### Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-12-08</td>
<td>4.3.1</td>
<td>AUTOSAR Release Management</td>
<td>• Runtime error introduced; part of development errors changed into runtime errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Exclude delta sigma ADC hardware from scope of ADC driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Minor modifications in API Adc_SetupResultBuffer and Adc_ReadGroup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Header file structure update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Editorial changes</td>
</tr>
<tr>
<td>2016-11-30</td>
<td>4.3.0</td>
<td>AUTOSAR Release Management</td>
<td>• Variant-Post-Build requirements removed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Variant specific requirements for initialization API removed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Error classification table update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Editorial changes</td>
</tr>
<tr>
<td>2015-07-31</td>
<td>4.2.2</td>
<td>AUTOSAR Release Management</td>
<td>• DET changed from ‘Development Error Tracer’ to ‘Default Error Tracer’.</td>
</tr>
<tr>
<td>2014-10-31</td>
<td>4.2.1</td>
<td>AUTOSAR Release Management</td>
<td>• AdcGroupId is changed to pre-compile time value in all variants.</td>
</tr>
<tr>
<td>2014-03-31</td>
<td>4.1.3</td>
<td>AUTOSAR Release Management</td>
<td>• “Common” Published Information corrected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ARXML adaptations</td>
</tr>
<tr>
<td>2013-10-31</td>
<td>4.1.2</td>
<td>AUTOSAR Release Management</td>
<td>• Editorial changes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed chapter(s) on change documentation</td>
</tr>
</tbody>
</table>
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-03-15</td>
<td>4.1.1</td>
<td>AUTOSAR Administration</td>
<td>• API and configuration parameter added to support ECU degradation concept  &lt;br&gt;• Common Published Information removed  &lt;br&gt;• BSW General rework</td>
</tr>
<tr>
<td>2011-12-22</td>
<td>4.0.3</td>
<td>AUTOSAR Administration</td>
<td>• Requirement of ADC group status to be available for debugging removed</td>
</tr>
<tr>
<td>2009-12-18</td>
<td>4.0.1</td>
<td>AUTOSAR Administration</td>
<td>• ADC444 add  Adc_ResultAlignmentType  &lt;br&gt;• SWS_Adc_00124 version number check correction  &lt;br&gt;• SWS_Adc_00337 reformulation  &lt;br&gt;• Limitation of ranges for AdcPrescale and AdcChannelId  &lt;br&gt;• InstanceId removed  &lt;br&gt;• ADC324 removed,  &lt;br&gt;• SWS_Adc_00458 introduced , DET for Adc_GetVersionInfo</td>
</tr>
<tr>
<td>2010-02-02</td>
<td>3.1.4</td>
<td>AUTOSAR Administration</td>
<td>• Limit checking support included; new config parameters added  AdcEnableLimitCheck,  AdcChannelLimitCheck,  AdcChannelLowLimit,  AdcChannelHighLimit and  AdcChannelRangeSelect introduced.  &lt;br&gt;• ADC debug support added.  &lt;br&gt;• ADC configurable ADC data buffer alignment added.  &lt;br&gt;• Min/max values for AdcGroupIld,  AdcStreamingNumSamples,  AdcMaxChannelResolution and  AdcChannelResolution added.  &lt;br&gt;• Legal disclaimer revised</td>
</tr>
<tr>
<td>2008-08-13</td>
<td>3.1.1</td>
<td>AUTOSAR Administration</td>
<td>• Legal disclaimer revised</td>
</tr>
<tr>
<td>2008-02-01</td>
<td>3.0.2</td>
<td>AUTOSAR Administration</td>
<td>• Correction of: Table of Content</td>
</tr>
</tbody>
</table>
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
</table>
| 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_LEVE L  
- *ADC_STREAMING_BUFFER_POI NTER  
- 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 |
### Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-01-24</td>
<td>2.1.15</td>
<td>AUTOSAR Administration</td>
<td>• Part of existing requirements reformulated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added new requirement ID's SWS_Adc_00321-SWS_Adc_00432</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Document meta information extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Small layout adaptations made</td>
</tr>
<tr>
<td>2006-11-28</td>
<td>2.1.14</td>
<td>AUTOSAR Administration</td>
<td>• “Advice for users” revised</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• “Revision Information” added</td>
</tr>
<tr>
<td>2006-05-16</td>
<td>2.0</td>
<td>AUTOSAR Administration</td>
<td>• Removed the &quot;On Demand&quot; functionality. Related services not available anymore.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed the &quot;Gated Continuous&quot; conversion mode. Related services not available anymore.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed the distinction between internal and external hardware trigger.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 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).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Reworked the “Streaming Access Mode”. A dedicated data structure for the returned values of a conversion is now clearly defined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Conversion values access now allowed only through channel groups (no single channel value available. Related service not available anymore).</td>
</tr>
<tr>
<td>2005-05-31</td>
<td>1.0</td>
<td>AUTOSAR Administration</td>
<td>• Document structure adapted to common Release 2.0 SWS Template.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Initial Release.</td>
</tr>
</tbody>
</table>

2005-05-31 1.0 AUTOSAR Administration • Initial Release.
Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.
# Table of Contents

1. Introduction and functional overview ................................................................. 9  
2. Acronyms and abbreviations ............................................................................. 10  
3. Related documentation ................................................................................... 11  
   3.1 Input documents .......................................................................................... 11  
   3.2 Related specification ................................................................................ 11  
4. Constraints and assumptions .......................................................................... 13  
   4.1 Limitations ................................................................................................. 13  
   4.2 Applicability to car domains ...................................................................... 13  
5. Dependencies to other modules ....................................................................... 14  
   5.1 File structure ............................................................................................. 14  
       5.1.1 Header file structure .......................................................................... 14  
6. Requirements traceability ................................................................................. 16  
7. Functional specification ...................................................................................... 24  
   7.1 General behavior ....................................................................................... 24  
       7.1.1 Background & Rationale .................................................................... 24  
       7.1.2 Requirements ................................................................................... 24  
       7.1.3 ADC Buffer Access Mode Example .................................................. 32  
   7.2 Conversion processing and interaction ....................................................... 35  
       7.2.1 Background & Rationale .................................................................... 35  
       7.2.2 Requirements ................................................................................... 36  
   7.3 State Diagrams ............................................................................................ 37  
       7.3.1 ADC State Diagram for One-Shot/Continuous Group Conversion Mode ................................................................. 37  
       7.3.2 ADC State Diagram for HW/SW Trigger in One-Shot Group Conversion Mode ...................................................... 38  
       7.3.3 ADC State Diagram for SW Trigger in Continuous Conversion Mode ................................................................. 39  
       7.3.4 ADC State Diagram for One-Shot Conversion Mode, Software Trigger Source, Single Access Mode ......................... 40  
       7.3.5 ADC State Diagram for One-Shot Conversion, Hardware Trigger Source, Single Access Mode ............................ 41  
       7.3.6 ADC State Diagram for One-Shot Conversion Mode, Hardware Trigger Source, Linear and Circular Streaming Access Mode ................................. 42  
       7.3.7 ADC State Diagram for Continuous Conversion Mode, Software Trigger Source, Single Access Mode ..................... 43  
       7.3.8 ADC State Diagram for Continuous Conversion Mode, Software Trigger Source, Linear and Circular Streaming Access Mode ................................. 44  
   7.4 Support and management of HW low power states ...................................... 45  
       7.4.1 Background ....................................................................................... 45  
       7.4.2 Requirements ................................................................................... 45  
   7.5 Version check .............................................................................................. 47  
       7.5.1 Background & Rationale .................................................................... 47  
   7.6 Error detection ............................................................................................ 50  

