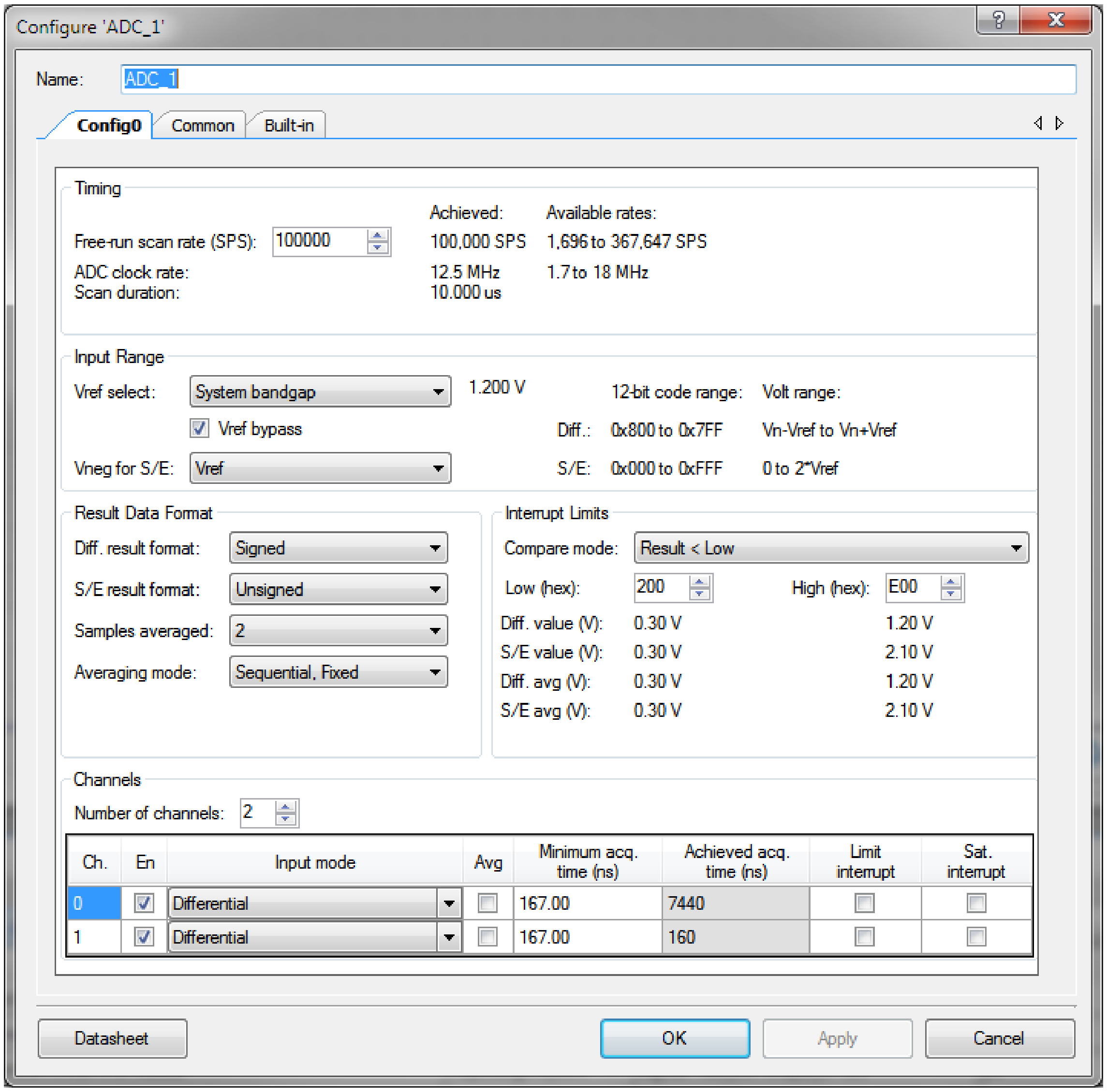
A rising edge on the end of scan (eos) output means that the current scan is complete. At this moment, conversion result registers contain valid sample data for all enabled channels. Internally, it is used to provide an interrupt.

Component Parameters

This section covers the various parameters that can be altered or inspected through the setup customizer of the Component, grouped within a series of tabs. The customizer supports up to four configurations, selectable at run time, each with its own schematic symbol and configuration sub-tab. To explore this, drag a Scanning SAR ADC onto your design and double click it to open the Configure dialog.

For any selectable parameter, the option shown here in **bold** is the default.

Config Tab (for each configuration) – Scan Sub-Tab



Name: ADC_1

Timing

		Achieved:	Available rates:
Free-run scan rate (SPS):	100000	100,000 SPS	1,696 to 367,647 SPS
ADC clock rate:		12.5 MHz	1.7 to 18 MHz
Scan duration:		10.000 us	

Input Range

Vref select: System bandgap 1.200 V

Vref bypass ✓

Vneg for S/E: Vref

	12-bit code range:	Volt range:
Diff.:	0x800 to 0x7FF	Vn-Vref to Vn+Vref
S/E:	0x000 to 0xFFF	0 to 2*Vref

Result Data Format

Diff. result format: Signed
S/E result format: Unsigned
Samples averaged: 2
Averaging mode: Sequential, Fixed

Interrupt Limits

Compare mode: Result < Low

	Low	High
(hex):	200	E00
Diff. value (V):	0.30 V	1.20 V
S/E value (V):	0.30 V	2.10 V
Diff. avg (V):	0.30 V	1.20 V
S/E avg (V):	0.30 V	2.10 V

Channels

Number of channels: 2

Ch.	En	Input mode	Avg	Minimum acq. time (ns)	Achieved acq. time (ns)	Limit interrupt	Sat. interrupt
0	✓	Differential	☐	167.00	7440	☐	☐
1	✓	Differential	☐	167.00	160	☐	☐

Timing

Free-run scan rate (SPS)

This is the fundamental parameter for the Scanning SAR ADC; the desired rate at which completed scans should be executed when the Component is running in Continuous mode. It is the rate at which each signal included in the scan is sampled. The Scanning SAR ADC Component customizer has a schedule calculator that works to get this sample rate as close as possible to the value that is entered. It does this by intelligent selection of ADC clock frequency (when an internal clock source is selected) and channel sampling times, taking all the other user-entered requirements into account.

When selected, the ADC clock rate is automatically calculated based on the number of channels, averaging, resolution, and acquisition time parameters to meet the entered sample rate.

Achieved (display only)

This field displays the currently-achieved scan rate that the Component will implement in a running system. The scheduler adjusts everything available to get as close as it can to the desired scan rate, but it is not always possible to achieve the desired scan rate.

