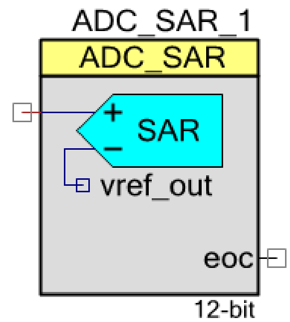
ADC Successive Approximation Register (ADC_SAR)

3.0



Features

- Supports PSoC 5LP family of devices
- 12-bit resolution at up to 1 msps maximum
- Four power modes
- Selectable resolution and sample rate
- Single-ended or differential input

General Description

The ADC Successive Approximation Register (ADC_SAR) component provides medium-speed (maximum 1-msps sampling), medium-resolution (12 bits maximum), analog-to-digital conversion.

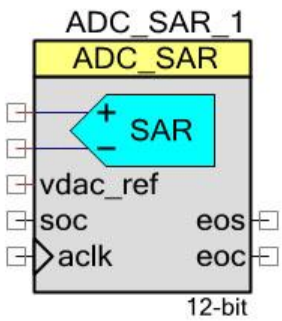
When to Use an ADC_SAR

Typical applications for the ADC_SAR component include:

- LED lighting control
- Motor control
- Magnetic card reader
- High-speed data collection
- Power meter
- Pulse oximeter

Input/Output Connections

This section describes the input and output connections for the ADC_SAR. An asterisk (*) in the list of I/Os indicates 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. The conversion result is a function of the +Input signal minus the voltage reference. The voltage reference is either the – Input signal or V_{SSA}.

–Input – Analog *

When shown, this optional input is the negative analog signal (or reference) input to the ADC_SAR. The conversion result is a function of +Input minus–Input. You see this pin when you set the **Input Range** parameter to one of the differential modes.

vdac_ref – Input *

The VDAC reference (vdac_ref) is an optional pin. You see it if you have selected **Vssa to VDAC*2 (Single Ended)** or **0.0 +/- VDAC (Differential)** input range; otherwise, this I/O is hidden. You can only connect this pin to a VDAC component output. Do not connect it to any other signal.

soc – Input *

The start of conversion (soc) is an optional pin. You see it if you select the **Hardware Trigger** or **Software trigger** sample mode. A rising edge on this input starts an ADC conversion. If this input is high when the SAR_Start() function is called, a conversion will start immediately. After the first conversion, a rising edge on the input will start an ADC conversion. The first **soc** rising edge should be generated at least 10 us after the component is started to guarantee reference and pump voltage stability. You can connect the output of a PWM component to this input. It can also be connected to any GPIO pin or a UDB. This signal should be synchronized to the ADC_SAR clock and must be at least one ADC_SAR clock cycle wide. If you set the **Sample**

Mode parameter to **Free Running**, this I/O is hidden. Refer to Sample Mode section for more information.

aclk – Input *

You can see 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 and resolution.

eos – Output *

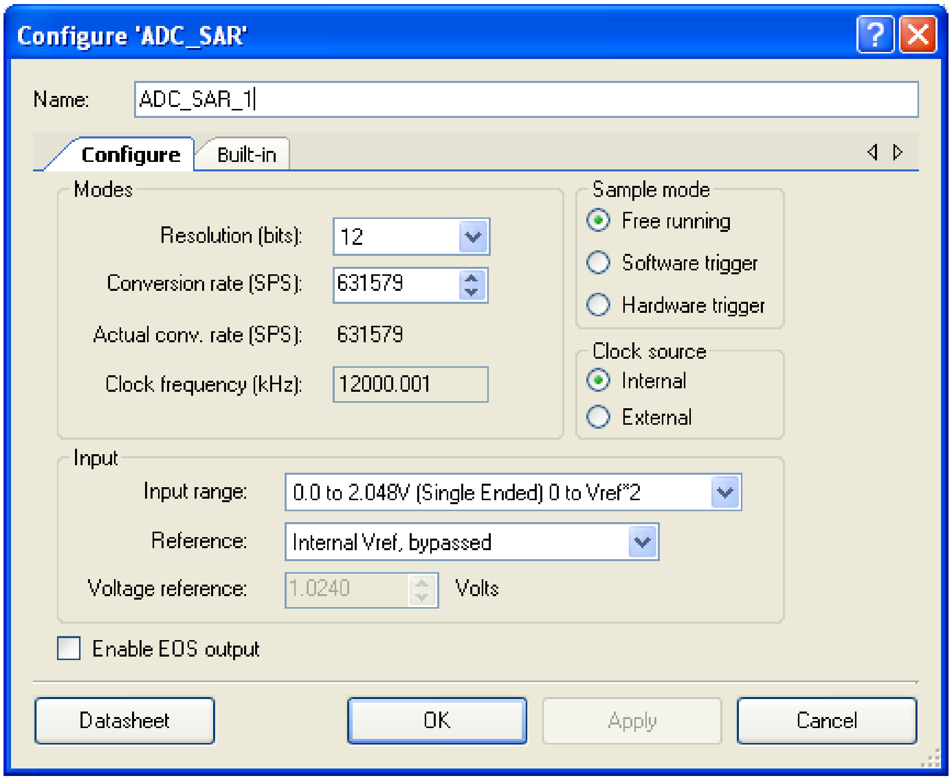
A rising edge on the end of sampling (eos) output indicates the completion of the sampling window. This signal can be used to control the input channel multiplexer. The input multiplexer selection can be changed after sampling is complete, but still during the conversion. The eos signal allows the SAR ADC to operate at its maximum speed. This output is visible if the **Enable EOS output** parameter is selected.

eoc – Output

A rising edge on the end of conversion (eoc) output means that a conversion is complete. A DMA request can be connected to this pin to transfer the conversion output to system RAM, DFB, or other component. An internal interrupt is also connected to this signal, or you may connect your own interrupt.