- AUTOSAR confidential -
7.6.1 Development Error .................................................. 50
7.6.2 Runtime Error .......................................................... 52
7.6.3 Transient Faults ....................................................... 52

8 API specification .......................................................... 54

8.1 Imported types .................................................................. 54
8.2 Type definitions .............................................................. 54
  8.2.1 Adc_ConfigType ...................................................... 54
  8.2.2 Adc_ChannelType ................................................... 54
  8.2.3 Adc_GroupType ..................................................... 54
  8.2.4 Adc_ValueGroupType ............................................... 54
  8.2.5 Adc_PrescaleType .................................................. 55
  8.2.6 Adc_ConversionTimeType ....................................... 55
  8.2.7 Adc_SamplingTimeType ........................................... 56
  8.2.8 Adc_ResolutionType ............................................... 56
  8.2.9 Adc_StatusType .................................................... 56
  8.2.10 Adc_TriggerSourceType ......................................... 56
  8.2.11 Adc_GroupConvModeType ..................................... 57
  8.2.12 Adc_GroupPriorityType ....................................... 57
  8.2.13 Adc_GroupDefType .............................................. 57
  8.2.14 Adc_StreamNumSampleType ................................... 57
  8.2.15 Adc_StreamBufferModeType ................................... 58
  8.2.16 Adc_GroupAccessModeType .................................... 58
  8.2.17 Adc_HwTriggerSignalType .................................... 58
  8.2.18 Adc_HwTriggerTimerType ...................................... 58
  8.2.19 Adc_PriorityImplementationType ............................ 59
  8.2.20 Adc_GroupReplacementType .................................. 59
  8.2.21 Adc_ChannelRangeSelectType ................................ 59
  8.2.22 Adc_ResultAlignmentType ..................................... 60
  8.2.23 Adc_PowerStateType .......................................... 60
  8.2.24 Adc_PowerStateRequestResultType ......................... 60

8.3 Function definitions .................................................... 60
  8.3.1 Adc_Init ............................................................. 60
  8.3.2 Adc_SetupResultBuffer ........................................ 62
  8.3.3 Adc_DeInit .......................................................... 63
  8.3.4 Adc_StartGroupConversion ..................................... 64
  8.3.5 Adc_StopGroupConversion ...................................... 67
  8.3.6 Adc_ReadGroup .................................................... 69
  8.3.7 Adc_EnableHardwareTrigger .................................. 71
  8.3.8 Adc_DisableHardwareTrigger ................................. 73
  8.3.9 Adc_EnableGroupNotification ................................. 74
  8.3.10 Adc_DisableGroupNotification .............................. 75
  8.3.11 Adc_GetGroupStatus ........................................... 76
  8.3.12 Adc_GetStreamLastPointer ................................... 79
  8.3.13 Adc_GetVersionInfo ........................................... 81
  8.3.14 Adc_SetPowerState ............................................ 81
  8.3.15 Adc_GetCurrentPowerState .................................. 83
  8.3.16 Adc_GetTargetPowerState .................................... 84
  8.3.17 Adc_PreviewPowerState ...................................... 85

8.4 Call-back Notifications ............................................... 86
8.5 Scheduled functions .................................................................................86
  8.5.1 Adc_Main_PowerTransitionManager ................................................86
8.6 Expected Interfaces .................................................................................88
  8.6.1 Mandatory Interfaces ..........................................................................88
  8.6.2 Optional Interfaces .............................................................................88
  8.6.3 Configurable interfaces .......................................................................88
9 Sequence diagrams .....................................................................................91
  9.1 Initialization of the ADC Driver ..............................................................91
  9.2 De-Initialization of the ADC Driver .......................................................91
  9.3 Software triggered One-Shot conversion without notification .................91
  9.4 Software triggered continuous conversion with notification ....................93
  9.5 Hardware triggered One-Shot conversion with notification .......................94
  9.6 HW Trigger - One-Shot conversion - Linear Streaming .........................95
  9.7 No Priority Mechanism – No Queuing ...................................................96
  9.8 No Priority Mechanism – SW Queuing ..................................................97
  9.9 HW_SW Priority Mechanism – SW Queuing .........................................98
  9.10 HW Priority Mechanism – HW Queuing ...............................................99
  9.11 HW_SW Priority Mechanism – HW/SW Queuing .................................100
10 Configuration specification ........................................................................102
  10.1 How to read this chapter .......................................................................102
  10.2 Configuration and configuration parameters .......................................102
    10.2.1 Adc ...............................................................................................102
    10.2.2 AdcGeneral ..................................................................................103
    10.2.3 AdcPowerStateConfig .................................................................107
    10.2.4 AdcConfigSet ...............................................................................108
    10.2.5 AdcChannel ..................................................................................108
    10.2.6 AdcGroup ....................................................................................113
    10.2.7 AdcHwUnit ..................................................................................119
  10.3 Published information ..........................................................................120
    10.3.1 AdcPublishedInformation ............................................................120
  10.4 Configuration of symbolic names ...........................................................121
11 Not applicable requirements .......................................................................122
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

<table>
<thead>
<tr>
<th>Abbreviation / Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEM</td>
<td>Diagnostic Event Manager</td>
</tr>
<tr>
<td>DET</td>
<td>Default Error Tracer</td>
</tr>
<tr>
<td>ADC</td>
<td>Analogue Digital Converter</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller Unit</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>ADC HW Unit</td>
<td>Represents a microcontroller input electronic device that includes all parts necessary to perform an “analogue to digital conversion”.</td>
</tr>
<tr>
<td>ADC Module</td>
<td>ADC Basic Software module ADC Driver, abbreviated also with ADC Driver</td>
</tr>
<tr>
<td>ADC Channel</td>
<td>Represents a logical ADC entity bound to one port pin. Multiple ADC entities can be mapped to the same port pin.</td>
</tr>
<tr>
<td>ADC Channel Group</td>
<td>A group of ADC channels linked to the same ADC hardware unit (e.g. one Sample&amp;Hold and one A/D converter). The conversion of the whole group is triggered by one trigger source.</td>
</tr>
<tr>
<td>ADC Result Buffer (ADC Streaming Buffer, ADC Stream Buffer)</td>
<td>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.</td>
</tr>
<tr>
<td>Software Trigger</td>
<td>Software API call that starts the conversion of one ADC channel group or a continuous series of ADC channel group conversions.</td>
</tr>
<tr>
<td>Hardware Trigger</td>
<td>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. <strong>Note:</strong> 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.</td>
</tr>
</tbody>
</table>
| 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 1: Acronyms and abbreviations used in this document
3 Related documentation

3.1 Input documents


[12] Basic Software Module Description Template, AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf


3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [13] (SWS BSW General), 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.

5.1 File structure

5.1.1 Header file structure

[SWS_Adca_00267] The file include structure shall be as follows.