The achieved scan rate is dependent on the following:

- ADC clock rate
- Number of channels
- Averaging
- Resolution
- Achieved acquisition time

The sample time for a channel is the time required to acquire the analog signal and convert it to a digital code:

$$Ch(i)\ Sample\ Time = Ch(i)\ Achieved\ Acquisition\ Time + \frac{(Resolution + 3)}{ADC\ Clock\ Rate}$$

Channels using one of the sequential averaging modes are sampled multiple times in each scan. Channels that are not averaged or use Interleaved averaging mode are only sampled once per scan. Let N be the number of channels in the scan and $Ch(i)SamplesPerScan$ be the number of samples averaged per scan for a channel. The achieved scan rate is therefore:

$$Achieved\ Scan\ Time = \sum_{i=0}^{N-1} (Ch(i)Sample\ Time) \cdot (Ch(i)SamplesPerScan)$$

$$Achieved\ Scan\ Rate = \frac{1}{Achieved\ Scan\ Time}$$

Example Configuration 1

- ADC clock rate = 18 MHz
- Number of channels = 1
- Resolution = 12-bit
- CH0
  - Averaging = None
  - Resolution = 12-bit
  - Achieved acquisition Time = 167 ns

Achieved Scan Rate = $1/\left(\left(167\,ns\ +\frac{(12+3)}{18\,MHz}\right)*1\right)=1\,MSPS$

Example Configuration 2

- ADC clock rate = 18 MHz
- Number of channels = 3
- Resolution = 12-bit
- CH0
  - Averaging = None
  - Achieved acquisition time = 167 ns
- CH1
  - Averaging = Sequential, Sum with 4 samples averaged
  - Achieved acquisition time = 167 ns
- CH2
  - Averaging = None
  - Achieved acquisition time = 167 ns

Achieved Scan Rate =

$$1/\left(\left(\left(167\,ns\ +\frac{(12+3)}{18\,MHz}\right)*1\right)+\left(\left(167\,ns\ +\frac{(12+3)}{18\,MHz}\right)*4\right)+\left(\left(167\,ns\ +\frac{(12+3)}{18\,MHz}\right)*1\right)\right)=167\,kSPS$$

Available rates (display only)

This field shows the approximate minimum to maximum range of scan rates that can currently be attained with the setup as defined. If the desired free-running rate is less than the minimum rate shown here, one solution is to set up a TC/PWM timer on the schematic and use it to trigger the ADC periodically in single shot triggered mode. Other options are to use averaging or set up a dummy channel.

ADC clock rate (display only)

This field displays the currently-selected actual ADC clock frequency. It is an integer divide from the PeriClk on PSoC 6. This clock frequency is configured at run time; therefore, this value will not necessarily match what is in the DWR clock tab during build time.

Scan duration (display only)

This field gives the duration of the achieved overall scan, in ns.

Input Range

Vref select

The **Vref** parameter selects the reference voltage source that is used for the ADC core, and optionally enables a numeric value to be given to it if the customizer does not know it.

Reference	Description
System Bandgap	**Dedicated internal connection to the main 1.2 V reference**
External device pin	Some PSoC 6 devices support a dedicated pin used for the Vref off-chip bypass capacitor and for the injection of a reference external to the chip. Checking the Vref bypass box has no effect in this mode. If the selected PSoC 6 device does not support this dedicated pin, this reference option will not be visible.
Vdda/2	An internal resistor divider produces Vdda/2 as a reference
Vdda	Uses the internal analog supply voltage applied to the Vdda terminal(s). An off-chip bypass capacitor has no effect in this mode.

The internal Vref startup time varies with different bypass capacitors. This table lists two common values for the bypass capacitor and its startup time specification.

Internal Vref Startup Time	Maximum Specification
Startup time for reference with external capacitor (50 nF)	120 µs
Startup time for reference with external capacitor (100 nF)	210 µs

Vref value (user entry or parameter display)

To the right of the Vref select pull-down, this parameter either displays the reference voltage value that is being used for the SAR ADC (if this is 'known' to PSoC Creator) or enables the entry of a value for display purposes, if only the user knows this value.

Vref shall not be less than 0.85 V, and setting it so causes an error.

Vref bypass

Checking this box indicates to the Component customizer that you have attached an off-chip bypass capacitor to the specific device pin set aside for this. It permits the Component to select higher ADC clock rates and therefore significantly higher overall scan rates.

The use of an off-chip reference bypass capacitor (ideally 50 nF or greater, ideally X7R dielectric or better) is recommended in all systems. It should only be omitted when there is really no room for it on the build. When omitted, the maximum aggregate sample rate is reduced by at least a factor of ten, and conversions are more prone to digital noise on the circuit board.

If the selected PSoC 6 device does not support a dedicated device pin for an off-chip bypass capacitor, this check box will not be visible.

Vneg for S/E

This parameter selects where the negative input to the SAR ADC is connected if any channels are configured for single-ended operation.

Negative input	Description
Vssa	Input range is 0.0 to Vref, effective resolution will be one bit less than selected in the customizer.
Vref	**Input range is 0.0 to Vref*2.**
External	This mode is configured for "quasi-differential" inputs. Multiple channels share one common –ve (inverting) connection. This is often used for common-mode rejection of ground noise in multi-channel systems.

12-bit code range (display only)

This field displays what code ranges will be returned by the SAR ADC. The values displayed are truncated at 12-bits. However, the results returned will be sign extended to the 16 or 32 bit format depending on which GetResult function is used.

Volt range (display only)

This field displays the voltage range of the SAR ADC using the selected Vref. For single ended channels the selection of Vneg is also used to determine the range.

Result Data Format

Differential (Diff.) result format

This parameter determines whether or not the result from a differential measurement is **Signed** or Unsigned. This is a global setting for all differential channels. Results are always right-justified.

S/E result format

This parameter determines whether or not the result from a single-ended measurement is Signed or **Unsigned.** This is a global setting for all single-ended channels. Results are always right-justified.

The following table shows how these parameters affect conversion of the input voltage to the 12-bit digital sample value.

s/e or diff	Signed / Unsigned	Single-ended negative input	-Input	+Input	Result Register
s/e	Unsigned: Use this mode only with caution	Vssa	Vssa	Vref Vssa -noise	0x0FFF 0x0800 0x07xx (this causes a wrap-round in calculations)
s/e	Signed	Vssa	Vssa	Vref Vssa -noise	0x07FF 0x0000 0xFFxx
s/e	Signed	External	Vneg	Vneg+Vref Vneg Vneg-Vref	0x07FF 0x0000 0xF800
s/e	***Unsigned***	***Vref***	***Vref***	***2*Vref*** ***Vref*** ***Vssa***	***0x0FFF*** ***0x0800*** ***0x0000***
s/e	Signed	Vref	Vref	2*Vref Vref Vssa	0x07FF 0x0000 0xF800
diff	Unsigned	N/A	Vx	Vx+Vref Vx Vx-Vref	0x0FFF 0x0800 0x0000
diff	**Signed**	**N/A**	**Vx**	**Vx+Vref** **Vx** **Vx-Vref**	**0x07FF** **0x0000** **0xF800**

For single-ended conversions with the **Vneg for S/E** parameter set to **Vssa**, the usable conversion is effectively 11-bit. Noise or offset on the **+Input** terminal with a level slightly below

Vssa produces a result that appears more positive than full scale. This can cause severe system problems, so this mode should be used with caution.

Samples averaged

This parameter sets the averaging rate for any channel with the averaging option enabled. This is a global setting for all channels that have averaging enabled. Default value is **2**.

Note that the Interleaved, Sum averaging mode option does not support result realignment, it is a simple accumulation in a 16-bit register. This mode does not support more than 16 samples of averaging.

Averaging mode

This parameter sets how the hardware averaging mode operates. If ***Sequential, Sum*** is selected, each ADC conversion result is added to a running sum. It's then shifted at the end of the scan so that it fits into a 16-bit result word. If the ***Sequential, Fixed*** mode is selected, accumulated result is shifted back into a 12-bit result.

In either sequential mode, the scan pauses on the channel being averaged and all the samples for the average are taken before moving onto the next channel in the scan. This can reduce the maximum available scan rate substantially when any channel in the scan is averaged in this way. For this reason, the Interleaved, Sum mode is also available. In Interleaved mode, only one conversion is taken on each channel before moving on, but channels that have averaging enabled get the preset number of samples accumulated in their result register.

In ***Interleaved, Sum*** *mode* the overall scan rate is not reduced. This means that channels not requiring averaging can still be sampled at the original scan rate. An end of scan interrupt is still produced at the end of every scan; channels that utilize interleaved averaging are not marked as 'valid' until the correct number of scans have been taken.

If every channel is set to use averaging and the mode is set to Interleaved, Sum then the rate of end-of-scan interrupts is significantly reduced. The interrupt will happen when all the channels have new 'valid' data, after completing the same number of scans as the Samples averaged parameter.