Component Parameters

Drag an ADC_SAR component onto your design and double-click it to open the **Configure** dialog.



The ADC_SAR has the following parameters. The option shown in bold is the default.

Modes

Resolution

Sets the resolution of the ADC.

ADC_Resolution	Value	Description
12	12	Sets resolution to 12 bits.
10	10	Sets resolution to 10 bits.
8	8	Sets resolution to 8 bits.

Conversion Rate

This parameter sets the ADC conversion. The conversion time is the inverse of the conversion rate. Enter the conversion rate in samples per second. Converting one sample in free running sample mode takes 18 clock cycles, or 16 clock cycles if Reference is Internal Vref (not bypassed). The conversion time of each sample is more than four cycles when hardware trigger sample mode is used. The actual conversion rate may differ based on available clock speed and divider range.

Clock Frequency

This text box is a read-only (always grayed out) area that displays the required clock rate for the selected operating conditions: resolution and conversion rate. It is updated when either or both of these conditions change. Clock frequency can be anywhere between 1 MHz and 18 MHz. The duty cycle should be 50 percent. The minimum pulse width should be greater or equal to 25.5 ns. PSoC Creator will generate an error during the build process if the clock does not fall within these limits. In that case, change the Master Clock in the Design-Wide Resources Clock Editor.

At high conversation rates, the ADC can generate large amounts of data to process. In these cases, the data should either be collected using DMA or by using the CPU. If using the CPU, the CPU clock should be at a high clock rate and with a minimal interrupt service routine. For example, at a conversion rate of 700,000 samples per second and a CPU clock rate of 66 MHz, there are only 66 MHz / 700,000 sps = 94 CPU clock cycles per sample. Refer to the Interrupt Service Routine section for guidance on optimizing the ISR.

Sample Mode

This parameter determines how the ADC operates.

Start_of_Conversion	Description
Free Running	ADC runs continuously. Use the ADC_StartConvert() function to start and the ADC_StopConvert() to stop continuous conversions.
Software trigger	This is a mix of hardware and software triggered operation. Hardware triggering works until ADC_StartConvert() function has been called first time. ADC_StartConvert() function disables the external SOC input and starts a single conversion. ADC_StopConvert() function enables SOC input and Hardware triggering sampling mode.
Hardware trigger	A rising-edge pulse on the SOC pin starts a single conversion.

Clock Source

This parameter allows you to select either a clock that is internal to the ADC_SAR module or an external clock.

ADC_Clock	Description
Internal	Use the internal clock of the ADC_SAR.
External	Use an external clock. The clock source can be analog, digital, or generated by another component.

Input

Input Range

This parameter configures the ADC for a given input range. The analog signals connected to the PSoC must be between V_{SSA} and V_{DDA} regardless of the input range settings.

Input Range	Description
0.0 to 2.048V (Single Ended) **0 to Vref*2**	When using the internal reference (1.024 V), the usable input range is 0.0 to 2.048 V. The ADC is configured to operate in a single-ended input mode with –Input connected internally to Vrefhi_out. If you are using an external reference voltage, the usable input range is 0.0 to Vref*2.
Vssa to Vdda (Single Ended)	This mode uses the V_{DDA}/2 reference; the usable input range covers the full analog supply voltage. The ADC is put in a single-ended input mode with –Input connected internally to Vrefhi_out. If you are using an external reference voltage, the usable input range is 0.0 to Vref*2.
Vssa to VDAC*2 (Single Ended)	This mode uses the VDAC reference, which should be connected to the vdac_ref pin. The usable input range is Vssa to VDAC*2 volts. The ADC is configured to operate in a single-ended input mode with –Input connected internally to Vrefhi_out.
0.0 ± 1.024V (Differential) –Input ± Vref	This mode is configured for differential inputs. When using the internal reference (1.024 V), the input range is –Input ± 1.024 V. For example, if –Input 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 –Input 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, –Input ± Vref.
0.0 ± Vdda (Differential) –Input ± Vdda	This mode is configured for differential inputs and is ratiometric with the supply voltage. The input range is –Input ± Vdda. For systems in which both single-ended and differential signals are scanned, connect –Input to Vssa when scanning a single-ended input. If you are using an external reference voltage, the usable input range is –Input ± Vref.
0.0 ± Vdda/2 (Differential) –Input ± Vdda/2	This mode is configured for differential inputs and is ratiometric to the supply voltage. The input range is –Input ± Vdda/2. For systems in which both single-ended and differential signals are scanned, connect –Input to Vssa when scanning a single-ended input. If you are using an external reference voltage, the usable input range is –Input ± Vref.
0.0 ± VDAC (Differential) –Input ± VDAC	This mode is configured for differential inputs and uses the VDAC reference, which should be connected to the vdac_ref pin. The input range is –Input ±VDAC. For systems in which both single-ended and differential signals are scanned, connect –Input to Vssa when scanning a single-ended input.

Reference

This parameter selects the switches for reference configuration for the ADC_SAR.

