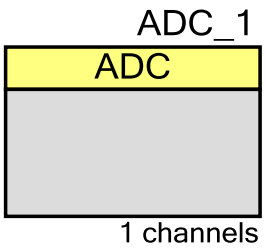
12-bit A/D Converter (PDL_ADC)

1.0



Features

- 12-bit resolution
- Converter using a type of RC Successive Approximation Register with sample and hold circuits
- Supports up to 24 ADC channels; consult the device-specific datasheet to determine how many channels for a particular device
- Different trigger sources to start A/D conversion
  - Trigger input from an external pin
  - Timer trigger input (base timer or multifunction timer)
  - Software activation
- DMA transfer triggered by the A/D interrupt request
- Priority conversion operation
- FIFO operation capability

General Description

The Peripheral Driver Library (PDL) 12-bit A/D Converter (PDL_ADC) component allows configuring the different operational modes of the ADC peripheral block in a device. You are able to configure multiple channels that are automatically scanned with the results placed in FIFO.

This component uses firmware drivers from the PDL_ADC module, which is automatically added to your project after a successful build.

When to Use a PDL_ADC Component

The PDL_ADC component to convert analog input voltage from an external pin to a digital value. The component can be used in high sample rate systems where you need to sample multiple channels without CPU intervention until all channels are sampled. It can also be used in low sample rate designs or in designs that have just a single channel to sample.

Quick Start

Follow these quick start steps to configure and initialize the ADC peripheral block:

1. Drag a PDL_ADC component from the Component Catalog FMx/Analog/ folder onto your schematic. The placed instance takes the name ADC_1. If you are using a pre-populated schematic, then find the ADC on the System tab (that this instance is named ADC).
2. By default, the component is configured for a simple, software-polled, single-channel setup. If your application needs a different setup, double-click to open the component's Configure dialog to provide configuration parameters such as the number of ADC channels, trigger signals for scan conversion, and interrupt sources.
3. In the Pin Editor, assign ADC channels to analog input pins in your device. If you are creating a design for particular development kit, refer the kit User Guide for suitable pin assignments.
4. Build the project to verify the correctness of your design. This will add the required PDL modules to the Workspace Explorer and generate configuration data for the ADC_1 instance.
5. In the *main.c* file, initialize and enable the peripheral.

```
uint32_t adcfifo;
uint16_t adcdata;

Adc_Init(&ADC_1_HW, &ADC_1_Config);
Adc_EnableWaitReady(&ADC_1_HW);
```

6. Then, start conversion and read the results.

```
for(;;)
{
    /* Trigger a scan and wait for it to complete */
    Adc_SwTriggerScan(&ADC_1_HW);
    while(TRUE == Adc_GetStatus(&ADC_1_HW, ScanStatus))
    {
    }

    /* Read the FIFO (contains data and other info) */
    adcfifo = Adc_ReadScanFifo(&ADC_1_HW);

    /* If the data is valid, extract the data from the FIFO.
    * Otherwise drop current reading.
    */
    if (AdcFifoDataValid == Adc_GetScanDataValid(&ADC_1_HW, adcfifo))
    {
        adcdata = Adc_GetScanData(&ADC_1_HW, adcfifo);
    }
}
```

7. Build and program the device.
8. Use the debugger to watch the `adcdata` variable as you vary the input voltage.

Component Parameters

The PDL_ADC component Configure dialog allows you to edit the configuration parameters for the component instance.

Basic Tab

This tab contains the component parameters used in the basic peripheral initialization settings.

Parameter Name	Description
bLsbAlignment	Alignment of the sampled data in a 16-bit word. When true, the conversion results are placed at bits [27:16] of the FIFO data register. Otherwise, the data are placed at bits [31:20].
numLogicalChannels	The number of ADC channels. Each channel can be assigned to any analog input pin in a device.
u8ComparingClockDiv	ADC input clock divider. Range=1-65. (Default=9).
u8EnableTime	Select the cycle count of operation enable state transition period.
enSamplingTimeN0	Sampling time multiplier to determine the actual sampling time for the channel.
u8SamplingTime0	Sampling time for the channel.
enSamplingTimeN1	Sampling time multiplier to determine the actual sampling time for the channel.
u8SamplingTime1	Sampling time for the channel.