Figure 1: ADC Driver file include structure

[SRS_BSW_00381, SRS_BSW_00412, SRS_BSW_00383, SRS_BSW_00415, SRS_BSW_00300, SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00314, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361]
Note:
By this inclusion the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.
## 6 Requirements traceability

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Satisfied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRS_Adc_12280</td>
<td>The ADC Driver shall allow a specific result access modes for each ADC Channel Group</td>
<td>SWS_Adc_00140, SWS_Adc_00382, SWS_Adc_00383</td>
</tr>
<tr>
<td>SRS_Adc_12283</td>
<td>The ADC driver shall mask out information bits from the conversion result not belonging to the ADC value</td>
<td>SWS_Adc_00122</td>
</tr>
<tr>
<td>SRS_Adc_12291</td>
<td>The ADC Driver shall provide a service for querying the status of an ADC Channel Group</td>
<td>SWS_Adc_00219, SWS_Adc_00220, SWS_Adc_00221, SWS_Adc_00224, SWS_Adc_00325, SWS_Adc_00327, SWS_Adc_00329, SWS_Adc_00331</td>
</tr>
<tr>
<td>SRS_Adc_12292</td>
<td>If the ADC provides signed values, the ADC driver shall put the sign bit into the MSB of the return value</td>
<td>SWS_Adc_00113, SWS_Adc_00214</td>
</tr>
<tr>
<td>SRS_Adc_12307</td>
<td>The ADC Driver shall support a specific basic static configurations per channel</td>
<td>SWS_Adc_00099</td>
</tr>
<tr>
<td>SRS_Adc_12317</td>
<td>The ADC Driver shall provide notification functions to inform the caller about the end of a conversion for a Channel Group</td>
<td>SWS_Adc_00104, SWS_Adc_00155, SWS_Adc_00156, SWS_Adc_00157</td>
</tr>
<tr>
<td>SRS_Adc_12318</td>
<td>The ADC driver shall provide a service to enable and disable each notification function separately</td>
<td>SWS_Adc_00057, SWS_Adc_00077, SWS_Adc_00157</td>
</tr>
<tr>
<td>SRS_Adc_12364</td>
<td>The ADC driver shall provide services to start and stop the conversion of an ADC Channel Group for all conversion modes</td>
<td>SWS_Adc_00060, SWS_Adc_00145, SWS_Adc_00157, SWS_Adc_00357, SWS_Adc_00386</td>
</tr>
<tr>
<td>SRS_Adc_12447</td>
<td>The ADC Driver shall allow to group ADC channels that belong to the same ADC HW unit</td>
<td>SWS_Adc_00090, SWS_Adc_00098, SWS_Adc_00100, SWS_Adc_00104, SWS_Adc_00280</td>
</tr>
<tr>
<td>SRS_Adc_12802</td>
<td>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</td>
<td>SWS_Adc_00214, SWS_Adc_00219</td>
</tr>
<tr>
<td>SRS_Adc_12817</td>
<td>The ADC Driver shall allow for each ADC channel group the</td>
<td>SWS_Adc_00146, SWS_Adc_00283, SWS_Adc_00279, SWS_Adc_00356</td>
</tr>
<tr>
<td>Requirement ID</td>
<td>Description</td>
<td>Compliance IDs</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>SRS_Ad1_1218</td>
<td>The ADC Driver shall allow accessing one ADC channel to more than one ADC Channel Group</td>
<td>SWS_Ad1_00092, SWS_Ad1_00100</td>
</tr>
<tr>
<td>SRS_Ad1_1219</td>
<td>The ADC Driver shall provide a synchronous service for reading the last valid conversion results of the selected channel group</td>
<td>SWS_Ad1_00113, SWS_Ad1_00114, SWS_Ad1_00122, SWS_Ad1_00129</td>
</tr>
<tr>
<td>SRS_Ad1_1220</td>
<td>The ADC driver shall allow the configuration of a priority level for each channel group</td>
<td>SWS_Ad1_00288, SWS_Ad1_00289, SWS_Ad1_00300, SWS_Ad1_00340, SWS_Ad1_00341</td>
</tr>
<tr>
<td>SRS_Ad1_1222</td>
<td>The structure containing the results of a channel group conversion shall be generated with a uniform dimension</td>
<td>SWS_Ad1_00320, SWS_Ad1_00319, SWS_Ad1_00324</td>
</tr>
<tr>
<td>SRS_Ad1_1223</td>
<td>The ADC driver shall provide services to enable and disable HW triggers for each channel group</td>
<td>SWS_Ad1_00114, SWS_Ad1_00116, SWS_Ad1_00144, SWS_Ad1_00273, SWS_Ad1_00281, SWS_Ad1_00282</td>
</tr>
<tr>
<td>SRS_Ad1_1224</td>
<td>The result alignment shall be configurable between right-alignment and left-alignment</td>
<td>SWS_Ad1_00113</td>
</tr>
<tr>
<td>SRS_Ad1_1225</td>
<td>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</td>
<td>SWS_Ad1_00319</td>
</tr>
<tr>
<td>SRS_BSW_00005</td>
<td>Modules of the µC Abstraction Layer (MCAL) may not have hard-coded horizontal interfaces</td>
<td>SWS_Ad1_00460</td>
</tr>
<tr>
<td>SRS_BSW_00006</td>
<td>The source code of software modules above the µC Abstraction Layer (MCAL) shall not be processor and compiler dependent.</td>
<td>SWS_Ad1_00460</td>
</tr>
<tr>
<td>SRS_BSW_00007</td>
<td>All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.</td>
<td>SWS_Ad1_00460</td>
</tr>
<tr>
<td>SRS_BSW_00009</td>
<td>All Basic SW Modules shall be documented according to a common standard.</td>
<td>SWS_Ad1_00460</td>
</tr>
<tr>
<td>SRS_BSW_00010</td>
<td>The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.</td>
<td>SWS_Ad1_00460</td>
</tr>
<tr>
<td>Specification Item</td>
<td>Description</td>
<td>ID Reference</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>SRS_BSW_00101</strong></td>
<td>The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function</td>
<td>SWS_Adcd_00054</td>
</tr>
<tr>
<td><strong>SRS_BSW_00160</strong></td>
<td>Configuration files of AUTOSAR Basic SW module shall be readable for human beings</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00161</strong></td>
<td>The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00162</strong></td>
<td>The AUTOSAR Basic Software shall provide a hardware abstraction layer</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00164</strong></td>
<td>The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00167</strong></td>
<td>All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00168</strong></td>
<td>SW components shall be tested by a function defined in a common API in the Basis-SW</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00170</strong></td>
<td>The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00171</strong></td>
<td>Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time</td>
<td>SWS_Adcd_00120, SWS_Adcd_00121, SWS_Adcd_00228, SWS_Adcd_00259, SWS_Adcd_00260, SWS_Adcd_00265, SWS_Adcd_00266</td>
</tr>
<tr>
<td><strong>SRS_BSW_00301</strong></td>
<td>All AUTOSAR Basic Software Modules shall only import the necessary information</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00302</strong></td>
<td>All AUTOSAR Basic Software Modules shall only export information needed by other modules</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00306</strong></td>
<td>AUTOSAR Basic Software Modules shall be compiler and platform independent</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td><strong>SRS_BSW_00307</strong></td>
<td>Global variables naming</td>
<td>SWS_Adcd_00460</td>
</tr>
<tr>
<td>Specification of ADC Driver</td>
<td>AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00312</td>
<td>Shared code shall be reentrant</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00323</td>
<td>All AUTOSAR Basic Software Modules shall check passed API parameters for validity</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00325</td>
<td>The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00328</td>
<td>All AUTOSAR Basic Software Modules shall avoid the duplication of code</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00330</td>
<td>It shall be allowed to use macros instead of functions where source code is used and runtime is critical</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00334</td>
<td>All Basic Software Modules shall provide an XML file that contains the meta data</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00335</td>
<td>Status values naming convention</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00336</td>
<td>Basic SW module shall be able to shutdown</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00341</td>
<td>Module documentation shall contain all needed informations</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00342</td>
<td>It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00343</td>
<td>The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00344</td>
<td>BSW Modules shall support link-time configuration</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00347</td>
<td>A Naming separation of different instances of BSW drivers shall be in place</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00357</td>
<td>For success/failure of an API call a standard return type shall be defined</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00359</td>
<td>All AUTOSAR Basic Software</td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>Modules callback functions shall avoid return types other than void if possible</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00360</td>
<td>AUTOSAR Basic Software Modules callback functions are allowed to have parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00082</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00371</td>
<td>The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00373</td>
<td>The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00375</td>
<td>Basic Software Modules shall report wake-up reasons</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00386</td>
<td>The BSW shall specify the configuration for detecting an error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00107, SWS_Ad_00125, SWS_Ad_00126, SWS_Ad_00129, SWS_Ad_00133, SWS_Ad_00137, SWS_Ad_00152, SWS_Ad_00154, SWS_Ad_00164, SWS_Ad_00165, SWS_Ad_00218, SWS_Ad_00225, SWS_Ad_00241</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00398</td>
<td>The link-time configuration is achieved on object code basis in the stage after compiling and before linking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00405</td>
<td>BSW Modules shall support multiple configuration sets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00054</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00406</td>
<td>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</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00107, SWS_Ad_00125, SWS_Ad_00126, SWS_Ad_00129, SWS_Ad_00133, SWS_Ad_00137, SWS_Ad_00152, SWS_Ad_00154, SWS_Ad_00294, SWS_Ad_00295, SWS_Ad_00297, SWS_Ad_00299, SWS_Ad_00300, SWS_Ad_00301, SWS_Ad_00302</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00413</td>
<td>An index-based accessing of the instances of BSW modules shall be done</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00414</td>
<td>Init functions shall have a pointer to a configuration structure as single parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00054</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00416</td>
<td>The sequence of modules to be initialized shall be configurable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00417</td>
<td>Software which is not part of the SW-C shall report error events only after the DEM is fully operational.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00423</td>
<td>BSW modules with</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWS_Ad_00460</td>
<td></td>
</tr>
<tr>
<td>Requirement ID</td>
<td>Description</td>
<td>Compliance ID</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>SRS_BSW_00424</td>
<td>BSW module main processing functions shall not be allowed to enter a wait state</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_BSW_00425</td>
<td>The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_BSW_00426</td>
<td>BSW Modules shall ensure data consistency of data which is shared between BSW modules</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_BSW_00427</td>
<td>ISR functions shall be defined and documented in the BSW module description template</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_BSW_00428</td>
<td>A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_BSW_00429</td>
<td>Access to OS is restricted</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_BSW_00432</td>
<td>Modules should have separate main processing functions for read/receive and write/transmit data path</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_BSW_00433</td>
<td>Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_00157</td>
<td>All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers</td>
<td>SWS_Adc_00057, SWS_Adc_00058, SWS_Adc_00082, SWS_Adc_00083, SWS_Adc_00104</td>
</tr>
<tr>
<td>SRS_SPAL_12056</td>
<td>All driver modules shall allow the static configuration of notification mechanism</td>
<td>SWS_Adc_00080, SWS_Adc_00085, SWS_Adc_00084, SWS_Adc_00084</td>
</tr>
<tr>
<td>SRS_SPAL_12057</td>
<td>All driver modules shall implement an interface for initialization</td>
<td>SWS_Adc_00054</td>
</tr>
<tr>
<td>SRS_SPAL_12063</td>
<td>All driver modules shall only support raw value mode</td>
<td>SWS_Adc_00113</td>
</tr>
<tr>
<td>SRS_SPAL_12064</td>
<td>All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12067</td>
<td>All driver modules shall set</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>Specification</td>
<td>Description</td>
<td>Reference</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>their wake-up conditions depending on the selected operation mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRS_SPAL_12068</td>
<td>The modules of the MCAL shall be initialized in a defined sequence</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12069</td>
<td>All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12077</td>
<td>All drivers shall provide a non-blocking implementation</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12078</td>
<td>The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12092</td>
<td>The driver's API shall be accessed by its handler or manager</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12125</td>
<td>All driver modules shall only initialize the configured resources</td>
<td>SWS_Adc_00056</td>
</tr>
<tr>
<td>SRS_SPAL_12129</td>
<td>The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function</td>
<td>SWS_Adc_00078</td>
</tr>
<tr>
<td>SRS_SPAL_12163</td>
<td>All driver modules shall implement an interface for de-initialization</td>
<td>SWS_Adc_00110, SWS_Adc_00111</td>
</tr>
<tr>
<td>SRS_SPAL_12169</td>
<td>All driver modules that provide different operation modes shall provide a service for mode selection</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12265</td>
<td>Configuration data shall be kept constant</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12267</td>
<td>Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>SRS_SPAL_12448</td>
<td>All driver modules shall have a specific behavior after a development error detection</td>
<td>SWS_Adc_000107, SWS_Adc_000125, SWS_Adc_000126, SWS_Adc_000128, SWS_Adc_000129, SWS_Adc_000131, SWS_Adc_000133, SWS_Adc_000136, SWS_Adc_000137, SWS_Adc_000152, SWS_Adc_000154, SWS_Adc_000164, SWS_Adc_000165, SWS_AdS_000225, SWS_AdS_000241</td>
</tr>
<tr>
<td>SRS_SPAL_12461</td>
<td>Specific rules regarding initialization of controller registers shall apply to all driver implementations</td>
<td>SWS_Adc_000054, SWS_Adc_000246, SWS_Adc_000247, SWS_Adc_000248, SWS_Adc_000249, SWS_Adc_000250</td>
</tr>
<tr>
<td>SRS_SPAL_12463</td>
<td>The register initialization</td>
<td>SWS_Adc_00460</td>
</tr>
<tr>
<td>settings shall be combined and forwarded</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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).

