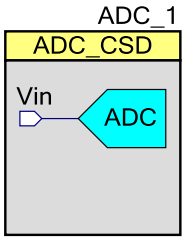
PSoC 4 10-Bit ADC (CSD)

4.0

Features



- Selectable 8- or 10-bit resolution
- Sample rates of up to 11.6 ksps with 10-bit resolution
- Input measurement range from VSSA to VDDA on any GPIO pin

General Description

The PSoC 4 single-slope 10-bit ADC (CSD) Component is a triggered analog to digital converter. This Component allows you to configure the initial state and provides APIs to control the Component from application firmware. This datasheet includes the following sections:

- *Component Configuration Parameters* – Contains descriptions of the Component's parameters in the configuration wizard.
- *Application Programming Interface* – Provides descriptions of all APIs in the firmware library, as well as descriptions of all data structures (Register map) used by the firmware library.
- *DC and AC Electrical Characteristics* – Provides the Component performance specifications and other details such as certification specifications.

Note The ADC operation is dependent on a high-frequency (system clock) input to the block. Changing the clock frequency during run-time will impact ADC operation, and the ADC may not operate as expected.

Input / Output Connections

This Component's terminals are not exposed as connectable terminals on the symbol. However, the AdcInput terminal can be assigned to the port pins in the PSoC Creator Design-Wide Resources Pin Editor. The Pin Editor provides the guidelines on the recommended pins for each terminal and does not allow an invalid pin assignment.

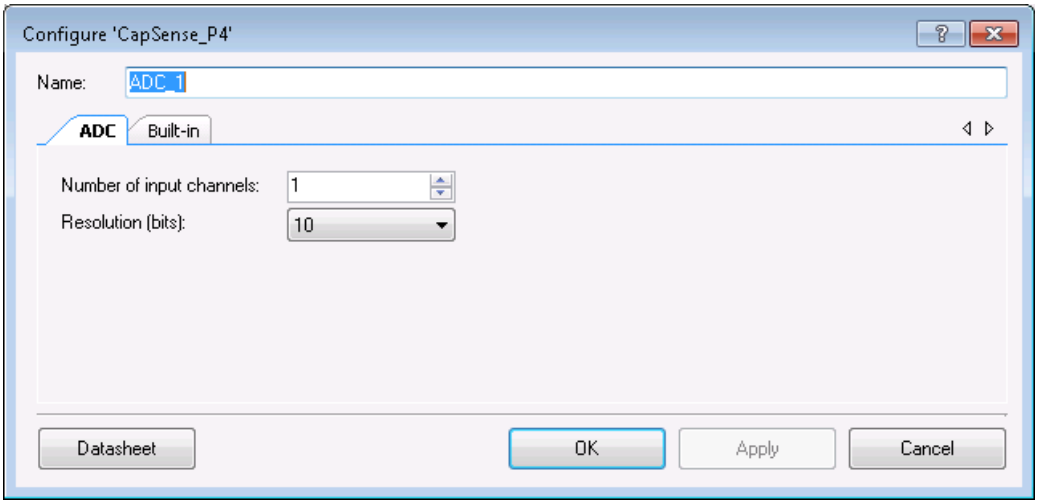
Name	I/O Type	Description
AdcInput	Analog	ADC voltage inputs. The number of inputs is set by the Component parameter.

Component Configuration Parameters

Drag a Component onto the design canvas and double-click to open the dialog.

ADC Tab

The parameters in this tab apply to the ADC functionality.



Name	Description
Number of input channels	Increment/decrement this value to specify the total input channels for the ADC. The range of valid values is 1-10.
Resolution (bits)	This drop-down is used to select the ADC resolution. The possible options are: ▪ 8 bits ▪ 10 bits

Application Programming Interface

Application Programming Interface (API) routines allow you to control and execute specific tasks using the Component firmware. The following sections list and describe each function and dependency.

By default, the instance name of the Component is “ADC_1” for the first instance of a Component in a given design. You can rename it to any unique text that follows the syntactic rules for identifiers. Every API function, variable, and constant name contains the instance name prefix. For readability, this section assumes “ADC” as the instance name.

ADC Application Public Interface

Description

The ADC application public interface represents the abstraction layer of the ADC function. The ADC public interface is exposed to the user to implement the ADC function.

