**ADC PROTOCOL**

**==============================================================================**

**ADC Peripheral features**

**==============================================================================**

* 12-bit, 10-bit, 8-bit or 6-bit configurable resolution.
* Interrupt generation at the end of conversion, end of injected conversion, and in case of analog watchdog or overrun events
* Single and continuous conversion modes.
* Scan mode for automatic conversion of channel 0 to channel x.
* Data alignment with in-built data coherency.
* Channel-wise programmable sampling time.
* External trigger option with configurable polarity for both regular and injected conversion.
* Dual/Triple mode (on devices with 2 ADCs or more).
* Configurable DMA data storage in Dual/Triple ADC mode.
* Configurable delay between conversions in Dual/Triple interleaved mode.
* ADC conversion type (refer to the datasheets).
* ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at speed.
* ADC input range: VREF(minus) = VIN = VREF(plus).
* DMA request generation during regular channel conversion.

**HOW TO USE THE DRIVER**

**==============================================================================**

**Initialize the ADC** low level resources by implementing the **HAL\_ADC\_MspInit():**

**Enable the ADC** interface clock using **\_\_HAL\_RCC\_ADC\_CLK\_ENABLE()**

ADC pins configuration

1. **Enable the clock** for the ADC GPIOs using the following function: **\_\_HAL\_RCC\_GPIOx\_CLK\_ENABLE()**
2. Configure these **ADC pin**s in analog mode using **HAL\_GPIO\_Init()**

In case of using interrupts (e.g. HAL\_ADC\_Start\_IT())

1. Configure the ADC interrupt priority using **HAL\_NVIC\_SetPriority()**
2. Enable the ADC IRQ handler using **HAL\_NVIC\_EnableIRQ()**
3. In ADC IRQ handler, call **HAL\_ADC\_IRQHandler()**

In case of using DMA to control data transfer (e.g. HAL\_ADC\_Start\_DMA())

1. Enable the DMAx interface clock using **\_\_HAL\_RCC\_DMAx\_CLK\_ENABLE()**
2. **Configure and enable two DMA** streams stream for managing data transfer from peripheral to memory (output stream)
3. Associate the initialized DMA handle to the CRYP DMA handle using **\_\_HAL\_LINKDMA()**
4. Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher than the input stream.

**CONFIGURATION OF ADC, GROUPS REGULAR/INJECTED, CHANNEL PARAMETERS**

**==============================================================================**

1. **Configure the ADC parameters** (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function **HAL\_ADC\_Init().**
2. **Configure the channels** for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function **HAL\_ADC\_ConfigChannel().**
3. Optionally, configure the injected group parameters (conversion trigger, sequencer, ..., of injected group) and the channels for injected group parameters (channel number, channel rank into sequencer, ..., into injected group) using function **HAL\_ADCEx\_InjectedConfigChannel().**
4. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function **HAL\_ADC\_AnalogWDGConfig().**
5. Optionally, for devices with several ADC instances: configure the multimode parameters using function **HAL\_ADCEx\_MultiModeConfigChannel().**

**EXECUTION OF ADC CONVERSIONS**

**==============================================================================**

ADC driver can be used among three modes: polling, interruption, transfer by DMA.

**POLLING MODE IO OPERATION**

**=================================**

1. Start the ADC peripheral using **HAL\_ADC\_Start()**
2. Wait for end of conversion using **HAL\_ADC\_PollForConversion(),** at this stage user can specify the value of timeout according to his end application
3. To read the ADC converted values, use the **HAL\_ADC\_GetValue()** function.
4. Stop the ADC peripheral using **HAL\_ADC\_Stop()**

**INTERRUPT MODE IO OPERATION**

**===================================**

1. Start the ADC peripheral using **HAL\_ADC\_Start\_IT()**
2. Use **HAL\_ADC\_IRQHandler()** called under **ADC\_IRQHandler()** Interrupt subroutine
3. At ADC end of conversion **HAL\_ADC\_ConvCpltCallback()** function is executed and user can add his own code by customization of function pointer **HAL\_ADC\_ConvCpltCallback**
4. In case of ADC Error**, HAL\_ADC\_ErrorCallback()** function is executed and user can add his own code by customization of function pointer **HAL\_ADC\_ErrorCallback**
5. Stop the ADC peripheral using **HAL\_ADC\_Stop\_IT()**

**DMA MODE IO OPERATION**

**==============================**

1. Start the ADC peripheral using HAL\_ADC\_Start\_DMA(), at this stage the user specify the length data to be transferred at each end of conversion
2. At The end of data transfer by HAL\_ADC\_ConvCpltCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ConvCpltCallback
3. In case of transfer Error, HAL\_ADC\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ErrorCallback
4. Stop the ADC peripheral using HAL\_ADC\_Stop\_DMA()

**DEINITIALIZATION OF ADC**

**==============================================================================**

Disable the ADC interface

1. ADC clock can be hard reset and disabled at RCC top level.
2. Hard reset of ADC peripherals using macro \_\_HAL\_RCC\_ADC\_FORCE\_RESET(), \_\_HAL\_RCC\_ADC\_RELEASE\_RESET().
3. ADC clock disable using the equivalent macro/functions as configuration step.

Example:

1. Into HAL\_ADC\_MspDeInit() (recommended code location) or with other device clock parameters configuration:
2. HAL\_RCC\_GetOscConfig(&RCC\_OscInitStructure);
3. RCC\_OscInitStructure.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;
4. RCC\_OscInitStructure.HSIState = RCC\_HSI\_OFF; (if not used for system clock)
5. HAL\_RCC\_OscConfig(&RCC\_OscInitStructure);

ADC pins configuration :

1. Disable the clock for the ADC GPIOs using macro \_\_HAL\_RCC\_GPIOx\_CLK\_DISABLE()
2. Optionally, in case of usage of ADC with interruptions: disable the NVIC for ADC using function HAL\_NVIC\_DisableIRQ(ADCx\_IRQn)
3. Optionally, in case of usage of DMA: deinitialize the DMA using function HAL\_DMA\_DeInit(). Disable the NVIC for DMA using function HAL\_NVIC\_DisableIRQ(DMAx\_Channelx\_IRQn)