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

Table 2: 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.] ()
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” 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

---

1 On some microcontroller also called „auto-scan mode“.

Document ID 010: AUTOSAR_SWS_ADCDriver

- AUTOSAR confidential -
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.
The ADC module’s priority mechanism shall allow aborting and restarting of channel group conversions. (SRS_Adc_12820)

The ADC module’s priority mechanism shall allow suspending and resuming of channel group conversions. ()

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.

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.

In the ADC module’s priority mechanism the lowest priority is 0. ()

The ADC module’s priority mechanism shall allow the configuration of 256 priority levels (0...255). (SRS_Adc_12820)

The ADC module shall support the static configuration option to disable the priority mechanism. ()

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)

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_Ad_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_Ad_00332] [If the priority mechanism is active, the ADC module shall support a queuing of conversion requests. The conversion requests 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_Ad_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_Ad_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_Ad_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_Ad_00336] ['Enable hardware trigger requests', generated with API function Adc_EnableHardwareTrigger, shall not be stored in any queue.] ()

[SWS_Ad_00337] [The hardware prioritization mechanism shall be used in case of hardware triggered conversion requests.] ()
[SWS_Ad_00338] [The ADC module shall not store additional software conversion requests for the same group, whose group status is not equal to ADC_IDLE. ] ()
[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_DelInit, 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 implicitly 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

<table>
<thead>
<tr>
<th>Group</th>
<th>ADC_GROUP_DEFINITION</th>
<th>ADC_RESULT_POINTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>group G1:</td>
<td>CH0, CH1</td>
<td>G1_ResultPtr</td>
</tr>
<tr>
<td>group G2:</td>
<td>CH2</td>
<td>G2_ResultPtr</td>
</tr>
<tr>
<td>group G3:</td>
<td>CH3</td>
<td>G3_ResultPtr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>ADC_GROUP_ACCESS_MODE</th>
<th>ADC_STREAMING_NUM_SAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>group G1:</td>
<td>ADC_ACCESS_MODE_STREAMING</td>
<td>3</td>
</tr>
<tr>
<td>group G2:</td>
<td>ADC_ACCESS_MODE_STREAMING</td>
<td>2</td>
</tr>
<tr>
<td>group G3:</td>
<td>ADC_ACCESS_MODE_SINGLE</td>
<td>1</td>
</tr>
</tbody>
</table>

