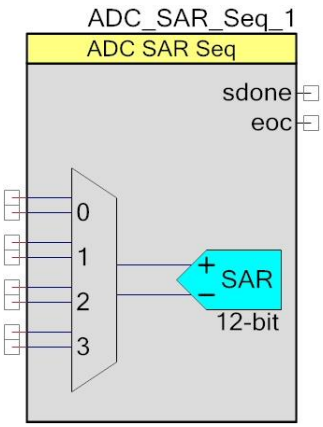
PSoC 4 Sequencing Successive Approximation ADC (ADC_SAR_Seq)

2.30



Features

- Selectable 8-, 10-, and 12-bit resolutions
- Sample rates of up to 1 Msps with 12-bit resolution
- Supports both Single Ended and Differential inputs
- Different ranges of inputs with multiple reference options
- Scan up to sixteen channels automatically, or just a single input
- Allows an “injection” channel to be added to the scan sequence with firmware control at runtime
- Hardware averaging support

General Description

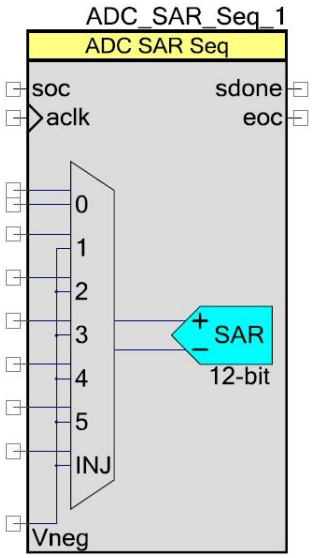
The Sequencing SAR ADC component gives you the ability to configure and use the different operational modes of the SAR ADC on PSoC 4. You have schematic- and firmware-level support for seamless use of the Sequencing SAR ADC in PSoC Creator designs and projects. You also have the ability to configure up to sixteen analog channels (depending on the device) that are automatically scanned with the results placed in individual result registers. An optional “Injection channel” may also be enabled by firmware to occasionally scan a signal that does not need to be scanned at the same rate as other channels.

When to Use a Sequencing SAR ADC

The Sequencing SAR ADC is the component used to access the ADC functionality in PSoC 4. The sequencing and muxing capabilities are integral parts of the SAR hardware. The component can be used in high sample rate systems where you need to sample multiple channels without CPU intervention until all channels are sampled. It can also be used in low sample rate designs or in designs that have just a single channel to sample.

Input/Output Connections

This section describes the various input and output connections for the Sequencing SAR ADC. An asterisk (*) in the list of I/Os states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.



+Input – Analog

This input is the positive analog signal input to the ADC SAR Seq. The conversion result is a function of the +Input signal minus the voltage reference. The voltage reference is either the –Input signal, Vneg, Vref, or Vss.

There are always the same number of analog signal input terminals as there are sequenced channels selected, including the injection channel.

Analog inputs are low impedance. This is solely driven by the need to settle the bus within the short sampling aperture of the Sequencing SAR ADC. Use the operational amplifier as a buffer for the high impedance signals.

–Input – Analog *

The number of negative input terminals varies depending on the number of channels and how many single ended channels are present. When a channel is selected as single ended, all the negative input signals are combined to form a single net internally.

Vneg – Analog *

This is a common negative input reference. This terminal is present only if one or more analog channels are defined as a single ended input and **Single ended negative input** parameter is set to **External**.

soc – Input *

This input is used for the Start of conversion or scan. It is visible if you set the **Sample mode** parameter to **Hardware trigger**. A rising edge on this input starts an ADC conversion. The first **soc** rising edge should be generated at least 10us after the component is started to guarantee reference and pump voltage stability. You can connect the output of a TCPWM component to this input. It can also be connected to any GPIO pin or a UDB.

This input is hidden if you set the **Sample mode** parameter to **Free Running**.

aclk – Input *

This is the analog ADC clock signal. You can add this optional pin if you set the **Clock source** parameter to **External**; otherwise, the pin is hidden. This clock determines the conversion rate as a function of conversion method, number on sequenced channels and their parameters.

sdone – Output

This signal goes high for one ADC clock to signal that the ADC has sampled the current input channel and that the input mux may be switched. The input multiplexer selection can be changed after sampling is complete even though the conversion has not yet completed.

