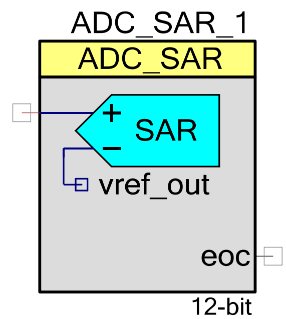
ADC Successive Approximation Register (ADC_SAR)

1.70



Features

- Supports PSoC 5 devices
- Resolution: 12-bit at 700 ksps 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 700-ksps sampling), medium-resolution (12 bits maximum) analog-to-digital conversion.

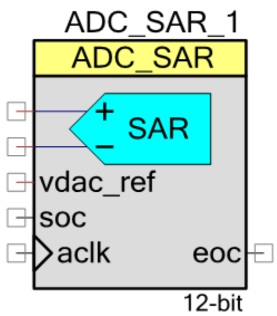
When to Use an ADC_SAR

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 various 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 minus the voltage reference. The voltage reference is either the –Input 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 the +Input minus the –Input. This pin is visible when the **Input Range** parameter is set to one of the differential modes.

vdac_ref – Input *

The VDAC reference (vdac_ref) is an optional pin. It is visible if you have selected **Vssa to VDAC*2 (Single Ended)** or **0.0 +/- VDAC (Differential)** input range; otherwise, this I/O is hidden. This pin can only be used for VDAC component output. No other signal can be connected here.

soc – Input *

The start of conversion (soc) is an optional pin. It is shown if you have selected the **Triggered** sample mode. A rising edge on this input starts an ADC conversion. If the **Sample Mode** parameter is set to **Free Running**, this I/O is hidden.

aclk – Input *

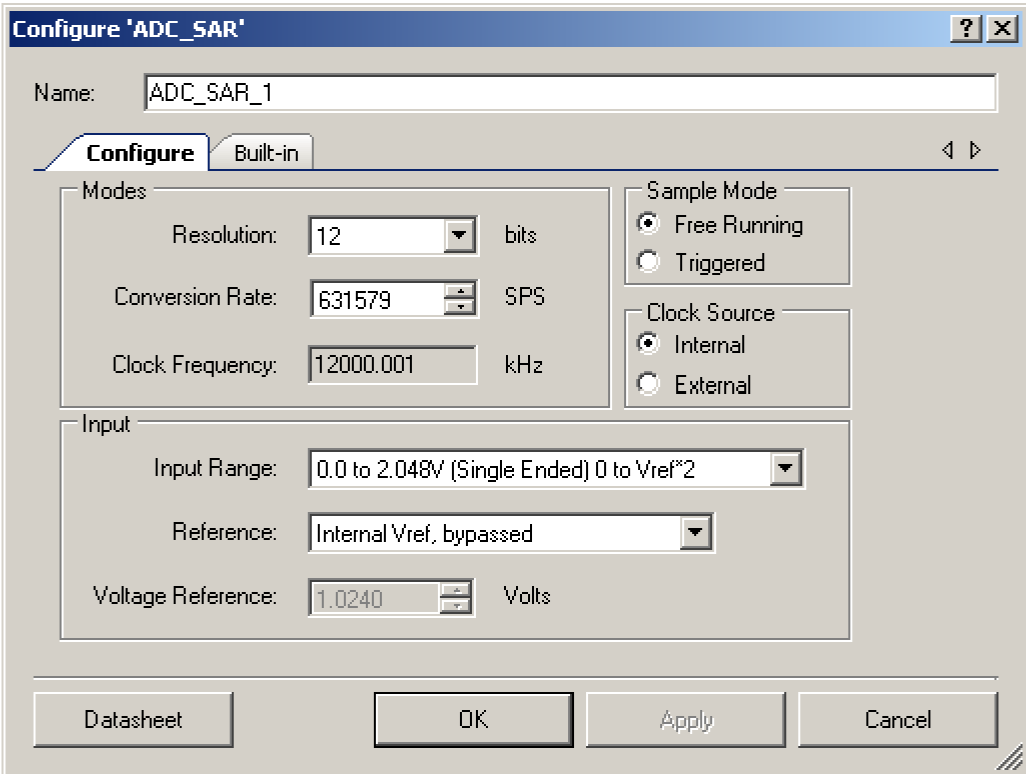
This optional pin is present if the **Clock Source** parameter is set to **External**; otherwise, the pin is not shown. This clock determines the conversion rate as a function of conversion method and resolution. If **Clock Source** is set to **Internal**, this I/O is hidden.

eoc – Output

A rising edge on the End Of Conversion (eoc) signals 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.

SAR always operates in 12-bit mode. The 8- and 10-bit options remain but only impact the ADC_GetResult16() and ADC_GetResult8() APIs.

Conversion Rate

This parameter sets the ADC conversion. The conversion time is the inverse of the conversion rate. The conversion rate is entered in samples per second. Converting one sample takes 19 cycles.