Interrupt Limits

Compare mode

The Scanning SAR ADC supports range detection to allow for the automatic detection of sample values compared to two programmable thresholds without CPU involvement. A range detect is defined by two global thresholds and a condition.

This parameter sets the condition under which a limit condition will occur and trigger a maskable range detect interrupt.

Compare Mode	Description
Result < Low	Below range

Compare Mode	Description
Low <= Result < High	Inside range
High <= Result	Above range
(Result < Low) or (High <= Result)	Outside range

Low (hex)

This parameter sets the low threshold in hex for a limit compare. Default value is **0x0200**. For Signed modes, the SAR results are two's-complement.

High (hex)

This parameter sets the high threshold in hex for a limit compare. Default value is **0x0E00**.

A range detect is done after averaging, alignment, and sign extension (if applicable). In other words, the thresholds values must have the same data format as the final 16-bit conversion result.

Equivalent input voltages

Directly beneath the low and high limit entry fields, the corresponding voltage values are displayed for individual and averaged differential and single-ended measurements.

Channels

Number of channels

This parameter selects how many input signal channels are scanned. By default, there are **2** channels. The maximum number of channels is 16. The minimum number of channels is 1.

A set of parameters is available for each entry. The actual number of entries depends on the **Number of channels** parameter. The symbol shows as many channels as are selected by the **Number of channels** parameter even if the channel is not enabled.

Ch.

Shows the number of the channel, starting from 0. The number of entries here is determined by the Number of Channels parameter.

En

If checked, the channel is enabled in the scan. If unchecked, no time is consumed and the scan jumps immediately to the next enabled channel in the scan list.

Input mode

For any channel, this parameter selects the input mode to the ADC as either **Differential** or Single ended.

Avg

This option selects whether or not the channel is averaged. When selected and a sequential averaging mode is selected, the SAR sequencer stays on the channel and takes N readings, then adds the results together. The number of samples taken is determined by the **Samples averaged** parameter. Averaging is always right-aligned.

Minimum acq. time (ns)

The user can enter a minimum acquisition time (in ns) that the input sampling process will dwell on this channel before actually making the conversion. The field is editable but is pre-populated with the shortest value currently possible with the system clock parameters.

Achieved acq. time (ns)

This display field shows the acquisition time (in ns) that the scheduler has selected. It is always equal to or higher (longer duration) than the user-requested value.

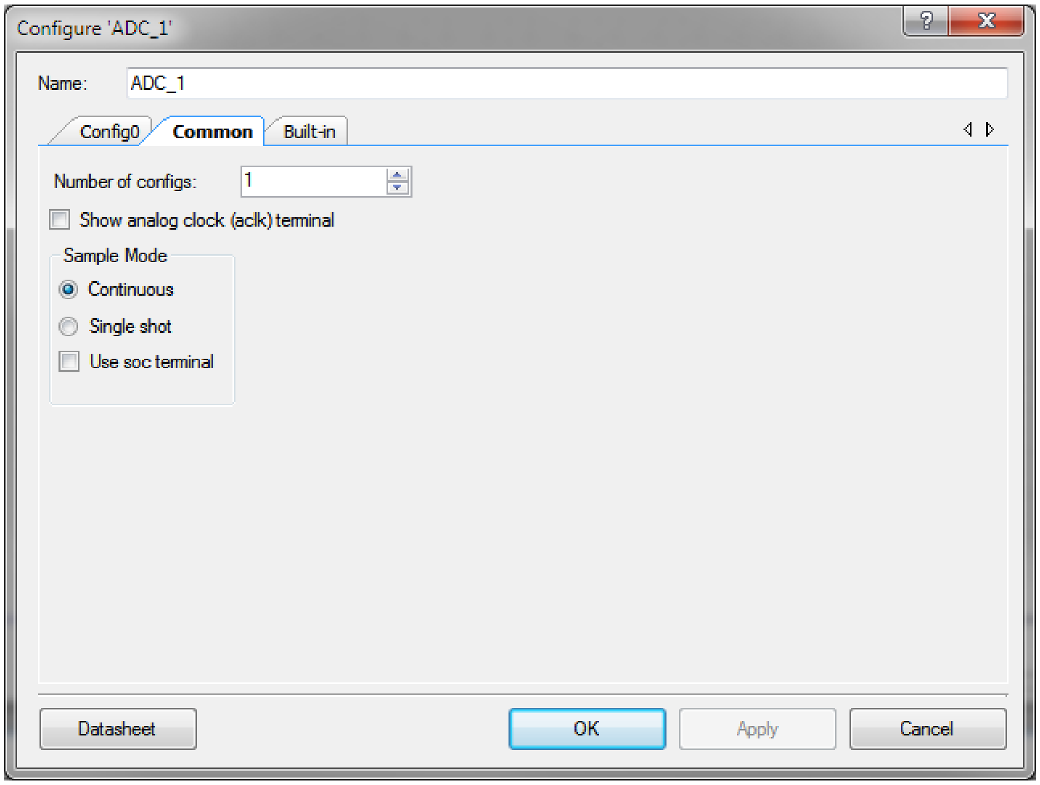
Limit interrupt

This option allows you to enable an interrupt if any of the channels trigger the limit criteria set by the **Low** or **High** thresholds and the **Compare mode** parameter after averaging.

Sat. interrupt

This option allows you to enable an interrupt from any channel where the result is saturated at either the lowest or the highest value for the given format before averaging.

Common Tab



Name: ADC_1

Number of configs: 1

[ ] Show analog clock (aclk) terminal

Sample Mode

(●) Continuous

( ) Single shot

[ ] Use soc terminal

Number of configs

Between **1** and 4 complete configurations can be defined in the Component. There is an API function call to select which configuration is in operation. Each configuration gets its own symbol and its own tab.

Space between config symbols (grid units)

When using more than one configuration, this controls the space between the symbols. This space can be between 10 and 45 grid units wide, the default is **15**.

Show analog clock (aclk) terminal

If this box is checked, the external analog clock (aclk) terminal will appear on the symbol.

Sample Mode

The Scanning SAR ADC can operate in one of two modes, triggered by firmware or hardware in either mode:

Sample mode	Trigger Source	Description
Continuous	ADC_StartConvert() function	Once started, Scanning SAR ADC runs continuously until stopped by the ADC_StopConvert() function.
	soc terminal	SAR ADC runs continuously while the signal connected to the soc terminal is high. The ADC completes the last scan that began before the signal transitioned low. Recommended for connection to level sensitive triggering.
Single shot	ADC_StartConvert() function	Scanning SAR ADC takes one scan per API call. valid firmware or hardware trigger.
	soc terminal	Scanning SAR ADC takes one scan per API hardware rising edge trigger. Recommended for edge sensitive triggering.

Use soc terminal

The Scanning SAR ADC can always be started and stopped in firmware with the ADC_StartConvert() and ADC_StopConvert() functions.

If this box is checked, hardware triggering via the start-of-conversion (soc) terminal on the Component is enabled. The soc terminal is created on the Component symbol by checking the “Use signal on soc terminal” on the Scan sub tab.

With this hardware triggering enabled, in single-shot mode a single complete scan of the Scanning SAR ADC is triggered by a positive-going edge applied to the soc terminal. In continuous mode, the ADC takes scans back-to-back if a ‘1’ level is applied to the soc terminal.

Enabling hardware triggering does not suppress the firmware triggering function. Exercise caution in interpreting data sets resulting from a combination of both forms of triggering, since the trigger source is not reflected in the output data.

Application Programming Interface

By default, PSoC Creator assigns the instance name "ADC _1" to the first instance of a Component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "ADC".