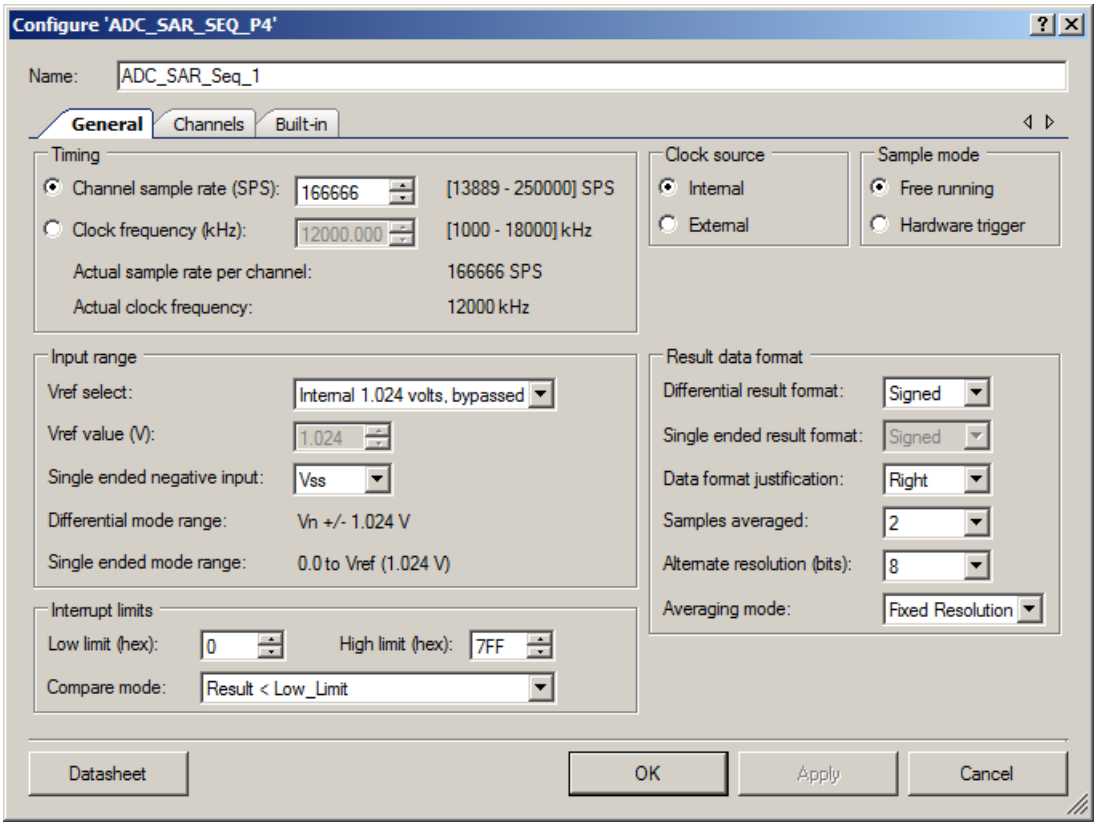
eoc – Output

A rising edge on the end of conversion (eoc) output means that one conversion cycle is complete. At this moment, conversion results for all enabled channels are ready to be read from the registers. Internally, it is used for the component interrupt. You may connect your own interrupt or route the signal to control additional logic.

Component Parameters

Drag a Sequencing SAR ADC onto your design and double click it to open the Configure dialog.

General Tab



The Sequencing SAR ADC has the following parameters. The option shown in bold is the default.

Sample rate

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

Clock frequency

When selected, enter the desired clock rate. This parameter only applies when an internal clock source is selected. The clock frequency can be anywhere between 1 MHz and 18 MHz (14.508 MHz in CY8C41). If the clock does not fall within these limits, PSoC Creator generates an error during the build process. The actual clock rate may differ based on the available source clock speed and the resulting clock, based on an integer divide of the source clock. The read-only field below this field displays the effective sample rate based on the generated nominal clock frequency taken from the Design-Wide Resources (DWR) Clock Editor.

If the sample rate is fixed, the following parameters will alter the ADC clock frequency parameter:

- Sequenced Channels
- Averaging Enabled ("AVG" check box per channel, averaged samples)
- Acquisition Time
- Alternate resolution

At high sample rates, the ADC can generate large amounts of data. The CPU clock will need to be running fast enough to process the data and the interrupt service routine overhead will need to be minimized. For example, at a conversion rate of 700,000 samples per second and a CPU clock rate of 48 MHz, there are only 48 MHz/700,000 sps = 68 CPU clock cycles per sample. See the Interrupt Service Routine section for guidance on optimizing an ISR.

Actual sample rate

This field displays an actual recalculated sample rate based on the generated nominal clock frequency taken from the DWR Clock Editor. The actual sample rate may differ from “Sample rate” field. This is a read-only field.

Actual clock frequency

This field displays an actual calculated clock frequency rate based on the generated nominal clock frequency taken from the DWR Clock Editor. The actual clock frequency may differ from the Clock frequency field. This value is based on the available source clock speed and the resulting clock, based on an integer divide of the source clock. This is a read-only field.

Clock source

This parameter allows you to select a clock that is **internal** to the component or a clock source outside the component.

Sample mode

Sample mode determines if each scan must be triggered by the SOC terminal or continuously runs after the ADC is enabled and continues until the ADC_StopConvert() API is called.

Sample Mode	Description
Free Running	ADC SAR Seq runs continuously.
Hardware trigger	A rising-edge pulse on the SOC pin starts a single conversion. The ADC_StartConvert() function also starts a single conversion.

Vref select

The **Vref Select** parameter selects the reference voltage that is used for the SAR ADC.

Reference	Allowed Clock Frequency (MHz)	Description
VDDA/2 Internal 1.024 volts	1 ~ 1.6 MHz	Uses the internal reference without a bypass capacitor.
VDDA	1 ~ 9 MHz	Uses the internal VDDA reference without a bypass capacitor.
Internal 1.024 volts, bypassed VDDA/2, bypassed	1 ~ 18 MHz	Uses the internal reference with a bypass capacitor. You must place a bypass capacitor on pin P1[7] or the dedicated Vref pin (depending on the device). [1] [2]
External Vref	1 ~ 18 MHz	Uses an external reference on pin P1[7] or the dedicated Vref pin (depending on device). [2]
Internal Vref Internal Vref, bypassed	N/A	These options are not supported by current PSoC 4 devices.

The 1.024 V 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 (1 µF)	2 ms
Startup time for reference with external capacitor (100 nF)	200 µs

Note If “non bypassed” mode is selected, Acquisition times changes from 4 to minimum value (2) to try to achieve clock frequency ranges.

---

[1] The use of an external bypass capacitor is recommended if the internal noise caused by digital switching exceeds an application's analog performance requirements. To use this option, connect an external capacitor with a value between 0.01 µF and 10 µF to port pin P1[7] or the dedicated Vref pin (depending on the device).

[2] Refer to the Pinouts section of the device datasheet for more information.

Vref value

This parameter displays the reference voltage value that is used for the SAR ADC reference. If the internal reference is selected with the **Vref select** parameter, this becomes a fixed value. If an internal reference such as VDDA or VDDA/2 is selected, the value is derived from the DWR System Editor Vdda parameter. In cases when Vref is unknown, such as using an external reference (external to the PSoC or component), the value may be entered by the user.

Single ended negative input

This parameter selects where the negative input to the SAR ADC is connected if any channels are configured for single ended operation. This choice affects the voltage range and effective resolution. The analog signals connected to the PSoC must be between V_{SSA} and V_{DDA} regardless of the input range settings.

