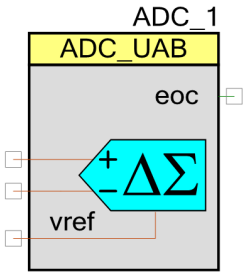
Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

Incremental ADC (ADC_UAB) 2.0

Features



- Single sample incremental ADC
- Rail to rail input voltage capability
- Full scale ranges from ±Vdda down to ±150 mV
- Data scaling, saturation, and calibration managed by API functions

General Description

The Incremental ADC (ADC_UAB) Component is a general purpose retriggerable ADC that can provide results with a resolution of 14 bits and a conversion time of 2.56 ms.

Results are returned through API function calls that perform scaling, saturation, and calibration tasks.

When to Use an Incremental ADC

The ADC_UAB is useful for making slow, irregular, or asynchronous measurements of system voltages.

Input/Output Connections

This section describes the various input and output connections for the ADC_UAB Component.

Terminal Name	I/O Type	Description
eoc	Digital Output	This output can be used to trigger the transfer of data once the conversion is complete. This terminal is always present.
+Input	Analog Input	This input (not marked; it is always the upper terminal of the differential input pair on the symbol) is the 'positive' (also called non-inverting) analog signal input to the ADC.
-Input	Analog Input	This input (not marked; it is always the lower terminal of the differential input pair on the symbol) is the 'negative' (also called inverting) analog signal input to the ADC.
vref	Analog Input	This input is the reference for the ADC. It can be connected to an internal reference using the Vref or PVref components or to a pin. **Note:** The voltage on this input combined with the Input range parameter must not exceed the operating range of the component.