The Component is designed to use API from the SAR Peripheral Driver Library (PDL) module. Launch the PDL API Reference Manual by right clicking on the Component instance in the schematic and selecting **Open PDL Documentation…**

Note Do not use the ADC_Stop() API to halt conversions. Instead use the ADC_StopConvert() API. If you use the ADC_Stop() API to halt conversions then later use the ADC_Start() and ADC_StartConvert() APIs to resume conversions, the first channel of the scan may be corrupt. The ADC_StopConvert() API will enable the Scanning SAR ADC to complete the current scan of channels. After the channel scan is complete, the Scanning SAR ADC will stop all conversions, which can be detected by the use of an ISR or the ADC_IsEndConversion() flag.

Note that no explicit functions for saving and loading the hardware state are provided. Everything needed to set up the SAR hardware is provided in the main API functions.

Functions

Function	Description
ADC_Start()	Performs all required initialization for this Component and enables the power. The power will be set to the appropriate power based on the clock frequency.
ADC_StartEx()	Performs the same function as ADC_Start() as well as setting the interrupt vector to a user defined address.
ADC_Stop()	This function stops ADC conversions and puts the ADC into its lowest power mode.
ADC_SelectConfig()	Selects the predefined configuration for scanning.
ADC_StartConvert()	For continuous mode, this API starts the conversion process and it runs continuously. In a triggered mode, this routine triggers every conversion.
ADC_StopConvert()	Forces the ADC to stop conversions. If a conversion is currently executing, that conversion will complete, but no further conversions will occur.
ADC_SetConvertMode()	Sets the conversion mode to either Single-Shot or continuous.
ADC_IRQ_Enable()	Enables interrupts to occur at the end of a conversion. Global interrupts must also be enabled for the ADC interrupts to occur.
ADC_IRQ_Disable()	Disables interrupts at the end of a conversion.
ADC_SetEosMask()	This function sets or clears the End of Scan (EOS) interrupt mask bit.
ADC_SetChanMask()	Sets enable/disable mask for all channels.
ADC_IsEndConversion()	Immediately returns the status of the conversion or does not return (blocking) until the conversion completes, depending on the retMode parameter.

Function	Description
ADC_GetResult16()	Gets the data available in the SAR result register, returns 16-bit
ADC_GetResult32()	Gets the data available in the SAR result register, returns 32-bit
ADC_SetLowLimit()	This parameter sets the low limit for a limit compare.
ADC_SetHighLimit()	This parameter sets the high limit for a limit compare.
ADC_SetLimitMask()	Sets which channels may cause a limit condition interrupt.
ADC_SetSatMask()	Sets which channels may cause a saturation event interrupt.
ADC_SetOffset()	Sets the offset of the ADC channel.
ADC_SetGain()	Sets the gain in counts per 10 volt for the ADC channel.
ADC_CountsTo_Volts()	Converts the ADC output to volts as a floating point number.
ADC_CountsTo_mVolts()	Converts the ADC output to millivolts.
ADC_CountsTo_uVolts()	Converts the ADC output to microvolts.

void ADC_Start(void)

Description: Performs all required initialization for this Component and enables the power. The power will be set to the appropriate power based on the clock frequency.

void ADC_StartEx(cyisaddress address)

Description: This function starts the ADC and sets the Interrupt Service Routine to the provided address using the ADC_IRQ_StartEx() function. Refer to the Interrupt Component datasheet for more information on the ADC_IRQ_StartEx() function.

Parameters: address: This is the address of a user defined function for the ISR .

void ADC_Stop(void)

Description: This function stops ADC conversions and puts the ADC into its lowest power mode.

Side Effects: Don’t use the ADC_Stop() API to halt conversions. Instead use the ADC_StopConvert() API. If you use the ADC_Stop() API to halt conversions then later use the ADC_Start() and ADC_StartConvert() APIs to resume conversions, the first channel of the scan may be corrupt. The StopConvert() API will enable the Scanning SAR ADC to complete the current scan of channels. After the channel scan is complete, the Scanning SAR ADC will stop all conversions, which can be detected by the use of an ISR or the ADC_IsEndConversion() flag.

void ADC_SelectConfig(uint32 config, uint32 restart)

Description: Selects the predefined configuration for scanning.

Parameters: config: Number of configuration in the ADC.

restart: Set to 1u if the ADC should be restarted after selecting the configuration.

void ADC_StartConvert(void)

Description: In continuous mode, this API starts the conversion process and it runs continuously.

In Single Shot mode, the function triggers a single scan and every scan requires a call of this function. The mode is set with the Sample Mode parameter in the customizer. The customizer setting can be overridden at run time with the ADC_SetConvertMode() function.

void ADC_StopConvert(void)

Description: Forces the ADC to stop conversions. If a conversion is currently executing, that conversion will complete, but no further conversions will occur.

void ADC_SetConvertMode(cy_en_sar_start_convert_sel_t mode)

Description: Sets the conversion mode to either Single-Shot or continuous. This function overrides the settings applied in the customizer. Changing configurations will restore the values set in the customizer.

Parameters: mode: Sets the conversion mode. See table below for details.

Options	Description
CY_SAR_START_CONVERT_SINGLE_SHOT	Calling the ADC_StartConvert() function after setting mode this will trigger a single scan. Sets the SOC signal to be edge sensitive, each edge will trigger a single scan.
CY_SAR_START_CONVERT_CONTINUOUS	Calling the ADC_StartConvert() function after setting this mode trigger continuous scanning. This mode sets the SOC signal to be level sensitive. The ADC will continuously scan while soc is active.

void ADC_IRQ_Enable(void)

Description: Enables interrupts to occur at the end of a conversion. Global interrupts must also be enabled for the ADC interrupts to occur.

Parameters: None

Return Value: None

Side Effects: None

void ADC_IRQ_Disable(void)

Description: Disables end of conversion interrupts.

void ADC_SetEosMask(uint32_t mask)

Description: Sets of clears the End of Scan (EOS) interrupt mask.

Parameters: mask: 1 to set the mask, 0 to clear the mask.

Side Effects: All other bits in the INTR register are cleared by this function.

void ADC_SetChanMask(uint32_t mask)

Description: Sets enable/disable mask for all channels.

Parameters: mask: 1 to set the mask, 0 to clear the mask.

Side Effects: Enabling or disabling a channel disrupts the scheduled timing and changes the sample rate.

uint32_t ADC_IsEndConversion(cy_en_sar_return_mode_t retMode)

Description: Immediately returns the status of the conversion or does not return (blocking) until the conversion completes, depending on the retMode parameter.

Parameters: retMode: Check conversion return mode. See the following table for options.

Options	Description
CY_SAR_RETURN_STATUS	Immediately returns the conversion status for sequential channels. If the value returned is zero, the conversion is not complete, and this function should be retried until a nonzero result is returned.
CY_SAR_WAIT_FOR_RESULT	Does not return a result until the ADC conversion of all sequential channels is complete.

Return Value: uint32_t: If a nonzero value is returned, the last conversion is complete. If the returned value is zero, the ADC is still calculating the last result.

Side Effects: This function reads the end of conversion status, and clears it afterward.

int16_t ADC_GetResult16(uint32_t chan)

Description: Gets the data available in the channel result data register.

Parameters: chan: The ADC channel to read the result from. The first channel is 0 and the injection channel if enabled is the number of valid channels.

Return Value: Returns converted data as a signed 16-bit integer

int32_t ADC_GetResult32(uint32_t chan)

Description: Gets the data available in the channel result data register.

Parameters: chan: The ADC channel to read the result from. The first channel is 0 and the injection channel if enabled is the number of valid channels.

Return Value: Returns converted data as a signed 32-bit integer

void ADC_SetLowLimit(uint32_t lowLimit)

Description: Sets the low limit parameter for a limit condition.

Parameters: lowLimit: The low limit for a limit condition.

void ADC_SetHighLimit(uint32_t highLimit)

Description: Sets the high limit parameter for a limit condition.