Negative input	Description
Vss	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. When using the internal reference (1.024 V), the usable input range is 0.0 to 2.048 V.
External	This mode is configured for differential inputs. Connect common single ended negative input to Vneg terminal. When using the internal reference (1.024 V), the input range is Vneg ± 1.024 V. For example, if Vneg is connected to 2.048 V, the usable input range is 2.048 ± 1.024 V or 1.024 to 3.072 V. For systems in which both single-ended and differential signals are scanned, connect Vneg to Vssa when scanning a single-ended input. You can use an external reference to provide a wider operating range. You can calculate the usable input range with the same equation, Vneg ± Vref.

Differential mode range

This is a noneditable text box that shows the range for the differential mode inputs. It is based on the **Vref select** and **Vref value** parameters. Examples of this text box are (Vn +/- 1.024 V, Vn +/- Vdda/2, Vn +/- Vdda, etc) .

Single ended mode range

This is a noneditable text box that shows the range for the single ended mode inputs. It is based on the **Vref select**, **Vref value** and **Single ended negative input** parameters. Examples of this text box are (0.0 to Vref (1.024V), 0.0 to Vref (2.048 V), 0.0 to Vref (5 V), etc).

Differential 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.

Single ended 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.

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

Single / Differential	Signed / Unsigned	Single ended negative input	-Input	+Input	Result Register
Single	Signed	Vss	Vss	Vref Vss -noise	0x07FF 0x0000 0xFFxx
Single	Signed	External	Vneg	Vneg+Vref Vneg Vneg-Vref	0x07FF 0x0000 0xF800
Single	Unsigned	Vref	Vref	2*Vref Vref Vss	0x0FFF 0x0800 0x0000
Single	Signed	Vref	Vref	2*Vref Vref Vss	0x07FF 0x0000 0xF800
Differential	Unsigned	N/A	Vx	Vx+Vref Vx Vx-Vref	0x0FFF 0x0800 0x0000
Differential	Signed	N/A	Vx	Vx+Vref Vx Vx-Vref	0x07FF 0x0000 0xF800

For single-ended conversions with the **Single ended negative input** parameter set to **Vss**, the conversion is effectively 11-bit, because voltages below Vss are illegal on any PSoC 4 pin. Because of this, the “Unsigned” option of the **Single ended result format** parameter is not available. Noise on the **+Input** pin with a level slightly below internal Vss, produces a result that is negative.

Note that single-ended conversions with an external common alternate ground are electrically equivalent to differential mode, where the pin of each differential pair is connected to the common alternate ground. Assuming that the measured signal value (**+Input**) cannot go below that common alternate ground, then these conversions are also effectively 11-bit.

Data Format Justification

This parameter sets whether or not the output data is Left or **Right** justified in a 16-bit word. For signed values, the result is signed extended when in right justification mode. This is a global setting for all channels. This table shows all the details.

Justification	Signed / Unsigned	Resolution	Result register bits ("0" – LSB, "-" – null)															
			15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Right	Unsigned	12	-	-	-	-	11	10	9	8	7	6	5	4	3	2	1	0
		10	-	-	-	-	-	-	9	8	7	6	5	4	3	2	1	0
		8	-	-	-	-	-	-	-	-	7	6	5	4	3	2	1	0
Right	Signed	12	11	11	11	11	11	10	9	8	7	6	5	4	3	2	1	0
		10	9	9	9	9	9	9	9	8	7	6	5	4	3	2	1	0
		8	7	7	7	7	7	7	7	7	7	6	5	4	3	2	1	0
Left	N/A	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-	-	-
		10	9	8	7	6	5	4	3	2	1	0	-	-	-	-	-	-
		8	7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	-

Samples averaged

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

Alternate Resolution

This parameter sets the alternate ADC resolution to either **8** or 10 bits. Conversions for each input are selectable as either 12 bits or this alternate resolution.

Averaging Mode

This parameter sets how the averaging mode operates. If the accumulate option is selected, each ADC result is added to the sum and allowed to grow until the sum attempts to outgrow a 16 bit value, at which point it is truncated. If the **Fixed Resolution** mode is selected, the LSb is truncated so that the value does not grow beyond the maximum value for the given resolution.

Compare Mode

The Sequencing 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 Limit	Below range
Low Limit <= Result < High Limit	Inside range
High Limit <= Result	Above range
(Result < Low Limit) or (High Limit <= Result)	Outside range

Low Limit

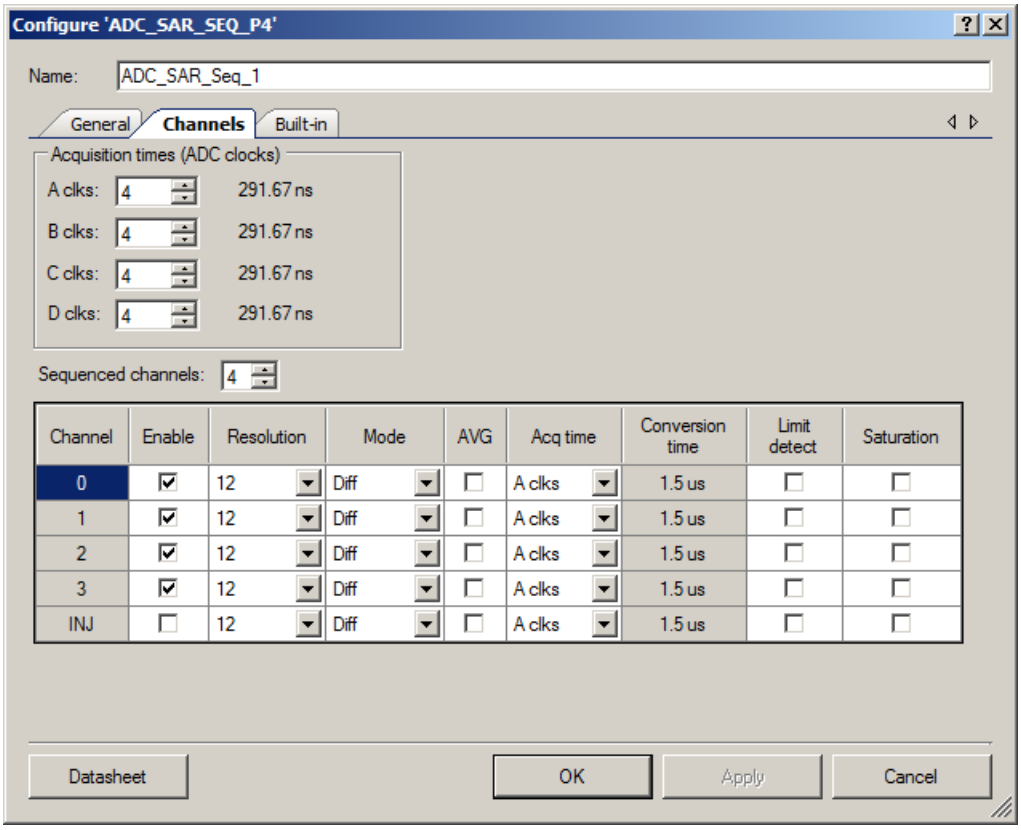
This parameter sets the low threshold for a limit compare. Default value is 0.