If ADC is not configured then ADC-related functions are not available.

Functions

- void ADC_Start(void)
  Configures the hardware and performs calibration.
- void ADC_Sleep(void)
  Prepares the component for deep sleep.
- void ADC_Wakeup(void)
  This function resumes the component after sleep.
- cystatus ADC_StartConvert(uint8 chId)
  Initializes the hardware and initiates an analog-to-digital conversion on the selected input channel.
- uint8 ADC_IsBusy(void)
  The function returns the status of the ADC's operation.
- uint16 ADC_ReadResult_mVolts(uint8 chId)
  This is a blocking API. It initiates a conversion, waits for completion and returns the result.
- uint16 ADC_GetResult_mVolts(uint8 chId)
  This API does not perform an ADC conversion and returns the last valid result for the specified channel.
- cystatus ADC_Calibrate(void)
  Performs calibration of the ADC module.
- void ADC_Stop(void)
  Disables the hardware sub-blocks that are in use while in the ADC mode, and frees the routing.
- void ADC_Resume(void)
  Resumes the ADC operation after a stop call.

Function Documentation

void ADC_Start (void)

Configures the hardware and performs calibration.

Go to the top of the ADC Application Public Interface section.

void ADC_Sleep (void)

Currently this function is empty and exists as a place for future updates, this function shall be used to prepare the component to enter deep sleep.

Go to the top of the ADC Application Public Interface section.

void ADC_Wakeup (void)

Currently this function is empty and exists as a place for future updates, this function shall be used to resume the component after exiting deep sleep.

Go to the top of the ADC Application Public Interface section.

cystatus ADC_StartConvert (uint8 *chId*)

Initializes the hardware and initiates an analog-to-digital conversion on the selected input channel. This API only initiates a conversion and does not wait for the conversion to be completed, therefore the ADC_IsBusy() API must be used to check the status and ensure that the conversion is complete prior to reading the result, starting a new conversion with the same or a different channel or reconfiguring the hardware for different functionality.

Parameters:

chId	The ID of the channel to be converted.

Returns:

The function returns cystatus of its operation.

- CYRET_SUCCESS - A conversion has started.
- CYRET_LOCKED - The hardware is already in-use by a previously initialized conversion or other functionality. No new conversion is started by this API.
- CYRET_BAD_PARAM - An invalid channel Id. No conversion is started.

Go to the top of the ADC Application Public Interface section.

uint8 ADC_IsBusy (void)

The function returns the status of the ADC's operation. A new conversion or calibration must not be started unless the ADC is in the IDLE state.

Returns:

The function returns the status of the ADC's operation.

- ADC_STATUS_IDLE - The ADC is not busy, a new conversion can be initiated.
- ADC_STATUS_CONVERTING - A previously initiated conversion is in progress.
- ADC_STATUS_CALIBPH1 - The ADC is in the first phase (of 3) of calibration.
- ADC_STATUS_CALIBPH2 - The ADC is in the second phase (of 3) of calibration.
- ADC_STATUS_CALIBPH3 - The ADC is in the third phase (of 3) of calibration.
- ADC_STATUS_OVERFLOW - The most recent measurement caused an overflow. The root cause of the overflow may be the previous calibration values being invalid or the VDDA setting in cydwr and hardware do not match. Perform re-calibration or set the appropriate VDDA value in cydwr to avoid this error condition.

Go to the top of the ADC Application Public Interface section.

uint16 ADC_ReadResult_mVolts (uint8 *chId*)

This is a blocking API. Internally, it starts a conversion using ADC_StartConvert(), checks the status using ADC_IsBusy(), waits until the conversion is completed and returns the result.

Parameters:

chId	The ID of the channel to be measured

Returns:

The function returns voltage in milli-volts or ADC_VALUE_BAD_RESULT if:

- chId is invalid
- the ADC conversion is not started
- the ADC conversion watch-dog triggered.

Go to the top of the ADC Application Public Interface section.

uint16 ADC_GetResult_mVolts (uint8 *chId*)

Returns the last valid result from the data structure for the specified channel. This function can be used to read a previous result of any channel even if the ADC is busy or a conversion is in progress. However, it is highly recommended not to use this function with a channel that is in an active conversion.