Parameters: highLimit: The high limit for a limit condition.

void ADC_SetLimitMask(uint32_t mask)

Description: Sets the channel limit condition mask.

Parameters: mask: Sets which channels that may cause a limit condition interrupt. Setting bits for channels that do not exist will have no effect. For example, if only 6 channels were enabled, setting a mask of 0x0103 would only enable the last two channels (0 and 1).

void ADC_SetSatMask(uint32_t mask)

Description: Sets the channel saturation event mask.

Parameters: mask: Sets which channels that may cause a saturation event interrupt. Setting bits for channels that do not exist will have no effect. For example, if only 8 channels were enabled, setting a mask of 0x01C0 would only enable two channels (6 and 7).

void ADC_SetOffset(uint32_t chan, int16_t offset)

Description: Sets the ADC offset that is used by the functions ADC_CountsTo_uVolts, ADC_CountsTo_mVolts, and ADC_CountsTo_Volts.

Offset is applied to counts before unit scaling and gain. All CountsTo_[mV, uV, V]olts() functions use the following equation:

V = (Counts/AvgDivider - Offset)*TEN_VOLT/Gain

See CountsToVolts() for more about this formula.

To set channel 0's offset based on known V_offset_mV, use:

ADC_SetOffset(0uL, -1 * V_offset_mV * (1uL << (Resolution - 1)) / V_ref_mV);

Parameters: chan: ADC channel number.

offset: This value is a measured value when the inputs are shorted or connected to the same input voltage.

void ADC_SetGain(uint32_t chan, int32_t adcGain)

Description: Sets the ADC gain in counts per 10 volt for the voltage conversion functions below. This value is set by default by the reference and input range settings. Gain is applied after offset and unit scaling. All CountsTo_[mV, uV, V]olts()

functions use the following equation:

V = (Counts/AvgDivider - Offset)*TEN_VOLT/Gain

See CountsToVolts() for more about this formula.

To set channel 0's gain based on known V_ref_mV, use:

ADC_SetGain(0uL, 10000 * (1uL << (Resolution - 1)) / V_ref_mV);

Parameters: chan: ADC channel number.

adcGain: ADC gain in counts per 10 volt.

float32_t ADC_CountsTo_Volts(uint32_t chan, int16_t adcCounts)

Description: Converts the ADC output Volts as a float32. For example, if the ADC measured 0.534 volts, the return value would be 0.534.

The calculation of voltage depends on the contents of Cy_SAR_offset[], Cy_SAR_countsPer10Volt[], and other parameters. The equation used is:

V = (Counts/AvgDivider - Offset)*TEN_VOLT/Gain

-Counts = Raw Counts from SAR register

-AvgDivider = divider based on averaging mode

-Sequential, Sum: AvgDivider = number averaged

Note: The divider should be a maximum of 16. If using more averages, pre-scale Counts by (number averaged / 16)

-Interleaved, Sum: AvgDivider = number averaged

-Sequential, Fixed: AvgDivider = 1

-Offset = Cy_SAR_offset[]

-TEN_VOLT = 10V constant and unit scalar.

-Gain = Cy_SAR_countsPer10Volt[]

When the Vref is based on Vdda, the value used for Vdda is set for the project in the System tab of the DWR.

Parameters: chan: ADC channel number.

adcCounts: Result from the ADC conversion

Return Value: Result in Volts

int16_t ADC_CountsTo_mVolts(uint32_t chan, int16_t adcCounts)

Description: Converts the ADC output to millivolts as an int16. For example, if the ADC measured 0.534 volts, the return value would be 534.

The calculation of voltage depends on the contents of Cy_SAR_offset[], Cy_SAR_countsPer10Volt[], and other parameters. The equation used is:

V = (Counts/AvgDivider - Offset)*TEN_VOLT/Gain

-Counts = Raw Counts from SAR register

-AvgDivider = divider based on averaging mode

-Sequential, Sum: AvgDivider = number averaged

Note: The divider should be a maximum of 16. If using more averages, pre-scale Counts by (number averaged / 16)

-Interleaved, Sum: AvgDivider = number averaged

-Sequential, Fixed: AvgDivider = 1

-Offset = Cy_SAR_offset[]

-TEN_VOLT = 10V constant and unit scalar.

-Gain = Cy_SAR_countsPer10Volt[]

When the Vref is based on Vdda, the value used for Vdda is set for the project in the System tab of the DWR.

Parameters: chan: ADC channel number.

adcCounts: Result from the ADC conversion.

Return Value: Result in mV.

int32_t ADC_CountsTo_uVolts(uint32_t chan, int16_t adcCounts)

Description: Converts the ADC output to microvolts as an int32. For example, if the ADC measured 0.534 volts, the return value would be 534000.

The calculation of voltage depends on the contents of Cy_SAR_offset[], Cy_SAR_countsPer10Volt[], and other parameters. The equation used is:

V = (Counts/AvgDivider - Offset)*TEN_VOLT/Gain

-Counts = Raw Counts from SAR register

-AvgDivider = divider based on averaging mode

-Sequential, Sum: AvgDivider = number averaged

Note: The divider should be a maximum of 16. If using more averages, pre-scale Counts by (number averaged / 16)

-Interleaved, Sum: AvgDivider = number averaged

-Sequential, Fixed: AvgDivider = 1

-Offset = Cy_SAR_offset[]

-TEN_VOLT = 10V constant and unit scalar.

-Gain = Cy_SAR_countsPer10Volt[]

When the Vref is based on Vdda, the value used for Vdda is set for the project in the System tab of the DWR

Parameters: chan: ADC channel number.

adcCounts: Result from the ADC conversion

Return Value: Result in µV

Global Variables

Function	Description
ADC_initVar	The initVar variable is used to indicate initial configuration of this Component. The variable is initialized to zero and set to 1 the first time ADC_Start() is called. This allows for Component initialization without reinitialization in all subsequent calls to the ADC_Start() routine. If reinitialization of the Component is required, then the ADC_Init() function can be called before the ADC_Start() or ADC_Enable() functions.
ADC_selected	The selected variable is used to keep track of whether ADC_SelectConfig or ADC_Init has been called at least once. It is tested during initialization to determine whether to run the single-configuration initializing code.
Cy_SAR_offset[]	This array calibrates the offset for each channel. The first time Start() is called, the offset array's entries are initialized to 0, except for channels which are Single-Ended, Signed, and have Vneg=Vref, for which it is set to -2^(Resolution-1)/Vref(mV). It can be modified using ADC_SetOffset(). The array is used by the ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), and ADC_CountsTo_uVolts() functions.

Function	Description
Cy_SAR_countsPer10Volt[]	This array is used to calibrate the gain for each channel. It is calculated the first time ADC_Start() is called. The value depends on channel resolution and voltage reference. It can be changed using ADC_SetGain(). This array affects the ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), and ADC_CountsTo_uVolts() functions by supplying the correct conversion between ADC counts and the applied input voltage.

Usable Constants

Function	Description
ADC_TOTAL_CHANNELS_NUM	This constant represents the amount of input channels available for scanning across all configs.

Registering the Deep Sleep callback

The Deep Sleep callback ensures that the ADC's settings are saved and the ADC is shut down properly before going to Deep Sleep mode and the same settings are restored when transitioning out of Deep Sleep mode. The callback must by registered by calling the SysPM function `Cy_SysPm_RegisterCallback` and passing it the address of the component's Deep Sleep callback structure as follows:

```
Cy_SysPm_RegisterCallback(&ADC_DeepSleepCallbackStruct);


/* Put device into Deep Sleep mode. */
Cy_SysPm_DeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);
```

In the above code snippet, "ADC" is the name of the Component instance, similar to the API descriptions.