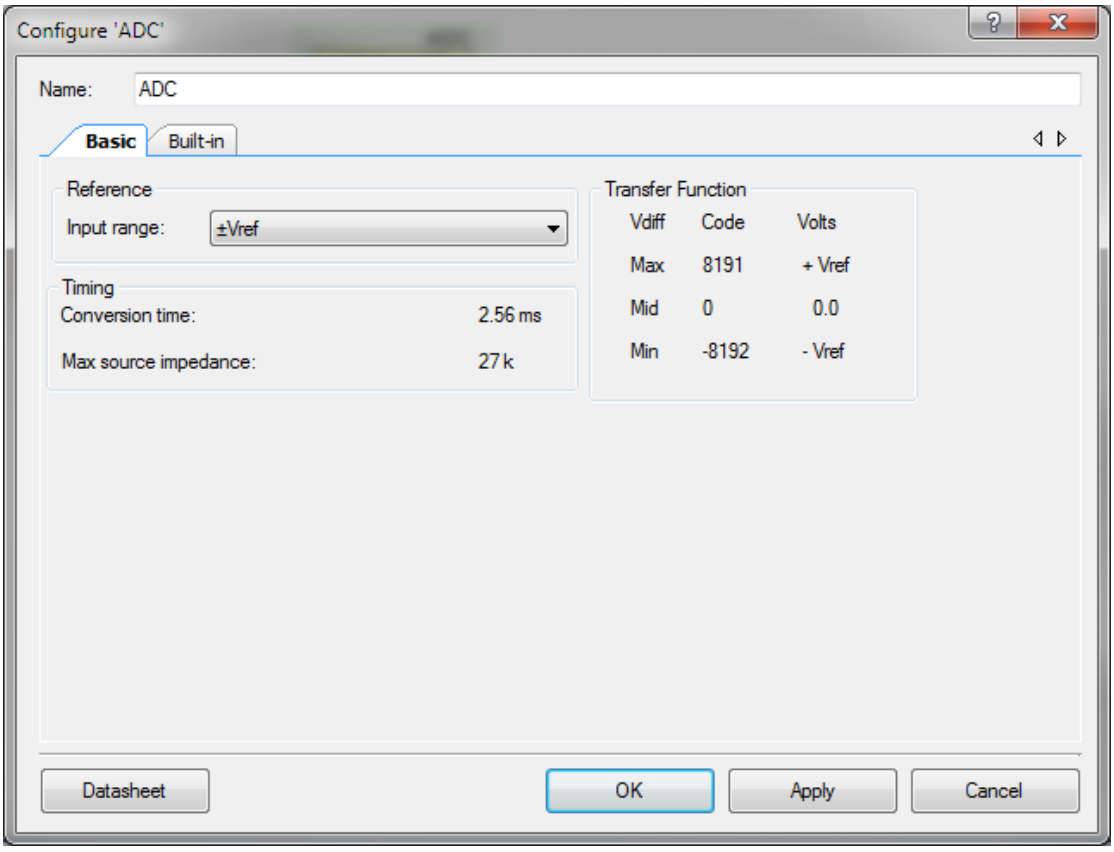
Component Parameters

This section covers the various parameters that can be altered or inspected through the Component Configure dialog, grouped within a series of tabs. To explore this, drag an Incremental ADC onto your design

and double-click it to open the Configure dialog.

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

Basic Tab



Input range

Some input ranges are expressed as a fraction or multiple of the voltage on the vref terminal.

Range selections mode

- ±Vref/4
- ±Vref/2
- **±Vref**
- ±2Vref
- ±4Vref

In differential operation, full code range can be obtained if the voltage difference between the inputs meets the range criterion and both input voltages are between -0.15 V and Vdda.

Conversion time (display only)

This field displays the total conversion time in milliseconds. This is the time from calling the StarConvert() function to returning a calibrated value from the GetResult16() function in the ISR.

Maximum source impedance (display only)

This field displays the maximum resistive source impedance to ensure specified accuracy.

Transfer Function (display only)

This section displays the converter transfer functions relating output data to input voltage (for data returned by the API function call, not the raw data).

Application Programming Interface

By default, PSoC Creator assigns the instance name **ADC_1** to the first instance of the ADC_UAB 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 section is **ADC**.

This Component's API contains the following sections:

- Initialization
- General
- Interrupt Service Routine
- Power
- Global Variables
- Global Constants
- Code Examples and Applications
- API Memory Usage

Initialization

Functions that are used for starting and stopping the Component.

Functions

- void ADC_Start (void)
- uint32 ADC_StartConvert (void)
- void ADC_Stop (void)
- void ADC_Init (void)
- void ADC_Enable (void)

Function Documentation

void ADC_Start (void)

Performs all required initialization for this component, and enables the power. The power is set to the appropriate level based on the clock frequency. Does not start a conversion. This function calls ADC_-Calibrate which performs two conversions. Double the conversion time reported in the GUI to determine how long this function will require to complete.

uint32 ADC_StartConvert (void)

Begins a conversion.

Return values

uint32	The function returns cystatus of its operation. ■ CYRET_STARTED - The ADC is already converting. No new conversion started. ■ CYRET_SUCCESS - A new conversion has started.

void ADC_Stop (void)

Disables underlying hardware. Terminates conversion if there is a conversion in progress.

void ADC_Init (void)

Configures hardware and data structures according to component parameters. Does not power hardware. Called from ADC_Start(). ADC_Start() is the preferred method for initializing the component. The ISR is set to the ADC_ConversionInt() here. During calibration, the ISR is temporarily set to the ADC_CalibrateInt() and then restored to ADC_ConversionInt().

void ADC_Enable (void)

Enables underlying hardware. Does not start a conversion. Called from ADC_Start(). ADC_Start() is the preferred method for initializing the component.

General

General APIs that are used for run-time configuration of the Component during active power mode.

These include reading from registers, and writing to registers.

Functions

- uint32 ADC_IsEndConversion (ADC_eoc_enum retMode)
- uint32 ADC_Calibrate (ADC_calibrate_mode_enum calibrateMode)
- int16 ADC_RawToResult16 (int32 raw)
- int16 ADC_CountsTo_mVolts (int32 counts, uint32 vrefMillivolts)
- int32 ADC_CountsTo_uVolts (int32 counts, uint32 vrefMicrovolts)
- float32 ADC_CountsTo_Volts (int32 counts, float32 vrefVolts)
- int32 ADC_GetRaw (void)
- int16 ADC_GetResult16 (void)

Function Documentation

uint32 ADC_IsEndConversion (ADC_eoc_enum retMode *)*

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 or calibrate status. If the value returned is non-zero, the conversion is not complete, and this function should be retried until a result of zero is returned.
ADC_WAIT_FOR_RESULT	Does not return a result until the ADC conversion is complete.

