Document Title	Specification of ADC Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	010

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R22-11

Document Change History

Date	Release	Changed by	Description
2022-11-24	R22-11	AUTOSAR Release Management	• SWS_ADC_00460 editorially adapted according to TPS_STDT_00042
2021-11-25	R21-11	AUTOSAR Release Management	• SWS_ADC_00338 modified • Chapter 9 picture includes from model
2020-11-30	R20-11	AUTOSAR Release Management	• Error classification tables updated
2019-11-28	R19-11	AUTOSAR Release Management	• API changed to asynchronous API: Adc_SetupResultBuffer, Adc_EnableHardwareTrigger, Adc_DisableHardwareTrigger, Adc_EnableGroupNotification, Adc_DisableGroupNotification • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Header file structure removed • Sequence chart and state diagram updated • Minor modification in API for input parameter passing • Editorial changes

2017-12-08	4.3.1	AUTOSAR Release Management	• Runtime error introduced; part of development errors changed into runtime errors • Exclude delta sigma ADC hardware from scope of ADC driver • Minor modifications in API Adc_SetupResultBuffer and Adc_ReadGroup • Header file structure update • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Variant-Post-Build requirements removed • Variant specific requirements for initialization API removed • Error classification table update • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• DET changed from 'Development Error Tracer' to 'Default Error Tracer'.
2014-10-31	4.2.1	AUTOSAR Release Management	• AdcGroupId is changed to precompile time value in all variants.
2014-03-31	4.1.2	AUTOSAR Release Management	• "Common" Published Information corrected • ARXML adaptations
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• API and configuration parameter added to support ECU degradation concept • Common Published Information removed • BSW General rework
2011-12-22	4.0.3	AUTOSAR Administration	• Requirement of ADC group status to be available for debugging removed

2009-12-18	4.0.1	AUTOSAR Administration	• ADC444 add Adc_ResultAlignmentType • SWS_Adc_00124 version number check correction • SWS_Adc_00337 reformulation • Limitation of ranges for AdcPrescale and AdcChannelId • InstanceId removed • ADC324 removed, • SWS_Adc_00458 introduced , DET for Adc_GetVersionInfo
2010-02-02	3.1.4	AUTOSAR Administration	• Limit checking support included; new config parameters added AdcEnableLimitCheck, AdcChannelLimitCheck, AdcChannelLowLimit, AdcChannelHighLimit and AdcChannelRangeSelect introduced. • ADC debug support added. • ADC configurable ADC data buffer alignment added. • Min/max values for AdcGroupId, AdcStreamingNumSamples, AdcMaxChannelResolution and AdcChannelResolution added. • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2008-02-01	3.0.2	AUTOSAR Administration	• Correction of: Table of Content

2007-12-21	3.0.1	AUTOSAR Administration	• New API Adc_ReadGroup introduced • Removed API Adc_ValueReadGroup • Modified API Adc_GetStreamLastPointer • New configuration parameter added • *AdcGroupReplacement • *AdcPriorityImplementation • *AdcResultBufferPointer • *AdcEnableQueuing • *AdcReadGroupApi • Configuration parameter removed • *ADC_GRP_PRIORITY_IMP_LEVEL • *ADC_STREAMING_BUFFER _POINTER • Priority mechanism improved • Type definitions modified and extended • State diagrams added • New state transitions defined • New state ADC_STREAM_COMPLETED added • State based requirements added • Sequence charts modified and extended • ADC buffer access mode example added • New DET's defined • *new DET ADC_E_ALREADY_INITIALIZED • *new DET ADC_E_PARAM_CONFIG • *new DET ADC_E_BUFFER_UNINIT • Part of existing requirments reformulated • Added new requirement ID's SWS_Adc_00321-SWS_Adc_00432 • Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added

2006-11-28	2.1.14	AUTOSAR Administration	• Removed the "On Demand" functionality. Related services not available anymore. • Removed the "Gated Continuous" conversion mode. Related services not available anymore. • Removed the distinction between internal and external hardware trigger. • Introduced a priority mechanism for channel groups for allowing channel groups with higher priority to interrupt ongoing conversions (can cover also the "On demand" functionality). • Reworked the "Streaming Access Mode". A dedicated data structure for the returned values of a conversion is now clearly defined. • Conversion values access now allowed only through channel groups (no single channel value available. Related service not available anymore).
2009-12-18	4.0.1	AUTOSAR Administration	• Document structure adapted to common Release 2.0 SWS Template.
2009-12-18	4.0.1	AUTOSAR Administration	• Initial Release.

Disclaimer

Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration of the AUTOSAR Basic Software module ADC Driver. The ADC driver is targeting Successive Approximation ADC Hardware. Delta Sigma ADC conversion use cases are out of scope of this specification.

The ADC module initializes and controls the internal Analogue Digital Converter Unit(s) of the microcontroller. It provides services to start and stop a conversion respectively to enable and disable the trigger source for a conversion. Furthermore it provides services to enable and disable a notification mechanism and routines to query the status and result of a conversion.

The ADC module works on so called ADC Channel Groups, which are build from so called ADC Channels.An ADC Channel Group combines an analogue input pin (ADC Channel), the needed ADC circuitry itself and conversion result register into an entity that can be individually controlled and accessed via the ADC module.

2 Acronyms and Abbreviations

Abbreviation / Acronym:	Description:
DEM	Diagnostic Event Manager
DET	Default Error Tracer
ADC	Analogue Digital Converter
MCU	Microcontroller Unit
API	Application Programming Interface
HW	Hardware
SW	Software
ADC HW Unit	Represents a microcontroller input electronic device that includes all parts necessary to perform an "analogue to digital conversion".
ADC Module	ADC Basic Software module ADC Driver, abbreviated also with ADC Driver
ADC Channel	Represents a logical ADC entity bound to one port pin. Multiple ADC entities can be mapped to the same port pin.
ADC Channel Group	A group of ADC channels linked to the same ADC hardware unit (e.g. one Sample&Hold and one A/D converter). The conversion of the whole group is triggered by one trigger source.
ADC Result Buffer (ADC Streaming Buffer, ADC Stream Buffer)	The user of the ADC Driver has to provide a buffer for every group. This buffer can hold multiple samples of the same group channel if streaming access mode is selected. If single access mode is selected one sample of each group channel is held in the buffer.
Software Trigger	Software API call that starts the conversion of one ADC channel group or a continuous series of ADC channel group conversions.
Hardware Trigger	ADC internal trigger signal that starts one conversion of an ADC channel group. ADC hardware trigger are generated internally in the ADC hardware, e.g. based on an ADC timer or a trigger edge signal. The trigger hardware is tightly coupled or integrated in the ADC hardware. No software is required to start the ADC channel group conversion after the hardware trigger is detected. Note: If the ADC hardware does not support hardware trigger, a similar behavior can be realized with software trigger in combination with the GPT/ICU driver. E.g. in a GPT timer notification function a software triggered ADC channel group conversion can be started.
Conversion Mode	One-Shot: The conversion of an ADC channel group is performed once after a trigger and the results are written to the assigned result buffer. A trigger can be a software API call or a hardware event. Continuous: The conversions of an ADC channel group are performed continuously after a software API call (start) and the results are written to the assigned result buffer. The conversions themselves are running automatically (hardware/interrupt controlled). The Continuous conversions can be stopped by a software API call (stop).
Sampling Time, Sample Time	Time during which the analogue value is sampled (e.g. loading the capacitor, ...)
Conversion Time	Time during which the sampled analogue value is converted into digital representation.
Acquisition Time	Sample Time + Conversion Time.

Table 2.1: Acronyms and abbreviations used in this document

3 Related documentation

3.1 Input documents & related standards and norms

[1] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [1], which is also valid for ADC Driver.

Thus, the specification SWS BSW General shall be considered as additional and required specification for ADC Driver.

4 Constraints and assumptions

4.1 Limitations

Power State Control APIs are implementable only if the MCAL driver owns the complete underlying HW peripheral i.e. the HW peripheral is not accessed by other MCAL modules

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

Module MCU Driver

The Microcontroller Unit Driver (MCU Driver) is primarily responsible for initializing and controlling the chip's internal clock sources and clock prescalers. The clock frequency may affect:

- Trigger frequency
- Conversion time
- Sampling time

Module PORT driver

The PORT module shall configure the port pins used by the ADC module. Both analogue input pins and external trigger pins have to be considered.

6 Requirements Tracing

Requirement	Description	Satisfied by
[SRS_Adc_12280]	The ADC Driver shall allow a specific result access modes for each ADC Channel Group	[SWS_Adc_00140] [SWS_Adc_00382] [SWS_Adc_00383]
[SRS_Adc_12283]	The ADC driver shall mask out information bits from the conversion result not belonging to the ADC value	[SWS_Adc_00122]
[SRS_Adc_12291]	The ADC Driver shall provide a service for querying the status of an ADC Channel Group	[SWS_Adc_00219] [SWS_Adc_00220] [SWS_Adc_00221] [SWS_Adc_00222] [SWS_Adc_00224] [SWS_Adc_00226] [SWS_Adc_00325] [SWS_Adc_00326] [SWS_Adc_00327] [SWS_Adc_00328] [SWS_Adc_00329] [SWS_Adc_00330] [SWS_Adc_00331]
[SRS_Adc_12292]	If the ADC provides signed values, the ADC driver shall put the sign bit into the MSB of the return value	[SWS_Adc_00113] [SWS_Adc_00214]
[SRS_Adc_12307]	The ADC Driver shall support a specific basic static configurations per channel	[SWS_Adc_00099]
[SRS_Adc_12317]	The ADC Driver shall provide notification functions to inform the caller about the end of a conversion for a Channel Group	[SWS_Adc_00104] [SWS_Adc_00155] [SWS_Adc_00156] [SWS_Adc_00157]
[SRS_Adc_12318]	The ADC driver shall provide a service to enable and disable each notification function separately	[SWS_Adc_00057] [SWS_Adc_00058] [SWS_Adc_00077] [SWS_Adc_00156] [SWS_Adc_00157]
[SRS_Adc_12364]	The ADC driver shall provide services to start and stop the conversion of an ADC Channel Group for all conversion modes	[SWS_Adc_00060] [SWS_Adc_00061] [SWS_Adc_00145] [SWS_Adc_00146] [SWS_Adc_00157] [SWS_Adc_00356] [SWS_Adc_00357] [SWS_Adc_00385] [SWS_Adc_00386]
[SRS_Adc_12447]	The ADC Driver shall allow to group ADC channels that belong to the same ADC HW unit	[SWS_Adc_00090] [SWS_Adc_00091] [SWS_Adc_00098] [SWS_Adc_00099] [SWS_Adc_00100] [SWS_Adc_00101] [SWS_Adc_00104] [SWS_Adc_00277] [SWS_Adc_00280]
[SRS_Adc_12802]	The ADC driver shall provide (for streaming access mode) a service to identify most recent sample and number of available samples of a channel group	[SWS_Adc_00214] [SWS_Adc_00216] [SWS_Adc_00219]
[SRS_Adc_12817]	The ADC Driver shall allow for each ADC channel group the static configuration of exactly one trigger source	[SWS_Adc_00146] [SWS_Adc_00279] [SWS_Adc_00283] [SWS_Adc_00356] [SWS_Adc_00357]
[SRS_Adc_12818]	The ADC Driver shall allow assigning one ADC channel to more than one ADC Channel Group	[SWS_Adc_00092]
[SRS_Adc_12819]	The ADC Driver shall provide a synchronous service for reading the last valid conversion results of the selected channel group	[SWS_Adc_00113] [SWS_Adc_00122] [SWS_Adc_00318]
[SRS_Adc_12820]	The ADC driver shall allow the configuration of a priority level for each channel group	[SWS_Adc_00288] [SWS_Adc_00289] [SWS_Adc_00310] [SWS_Adc_00340] [SWS_Adc_00341]

△

Requirement	Description	Satisfied by
[SRS_Adc_12822]	The structure containing the results of a channel group conversion shall be generated with a uniform dimension	[SWS_Adc_00320]
[SRS_Adc_12823]	The ADC driver shall provide services to enable and disable HW triggers for each channel group	[SWS_Adc_00114] [SWS_Adc_00116] [SWS_Adc_00144] [SWS_Adc_00273] [SWS_Adc_00281] [SWS_Adc_00282]
[SRS_Adc_12824]	The result alignment shall be configurable between right-alignment and left-alignment	[SWS_Adc_00113]
[SRS_Adc_12825]	The results of the conversion of a channel group configured in streaming access mode shall be returned into a buffer with a fixed number of elements	[SWS_Adc_00319]
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_Adc_NA_00460]
[SRS_BSW_00006]	The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	[SWS_Adc_NA_00460]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Adc_NA_00460]
[SRS_BSW_00009]	All Basic SW Modules shall be documented according to a common standard.	[SWS_Adc_NA_00460]
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_Adc_NA_00460]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Adc_00054]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_Adc_NA_00460]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_Adc_NA_00460]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_Adc_NA_00460]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_Adc_NA_00460]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Adc_NA_00460]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_Adc_NA_00460]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_Adc_NA_00460]
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_Adc_00120] [SWS_Adc_00121] [SWS_Adc_00228] [SWS_Adc_00259] [SWS_Adc_00260] [SWS_Adc_00265] [SWS_Adc_00266]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_Adc_NA_00460]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_Adc_NA_00460]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Adc_NA_00460]
[SRS_BSW_00307]	Global variables naming convention	[SWS_Adc_NA_00460]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Adc_NA_00460]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Adc_NA_00460]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Adc_00125] [SWS_Adc_00126] [SWS_Adc_00128] [SWS_Adc_00129] [SWS_Adc_00131] [SWS_Adc_00152] [SWS_Adc_00225] [SWS_Adc_00241]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_Adc_NA_00460]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_Adc_NA_00460]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Adc_NA_00460]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_Adc_NA_00460]
[SRS_BSW_00335]	Status values naming convention	[SWS_Adc_00221] [SWS_Adc_00222] [SWS_Adc_00224]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Adc_00111]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_Adc_NA_00460]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_Adc_NA_00460]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_Adc_NA_00460]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Adc_NA_00460]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Adc_NA_00460]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_Adc_NA_00460]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_Adc_00082]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_Adc_00082]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_Adc_NA_00460]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_Adc_NA_00460]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_Adc_00107] [SWS_Adc_00125] [SWS_Adc_00126] [SWS_Adc_00128] [SWS_Adc_00129] [SWS_Adc_00131] [SWS_Adc_00133] [SWS_Adc_00136] [SWS_Adc_00137] [SWS_Adc_00152] [SWS_Adc_00154] [SWS_Adc_00164] [SWS_Adc_00165] [SWS_Adc_00166] [SWS_Adc_00218] [SWS_Adc_00225] [SWS_Adc_00241]
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	[SWS_Adc_NA_00460]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Adc_00054]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_Adc_00107] [SWS_Adc_00154] [SWS_Adc_00294] [SWS_Adc_00295] [SWS_Adc_00297] [SWS_Adc_00298] [SWS_Adc_00299] [SWS_Adc_00300] [SWS_Adc_00301] [SWS_Adc_00302]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_Adc_NA_00460]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Adc_00054]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_Adc_NA_00460]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.	[SWS_Adc_NA_00460]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_Adc_NA_00460]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_Adc_NA_00460]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_Adc_NA_00460]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_Adc_NA_00460]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_Adc_NA_00460]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_Adc_NA_00460]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_Adc_NA_00460]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_Adc_NA_00460]
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_Adc_NA_00460]
[SRS_SPAL_00157]	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	[SWS_Adc_00057] [SWS_Adc_00058] [SWS_Adc_00082] [SWS_Adc_00083] [SWS_Adc_00104]
[SRS_SPAL_12056]	All driver modules shall allow the static configuration of notification mechanism	[SWS_Adc_00080] [SWS_Adc_00084] [SWS_Adc_00085]
[SRS_SPAL_12057]	All driver modules shall implement an interface for initialization	[SWS_Adc_00054]
[SRS_SPAL_12063]	All driver modules shall only support raw value mode	[SWS_Adc_00113]
[SRS_SPAL_12064]	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	[SWS_Adc_NA_00460]
[SRS_SPAL_12067]	All driver modules shall set their wake-up conditions depending on the selected operation mode	[SWS_Adc_NA_00460]
[SRS_SPAL_12068]	The modules of the MCAL shall be initialized in a defined sequence	[SWS_Adc_NA_00460]
[SRS_SPAL_12069]	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	[SWS_Adc_NA_00460]
[SRS_SPAL_12077]	All drivers shall provide a non blocking implementation	[SWS_Adc_NA_00460]
[SRS_SPAL_12078]	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	[SWS_Adc_NA_00460]
[SRS_SPAL_12092]	The driver's API shall be accessed by its handler or manager	[SWS_Adc_NA_00460]
[SRS_SPAL_12125]	All driver modules shall only initialize the configured resources	[SWS_Adc_00056]

▽

△

Requirement	Description	Satisfied by
[SRS_SPAL_12129]	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	[SWS_Adc_00078]
[SRS_SPAL_12163]	All driver modules shall implement an interface for de-initialization	[SWS_Adc_00110] [SWS_Adc_00111]
[SRS_SPAL_12169]	All driver modules that provide different operation modes shall provide a service for mode selection	[SWS_Adc_NA_00460]
[SRS_SPAL_12265]	Configuration data shall be kept constant	[SWS_Adc_NA_00460]
[SRS_SPAL_12267]	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	[SWS_Adc_NA_00460]
[SRS_SPAL_12448]	All driver modules shall have a specific behavior after a development error detection	[SWS_Adc_00107] [SWS_Adc_00125] [SWS_Adc_00126] [SWS_Adc_00128] [SWS_Adc_00129] [SWS_Adc_00131] [SWS_Adc_00133] [SWS_Adc_00136] [SWS_Adc_00137] [SWS_Adc_00152] [SWS_Adc_00154] [SWS_Adc_00164] [SWS_Adc_00165] [SWS_Adc_00166] [SWS_Adc_00225] [SWS_Adc_00241]
[SRS_SPAL_12461]	Specific rules regarding initialization of controller registers shall apply to all driver implementations	[SWS_Adc_00054] [SWS_Adc_00246] [SWS_Adc_00247] [SWS_Adc_00248] [SWS_Adc_00249] [SWS_Adc_00250]
[SRS_SPAL_12463]	The register initialization settings shall be combined and forwarded	[SWS_Adc_NA_00460]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 General behavior

7.1.1 Background & Rationale

The table below shows a list of possible desired functionalities of an ADC user and in which way they are provided by the ADC module. Furthermore the table also depicts a possible realization and the mapping of these functionalities to the capabilities of a commercial microcontroller (C16x).

Desired Functionality	ADC Driver Function	Example: C16x Derivate Wording
Just one conversion result of a single channel.	Software triggered one-shot conversion where the converted group consists of exactly one channel.	Fixed channel, single conversion, software trigger.
Cyclic conversion of a single channel.	Hardware triggered one-shot conversion where the converted group consists of exactly one channel.	Fixed channel, single conversion, hardware trigger.
Repeated conversion of a single channel.	Continuous conversion where the converted group consists of exactly one channel.	Fixed channel,continuous conversion.
Just one conversion result of each channel within a group.	Software triggered one-shot conversion where the converted group consists of more than one channel.	Auto scan, single conversion, software trigger.
Cyclic conversion of each channel within a group.	Hardware triggered one-shot conversion where the converted group consists of more than one channel.	Auto scan, single conversion, hardware trigger.
Repeated conversion of each channel within a group.	Continuous conversion where the converted group consists of more than one channel.	Auto scan, continuous conversion.

Table 7.1: Different possibilities of One-shot and Continuous conversions

7.1.2 Requirements

[SWS_Adc_00090] ⌈The ADC module shall allow grouping of one or more ADC channels into so called ADC Channel groups.⌋*(SRS_Adc_12447)*

[SWS_Adc_00091] ⌈The ADC module's configuration shall be such that an ADC Channel group contains at least one ADC Channel.⌋*(SRS_Adc_12447)*

[SWS_Adc_00451] ⌈The ADC module's configuration shall be such that an ADC Channel group contains exactly one ADC Channel if the global limit checking feature is enabled and the channel specific limit checking is enabled for the ADC Channel.⌋*()*

[SWS_Adc_00092] ⌈The ADC module shall allow the assignment of an ADC channel to more than one group.⌋*(SRS_Adc_12818)*

[SWS_Adc_00277] ⌈The ADC module's configuration shall be such that all channels contained in one ADC Channel group shall belong to the same ADC HW Unit.⌋*(SRS_Adc_12447)*

The ADC module supports the following conversion modes:

- **[SWS_Adc_00380]** ⌈The ADC module shall support the conversion mode "One-shot Conversion" for all ADC Channel groups. One-shot conversion means that exactly one conversion is executed for each channel configured for the group being converted.⌋*()*
- **[SWS_Adc_00381]** ⌈The ADC module shall support the conversion mode "Continuous Conversion[1]" for all ADC Channel groups with trigger source software."Continuous Conversion" means that after the conversion has been completed, the conversion of the whole group is repeated. The conversions of the individual ADC channels within the group as well as the repetition of the whole group don't need any additional trigger events to be executed. Converting the individual channels within the group can be done sequentially or in parallel depending on hardware and/or software capabilities.⌋*()*

The ADC module supports the following start conditions or trigger sources:

- **[SWS_Adc_00356]** ⌈The ADC module shall support the start condition "Software API Call" for all conversion modes. The trigger source "Software API Call" means that the conversion of an ADC Channel group is started/stopped with a service provided by the ADC module.⌋*(SRS_Adc_12817, SRS_Adc_12364)*
- **[SWS_Adc_00357]** ⌈The ADC module shall support the start condition "Hardware Event" for groups configured in One-Shot conversion mode. The trigger source "Hardware Event" means that the conversion of an ADC Channel group can be started by a hardware event, e.g. an expired timer or an edge detected on an input line.⌋*(SRS_Adc_12817, SRS_Adc_12364)*

[SWS_Adc_00279] ⌈The ADC module shall allow configuring exactly one trigger source for each ADC Channel group.⌋*(SRS_Adc_12817)*

The ADC module supports the following result access modes:

- **[SWS_Adc_00382]** ⌈The ADC module shall support result access using the API function Adc_GetStreamLastPointer. Calling Adc_GetStreamLastPointer informs the user about the position of the group conversion results of the latest conversion round in the result buffer and about the number of valid conversion results in the result buffer. The result buffer is an external buffer provided from the application.⌋ *(SRS_Adc_12280)*

Note: The function is used for both types of groups, configured in Streaming Access Mode and in Single Access Mode (Single Access Mode is handled equal to Streaming Access Mode with Streaming Counter equal to 1).

- **[SWS_Adc_00383]** ⌈The ADC module shall support result access using the API function Adc_ReadGroup, if the generation of this API function is statically configured. Calling Adc_ReadGroup copies the group conversion results of the latest conversion round to an application buffer which start address is specified as API parameter of Adc_ReadGroup.⌋*(SRS_Adc_12280)*

Note: The function is used for both types of groups, configured in Streaming Access Mode and in Single Access Mode.

[SWS_Adc_00140] ⌈The ADC module shall guarantee the consistency of the returned result value for each completed conversion.⌋*(SRS_Adc_12280)*

Note:

The consistency of the group channel results can be obtained with the following methods on the application side:

- Using group notification mechanism
- Polling via API function Adc_GetGroupStatus

In any case, new result data must be read out from the result buffer (e.g. via Adc_ReadGroup) before they are overwritten. If the function Adc_GetGroupStatus reports state ADC_STREAM_COMPLETED and conversions for the same group are still ongoing (continuous conversion or hardware triggered conversion), the user is responsible to access the results in the result buffer, before the ADC driver overwrites the group result buffer.

[SWS_Adc_00384] ⌈The ADC module's environment shall ensure that a conversion has been completed for the requested group before requesting the conversion result.⌋*()*

Note: If no conversion has been completed for the requested channel group (e.g. because the conversion of the ADC Channel group has been stopped by the user) the value returned by the ADC module will be arbitrary (Adc_GetStreamLastPointer will return 0 and read NULL_PTR; Adc_ReadGroup will return E_NOT_OK).

[SWS_Adc_00288] ⌈The ADC module shall allow the configuration of a priority level for each channel group.⌋*(SRS_Adc_12820)*

Note: This implies a prioritization mechanism, implemented in SW, or where available, supported by the HW. Groups with trigger source HW are prioritized always with the HW prioritization mechanism.

[SWS_Adc_00310] ⌈The ADC module's priority mechanism shall allow aborting and restarting of channel group conversions.⌋*(SRS_Adc_12820)*

[SWS_Adc_00345] ⌈The ADC module's priority mechanism shall allow suspending and resuming of channel group conversions.⌋*()*

[SWS_Adc_00430] ⌈The ADC module shall allow a group specific configuration whether the abort/restart or suspend/resume mechanism is used for interrupted channel groups.⌋*()*

Note: In contrast to the software controlled abort/restart or suspend/resume mechanism on channel group level, the ADC hardware can support abort/restart and suspend/resume mechanism on ADC channel level. It is up to the implementation which of both mechanisms is implemented on channel level.