Parameters:

chId	The ID of the channel to be measured

Returns:

The function returns a voltage in milli-volts or ADC_VALUE_BAD_CHAN_ID if chId is invalid.

Go to the top of the ADC Application Public Interface section.

cystatus ADC_Calibrate (void)

Performs calibration for the ADC to identify the appropriate hardware configuration to produce accurate results. It is recommended to run the calibration periodically (for example every 10 seconds) for accuracy and compensations.

Returns:

The function returns cystatus of its operation.

- CYRET_SUCCESS - The block is configured for the ADC use.
- CYRET_LOCKED - The hardware is already in-use by a previously initialized conversion or other functionality. No new conversion is started by this API.

Go to the top of the ADC Application Public Interface section.

void ADC_Stop (void)

This function stops the component operation, no ADC conversion can be initiated when the component is stopped. Once stopped, the hardware block may be reconfigured by the application program for any other special usage. The ADC operation can be resumed by calling the ADC_Resume() function or the component can be reset by calling the ADC_Start() function. This function should be called when no ADC conversion is in progress.

Go to the top of the ADC Application Public Interface section.

void ADC_Resume (void)

This function resumes the ADC operation if the operation is stopped previously by the ADC_Stop() API.

Go to the top of the ADC Application Public Interface section.

Interrupt Service Routine

Description

The ADC component uses an interrupt to complete measurement.

After the measurement is complete, the ISR copies the measured channel voltage to the Data Structure.

The Component implementation avoids using critical sections in the code. In an unavoidable situation, the critical section is used and the code is optimized for the shortest execution time.

The ADC component does not alter or affect the priority of other interrupts in the system.

These APIs should not be used in the application layer.

Functions

- CY_ISR(ADC_IntrHandler)
  This is an internal ISR function for the ADC implementation.

Function Documentation

CY_ISR (ADC_IntrHandler)

This ISR is triggered after a measurement completes or during the calibration phases.

To use the entry or exit callbacks, define ADC_[ENTRY|EXIT]_CALLBACK and define the corresponding function, ADC_[Entry|Exit]Callback().

Go to the top of the Interrupt Service Routine section.

Macro Callbacks

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

In order to add the 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 using the provided in the table name (in cyapicallbacks.h). This will make this function visible to all the project files.
- Write the function implementation (in any user file).

ADC Macro Callbacks

Macro Callback Function Name	Associated Macro	Description
ADC_EntryCallback	ADC_ENTRY_CALLBACK	Used at the beginning of the ADC interrupt handler to perform additional application-specific actions
ADC_ExitCallback	ADC_EXIT_CALLBACK	Used at the end of the ADC interrupt handler to perform additional application-specific actions

Global Variables

Description

The section documents the ADC component related global Variables.

The ADC component stores the component configuration and scanning data in the data structure. Refer to the Data Structure section for details of organization of the data structure.

Variables

- ADC_RAM_STRUCT ADC_dsRam

Variable Documentation

ADC_RAM_STRUCTADC_dsRam

The variable that contains the ADC configuration, settings and scanning results. ADC_dsRam represents RAM Data Structure.

API Constants

Description

The section documents the ADC component related API Constants.

Variables

- const ADC_FLASH_IO_STRUCT ADC_adcIoList[ADC_ADC_TOTAL_CHANNELS]

Variable Documentation

const ADC_FLASH_IO_STRUCT ADC_adcIoList[ADC_ADC_TOTAL_CHANNELS]

The array of the pointers to the ADC input channels specific register.

Data Structure

Description

This section provides the list of structures/registers available in the component.

Data Structures

- struct ADC_RAM_STRUCT
  Declares the top-level RAM Data Structure.
- struct ADC_FLASH_IO_STRUCT
  Declares the Flash IO object.

Data Structure Documentation

struct ADC_RAM_STRUCT

Go to the top of the Data Structures section.

Data Fields:

uint16	adcResult[ADC_ADC_TOTAL_CHANNELS]	Stores the latest ADC result for the channel. The array size is equal to the number of ADC channels used in the project.
uint16	adcCode[ADC_ADC_TOTAL_CHANNELS]	Stores the latest ADC conversion result for the channel. The array size is equal to the number of ADC channels used in the project.
uint8	adcStatus	Stores the status of ADC.
uint8	adcIdac	ADC IDAC
uint8	adcResolution	Stores the ADC resolution.
uint8	adcAzTime	Stores the AZ time used for ADC conversion.

struct ADC_FLASH_IO_STRUCT

Go to the top of the Data Structures section.