High Limit

This parameter sets the high threshold for a limit compare. Default value is 0x7FF.

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 conversion result.

Channels Tab



Acquisition times

This parameter sets up to four different acquisition times to configure individual channels, entered in ADC clocks. The field to the right of each selection shows the actual delay given the current clock rate. The displayed time is equal to $(N_{clk} - 0.5)*(1/F_{clk})$, where N_{clk} is the number of acquisition ADC clocks, and F_{clk} is the clock frequency. If the clock is changed for any reason (refer to Clock frequency section for details), the time displayed changes as well. The default is **4** clock periods and adjustable from 2 to 1023 clock periods.

Sequenced channels

This parameter selects how many input signals are scanned, not counting the injection channel. The maximum number of channels is either 8 or 16 depending on the device. It depends also on mode (differential or single ended) and available resources outside of the SAR. The minimum number of channels is always 1.

A set of entries is available for each parameter. The actual number of entries depends on the **Sequenced channels** parameter. The injection channel **INJ** parameter is always present. If the injection channel is not enabled, it does not appear on the symbol. The symbol shows as many channels as are selected by the **Sequenced channel** parameter even if the channel is not enabled, except for the injection channel.

Enable

For channels 0 to 7(15), it enables the channel for scanning during runtime. For the injection channel, it determines whether or not the symbol displays the input.

Resolution

This parameter selects either **12** bits or an alternative (ALT) resolution of 8 or 10 bits depending on the **Alternate resolution** parameter.

Mode

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, 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 available only for the maximum **Resolution** selected in a particular channel. Select ALT resolution for all channels to allow averaging on fewer than 12 bits resolution. Averaging is always right-aligned; therefore, the **Data Format Justification** parameter is ignored.

Acq Time

This parameter selects the acquisition time (sample and hold) during which the SAR input settles. The time is based on the SAR ADC clocks periods. These **Acquisition times** parameters are labeled **A**, B, C, and D.

Limit detect

This option allows you to enable an interrupt if any of the channels 0 through 7(15) or the injection channel trigger the limit criteria set by the **Low limit** or **High limit** and the **Compare mode** parameter.

Saturation

This option allows you to enable an interrupt from any channel where the result is saturated from either a conversion result of 0x0000 or the highest value for the given resolution.

Application Programming Interface

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

By default, PSoC Creator assigns the instance name "ADC_SAR_Seq_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".

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 StopConvert() API will enable the Sequencing SAR ADC to complete the current scan of channels. After the channel scan is complete, the Sequencing SAR ADC will stop all conversions, which can be detected by the use of an ISR or the ADC_IsEndConversion() flag.

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_Stop()	This function stops ADC conversions and puts the ADC into its lowest power mode.
ADC_StartConvert()	For free running mode, this API starts the conversion process and it runs continuously. In a triggered mode, this routine triggers every conversion.

Function	Description
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_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_IsEndConversion()	Immediately returns the status of the conversion or does not return (blocking) until the conversion completes, depending on the retMode parameter.
ADC_GetResult16()	Gets the data available in the SAR result register.
ADC_SetChanMask()	Sets the channel enable mask. Sets which channels that will be scanned.
ADC_EnableInjection()	Enables the injection channel for the next scan only.
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.
ADC_Sleep()	Stops the ADC operation and saves the configuration registers and component enable state.
ADC_Wakeup()	Restores the component enable state and configuration registers.
ADC_SaveConfig()	Save the current configuration of ADC non-retention registers.
ADC_RestoreConfig()	Restores the configuration of ADC non-retention registers.

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.

Parameters: None

Return Value: None

Side Effects: None

void ADC_Stop(void)

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

Parameters: None

Return Value: None

Side Effects: Don't use the Stop() API to halt conversions. Instead use the StopConvert() API. If you use the 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 Sequencing SAR ADC to complete the current scan of channels. After the channel scan is complete, the Sequencing SAR ADC will stop all conversions, which can be detected by the use of an ISR or the ADC_IsEndConversion() flag.

void ADC_StartConvert(void)

Description: For free running mode, this API starts the conversion process and it runs continuously.

In **Hardware trigger** mode, the function also acts as a software version of the SOC and every conversion requires a call of this function.

Parameters: None

Return Value: None

Side Effects: None

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.

Parameters: None

Return Value: None

Side Effects: None

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.

Parameters: None

Return Value: None

Side Effects: None

uint32 ADC_IsEndConversion(uint32 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
ADC_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.
ADC_WAIT_FOR_RESULT	Does not return a result until the ADC conversion of all sequential channels is complete.
ADC_RETURN_STATUS_INJ	Immediately returns the conversion status for the injection channel. If the value returned is zero, the conversion is not complete, and this function should be retried until a nonzero result is returned.
ADC_WAIT_FOR_RESULT_INJ	Does not return a result until the ADC completes injection channel conversion.