ADC_Reference	Allowed Clock Frequency (MHz)	Description
Internal Vref	1 ~ 1.6 MHz	Uses the internal reference. This clock frequency range is valid with all input ranges except "0.0 ± Vdda." Uses the Internal Vref, bypassed option for higher clock frequency.
	1 ~ 9 MHz	Uses the internal reference. This clock frequency range is valid for the "0.0 ± Vdda" input range.
Internal Vref, bypassed	1 ~ 18 MHz	Uses the internal reference. You must place a bypass capacitor on pin P0[2]* for SAR1 or on pin P0[4]* for SAR0. This mode is not applicable with "0.0 ± Vdda" input range.
External Vref	1 ~ 18 MHz	Uses an external reference on pin P0[2] for SAR1 or on pin P0[4] for SAR0.

* 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, configure either port pin P0[2] or P0[4] as an analog HI-Z pin and connect an external capacitor with a value between 0.01 µF and 10 µF.

Note The same internal reference is used for ADC_SAR and for ADC_DelSig components. If both types of the ADC have to work with internal reference simultaneously, use the **Internal Vref, bypassed** option for the best performance.

Note When using an external reference or externally bypassing the internal reference, use the Lock feature in the Pins tab of the Design Wide Resources(DWR) on the ADC_SAR:ExtVref or ADC_SAR:Bypass pin. This will lock the SAR component to the designated SAR hardware block.

Voltage Reference

The voltage reference is used for the ADC count to voltage conversion functions discussed in the Application Programming Interface section. This parameter is read-only when using the internal reference. When using an external reference, you can edit this value to match the external reference voltage.

- When selecting input range **Vssa to Vdda**, **-Input +/- Vdda**, or **-Input +/- Vdda/2**, the value is derived from the V_{DDA} setting in System tab of the DWR.
- When selecting the input range **Vssa to VDAC*2** or **–Input +/- VDAC**, enter the VDAC supply voltage value.

Note The input range and reference voltage is limited by the V_{DDA} voltage.

Enable EOS output

This parameter enables the End-of-Sampling output.

Application Programming Interface

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

By default, PSoC Creator assigns the instance name "ADC_SAR_1" to the first instance of a component in a given design. You can rename the instance 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."

Functions

Function	Description
ADC_Start()	Powers up the ADC and resets all states
ADC_Stop()	Stops ADC conversions and reduces the power to the minimum
ADC_SetPower()	Sets the power mode
ADC_SetResolution()	Sets the resolution of the ADC
ADC_StartConvert()	Starts conversions
ADC_StopConvert()	Stops conversions
ADC_IRQ_Enable()	An internal IRQ is connected to the eoc. This API enables the internal ISR.
ADC_IRQ_Disable()	An internal IRQ is connected to the eoc. This API disables the internal ISR.
ADC_IsEndConversion()	Returns a nonzero value if conversion is complete
ADC_GetResult8()	Returns a signed 8-bit conversion result
ADC_GetResult16()	Returns a signed16-bit conversion result
ADC_SetOffset()	Sets the offset of the ADC
ADC_SetScaledGain()	Sets the ADC gain in counts per 10 volts
ADC_CountsTo_Volts()	Converts ADC counts to floating-point volts
ADC_CountsTo_mVolts()	Converts ADC counts to millivolts
ADC_CountsTo_uVolts()	Converts ADC counts to microvolts
ADC_Sleep()	Stops ADC operation and saves the user configuration
ADC_Wakeup()	Restores and enables the user configuration
ADC_Init()	Initializes the default configuration provided with the customizer
ADC_Enable()	Enables the clock and power for the ADC
ADC_SaveConfig()	Saves the current user configuration
ADC_RestoreConfig()	Restores the user configuration

void ADC_Start(void)

Description: This is the preferred method to begin component operation. ADC_Start() sets the initVar variable, calls the ADC_Init() function, and then calls the ADC_Enable() function.

Parameters: None

Return Value: None

Side Effects: If the initVar variable is already set, this function only calls the ADC_Enable() function.

void ADC_Stop(void)

Description: Stops ADC conversions and reduces the power to the minimum.

Parameters: None

Return Value: None

Side Effects: None

void ADC_SetPower(uint8 power)

Description: Sets the operational power of the ADC. You should use the higher power settings with faster clock speeds.

Parameters: uint8 power: Power setting

Parameter Name	Value	Description	Clock Rate
ADC__HIGHPOWER	0	Normal power	18 MHz
ADC__MEDPOWER	1	1/2 power	4.5 MHz
ADC__LOWPOWER	2	1.25 power	reserved
ADC__MINPOWER	3	1/4 power	2.25 MHz

Return Value: None

Side Effects: The power setting may affect conversion accuracy.

void ADC_SetResolution(uint8 resolution)

Description: Sets the resolution for the GetResult16() and GetResult8() APIs.

Parameters: uint8 resolution: Resolution setting

Parameter Name	Value	Description
ADC__BITS_12	12	Sets resolution to 12 bits.
ADC__BITS_10	10	Sets resolution to 10 bits.
ADC__BITS_8	8	Sets resolution to 8 bits.

Return Value: None

Side Effects: The ADC resolution cannot be changed during a conversion cycle. The recommended best practice is to stop conversions with ADC_StopConvert(), change the resolution, then restart the conversions with ADC_StartConvert().