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 14 MHz. The duty cycle should be 50 percent. The minimum pulse width should be greater than 33 ns. PSoC Creator will generate an error during the build process if the clock does not fall within these limits. In that case, modify the Master Clock in the Design-Wide Resources Clock Editor.

Sample Mode

This parameter determines how the ADC operates.

Start_of_Conversion	Description
Free Running	ADC runs continuously.
Triggered	A rising-edge pulse on the SOC pin causes a single conversion to start.

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 an internal clock that is part of the ADC_SAR component.
External	Use an external clock. The clock source may 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 IC 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 the –Input connected internally to Vrefhi_out. If you are using an external reference voltage, the usable input range is 0.0 to Vref*2.

Input Range	Description
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 the -Input connected internally to Vrefhi_out.
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 the –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 the –Input to Vssa when scanning a single-ended input. An external reference can be used to provide a wider operating range. The usable input range can be calculated 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 the –Input to Vssa when scanning a single-ended input.
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 the –Input to Vssa when scanning a single-ended input
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 the –Input to Vssa when scanning a single-ended input.

Reference

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

ADC_Reference	Description
Internal Vref	Uses the internal reference. The maximum sampling rate allowed with this option is 100,000 sps. Use the **Internal Vref, bypassed** option for higher rates.
Internal Vref, bypassed	Uses internal reference; a bypass capacitor must be placed on pin P0[2]* for SAR1 or on pin P0[4]* for SAR0.
External Vref	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.

Voltage Reference

The voltage reference is used for the ADC count to voltage conversion functions discussed in the API section. This parameter is read-only when using the internal 1.024-V 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**, enter the V_{DDA} supply voltage.
- When selecting the input range **Vssa to VDAC*2** or **–Input +/- VDAC**, enter the VDAC supply voltage value.
- Voltage reference value must be between 0.15 V and V_{DDA}.

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

Placement

The ADC_SAR component is placed in one of two available SAR blocks. Placement information is provided to the API through the *cyfitter.h* file. If you need to change default placement, use the Design-Wide Resources – Directives Editor (in the project's .cydwr file) to edit the parameters.

Resources

The ADC_SAR uses a fixed block SAR in the silicon, as well as a clock source.

Resources	Resource Type				API Memory (Bytes)		Pins (per External I/O)
	Clock Dividers	Macrocells	Interrupts	SAR Fixed Blocks	Flash	RAM	
8 to 12 Bits	1	1	1	1	1106	7	1

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 cover 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."

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 offset of ADC
ADC_SetGain()	Sets ADC gain in counts per volt
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 default configuration provided with customizer
ADC_Enable()	Enables the clock and power for ADC
ADC_SaveConfig()	Saves the current user configuration
ADC_RestoreConfig()	Restores the user configuration

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_countsPerVolt	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 modified using ADC_SetGain(). 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.

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.

Note This API does not power down the ADC when using PSoC 5 silicon, but reduces the power to the minimum. This device has a defect that causes connections to several analog resources to be unreliable when the device is not powered. The unreliability manifests itself in silent failures (for example, unpredictably bad results from analog components) when the component using that resource is stopped.

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

Parameters Name	Value	Description	Clock Rate
ADC__HIGHPOWER	0	Normal power	14 MHz
ADC__MEDPOWER	1	1/2 power	7 MHz
ADC__LOWPOWER	2	1/3 power	4.6 MHz
ADC__MINPOWER	3	1/4 power	3.5 MHz

Return Value: None

Side Effects: Power setting may affect conversion accuracy.

void ADC_SetResolution(uint8 resolution)

Description: Sets the resolution for the GetResult16() and GetResult8() APIs. This function does not affect the actual conversion.

Parameters: uint8 resolution: Resolution setting

Parameters 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, you should 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 be changed 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 triggered mode, the function also acts as a software version of the SOC and every conversion must be triggered by ADC_StartConvert().

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.

Parameters: None

Return Value: None

Side Effects: In triggered 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.

Options	Description
ADC_RETURN_STATUS	Immediately returns 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 result until ADC conversion is complete.

Return Value: uint8: If a nonzero value is returned, the last conversion has completed. 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 LSB of the result is returned. This function returns a shifted value when the resolution is set to less than 12 bits. 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. This function returns a shifted value when the resolution is set to less than 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_SetGain(int16 adcGain)

Description: Sets the ADC gain in counts per volt 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 an external reference is used.

Parameters: int16 adcGain: ADC gain in counts per volt

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.

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.

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.

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 and calls ADC_SaveConfig() to save the hardware configuration.

Call the ADC_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to 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. 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 will also re-enable the component.

Parameters: None

Return Value: None

Side Effects: Calling the ADC_Wakeup() function without first calling the ADC_Sleep() or ADC_SaveConfig() function may 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 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 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.