[SWS_Adc_00311] ⌈The ADC module's priority mechanism shall allow the queuing of requests for different groups.⌋*()*

Note: Higher priority groups can abort or suspend lower priority groups. In this case the priority handler should put the interrupted channel group conversion in the queue and this channel group conversion will be restarted or resumed later, transparently to the user.

[SWS_Adc_00312] ⌈In the ADC module's priority mechanism the lowest priority is 0.⌋ *()*

[SWS_Adc_00289] ⌈The ADC module's priority mechanism shall allow the configuration of 256 priority levels (0...255).⌋*(SRS_Adc_12820)*

[SWS_Adc_00315] ⌈The ADC module shall support the static configuration option to disable the priority mechanism.⌋*()*

[SWS_Adc_00340] ⌈The ADC module shall support the static configuration option to enable the priority mechanism ADC_PRIORITY_HW_SW, using both hardware and software prioritization mechanism. If the hardware does not provide the hardware prioritization mechanism a pure software prioritization mechanism shall be implemented.⌋ *(SRS_Adc_12820)*

[SWS_Adc_00341] ⌈If the priority mechanism is supported by the hardware: The ADC module shall support the static configuration option ADC_PRIORITY_HW to enable the priority mechanism using only the hardware priority mechanism.⌋*(SRS_Adc_12820)*

Note: If hardware priority mechanism is selected, also groups with software trigger source are prioritized from the hardware prioritization mechanism.

[SWS_Adc_00339] ⌈If hardware priority mechanism is supported and selected: The ADC module shall allow the mapping of the configured priority levels (0-255) to the available hardware priority levels.⌋*()*

Note: The specific implementation of the ADC module describes restrictions concerning the available hardware priority levels and the possible mapping of the available hardware priorities to the priorities of the ADC channel groups.

[SWS_Adc_00332] ⌈If the priority mechanism is active, the ADC module shall support a queuing of conversion requests. The conversion requsts shall be queued when, if channel group with higher priority is requested for conversion while lower priority channel group conversion is ongoing (here lower priority group shall be queued) OR channel group conversion requests can not immediately be handled, because a higher priority channel group conversion is ongoing.⌋*()*

[SWS_Adc_00417] ⌈If the priority mechanism is active, the ADC module shall handle channel group conversion requests for groups with the same priority level, in a 'first come first served' order.⌋*()*

[SWS_Adc_00333] ⌈If the priority mechanism is not active and if the static configuration parameter AdcEnableQueuing is set to ON, the ADC module shall support a queuing of conversion requests and shall service the software groups in a 'first come first served' order.⌋*()*

Note: Software conversion requests storage shall be supported in a software implemented queue or by the hardware.

[SWS_Adc_00335] ⌈If the queuing mechanism is active (priority mechanism active or queuing explicitly activated), the ADC module shall store each software conversion request per channel group at most one time in the software queue.⌋*()*

Note: The ADC module shall only store one conversion request per channel group, not multiple requests, which may occur if a high priority long-term conversion blocks the hardware.

[SWS_Adc_00336] ⌈'Enable hardware trigger requests', generated with API function Adc_EnableHardwareTrigger, shall not be stored in any queue.⌋*()*

[SWS_Adc_00337] ⌈The hardware prioritization mechanism shall be used in case of hardware triggered conversion requests.⌋*()*

[SWS_Adc_00338] ⌈When the group status is equal to ADC_IDLE or group status is equal to ADC_STREAM_COMPLETED and if an ADC group can be implicitly stopped, then ADC module shall allow storing an additional software conversion request for the same group.⌋*()*

[SWS_Adc_00060] ⌈The ADC module shall call the group notification function, whenever a conversion of all channels of the requested group is completed and if the notification is configured and enabled.⌋*(SRS_Adc_12364)*

[SWS_Adc_00413] ⌈The ADC module functions shall be reentrant, if the functions are called for different channel groups. This requirement shall be applicable for all API functions, except Adc_Init, Adc_DeInit,Adc_GetVersionInfo, Adc_SetPowerState, Adc_GetTargetPowerState, Adc_GetCurrentPowerState and Adc_PreparePowerState.⌋*()*

Note: The reentrancy of the API functions applies only if the caller takes care that there is no simultaneous usage of the same group.

[SWS_Adc_00503] ⌈Simple read calls,as implemented in Adc_ReadGroup and Adc_GetGroupStatus,shall always be reentrant even if the functions are called for same channel groups. It is up to the implementation to use adequate protection mechanisms (e.g. disabling/enabling interrupts.⌋*()*

Note: Calling Adc_ReadGroup can implicitely change the group status.

[SWS_Adc_00414] ⌈The ADC module's environment shall check the integrity (see Note SWS_Adc_00413) if several calls for the same ADC group are used during runtime in different tasks or ISR's.⌋*()*

[SWS_Adc_00415] ⌈The ADC module shall not check the integrity (see Note SWS_Adc_00413) if several calls for the same ADC group are used during runtime in different tasks or ISRs.⌋*()*

[SWS_Adc_00445] ⌈The ADC module shall allow configuring limit checking for ADC Channels.⌋*()*

[SWS_Adc_00446] ⌈If limit checking is active for an ADC Channel, only ADC conversion results, which are in the configured range, are taken into account for updating the user specified ADC result buffer.⌋*()*

[SWS_Adc_00447] ⌈If limit checking is active for an ADC Channel, only ADC conversion results, which are in the configured range, are taken into account for triggering state transitions of the ADC group status.⌋*()*

[SWS_Adc_00448] ⌈If continuous conversion mode with SW trigger source is selected: if limit checking is active for an ADC Channel, ADC conversion results, which are not in the configured range, are neglected from the ADC driver, and the conversion is reiterated.⌋*()*

[SWS_Adc_00449] ⌈If one-shot conversion mode with SW trigger source is selected: if limit checking is active for an ADC Channel, an ADC conversion result, which is not in the configured range, is neglected from the ADC driver, and the ADC group, containing the ADC channel, will stay in state ADC_BUSY.⌋*()*

Note: Before a new SW triggered one-shot conversion can be reissued, it is required to set the ADC group status to ADC_IDLE, using the API Adc_StopGroupConversion().

[SWS_Adc_00450] ⌈If one-shot conversion mode with HW trigger source is selected: if limit checking is active for an ADC Channel, ADC conversion results, which are not in the configured range, are neglected from the ADC driver, and the conversion is reissued, triggered by the next HW trigger.⌋*()*

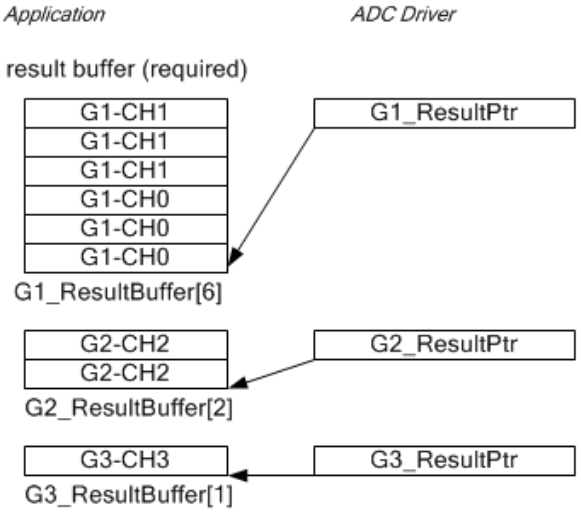
7.1.3 ADC Buffer Access Mode Example

1. Configuration

Group	ADC_GROUP_DEFINITION	ADC_RESULT_POINTER
group G1:	CH0, CH1	G1_ResultPtr
group G2:	CH2	G2_ResultPtr
group G3:	CH3	G3_ResultPtr

Group	ADC_GROUP_ACCESS_MODE	ADC_STREAMING_NUM_SAMPLES
group G1:	ADC_ACCESS_MODE_STREAMING	3
group G2:	ADC_ACCESS_MODE_STREAMING	2
group G3:	ADC_ACCESS_MODE_SINGLE	(1)

2. Result Pointer Initialization with Adc_SetupResultBuffer API function



3. Result access with Adc_GetStreamLastPointer API function

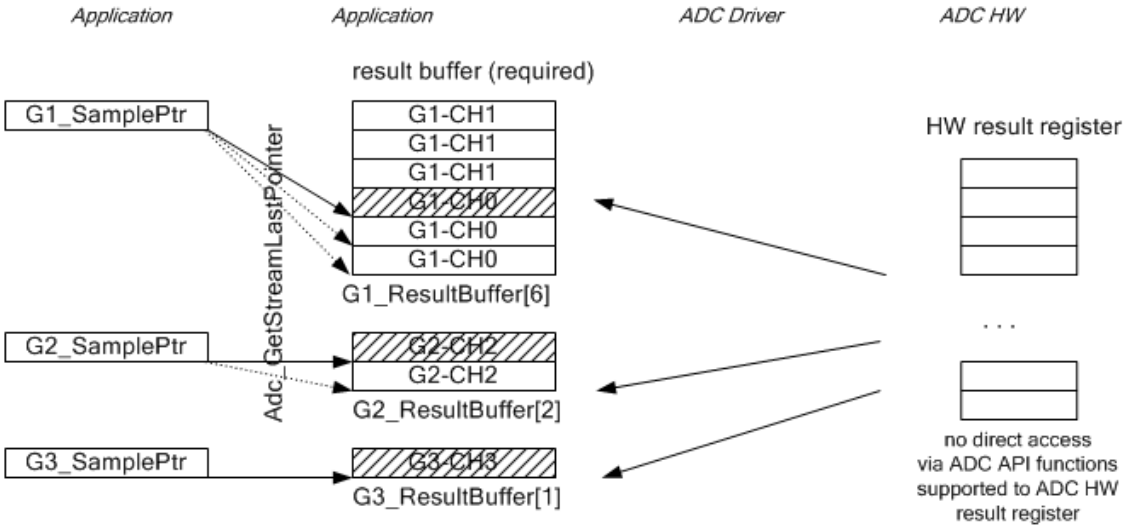


Figure 7.1: Example for Group and Result Buffer configuration - Result pointer initialization and calling Adc_GetStreamLastPointer for accessing results of latest conversion round in the Result Buffer

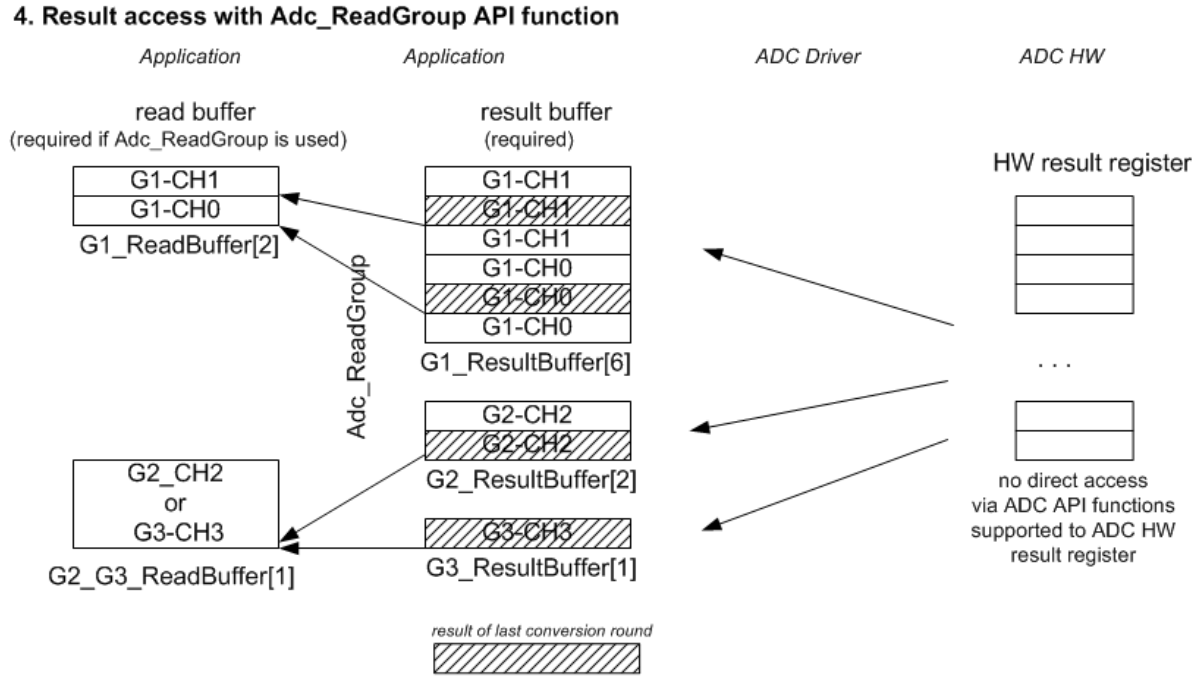


Figure 7.2: Example for calling Adc_ReadGroup which copies results from Result Buffer to optional Read Buffer

7.1.3.1 Example: Configuration

The example configuration consists of three ADC groups. Group 1 consists of 2 channels, group 2 and group 3 consist of one channel each. For group 1 and 2 the group access mode ADC_ACCESS_MODE_STREAMING is configured. The group access mode of group 3 is ADC_ACCESS_MODE_SINGLE. The ADC driver will store the conversion results of group 1-3 in three application buffers, accessed with three configured ADC_RESULT_POINTER :

G1_ResultPtr, G2_ResultPtr and G3_ResultPtr.

7.1.3.2 Example: Initialization

The user has to provide application result buffers for the ADC group results. One buffer is required for each group. The buffer size depends on the number of group channels, the group access mode and from the number of streaming samples, if streaming access mode is selected. Before starting a group conversion, the user has to initialize the group result pointer using API function Adc_SetupResultBuffer which initializes the group result pointer to point to the specified application result buffer.

7.1.3.3 Example: Adc_GetStreamLastPointer Usage

The ADC driver stores the conversion results of group G1, G2 and G3 in the according result buffer G1_ResultBuffer[], G2_ResultBuffer[] and G3_ResultBuffer[]. A direct access from the ADC API functions to the ADC hardware result register is not supported from the ADC driver.

The user provides three pointers G1_SamplePtr, G2_SamplePtr and G3_SamplePtr

which will point to the ADC application result buffer after calling Adc_GetStream LastPointer.Precisely pointer G1_SamplePtr points, after calling Adc_GetStreamLast Pointer, to the latest G1_CH0 result of the latest completed conversion round (G1_CH0 is the first channel in G1 group definition).The application result buffer layout is shown in Figure 2. The application result buffer of group 1 holds three times the streaming results of G1_CH0 and then three times the streaming results of G1_CH1. Knowing the application result buffer layout, the user is able to access all group channel results of the latest conversion round. G2_SamplePtr and G3_SamplePtr are also aligned, after calling Adc_GetStreamLastPointer, to point to the latest result of the first group channel of the according group. Both groups have only one channel. G2_SamplePtr points to one of the G2_CH2 results (the latest result). Because group 3 is configured in single access mode, G3_SamplePtr points always to G3_CH3.

Adc_GetStreamLastPointer returns the number of valid samples per channel, stored in the application result buffer (number of complete group conversion rounds). If the return value is equal to the configured parameter 'number of streaming samples', all conversion results in the streaming buffer are valid. If the return value is 0, no conversion results are available in the streaming buffer (the sample pointer will be aligned to NULL).

To enable Adc_GetStreamLastPointer to align the sample pointer (G1_SamplePtr, G2_ SamplePtr and G3_SamplePtr) to point to the latest channel result, the API is defined to pass a pointer to the result pointer instead the result pointer itself.

7.1.3.4 Example: Adc_ReadGroup Usage

If the optional API function Adc_ReadGroup is enabled, the user has to provide additional buffers for the selected groups, which can hold the results of one group conversion round. Calling Adc_ReadGroup copies the latest results from the application result buffer to the application read group buffer. In the example, one application read buffer (G2_G3_ReadBuffer) is used for group G2 and G3.

7.2 Conversion processing and interaction

7.2.1 Background & Rationale

The following examples specify the order of channel conversion depending on group and conversion type:

- Example 1: Channel group containing channels [CH0, CH1, CH2, CH3, and CH4] is configured in Continuous conversion mode. After finishing each scan, the notification (if enabled) is called. Then a new scan is started automatically.
- Example 2: Channel group containing channels [CH0, CH1, CH2, CH3, and CH4] is configured in One-Shot conversion mode. After finishing the scan the notification (if enabled) is called.
- Example 3: Channel group containing channel [CH3] is configured in Continuous conversion mode. After finishing each scan the notification (if enabled) is called. Then a new scan is started automatically.
- Example 4: Channel group containing channel [CH4] is configured in One-Shot conversion mode. After finishing the scan the notification (if enabled) is called.

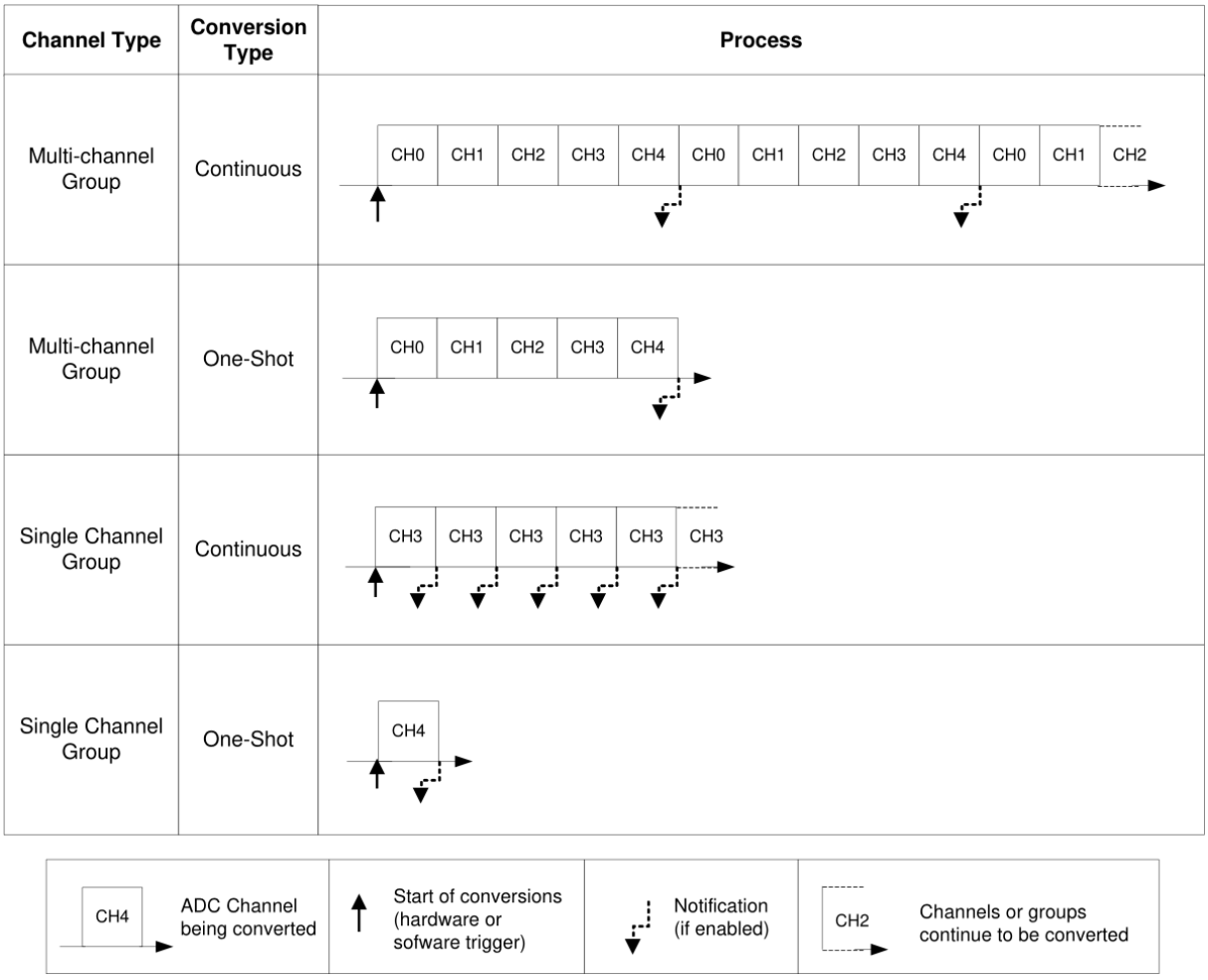


Figure 7.3: Conversion Mode behavior examples

7.2.2 Requirements

[SWS_Adc_00280] ⌈The ADC module shall convert only one ADC Channel group per ADC HW Unit at a time. The ADC module shall not support the concurrent conversion of different (even exclusive) ADC Channel groups on the same ADC HW Unit.⌋*(SRS_Adc_12447)*

Note: Concurrent conversion of ADC Channel groups on different ADC HW Units may be possible, depending on the capabilities of the hardware. Also concurrent conversion of individual channels within one channel group may be possible if supported by the hardware.

Note: If a channel shall be used in different conversion modes (e.g. continuous conversion mode during normal operation and one-shot conversion mode for a special conversion at a dedicated point in time), this channel shall be assigned to different groups configured with the respective conversion modes.

Note: In order to request the conversion of a channel shared between two groups, the ADC user has to stop the conversion of the first group containing the specified channel and then start the conversion of the second group containing the specified channel.

7.3 State Diagrams

The ADC module has a state machine that is shown in the following figures. The states are group specific and not module specific. The diagrams show all possible configuration options for ADC groups. The state transitions depend on the ADC group configuration.