Data Fields:

reg32 *	hsiomPtr	Pointer to the HSIOM configuration register of the IO.
reg32 *	pcPtr	Pointer to the port configuration register of the IO.
reg32 *	drPtr	Pointer to the port data register of the IO.
reg32 *	psPtr	Pointer to the pin state data register of the IO.
uint32	hsiomMask	IO mask in the HSIOM configuration register.
uint32	mask	IO mask in the DR and PS registers.
uint8	hsiomShift	Position of the IO configuration bits in the HSIOM register.
uint8	drShift	Position of the IO configuration bits in the DR and PS registers.
uint8	shift	Position of the IO configuration bits in the PC register.

Memory Usage

The Component Flash and RAM memory usage varies significantly depending on the compiler, device, number of APIs called by the application program and Component configuration. The table below provides the total memory usage of firmware for given Component configuration.

The measurements were done with an 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.

PSoC 4 (GCC)

The following Component configuration is used to represent the memory usage:

Configuration	Memory Consumption	
	Flash	SRAM
ADC only: *Resolution (bits)* = 10-bit / *Number of input channels* = 10	<2500	<100

MISRA Compliance Report

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

- Project deviations – applicable for all PSoC Creator Components
- Specific deviations –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 10-bit ADC Component has the following specific deviations:

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
8.8	R	An external object or function shall be declared in only one file.	Some arrays are generated based on the Component configuration and these arrays are declared locally in the .c source files where they are used instead of in .h include files.
12.13	A	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.	These violations are reported for the GCC ARM optimized form of the “for” loop that have the following syntax: for(index = COUNT; index --> 0u;) It is used to improve performance.

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
14.2	R	All non-null statements shall either have at least one side effect however executed, or cause the control flow to change.	These violations are caused by expressions suppressing the C-compiler warnings about the unused function parameters. This Component has many different configurations. Some of them do not use specific function parameters. To avoid the complier's warning, the following code is used: (void)paramName.
16.7	A	A pointer parameter in a function prototype should be declared as the pointer to const if the pointer is not used to modify the addressed object.	Mostly all data processing for variety configuration, widgets and data types is required to pass the pointers as an argument. The architecture and design are intended for this casting.
18.4	R	Unions shall not be used.	There are two general cases in the code where this rule is violated. 1. <INSTANCE_NAME>_PTR_FILTER_VARIANT definition and usage. This union is used to simplify the pointer arithmetic with the Filter History Objects. Widgets may have two kinds of Filter History: Regular History Object and Proximity History Object. The mentioned union defines three different pointers: void, RegularObjPtr, and ProximityObjPtr. 2. APIs use unions to simplify operation with pointers on the parameters. The union defines four pointers: void*, uint8*, uint16*, and uint32*. In all cases, the pointers are verified for proper alignment before usage.

Component Debug Window

PSoC Creator allows you to view debug information about Components in your design. Each Component window lists the memory and registers for the instance. For detailed hardware registers descriptions, refer to the appropriate device technical reference manual.

To open the Component Debug window:

1. Make sure the debugger is running or in the break mode.
2. Choose Windows > Components… from the Debug menu.
3. In the Component Window Selector dialog, select the Component instances to view and click OK.

The selected Component Debug window(s) will open within the debugger framework. Refer to the "Component Debug Window" topic in the PSoC Creator Help for more information.

Resources

The 10-bit ADC Component always consumes one CSD (one 7-bit IDAC is available for general purpose) block, one Analog Mux bus B, and one port-pin for each input.

References

General References

- *Cypress Semiconductor web site*
- *PSoC 4 Device datasheets*

Application Notes

Cypress provides a number of application notes describing how PSoC can be integrated into your design. You can access them at the *Cypress Application Notes web page*.

Code Examples

PSoC Creator provides access to code examples in the Code Example 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 "Code Example" topic in the PSoC Creator Help for more information.