2. Result Pointer Initialization with Adc_SetupResultBuffer API function

Application

result buffer (required)

 ADC Driver

3. Result access with Adc_GetStreamLastPointer API function

Application

result buffer (required)

ADC Driver

ADC HW

Figure 2: 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
### 4. Result access with Adc_ReadGroup API function

<table>
<thead>
<tr>
<th>Application</th>
<th>ADC Driver</th>
<th>ADC HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>read buffer (required if Adc_ReadGroup is used)</td>
<td>result buffer (required)</td>
<td></td>
</tr>
<tr>
<td>G1_CH1</td>
<td>G1_CH1</td>
<td></td>
</tr>
<tr>
<td>G1_CH0</td>
<td>G1_CH1</td>
<td></td>
</tr>
<tr>
<td>G1_ReadBuffer[2]</td>
<td>G1_CH0</td>
<td></td>
</tr>
<tr>
<td>G2_CH2</td>
<td>G2_CH2</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>G3_CH3</td>
<td></td>
</tr>
<tr>
<td>G2_G3_ReadBuffer[1]</td>
<td>G3_CH3</td>
<td></td>
</tr>
<tr>
<td>G3_ResultsBuffer[1]</td>
<td>G3_ResultsBuffer[1]</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 3: Example for calling Adc_ReadGroup which copies results from Result Buffer to optional Read Buffer](image)

### 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_GetStreamLastPointer. Precisely pointer G1_SamplePtr points, after calling Adc_GetStreamLastPointer, 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.

### Table: Conversion Mode behavior examples

<table>
<thead>
<tr>
<th>Channel Type</th>
<th>Conversion Type</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-channel Group</td>
<td>Continuous</td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Multi-channel Group</td>
<td>One-Shot</td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Single Channel Group</td>
<td>Continuous</td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Single Channel Group</td>
<td>One-Shot</td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Figure 4**: Conversion Mode behavior examples
7.2.2 Requirements

[SWS_Adcd_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_Adcd_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

![ADC State Diagram](image)

The 'concurrent states' ONE-SHOT and CONTINUOUS are configuration options for ADC groups. One ADC group can be only in one of the two states.

Figure 5: 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

The 'concurrent states' SW-TRIGGER and HW-TRIGGER are configuration options for ADC groups. One ADC group can be only in one of the two states.

The 'concurrent states' SINGLE-ACCESS and STREAMING-ACCESS are configuration options for ADC groups. One ADC group can be only in one of the two states.

**Figure 6: State Diagram HW/SW Trigger in One-Shot Group Conversion Mode**
7.3.3 ADC State Diagram for SW Trigger in Continuous Conversion Mode

The 'concurrent states' SINGLE-ACCESS and STREAMING-ACCESS are configuration options for ADC groups. One ADC group can be only in one of the two states.

Figure 7: 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

![State Diagram](image)

**Figure 8: State Diagram On-Shot, SW Trigger, Single Access**
7.3.5 ADC State Diagram for One-Shot Conversion, Hardware Trigger Source, Single Access Mode

ADC group configuration:
- one-shot conversion
- hardware trigger source
- single access

Figure 9: 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

ADC group configuration:
- one-shot conversion
- hardware trigger source
- streaming access linear and circular

Figure 10: State Diagram One-Shot, HW Trigger, Streaming Access
7.3.7 ADC State Diagram for Continuous Conversion Mode, Software Trigger Source, Single Access Mode

ADC group configuration:
- continuous conversion
- software trigger source
- single access

Adc_GetStreamLastPointer, Adc_ReadGroup

Figure 11: 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 12: State Diagram Conversion, SW Trigger, Streaming Access**

ADC group configuration:
- continuous conversion
- software trigger source
- streaming access linear and circular

- ADC_IDLE
- ADC_BUSY
- ADC_STREAM_COMPLETED
- ADC_COMPLETED

Adc_ReadGroup, Adc_GetStreamLastPointer

Adc_StopGroupConversion

Adc_StartGroupConversion

Adc_ReadGroup, Adc_GetStreamLastPointer [linear streaming buffer]

Adc_ReadGroup, Adc_GetStreamLastPointer [circular streaming buffer]

streaming buffer filled completely

streaming buffer filled completely

conversion of all group channels completed [remaining nr of streaming samples >= 1]
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, IoHwAbstraction 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_GetCurrentPowerState 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_PreparePowerState 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_SetPowerState 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 power state if the parameter AdcLowPowerStatesSupport is set to TRUE.

**SWS_Adc_00470** After the initialization 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 synchronous and asynchronous power state transitions, depending on the value of the configuration parameter AdcPowerStateAsynchTransitionMode.

**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 Version check

7.5.1 Background & Rationale

The integration of incompatible files is to be avoided. Minimum implementation is the version check of the header file inside the .c file (version numbers of .c and .h files must be identical).
<table>
<thead>
<tr>
<th>Type of error</th>
<th>Relevance</th>
<th>Related error code</th>
<th>Value [hex]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adc_Init has not been called prior to another function call (see SWS_Adc_00154,</td>
<td>Development</td>
<td>ADC_E_UNINIT</td>
<td>0x0A</td>
</tr>
<tr>
<td>SWS_Adc_00294, SWS_Adc_00295, SWS_Adc_00296, SWS_Adc_00297, SWS_Adc_00298,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWS_Adc_00299, SWS_Adc_00300, SWS_Adc_00301, SWS_Adc_00302, SWS_Adc_00486,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWS_Adc_00491, SWS_Adc_00493, SWS_Adc_00496,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWS_Adc_00486, SWS_Adc_00491, SWS_Adc_00493, SWS_Adc_00496,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_StartGroupConversion was called while another conversion is already</td>
<td>Runtime</td>
<td>ADC_E_BUSY</td>
<td>0x0B</td>
</tr>
<tr>
<td>running or a HW trigger is already enabled or a request is already stored in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the queue (see SWS_Adc_00346, SWS_Adc_00348, ADC350, SWS_Adc_00351, ADC352).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_EnableHardwareTrigger was called while a conversion is ongoing or a HW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trigger is already enabled or the maximum number of HW triggers is already</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enabled (see SWS_Adc_00321, SWS_Adc_00349, SWS_Adc_00353)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_DeInit was called while a conversion is still ongoing (see SWS_Adc_00112).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_StopGroupConversion was called while no conversion was running (see SWS_Adc_00241), Adc_DisableHardwareTrigger was called while group is not enabled (see SWS_Adc_00304)</td>
<td>Runtime</td>
<td>ADC_E_IDLE</td>
<td>0x0C</td>
</tr>
<tr>
<td>Adc_Init has been called while ADC is already initialized (see SWS_Adc_00107)</td>
<td>Development</td>
<td>ADC_E_ALREADY_INITIALIZED</td>
<td>0x0D</td>
</tr>
<tr>
<td>Adc_Init has been called with incorrect configuration parameter (configuration</td>
<td>Development</td>
<td>ADC_E_PARAM_POINTER</td>
<td>0x0E</td>
</tr>
<tr>
<td>pointer is NULL_PTR for post-build configuration or configuration pointer is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not equal NULL_PTR for pre-compile configuration)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_SetupResultBuffer or Adc_GetVersionInfo called with invalid data buffer</td>
<td>Development</td>
<td>ADC_E_PARAM_POINTER</td>
<td>0x14</td>
</tr>
<tr>
<td>pointer,</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3: Error classification