Return values

uint32	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.

uint32 ADC_Calibrate (ADC_calibrate_mode_enum calibrateMode *)*

Configures hardware for calibration. The status of the calibration is either returned immediately or is blocked until the calibration completes, depending on the calibrateMode input.

Parameters

calibrateMode	Check calibration return mode. See the following table for options.

Options	Description
ADC_USE_INTERRUPT	Calibration will be interrupt-driven. Will enter calibration interrupt twice, then set ADC_calStatus to IDLE. Other tasks may be performed while waiting for calibration to complete.
ADC_WAIT_FOR_CALIBRATE	Does not return a result until the ADC calibration is complete.

Return values

uint32	The status of the calibration. If nonzero, calibration was successful.

Side Effects

Sets the UAB interrupt vector to ADC_CalibrateInt during calibration. Sets it back to ADC_ConversionInt when finished.

int16 ADC_RawToResult16 (int32 raw *)*

For given raw input, returns the calibrated ADC result. This result is saturated at the lower and upper limits of the operating resolution. The calibration formula is:

(((raw - calibrateHigh) ∗ (MAX_HIGH - MAX_LOW)) / (calibrateHigh - calibrateLow)) + MAX_LOW.

Parameters

raw	The direct output of the decimator register.

Return values

int16	Saturated, calibrated ADC result.

int16 ADC_CountsTo_mVolts (int32 counts, *uint32* vrefMillivolts *)*

For given counts input, returns the voltage relative to Vneg. For example, if the input voltage to the ADC was 0.534 volts, the return value would be 534. The equation for the result is:

result (mV) = counts $*$ (reference/(Resolution/2)) $*$ gain.

Resolution is 14-bit.

Note that the multiplication of counts (14-bit max), reference (14-bit max) and gain (4 max) results in a maximum of of 30-bits. An increase in any of these parameters will cause the result to overflow 32 bits.

Parameters

counts	A calibrated ADC result.
vrefMillivolts	applied to the vref terminal of the ADC in millivolts. Note: This parameter must be a positive number or this API will only return zero.

Return values

int16	The voltage relative to Vneg in units of millivolts.

int32 ADC_CountsTo_uVolts (int32 counts, *uint32* vrefMicrovolts *)*

For given counts input, returns the voltage relative to Vneg. For example, if the ADC measured 0.534 volts, the return value would be 534000. The equation for the result is:

result (μV) = counts $*$ (reference/(Resolution/2)) $*$ gain.

Parameters

counts	A calibrated ADC result.
vrefMicrovolts	applied to the vref terminal of the ADC in microvolts. Note: This parameter must be a positive number or this API will only return zero.

Return values

int32	The voltage relative to Vneg in units of microvolts.

float32 ADC_CountsTo_Volts (int32 counts, *float32* vrefVolts *)*

For given counts input, returns the voltage relative to Vneg. For example, if the ADC measured 0.534 volts, the return value would be 0.534. The equation for the result is:

result (V) = counts ∗ (reference/(Resolution/2)) ∗ gain.

Parameters

counts	A calibrated ADC result.
vrefVolts	applied to the vref terminal of the ADC in Volts. Note: This parameter must be a positive number or this API will only return zero.

Return values

float32	The voltage relative to Vneg in units of volts.

int32 ADC_GetRaw (void)

Get a raw result value from the ADC's decimator output register.

Return values

int32	Uncalibrated contents of decimator's result register.

int16 ADC_GetResult16 (void)

Returns the calibrated ADC result. This result is saturated at the lower and upper limits of the 14-bit operating resolution.

Return values

int16	Saturated, calibrated ADC result.