If you decide not to stop conversions before calling this API, use ADC_IsEndConversion() to wait until conversion is complete before changing the resolution.

If you call ADC_SetResolution() during a conversion, the resolution will not change until the current conversion is complete. Data will not be available in the new resolution for another 6 + "New Resolution(in bits)" clock cycles. You may need add a delay of this number of clock cycles after ADC_SetResolution() is called before data is valid again.

Affects ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), and ADC_CountsTo_uVolts() by calculating the correct conversion between ADC counts and the applied input voltage. Calculation depends on resolution, input range, and voltage reference.

void ADC_StartConvert(void)

Description: Forces the ADC to initiate a conversion. In free-running mode, the ADC runs continuously. In software trigger mode, the function also acts as a software version of the SOC and every conversion must be triggered by ADC_StartConvert().This function is not available when the **Hardware Trigger** sample mode is selected.

Parameters: None

Return Value: None

Side Effects: Calling ADC_StartConvert() disables the external SOC pin.

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. This function is not available when the **Hardware Trigger** sample mode is selected.

Parameters: None

Return Value: None

Side Effects: In **Software Trigger** sample mode, this function sets a software version of the SOC to low level and switches the SOC source to hardware SOC input.

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. To enable global interrupts, call the enable global interrupt macro "`CYGlobalIntEnable;`" in your *main.c* file before enabling any interrupts.

Parameters: None

Return Value: None

Side Effects: Enables interrupts to occur. Reading the result clears the interrupt.

void ADC_IRQ_Disable(void)

Description: Disables interrupts at the end of a conversion.

Parameters: None

Return Value: None

Side Effects: None

uint8 ADC_IsEndConversion(uint8 retMode)

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

Parameters: uint8 retMode: Check conversion return mode. See the following table for options.

Option	Description
ADC_RETURN_STATUS	Immediately returns the status. 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 is complete.

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, which is cleared on read.

int8 ADC_GetResult8(void)

Description: Returns the result of an 8-bit conversion. If the resolution is set greater than 8 bits, the function returns the LSB of the result. ADC_IsEndConversion() should be called to verify that the data sample is ready.

Parameters: None

Return Value: int8: The LSB of the last ADC conversion.

Side Effects: Converts the ADC counts to the 2's complement form.

int16 ADC_GetResult16(void)

Description: Returns a 16-bit result for a conversion with a result that has a resolution of 8 to 12 bits. ADC_IsEndConversion() should be called to verify that the data sample is ready.

Parameters: None

Return Value: int16: The 16-bit result of the last ADC conversion

Side Effects: Converts the ADC counts to the 2's complement form.

void ADC_SetOffset(int16 offset)

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

Parameters: int16 offset: This value is measured when the inputs are shorted or connected to the same input voltage.

Return Value: None

Side Effects: Affects ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), and ADC_CountsTo_uVolts() by subtracting the given offset.

void ADC_SetScaledGain(int16 adcGain)

Description: Sets the ADC gain in counts per 10 volts for the voltage conversion functions that follow. 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 the ADC is using an external reference.

Parameters: int16 adcGain: ADC gain in counts per 10 volts. To calibrate the gain, supply close to reference voltage to ADC inputs and measure it by multimeter. Calculate the gain coefficient using following formula.

$$adcGain = \frac{counts \times 10}{V_{measured}}$$

Where the **counts** is returned from ADC_GetResult16() value, $V_{measued}$ – measured by multimeter voltage in volts.

Return Value: None

Side Effects: Affects ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), ADC_CountsTo_uVolts() by supplying the correct conversion between ADC counts and the applied input voltage.

float ADC_CountsTo_Volts(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 Design Wide Resources (DWR).

Parameters: int16 adcCounts: Result from the ADC conversion

Return Value: Float: Result in volts

Side Effects: None

int16 ADC_CountsTo_mVolts(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 Design Wide Resources (DWR).

Parameters: int16 adcCounts: Result from the ADC conversion

Return Value: int16: Result in mV

Side Effects: None

int32 ADC_CountsTo_uVolts(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 Design Wide Resources (DWR).

Parameters: int16 adcCounts: Result from the ADC conversion

Return Value: int32: 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.

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

Parameters: None

Return Value: None

Side Effects: None

void ADC_Wakeup(void)

Description: This is the preferred routine to restore the component to the state when ADC_Sleep() was called. 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 the ADC_Wakeup() function without first calling the ADC_Sleep() or ADC_SaveConfig() function can produce unexpected behavior.

void ADC_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call ADC_Init() because the ADC_Start() routine calls this function and is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: All registers will be set to values according to the customizer Configure dialog.

void ADC_Enable(void)

Description: Activates the hardware and begins component operation. The higher power is set automatically depending on clock speed. The ADC_SetPower() API description contains the relation of the power from the clock rate. It is not necessary to call ADC_Enable() because the ADC_Start() routine calls this function, which is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: None

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 first calling the ADC_Sleep() or ADC_SaveConfig() function can 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

Variable	Description
ADC_initVar	This variable indicates whether the ADC has been initialized. The variable is initialized to 0 and set to 1 the first time ADC_Start() is called. This allows the component to restart without reinitialization after the first call 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 variable calibrates the offset. It is set to 0 the first time ADC_Start() is called and can be modified using ADC_SetOffset(). The variable affects the ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), and ADC_CountsTo_uVolts() functions by subtracting the given offset.
ADC_countsPer10Volt	This variable is used to calibrate the gain. It is calculated the first time ADC_Start() is called and each time ADC_SetResolution() is called. The value depends on resolution, input range, and voltage reference. It can be changed using ADC_SetScaledGain(). This variable 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.
ADC_shift	In differential input mode the SAR ADC outputs digitally converted data in a binary offset scheme. This variable is used to convert the ADC counts to 2’s complement form. This variable is calculated the first time ADC_Start() is called and each time ADC_SetResolution() is called. The calculated value depends on the resolution and input mode. This variable affects the ADC_GetResult8() and ADC_GetResult16() functions by subtracting the correct shift value.