Return Value: uint8: 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 ADC_GetResult16(uint32 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

Side Effects: None.

void ADC_SetChanMask(uint32 mask)

Description: Sets the channel enable mask.

Parameters: mask: Sets which channels that will be scanned. 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). This API will not enable the injection channel.

Return Value: None

Side Effects: None

void ADC_EnableInjection(void)

Description: Enables the injection channel for the next scan only.

Parameters: None

Return Value: None

Side Effects: None

void ADC_SetLowLimit(uint32 lowLimit)

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

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

Return Value: None

Side Effects: None

void ADC_SetHighLimit(uint32 highLimit)

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

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

Return Value: None

Side Effects: None

void ADC_SetLimitMask(uint32 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).

Return Value: None

Side Effects: None

void ADC_SetSatMask(uint32 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).

Return Value: None

Side Effects: None

void ADC_SetOffset(uint32 chan, int16 offset)

Description: Sets the ADC offset which is used by the functions ADC_CountsTo_uVolts, ADC_CountsTo_mVolts and ADC_CountsTo_Volts to subtract the offset from the given reading before calculating the voltage conversion.

Parameters: chan: ADC channel number.

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

Return Value: None

Side Effects: None.

void ADC_SetGain(uint32 chan, int32 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. It should only be used to further calibrate the ADC with a known input or if an external reference is used. Affects the ADC_CountsTo_uVolts, ADC_CountsTo_mVolts and ADC_CountsTo_Volts functions by supplying the correct conversion between ADC counts and voltage.

Parameters: chan: ADC channel number.

adcGain: ADC gain in counts per 10 volt.

Return Value: None

Side Effects: None.

float32 ADC_CountsTo_Volts(uint32 chan, int16 adcCounts)

Description: Converts the ADC output to Volts as a floating point number. For example, if the ADC measured 0.534 volts, the return value would be 0.534. The calculation of voltage depends on the value of the voltage reference. 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

Side Effects: None

int16 ADC_CountsTo_mVolts(uint32 chan, int16 adcCounts)

Description: Converts the ADC output to millivolts as a 16-bit integer. For example, if the ADC measured 0.534 volts, the return value would be 534. The calculation of voltage depends on the value of the voltage reference. 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.

Side Effects: None

int32 ADC_CountsTo_uVolts(uint32 chan, int16 adcCounts)

Description: Converts the ADC output to microvolts as a 32-bit integer. For example, if the ADC measured 0.534 volts, the return value would be 534000. The calculation of voltage depends on the value of the voltage reference. 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

Side Effects: None

void ADC_Sleep(void)

Description: This is the preferred routine to prepare the component for sleep. The ADC_Sleep() routine saves the current component state. Then it calls the ADC_Stop() function and calls ADC_SaveConfig() to save the hardware configuration.

Call the ADC_Sleep() function before calling the CySysPmDeepSleep() or the CySysPmHibernate() function. See the PSoC Creator *System Reference Guide* for more information about power-management functions.

Parameters: None

Return Value: None

Side Effects: If this function is called twice in the enable state of the component, the disabled state of the component will be stored. So ADC_Enable() and ADC_StartConvert() must be called after ADC_Wakeup() in this case.

void ADC_Wakeup(void)

Description: This is the preferred routine to restore the component to the state when ADC_Sleep() was called. The ADC_Wakeup() function calls the ADC_RestoreConfig() function to restore the configuration. If the component was enabled before the ADC_Sleep() function was called, the ADC_Wakeup() function also re-enables the component.

Parameters: None

Return Value: None

Side Effects: Calling this function without previously calling ADC_Sleep() may lead to unpredictable results.

void ADC_SaveConfig(void)

Description: This function saves the component configuration and nonretention registers. It also saves the current component parameter values, as defined in the Configure dialog or as modified by the appropriate APIs. This function is called by the ADC_Sleep() function.

Parameters: None

Return Value: None

Side Effects: All ADC configuration registers are retained. This function does not have an implementation and is meant for future use. It is provided here so that the APIs are consistent across components.

void ADC_RestoreConfig(void)

Description: This function restores the component configuration and nonretention registers. It also restores the component parameter values to what they were before calling the ADC_Sleep() function.

Parameters: None

Return Value: None

Side Effects: Calling this function without previously calling ADC_SaveConfig() or ADC_Sleep() may produce unexpected behavior. This function does not have an implementation and is meant for future use. It is provided here so that the APIs are consistent across components.

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_offset[]	This array calibrates the offset for each channel. It is set to 0 the first time ADC_Start() is called and can be modified using ADC_SetOffset(). The array affects the ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), and ADC_CountsTo_uVolts() functions by subtracting the given offset.
ADC_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_SEQUENCED_CHANNELS_NUM	This constant represents the amount of input sequencing channels available for scanning.
ADC_TOTAL_CHANNELS_NUM	This constant represents the total number of input channels including the injection channel.

Macro Callbacks

Macro callbacks allow users to execute code from the API files that are automatically generated by PSoC Creator. Refer to the PSoC Creator Help and *Component Author Guide* for the more details.

In order to add code to the macro callback present in the component's generated source files, perform the following:

- Define a macro to signal the presence of a callback (in *cyapicallbacks.h*). This will "uncomment" the function call from the component's source code.
- Write the function declaration (in *cyapicallbacks.h*). This will make this function visible by all the project files.
- Write the function implementation (in any user file).

Callback Function [3]	Associated Macro	Description
ADC_ISR_InterruptCallback	ADC_ISR_INTERRUPT_CALLBACK	Used in the ADC_ISR() interrupt handler to perform additional application-specific actions.

Sample Firmware Source Code

PSoC Creator provides numerous example projects 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 Example Project" topic in the PSoC Creator Help for more information.