Interrupt Service Routine

The ADC generates an interrupt before and after calibration and conversion.

Call ADC_EnableIrq() to enable Component interrupts.

Functions

- void ADC_EnableIRQ (void)
- void ADC_DisableIRQ (void)
- void ADC_ConversionInt_EntryCallback (void)
- void ADC_ConversionInt_ExitCallback (void)
- void ADC_CalibrateInt_EntryCallback (void)

- void ADC_CalibrateInt_ExitCallback (void)
- void ADC_ClearInterrupt (void)

Function Documentation

void ADC_EnableIRQ (void)

Enables interrupt after a conversion finishes. Conversion entry and exit callbacks should be defined in *cyapicallbacks.h*.

void ADC_DisableIRQ (void)

Disables interrupt after a conversion finishes.

void ADC_ConversionInt_EntryCallback (void)

Conversion interrupt entry callback. Perform a task before ADC_StartConvert() takes place.

Enabled by defining ADC_CONVERSIONINT_ENTRY_CALLBACK in the automatically generated *cyapicallbacks.h* file, and defining this function.

Conversion interrupts are only called when interrupts are enabled (see ADC_EnableIRQ()).

void ADC_ConversionInt_ExitCallback (void)

Conversion interrupt exit callback. Perform a task after ADC_StartConvert() completes.

Enabled by defining ADC_CONVERSIONINT_EXIT_CALLBACK in the automatically generated *cyapicallbacks.h* file, and defining this function.

Conversion interrupts are only called when interrupts are enabled (see ADC_EnableIRQ()).

void ADC_CalibrateInt_EntryCallback (void)

Calibrate interrupt entry callback. Perform a task before **each** calibration phase (ADC_Calibrate() has two phases).

Enabled by defining ADC_CALIBRATEINT_ENTRY_CALLBACK in the automatically generated *cyapicallbacks.h* file, and defining this function.

Calibrate interrupts are called every time ADC_Calibrate() is called, even if ADC_EnableIRQ() hasn't been called.

void ADC_CalibrateInt_ExitCallback (void)

Calibrate interrupt exit callback. Perform a task after **each** calibration phase (ADC_Calibrate() has two phases).

Enabled by defining ADC_CALIBRATEINT_EXIT_CALLBACK in the automatically generated *cyapicallbacks.h* file, and defining this function.

Calibrate interrupts are called every time ADC_Calibrate() is called, even if ADC_EnableIRQ() hasn't been called.

void ADC_ClearInterrupt (void)

Clear a pending interrupt. Sets the ADC_HALF_A_INTR_REG register to ADC_DCM_INTR_MASK.

Power

Functions used to control power consumption of the ADC

The following power control functions are available in the Component

Functions

- void ADC_Sleep (void)
- void ADC_Wakeup (void)

Function Documentation

void ADC_Sleep (void)

Prepare the component for entering deep sleep. The underlying hardware requires a high speed clock, so it will not run in deep sleep. Do not call this function before normal sleep (cpu off). For normal sleep, a conversion will continue operating as normal.

void ADC_Wakeup (void)

Restore the component to its pre-deep sleep condition. Should be called just after awaking from deep sleep.

Global Variables

Global variables used in the Component.

Globals are noted in the description of the functions that use globals.

Variables

- uint8 ADC_initVar
- uint8 ADC_irqEnabled

Variable Documentation

uint8 ADC_initVar

Indicates whether the ADC has been initialized. Set by ADC_Start().

uint8 ADC_irqEnabled

Indicates whether the ADC uses interrupts. Set by ADC_EnableIRQ(), cleared by ADC_DisableIRQ().

Global Constants

Global constants used in the Component.

Variables

- ADC_PARAM_VREF_VOLTAGE_MV
- ADC_PARAM_VREF_VOLTAGE_UV_SCALEDOWN
- ADC_PARAM_VREF_VOLTAGE_V

Variable Documentation

ADC_PARAM_VREF_VOLTAGE_MV

Full-scale center voltage, in millivolts.