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 [1]	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 many 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.

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

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

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 ADC_SAR component does not have any specific deviations.

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. The following table provides the memory usage for all APIs available in the default component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration	PSoC 3 (Keil_PK51)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default	N/A	N/A	908	14

Interrupt Service Routine

The ADC_SAR contains a blank interrupt service routine in the file *ADC_SAR_1_INT.c* file, where "ADC_SAR_1" is the instance name. 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_SAR_1_ISR )
{
    /* Place user ADC ISR code here. This can be a good place */
    /* to place code that is used to switch the input to the  */
    /* ADC. It may be good practice to first stop the ADC     */
    /* before switching the input then restart the ADC.       */

    /* `#START MAIN_ADC_ISR` */
    /* Place user code here. */
    /* `#END`  */
}
```

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 in the free running sample mode follows.

```
#include <project.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_SAR_1_INT.c*.

```
   /**********************************
   *      System variables
   **********************************/
   /* `#START ADC_SYS_VAR`  */
    extern int16 result;
    extern uint8 dataReady;
    /* `#END`  */

CY_ISR(ADC_SAR_1_ISR )
{
    /**********************************************/
    /* Place user ADC ISR code here.              */
    /* This can be a good place to place code     */
    /* that is used to switch the input to the    */
    /* ADC. It may be good practice to first      */
    /* stop the ADC before switching the input    */
    /* then restart the ADC.                      */
    /**********************************************/
    /* `#START MAIN_ADC_ISR`  */
        result = ADC_SAR_1_GetResult16();
```

```
        dataReady = 1;
    /* `#END`  */
}
```

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

For example, at the maximum conversion rate (700 ksps at 12 bits) set the Master Clock to 53 MHz in the Design-Wide Resources Clock Editor, and optimize the ISR routine. Otherwise, the processor will not be able to handle the ISR quickly enough. If you select a lower Master Clock, the run time of the ISR will be longer than ADC_SAR conversion time.

You can optimize the ISR by reading sample registers directly:

```
CY_ISR(ADC_SAR_1_ISR )
{
    /**********************************************/
    /* Place user ADC ISR code here.              */
    /* This can be a good place to place code     */
    /* that is used to switch the input to the    */
    /* ADC. It may be good practice to first      */
    /* stop the ADC before switching the input    */
    /* then restart the ADC.                      */
    /**********************************************/
    /* `#START MAIN_ADC_ISR`  */
        result = CY_GET_REG16(ADC_SAR_1_SAR_WRK0_PTR);
        dataReady = 1;
    /* `#END`  */
}
```

Note 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_ISR_LOC )
{
         /* Place your code here */
}
```

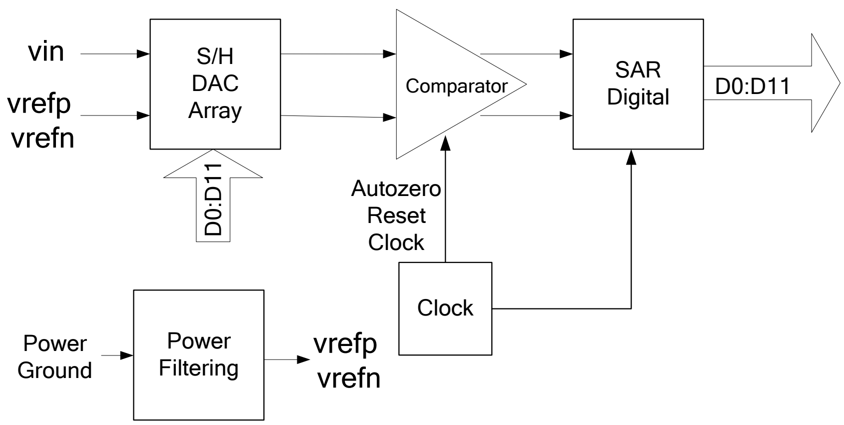
Enable ADC interrupt and set interrupt handler to local routine:

```
ADC_SAR_1_IRQ_StartEx(ADC_SAR_ISR_LOC);
```

Refer to the Interrupt component datasheet for more information.

Functional Description

The following figure shows a block diagram. An input analog signal is sampled and compared with the output of a DAC using a binary search algorithm to determine the conversion bits in succession from MSB to LSB.



DMA

You can use the DMA component to transfer converted results from ADC_SAR register to RAM. You should connect the DMA data request signal (DRQ) to the EOC pin from the ADC. You can use the DMA Wizard to configure DMA operation as follows:

Name of DMA Source	Length	Direction	DMA Request Signal	DMA Request Type	Description
ADC_SAR_WRK0_PTR	2	Source	EOF	Rising Edge	Receive a 2-byte result for a conversion with a result that always has 12-bit resolution. Note that this register is not sign extended; the result is always unsigned. A 0-V differential input returns a half-scale code. A full negative input returns a 0 code, and a full positive input returns a full-scale code.

Registers

Sample Registers

The ADC results can be between 8 and 12 bits of resolution. The output is divided into two 8-bit registers. The CPU or DMA can access these registers to read the ADC result.

ADC_SAR_WRK0_REG (SAR working register 0)

Bits	7	6	5	4	3	2	1	0
Value	Data[7:0]							

ADC_SAR_WRK1_REG (SAR working register 1)

Bits	7	6	5	4	3	2	1	0
Value	overrun_det NA	N/A			Data[11:8]			

- Data[11:0]: The ADC results
- overrun_det: Data overrun detection flag. This function is disabled by default.

Resources

The ADC_SAR uses a fixed-block SAR in the silicon.

DC and AC Electrical Characteristics

The following values indicate performance for PSoC 5LP.

DC Specifications

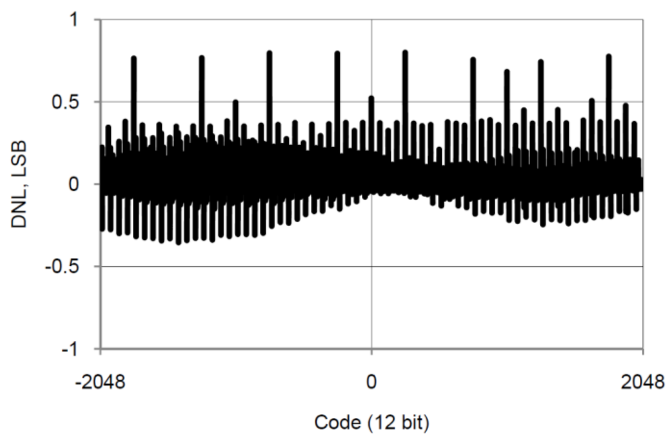
Parameter	Description	Conditions	Min	Typ	Max	Units
	Resolution		–	–	12	bits
	Number of channels – single-ended		–	–	No of GPIO	
	Number of channels – differential	Differential pair is formed using a pair of neighboring GPIO.	–	–	No of GPIO/2	

Parameter	Description	Conditions	Min	Typ	Max	Units
	Monotonicity [2]		Yes	–	–	
Ge	Gain error [3]	External reference	–	–	±0.1	%
VOS	Input offset voltage		–	–	±2	mV
IDD	Current consumption [2]		–	–	1	mA
	Input voltage range – single-ended [2]		Vssa	–	VDDA	V
	Input voltage range – differential [2]		Vssa	–	VDDA	V
PSRR	Power supply rejection ratio [2]		70	–	–	dB
CMRR	Common mode rejection ratio		70	–	–	dB
INL	Integral non linearity [2]	VDDA 1.71 to 5.5 V, 1 Msps, VREF 1 to 5.5 V, bypassed at ExtRef pin	–	–	+2/–1.5	LSB
		VDDA 2.0 to 3.6 V, 1 Msps, VREF 2 to VDDA, bypassed at ExtRef pin	–	–	±1.2	LSB
		VDDA 1.71 to 5.5 V, 500 ksps, VREF 1 to 5.5 V, bypassed at ExtRef pin	–	–	±1.3	LSB
DNL	Differential non linearity [2]	VDDA 1.71 to 5.5 V, 1 Msps, VREF 1 to 5.5 V, bypassed at ExtRef pin	–	–	+2/–1	LSB
		VDDA 2.0 to 3.6 V, 1 Msps, VREF 2 to VDDA , bypassed at ExtRef pin No missing codes	–	–	1.7/–0.99	LSB
		VDDA 1.71 to 5.5 V, 500 ksps, VREF 1 to 5.5 V , bypassed at ExtRef pin No missing codes	–	–	+2/–0.99	LSB
RIN	Input resistance [2]		–	180	–	kΩ

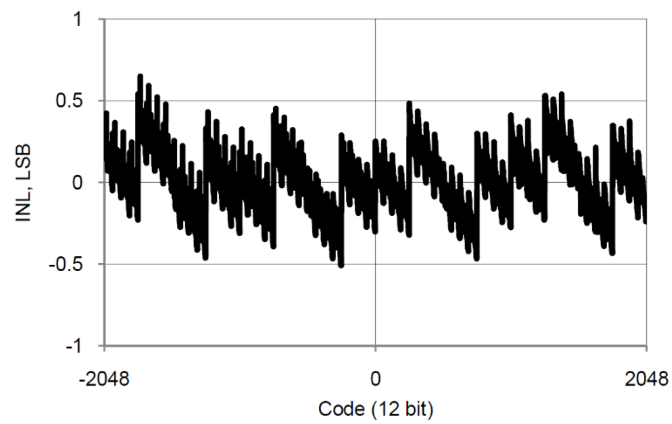
[2] Based on device characterization (Not production tested).

[3] For total analog system Idd < 5 mA, depending on package used. With higher total analog system currents it is recommended that the SAR ADC be used in differential mode.

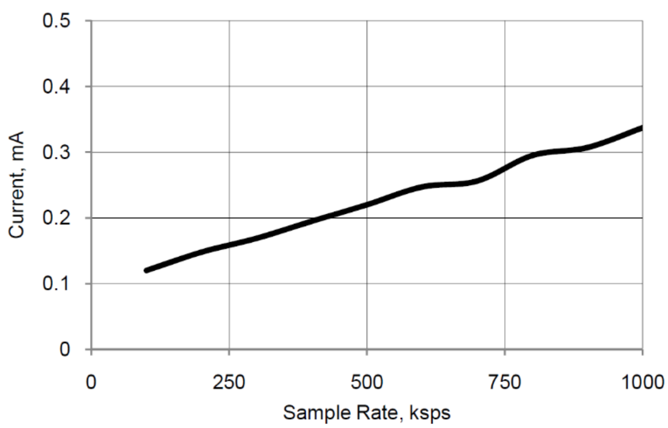
SAR ADC DNL vs Output Code
Bypassed Internal Reference Mode



SAR ADC INL vs Output Code,
Bypassed Internal Reference Mode



SAR ADC IDD vs sps, VDDA = 5 V,
Continuous Sample Mode,
External Reference Mode

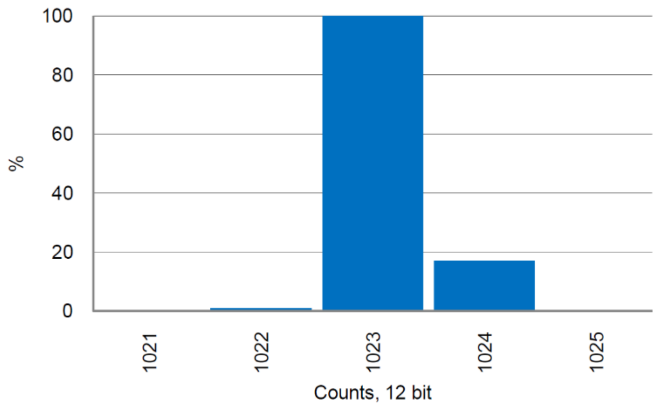


AC Specifications

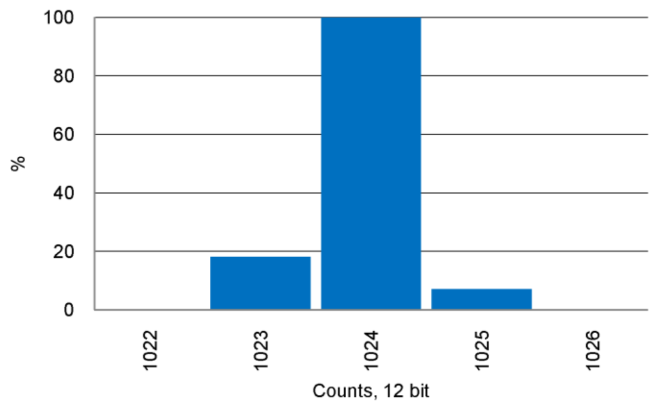