Channel Tab

This tab contains the Channel configuration settings.

Parameter Name	Description
ch*n*Active	Indicate channel is actively scanned, *n* - channel number.
ch*n*SamplingTimeSelect	Select sampling time 0 or 1 for the channel, *n* - channel number.

Compare Tab

This tab contains the Compare configuration settings.

Parameter Name	Description
bRangeCompareAllChannels	Compare all selected channels or an individually-specified channel
bWithinRange	Compare within range or outside of range
logicalRangeCompareChannel	Channel to compare with u16LowerLimitRangeValue and u16UpperLimitRangeValue (if bRangeCompareAllChannels is false)

Parameter Name	Description
u16LowerLimitRangeValue	Lower limit for range comparison
u16UpperLimitRangeValue	Upper limit for range comparison
u8RangeCountValue	Number of consecutive comparisons needed for comparison to register
bCompareAllChannels	Compare all selected channels or an individually-specified channel
bCompIrqEqualGreater	Generate an IRQ if ADC value is greater than or equal to the compare value
u16CompareValue	ADC Comparison Value (CMPD)
u8CompareChannel	Logical channel to compare with u16CompareValue (if bCompareAllChannels is false)

Interrupts Tab

This tab contains the Interrupts configuration settings.

Parameter Name	Description
bComparisonIrq	Enable conversion comparison interrupt
bRangeComparisonIrq	Enable range comparison interrupt
pfnComparisonIrqCb	Callback function for conversion comparison IRQ. Note: this generates a declaration only - USER must implement the function
pfnRangeComparisonIrqCb	Callback function for range comparison IRQ. Note: this generates a declaration only - USER must implement the function
bTouchNvic	Enable ADC interrupts
bFifoOverrunIrq	Enable FIFO overrun interrupt
pfnPrioErrIrqCb	Callback function for priority sampling FIFO overrun error IRQ. Note: this generates a declaration only - USER must implement the function
pfnScanErrIrqCb	Callback function for scan sampling FIFO overrun error IRQ. Note: this generates a declaration only - USER must implement the function
bPrioIrq	Enable priority conversion interrupt
pfnPrioIrqCb	Callback function for priority IRQ. Note: this generates a declaration only - USER must implement the function
bScanIrq	Enable scan conversion interrupt
pfnScanIrqCb	Callback function for scan IRQ. Note: this generates a declaration only - USER must implement the function

Prio Tab

This tab contains the Prio configuration settings.

Parameter Name	Description
u8PrioFifoDepth	FIFO depth for priority conversion results
u8PrioLevel1AnalogChSel	Logical channel number for priority 1 conversions
u8PrioLevel2AnalogChSel	Logical channel number for priority 2 conversions
bPrioExtTrigStartEnable	Triggers priority conversion on falling edge of external signal
prioBaseTimerInstance	Base Timer component that is triggering the scan. The string "_ADC_TRIGGER" will be appended to this parameter to identify the channel
prioTriggerSource	Triggers priority conversion by a timer

Scan Tab

This tab contains the Scan configuration settings.

Parameter Name	Description
enScanMode	Scan conversion mode – one-shot or continuous. This determines if each scan must be triggered (one-shot) or continuously runs (continuous) after ADC is enabled.
scanBaseTimerInstance	Base Timer component instance name that is triggering the scan. This applies only when BT option is selected for the scanTriggerSource parameter.
scanTriggerSource	Trigger for scan conversion start. The conversion can be started by software or a timer. Supported option are: ▪ None - use software trigger to start conversion (default) ▪ BT - use Base Timer channel to start conversion ▪ MFT - use Multi-Function Timer to start conversion
u8ScanFifoDepth	FIFO depth for scan conversion results

Component Usage

After a successful build firmware drivers from the PDL_ADC module, are added to your project in the pdl/drivers/adc folder. Pass the generated data structures to the associated PDL functions in your application initialization code to configure the peripheral.

Generated Data

The PDL_ADC component populates the following peripheral initialization data structure(s). The generated code is placed in C source and header files that are named after the instance of the component (e.g. *ADC_1_config.c*). Each variable is also prefixed with the instance name of the component.