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

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.

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 may 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 made 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_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.

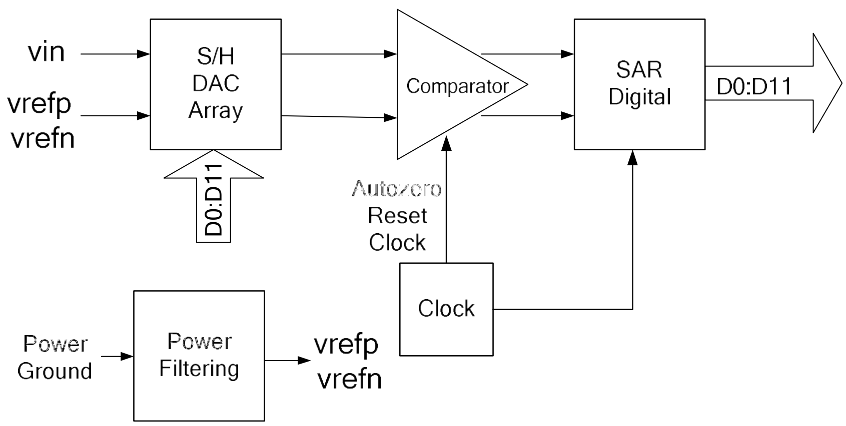
For example, at the maximum conversion rate (700 ksps at 12 bits) set the Master Clock to 53 MHz in the Design-Wide Resources Clock Editor, and optimize the ISR routine. Otherwise, the processor will not be able to handle the ISR quickly enough. If a lower Master Clock is selected, the run time of the 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 = (ADC_SAR_1_SAR_WRK1_REG << 8) | ADC_SAR_1_SAR_WRK0_REG;
        dataReady = 1;
    /* `#END`   */
}
```

Functional Description

The block diagram is shown in the following figure. 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.



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			Data[11:8]			

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

DC and AC Electrical Characteristics

The following values indicate expected performance and are based on initial characterization data. Unless otherwise specified, operating conditions are:

- Operation in continuous sample mode
- Fclk = 14 MHz
- Input range = ±V_{REF}
- Bypass capacitor of 10 µF

SAR ADC DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Resolution		8	–	12	bits
	Number of channels – single-ended		–	–	No. of GPIO	–
	Number of channels – differential	Differential pair is formed using a pair of GPIOs.	–	–	No. of GPIO/2	–
	Monotonicity[1]		Yes	–	–	
Ge	Gain error	External reference	–	–	±0.2	%
V_{OS}	Input offset voltage	V_{CM} = 0 V	–	–	±2	mV
		$V_{CM} = V_{DD}/2$	–	–	±6	
I_{DD}	Current consumption		–	–	1	mA
	Input voltage range – single-ended [1]		V_{SSA}	–	V_{DDA}	V
	Input voltage range – differential [1]		V_{SSA}	–	V_{DDA}	V
PSRR	Power supply rejection ratio [1]		70	–	–	dB
CMRR	Common mode rejection ratio		35	–	–	dB
INL	Integral nonlinearity [1]	Internal reference from V_{BG}	–	–	±2	LSB
DNL	Differential nonlinearity [1]	Internal reference from V_{BG}	–	–	±2	LSB

[1] Based on device characterization (not production tested).

SAR ADC AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Sample rate [2]	With bypass capacitor	–	–	700	ksps
		Without bypass capacitor	–	–	100	
	Startup time [2]		–	–	10	µs
SINAD	Signal-to-noise ratio [2]	$V_{DDA} \leq 3.6$ V, $V_{REF} \leq 3.6$ V	57	–	–	dB
		3.6 V < $V_{DDA} \leq 5.5$ V V_{REF} < 1.3 V or V_{REF} > 1.8 V	57	–	–	
THD	Total harmonic distortion [2]	$V_{DDA} \leq 3.6$ V, $V_{REF} \leq 3.6$ V	–	–	0.1	dB
		3.6 V < $V_{DDA} \leq 5.5$ V V_{REF} < 1.3 V or V_{REF} > 1.8 V	–	–	0.1	

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

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.70	Corrected minimum value in SampleRate error provider message.	
	Hid “External Vref” item from Reference drop-down list when "VDAC" is selected as Input Range.	External reference is not usable when VDAC range is selected.
	External pin renamed 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 “Power” parameter from customizer.	The higher power has been 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.

Version	Description of Changes	Reason for Changes / Impact
	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, as well as 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 converted result in Volts and uVolts.
	Added DMA Capabilities file to the component.	This file allows the ADC_SAR to be supported by the DMA Wizard tool in PSoC Creator.
	Conversion of the ADC counts to the 2's complement form has been implemented in the ADC_GetResult8 and ADC_GetResult16 APIs. The same removed from ADC_CountsTo_mVolts function.	This change has been done for consistency with the ADC DelSig.