Parameter	Description	Conditions	Min	Typ	Max	Units
A_SAMP_1	Sample rate with external reference bypass cap		–	–	1	Msps
A_SAMP_2	Sample rate with no bypass cap. Reference = Internal and Input range = 0.0 +/- Vdda		–	–	500	Ksps
A_SAMP_3	Sample rate with no bypass cap. Internal reference		–	–	100	Ksps
	Startup time		–	–	10	µs

Parameter	Description	Conditions	Min	Typ	Max	Units
SINAD	Signal-to-noise ratio		68	–	–	dB
THD	Total harmonic distortion		–	–	0.02	%

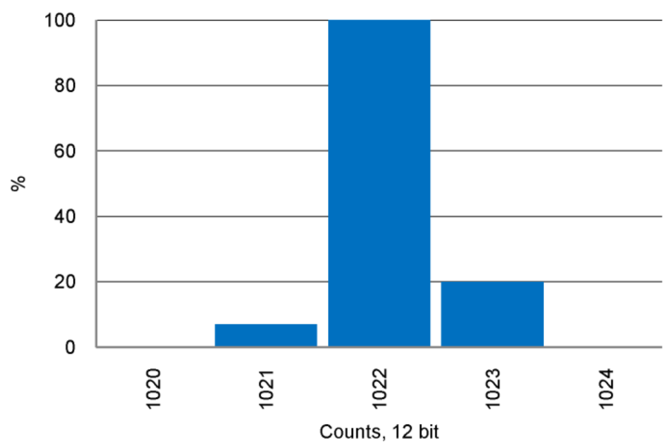
SAR ADC Noise Histogram, 100 ksps
Internal Reference No Bypass



SAR ADC Noise Histogram, 1 msps,
Internal Reference Bypassed



SAR ADC Noise Histogram, 1 msps,
External Reference



Voltage Reference Specifications

Parameter	Description	Conditions		Min	Typ	Max	Units
Vref [4]	Precision reference voltage	Initial trimming, 25 °C		1.023 (–0.1%)	1.024	1.025 (+0.1%)	V
	After typical PCB assembly, post reflow	Typical (non-optimized) board layout and 250 °C solder reflow. Device may be calibrated after assembly to improve performance.	–40 °C	–	±5	–	%
			25 °C	–	±0.2	–	%
			85 °C	–	±0.2	–	%
	Temperature drift [5]			–	–	30	ppm/°C
	Long term drift [5]			–	100	–	ppm/Khr
	Thermal cycling drift (stability) [5]			–	100	–	ppm

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
3.0.d	Minor datasheet edits.	
3.0.c	Datasheet update.	Updated Conversion Rate section with the conversion time difference in hardware trigger sample mode.
3.0.b	Datasheet update.	Added Macro Callbacks section.
3.0.a	Updated this datasheet change section to note that the ADC_SAR 3.0 clock configuration was modified.	Any project that uses the maximum number of digital clocks will fail to build after updating the ADC_SAR component from v2.10 to v3.0. This is because the ADC_SAR_theACLK internal clock uses the Digital domain by default. To resolve this problem, change the clock domain of the internal ADC_SAR_theACLK clock from Digital to Analog in the Design-Wide Resources (<*project*>*.cydwr*) file **Clocks** tab.

[4] Vref is measured after packaging, and thus accounts for substrate and die attach stresses.

[5] Based on device characterization (Not production tested).

Version	Description of Changes	Reason for Changes / Impact
3.0	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.
	Changed the conversion time from 18 to 16 cycles for Internal Vref Reference. Limited usage of the XTAL clock source. Higher than 15 MHz XTAL must be divided by 2 or greater. Updated default operational power of the ADC and trim value.	Limited configurations which violate DC/AC Specifications.
	Datasheet edits	Updated the Reference section.
2.10	Fixed possible overflow of the fixed point arithmetic in ADC_CountsTo_uVolts() API.	The issue happened when offset with minus sign was set by ADC_SetOffset() API.
	The gain calibration updated to have better resolution. Added new global variable ADC_countsPer10Volt and ADC_SetScaledGain() API which set the gain coefficient with more than 12-bit resolution.	The gain coefficient (ADC_countsPerVolt) had 10-bits resolution in Vssa to Vdda input range. This issue affected ADC_CountsTo_Volts(), ADC_CountsTo_mVolts(), and ADC_CountsTo_uVolts() API.
2.0	Added MISRA Compliance section.	The component does not have any specific deviations.
	Renamed **Triggered** sample mode to **Software Trigger**. Added **Hardware Trigger** sample mode. SOC input is present in **Software Trigger** sample mode for the backward compatible.	ADC_StartConvert() API disables the external SOC input in Triggered mode. To resolve the confusion, **Hardware Trigger** mode has been added, where ADC_StartConvert() and ADC_StopConvert() APIs do not have implementation.