7.3.1 ADC State Diagram for One-Shot/Continuous Group Conversion Mode

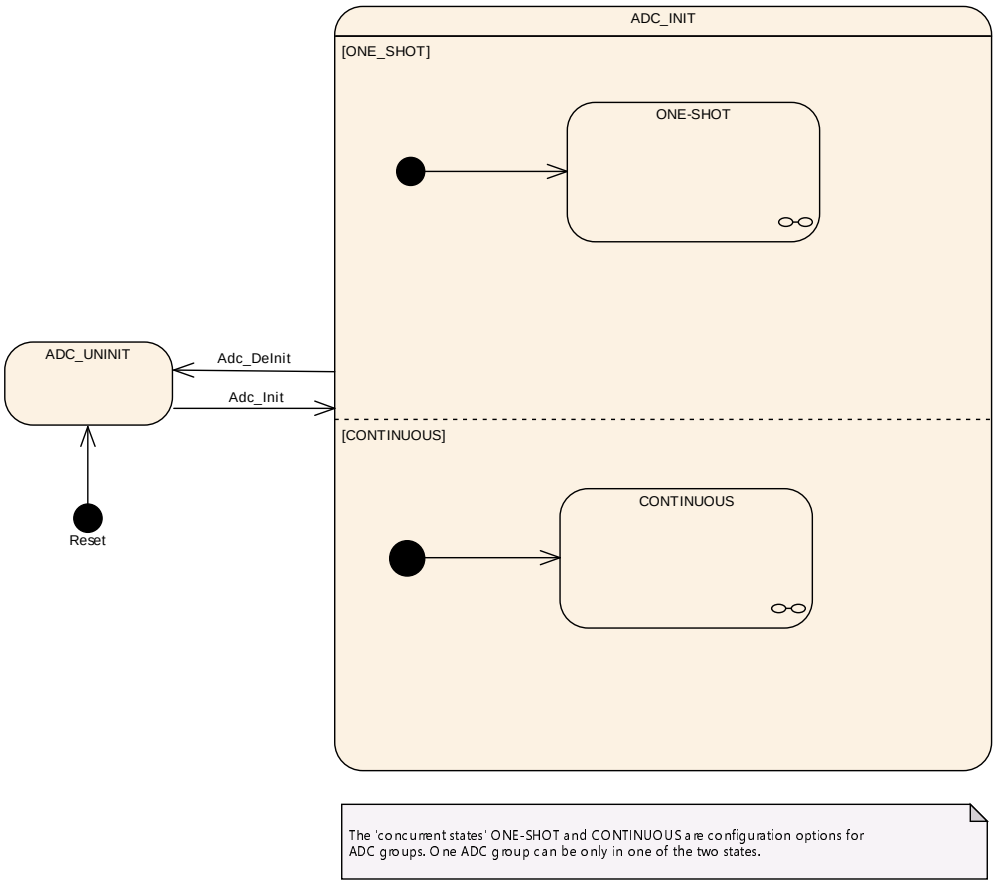


Figure 7.4: ADC State Diagram for One-Shot/Continuous Group Conversion Mode

7.3.2 ADC State Diagram for HW/SW Trigger in One-Shot Group Conversion Mode

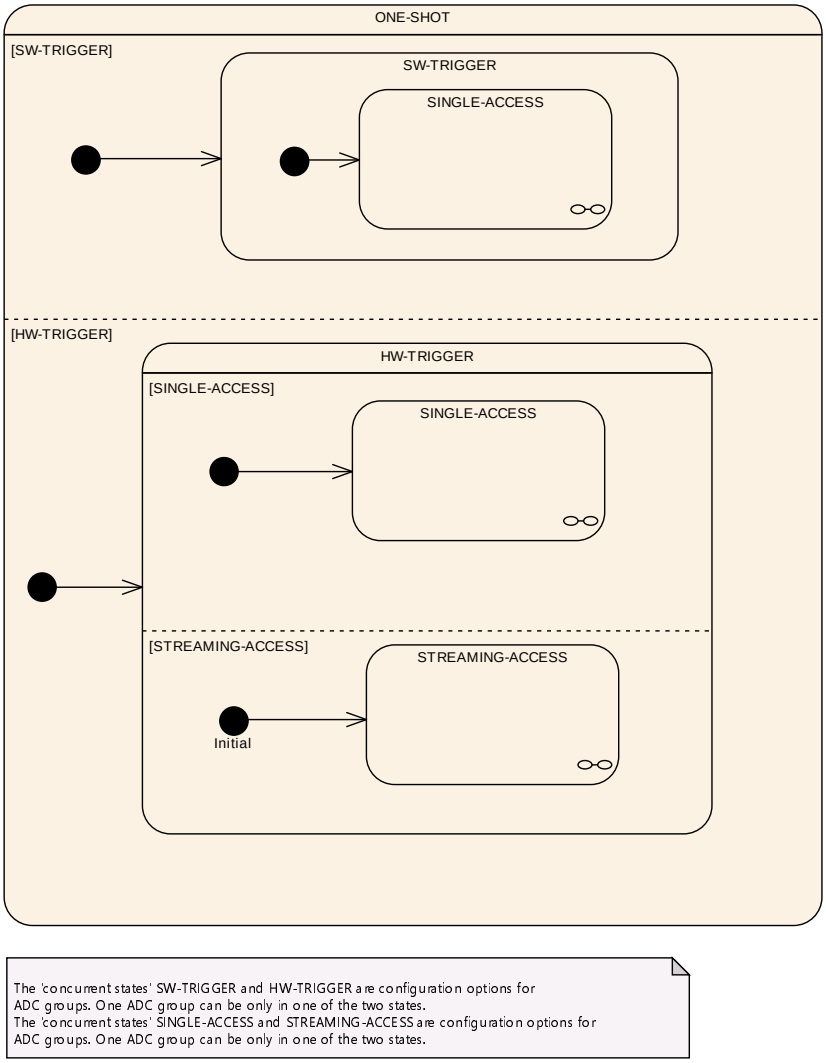


Figure 7.5: State Diagram HW/SW Trigger in One-Shot Group Conversion Mode

7.3.3 ADC State Diagram for SW Trigger in Continuous Conversion Mode

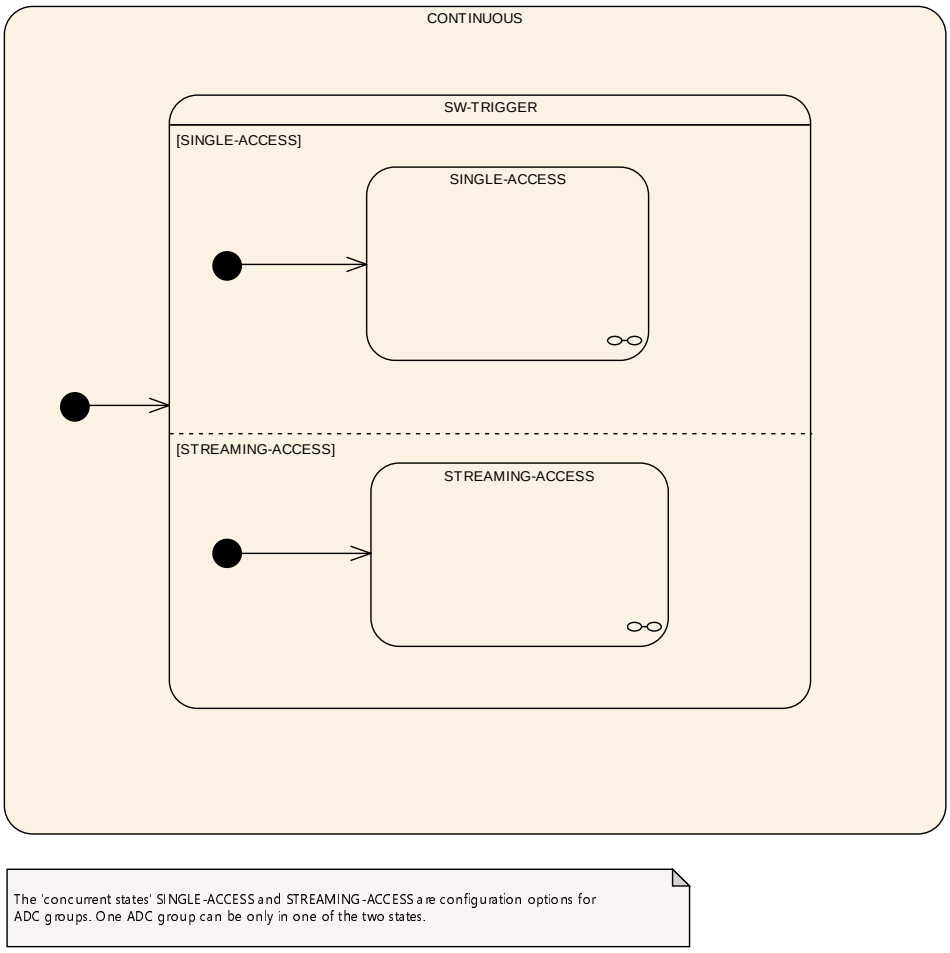


Figure 7.6: State Diagram SW Trigger in Continuous Conversion Mode

7.3.4 ADC State Diagram for One-Shot Conversion Mode, Software Trigger Source, Single Access Mode

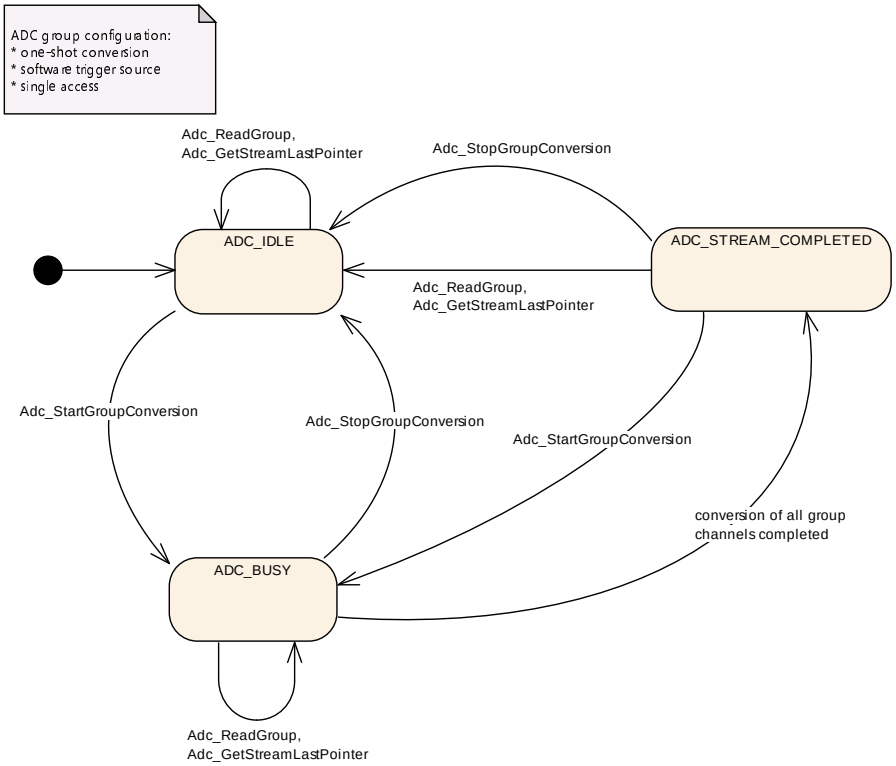


Figure 7.7: State Diagram On-Shot, SW Trigger, Single Access

7.3.5 ADC State Diagram for One-Shot Conversion, Hardware Trigger Source, Single Access Mode

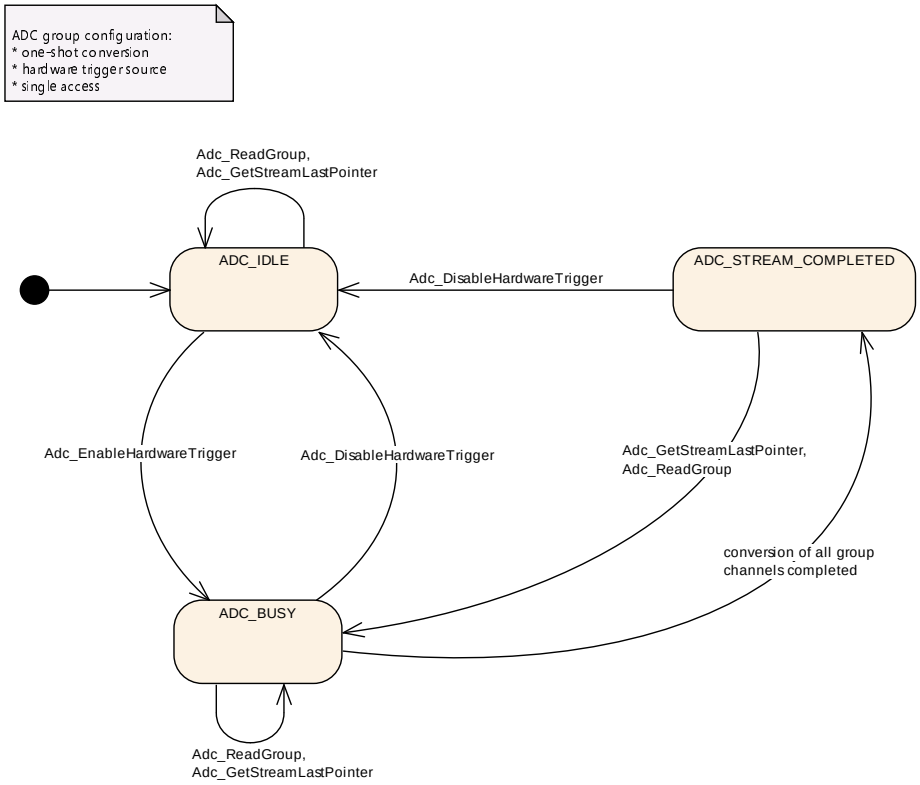


Figure 7.8: State Diagram One-Shot, HW Trigger, Single Access

7.3.6 ADC State Diagram for One-Shot Conversion Mode, Hardware Trigger Source, Linear and Circular Streaming Access Mode

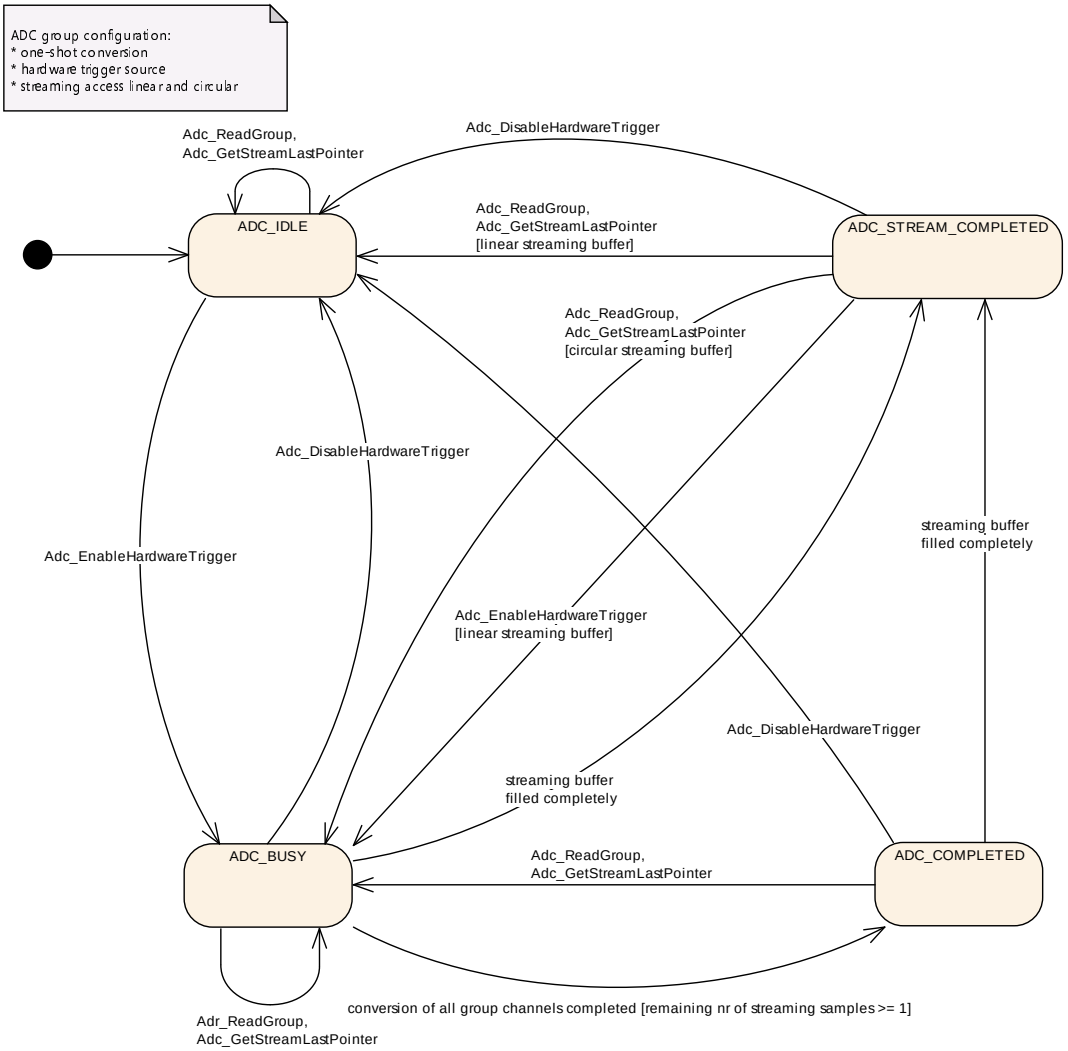


Figure 7.9: State Diagram One-Shot, HW Trigger, Streaming Access

7.3.7 ADC State Diagram for Continuous Conversion Mode, Software Trigger Source, Single Access Mode

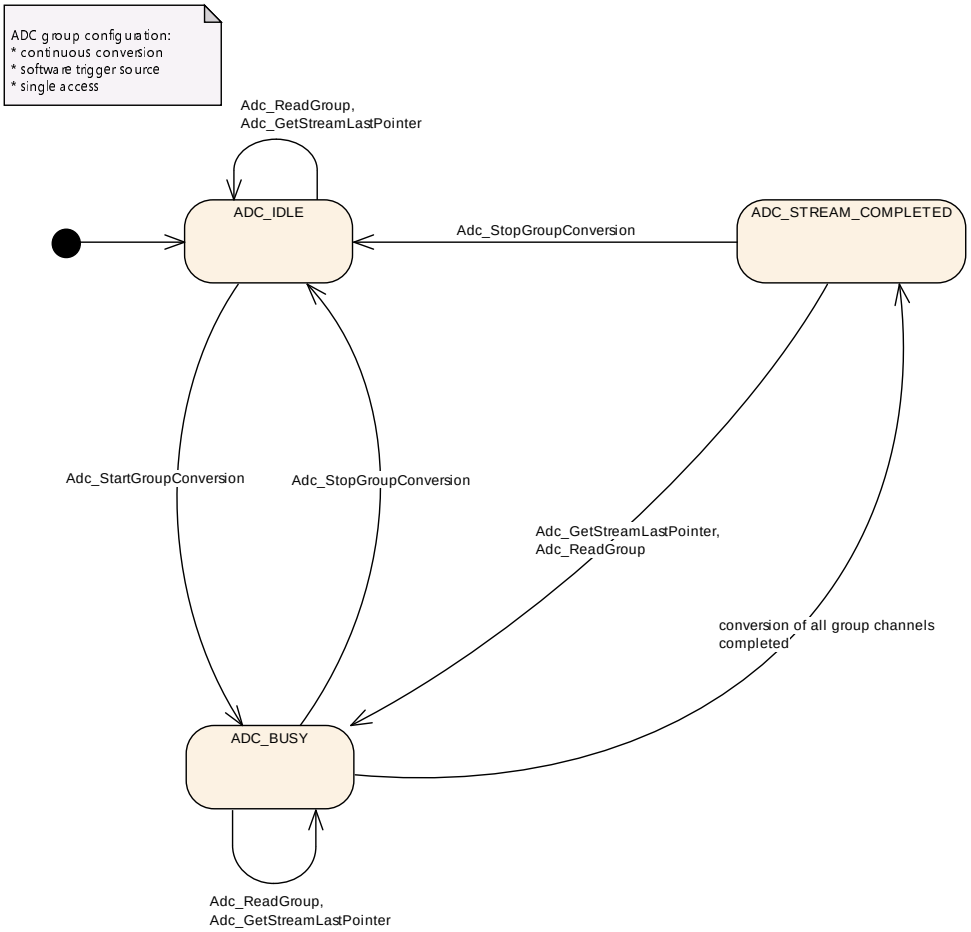


Figure 7.10: State Diagram Continuous, SW Trigger, Single Access

7.3.8 ADC State Diagram for Continuous Conversion Mode, Software Trigger Source, Linear and Circular Streaming Access Mode

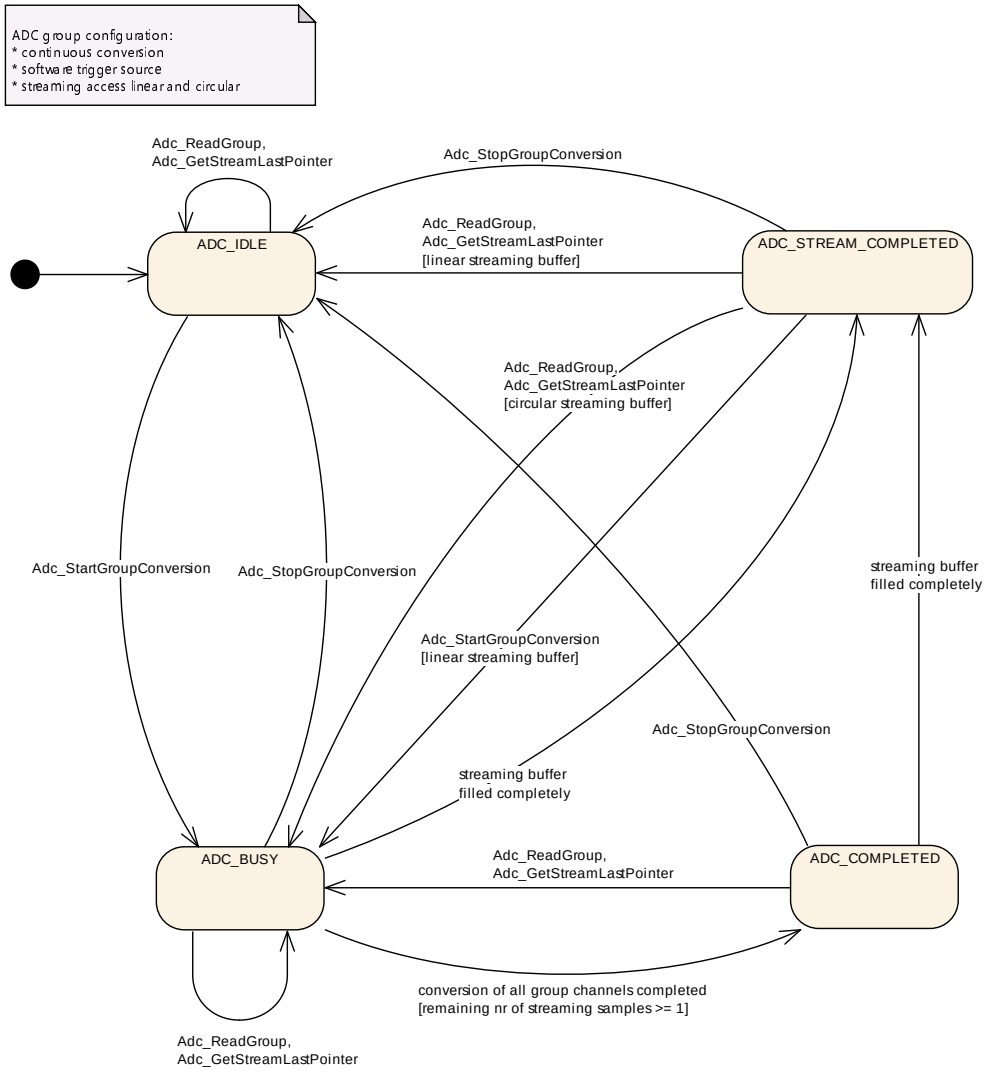


Figure 7.11: State Diagram Conversion, SW Trigger, Streaming Access

7.4 Support and management of HW low power states

Some ADC HW Module allow to be set in some operation modes which reduce the power consumption, eventually at the cost of a slower reaction time, a lower performance or eventually complete unavailability. Each ADC module could support one or more low power operation modes, considering the Full Power Mode as always present and set per default at startup.

7.4.1 Background

The ADC Driver offers power state control APIs and a background elaboration mechanism to handle asynchronous power state change processes (i.e. power state changes which are not immediately complete as the they are requested, but need some longer operations).

It is assumed that all constraints deriving from ECU and SW architecture are already satisfied by the upper layers (Application, Mode Management in the service layer, Io HwAbstraction components dealing with peripheral control), thus the scope of control is limited to the ADC HW peripheral.

A check on the operation sequence is executed by the ADC Driver in order to avoid requesting a different power state before the previous request is still being processed or activating a power state when no preparation for the same has been requested.

The ADC module shall support power control capabilities as an optional function. This module neither mandates to use only power control enabled MCUs nor to configure the same. Rather it proposes a way to handle power states if this is supported by the suppliers.

7.4.2 Requirements

[SWS_Adc_00462] ⌈The ADCDriver shall support power state changes and its APIs when the corresponding configuration parameter AdcLowPowerStatesSupport is set to TRUE.⌋*()*

[SWS_Adc_00463] ⌈If the parameter AdcLowPowerStatesSupport is enabled then the APIs Adc_PreparePowerState, Adc_SetPowerState, Adc_GetCurrentPowerState, Adc_GetTargetPowerState shall be generated and shall be used to manage and get informations on power state transitions.⌋*()*

[SWS_Adc_00464] ⌈The APIs Adc_GetTargetPowerState and Adc_GetCurrentPower State shall be respectively used to gather information on the requested and the target ADC power states.⌋*()*

[SWS_Adc_00465] ⌈The API Adc_PreparePowerState shall be used to start a power state transition.⌋*()*

[SWS_Adc_00466] ⌈After preparation for a power state is achieved by API Adc_PreparePowertState then the API Adc_SetPowerState shall be used to achieve the requested power state of the ADC module.