There are also numerous code examples that include schematics and code examples available online at the *Cypress Code Examples web page*.

Development Kit Boards

Cypress provides a number of development kits. You can access them at the *Cypress Development Kit web page*. Mentioned Code Examples uses the following development kits:

- *CY8CKIT-041 PSoC® 4 S-Series Pioneer Kit*
- *CY8CKIT-048 PSoC® Analog Coprocessor Pioneer Kit*

DC and AC Electrical Characteristics

Specifications are valid for +25° C, VDD 3.3v, Cmod = 2.2nF, Csh = 10nF, and CintA = CintB = 470 pF except where noted.

Note Final characterization data for the PSoC 4100PS and PSoC Analog Coprocessor devices is not available at this time. Once the data is available, the Component datasheet will be updated on the Cypress web site.

ADC Performance Characteristics

Parameter	Min	Typ	Max	Units	Details/ Conditions
Voltage Refference (Vref)	-	1.2	-	V	VDDA < 2.733V
	-	2.133	-	V	2.733V <= VDDA < 4.5V
	-	3.840	-	V	4.5V <= VDDA
Resolution	-	-	10	bits	Auto-zeroing is required every millisecond
Number of channels - single ended	-	-	10		
Monotonicity	-	-	-	Yes	Yes
Gain error	-	-	±2	%	In V_{REF} (2.4 V) mode with V_{DDA} bypass capacitance of 10 µF
Input offset voltage	-	-	3	mV	In V_{REF} (2.4 V) mode with V_{DDA} bypass capacitance of 10 µF
Current consumption	-	-	0.25	mA	
Input voltage range - single ended	V_{SSA}	-	V_{DDA}	V	
Input resistance	-	2.2	-	KΩ	
Input capacitance	-	20	-	pF	
Power supply rejection ratio	-	60	-	dB	In V_{REF} (2.4 V) mode with V_{DDA} bypass capacitance of 10 µF
Sample acquisition time	-	10	-	µs	
Conversion time for 8-bit resolution at clock frequency = 48 MHz.	-	-	10.7	µs	Does not include acquisition and processing time.
Conversion time for 10-bit resolution at clock frequency = 48 MHz.	-	-	42.7	µs	Does not include acquisition and processing time.
Signal-to-noise and Distortion ratio (SINAD)	-	61	-	dB	With 10Hz input sine wave, external 2.4V reference, V_{REF} (2.4 V) mode
Input bandwidth without aliasing	-	-	22.4	KHz	8-bit resolution

Parameter	Min	Typ	Max	Units	Details/ Conditions
Integral Non Linearity. 1 KSPS	-	-	2	LSB	V_{REF} = 2.4 V or greater
Differential Non Linearity. 1 KSPS	-	-	1	LSB	

IDAC Characteristic

PSoC 4000S, PSoC 4100S:

Parameter	Description	Min	Typ	Max	Units	Conditions
$IDAC1_{DNL}$	DNL	-1	–	1	LSB	
$IDAC1_{INL}$	INL	-2	–	2	LSB	INL is ±5.5 LSB for VDDA < 2 V
$IDAC2_{DNL}$	DNL	-1	–	1	LSB	
$IDAC2_{INL}$	INL	-2	–	2	LSB	INL is ±5.5 LSB for VDDA < 2 V

PSoC 4100PS, PSoC Analog Coprocessor:

Parameter	Description	Min	Typ	Max	Units	Conditions
$IDAC1_{DNL}$	DNL	-1	–	1	LSB	
$IDAC1_{INL}$	INL	-3	–	3	LSB	
$IDAC2_{DNL}$	DNL	-1	–	1	LSB	
$IDAC2_{INL}$	INL	-3	–	3	LSB	

DC/AC Specifications

Refer to devices specific datasheet *PSoC 4 Device datasheets* for more details.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
4.0	Added support for PSoC 4100PS device family. Improved the Component.	Fixed issue for the 261817 errata item and removed it from the datasheet. Added IDAC characteristic data.
3.10.a	Updated datasheet.	Added errata item 261817.

Version	Description of Changes	Reason for Changes / Impact
3.10	The initial version of PSoC 4 10-bit ADC (CSD) Component.	First release of ADC component.

© Cypress Semiconductor Corporation, 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.