Sample Firmware Source Code

PSoC Creator provides numerous code examples that include schematics and example code in the Find Example Project dialog. For Component-specific examples, open the dialog from the Component Catalog or an instance of the Component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Code Example" topic in the PSoC Creator Help for more information.

Interrupt Service Routine

Interrupts must be enabled for in the Design-Wide Resources Interrupt Editor for the intended core.

The Scanning SAR ADC interrupt can be triggered by the following conditions:

- End of Scan – Scanning of all channels complete.
- Limit – High/Low limit compare, enabled on a channel by channel basis.
- Saturate – Saturation condition, enabled on a channel by channel basis.

The Scanning SAR ADC contains a Macro Callback for the interrupt service routine. You can use the Macro Callback by adding the callback definition and prototype to the header file named *cyapicallbacks.h* as below. The ISR can then be written in any user file.

```
#define ADC_ISR_CALLBACK
void ADC_ISR_Callback( void );
```

The following is an example of code that uses an interrupt to capture data. This interrupt is triggered by the default End of Scan condition.

```
#include "project.h"

volatile int16_t result;
volatile uint8_t dataReady;

void ADC_ISR_Callback(void)
{
    dataReady = 1;
    result = ADC_GetResult16(0);
}

int main(void)
{
    int16_t newReading = 0;

    __enable_irq(); /* Enable global interrupts. */

    ADC_Start();
    ADC_IRQ_Enable();
    ADC_StartConvert();

    for(;;)
    {
        if(dataReady != 0)
        {
            dataReady = 0;
            newReading = result;
            /* Process newReading */
        }
    }
}
```

You may use an alternative Interrupt Service Routine instead of using the provided callback. To do this, call the ADC_1_StartEx function instead of ADC_Start, passing it the name of your ISR. However, you must clear the interrupt in this routine. The provided callback clears the interrupt.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Refer to PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) for information on MISRA compliance and deviations for files generated by PSoC Creator.

The Scanning SAR ADC Component has no MISRA deviations. The SAR PDL driver does have MISRA violations. Refer to the PDL documentation for more information about the SAR PDL driver.

This Component has the following embedded Components: Interrupt, Clock. Refer to the corresponding Component datasheet for information on their MISRA compliance and specific deviations.

API Memory Usage

The Component memory usage varies significantly, depending on the compiler, device, number of APIs used, and Component configuration. This table illustrates the memory usage for all APIs available in the default Component configuration.

The measurements were done with the associated compiler configured with optimization set for size. For a specific design analyze the map file generated by the compiler to determine the memory usage.

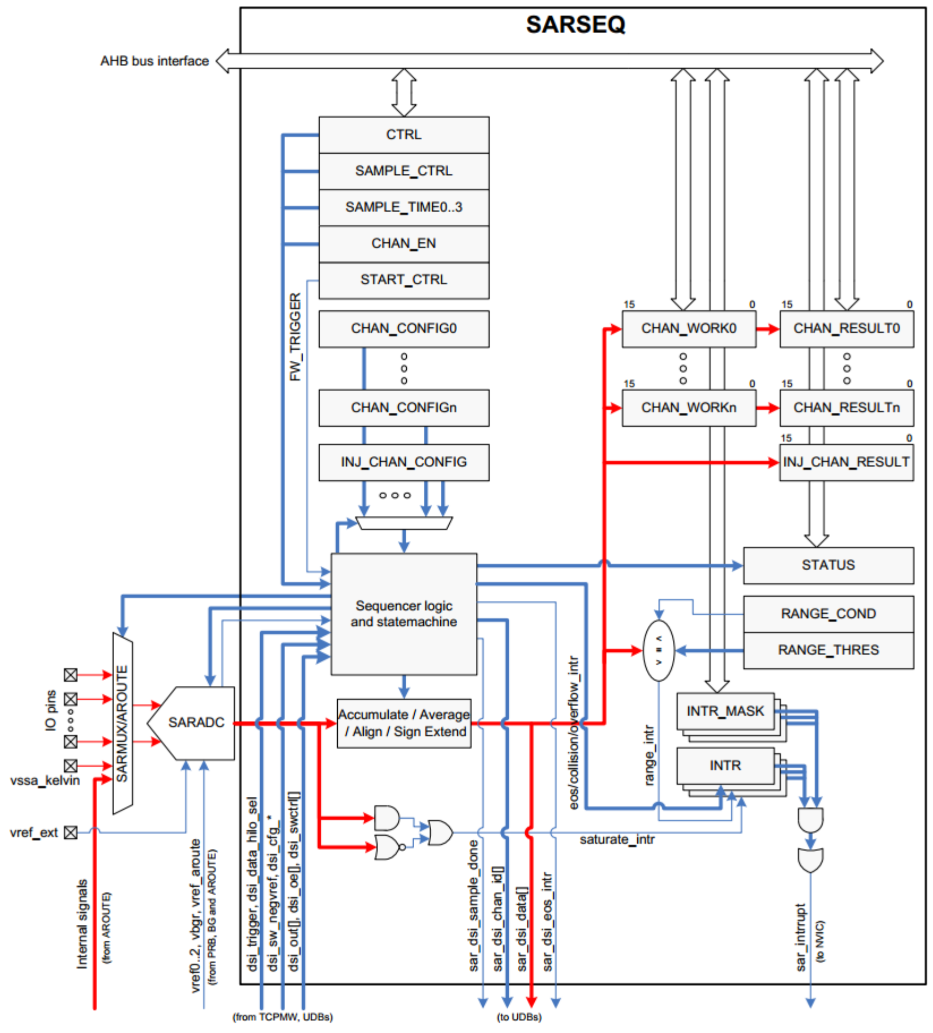
Configuration	**PSoC 6**	
	Flash Bytes	**SRAM Bytes**
Default, No CountsTo_Volts()	1416	104
Default, CountsTo_Volts()	1912	104
4 Config, No CountsTo_Volts()	1848	104
4 Config, CountsTo_Volts()	2344	104

Functional Description

The Scanning SAR ADC Component is implemented on a hardware block that contains the following elements:

- SAR ADC
  - SARMUX
  - SARADC core
  - SARREF
  - SARSEQ

SAR ADC



The SARADC core is a fast 12-bit ADC with SAR architecture. Preceding the SARADC is the SARMUX, which can route a combination of external pins and internal signals to inputs of the SARADC core. SARREF is a buffer used for multiple reference voltage selection. The SARSEQ sequencer block controls the SARMUX and the SARADC and does an automatic scan on all enabled channels as well as post-processing, such as averaging the output data.

Each channel has 16-bit conversion-result storage registers. At the end of the scan, a maskable interrupt is asserted. The sequencer also flags overflow and saturation errors that can be configured to assert an interrupt.

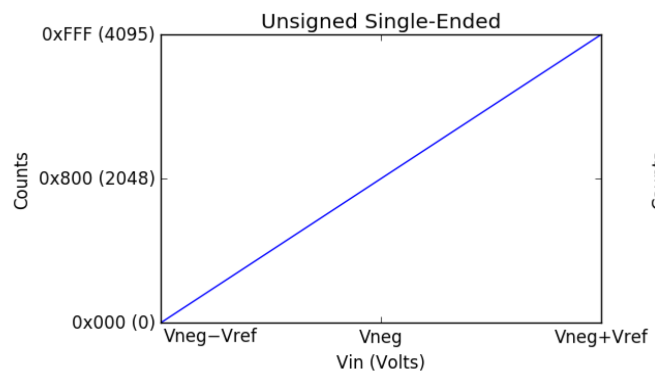
Input Modes and Signedness

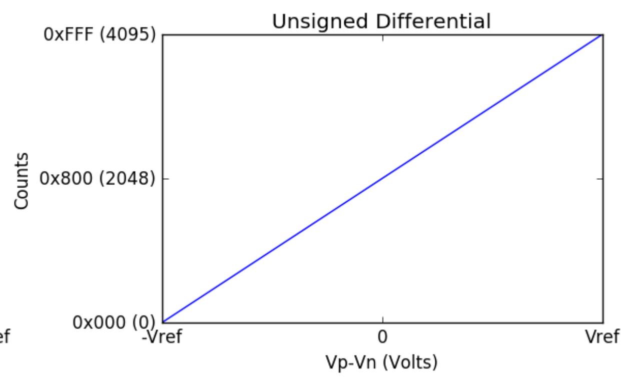
The input mode (S/E or Differential) determines the range of input voltages, and the signedness determines the digital codes to which the input range corresponds.