<table>
<thead>
<tr>
<th>Error Description</th>
<th>Module</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL_PTR passed SWS_Ad_00269, SWS_Ad_00458</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalid group ID requested (see SWS_Ad_00125, SWS_Ad_00126, SWS_Ad_00152,</td>
<td>Development</td>
<td>ADC_E_PARAM_GROUP 0x15</td>
</tr>
<tr>
<td>SWS_Ad_00128, SWS_Ad_00129, SWS_Ad_00130, SWS_Ad_00131, SWS_Ad_00225, SWS_Ad_00218)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_EnableHardwareTrigger or Adc_DisableHardwareTrigger called on a group with</td>
<td>Development</td>
<td>ADC_E_WRONG_CONV_MODE 0x16</td>
</tr>
<tr>
<td>conversion mode configured as continuous (see SWS_Ad_00281, SWS_Ad_00282).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_StartGroupConversion or Adc_StopGroupConversion called on a group with trigger</td>
<td>Development</td>
<td>ADC_E_WRONG_TRIGG_SRC 0x17</td>
</tr>
<tr>
<td>source configured as hardware (see SWS_Ad_00133, SWS_Ad_00164).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adc_EnableHardwareTrigger or Adc_DisableHardwareTrigger called on a group with</td>
<td>Development</td>
<td>ADC_E_NOTIF_CAPABILITY 0x18</td>
</tr>
<tr>
<td>trigger source configured as software API (see SWS_Ad_00136, SWS_Ad_00137).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable/disable notification function for a group whose configuration set has no</td>
<td>Development</td>
<td>ADC_E_BUFFER_UNINIT 0x19</td>
</tr>
<tr>
<td>notification available (see SWS_Ad_00165, SWS_Ad_00166).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversion started and result buffer pointer is not initialized (see SWS_Ad_00424,</td>
<td>Development</td>
<td>ADC_E_TRANSITION_NOT_POSSIBLE 0x1C</td>
</tr>
<tr>
<td>SWS_Ad_00425).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One or more ADC group/channel not in IDLE state SWS_Ad_00486</td>
<td>Runtime</td>
<td>ADC_E_NOT_DISENGAGED 0x1A</td>
</tr>
<tr>
<td>Unsupported power state request SWS_Ad_00488, SWS_Ad_00497</td>
<td>Development</td>
<td>ADC_E_POWER_STATE_NOT_SUPPORTED 0x1B</td>
</tr>
<tr>
<td>Requested power state can not be reached directly SWS_Ad_00489, SWS_Ad_00498.</td>
<td>Runtime</td>
<td>ADC_E_TRANSITION_NOT_POSSIBLE 0x1C</td>
</tr>
<tr>
<td>ADC not prepared for target power state SWS_Ad_00490</td>
<td>Development</td>
<td>ADC_E_PERIPHERAL_NOT_PREPARED 0x1D</td>
</tr>
<tr>
<td>--</td>
<td>Production</td>
<td>--</td>
</tr>
</tbody>
</table>

- AUTOSAR confidential -
### 7.6 Error detection

#### 7.6.1 Development Error

<table>
<thead>
<tr>
<th>Function</th>
<th>Criteria of detection</th>
<th>Related error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adc_Init</td>
<td>ADC driver and hardware already initialized.</td>
<td>ADC_E_ALREADY_INITIALIZED</td>
</tr>
<tr>
<td></td>
<td>ADC initialization API called with incorrect configuration pointer</td>
<td>ADC_E_PARAM_POINTER</td>
</tr>
<tr>
<td>Adc_DelInit</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td>Adc_StartGroupConversion</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Function called for a group configured for hardware trigger source.</td>
<td>ADC_E_WRONG_TRIGG_SRC</td>
</tr>
<tr>
<td></td>
<td>Function called while result buffer pointer is not initialized</td>
<td>ADC_E_BUFFER_UNINIT</td>
</tr>
<tr>
<td>Adc_StopGroupConversion</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Function called for a group configured for hardware trigger source.</td>
<td>ADC_E_WRONG_TRIGG_SRC</td>
</tr>
<tr>
<td>Adc_GetGroupStatus</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td></td>
</tr>
<tr>
<td>Adc_ReadGroup</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td></td>
</tr>
<tr>
<td>Adc_EnableHardwareTrigger</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Function called for a group configured for software API trigger source.</td>
<td>ADC_E_WRONG_TRIGG_SRC</td>
</tr>
<tr>
<td></td>
<td>Function called for a group configured for Continuous conversion mode.</td>
<td>ADC_E_WRONG_CONV_MODE</td>
</tr>
<tr>
<td></td>
<td>Function called while result buffer pointer is not initialized</td>
<td>ADC_E_BUFFER_UNINIT</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Error Code</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Adc_DisableHardwareTrigger</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td>ADC_E_PARAM_GROUP</td>
</tr>
<tr>
<td></td>
<td>Function called for a group configured for software API trigger source.</td>
<td>ADC_E_WRONG_TRIGG_SRC</td>
</tr>
<tr>
<td></td>
<td>Function called for a group configured for Continuous conversion mode.</td>
<td>ADC_E_WRONG_CONV_MODE</td>
</tr>
<tr>
<td>Adc_EnableGroupNotification</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td>ADC_E_PARAM_GROUP</td>
</tr>
<tr>
<td></td>
<td>Function called and notification function pointer is NULL.</td>
<td>ADC_E_NOTIF_CAPABILITY</td>
</tr>
<tr>
<td>Adc_DisableGroupNotification</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td>ADC_E_PARAM_GROUP</td>
</tr>
<tr>
<td></td>
<td>Function called and notification function pointer is NULL.</td>
<td>ADC_E_NOTIF_CAPABILITY</td>
</tr>
<tr>
<td>Adc_SetupResultBuffer</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td>ADC_E_PARAM_GROUP</td>
</tr>
<tr>
<td></td>
<td>Function called and DataBufferPtr is NULL_PTR.</td>
<td>ADC_E_PARAM_POINTER</td>
</tr>
<tr>
<td>Adc_GetStreamLastPointer</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Function called with non existing group.</td>
<td>ADC_E_PARAM_GROUP</td>
</tr>
<tr>
<td>Adc_GetVersionInfo</td>
<td>Function called with NULL pointer.</td>
<td>ADC_E_PARAM_POINTER</td>
</tr>
<tr>
<td>Adc_SetPowerState</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Unsupported power state request</td>
<td>ADC_E_POWER_STATE_NOT_SUPPORTED</td>
</tr>
<tr>
<td></td>
<td>ADC not prepared for target power state</td>
<td>ADC_E_PERIPHERAL_NOT_PREPARED</td>
</tr>
<tr>
<td>Adc_GetCurrentPowerState</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td>Adc_GetTargetPowerState</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td>Adc_PreparePowerState</td>
<td>Function called prior to initialization.</td>
<td>ADC_E_UNINIT</td>
</tr>
<tr>
<td></td>
<td>Unsupported power state request</td>
<td>ADC_E_POWER_STATE_NOT_SUPPORTED</td>
</tr>
</tbody>
</table>