Interrupt Service Routine

The Sequencing SAR ADC contains a blank interrupt service routine in the file *ADC_INT.c*. You can place custom code in the designated areas to perform whatever function is required at the end of a conversion. A copy of the blank interrupt service routine is shown below. Place custom code between the "`/* `#START MAIN_ADC_ISR` */`" and "`/* `#END` */`" comments. This ensures that the code will be preserved, when a project is regenerated.

```
CY_ISR( ADC_ISR )
{
    uint32 intr_status;
```

[3] The callback function name is formed by component function name optionally appended by short explanation and "Callback" suffix.

```
        /* Rear interrupt status register */
        intr_status = ADC_SAR_INTR_REG;

        /***************************************************************************
        *  Custom Code
        *  - add user ISR code between the following #START and #END tags
        ***************************************************************************/
        /* `#START MAIN_ADC_ISR`  */

        /* `#END`  */

        /* Clear handled interrupt */
        ADC_SAR_INTR_REG = intr_status;
}
```

A second designated area is available to place variable definitions and constant definitions.

```
/* System variables */

/* `#START ADC_SYS_VAR`  */
    /* Place user code here. */
/* `#END`  */
```

An example of code that uses an interrupt to capture data follows.

```
#include <device.h>

int16 result = 0;
uint8 dataReady = 0;
void main()
{
    int16 newReading = 0;
    CYGlobalIntEnable;           /* Enable Global interrupts */
    ADC_SAR_1_Start();           /* Initialize ADC */
    ADC_SAR_1_IRQ_Enable();      /* Enable ADC interrupts */
    ADC_SAR_1_StartConvert();    /* Start ADC conversions */
    for(;;)
    {
        if (dataReady != 0)
        {
           dataReady = 0;
           newReading = result;
           /* More user code */
        }
    }
}
```

Interrupt code segments in the file *ADC_INT.c*.

```
    /*********************************
    *      System variables
    *********************************/
    /* `#START ADC_SYS_VAR`  */
    extern int16 result;
```

```
extern uint8 dataReady;
/* `#END`  */

CY_ISR( ADC_ISR )
{
     uint32 intr_status;

     /* Read interrupt status register */
     intr_status = ADC_SAR_INTR_REG;

     /***************************************************************************
     *  Custom Code
     *  - add user ISR code between the following #START and #END tags
     ***************************************************************************/
     /* `#START MAIN_ADC_ISR`  */
     result = ADC_GetResult16(0);
     dataReady = 1;
     /* `#END`  */

     /* Clear handled interrupt */
     ADC_SAR_INTR_REG = intr_status;
}
```

It is important to set the Sample Rate and Master Clock parameters correctly.

You can optimize the ISR by reading result registers directly:

```
CY_ISR( ADC_ISR )
{
     uint32 intr_status;

     /* Rear interrupt status register */
     intr_status = ADC_SAR_INTR_REG;

     /***************************************************************************
     *  Custom Code
     *  - add user ISR code between the following #START and #END tags
     ***************************************************************************/
     /* `#START MAIN_ADC_ISR`  */
     result = (int16)(ADC_SAR_CHAN0_RESULT_REG & ADC_RESULT_MASK);
     dataReady = 1;
     /* `#END`  */

     /* Clear handled interrupt */
     ADC_SAR_INTR_REG = intr_status;
}
```

Note that you may use an alternative Interrupt service routine, located in your *main.c* file. In this case use the following template:

Implement interrupt service routine in *main.c*:

```
CY_ISR( ADC_SAR_SEQ_ISR_LOC )
{
    uint32 intr_status;
```

```
    /* Read interrupt status register */
    intr_status = ADC_SAR_SEQ_SAR_INTR_REG;

    /* Place your code here */

    /* Clear handled interrupt */
    ADC_SAR_SEQ_SAR_INTR_REG = intr_status;
}
```

Enable ADC interrupt and set interrupt handler to local routine:

```
ADC_SAR_SEQ_IRQ_StartEx(ADC_SAR_SEQ_ISR_LOC);
```

Refer to the Interrupt component datasheet for more information.

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. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Sequencing SAR ADC component has the following specific deviation:

MISRA-C: 2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
8.7	R	Objects shall be defined at block scope if they are only accessed from within a single function.	The object 'ADC_channelsConfig' is always accessed from ADC_Init() function and optionally, depend on component configuration, from ADC_CountsTo_mVolts(), ADC_CountsTo_uVolts, ADC() and ADC_CountsTo_Volts() functions. The intention of this publicly available static variable is to allow more efficient code.

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 in release mode with optimization set for size. For a specific design analyze the map file generated by the compiler to determine the memory usage.

Configuration	PSoC 4100/PSoC 4200 (GCC)		Other PSoC 4 Devices (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default	1064	26	1064	26
Default with INJ	1148	34	1152	34

Functional Description

Sequencing SAR ADC contains the following blocks:

- SARMUX
- SARADC core
- SARREF
- SARSEQ

The SARADC core is a fast 12-bit ADC with SAR architecture. Preceding the SARADC is the SARMUX, which can route external pins and internal signals such as the temperature sensor (DieTemp) or operational amplifier (Opamp), to the eight (sixteen) internal channels of the SARADC. SARREF is used for multiple reference 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. The possible SARMUX connections are:

- Dedicated analog pins
- Analog pins, routed through AMUXA, AMUXB buses
- Opamps
- DieTemp