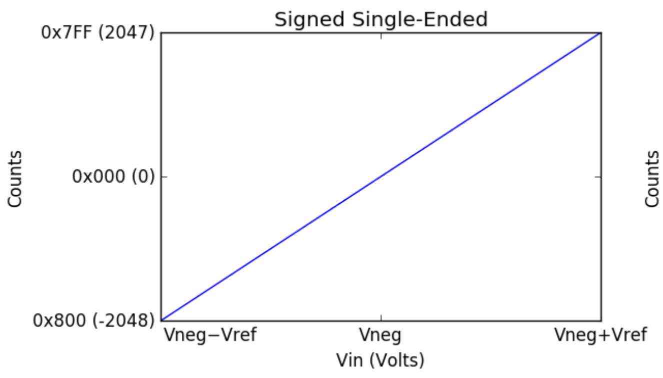
The smallest voltage in the range always corresponds to the lowest code.

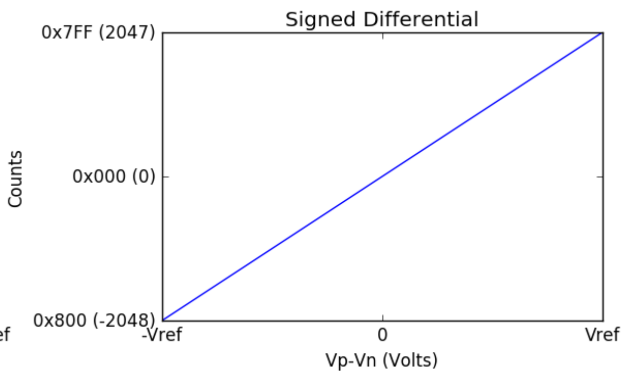
The diagrams in this section show the various input ranges and their corresponding codes, represented in both 12-bit hexadecimal and decimal.

Note It is recommended to use settings with intuitive results, such as S/E with Vneg = Vref and such as Signed Differential.









DMA Support

The DMA Component can be used to transfer data from the Component registers to RAM or another Component.

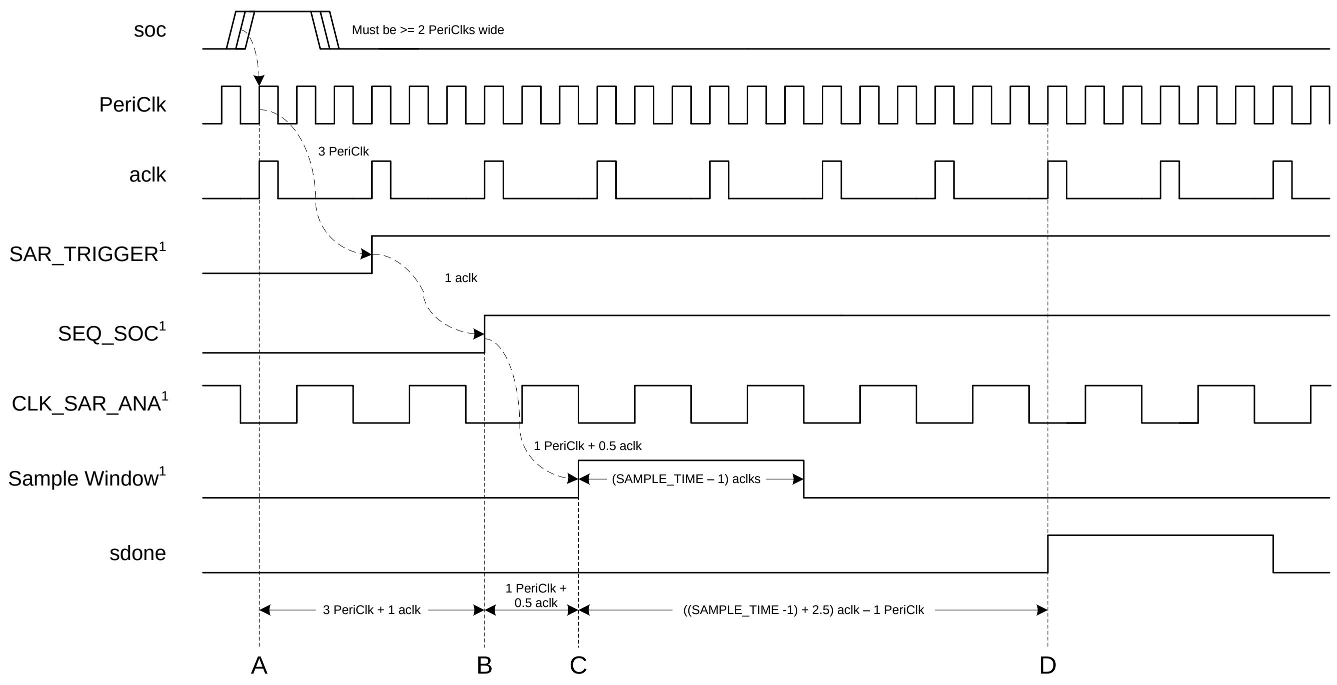
Name of DMA Source	Width	Direction	DMA Req Signal	DMA Trigger Type	Description
(ADC_SAR_CHAN_RESULT_PTR + (***X*** << 2u)) * or ADC_SAR_CHAN***X***_RESULT_PTR *	32	Source	eoc	Pulse	Channel result data register. This 32-bit register contains 16-bit ADC results. **Note** This register also contains status bits in the upper three bits of the register.

* where ***X*** – is a channel number. The first channel is 0.

Note The Component has a DMA bus interface that supports 32-bit (word) transfers only. If the data element size used for DMA transfer is less than a word, set the DMA descriptor with the correct width; for example, data element size is halfword (2 bytes). The Component register is used as Source; make sure the DMA descriptor is configured as "Word to Halfword."

Sample Timing

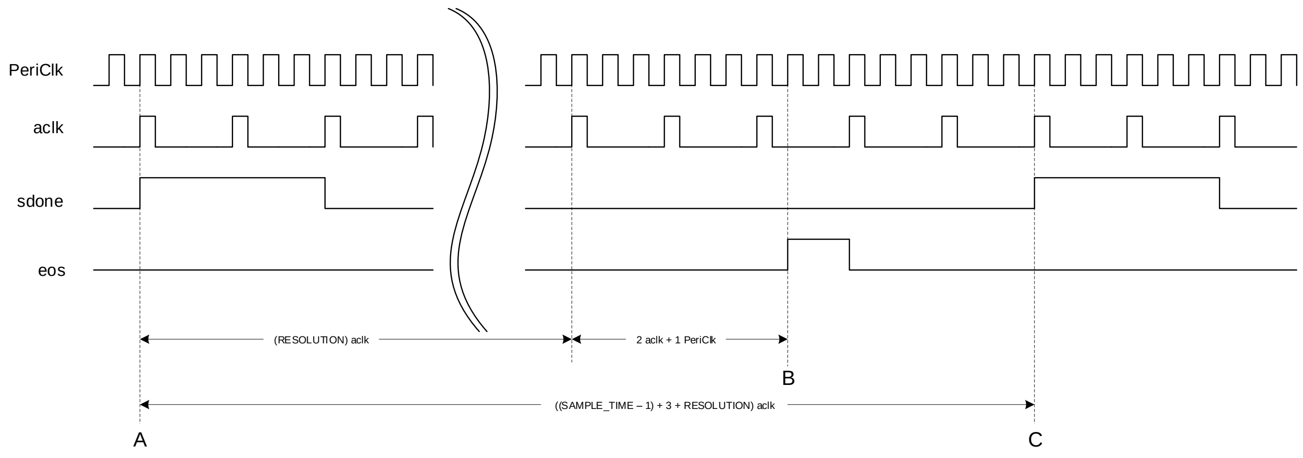
The following diagram shows the timing from an external trigger on the soc input to the sdone signal going high. This diagram illustrates where the ADC is sampling with respect to these external inputs. The aclk clock is PeriClk divided by three for this example. The internal CLK_SAR_ANA is a 50% duty cycle version of aclk that is delayed by one PeriClk. Key points of this diagram are explained below.