In order to avoid incoherent power state conditions, some APIs (Adc_SetPowerState, Adc_PreparePowerState) have to be called in a given sequence, otherwise an error (if DET tracing is enabled) is stored and the action is interrupted. The ADC Driver keeps track of the call sequence.⌋*()*

[SWS_Adc_00467] ⌈ADC Driver shall keep track of the call order of the APIs Adc_Set PowerState and Adc_PreparePowerState. In case the first one is called before the second one is called, a DET entry shall be stored and the action shall not be executed.⌋ *()*

[SWS_Adc_00469] ⌈The Adc Module shall keep track of the current and of the target powerstate if the parameter AdcLowPowerStatesSupport is set to TRUE.⌋*()*

[SWS_Adc_00470] ⌈After the Initiliazation the power state of the module shall be always FULL POWER if the AdcLowPowerStatesSupport is set to TRUE.⌋*()*

[SWS_Adc_00471] ⌈The ADC Driver shall support synchronuous and asynchronous power state transitions, depending on the value of the configuration parameter Adc PowerStateAsynchTransitionMode.⌋*()*

[SWS_Adc_00472] ⌈In case the configuration parameter AdcPowerStateAsynchTransitionMode is set to FALSE, the preparation process and the setting process shall be considered concluded as soon as the respective APIs return.⌋*()*

[SWS_Adc_00473] ⌈In case the configuration parameter AdcPowerStateAsynchTransitionMode is set to TRUE, the preparation process shall continue in background after the relative API returns and its completion shall be notified by means of the configured callback.⌋*()*

7.5 Error Classification

Section 7.x "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.:

7.5.1 Development Errors

[SWS_Adc_91005] ⌈

Type of error	*Related error code*	*Error value*
API is called prior to initialization.	ADC_E_UNINIT	0x0A
API called while ADC is already initialized.	ADC_E_ALREADY_INITIALIZED	0x0D
API called with incorrect buffer pointer.	ADC_E_PARAM_POINTER	0x14
API called with non existing group.	ADC_E_PARAM_GROUP	0x15
API called for a group configured for continuous conversion mode.	ADC_E_WRONG_CONV_MODE	0x16

▽

△

Type of error	*Related error code*	*Error value*
API call not allowed according group configuration.	ADC_E_WRONG_TRIGG_SRC	0x17
API called and notification function pointer is NULL.	ADC_E_NOTIF_CAPABILITY	0x18
API called while result buffer pointer is not initialized.	ADE_E_BUFFER_UNINIT	0x19
API call with unsupported power state request.	ADE_E_POWER_STATE_NOT_SUPPORTED	0x1B
ADC not prepared for requested target power state.	ADC_E_PERIPHERAL_NOT_PREPARED	0x1D

⌋()

7.5.2 Runtime Errors

[SWS_Adc_91006] ⌈

Type of error	*Related error code*	*Error value*
API is called while another conversion is already running, a HW trigger is already enabled, a request is already stored in the queue.	ADC_E_BUSY	0x0B
API is called while group is in state ADC_IDLE or non enabled group.	ADC_E_IDLE	0x0C
API called while one or more ADC groups are not in IDLE state.	ADC_E_NOT_DISENGAGED	0x1A
Requested power state can not be reached.	ADC_E_TRANSITION_NOT_POSSIBLE	0x1C

⌋()

7.5.3 Transient Faults

There are no transient faults.

7.5.4 Production Errors

There are no production errors.

7.5.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed:

[SWS_Adc_00364] ⌈

Module	*Header File*	*Imported Type*
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋()

8.2 Type definitions

8.2.1 Adc_ConfigType

[SWS_Adc_00505] ⌈

Name	Adc_ConfigType	
Kind	Structure	
Elements	–	
	Type	–
	Comment	Implementation specific configuration data structure.
Description	Data structure containing the set of configuration parameters required for initializing the ADC Driver and ADC HW Unit(s).	
Available via	Adc.h	

⌋()

8.2.2 Adc_ChannelType

[SWS_Adc_00506] ⌈

Name	Adc_ChannelType		
Kind	Type		
Derived from	uint		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.
Description	Numeric ID of an ADC channel.		
Available via	Adc.h		

⌋()

8.2.3 Adc_GroupType

[SWS_Adc_00507] ⌈

Name	Adc_GroupType		
Kind	Type		
Derived from	uint		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.
Description	Numeric ID of an ADC channel group.		
Available via	Adc.h		

⌋()

8.2.4 Adc_ValueGroupType

[SWS_Adc_00508] ⌈

Name	Adc_ValueGroupType		
Kind	Type		
Derived from	int		
Range	–	–	Implementation specific.
Description	Type for reading the converted values of a channel group (raw, without further scaling, alignment according precompile switch ADC_RESULT_ALIGNMENT).		
Available via	Adc.h		

⌋() The result values shall be stored in an integer buffer, i.e. an array of integers.

The following rules shall apply to the driver implementation:

- **[SWS_Adc_00318]** ⌈In single value access mode the result buffer shall have as many elements as channels belonging to the group. In this way each buffer element corresponds to a channel, in the order the channels are defined in the group.⌋*(SRS_Adc_12819)*
- **[SWS_Adc_00319]** ⌈In streaming access mode the result buffer shall have m*n elements, where n is the number of channels belonging to the group, m the number of samples acquired per channel. In this way the first m elements belong to the first channel in the group, the second m elements to the second channel and so on.⌋*(SRS_Adc_12825)*
- **[SWS_Adc_00320]** ⌈The dimension (in number of bits) of each buffer element (of type integer) shall be uniform, tailored on the largest (in number of bits) channel belonging to any group.⌋*(SRS_Adc_12822)*

Note: Only if all ADC channels of all ADC groups have 8 bit resolution,

Adc_ValueGroupType can be configured as 8 bit data type.

Note: The information about number of channels belonging to the group and number of samples acquired per channel can be derived from the group configuration data.

8.2.5 Adc_PrescaleType

[SWS_Adc_00509] ⌈

Name	Adc_PrescaleType		
Kind	Type		
Derived from	uint		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.
Description	Type of clock prescaler factor. (This is not an API type).		
Available via	Adc.h		

⌋()

8.2.6 Adc_ConversionTimeType

[SWS_Adc_00510] ⌈

Name	Adc_ConversionTimeType		
Kind	Type		
Derived from	uint		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.
Description	Type of conversion time, i.e. the time during which the sampled analogue value is converted into digital representation. (This is not an API type).		
Available via	Adc.h		

⌋()

8.2.7 Adc_SamplingTimeType

[SWS_Adc_00511] ⌈

Name	Adc_SamplingTimeType		
Kind	Type		
Derived from	uint		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.

▽

△

Description	Type of sampling time, i.e. the time during which the value is sampled, (in clock-cycles). (This is not an API type).
Available via	Adc.h

⌋()

8.2.8 Adc_ResolutionType

[SWS_Adc_00512] ⌈

Name	Adc_ResolutionType		
Kind	Type		
Derived from	uint8		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.
Description	Type of channel resolution in number of bits. (This is not an API type).		
Available via	Adc.h		

⌋()

8.2.9 Adc_StatusType

[SWS_Adc_00513] ⌈

Name	Adc_StatusType		
Kind	Enumeration		
Range	ADC_IDLE	0x00	• The conversion of the specified group has not been started. • No result is available.
	ADC_BUSY	0x01	• The conversion of the specified group has been started and is still going on. • So far no result is available.
	ADC_COMPLETED	0x02	• A conversion round (which is not the final one) of the specified group has been finished. • A result is available for all channels of the group.
	ADC_STREAM_ COMPLETED	0x03	• The result buffer is completely filled • For each channel of the selected group the number of samples to be acquired is available
Description	Current status of the conversion of the requested ADC Channel group.		
Available via	Adc.h		

⌋()

8.2.10 Adc_TriggerSourceType

[SWS_Adc_00514] ⌈

Name	Adc_TriggerSourceType		
Kind	Enumeration		
Range	ADC_TRIGG_SRC_SW	0x00	Group is triggered by a software API call.
	ADC_TRIGG_SRC_HW	0x01	Group is triggered by a hardware event.
Description	Type for configuring the trigger source for an ADC Channel group.		
Available via	Adc.h		

⌋()

8.2.11 Adc_GroupConvModeType

[SWS_Adc_00515] ⌈

Name	Adc_GroupConvModeType		
Kind	Enumeration		
Range	ADC_CONV_MODE_ ONESHOT	0x00	Exactly one conversion of each channel in an ADC channel group is performed after the configured trigger event. In case of 'group trigger source software', a started One-Shot conversion can be stopped by a software API call. In case of 'group trigger source hardware', a started One-Shot conversion can be stopped by disabling the trigger event (if supported by hardware).
	ADC_CONV_MODE_ CONTINUOUS	0x01	Repeated conversions of each ADC channel in an ADC channel group are performed. 'Continuous conversion mode' is only available for 'group trigger source software'. A started 'Continuous conversion' can be stopped by a software API call.
Description	Type for configuring the conversion mode of an ADC Channel group.		
Available via	Adc.h		

⌋()

8.2.12 Adc_GroupPriorityType

[SWS_Adc_00516] ⌈

Name	Adc_GroupPriorityType		
Kind	Type		
Derived from	uint8		
Range	0..255	–	–
Description	Priority level of the channel. Lowest priority is 0.		
Available via	Adc.h		

⌋()

8.2.13 Adc_GroupDefType

[SWS_Adc_00517] ⌈

Name	Adc_GroupDefType
Kind	Type
Derived from	implementation_specific
Description	Type for assignment of channels to a channel group (this is not an API type).
Available via	Adc.h

⌋()

8.2.14 Adc_StreamNumSampleType

[SWS_Adc_00518] ⌈

Name	Adc_StreamNumSampleType		
Kind	Type		
Derived from	uint		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.
Description	Type for configuring the number of group conversions in streaming access mode (in single access mode, parameter is 1).		
Available via	Adc.h		

⌋()

8.2.15 Adc_StreamBufferModeType

[SWS_Adc_00519] ⌈

Name	Adc_StreamBufferModeType		
Kind	Enumeration		
Range	ADC_STREAM_BUFFER_LINEAR	0x00	The ADC Driver stops the conversion as soon as the stream buffer is full (number of samples reached).
	ADC_STREAM_BUFFER_CIRCULAR	0x01	The ADC Driver continues the conversion even if the stream buffer is full (number of samples reached) by wrapping around the stream buffer itself.
Description	Type for configuring the streaming access mode buffer type.		
Available via	Adc.h		

⌋()

8.2.16 Adc_GroupAccessModeType

[SWS_Adc_00528] ⌈

Name	Adc_GroupAccessModeType		
Kind	Enumeration		
Range	ADC_ACCESS_MODE_SINGLE	0x00	Single value access mode.
	ADC_ACCESS_MODE_STREAMING	0x01	Streaming access mode.
Description	Type for configuring the access mode to group conversion results.		
Available via	Adc.h		

⌋()

8.2.17 Adc_HwTriggerSignalType

[SWS_Adc_00520] ⌈

Name	Adc_HwTriggerSignalType		
Kind	Enumeration		
Range	ADC_HW_TRIG_RISING_EDGE	0x00	React on the rising edge of the hardware trigger signal (only if supported by the ADC hardware).
	ADC_HW_TRIG_FALLING_EDGE	0x01	React on the falling edge of the hardware trigger signal (only if supported by the ADC hardware).
	ADC_HW_TRIG_BOTH_EDGES	0x02	React on both edges of the hardware trigger signal (only if supported by the ADC hardware).
Description	Type for configuring on which edge of the hardware trigger signal the driver should react, i.e. start the conversion (only if supported by the ADC hardware).		
Available via	Adc.h		

⌋()

8.2.18 Adc_HwTriggerTimerType

[SWS_Adc_00521] ⌈

Name	Adc_HwTriggerTimerType		
Kind	Type		
Derived from	uint		
Range	–	–	The range of this type is μC specific and has to be described by the supplier.

▽

△

Description	Type for the reload value of the ADC module embedded timer (only if supported by the ADC hardware).
Available via	Adc.h

⌋()

8.2.19 Adc_PriorityImplementationType

[SWS_Adc_00522] ⌈

Name	Adc_PriorityImplementationType		
Kind	Enumeration		
Range	ADC_PRIORITY_NONE	0x00	priority mechanism is not available
	ADC_PRIORITY_HW	0x01	Hardware priority mechanism is available only
	ADC_PRIORITY_HW_SW	0x02	Hardware and software priority mechanism is available
Description	Type for configuring the prioritization mechanism.		
Available via	Adc.h		

⌋()

8.2.20 Adc_GroupReplacementType

[SWS_Adc_00523] ⌈

Name	Adc_GroupReplacementType		
Kind	Enumeration		
Range	ADC_GROUP_REPL_ ABORT_RESTART	0x00	Abort/Restart mechanism is used on group level, if a group is interrupted by a higher priority group. The complete conversion round of the interrupted group (all group channels)is restarted after the higher priority group conversion is finished. If the group is configured in streaming access mode, only the results of the interrupted conversion round are discarded. Results of previous conversion rounds which are already written to the result buffer are not affected.
	ADC_GROUP_REPL_ SUSPEND_RESUME	0x01	Suspend/Resume mechanism is used on group level, if a group is interrupted by a higher priority group. The conversion round of the interrupted group is completed after the higher priority group conversion is finished. Results of previous conversion rounds which are already written to the result buffer are not affected.
Description	Replacement mechanism, which is used on ADC group level, if a group conversion is interrupted by a group which has a higher priority.		
Available via	Adc.h		

⌋()

8.2.21 Adc_ChannelRangeSelectType

[SWS_Adc_00524] ⌈

Name	Adc_ChannelRangeSelectType		
Kind	Enumeration		
Range	ADC_RANGE_UNDER_LOW	0x00	Range below low limit - low limit value included
	ADC_RANGE_BETWEEN	0x01	Range between low limit and high limit - high limit value included
	ADC_RANGE_OVER_HIGH	0x02	Range above high limit
	ADC_RANGE_ALWAYS	0x03	Complete range - independent from channel limit settings
	ADC_RANGE_NOT_UNDER_LOW	0x04	Range above low limit
	ADC_RANGE_NOT_BETWEEN	0x05	Range above high limit or below low limit - low limit value included
	ADC_RANGE_NOT_OVER_HIGH	0x06	Range below high limit - high limit value included
Description	In case of active limit checking: defines which conversion values are taken into account related to the boardes defineed with AdcChannelLowLimit and AdcChannelHighLimit.		
Available via	Adc.h		

⌋()

8.2.22 Adc_ResultAlignmentType

[SWS_Adc_00525] ⌈

Name	Adc_ResultAlignmentType		
Kind	Enumeration		
Range	ADC_ALIGN_LEFT	0x00	left alignment
	ADC_ALIGN_RIGHT	0x01	right alignment
Description	Type for alignment of ADC raw results in ADC result buffer (left/right alignment).		
Available via	Adc.h		

⌋()

8.2.23 Adc_PowerStateType

[SWS_Adc_00526] ⌈

Name	Adc_PowerStateType		
Kind	Enumeration		
Range	1..255	–	power modes with decreasing power consumptions.
	ADC_FULL_POWER	0	Full Power

▽

△

Description	Power state currently active or set as target power state.
Available via	Adc.h

⌋()

8.2.24 Adc_PowerStateRequestResultType

[SWS_Adc_00527] ⌈

Name	Adc_PowerStateRequestResultType		
Kind	Enumeration		
Range	ADC_SERVICE_ ACCEPTED	0	Power state change executed.
	ADC_NOT_INIT	1	ADC Module not initialized.
	ADC_SEQUENCE_ERROR	2	Wrong API call sequence.
	ADC_HW_FAILURE	3	The HW module has a failure which prevents it to enter the required power state.
	ADC_POWER_STATE_ NOT_SUPP	4	ADC Module does not support the requested power state.
	ADC_TRANS_NOT_ POSSIBLE	5	ADC Module cannot transition directly from the current power state to the requested power state or the HW peripheral is still busy.
Description	Result of the requests related to power state transitions.		
Available via	Adc.h		

⌋()

8.3 Function definitions

8.3.1 Adc_Init

[SWS_Adc_00365] ⌈

Service Name	Adc_Init	
Syntax	`void Adc_Init ( const Adc_ConfigType* ConfigPtr )`	
Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration set in Variant PB (Variant PC requires a NULL_PTR).
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	Initializes the ADC hardware units and driver.
Available via	Adc.h

⌋*()*

[SWS_Adc_00054] ⌈In case of Variant PB: The function Adc_Init shall initialize the ADC hardware units and driver according to the configuration set referenced by Config Ptr.⌋*(SRS_BSW_00405, SRS_BSW_00101, SRS_BSW_00414, SRS_SPAL_12057, SRS_SPAL_12461)*

[SWS_Adc_00056] ⌈The function Adc_Init shall only initialize the configured resources. Resources that are not contained in the configuration file shall not be touched.⌋*(SRS_SPAL_12125)*

The following rules regarding initialization of controller registers apply to this driver implementation:

- **[SWS_Adc_00246]** ⌈If the hardware allows for only one usage of the register, the driver module implementing that functionality is responsible for initializing the register.⌋*(SRS_SPAL_12461)*
- **[SWS_Adc_00247]** ⌈If the register can affect several hardware modules and if it is an I/O register, it shall be initialized by the PORT driver.⌋*(SRS_SPAL_12461)*
- **[SWS_Adc_00248]** ⌈If the register can affect several hardware modules and if it is not an I/O register, it shall be initialized by the MCU driver.⌋*(SRS_SPAL_12461)*
- **[SWS_Adc_00249]** ⌈One-time writable registers that require initialization directly after reset shall be initialized by the startup code.⌋*(SRS_SPAL_12461)*
- **[SWS_Adc_00250]** ⌈All other registers shall be initialized by the startup code.⌋*(SRS_SPAL_12461)*

[SWS_Adc_00077] ⌈The function Adc_Init shall disable the notifications and hardware trigger capability (if statically configured as active).⌋*(SRS_Adc_12318)*

[SWS_Adc_00307] ⌈The function Adc_Init shall set all groups to ADC_IDLE state.⌋*()*

[SWS_Adc_00107] ⌈If development error detection for the ADC module is enabled:if called when the ADC driver and hardware are already initialized, the function Adc_Init shall raise development error ADC_E_ALREADY_INITIALIZED and return without any action.⌋*(SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448)*

8.3.2 Adc_SetupResultBuffer

[SWS_Adc_91000] ⌈

Service Name	Adc_SetupResultBuffer	
Syntax	`Std_ReturnType Adc_SetupResultBuffer ( Adc_GroupType Group, Adc_ValueGroupType* DataBufferPtr )`	
Service ID [hex]	0x0c	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC channel group.
	DataBufferPtr	pointer to result data buffer
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: result buffer pointer initialized correctly E_NOT_OK: operation failed or development error occured
Description	Initializes ADC driver with the group specific result buffer start address where the conversion results will be stored. The application has to ensure that the application buffer, where Data BufferPtr points to, can hold all the conversion results of the specified group. The initialization with Adc_SetupResultBuffer is required after reset, before a group conversion can be started.	
Available via	Adc.h	

⌋*()*

[SWS_Adc_00420] ⌈The function Adc_SetupResultBuffer shall initialize the result buffer pointer of the selected group with the address value passed as parameter.⌋ *()*

[SWS_Adc_00421] ⌈The ADC module's environment shall ensure that no group conversions are started without prior initialization of the according result buffer pointer to point to a valid result buffer.⌋*()*

[SWS_Adc_00422] ⌈The ADC module's environment shall ensure that the application buffer, which address is passed as parameter in Adc_SetupResultBuffer, has the according size to hold all group channel conversion results and if streaming access is selected, hold these results multiple times as specified with streaming sample parameter (see ADC292).⌋*()*

[SWS_Adc_00423] ⌈If development error detection for the ADC module is enabled: if the channel group ID is non-existing, the function Adc_SetupResultBuffer shall raise development error ADC_E_PARAM_GROUP and return without any action.⌋*()*

[SWS_Adc_00433] ⌈If called while group is not in state ADC_IDLE, function Adc_SetupResultBuffer shall report a runtime error ADC_E_BUSY.⌋*()*

[SWS_Adc_00434] ⌈If development error detection for the ADC module is enabled: when called prior to initializing the driver, the function Adc_SetupResultBuffer shall raise development error ADC_E_UNINIT.⌋*()*

[SWS_Adc_00457] ⌈If development error detection for the ADC module is enabled: when called with a NULL_PTR as DataBufferPtr, the function Adc_SetupResultBuffer shall raise development error ADC_E_PARAM_POINTER.⌋*()*

8.3.3 Adc_DeInit

[SWS_Adc_00366] ⌈

Service Name	Adc_DeInit
Syntax	`void Adc_DeInit ( void )`
Service ID [hex]	0x01
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Returns all ADC HW Units to a state comparable to their power on reset state.
Available via	Adc.h

⌋*()*

[SWS_Adc_00110] ⌈The function Adc_DeInit shall return all ADC HW Units to a state comparable to their power on reset state. Values of registers which are not writeable are excluded. It's the responsibility of the hardware design that this state does not lead to undefined activities in the μC.⌋*(SRS_SPAL_12163)*

[SWS_Adc_00111] ⌈The function Adc_DeInit shall disable all used interrupts and notifications.⌋*(SRS_BSW_00336, SRS_SPAL_12163)*

[SWS_Adc_00358] ⌈The ADC module's environment shall not call the function Adc_DeInit while any group is not in state ADC_IDLE.⌋*()*

[SWS_Adc_00228] ⌈The function Adc_DeInit shall be pre compile time configurable On/Off by the configuration parameter: AdcDeInitApi.⌋*(SRS_BSW_00171)*

[SWS_Adc_00112] ⌈If calledwhile not all groups are either in state ADC_IDLE or state ADC_STREAM_COMPLETED, while no conversion is ongoing (ADC groups which are implicitly stopped), the function Adc_DeInit shall report a runtime error.⌋*()*

[SWS_Adc_00154] ⌈If development error detection for the ADC module is enabled: if called before the module has been initialized, the function Adc_DeInit shall raise development error ADC_E_UNINIT and return without any action.⌋*(SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448)*

8.3.4 Adc_StartGroupConversion

[SWS_Adc_00367] ⌈

Service Name	Adc_StartGroupConversion	
Syntax	`void Adc_StartGroupConversion ( Adc_GroupType Group )`	
Service ID [hex]	0x02	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Starts the conversion of all channels of the requested ADC Channel group.	
Available via	Adc.h	

⌋*()*

[SWS_Adc_00061] ⌈The function Adc_StartGroupConversion shall start the conversion of all channels of the requested ADC Channel group. Depending on the group configuration, one-shot or continuous conversion is started.⌋*(SRS_Adc_12364)*

[SWS_Adc_00431] ⌈The function Adc_StartGroupConversion shall reset the internal result buffer pointer, that conversion result storage always starts, after calling Adc_StartGroupConversion, at the result buffer base address which was configured with Adc_SetupResultBuffer.⌋*()*

[SWS_Adc_00156] ⌈The function Adc_StartGroupConversion shall NOT automatically enable the notification mechanism for that group (this has to be done by a separate API call).⌋*(SRS_Adc_12317, SRS_Adc_12318)*

[SWS_Adc_00146] ⌈The ADC module's environment shall only call Adc_StartGroupConversion for groups configured with software trigger source.⌋*(SRS_Adc_12817, SRS_Adc_12364)*

[SWS_Adc_00259] ⌈The function Adc_StartGroupConversion shall be pre-compile time configurable On/Off by the configuration parameter AdcEnableStartStopGroupApi.⌋*(SRS_BSW_00171)*

[SWS_Adc_00125] ⌈If development error detection for the ADC module is enabled:when called with a non-existing channel group ID, function Adc_StartGroupConversion shall raise development error ADC_E_PARAM_GROUP and return without any action.⌋*(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00133] ⌈If development error detection for the ADC module is enabled: when called on a group with trigger source configured as hardware, function Adc_StartGroupConversion shall raise development error ADC_E_WRONG_TRIGG_SRC and return without any action.⌋*(SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00346] ⌈If the priority mechanism is disabled and the queuing is disabled : when called while any of the groups, which can not be implicitly stopped, is not in state ADC_IDLE, the function Adc_StartGroupConversion shall report a runtime error ADC_E_BUSY.⌋*()*

Note: The condition that any group is not in state ADC_IDLE means in this context:

- Any conversion is ongoing

or

- Any HW trigger is enabled

[SWS_Adc_00426] ⌈If the priority mechanism is disabled and the queuing is disabled: when called while any of the groups, which can be implicitly stopped, is not in state ADC_IDLE and not in state ADC_STREAM_COMPLETED, the function Adc_StartGroupConversion shall report a runtime error ADC_E_BUSY.⌋*()*

Note: Groups which can be implicitly stopped are:

- Software triggered groups configured in one-shot, single-access mode
- Software triggered groups configured in continuous, linear streaming access mode
- Hardware triggered groups configured in one-shot, linear streaming access mode

[SWS_Adc_00348] ⌈If the priority mechanism is enabled: when called while agroup, which can not be implicitly stopped, is not in state ADC_IDLE, the function Adc_StartGroupConversion shall report a runtime error ADC_E_BUSY.⌋*()*

Note: The condition that the group is not in state ADC_IDLE means in this context:

- The conversion of the same group is currently ongoing

or

- A conversion request for the same group is already stored one time in the queue

[SWS_Adc_00427] ⌈If the priority mechanism is enabled: when called while a group, which can be implicitly stopped, is not in state ADC_IDLE and not in state ADC_STREAM_COMPLETED, the function Adc_StartGroupConversion shall report a runtime error ADC_E_BUSY.⌋*()*

[SWS_Adc_00351] ⌈If the priority mechanism is disabled and the queuing is enabled: when called while a group, which can not be implicitly stopped, is not in state ADC_IDLE, the function Adc_StartGroupConversion shall report a runtime error ADC_E_BUSY.⌋*()*

[SWS_Adc_00428] ⌈If the priority mechanism is disabled and the queuing is enabled: when called while a group, which can be implicitly stopped, is not in state ADC_IDLE and not in state ADC_STREAM_COMPLETED, the function Adc_StartGroupConversion shall report a runtime error ADC_E_BUSY.⌋*()*

[SWS_Adc_00294] ⌈If development error detection for the ADC module is enabled:when called prior to initializing the driver, the function Adc_StartGroupConversion shall raise development error ADC_E_UNINIT.⌋*(SRS_BSW_00406)*

[SWS_Adc_00424] ⌈If development error detection for the ADC module is enabled: when called prior to initializing the result buffer pointer with function Adc_SetupResult Buffer, the function Adc_StartGroupConversion shall raise development error ADC_E_BUFFER_UNINIT.⌋*()*

8.3.5 Adc_StopGroupConversion

[SWS_Adc_00368] ⌈

Service Name	Adc_StopGroupConversion	
Syntax	`void Adc_StopGroupConversion ( Adc_GroupType Group )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Stops the conversion of the requested ADC Channel group.	
Available via	Adc.h	

⌋*()*

[SWS_Adc_00385] ⌈When the ADC Channel Group is in one-shot and software-trigger mode, the function Adc_StopGroupConversion shall stop an ongoing conversion of the group.⌋*(SRS_Adc_12364)*

[SWS_Adc_00437] ⌈When the ADC Channel Group is in one-shot and software-trigger mode, the function Adc_StopGroupConversion shall remove a start/restart request of the group from the queue, if queuing is enabled and a start/restart request is stored in the queue.⌋*()*

[SWS_Adc_00386] ⌈When the ADC Channel Group is in continuous-conversion and software-trigger mode, the function Adc_StopGroupConversion shall stop an ongoing conversion of the group.⌋*(SRS_Adc_12364)*

[SWS_Adc_00438] ⌈When the ADC Channel Group is in continuous-conversion and software-trigger mode, the function Adc_StopGroupConversion shall remove a start/restart request of the group from the queue, if queuing is enabled and a start/restart request is stored in the queue.⌋*()*

[SWS_Adc_00155] ⌈The function Adc_StopGroupConversion shall automatically disable group notification for the requested group.⌋*(SRS_Adc_12317)*

Note:

Groups which are implicitly stopped shall not disable the group notification until Adc_StopGroupConversion is called.