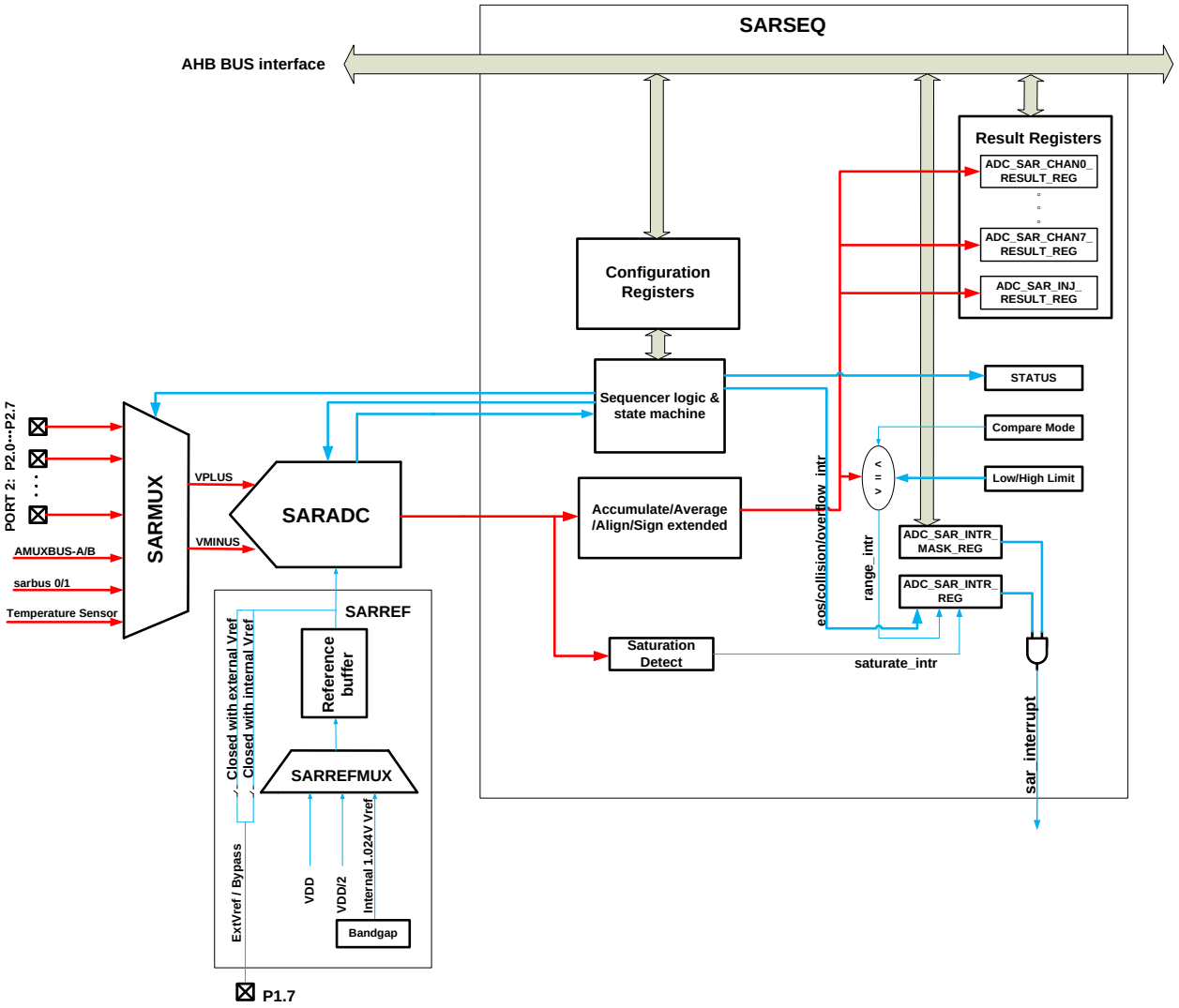
Take into account the following recommendations for the differential inputs:

- Pins may only be paired with an adjacent pin.
- When routing pins, Vplus must be an even-numbered pin. Vminus must be the next pin in the port (e.g., if vplus is P3[0], vminus would be P3[1]).
- Opamps may only be paired with an adjacent opamp.
- When routing opamps, vplus must be an even-numbered opamp. Vminus must be the next opamp (e.g., if vplus is OA2, vminus would be OA3).

The ninth (seventeenth) channel is an injection channel that firmware uses for infrequent and incidental sampling of pins and signals such as the temperature sensor.

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.

Block Diagram



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.

* 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."

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 7(15).

ADC_SAR_INJ_RESULT_REG

Bits	Name	Description
15:0	Data	SAR conversion result of the injection channel.
28	ADC_INJ_COLLISION_INTR_MIR	Mirror bit of corresponding bit in ADC_SAR_INTR_REG register
29	ADC_INJ_SATURATE_INTR_MIR	Mirror bit of corresponding bit in ADC_SAR_INTR_REG register
30	ADC_INJ_RANGE_INTR_MIR	Mirror bit of corresponding bit in ADC_SAR_INTR_REG register
31	ADC_INJ_EOC_INTR_MIR	Mirror bit of corresponding bit in ADC_SAR_INTR_REG register

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.
4	ADC_INJ_EOC_MASK*	Injection End of Conversion Interrupt: hardware sets this interrupt after completing the conversion for the injection channel. Note that the ADC_EOS_MASK is raised in parallel to starting the injection channel conversion. The injection channel is not considered part of the scan. Write with '1' to clear bit after picking up the data from the ADC_SAR_INJ_RESULT_REG register
5	ADC_INJ_SATURATE_MASK	Injection Saturation Interrupt: hardware sets this interrupt if an injection conversion result (before averaging) is either 0x000 or 0xFFF (for 12-bit resolution), this is an indication that the ADC likely saturated. Write with '1' to clear bit.

Bits	Name	Description
6	ADC_INJ_RANGE_MASK	Injection Range detect Interrupt: hardware sets this interrupt if the injection conversion result (after averaging) met the condition specified by the **Compare Mode** parameter. Write with '1' to clear bit.
7	ADC_INJ_COLLISION_MASK	Injection Collision Interrupt. This function is disabled by default.

These two 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 (for 12-bit resolution), this is an indication that the ADC likely saturated. When a 10-bit or 8-bit resolution is selected for the channel, then the upper bits are ignored. 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 sequencing SAR ADC is implemented as a fixed-function block. The component also uses one Interrupt.

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

Note The data for the PSoC 4200L device is preliminary. Final data will be delivered in an upcoming Component Pack.