ADC_PARAM_VREF_VOLTAGE_UV_SCALEDOWN

Full-scale center voltage, in microvolts.

ADC_PARAM_VREF_VOLTAGE_V

Full-scale center voltage, in volts.

Code Examples and Applications

No code examples are available with this initial release.

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 given Component configuration.

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

PSoC Analog Coprocessor (GCC)

Configuration	Flash Usage (Bytes)	SRAM Usage (Bytes)
Typical	1976	318
All APIs	2202	318

PSoC Analog Coprocessor Resources

The Incremental ADC Component uses the following device resources:

Resource Type	Used	Free
Interrupts	1	25
IO	3	35
Decimator	1	1
UAB Channels	2	0
Voltage References	1	3

Functional Description

The ADC has two unbuffered switched-capacitor inputs that connect to the pair of voltage signals to be sampled differentially.

Both inputs to the ADC can be taken up to 150 mV more positive than Vdda or 150 mV more negative than Vssa.

The 1-bit output of the modulator is decimated by a dedicated filter. The result of that process is further adjusted for scale factor and offset, shifted and saturated in the supplied API functions in order to deliver a final result.

The ADC Hardware makes one complete conversion and then stops.

Raw digital output from the ADC hardware can be accessed through DMA. It does not have any direct use since it has not been calibrated and may have saturated the 14-bit resolution. Fitting of the data into the chosen resolution, including calibration and saturation management, is achieved in the function ADC_RawToResult16(), which returns the desired result in a signed 16-bit integer variable. ADC_RawToResult16() can also post-process blocks of data stored in SRAM.

Capacitor values affect source impedance

The capacitor values and the clock rate are used to determine the maximum permissible source resistance. This is communicated in the customizer.

Each input connection to the ADC has an apparent resistive input impedance of approximately 1MΩ (under the default clocking arrangements), terminated at the reference potential.

Mismatches in the source impedance in differential mode can cause degradation of CMRR.

Calibration procedure

ADC_Calibrate() triggers a two-point calibration, which results in two calibration constants that are used to correct the offset and scale factor of the conversion results.

The final output is a valid (i.e. not wrapped) signed 14-bit representation of the input voltage, as long as the input voltage is within the input range of the ADC.

Calibration is done at the base 'unity gain' configuration, using differential input voltage of +reference and -reference. The modulator capacitors are then returned to the target values as set by the user's range selection, and the results are scaled appropriately.

This form of calibration does not correct for any error in the reference voltage itself, only in the transfer function that relates digital code to the input voltage as a fraction of the reference voltage.

Industry Standards

This section lists the industry standard compliance of the ADC_UAB Component.

MISRA Compliance

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

Deviation Type	Description

Deviation Type	Description
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 the Component-specific deviations. The 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_UAB Component has the following specific deviations:

Rule	Class	Description	Description of Deviation(s)
3.1	R	All usage of implementation-defined behaviour shall be documented.	Embedded Component, UABPRIM, source code has comments containing one of the following characters: '$', '@', or "".
19.7	A	A function should be used in preference to a function-like macro.	UABPRIM source code uses function-like macros to take generic functions and rename them for specific use cases and use predefined parameters making the API easier to use. Also, there are read-modify-write macros that are in most UABPRIM API functions and are used to improve readability of the code so that the intent is clearly understood. The macros have proper parenthesis shielding the parameters as well as the whole macro.

DC and AC Electrical Characteristics

Characterization data of the ADC Component is pending.

Component Changes

This section lists the changes in the ADC_UAB Component from the previous versions.

Version	Description of Changes	Reason for Changes / Impact
2.0	New architecture for improved performance.	Component only supports differential mode and uses an external references. New range of 4Vref available, deprecated range of Vref/8.
1.10	Update UABPRIM version to 1.20	Update UABPRIM to latest version
1.0.b	Added link to Component web page from Symbol	Link missing in prior versions
1.0.a	Edited datasheet	Updated Component symbol to show as Prototype

Version	Description of Changes	Reason for Changes / Impact
1.0	Initial Version	

© Cypress Semiconductor Corporation, 2018. 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.