[SWS_Adc_00360] ⌈The function Adc_StopGroupConversion shall set the group status to state ADC_IDLE.⌋*()*

[SWS_Adc_00283] ⌈The ADC module's environment shall only call the function Adc_StopGroupConversion for groups configured with trigger source software.⌋*(SRS_Adc_12817)*

[SWS_Adc_00260] ⌈The function Adc_StopGroupConversion shall be pre compile time configurable On/Off by the configuration parameter AdcEnableStartStopGroupApi.⌋*(SRS_BSW_00171)*

[SWS_Adc_00126] ⌈If development error detection for the ADC module is enabled:if the group ID is non-existing, the function Adc_StopGroupConversion shall raise development error ADC_E_PARAM_GROUP and return without any action.⌋*(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00164] ⌈If development error detection for the ADC module is enabled:if the group has a trigger source configured as hardware, function Adc_StopGroupConversion shall raise development error ADC_E_WRONG_TRIGG_SRC and return without any action.⌋*(SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00241] ⌈When called while the group is in state ADC_IDLE, the function Adc_StopGroupConversion shall report a runtime error ADC_E_IDLE.⌋*(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

Note: For groups which are implicitly stopped (groups with conversion mode one-shot or groups with linear streaming buffer mode), state is ADC_STREAM_COMPLETED until results are accessed with Adc_ReadGroup or Adc_GetStreamLastPointer API functions or until group is explicitly stopped by Adc_StopGroupConversion API.

[SWS_Adc_00295] ⌈If development error detection for the ADC module is enabled: if called prior to initializing the module, function Adc_StopGroupConversion shall raise development error ADC_E_UNINIT and return without any action.⌋*(SRS_BSW_00406)*

Note:

All groups which are started with Adc_StartGroupConversion should also be stopped with Adc_StopGroupConversion, before they are started again to reset the group status to ADC_IDLE. Exceptions to this rule are groups which are implicitly stopped because of the selected conversion mode (linear buffer with streaming access mode or one-shot conversion mode with single access).These groups can also be restarted while the group is in state ADC_STREAM_COMPLETED.

8.3.6 Adc_ReadGroup

[SWS_Adc_00369] ⌈

Service Name	Adc_ReadGroup	
Syntax	`Std_ReturnType Adc_ReadGroup ( Adc_GroupType Group, Adc_ValueGroupType* DataBufferPtr )`	
Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC channel group.
Parameters (inout)	None	
Parameters (out)	DataBufferPtr	ADC results of all channels of the selected group are stored in the data buffer addressed with the pointer.
Return value	Std_ReturnType	E_OK: results are available and written to the data buffer E_NOT_OK: no results are available or development error occured
Description	Reads the group conversion result of the last completed conversion round of the requested group and stores the channel values starting at the DataBufferPtr address. The group channel values are stored in ascending channel number order (in contrast to the storage layout of the result buffer if streaming access is configured).	
Available via	Adc.h	

⌋*()*

[SWS_Adc_00075] ⌈The function Adc_ReadGroup shall read the latest available conversion results of the requested group.⌋*()*

[SWS_Adc_00113] ⌈The function Adc_ReadGroup shall read the raw converted values without further scaling. The read values shall be aligned according the configuration parameter setting of ADC_RESULT_ALIGNMENT.⌋*(SRS_SPAL_12063, SRS_Adc_12819, SRS_Adc_12292, SRS_Adc_12824)*

[SWS_Adc_00122] ⌈If applicable, the function Adc_ReadGroup shall mask out all information or diagnostic bits provided by the conversion but not belonging to the conversion results themselves.⌋*(SRS_Adc_12283, SRS_Adc_12819)*

[SWS_Adc_00329] ⌈Calling function Adc_ReadGroup while group status is ADC_STREAM_COMPLETED shall trigger a state transition to ADC_BUSY for continuous conversion modes (single access mode or circular streaming buffer mode) and hardware triggered groups in single access mode or circular streaming access mode.⌋*(SRS_Adc_12291)*

[SWS_Adc_00330] ⌈Calling function Adc_ReadGroup while group status is ADC_STREAM_COMPLETED shall trigger a state transition to ADC_IDLE for software triggered conversion modes which automatically stop the conversion (streaming buffer with linear access mode or one-shot conversion mode with single access) and for the hardware triggered conversion mode in combination with linear streaming access mode.⌋*(SRS_Adc_12291)*

[SWS_Adc_00331] ⌈Calling function Adc_ReadGroup while group status is ADC_COMPLETED shall trigger a state transition to ADC_BUSY.⌋*(SRS_Adc_12291)*

[SWS_Adc_00359] ⌈The function Adc_ReadGroup shall be pre-compile configurable On/Off by the configuration parameter AdcReadGroupApi.⌋*()*

[SWS_Adc_00388] ⌈When called while the group status is ADC_IDLE and the group conversion was not started (no results are available from previous conversions), the function Adc_ReadGroup shall report a runtime error ADC_E_IDLE.⌋*()*

[SWS_Adc_00152] ⌈If development error detection for the ADC module is enabled: if the group ID is non-existing, the function Adc_ReadGroup shall raise development error ADC_E_PARAM_GROUP and return E_NOT_OK.⌋*(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00296] ⌈If development error detection for the ADC module is enabled: when called prior to initializing the driver, the function Adc_ReadGroup shall raise development error ADC_E_UNINIT and return E_NOT_OK.⌋*()*

8.3.7 Adc_EnableHardwareTrigger

[SWS_Adc_91001] ⌈

Service Name	Adc_EnableHardwareTrigger	
Syntax	`void Adc_EnableHardwareTrigger ( Adc_GroupType Group )`	
Service ID [hex]	0x05	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Enables the hardware trigger for the requested ADC Channel group.	
Available via	Adc.h	

⌋*()*

[SWS_Adc_00114] ⌈The function Adc_EnableHardwareTrigger shall enable the hardware trigger for the requested ADC Channel group.⌋*(SRS_Adc_12823)*

Note: Adc_EnableHardwareTrigger can only be used for ADC internal trigger sources controlled from the ADC hardware.

[SWS_Adc_00144] ⌈A group with trigger source hardware, whose trigger was enabled with Adc_EnableHardwareTrigger, shall execute the group channel conversions, whenever a trigger event occurs.⌋*(SRS_Adc_12823)*

[SWS_Adc_00432] ⌈The function Adc_EnableHardwareTrigger shall reset the internal group result buffer pointer, that conversion result storage always starts, after calling Adc_EnableHardwareTrigger, at the result buffer base address which was configured with Adc_SetupResultBuffer.⌋*()*

[SWS_Adc_00273] ⌈The ADC module's environment shall guarantee that no concurrent conversions take place on the same HW Unit (happening of different hardware triggers at the same time).⌋*(SRS_Adc_12823)*

Note: The reason for SWS_Adc_00273 is that the ADC module can only handle one group conversion request per HW Unit at the same time. In case of concurrent HW conversion requests, the HW prioritization mechanism controls the conversion order.

[SWS_Adc_00120] ⌈The ADC module's environment shall only call the function Adc_EnableHardwareTrigger for groups configured in hardware trigger mode (see AdcGroupTriggSrc).⌋*(SRS_BSW_00171)*

[SWS_Adc_00265] ⌈The function Adc_EnableHardwareTrigger shall be pre-compile time configurable On/Off by the configuration parameter AdcHwTriggerApi.⌋*(SRS_BSW_00171)*

[SWS_Adc_00321] ⌈If the priority mechanism is disabled and queuing disabled: when called while any group with trigger source SW is not in state ADC_IDLE, the function Adc_EnableHardwareTrigger shall report a runtime error ADC_E_BUSY.⌋*()*

[SWS_Adc_00349] ⌈If the HW trigger for the group is already enabled, the function Adc_EnableHardwareTrigger shall report a runtime error ADC_E_BUSY.⌋*()*

[SWS_Adc_00353] ⌈If the maximum number of available hardware triggers is already enabled (device and implementation specific), the function Adc_EnableHardwareTrigger shall report a runtime error ADC_E_BUSY.⌋*()*

[SWS_Adc_00128] ⌈If development error detection for the ADC module is enabled: if the channel group ID is invalid, the function Adc_EnableHardwareTrigger shall raise development error ADC_E_PARAM_GROUP and return without any action.⌋*(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00136] ⌈If development error detection for the ADC module is enabled: if the group is configured for software API trigger mode, the function Adc_EnableHardwareTrigger shall raise development error ADC_E_WRONG_TRIGG_SRC and return without any action.⌋*(SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00281] ⌈If development error detection for the ADC module is enabled: if a HW group is erroneously configured for continuous conversion mode, the function Adc_EnableHardwareTrigger shall raise development error ADC_E_WRONG_CONV_MODE and return without any action.⌋*(SRS_Adc_12823)*

Note: SW groups configured in continuous conversion mode shall raise development error ADC_E_WRONG_TRIGG_SRC instead.

[SWS_Adc_00297] ⌈If development error detection for the ADC module is enabled: if called prior to initializing the driver, the function Adc_EnableHardwareTrigger shall

raise development error ADC_E_UNINIT and return without any action.⌋*(SRS_BSW_00406)*

[SWS_Adc_00425] ⌈If development error detection for the ADC module is enabled: when called prior to initializing the result buffer pointer with function Adc_SetupResult Buffer, the function Adc_EnableHardwareTrigger shall raise development error ADC_E_BUFFER_UNINIT.⌋*()*

8.3.8 Adc_DisableHardwareTrigger

[SWS_Adc_91002] ⌈

Service Name	Adc_DisableHardwareTrigger	
Syntax	`void Adc_DisableHardwareTrigger ( Adc_GroupType Group )`	
Service ID [hex]	0x06	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Disables the hardware trigger for the requested ADC Channel group.	
Available via	Adc.h	

⌋*()*

[SWS_Adc_00116] ⌈The function Adc_DisableHardwareTrigger shall disable the hardware trigger for the requested ADC Channel group.⌋*(SRS_Adc_12823)*

[SWS_Adc_00429] ⌈The function Adc_DisableHardwareTrigger shall remove any queued start/restart request for the requested ADC Channel group if queuing is enabled.⌋*()*

[SWS_Adc_00145] ⌈The function Adc_DisableHardwareTrigger shall abort an ongoing conversion, if applicable (supported by the hardware).⌋*(SRS_Adc_12364)*

[SWS_Adc_00157] ⌈If enabled, the function Adc_DisableHardwareTrigger shall disable the notification mechanism for the requested group.⌋*(SRS_Adc_12317, SRS_Adc_12318, SRS_Adc_12364)*

[SWS_Adc_00361] ⌈The function Adc_DisableHardwareTrigger shall set the group status to state ADC_IDLE.⌋*()*

[SWS_Adc_00121] ⌈The ADC module's environment shall only call the function Adc_DisableHardwareTrigger for groups configured in hardware trigger mode (see AdcGroupTriggSrc).⌋*(SRS_BSW_00171)*

[SWS_Adc_00266] ⌈The function Adc_DisableHardwareTrigger shall be pre-compile time configurable On/Off by the configuration parameter AdcHwTriggerApi.⌋*(SRS_BSW_00171)*

[SWS_Adc_00129] ⌈If development error detection for the ADC module is enabled: if the channel group ID is non-existing, the function Adc_DisableHardwareTrigger shall raise development error ADC_E_PARAM_GROUP and return without any action.⌋*(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00137] ⌈If development error detection for the ADC module is enabled: if the group is configured for software API trigger mode, the function Adc_DisableHardwareTrigger shall raise development error ADC_E_WRONG_TRIGG_SRC and return without any action.⌋*(SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00282] ⌈If development error detection for the ADC module is enabled:if a HW group is erroneously configured for continuous conversion mode, the function Adc_DisableHardwareTrigger shall raise development error ADC_E_WRONG_CONV_MODE and return without any action.⌋*(SRS_Adc_12823)*

Note: SW groups configured in continuous conversion mode shall raise development error ADC_E_WRONG_TRIGG_SRC instead.

[SWS_Adc_00304] ⌈If the group is not enabled (with a previous call of Adc_EnableHardwareTrigger), the function Adc_DisableHardwareTrigger shall report a runtime error ADC_E_IDLE.⌋*()*

[SWS_Adc_00298] ⌈If development error detection for the ADC module is enabled: if called prior to initializing the ADC module, Adc_DisableHardwareTrigger shall raise development error ADC_E_UNINIT and return without any action.⌋*(SRS_BSW_00406)*

Note:

All groups which are enabled with Adc_EnableHardwareTrigger should also be disabled with Adc_DisableHardwareTrigger, before they are enabled again, even if they are implicitly stopped because of the selected conversion mode (streaming buffer with linear access mode).

8.3.9 Adc_EnableGroupNotification

[SWS_Adc_91003] ⌈

Service Name	Adc_EnableGroupNotification
Syntax	`void Adc_EnableGroupNotification ( Adc_GroupType Group )`
Service ID [hex]	0x07
Sync/Async	Asynchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Enables the notification mechanism for the requested ADC Channel group.	
Available via	Adc.h	

⌋()

[SWS_Adc_00057] ⌈The function Adc_EnableGroupNotification shall enable the notification mechanism for the requested ADC Channel group.⌋(*SRS_SPAL_00157*, *SRS_Adc_12318*)

[SWS_Adc_00100] ⌈The function Adc_EnableGroupNotification shall be pre-compile time configurable On/Off by the configuration parameter AdcGrpNotifCapability.⌋(*SRS_Adc_12447*)

[SWS_Adc_00130] ⌈If development error detection for the ADC module is enabled: if the channel group ID is non-existing, the function Adc_EnableGroupNotification shall raise development error ADC_E_PARAM_GROUP and return without any action.⌋()

(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448,)

[SWS_Adc_00165] ⌈If development error detection for the ADC module is enabled: if the group notification function pointer is NULL, the function Adc_EnableGroupNotification shall raise development error ADC_E_NOTIF_CAPABILITY and return without any action.⌋(*SRS_BSW_00386*, *SRS_SPAL_12448*)

[SWS_Adc_00299] ⌈If development error detection for the ADC module is enabled: if called prior to initializing the ADC module, Adc_EnableGroupNotification shall raise development error ADC_E_UNINIT and return without any action.⌋(*SRS_BSW_00406*)

8.3.10 Adc_DisableGroupNotification

[SWS_Adc_91004] ⌈

Service Name	Adc_DisableGroupNotification	
Syntax	`void Adc_DisableGroupNotification ( Adc_GroupType Group )`	
Service ID [hex]	0x08	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	Disables the notification mechanism for the requested ADC Channel group.
Available via	Adc.h

⌋()

[SWS_Adc_00058] ⌈The function Adc_DisableGroupNotification shall disable the notification mechanism for the requested ADC Channel group.⌋*(SRS_SPAL_00157, SRS_Adc_12318)*

[SWS_Adc_00101] ⌈The function Adc_DisableGroupNotification shall be pre-compile time configurable On/Off by the configuration parameter AdcGrpNotifCapability⌋ *(SRS_Adc_12447)*

[SWS_Adc_00131] ⌈If development error detection for the ADC module is enabled: if the channel group ID is non-existing, the function Adc_DisableGroupNotification shall raise development error ADC_E_PARAM_GROUP and return without any action.⌋*(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00166] ⌈If development error detection for the ADC module is enabled: if the group notification function pointer is NULL, the function Adc_DisableGroupNotification shall raise development error ADC_E_NOTIF_CAPABILITY and return without any action.⌋*(SRS_BSW_00386, SRS_SPAL_12448)*

[SWS_Adc_00300] ⌈If development error detection for the ADC module is enabled: if called prior to initializing the ADC module, Adc_DisableGroupNotification shall raise development error ADC_E_UNINIT and return without any action.⌋*(SRS_BSW_00406)*

8.3.11 Adc_GetGroupStatus

[SWS_Adc_00374] ⌈

Service Name	Adc_GetGroupStatus	
Syntax	`Adc_StatusType Adc_GetGroupStatus ( Adc_GroupType Group )`	
Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Adc_StatusType	Conversion status for the requested group.
Description	Returns the conversion status of the requested ADC Channel group.	
Available via	Adc.h	

⌋()

[SWS_Adc_00220] ⌈The function Adc_GetGroupStatus shall return the conversion status of the requested ADC Channel group.⌋*(SRS_Adc_12291)*

[SWS_Adc_00221] ⌈The function Adc_GetGroupStatus shall return ADC_IDLE:

- If Adc_GetGroupStatus is called before the conversion of the requested group has been started
- For groups with trigger source software: If Adc_GetGroupStatus is called after the conversion was stopped with Adc_StopGroupConversion
- In continuous group conversion mode with linear streaming access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer (group was in state ADC_STREAM_COMPLETED while calling Adc_GetStreamLastPointer).
- In continuous group conversion mode with linear streaming access mode: If Adc_GetGroupStatus is called after calling Adc_ReadGroup (group was in state ADC_STREAM_COMPLETED while calling Adc_ReadGroup).
- In one-shot SW conversion mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer.
- In one-shot SW conversion mode: If Adc_GetGroupStatus is called after calling Adc_ReadGroup.
- For groups with trigger source hardware: If Adc_GetGroupStatus is called after calling Adc_DisableHardwareTrigger
- For groups with trigger source hardware and linear streaming access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer (group was in state ADC_STREAM_COMPLETED while calling Adc_GetStreamLastPointer).
- For groups with trigger source hardware and linear streaming access mode: If Adc_GetGroupStatus is called after calling Adc_ReadGroup (group was in state ADC_STREAM_COMPLETED while calling Adc_ReadGroup).

⌋*(SRS_BSW_00335, SRS_Adc_12291)*

[SWS_Adc_00222] ⌈The function Adc_GetGroupStatus shall return ADC_BUSY:

- If it is called while the first conversion round of the requested group is still ongoing (continuous conversion mode).
- Once trigger is enabled for group with HW trigger source.
- Once Adc_StartGroupConversion is called for group with SW trigger source.
- In continuous group conversion mode with single access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer
- In continuous group conversion mode with single access mode: If Adc_GetGroupStatus is called after calling Adc_ReadGroup.
- In continuous group conversion mode with circular streaming access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer

- In continuous group conversion mode with circular streaming access mode If Adc_GetGroupStatus is called after calling Adc_ReadGroup.
- In continuous group conversion mode with linear streaming access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer (group was in state ADC_COMPLETED while calling Adc_GetStreamLastPointer).
- In continuous group conversion mode with linear streaming access mode: If Adc_GetGroupStatus is called after calling Adc_ReadGroup (group was in state ADC_COMPLETED while calling Adc_ReadGroup).
- In one-shot HW conversion mode and single access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer.
- In one-shot HW conversion mode and single access mode: If Adc_GetGroupStatus is called after calling Adc_ReadGroup.
- In one-shot HW conversion mode and circular streaming access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer.
- In one-shot HW conversion mode and circular streaming access mode:

If Adc_GetGroupStatus is called after calling Adc_ReadGroup.

- In one-shot HW conversion mode and linear streaming access mode: If Adc_GetGroupStatus is called after calling Adc_GetStreamLastPointer (group was in state ADC_COMPLETED while calling Adc_GetStreamLastPointer).
- In one-shot HW conversion mode and linear streaming access mode:

If Adc_GetGroupStatus is called after calling Adc_ReadGroup

(group was in state ADC_COMPLETED while calling Adc_ReadGroup).⌋*(SRS_BSW_00335, SRS_Adc_12291)*

[SWS_Adc_00224] ⌈The function Adc_GetGroupStatus shall return ADC_COMPLETED:

- If it is called after a conversion round (not the final one) of the requested group has been finished.

⌋*(SRS_BSW_00335, SRS_Adc_12291)*

[SWS_Adc_00325] ⌈The function Adc_GetGroupStatus shall return ADC_STREAM_COMPLETED:

- If it is called in single access mode after one conversion round is completed.
- If it is called in streaming access mode after the number of conversion rounds of the requested group have been finished, to fill the streaming buffer completely.

⌋*(SRS_Adc_12291)*

[SWS_Adc_00226] ⌈The function Adc_GetGroupStatus shall provide atomic access to the status data by the use of atomic instructions.⌋*(SRS_Adc_12291)*

[SWS_Adc_00305] ⌈To guarantee consistent returned values, it is assumed that ADC group conversion is always started (or enabled in case of HW group) successfully by SW before status polling begins.⌋*()*

[SWS_Adc_00225] ⌈If development error detection for the ADC module is enabled: if the channel group ID is non-existing, the function Adc_GetGroupStatus shall raise development error ADC_E_PARAM_GROUP and return ADC_IDLE without any action.⌋ *(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)*

.

[SWS_Adc_00301] ⌈If development error detection for the ADC module is enabled: if called prior to initializing the ADC module, Adc_GetGroupStatus shall raise development error ADC_E_UNINIT and return ADC_IDLE without any action.⌋*(SRS_BSW_00406)*

[SWS_Adc_00436] ⌈In case of an aborted/suspended group, the state of the queued group remains the same as it was before the group was aborted/suspended.⌋*()*

8.3.12 Adc_GetStreamLastPointer

[SWS_Adc_00375] ⌈

Service Name	Adc_GetStreamLastPointer	
Syntax	`Adc_StreamNumSampleType Adc_GetStreamLastPointer ( Adc_GroupType Group, Adc_ValueGroupType** PtrToSamplePtr )`	
Service ID [hex]	0x0b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Group	Numeric ID of requested ADC Channel group.
Parameters (inout)	None	
Parameters (out)	PtrToSamplePtr	Pointer to result buffer pointer.
Return value	Adc_StreamNumSampleType	Number of valid samples per channel.
Description	Returns the number of valid samples per channel, stored in the result buffer. Reads a pointer, pointing to a position in the group result buffer. With the pointer position, the results of all group channels of the last completed conversion round can be accessed. With the pointer and the return value, all valid group conversion results can be accessed (the user has to take the layout of the result buffer into account).	
Available via	Adc.h	

⌋*()*

[SWS_Adc_00214] ⌈The function Adc_GetStreamLastPointer shall set the pointer, passed as parameter (PtrToSamplePtr) to point in the ADC result buffer to the latest result of the first group channel of the last completed conversion round.⌋*(SRS_Adc_12292, SRS_Adc_12802)*

[SWS_Adc_00418] ⌈All values which the ADC driver stores in the ADC result buffer, are left without further scaling and shall be aligned according the configuration parameter setting of ADC_RESULT_ALIGNMENT.⌋*()*

[SWS_Adc_00387] ⌈The function Adc_GetStreamLastPointer shall return the number of valid samples per channel, stored in the ADC result buffer.⌋*()*

Note: Valid samples are in the ADC result buffer when the group is in state ADC_COMPLETED or ADC_STREAM_COMPLETED. In state ADC_BUSY or ADC_IDLE the value 0 is returned.