	Added label that shows actual conversion rate.	The actual conversion rate may differ from desired rate based on available clock speed and divider range.
1.90	Added optional EOS output.	This signal is useful when the SAR ADC is used along with an input channel multiplexer, to switch the multiplexer as soon as possible.
	Hid the “Internal Vref, bypassed” item from the reference drop-down list when "0.0 ± Vdda" is selected as the input range. A high conversion rate is allowed for this range without the bypass capacitor.	The bypass capacitor is not needed for Vdda reference.
1.80	Added support for PSoC 5LP silicon.	
	Edited the datasheet to clarify clock frequency.	
1.71	Fixed the ADC_GetResult8() and ADC_GetResult16() APIs to perform one 16-bit read operation instead of two 8-bit reads.	The resulting data can be corrupted if the SAR ADC updates the output sampling register after one of the bytes has been read.
	Fixed the ADC_IsEndConversion() API to wait until the EOF status bit is released.	This function can return an unexpected Conversion complete status after quick consecutive calls.

Version	Description of Changes	Reason for Changes / Impact
1.70	Corrected minimum value in SampleRate error provider message.	
	Hid the “External Vref” item from the Reference drop-down list when "VDAC" is selected as Input Range.	External reference is not usable when VDAC range is selected.
	Renamed the external pin to “ExtVref” when the External Vref option is chosen. The name “Bypass” is retained when Internal reference with Bypass option is chosen.	To match the pin name with functionality.
	Datasheet corrections	
1.60	Removed the “Power” parameter from the customizer.	The higher power is set automatically depending on clock speed. The ADC_SetPower() API description contains the relation of the power from the clock rate.
	SAR operates in 12-bit mode. The 8 and 10 bit options remain but only impact the ADC_GetResult16() API.	SAR ADC only showed ODD counts as output in 8- or 10-bit Mode.
	Changed default SAR conversion rate from 1 Msps to 631579 sps (12-MHz clock).	The SAR should be able to place and build with default settings.
	The ADC_Stop() API does not power down the ADC, but reduces the power to the minimum.	PSoC 5 silicon has a defect that causes connections to several analog resources to be unreliable when not powered.
	Changed the conversion time from 18 to 19 cycles.	To improve the SAR performance.
1.50.a	Added Clock Frequency verification.	This change provides a way to avoid using the SAR ADC with an out of spec clock. If updating from version 1.10 of the SAR ADC component and using an out of working range clock, select a correct clock frequency.
	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
	Minor datasheet edits and updates	
1.50	Added Sleep/Wakeup and Init/Enable APIs.	To support low-power modes and to provide common interfaces to separate control of initialization and enabling of most components.
	Added ADC_CountsTo_Volts and ADC_CountsTo_uVolts APIs.	Extend functionality. This APIs returns the converted result in Volts and uVolts.
	Added the DMA Capabilities file to the component.	This file allows the ADC_SAR to be supported by the DMA Wizard tool in PSoC Creator.

Version	Description of Changes	Reason for Changes / Impact
	Implemented conversion of the ADC counts to the 2’s complement form in the ADC_GetResult8 and ADC_GetResult16 APIs. The same was removed from the ADC_CountsTo_mVolts function.	This change has been done for consistency with the ADC DelSig.

© Cypress Semiconductor Corporation, 2015-2017. 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, WICED, 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.