[1]Internal hardware signals. Not available for external connections.

A) The soc input is captured on the rising edge of PeriClk. The conversion will begin 3 PeriClk cycles and one aclk cycle later. If the soc signal is synchronous to the PeriClk, the time from A to B can be reduced by two PeriClk cycles by clearing the DSI_SYNC_CONFIG bit in the SAR_CTRL register.
B) Conversion begins. The conversion will complete (SAMPLE_TIME + 14) aclk cycles after this point.
C) Sample window opens. The SAR starts sampling the signal here.
D) The sdone output is asserted.

The following diagram shows the timing from the sdone signal to the eos signal for a continuously scanned single channel ADC.



A) Sampling of Channel 1 is complete.
B) The eos output is asserted.
C) Channel 1 is sampled again.

The time between consecutive eos rising edges is governed by the $Achieved\ Scan\ Time$ from the Timing section.

Registers

Channel result data registers

This 32-bit register contains 16-bit ADC results from channel 0 along with 3 status bits that describe the results correctness.

ADC_SAR_CHAN_RESULT_REG

Bits	Name	Description
15:0	Data	SAR conversion result of the first channel. The data is copied here from the work field after all enabled channels in this scan have been sampled.
29	ADC_SATURATE_INTR_MIR	Mirror bit of corresponding bit in ADC_SAR_SATURATE_INTR_REG register
30	ADC_RANGE_INTR_MIR	Mirror bit of corresponding bit in ADC_SAR_RANGE_INTR_REG register
31	ADC_CHAN_RESULT_VALID_MIR	Mirror bit of corresponding bit in ADC_SAR_CHAN_RESULT_VALID_REG register

Result registers for the remaining channels are located sequentially in the memory. Direct defines for each channel are provided: ADC_SAR_CHAN*X*_RESULT_REG, were *X* is the channel number from 0 to 15.

Interrupt request registers

Each of the interrupts described in this section has an interrupt mask in the ADC_SAR_INTR_MASK_REG register. By making the interrupt mask low, the corresponding interrupt source is ignored. The SAR interrupt is raised any time the intersection (logic AND) of the interrupt flags in ADC_SAR_INTR_REG registers and the corresponding interrupt masks in ADC_SAR_INTR_MASK_REG register is non zero.

When servicing an interrupt, the interrupt service routine (ISR) clears the interrupt source by writing a ‘1’ to the interrupt bit after picking up the related data.

For firmware convenience, the intersection (logic AND) of the interrupt flags and the interrupt masks are also made available in the SADC_SAR_INTR_MASKED_REG register.

ADC_SAR_INTR_REG

Bits	Name	Description
0	ADC_EOS_MASK*	End Of Scan Interrupt: hardware sets this interrupt after completing a scan of all the enabled channels. Write with '1' to clear bit after picking up the data from the ADC_SAR_CHAN_RESULT_REG register.
1	ADC_OVERFLOW_MASK	Overflow Interrupt: hardware sets this interrupt when it sets a new ADC_EOS_MASK while that bit was not yet cleared by the firmware. Write with '1' to clear bit.
2	ADC_FW_COLLISION_MASK	Firmware Collision Interrupt: hardware sets this interrupt when in **Hardware trigger** sample mode firmware triggers the conversion using ADC_StartConvert() API while the SAR is BUSY. Raising this interrupt is delayed to when the scan caused by the ADC_StartConvert() API has been completed, i.e. not when the preceding scan with which this trigger collided is completed. When this interrupt is set it implies that the channels were sampled later than was intended (jitter). Write with '1' to clear bit.
3	ADC_DSI_COLLISION_MASK	DSI Collision Interrupt: hardware sets this interrupt when the hardware SOC trigger signal is asserted while the SAR is BUSY. Raising this interrupt is delayed to when the scan caused by the hardware SOC trigger has been completed, i.e. not when the preceding scan with which this trigger collided is completed. When this interrupt is set it implies that the channels were sampled later than was intended (jitter). Write with '1' to clear bit.

These bits are enabled by the Component by default in ADC_SAR_INTR_MASK_REG register and generate an interrupt.

ADC_SAR_SATURATE_INTR_REG

Bits	Name	Description
15:0	SATURATE_INTR	Saturate interrupt request register. Hardware sets saturate interrupt for each channel if a conversion result (before averaging) of that channel is either 0x000 or 0xFFF; this is an indication that the ADC likely saturated. Write with '1' to clear bit.

ADC_SAR_SATURATE_INTR_MASK_REG

Bits	Name	Description
15:0	SATURATE_MASK	Saturate interrupt mask register. It is set by default according to selection of the **Saturation** parameter. Use ADC_SetSatMask() API to change this mask register.

ADC_SAR_SATURATE_INTR_MASKED_REG

Bits	Name	Description
15:0	SATURATE_MASKED	Saturate interrupt masked request register. If the value is not zero then the SAR interrupt is raised. When read, this register reflects a bitwise AND between the saturate interrupt request and mask registers.

ADC_SAR_RANGE_INTR_REG

Bits	Name	Description
15:0	RANGE_INTR	Range detect interrupt request register. Hardware sets range detect interrupt for each channel if the conversion result (after averaging) of that channel met the condition specified by the **Compare Mode** parameter. Write with '1' to clear bit.

ADC_SAR_RANGE_INTR_MASK_REG

Bits	Name	Description
15:0	RANGE_MASK	Range detect interrupt mask register. It is set by default according to selection of the **Limit detect** parameter. Use ADC_SetLimitMask() API to change this mask register.

ADC_SAR_RANGE_INTR_MASKED_REG

Bits	Name	Description
15:0	RANGE_MASKED	Range interrupt masked request register. If the value is not zero then the SAR interrupt is raised. When read, this register reflects a bitwise AND between the range detect interrupt request and mask registers.

Resources

The Scanning SAR ADC is implemented as a fixed-function block. The Component also uses one Interrupt.

DC and AC Electrical Characteristics (Preliminary)

Note Final characterization data for PSoC 6 devices is not available at this time. Once the data is available, the Component datasheet will be updated on the Cypress web site.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.10.a	Corrected the acquisition time calculation.	The acquisition time calculation was short by one half clock cycle. This could result in an Achieved Acquisition Time less than the Minimum Acquisition Time, reducing the performance of the ADC.
2.10	Moved Sample Mode parameters to the Common tab.	Sample Mode is now a global control for all configurations, allowing PSoC Creator to check for consistency between signals connected to the soc terminal and the trigger type.
2.0.b	Minor datasheet edits.	
2.0.a	• Update scan rate equations in the Timing section. • Add a Sample Timing section. • Removed "Prototype" tag.	• Timing equations were incorrect. • Provide timing diagrams for common use cases. • Production qualified.
2.0	Added support for PSoC 6 devices.	Previous versions of this Component were not applicable to PSoC 6.