DC Specifications

Parameter	Description	Min	Typ	Max	Units	Conditions
A_RES	Resolution	–	–	12	bits	
A_CHNIS_S	Number of channels - single ended	–	–	8		
A_CHNIS_S	Number of channels - single ended	–	–	16		PSoC 4200-BLE, PSoC 4100M/ PSoC 4200M, PSoC 4200L
A-CHNKS_D	Number of channels - differential	–	–	4		Diff inputs use neighboring I/O

Parameter	Description	Min	Typ	Max	Units	Conditions
A-CHNKS_D	Number of channels - differential	–	–	8		PSoC 4200-BLE, PSoC 4100M / PSoC 4200M, PSoC4200L Diff inputs use neighboring I/O
A-MONO	Monotonicity	–	–	–		Yes. Based on characterization
A_GAINERR	Gain error	–	–	±0.1	%	With external reference
A_OFFSET	Input offset voltage	–	–	2	mV	Measured with 1-V V_{REF}
A_ISAR	Current consumption	–	–	1	mA	
A_VINS	Input voltage range - single ended	V_{SS}	–	V_{DDA}	V	Based on characterization
A_VIND	Input voltage range - differential	V_{SS}	–	V_{DDA}	V	Based on characterization
A_INRES	Input resistance	–	–	2.2	KΩ	Based on characterization
A_INCAP	Input capacitance	-	-	10	pF	Based on characterization

AC Specifications

Parameter	Description	Min	Typ	Max	Units	Conditions
A_PSRR	Power supply rejection ratio	70	–	–	dB	
A_CMRR	Common mode rejection ratio	66	–	–	dB	Measured at 1 V
A_SAMP_1	Sample rate with external reference bypass cap	–	–	1	Msps	806 Ksps for PSoC 4100 BLE, PSoC 4100M
A_SAMP_2	Sample rate with no bypass cap. Reference = VDD	–	–	500	Ksps	
A_SAMP_3	Sample rate with no bypass cap. Internal reference	–	–	100	Ksps	
A_SNDR	Signal-to-noise and distortion ratio (SINAD)	65	–	–	dB	F_{IN} = 10 kHz
A_INL	Integral non linearity	–1.7	–	+2	LSB	V_{DD} = 1.71 to 5.5, 1 Msps, Vref = 1 to 5.5
A_INL	Integral non linearity	–1.5	–	+1.7	LSB	V_{DDD} = 1.71 to 3.6, 1 Msps, Vref = 1.71 to V_{DDD}
A_INL	Integral non linearity	–1.5	–	+1.7	LSB	V_{DDD} = 1.71 to 5.5, 500 Ksps, Vref = 1 to 5.5
A_DNL	Differential non linearity	–1	–	+2.2	LSB	V_{DDD} = 1.71 to 5.5, 1 Msps, Vref = 1 to 5.5
A_DNL	Differential non linearity	–1	–	+2	LSB	V_{DDD} = 1.71 to 3.6, 1 Msps, Vref = 1.71 to V_{DDD}
A_DNL	Differential non linearity	–1	–	+2.2	LSB	V_{DDD} = 1.71 to 5.5, 500 Ksps, Vref = 1 to 5.5
A_THD	Total harmonic distortion	–	–	–65	dB	F_{IN} = 10 kHz.

Block Specs

Parameter	Description	Min	Typ	Max	Units	Conditions
VREFSAR	Trimmed internal reference to SAR	-1	–	+1	%	Percentage of Vbg (1.024 V). Guaranteed by characterization

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.30.a	Datasheet edit.	Updated characterization data.
2.30	Added PSoC 4200L device support.	New device support.
	Updated datasheet.	Updated Functional Description section.
2.20.a	Datasheet update.	Added Macro Callbacks section.
2.20	The ADC_Init() function was updated to perform configuration of eight upper channels (from 8 to 15). Added defines for eight upper channels.	Fixed erratum 210193. **Note** The Errata section was removed from this datasheet.
	Updated datasheet.	Updated to the numbers in the API Memory Usage section.
2.10.a	Edited the datasheet.	Added Component Errata section to document a defect in 2.0 and 2.10 versions of the component. Added DMA Support section to include support for PSoC 4100M/PSoC 4200M devices. Updated DC and AC Electrical Characteristics section with PSoC 4100M/PSoC 4200M data.
2.10	Updated ADC_Start() API to enable the internal SAR pump when the global pump is enabled. The SAR pump is disabled in ADC_Sleep() API and restored in ADC_Wakeup() API.	The SAR pump should be enabled for low VDDA (< 4V) voltage to meet specified AC electrical characteristics. For low VDDA application make sure that VDDA voltage is set correctly in DWR system settings. ADC_Sleep() API must be called before entering chip to low power mode.
	Removed Input Buffer Gain parameter.	This option is not supported by current PSoC 4 devices.
	Fixed an issue with placing the component on the schematic of another component.	The component caused build errors when was placed as a sub-component on the schematic of another component.
	Extended the Sequenced Channels parameter maximum value.	The maximum number depends on the device, mode (differential or single ended), and available resources outside of the SAR.

Version	Description of Changes	Reason for Changes / Impact
	Updated v2.0 change log to expand on the maximum sample rate limitation.	Absent description of the reason for the change.
2.0.a	Datasheet updates.	Updated DC and AC Electrical Characteristics section to support PSoC 4200 BLE devices. Updated Application Programming Interface section. Updated Component Parameters section. Updated Interrupt Service Routine section. Clarified Vref pin description/usage.
2.0	The maximum sampling rate when operating with VDDA reference was reduced from 1 Msps to 500 Ksps.	1 Msps rate is not guaranteed by the silicon when used without a bypass capacitor. Refer to the VRef Select parameter for more information.
	Added support for Bluetooth Low Energy (BLE) devices. Various datasheet updates	Clarified Actual sample rate description. Updated DC and AC Electrical Characteristics section. Added Note to Application Programming Interface section. Updated the Functional Description section.
1.10	Fixed an issue with the – input internally connected to reference in differential mode when "Single ended mode input" is selected to Vref in the customizer.	
	Updated trim value.	Changes required after device characterization.
1.0.a	Updated the MISRA-C Rule table.	
1.0	First component release	

© Cypress Semiconductor Corporation, 2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.