Note: The return value is 1 for groups with single access mode configuration, if valid samples are stored in the ADC result buffer.

[SWS_Adc_00216] ⌈When called while the group status is ADC_BUSY (a conversion of the group is in progress), the function Adc_GetStreamLastPointer shall set the pointer, passed as parameter (PtrToSamplePtr), to NULL and return 0.⌋*(SRS_Adc_12802)*

[SWS_Adc_00219] ⌈The ADC module's environment shall guarantee the consistency of the data that has been read by checking the return value of Adc_GetGroupStatus.⌋*(SRS_Adc_12291, SRS_Adc_12802)*

Note: See also SWS_Adc_00140.

[SWS_Adc_00326] ⌈Calling function Adc_GetStreamLastPointer while group status is ADC_STREAM_COMPLETED shall trigger a state transition to ADC_BUSY for continuous conversion modes (single access mode or circular streaming buffer mode) and hardware triggered groups in single access mode or circular streaming access mode.⌋*(SRS_Adc_12291)*

[SWS_Adc_00327] ⌈Calling function Adc_GetStreamLastPointer while group status is ADC_STREAM_COMPLETED shall trigger a state transition to ADC_IDLE for software conversion modes which automatically stop the conversion (streaming buffer with linear access mode or one-shot conversion mode with single access) and for the hardware triggered conversion mode in combination with linear streaming access mode.⌋*(SRS_Adc_12291)*

[SWS_Adc_00328] ⌈Calling function Adc_GetStreamLastPointer while group status is ADC_COMPLETED shall trigger a state transition to ADC_BUSY.⌋*(SRS_Adc_12291)*

[SWS_Adc_00215] ⌈When called while the group status is ADC_IDLE and the group conversion was not started (no results are available from previous conversions) , the function Adc_GetStreamLastPointer shall report a runtime error ADC_E_IDLE.⌋*()*

[SWS_Adc_00218] ⌈If development error detection for the ADC moduleis enabled: if the group ID is non-existent, the function Adc_GetStreamLastPointer shall raise development error ADC_E_PARAM_GROUP, set the pointer, passed as parameter (PtrTo SamplePtr), to NULL and return 0 without any further action.⌋*(SRS_BSW_00386)*

[SWS_Adc_00302] ⌈If development error detection for the ADC moduleis enabled:if called prior to initializing the driver, the function Adc_GetStreamLastPointer shall raise

development error ADC_E_UNINIT, set the pointer, passed as parameter (PtrToSamplePtr), to NULL and return 0 without any further action.⌋(*SRS_BSW_00406*)

8.3.13 Adc_GetVersionInfo

[SWS_Adc_00376] ⌈

Service Name	Adc_GetVersionInfo	
Syntax	`void Adc_GetVersionInfo ( Std_VersionInfoType* versioninfo )`	
Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versioninfo	Pointer to where to store the version information of this module.
Return value	None	
Description	Returns the version information of this module.	
Available via	Adc.h	

⌋()

[SWS_Adc_00458] ⌈If development error detection for the ADC module is enabled: The function Adc_GetVersionInfo shall check the parameter versioninfo for not being NULL and shall raise the development error ADC_E_PARAM_POINTER if the check fails.⌋()

8.3.14 Adc_SetPowerState

[SWS_Adc_00475] ⌈

Service Name	Adc_SetPowerState
Syntax	`Std_ReturnType Adc_SetPowerState ( Adc_PowerStateRequestResultType* Result )`
Service ID [hex]	0x10
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None

▽

△

Parameters (out)	Result	If the API returns E_OK: ADC_SERVICE_ACCEPTED: Power state change executed. If the API returns E_NOT_OK: ADC_NOT_INIT: ADC Module not initialized. ADC_SEQUENCE_ERROR: wrong API call sequence. ADC_HW_FAILURE: the HW module has a failure which prevents it to enter the required power state.
Return value	Std_ReturnType	E_OK: Power Mode changed E_NOT_OK: request rejected
Description	This API configures the Adc module so that it enters the already prepared power state, chosen between a predefined set of configured ones.	
Available via	Adc.h	

⌋()

[SWS_Adc_00481] ⌈The API configures the HW in order to enter the previously prepared Power State. All preliminary actions to enable this transition (e.g. setting all channels in IDLE status, de-registering of all notifications and so on) must already have been taken by the responsible SWCs (e.g. IoHwAbs).

The API shall not execute preliminary, implicit power state changes (i.e. if a requested power state is not reachable starting from the current one, no intermediate power state change shall be executed and the request shall be rejected)⌋()

[SWS_Adc_00482] ⌈In case the target power state is the same as the current one, no action is executed and the API returns immediately with an E_OK result.⌋()

[SWS_Adc_00483] ⌈In case the normal Power State is requested, the API shall refer to the necessary parameters contained in the same containers used by Adc_Init.⌋()

No separate container or hard coded data shall be used for the normal (i.e. full) power mode, in order to avoid misalignments between initialization parameters used during the init phase and during a power state change.

[SWS_Adc_00484] ⌈For the other power states, only power state transition specific reconfigurations shall be executed in the context of this API (i.e. the API cannot be used to apply a completely new configuration to the Adc module). Any other re-configuration not strictly related to the power state transition shall not take place.⌋()

[SWS_Adc_00485] ⌈The API shall refer to the configuration container related to the required Power State in order to derive some specific features of the state (e.g support of Power States).⌋()

[SWS_Adc_00486] ⌈In case development error reporting is activated:

The API shall report the DET error ADC_E_UNINIT in case this API is called before having initialized the HW unit.⌋()

[SWS_Adc_00487] ⌈The API shall report a runtime error ADC_E_NOT_DISENGAGED in case this API is called when one or more HW channels (where applicable) are in a state different then IDLE (or similar non-operational states) and/or there are still notification registered for the HW module channels.⌋()

[SWS_Adc_00488] ⌈In case development error reporting is activated:

The API shall report the DET error ADC_E_POWER_STATE_NOT_SUPPORTED in case this API is called with an unsupported power state or the peripheral does not support low power states at all.⌋()

[SWS_Adc_00489] ⌈The API shall report a runtime error ADC_E_TRANSITION_NOT_POSSIBLE in case the requested power state cannot be directly reached from the current power state.⌋()

[SWS_Adc_00490] ⌈In case development error reporting is activated:

The API shall report the DET error ADC_E_PERIPHERAL_NOT_PREPARED in case the HW unit has not been previously prepared for the target power state by use of the API Adc_PreparePowerState().⌋()

8.3.15 Adc_GetCurrentPowerState

[SWS_Adc_00476] ⌈

Service Name	Adc_GetCurrentPowerState	
Syntax	`Std_ReturnType Adc_GetCurrentPowerState ( Adc_PowerStateType* CurrentPowerState, Adc_PowerStateRequestResultType* Result )`	
Service ID [hex]	0x11	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	CurrentPowerState	The current power mode of the ADC HW Unit is returned in this parameter
	Result	If the API returns E_OK: ADC_SERVICE_ACCEPTED: Current power mode was returned. If the API returns E_NOT_OK: ADC_NOT_INIT: ADC Module not initialized.
Return value	Std_ReturnType	E_OK: Mode could be read E_NOT_OK: Service is rejected
Description	This API returns the current power state of the ADC HW unit.	
Available via	Adc.h	

⌋()

[SWS_Adc_00491] ⌈In case development error reporting is activated:

The API shall report the DET error ADC_E_UNINIT in case this API is called before having initialized the HW unit.⌋()

8.3.16 Adc_GetTargetPowerState

[SWS_Adc_00477] ⌈

Service Name	Adc_GetTargetPowerState	
Syntax	`Std_ReturnType Adc_GetTargetPowerState ( Adc_PowerStateType* TargetPowerState, Adc_PowerStateRequestResultType* Result )`	
Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	TargetPowerState	The Target power mode of the ADC HW Unit is returned in this parameter
	Result	If the API returns E_OK: ADC_SERVICE_ACCEPTED:Target power mode was returned. If the API returns E_NOT_OK: ADC_NOT_INIT: ADC Module not initialized.
Return value	Std_ReturnType	E_OK: Mode could be read E_NOT_OK: Service is rejected
Description	This API returns the Target power state of the ADC HW unit.	
Available via	Adc.h	

⌋()

[SWS_Adc_00492] ⌈The API returns the requested power state of the HW unit. This shall coincide with the current power state if no transition is ongoing.

The API is considered to always succeed except in case of HW failures.⌋()

[SWS_Adc_00493] ⌈In case development error reporting is activated:

The API shall report the DET error ADC_E_UNINIT in case this API is called before having initialized the HW unit.⌋()

8.3.17 Adc_PreparePowerState

[SWS_Adc_00478] ⌈

Service Name	Adc_PreparePowerState	
Syntax	`Std_ReturnType Adc_PreparePowerState ( Adc_PowerStateType PowerState, Adc_PowerStateRequestResultType* Result )`	
Service ID [hex]	0x13	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	PowerState	The target power state intended to be attained

▽

△

Parameters (inout)	None	
Parameters (out)	Result	If the API returns E_OK: ADC_SERVICE_ACCEPTED: ADC Module power state preparation was started. If the API returns E_NOT_OK: ADC_NOT_INIT: ADC Module not initialized. ADC_SEQUENCE_ERROR: wrong API call sequence (Current Power State = Target Power State). ADC_POWER_STATE_NOT_SUPP: ADC Module does not support the requested power state. ADC_TRANS_NOT_POSSIBLE: ADC Module cannot transition directly from the current power state to the requested power state or the HW peripheral is still busy.
Return value	Std_ReturnType	E_OK: Preparation process started E_NOT_OK: Service is rejected
Description	This API starts the needed process to allow the ADC HW module to enter the requested power state.	
Available via	Adc.h	

⌋()

[SWS_Adc_00494] ⌈This API initiates all actions needed to enable a HW module to enter the target power state.

The possibility to operate the periphery depends on the power state and the HW features. These properties should be known to the integrator and the decision whether to use the periphery or not is in his responsibility.⌋()

[SWS_Adc_00495] ⌈In case the target power state is the same as the current one, no action is executed and the API returns immediately with an E_OK result.

The responsibility of the preconditions is left to the environment.⌋()

[SWS_Adc_00496] ⌈In case development error reporting is activated:

The API shall report the DET error ADC_E_UNINIT in case this API is called before having initialized the HW unit.⌋()

[SWS_Adc_00497] ⌈In case development error reporting is activated:

The API shall report the DET error ADC_E_POWER_STATE_NOT_SUPPORTED in case this API is called with an unsupported power state is requested or the peripheral does not support low power states at all.⌋()

[SWS_Adc_00498] ⌈The API shall report a runtime error ADC_E_TRANSITION_NOT_POSSIBLE in case the requested power state cannot be directly reached from the current power state.

All asynchronous operation, needed to reach the target power state, can be executed in background in the context of Adc_Main_PowerTransitionManager.⌋()

8.4 Callback notifications

Since the ADC Driver is a module on the lowest architectural layer it doesn't provide any call-back functions for lower layer modules.

8.5 Scheduled functions

8.5.1 Adc_Main_PowerTransitionManager

[SWS_Adc_00479] ⌈

Service Name	Adc_Main_PowerTransitionManager
Syntax	`void Adc_Main_PowerTransitionManager ( void )`
Service ID [hex]	0x14
Description	This API is cyclically called and supervises the power state transitions, checking for the readiness of the module and issuing the callbacks IoHwAb_Adc_NotifyReadyForPower State<Mode> (see AdcPowerStateReadyCbkRef configuration parameter).
Available via	SchM_Adc.h

⌋*()*

[SWS_Adc_00499] ⌈This API executes any non-immediate action needed to finalize a power state transition requested by Adc_PreparePowerState().⌋*()*

[SWS_Adc_00500] ⌈The rate of scheduling shall be defined by Adc MainSchedule Period and shall be variable, as the function only needs to be called if a transition has been requested⌋*()*

[SWS_Adc_00501] ⌈This API shall also issue callback notifications to the eventually registered users (IoHwAbs) as configured, only in case the asynch mode is chosen.⌋*()*

[SWS_Adc_00502] ⌈In case the ADC module is not initialized, this function shall simply return without any further elaboration. This is needed to avoid to elaborate uninitialized variables. No DET error shall be entered, because this condition can easily be verified during the startup phase (tasks started before the initialization is complete).

Rationale: during the startup phase it can happen that the OS already schedules tasks, which call main functions, while some modules are not initialised yet. This is no real error condition, although need handling, i.e. returning without execution.

Although the transition state monitoring functionality is mandatory, the implementation of this API is optional, meaning that if the HW allows for other ways to deliver notification and watch the transition state the implementation of this function can be skipped.⌋*()*

8.6 Expected interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill a core functionality of the module.

[SWS_Adc_00530] ⌈

API Function	Header File	Description
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋()

8.6.2 Optional Interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of the module.

[SWS_Adc_00377] ⌈

API Function	Header File	Description
Det_ReportError	Det.h	Service to report development errors.

⌋()

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The names of this kind of interfaces are not fixed because they are configurable.

[SWS_Adc_00078] ⌈The ADC module's ISR's, providing the "conversion completed events", shall be responsible for resetting the interrupt flags (if needed by hardware) and calling the associated notification function.⌋(*SRS_SPAL_12129*)

Note: The notification functions IoHwAb_Adc_Notification_<GroupID>run in interrupt context. It's the responsibility of the user to keep the code of these functions reasonably short. The names of the group notification functions are configurable (see ADC402).

8.6.3.1 IoHwAb_Adc_Notification<#groupID>

[SWS_Adc_00082] ⌈

Service Name	IoHwAb_AdcNotification<#groupID>
Syntax	`void IoHwAb_AdcNotification<#groupID> ( void )`
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Will be called by the ADC Driver when a group conversion is completed for group <#groupID>.
Available via	IoHwAb_Adc.h

⌋*(SRS_BSW_00359, SRS_BSW_00360, SRS_SPAL_00157)*

[SWS_Adc_00104] ⌈The ADC Driver shall support an individual notification per ADC Channel group (if capability is configured) that is called whenever the conversion for all channels of that group is completed.⌋*(SRS_SPAL_00157, SRS_Adc_12447, SRS_Adc_12317)*

[SWS_Adc_00083] ⌈When the notification mechanism is disabled, the ADC module shall send no notification.⌋*(SRS_SPAL_00157)*

[SWS_Adc_00416] ⌈When the notifications are re-enabled, the ADC module shall not send notifications for events that occurred while notifications have been disabled.⌋*()*

[SWS_Adc_00084] ⌈For every group, a particular notification call-back has to be configured. This can be a function pointer or a NULL pointer.⌋*(SRS_SPAL_12056)*

[SWS_Adc_00080] ⌈If for a notification call-back the NULL pointer is configured, no call-back shall be executed.⌋*(SRS_SPAL_12056)*

[SWS_Adc_00085] ⌈The call-back notifications shall be configurable as pointers to user defined functions within the configuration structure.For all available channel groups, call-back functions have to be declared during the configuration phase of the module.⌋*(SRS_SPAL_12056)*

8.6.3.2 IoHwAb_Adc_NotifyReadyForPowerState<#Mode>

[SWS_Adc_00480] ⌈

Service Name	IoHwAb_Adc_NotifyReadyForPowerState<#Mode>
Syntax	`void IoHwAb_Adc_NotifyReadyForPowerState<#Mode> ( void )`
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	The API shall be invoked by the ADC Driver when the requested power state preparation for mode <#Mode> is completed.
Available via	IoHwAb_Adc.h

⌋() This interface provided by CDD or IoHwAbs controlling the peripheral is needed if at least one MCAL driver is configured for providing power mode control APIs.

There shall be one such a callback for each power mode in which the ADC has to change power state. It is possible to have the same power state for different power modes, but only one power state for a given power mode.