Table 4: Error detection – Development Error
### 7.6.2 Runtime Error

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Exception Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adc_DeInit</td>
<td>Function called while conversion is running.</td>
<td>ADC_E_BUSY</td>
</tr>
<tr>
<td>Adc_StartGroupConversion</td>
<td>Function called while any group is not in state ADC_IDLE.</td>
<td>ADC_E_BUSY</td>
</tr>
<tr>
<td></td>
<td>Function called while conversion request already stored in queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Function called while conversion of same group is already running.</td>
<td></td>
</tr>
<tr>
<td>Adc_EnableHardwareTrigger</td>
<td>Function called while HW trigger for the group is already enabled.</td>
<td>ADC_E_BUSY</td>
</tr>
<tr>
<td></td>
<td>Function called while maximum number of available hardware triggers is already enabled.</td>
<td></td>
</tr>
<tr>
<td>Adc_SetupResultBuffer</td>
<td>Function called while any group is not in state ADC_IDLE.</td>
<td>ADC_E_BUSY</td>
</tr>
<tr>
<td>Adc_StopGroupConversion</td>
<td>Function called while group is in state ADC_IDLE.</td>
<td>ADC_E_IDLE</td>
</tr>
<tr>
<td>Adc_ReadGroup</td>
<td>Function called while group status is ADC_IDLE.</td>
<td>ADC_E_IDLE</td>
</tr>
<tr>
<td>Adc_DisableHardwareTrigger</td>
<td>Function called for a non enabled group.</td>
<td>ADC_E_IDLE</td>
</tr>
<tr>
<td>Adc_GetStreamLastPointer</td>
<td>Function called while group status is ADC_IDLE.</td>
<td>ADC_E_IDLE</td>
</tr>
<tr>
<td>Adc_SetPowerState</td>
<td>One or more ADC group/channel not in IDLE state</td>
<td>ADC_E_NOT_DISENGAGED</td>
</tr>
<tr>
<td>Adc_SetPowerState</td>
<td>Requested power state can not be reached directly</td>
<td>ADC_E_TRANSITION_NOT_POSSIBLE</td>
</tr>
<tr>
<td>Adc_PreparePowerState</td>
<td>Requested power state can not be reached directly</td>
<td>ADC_E_TRANSITION_NOT_POSSIBLE</td>
</tr>
</tbody>
</table>

Table 5: Error detection – Runtime Error

### 7.6.3 Transient Faults

Table 6: Error detection – Transient Faults
8 API specification

8.1 Imported types

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Imported Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Std_Types</td>
<td>Std_ReturnType</td>
</tr>
<tr>
<td></td>
<td>Std_VersionInfoType</td>
</tr>
</tbody>
</table>

8.2 Type definitions

8.2.1 Adc_ConfigType

| Name:         | Adc_ConfigType         |
| Type:         | Structure              |
| Range:        | --                     |
| Description:  | Implementation specific configuration data structure. |

8.2.2 Adc_ChannelType

| Name:         | Adc_ChannelType        |
| Type:         | uint                   |
| Range:        | --                     |
| Description:  | Numeric ID of an ADC channel. |

8.2.3 Adc_GroupType

| Name:         | Adc_GroupType          |
| Type:         | uint                   |
| Range:        | --                     |
| Description:  | Numeric ID of an ADC channel group. |

8.2.4 Adc_ValueGroupType

| Name:         | Adc_ValueGroupType     |
| Type:         | int                    |
| Range:        | --                     |
| Description:  | Type for reading the converted values of a channel group (raw, without further scaling, alignment according precompile switch ADC_RESULT_ALIGNMENT). |
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 \( m \times n \) 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]**

<table>
<thead>
<tr>
<th>Name:</th>
<th>Adc_PrescaleType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>uint</td>
</tr>
<tr>
<td>Range:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>Type of clock prescaler factor. (This is not an API type).</td>
</tr>
</tbody>
</table>

8.2.6 Adc_ConversionTimeType

**[SWS_Adc_00510]**

<table>
<thead>
<tr>
<th>Name:</th>
<th>Adc_ConversionTimeType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>uint</td>
</tr>
<tr>
<td>Range:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>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).</td>
</tr>
</tbody>
</table>
8.2.7 Adc_SamplingTimeType

[SWS_Adc_00511]

<table>
<thead>
<tr>
<th>Name:</th>
<th>Adc_SamplingTimeType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>uint</td>
</tr>
<tr>
<td>Range:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>The range of this type is µC specific and has to be described by the supplier.</td>
</tr>
</tbody>
</table>

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

8.2.8 Adc_ResolutionType

[SWS_Adc_00512]

<table>
<thead>
<tr>
<th>Name:</th>
<th>Adc_ResolutionType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>uint8</td>
</tr>
<tr>
<td>Range:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>The range of this type is µC specific and has to be described by the supplier.</td>
</tr>
</tbody>
</table>

Type of channel resolution in number of bits.
(This is not an API type).

8.2.9 Adc_StatusType

[SWS_Adc_00513]

<table>
<thead>
<tr>
<th>Name:</th>
<th>Adc_StatusType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Range:</td>
<td>ADC_IDLE</td>
</tr>
<tr>
<td>Description:</td>
<td>The conversion of the specified group has not been started.</td>
</tr>
<tr>
<td></td>
<td>- No result is available.</td>
</tr>
<tr>
<td></td>
<td>ADC_BUSY</td>
</tr>
<tr>
<td>Description:</td>
<td>- The conversion of the specified group has been started and is still going on.</td>
</tr>
<tr>
<td></td>
<td>- So far no result is available.</td>
</tr>
<tr>
<td></td>
<td>ADC_COMPLETED</td>
</tr>
<tr>
<td>Description:</td>
<td>- A conversion round (which is not the final one) of the specified group has been finished.</td>
</tr>
<tr>
<td></td>
<td>- A result is available for all channels of the group.</td>
</tr>
<tr>
<td></td>
<td>ADC_STREAM_COMPLETED</td>
</tr>
<tr>
<td>Description:</td>
<td>- The result buffer is completely filled</td>
</tr>
<tr>
<td></td>
<td>- For each channel of the selected group the number of samples to be acquired is available</td>
</tr>
</tbody>
</table>

Current status of the conversion of the requested ADC Channel group.

8.2.10 Adc_TriggerSourceType

[SWS_Adc_00514]

<table>
<thead>
<tr>
<th>Name:</th>
<th>Adc_TriggerSourceType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Range:</td>
<td>ADC_TRIGG_SRC_SW</td>
</tr>
<tr>
<td>Description:</td>
<td>Group is triggered by a software API call.</td>
</tr>
<tr>
<td></td>
<td>ADC_TRIGG_SRC_HW</td>
</tr>
<tr>
<td>Description:</td>
<td>Group is triggered by a hardware event.</td>
</tr>
</tbody>
</table>

Type for configuring the trigger source for an ADC Channel group.
8.2.11 Adc_GroupConvModeType

[SWS_Ad_00515]

<table>
<thead>
<tr>
<th>Name: Adc_GroupConvModeType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Enumeration</td>
</tr>
</tbody>
</table>

**Range:**

<table>
<thead>
<tr>
<th>ADC_CONV_MODE_ONESHOT</th>
</tr>
</thead>
</table>

- 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).

<table>
<thead>
<tr>
<th>ADC_CONV_MODE_CONTINUOUS</th>
</tr>
</thead>
</table>

- 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.