Data Structure Type	Name	Description
stc_adc_irq_en_t	ADC_1_IrqEn	Interrupt enable structure. This is automatically referenced in the ADC_1_Config.
stc_adc_irq_cb_t	ADC_1_IrqCb	Interrupt callback function. This is automatically referenced in the ADC_1_Config.
stc_adc_scan_t	ADC_1_Scan	Scan conversion configuration. This is automatically referenced in the ADC_1_Config.
stc_adc_prio_t	ADC_1_Prio	Priority conversion configuration. This is automatically referenced in the ADC_1_Config.
stc_adc_comapre_t	ADC_1_Compare	Comparison configuration. This is automatically referenced in the ADC_1_Config.
stc_adc_range_compare_t	ADC_1_Range	Range comparison configuration. This is automatically referenced in the ADC_1_Config.
stc_adc_config_t	ADC_1_Config	Configuration structure. Pass this to Adc_Init() in order to initialize the peripheral.

Once the component is initialized, the application code should use the peripheral functions provided in the referenced PDL files. Refer to the PDL for the list of provided API functions. To access this document, right-click on the component symbol on the schematic and choose "**Open API Documentation…**" option in the drop-down menu.

Preprocessor Macros

The ADC component generates the following preprocessor macro(s). Note that each macro is prefixed with the instance name of the component (e.g. "ADC_1").

Macro	Description
ADC_1_ADC_ACTIVE_CHANNELS	Selects the ADC channels being actively scanned.
ADC_1_CH_NR	Number of logical ADC channels.

Macro	Description
ADC_1_CHn	Maps logical ADC channel into physical ADC channel. *n*=0..ADC_1_CH_NR-1 - channel number. Use this macro with the value returned from the Adc_ReadScanFifo() or Adc_ReadPrioFifo() function to determine the channel from which the data read.
ADC_1_CH*n*_MSK	ADC channel mask. *n*=0.. ADC_1_CH_NR-1 - channel number.
ADC_1_CH*n*_SMPL_TIME	Selects sampling time 0 or 1 for the channel, *n* - channel number
ADC_1_HW	Hardware pointer
ADC_1_SetPinFunc_ADTG()	Macros to select external trigger pin
ADC_1_SetPinFunc_AN(n)	Macro to connect the ADC channels to analog input pins. channel: Specifies the ADC channel number. CH0 - ADC channel 0 CH1 - ADC channel 1 ... CHn - ADC channel n

Data in RAM

The generated data may be placed in flash memory (const) or RAM. The former is the more memory-efficient choice if you do not wish to modify the configuration data at run-time. Under the **Built-In** tab of the Configure dialog set the parameter CONST_CONFIG to make your selection. The default option is to place the data in flash.

Interrupt Support

If the PDL_ADC component is specified to trigger interrupts, it will generate the callback function declaration that will be called from the ADC ISR. The user is then required to provide the actual callback code. If a null string is provided the struct is populated with zeroes and the callback declaration is not generated. Thus, it is the user's responsibility to modify the struct in firmware.

The component generates the following function declarations.

Function Callback	Description
ADC_1_ScanIrqCb	Scan conversion interrupt callback function. Note: this generates a declaration only - USER must implement the function.
ADC_1_PrioIrqCb	Priority conversion interrupt callback function. Note: this generates a declaration only - USER must implement the function.
ADC_1_ScanErrIrqCb	Scan FIFO overrun error callback function. Note: this generates a declaration only - USER must implement the function.

Function Callback	Description
ADC_1_PrioErrIrqCb	Priority FIFO overrun error callback function. Note: this generates a declaration only - USER must implement the function.
ADC_1_ComparisonIrqCb	Comparison interrupt callback function. Note: this generates a declaration only - USER must implement the function.
ADC_1_RangeComparisonIrqCb	Range condition interrupt callback function. Note: this generates a declaration only - USER must implement the function.

Code Examples and Application Notes

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

Cypress also provides a number of application notes describing how FMx devices can be integrated into your design. You can access the Cypress Application Notes search web page at www.cypress.com/appnotes.

Resources

The PDL_ADC component uses the A/D converter (ADC) peripheral block.

References

- FM0+ Family of 32-bit ARM® Cortex®-M0+ Microcontrollers Peripheral Manuals
- Cypress FM0+ Family of 32-bit ARM® Cortex®-M0+ Microcontrollers

Component Changes

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

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

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

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

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, 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.