9 Sequence diagrams

9.1 Initialization of the ADC Driver

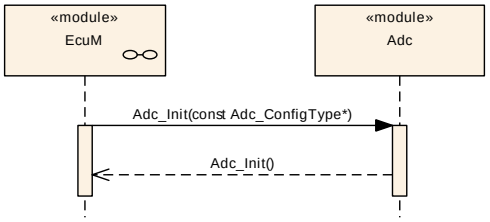


Figure 9.1: Initialization of the ADC Driver

9.2 De-Initialization of the ADC Driver

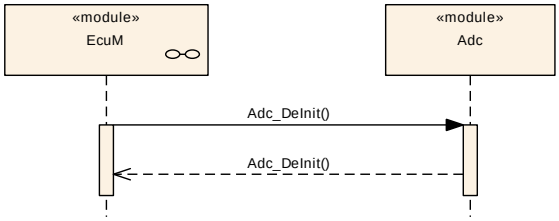


Figure 9.2: De-Initialization of the ADC Driver

9.3 Software triggered One-Shot conversion without notification

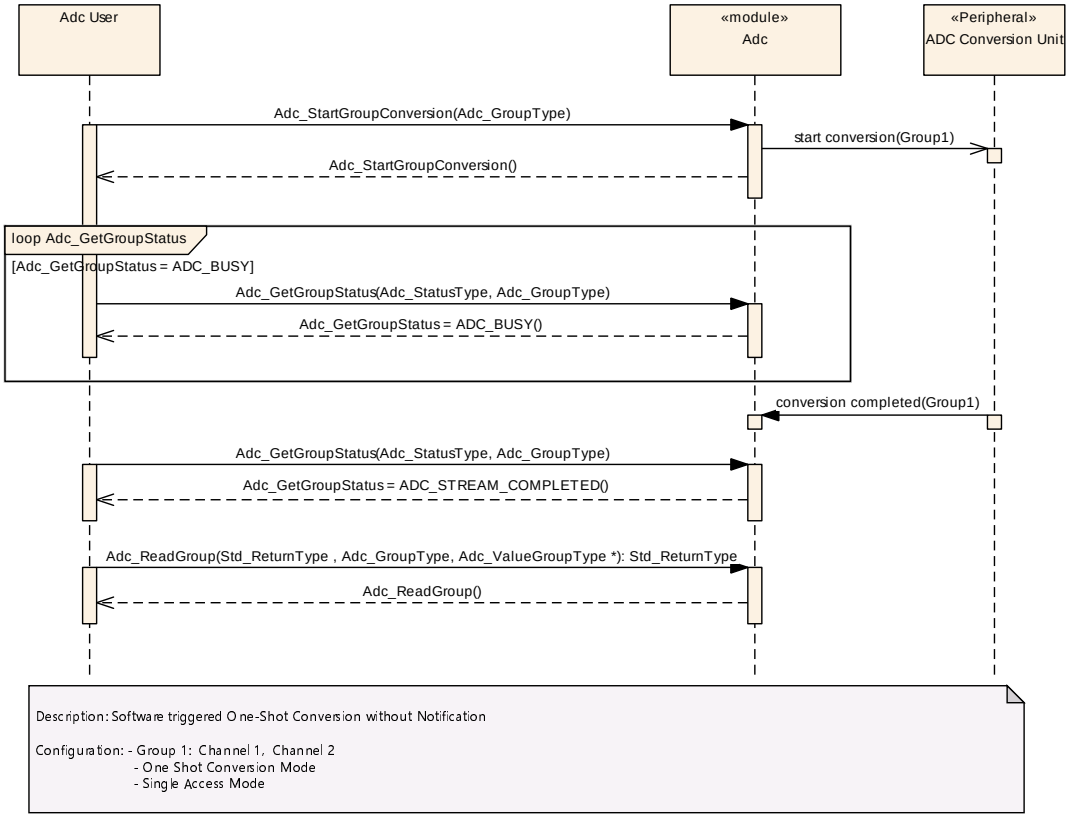


Figure 9.3: Software triggered one-shot conversion without notification

9.4 Software triggered continuous conversion with notification

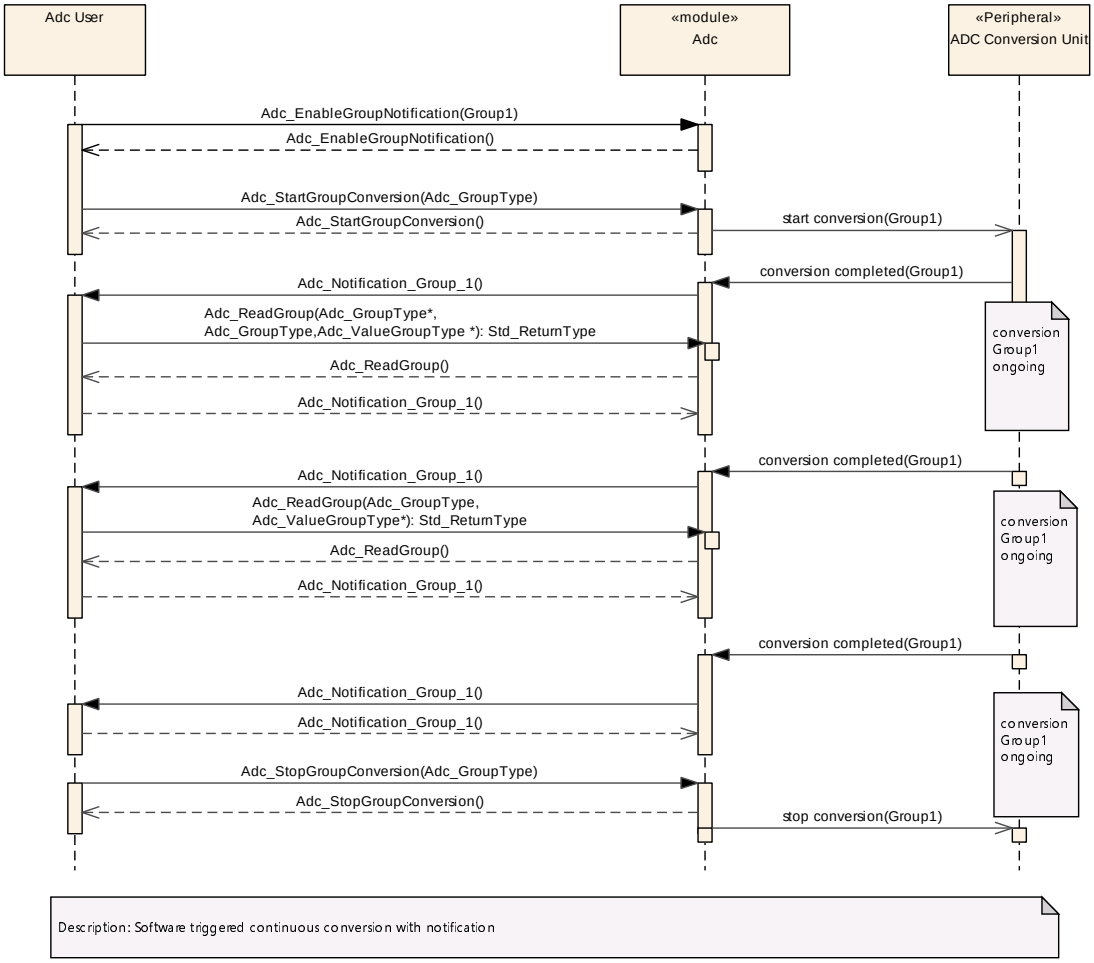


Figure 9.4: Software triggered continuous conversion with notification

9.5 Hardware triggered One-Shot conversion with notification

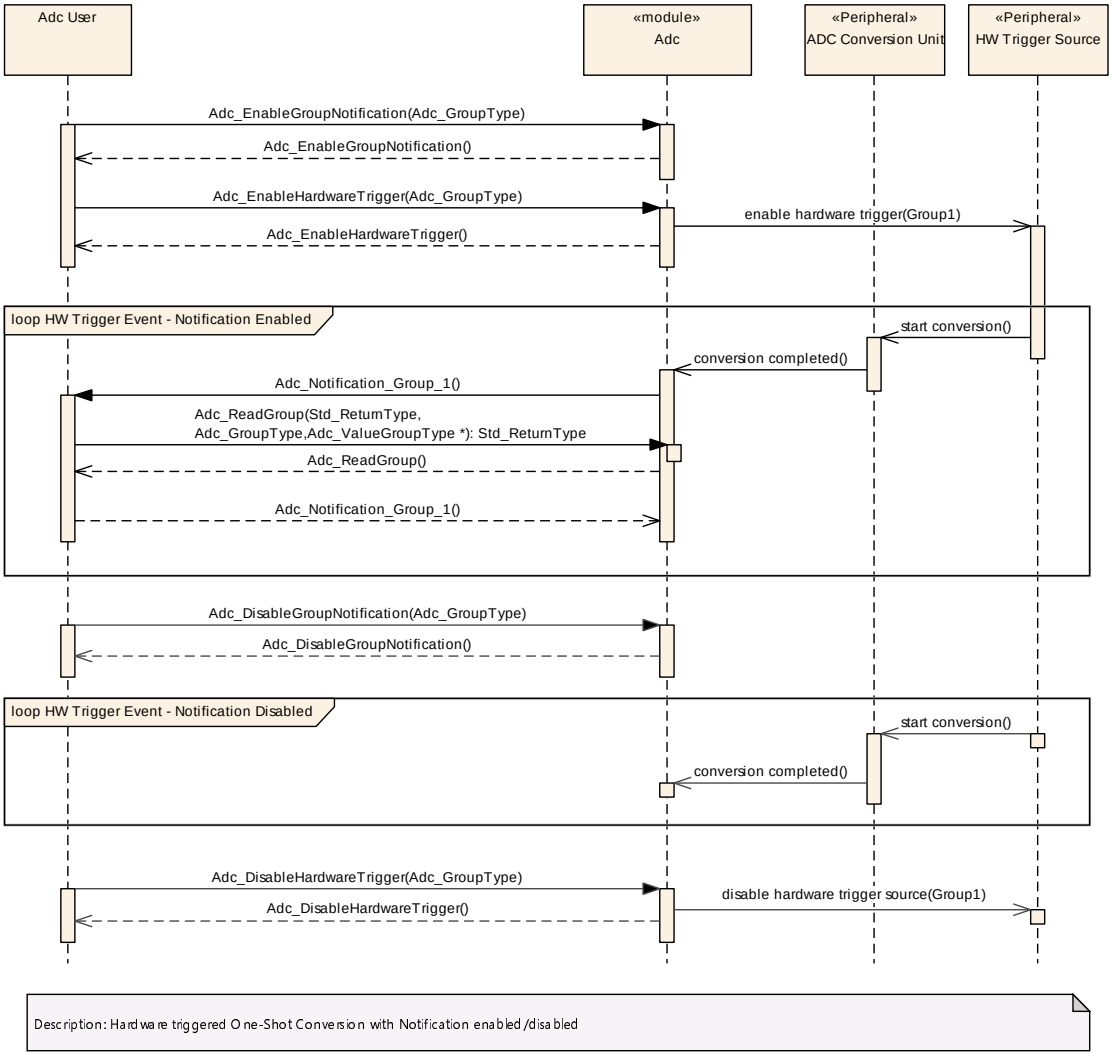


Figure 9.5: Hardware triggered one-shot conversion with notification

9.6 HW Trigger- One-Shot conversion - Linear Streaming

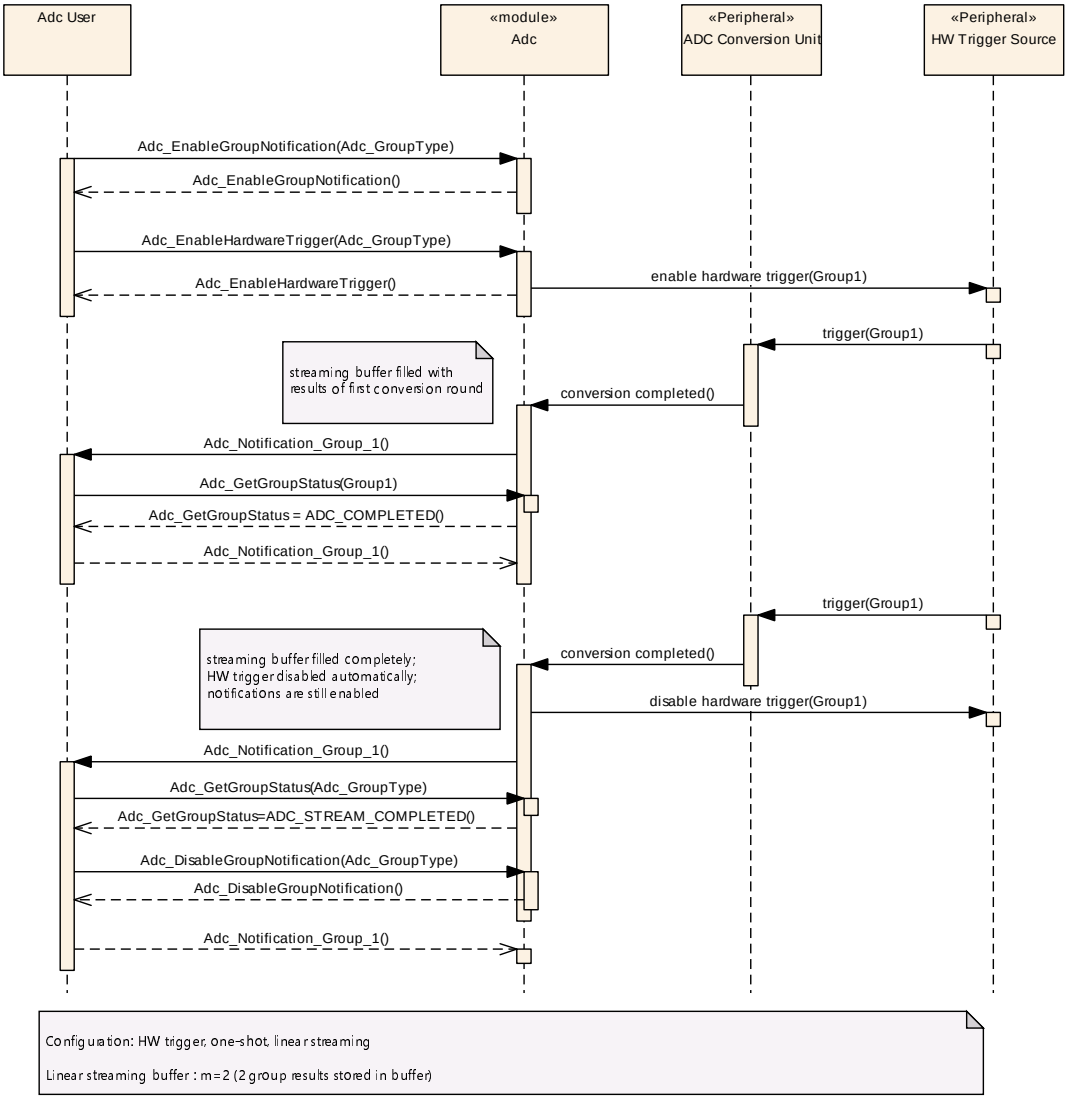


Figure 9.6: Hardware triggered one-shot conversion - linear streaming

9.7 No Priority Mechanism - No Queuing

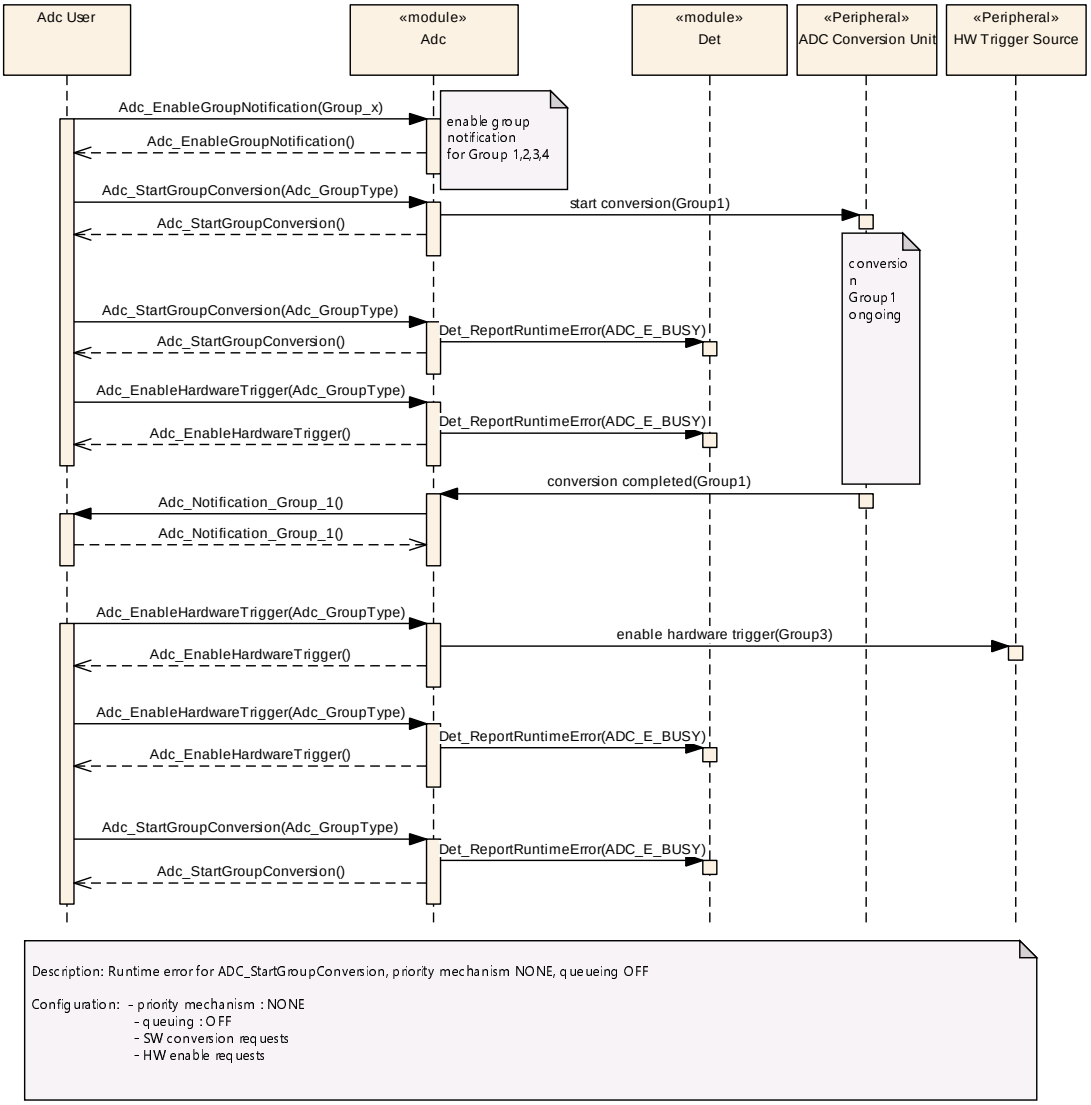


Figure 9.7: No priority mechanism - no queuing

9.8 No Priority Mechanism - SW Queuing

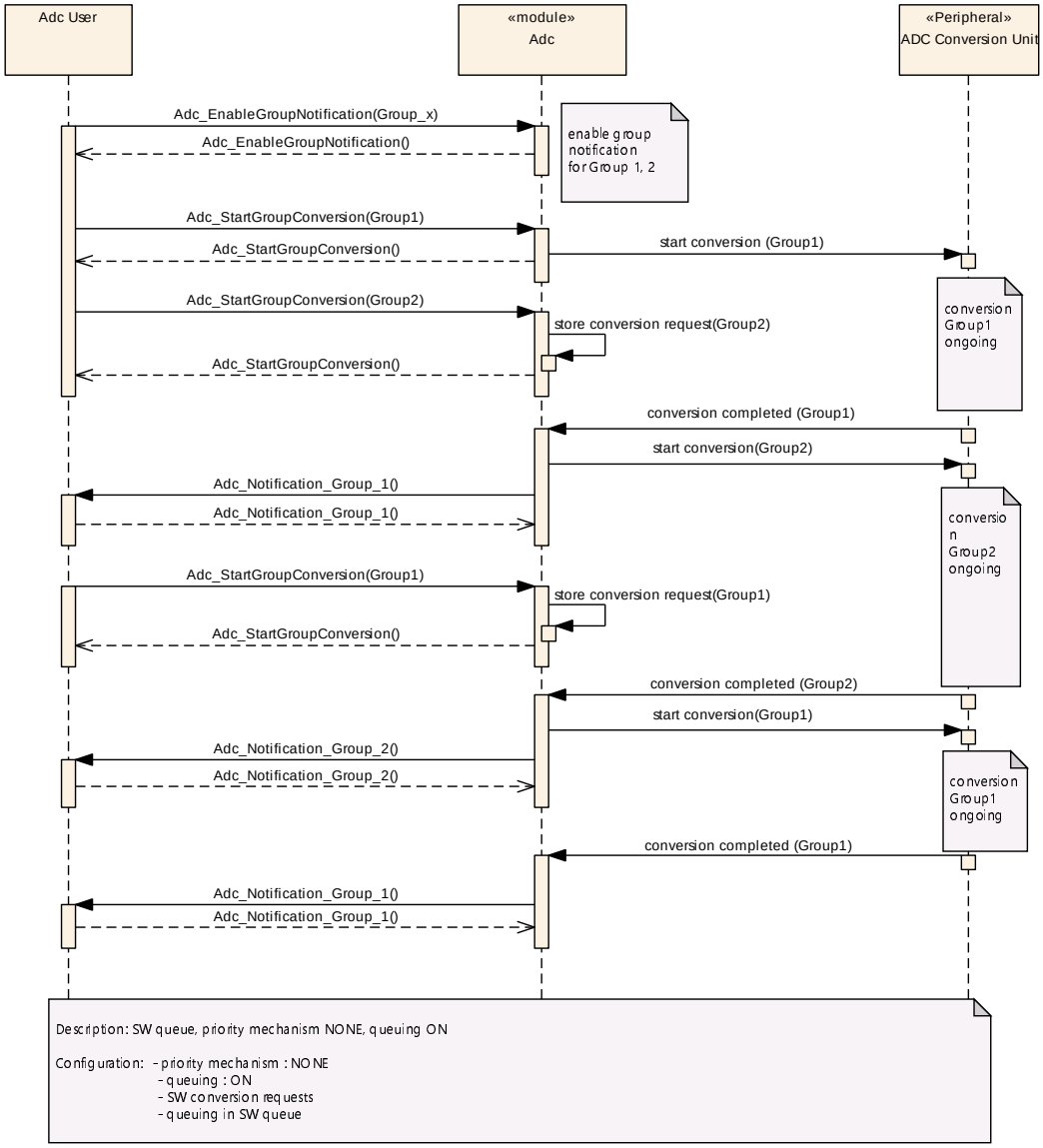


Figure 9.8: No priority mechanism – software queuing

9.9 HW_SW Priority Mechanism - SW Queuing

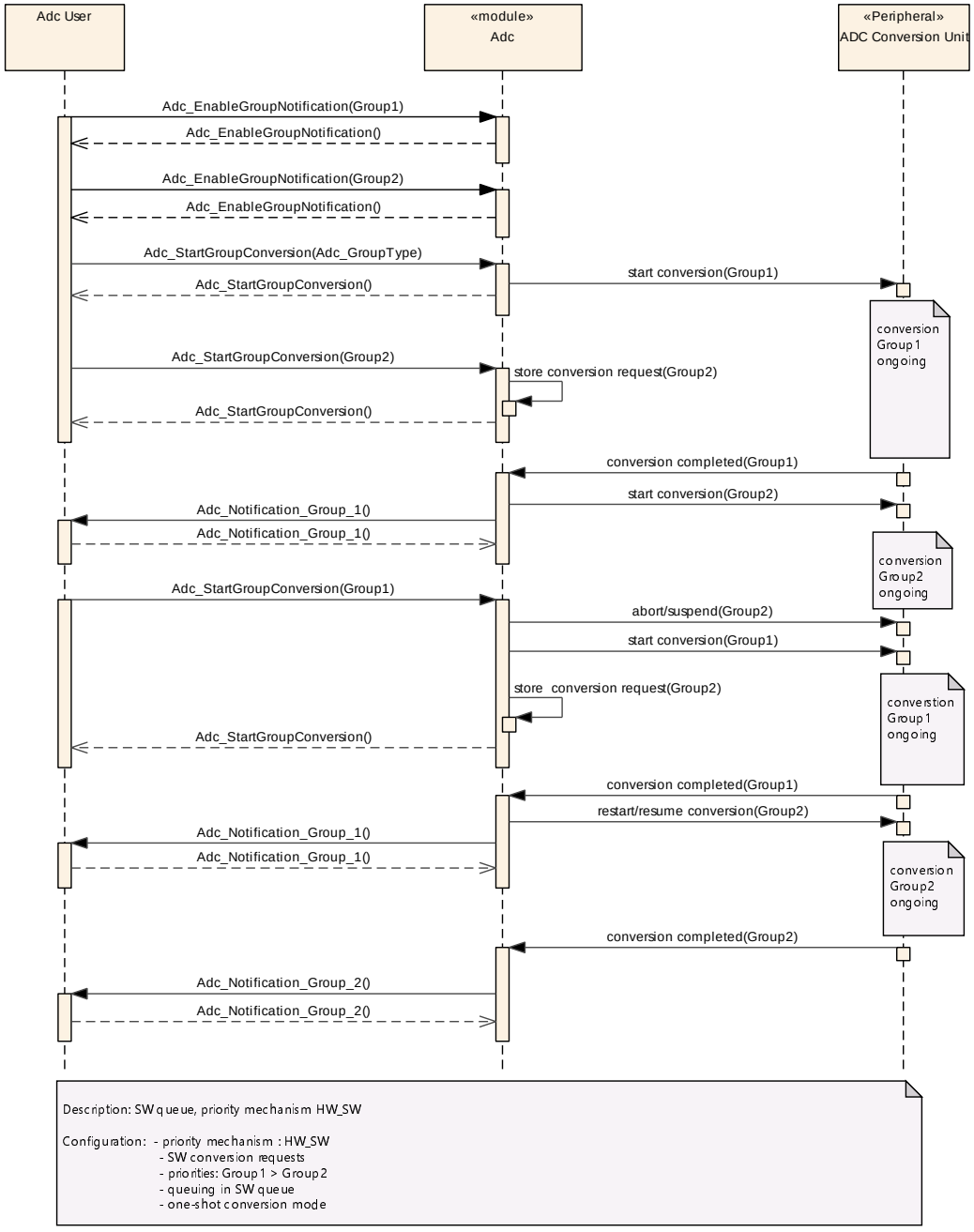


Figure 9.9: Hardware/software priority mechanism - SW queuing

9.10 HW Priority Mechanism - HW Queuing

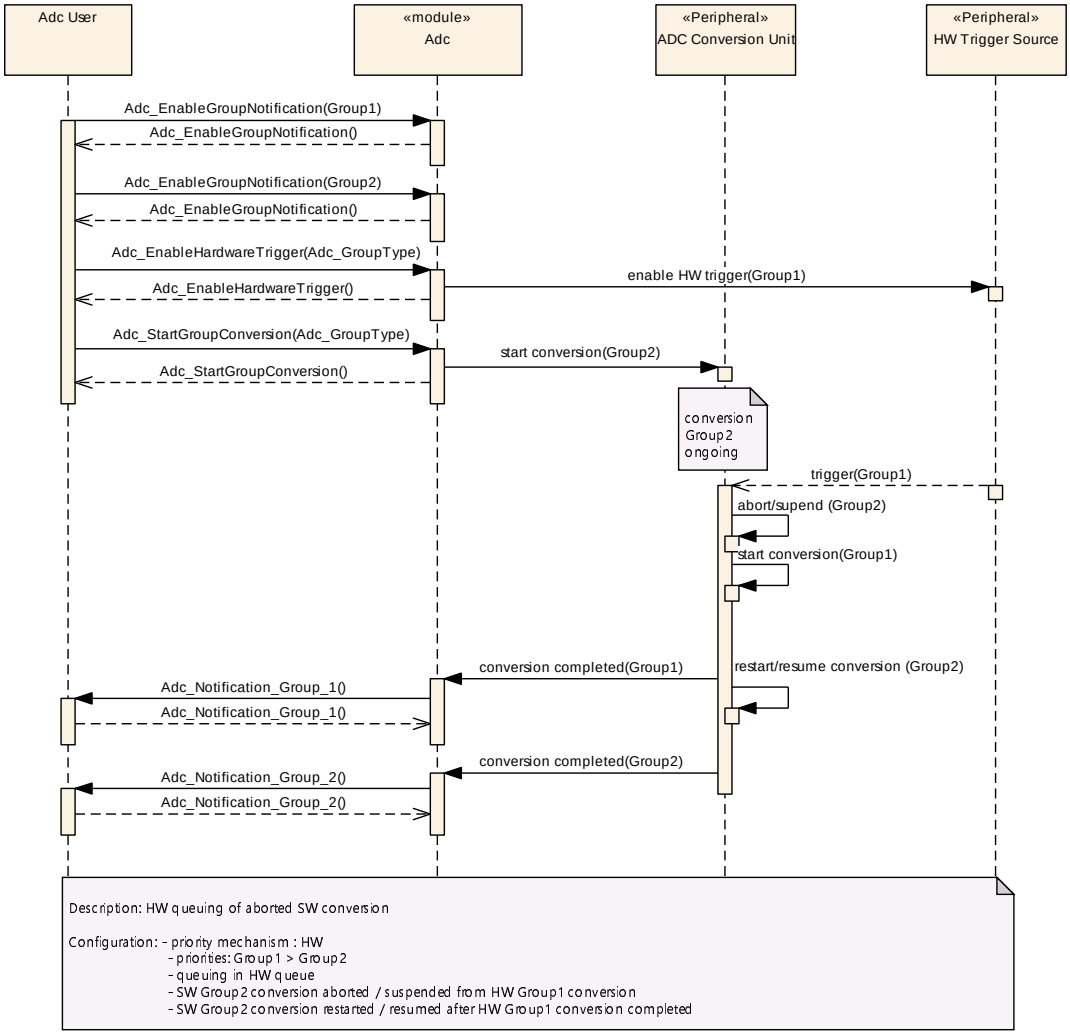


Figure 9.10: Hardware priority mechanism – HW queuing

9.11 HW_SW Priority Mechanism - HW/SW Queuing

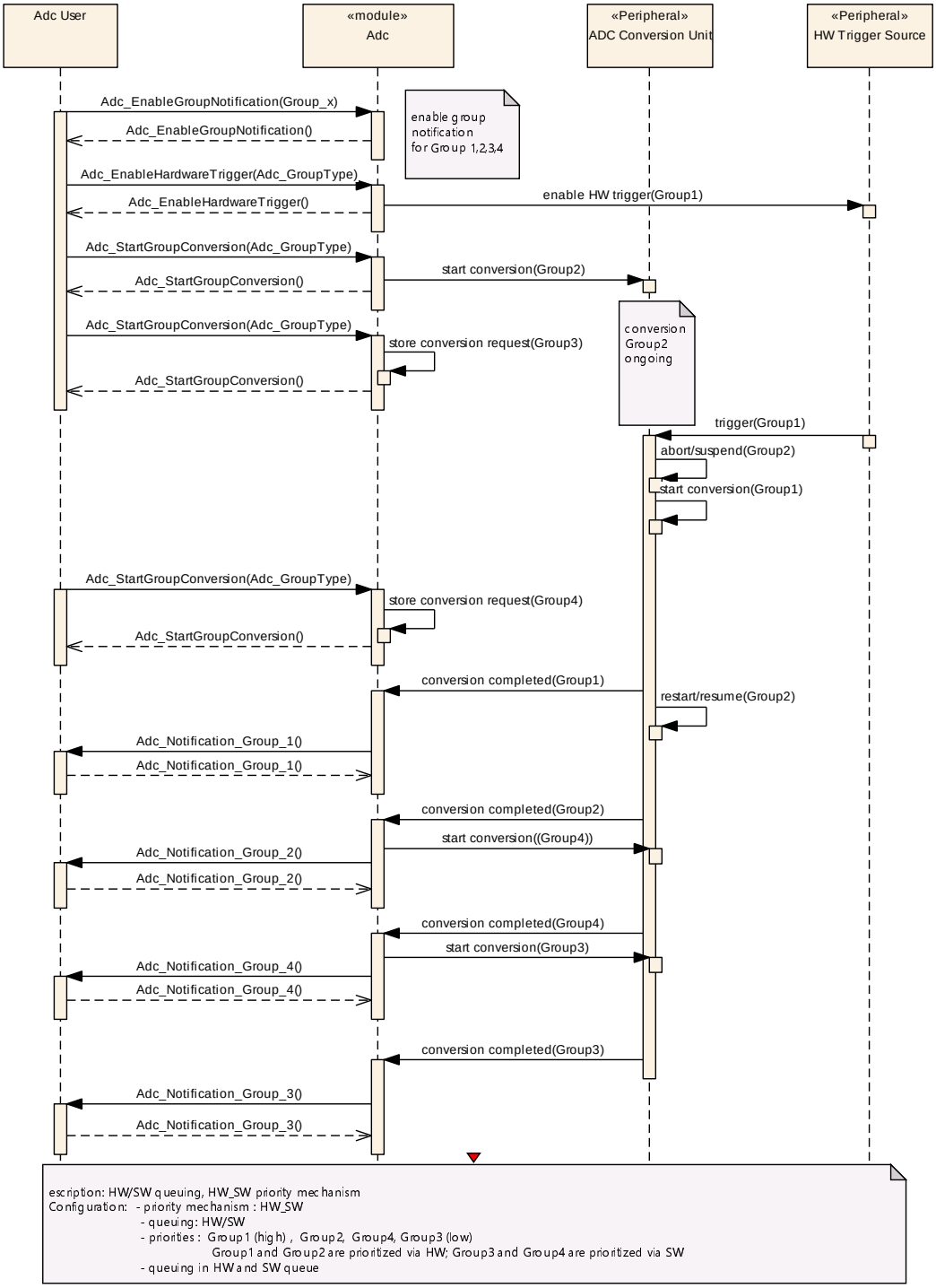


Figure 9.11: Hardware/software priority mechanism – hardware/software queuing

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module ADC_Driver.

Chapter 10.3 specifies published information of the module ADC_Driver.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in SWS_BSWGeneral.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

[SWS_Adc_00531] ⌈The ADC module shall reject configurations with partition mappings which are not supported by the implementation.⌋*()*

10.2.1 Adc

SWS Item	[ECUC_Adc_00462]
Module Name	Adc
Description	Configuration of the Adc (Analog Digital Conversion) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	Multiplicity	Scope / Dependency
AdcConfigSet	1	This container contains the configuration parameters and sub containers of the AUTOSAR Adc module.
AdcGeneral	1	General configuration (parameters) of the ADC Driver software module.
AdcPublishedInformation	1	Additional published parameters not covered by "Common" Published Information. Note that these parameters have "PUBLISHED-INFORMATION" configuration class setting, since they are published information.

10.2.2 AdcGeneral

SWS Item	[ECUC_Adc_00027]		
Container Name	AdcGeneral		
Parent Container	Adc		
Description	General configuration (parameters) of the ADC Driver software module.		
Configuration Parameters			

SWS Item	[ECUC_Adc_00404]		
Parameter Name	AdcDeInitApi		
Parent Container	AdcGeneral		
Description	Adds / removes the service Adc_DeInit() from the code. true: Adc_DeInit() can be used. false: Adc_DeInit() can not be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00405]		
Parameter Name	AdcDevErrorDetect		
Parent Container	AdcGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00452]		
Parameter Name	AdcEnableLimitCheck		
Parent Container	AdcGeneral		
Description	Enables or disables limit checking feature in the ADC driver.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Scope / Dependency	scope: local

SWS Item	**[ECUC_Adc_00391]**		
Parameter Name	AdcEnableQueuing		
Parent Container	AdcGeneral		
Description	Determines, if the queuing mechanism is active in case of priority mechanism disabled. Note: If priority mechanism is enabled, queuing mechanism is always active and the parameter ADC_ENABLE_QUEUING is not evaluated. true: Enabled. false: Disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: AdcPriorityImplementation: parameter is only evaluated for priority implementation ADC_PRIORITY_NONE.		

SWS Item	**[ECUC_Adc_00406]**		
Parameter Name	AdcEnableStartStopGroupApi		
Parent Container	AdcGeneral		
Description	Adds / removes the services Adc_StartGroupConversion() and Adc_StopGroup Conversion() from the code. true: Adc_StartGroupConversion() and Adc_StopGroup Conversion() can be used. false: Adc_StartGroupConversion() and Adc_StopGroup Conversion() can not be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00105]**		
Parameter Name	AdcGrpNotifCapability		
Parent Container	AdcGeneral		
Description	Determines, if the group notification mechanism (the functions to enable and disable the notifications) is available at runtime. true: Enabled. false: Disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00408]		
Parameter Name	AdcHwTriggerApi		
Parent Container	AdcGeneral		
Description	Adds / removes the services Adc_EnableHardwareTrigger() and Adc_DisableHardware Trigger() from the code. true: Adc_EnableHardwareTrigger() and Adc_Disable HardwareTrigger() can be used. false: Adc_EnableHardwareTrigger() and Adc_Disable HardwareTrigger() can not be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00457]		
Parameter Name	AdcLowPowerStatesSupport		
Parent Container	AdcGeneral		
Description	Adds / removes all power state management related APIs (ADC_SetPowerState, ADC_ GetCurrentPowerState, ADC_GetTargetPowerState, ADC_PreparePowerState, ADC_ Main_PowerTransitionManager), indicating if the HW offers low power state management.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00458]		
Parameter Name	AdcPowerStateAsynchTransitionMode		
Parent Container	AdcGeneral		
Description	Enables / disables support of the ADCDriver to the asynchronous power state transition.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: This parameter shall only be configured if the parameter AdcLowPower StatesSupport is set to true.		

SWS Item	**[ECUC_Adc_00393]**		
Parameter Name	AdcPriorityImplementation		
Parent Container	AdcGeneral		
Description	Determines whether a priority mechanism is available for prioritization of the conversion requests and if available, the type of prioritization mechanism. The selection applies for groups with trigger source software and trigger source hardware. Two types of prioritization mechanism can be selected. The hardware prioritization mechanism (AdcPriorityHw) uses the ADC hardware features for prioritization of the software conversion requests and hardware trigger signals for groups with trigger source hardware. The mixed hardware and software prioritization mechanism (Adc PriorityHwSw) uses the ADC hardware features for prioritization of ADC hardware trigger for groups with trigger source hardware and a software implemented prioritization mechanism for groups with trigger source software. The group priorities for software triggered groups are typically configured with lower priority levels than the group priorities for hardware triggered groups. ImplementationType: Adc_PriorityImplementationType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	ADC_PRIORITY_HW	Hardware priority mechanism is available only	
	ADC_PRIORITY_HW_SW	Hardware and software priority mechanism is available	
	ADC_PRIORITY_NONE	priority mechanism is not available	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00394]**		
Parameter Name	AdcReadGroupApi		
Parent Container	AdcGeneral		
Description	Adds / removes the service Adc_ReadGroup() and from the code. true: Adc_Read Group() can be used. false: Adc_ReadGroup() can not be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00444]		
Parameter Name	AdcResultAlignment		
Parent Container	AdcGeneral		
Description	Alignment of ADC raw results in ADC result buffer (left/right alignment). Implementation Type: Adc_ResultAlignmentType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	ADC_ALIGN_LEFT	left alignment	
	ADC_ALIGN_RIGHT	right alignment	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00409]		
Parameter Name	AdcVersionInfoApi		
Parent Container	AdcGeneral		
Description	Adds / removes the service Adc_GetVersionInfo() from the code. true: Adc_GetVersion Info() can be used. false: Adc_GetVersionInfor() can not be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00463]		
Parameter Name	AdcEcucPartitionRef		
Parent Container	AdcGeneral		
Description	Maps the ADC driver to zero or multiple ECUC partitions to make the driver API available in the according partition.		
Multiplicity	0..*		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	[ECUC_Adc_00464]		
Parameter Name	AdcKernelEcucPartitionRef		
Parent Container	AdcGeneral		
Description	Maps the ADC kernel to zero or one ECUC partitions to assign the driver kernel to a certain core. The ECUC partition referenced is a subset of the ECUC partitions where the ADC driver is mapped to.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
AdcPowerStateConfig	0..*	Each instance of this parameter defines a power state and the callback to be called when this power state is reached.

[SWS_Adc_CONSTR_00001] ⌈The ECUC partitions referenced by AdcKernelEcuc PartitionRef shall be a subset of the ECUC partitions referenced by AdcEcucPartition Ref.⌋*()*

[SWS_Adc_CONSTR_00003] ⌈If AdcEcucPartitionRef references one or more ECUC partitions, AdcKernelEcucPartitionRef shall have a multiplicity of one and reference one of these ECUC partitions as well.⌋*()*

10.2.3 AdcPowerStateConfig

SWS Item	[ECUC_Adc_00459]
Container Name	AdcPowerStateConfig
Parent Container	AdcGeneral
Description	Each instance of this parameter defines a power state and the callback to be called when this power state is reached.
Configuration Parameters	

SWS Item	[ECUC_Adc_00461]
Parameter Name	AdcPowerState
Parent Container	AdcPowerStateConfig

▽

△

Description	Each instance of this parameter describes a different power state supported by the ADC HW. It should be defined by the HW supplier and used by the ADCDriver to reference specific HW configurations which set the ADC HW module in the referenced power state. At least the power mode corresponding to full power state shall be always configured.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 18446744073709551615		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: This parameter shall only be configured if the parameter AdcLowPowerStatesSupport is set to true.		

SWS Item	**[ECUC_Adc_00460]**		
Parameter Name	AdcPowerStateReadyCbkRef		
Parent Container	AdcPowerStateConfig		
Description	Each instance of this parameter contains a reference to a power mode callback defined in a CDD or IoHwAbs component.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: This parameter shall only be configured if the parameter AdcLowPowerStatesSupport is set to true.		

No Included Containers

10.2.4 AdcConfigSet

SWS Item	**[ECUC_Adc_00390]**
Container Name	AdcConfigSet
Parent Container	Adc
Description	This container contains the configuration parameters and sub containers of the AUTOSAR Adc module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
AdcHwUnit	1..*	This container contains the Driver configuration (parameters) depending on grouping of channels This container could contain HW specific parameters which are not defined in the Standardized Module Definition. They must be added in the Vendor Specific Module Definition.

10.2.5 AdcChannel

[SWS_Adc_CONSTR_00002] ⌈The ECUC partitions referenced by AdcGroupEcuc PartitionRef shall be a subset of the ECUC partitions referenced by AdcEcucPartition Ref.⌋*()*

SWS Item	**[ECUC_Adc_00028]**
Container Name	AdcGroup
Parent Container	AdcHwUnit
Description	This container contains the Group configuration (parameters).
Configuration Parameters	

SWS Item	**[ECUC_Adc_00317]**		
Parameter Name	AdcGroupAccessMode		
Parent Container	AdcGroup		
Description	Type of access mode to group conversion results. ImplementationType: Adc_GroupAccessModeType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	ADC_ACCESS_MODE_SINGLE	Single value access mode	
	ADC_ACCESS_MODE_ STREAMING	Streaming access mode	
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcGroupTriggSrc / AdcGroupConvMode: streaming access mode is not available for one-shot conversion mode with software trigger source.		

SWS Item	**[ECUC_Adc_00397]**
Parameter Name	AdcGroupConversionMode
Parent Container	AdcGroup
Description	Type of conversion mode supported by the driver. ImplementationType: Adc_GroupConvModeType
Multiplicity	1
Type	EcucEnumerationParamDef

▽

△

Range	ADC_CONV_MODE_ CONTINUOUS	Conversions of an ADC channel group are performed continuously after a software API call (start). The conversions itself are running automatically (no additional software or hardware trigger needed).	
	ADC_CONV_MODE_ONESHOT	The conversion of an ADC channel group is performed once after a trigger.	
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcGroupTriggSrc: Continuous conversion mode only available for software triggered groups.		

SWS Item	**[ECUC_Adc_00398]**		
Parameter Name	AdcGroupId		
Parent Container	AdcGroup		
Description	Numeric ID of the group. This parameter is the symbolic name to be used on the API. This symbolic name allows accessing Channel Group data. This value will be assigned to the symbolic name derived of the AdcGroup container shortName. ImplementationType: Adc_GroupType		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 1023		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00287]**		
Parameter Name	AdcGroupPriority		
Parent Container	AdcGroup		
Description	Priority level of the AdcGroup. ImplementationType: Adc_GroupPriorityType		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD

▽

△

Scope / Dependency	scope: local dependency: ADC_PRIORITY_IMPLEMENTATION

SWS Item	**[ECUC_Adc_00435]**		
Parameter Name	AdcGroupReplacement		
Parent Container	AdcGroup		
Description	Replacement mechanism, which is used on ADC group level, if a group conversion is interrupted by a group which has a higher priority. ImplementationType: Adc_GroupReplacementType		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	ADC_GROUP_REPL_ABORT_RESTART	Abort/Restart mechanism is used on group level, if a group is interrupted by a higher priority group. The complete conversion round of the interrupted group (all group channels) is restarted after the higher priority group conversion is finished. If the group is configured in streaming access mode, only the results of the interrupted conversion round are discarded. Results of previous conversion rounds which are already written to the result buffer are not affected.	
	ADC_GROUP_REPL_SUSPEND_RESUME	Suspend/Resume mechanism is used on group level, if a group is interrupted by a higher priority group. The converions round (conversion of all group channels) of the interrupted group is completed after the higher priority group conversion is finished. If the group is configured in streaming access mode, only the results of the interrupted conversion round are discarded. Results of previous conversion rounds which are already written to the result buffer are not affected.	
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00399]**	
Parameter Name	AdcGroupTriggSrc	
Parent Container	AdcGroup	
Description	Type of source event that starts a group conversion. ImplementationType: Adc_TriggerSourceType	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	ADC_TRIGG_SRC_HW	Group is triggered by a hardware event.
	ADC_TRIGG_SRC_SW	Group is triggered by a software API call.
Post-Build Variant Value	true	

▽

△

Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcGroupConvMode: Trigger source HW is not available for continuous conversion mode.		

SWS Item	**[ECUC_Adc_00400]**		
Parameter Name	AdcHwTrigSignal		
Parent Container	AdcGroup		
Description	Configures on which edge of the hardware trigger signal the driver should react, i.e. start the conversion (only if supported by the ADC hardware). ImplementationType: Adc_HwTriggerSignalType		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	ADC_HW_TRIG_BOTH_EDGES	React on both edges of the hardware trigger signal (only if supported by the ADC hardware).	
	ADC_HW_TRIG_FALLING_EDGE	React on the falling edge of the hardware trigger signal (only if supported by the ADC hardware).	
	ADC_HW_TRIG_RISING_EDGE	React on the rising edge of the hardware trigger signal (only if supported by the ADC hardware).	
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcTriggSrcHw: Valid only if the group is configured to be triggered by a hardware event.		

SWS Item	**[ECUC_Adc_00401]**		
Parameter Name	AdcHwTrigTimer		
Parent Container	AdcGroup		
Description	Reload value of the ADC module embedded timer (only if supported by ADC hardware). ImplementationType: Adc_HwTriggerTimerType		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE

▽

△

	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcTriggSrcHw: Valid only if the group is configured to be triggered by a hardware event.		

SWS Item	**[ECUC_Adc_00402]**		
Parameter Name	AdcNotification		
Parent Container	AdcGroup		
Description	Callback function for each group		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: This parameter is only available, if notification capability is configured available by AdcGrpNotifCapability		

SWS Item	**[ECUC_Adc_00316]**		
Parameter Name	AdcStreamingBufferMode		
Parent Container	AdcGroup		
Description	Configure streaming buffer as "linear buffer" (i.e. the ADC Driver stops the conversion as soon as the stream buffer is full) or as "ring buffer" (wraps around if the end of the stream buffer is reached). ImplementationType: Adc_StreamBufferModeType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	ADC_STREAM_BUFFER_ CIRCULAR	The ADC Driver continues the conversion even if the stream buffer is full (number of samples reached) by wrapping around the stream buffer itself.	
	ADC_STREAM_BUFFER_ LINEAR	The ADC Driver stops the conversion as soon as sthe stream buffer is full (number of samples reached).	
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcGroupAccessMode: Valid only for streaming access mode.		

SWS Item	[ECUC_Adc_00292]		
Parameter Name	AdcStreamingNumSamples		
Parent Container	AdcGroup		
Description	Number of ADC values to be acquired per channel in streaming access mode. Note: in single access mode this parameter assumes value 1, since only one sample per channel is processed. ImplementationType: Adc_StreamNumSampleType		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	1		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcGroupAccessMode: Valid only for streaming access mode. In single access mode this parameter assumes value 1, since only one sample per channel is processed.		

SWS Item	[ECUC_Adc_00014]		
Parameter Name	AdcGroupDefinition		
Parent Container	AdcGroup		
Description	Assignment of AdcChannels to a AdcGroups. ImplementationType: Adc_GroupDefType		
Multiplicity	1..*		
Type	Reference to AdcChannel		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00465]		
Parameter Name	AdcGroupEcucPartitionRef		
Parent Container	AdcGroup		
Description	Maps an ADC channel group to zero or multiple ECUC partitions to limit the access to this channel group. The ECUC partitions referenced are a subset of the ECUC partitions where the ADC driver is mapped to.		
Multiplicity	0..*		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants

△

	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

SWS Item	**[ECUC_Adc_00268]**
Container Name	AdcChannel
Parent Container	AdcHwUnit
Description	This container contains the channel configuration (parameters) depending on the hardware capability. The organization of this data structure could contain dependencies to the microcontroller so this is left up to the implementer and its location is left up to the configuration. Note: Since a AdcChannel can be part of several AdcGroups, this container is not realized as a subcontainer of AdcGroup but instead as a subcontainer of AdcHwUnit.
Configuration Parameters	

SWS Item	**[ECUC_Adc_00011]**		
Parameter Name	AdcChannelConvTime		
Parent Container	AdcChannel		
Description	Configuration of conversion time, i.e. the time during which the analogue value is converted into digital representation, (in clock cycles) for each channel, if supported by hardware. ImplementationType: Adc_ConversionTimeType		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00455]**	
Parameter Name	AdcChannelHighLimit	
Parent Container	AdcChannel	
Description	High limit - used for limit checking.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	

▽

△

Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: AdcEnableLimitCheck: not available if limit checking is not globally enabled. AdcChannelLimitCheck: not available if channel specific limit check is not enabled. AdcChannelLowLimit: has to be greater or equal than AdcChannelLowLimit.		

SWS Item	**[ECUC_Adc_00392]**		
Parameter Name	AdcChannelId		
Parent Container	AdcChannel		
Description	This parameter defines the assignment of the channel to the physical ADC hardware channel. ImplementationType: Adc_ChannelType		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 1024		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00453]**		
Parameter Name	AdcChannelLimitCheck		
Parent Container	AdcChannel		
Description	Enables or disables limit checking for an ADC channel.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Scope / Dependency	scope: local dependency: AdcEnableLimitCheck: not available if limit checking is not globaly enabled. AdcGroupDefinition: ADC channels with limit checking feature enabled have to be assigned to ADC groups which consist exactly of one limit checking enabled ADC channel.

SWS Item	**[ECUC_Adc_00454]**		
Parameter Name	AdcChannelLowLimit		
Parent Container	AdcChannel		
Description	Low limit - used for limit checking.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: AdcEnableLimitCheck: not available if limit checking is not globally enabled. AdcChannelLimitCheck: not available if channel specific limit check is not enabled. AdcChannelHighLimit: has to be less or equal than AdcChannelHighLimit.		

SWS Item	**[ECUC_Adc_00456]**	
Parameter Name	AdcChannelRangeSelect	
Parent Container	AdcChannel	
Description	In case of active limit checking: defines which conversion values are taken into account related to the boarders defined with AdcChannelLowLimit and AdcChannelHighLimit. Implementation Type: Adc_ChannelRangeSelectType	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	ADC_RANGE_ALWAYS	Complete range - independent from channel limit settings.
	ADC_RANGE_BETWEEN	Range between low limit and high limit - high limit value included.
	ADC_RANGE_NOT_BETWEEN	Range above high limit or below low limit - low limit value included.
	ADC_RANGE_NOT_OVER_HIGH	Range below high limit - high limit value included.
	ADC_RANGE_NOT_UNDER_LOW	Range above low limit.
	ADC_RANGE_OVER_HIGH	Range above high limit.
	ADC_RANGE_UNDER_LOW	Range below limit - low limit value included.
Post-Build Variant Multiplicity	false	
Post-Build Variant Value	false	
Multiplicity Configuration Class	**Pre-compile time**	X All Variants

▽

△

	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: AdcEnableLimitCheck: not available if limit checking is not globally enabled. AdcChannelLimitCheck: not available if channel specific limit check is not enabled.		

SWS Item	**[ECUC_Adc_00089]**		
Parameter Name	AdcChannelRefVoltsrcHigh		
Parent Container	AdcChannel		
Description	Upper reference voltage source for each channel. Enumeration literals are defined vendor specific.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00023]**		
Parameter Name	AdcChannelRefVoltsrcLow		
Parent Container	AdcChannel		
Description	Lower reference voltage source for each channel. Enumeration literals are defined vendor specific.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00019]		
Parameter Name	AdcChannelResolution		
Parent Container	AdcChannel		
Description	Channel resolution in bits. ImplementationType: Adc_ResolutionType		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 63		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: AdcMaxChannelResolution: The actual resolution has to be less or equal than the maximum resolution.		

SWS Item	[ECUC_Adc_00290]		
Parameter Name	AdcChannelSampTime		
Parent Container	AdcChannel		
Description	Configuration of sampling time, i.e. the time during which the value is sampled, (in clock cycles) for each channel, if supported by hardware. ImplementationType: Adc_SamplingTimeType		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

[SWS_Adc_CONSTR_00004] ⌈If AdcEcucPartitionRef references one or more ECUC partitions, AdcGroupEcucPartitionRef shall have a multiplicity of greater than zero and reference one or several of these ECUC partitions as well.⌋*()*

[SWS_Adc_00098] ⌈(refers to ADC396): All channels of a group share the same group configuration (channel can have different channel specific configurations).⌋*(SRS_Adc_12447)*

10.2.6 AdcHwUnit

SWS Item	[ECUC_Adc_00242]		
Container Name	AdcHwUnit		
Parent Container	AdcConfigSet		
Description	This container contains the Driver configuration (parameters) depending on grouping of channels This container could contain HW specific parameters which are not defined in the Standardized Module Definition. They must be added in the Vendor Specific Module Definition.		
Configuration Parameters			

SWS Item	[ECUC_Adc_00087]		
Parameter Name	AdcClockSource		
Parent Container	AdcHwUnit		
Description	The ADC module specific clock input for the conversion unit can statically be configured to select different clock sources if provided by hardware. Enumeration literals are defined vendor specific.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00389]		
Parameter Name	AdcHwUnitId		
Parent Container	AdcHwUnit		
Description	Description: Numeric ID of the HW Unit. This symbolic name allows accessing Hw Unit data. Enumeration literals are defined vendor specific.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Adc_00088]		
Parameter Name	AdcPrescale		
Parent Container	AdcHwUnit		
Description	Optional ADC module specific clock prescale factor, if supported by hardware. ImplementationType: Adc_PrescaleType		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
AdcChannel	1..*	This container contains the channel configuration (parameters) depending on the hardware capability. The organization of this data structure could contain dependencies to the microcontroller so this is left up to the implementer and its location is left up to the configuration. Note: Since a AdcChannel can be part of several AdcGroups, this container is not realized as a subcontainer of AdcGroup but instead as a subcontainer of Adc HwUnit.
AdcGroup	1..*	This container contains the Group configuration (parameters).

[SWS_Adc_00138] ⌈(refers to ADC242): The ADC Driver shall support one or several ADC HW Units of the same type. The selection of ADC HW Unit shall be done by the configuration container AdcHwUnit.⌋*()*

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in SWS_BSWGeneral.

10.3.1 AdcPublishedInformation

SWS Item	[ECUC_Adc_00030]
Container Name	AdcPublishedInformation
Parent Container	Adc

▽

△

Description	Additional published parameters not covered by "Common" Published Information. Note that these parameters have "PUBLISHED-INFORMATION" configuration class setting, since they are published information.
Configuration Parameters	

SWS Item	**[ECUC_Adc_00410]**		
Parameter Name	AdcChannelValueSigned		
Parent Container	AdcPublishedInformation		
Description	Information whether the result value of the ADC driver has sign information (true) or not (false). If the result shall be interpreted as signed value it shall apply to C-language rules.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00411]**		
Parameter Name	AdcGroupFirstChannelFixed		
Parent Container	AdcPublishedInformation		
Description	Information whether the first channel of an ADC Channel group can be configured (false) or is fixed (true) to a value determined by the ADC HW Unit.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Adc_00412]**		
Parameter Name	AdcMaxChannelResolution		
Parent Container	AdcPublishedInformation		
Description	Maximum Channel resolution in bits (does not specify accuracy).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 63		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Published Information**	X	All Variants
Scope / Dependency	scope: local		

No Included Containers

10.4 Configuration of symbolic names

[SWS_Adc_00099] ⌈The symbolic names of ADC channels and ADC channel groups for use by the upper layer shall be defined by the configurator.They are to be defined in the modules configuration header file.⌋ *(SRS_Adc_12307, SRS_Adc_12447)*

A Not applicable requirements

[SWS_Adc_NA_00460] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00344, SRS_BSW_00167, SRS_BSW_00170, SRS_BSW_00398, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00373, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00357, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00330, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00341, SRS_BSW_00334, SRS_SPAL_12267, SRS_SPAL_12463, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_12169, SRS_SPAL_12064, SRS_SPAL_12067, SRS_SPAL_12077, SRS_SPAL_12078, SRS_SPAL_12092, SRS_SPAL_12265)*