8.2.12 Adc_GroupPriorityType

[SWS_Ad_00516]

<table>
<thead>
<tr>
<th>Name: Adc_GroupPriorityType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: uint8</td>
</tr>
</tbody>
</table>

**Range:**

<table>
<thead>
<tr>
<th>0..255</th>
</tr>
</thead>
</table>

**Description:** Priority level of the channel. Lowest priority is 0.

8.2.13 Adc_GroupDefType

[SWS_Ad_00517]

<table>
<thead>
<tr>
<th>Name: Adc_GroupDefType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: implementation specific</td>
</tr>
</tbody>
</table>

**Description:** Type for assignment of channels to a channel group (this is not an API type).

8.2.14 Adc_StreamNumSampleType

[SWS_Ad_00518]

<table>
<thead>
<tr>
<th>Name: Adc_StreamNumSampleType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: uint</td>
</tr>
</tbody>
</table>

**Range:**

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
</tr>
</tbody>
</table>

- 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).
8.2.15 Adc_StreamBufferModeType

[SWS_AdC_00519] [ ]

| Name: | Adc_StreamBufferModeType |
| Type: | Enumeration |
| Range: | ADC_STREAM_BUFFER_LINEAR | The ADC Driver stops the conversion as soon as the stream buffer is full (number of samples reached). |
| | 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. |

Description: Type for configuring the streaming access mode buffer type.

8.2.16 Adc_GroupAccessModeType

[SWS_AdC_00528] [ ]

| Name: | Adc_GroupAccessModeType |
| Type: | Enumeration |
| Range: | ADC_ACCESS_MODE_SINGLE | Single value access mode. |
| | ADC_ACCESS_MODE_STREAMING | Streaming access mode. |

Description: Type for configuring the access mode to group conversion results.

8.2.17 Adc_HwTriggerSignalType

[SWS_AdC_00520] [ ]

| Name: | Adc_HwTriggerSignalType |
| Type: | Enumeration |
| Range: | ADC_HW_TRIG_RISING_EDGE | React on the rising edge 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_BOTH_EDGES | 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).

8.2.18 Adc_HwTriggerTimerType

[SWS_AdC_00521] [ ]

| Name: | Adc_HwTriggerTimerType |
| Type: | 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).
### 8.2.19 Adc_PriorityImplementationType

<table>
<thead>
<tr>
<th>Name</th>
<th>Adc_PriorityImplementationType</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong></td>
<td>Enumeration</td>
</tr>
<tr>
<td><strong>Range:</strong></td>
<td></td>
</tr>
<tr>
<td>ADC_PRIORITY_NONE</td>
<td>priority mechanism is not available</td>
</tr>
<tr>
<td>ADC_PRIORITY_HW</td>
<td>Hardware priority mechanism is available only</td>
</tr>
<tr>
<td>ADC_PRIORITY_HW_SW</td>
<td>Hardware and software priority mechanism is available</td>
</tr>
</tbody>
</table>

**Description:** Type for configuring the prioritization mechanism.

### 8.2.20 Adc_GroupReplacementType

<table>
<thead>
<tr>
<th>Name</th>
<th>Adc_GroupReplacementType</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong></td>
<td>Enumeration</td>
</tr>
<tr>
<td><strong>Range:</strong></td>
<td></td>
</tr>
<tr>
<td>ADC_GROUP_REPL_ABORT_RESTART</td>
<td>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.</td>
</tr>
<tr>
<td>ADC_GROUP_REPL_SUSPEND_RESUME</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Description:** Replacement mechanism, which is used on ADC group level, if a group conversion is interrupted by a group which has a higher priority.

### 8.2.21 Adc_ChannelRangeSelectType

<table>
<thead>
<tr>
<th>Name</th>
<th>Adc_ChannelRangeSelectType</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong></td>
<td>Enumeration</td>
</tr>
<tr>
<td><strong>Range:</strong></td>
<td></td>
</tr>
<tr>
<td>ADC_RANGE_UNDER_LOW</td>
<td>Range below low limit - low limit value included</td>
</tr>
<tr>
<td>ADC_RANGE_BETWEEN</td>
<td>Range between low limit and high limit - high limit value included</td>
</tr>
<tr>
<td>ADC_RANGE_OVER_HIGH</td>
<td>Range above high limit</td>
</tr>
<tr>
<td>ADC_RANGE_ALWAYS</td>
<td>Complete range - independent from channel limit settings</td>
</tr>
</tbody>
</table>

**Description:** Type for configuring the channel range selection mechanism.
8.2.22 Adc_ResultAlignmentType

[SWS_Adc_00525]

Name: Adc_ResultAlignmentType
Type: Enumeration
Range: ADC_ALIGN_LEFT -- left alignment
       ADC_ALIGN_RIGHT -- right alignment

Description: Type for alignment of ADC raw results in ADC result buffer (left/right alignment).

8.2.23 Adc_PowerStateType

[SWS_Adc_00526]

Name: Adc_PowerStateType
Type: 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.

8.2.24 Adc_PowerStateRequestResultType

[SWS_Adc_00527]

Name: Adc_PowerStateRequestResultType
Type: 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.

8.3 Function definitions

8.3.1 Adc_Init
Service name: Adc_Init

Syntax:

```c
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.

[SWS_Adc_00365]

[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
ConfigPtr.] (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)
The function Adc_Init shall disable the notifications and hardware trigger capability (if statically configured as active). (SRS_Adcdm_12318)

The function Adc_Init shall set all groups to ADC_IDLE state. ()

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

The function Adc_SetupResultBuffer shall initialize the result buffer pointer of the selected group with the address value passed as parameter. ()

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. ()

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. ] ()

[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 called while 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] [ ]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Adc_StartGroupConversion</th>
</tr>
</thead>
</table>
| 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. |
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)

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.

(SRS_Adc_12364, SRS_Adc_12318)

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)

The ADC module’s environment shall only call Adc_StartGroupConversion for groups configured with software trigger source.

(SRS_Adc_12817, SRS_Adc_12364)

The function Adc_StartGroupConversion shall be pre-compile time configurable On/Off by the configuration parameter AdcEnableStartStopGroupApi.

(SRS_BSW_00171)

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)

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)
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

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

If the priority mechanism 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.} ()

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

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.} ()

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_Adcd_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_Adcd_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_Adcd_00424] If development error detection for the ADC module is enabled: when called prior to initializing the result buffer pointer with function Adc_SetupResultBuffer, the function Adc_StartGroupConversion shall raise development error ADC_E_BUFFER_UNINIT.

8.3.5 Adc_StopGroupConversion

[SWS_Adcd_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. |

...
| () |

[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) |

Text:

*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.
<table>
<thead>
<tr>
<th>Parameters (inout)</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters (out)</td>
<td>DataBufferPtr ADC results of all channels of the selected group are stored in the data buffer addressed with the pointer.</td>
</tr>
<tr>
<td>Return value</td>
<td>Std_ReturnType E_OK: results are available and written to the data buffer E_NOT_OK: no results are available or development error occurred</td>
</tr>
<tr>
<td>Description</td>
<td>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).</td>
</tr>
</tbody>
</table>

[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 to 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.] ()
8.3.7 Adc_EnableHardwareTrigger

[SWS_Adcd_00370] |  
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service name:</strong></td>
</tr>
</tbody>
</table>
| **Syntax:** | void Adc_EnableHardwareTrigger( 
| | Adc_GroupType Group |
| **Service ID[hex]:** | 0x05 |
| **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:** | Enables 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_Adcd_00144] |  
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[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)</td>
</tr>
</tbody>
</table>

[SWS_Adcd_00432] |  
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[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.] ()</td>
</tr>
</tbody>
</table>

[SWS_Adcd_00273] |  
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[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)</td>
</tr>
</tbody>
</table>
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)
8.3.8 Adc_DisableHardwareTrigger

[SWS_Adcd_00425] [If development error detection for the ADC module is enabled: when called prior to initializing the result buffer pointer with function Adc_SetupResultBuffer, the function Adc_EnableHardwareTrigger shall raise development error ADC_E_BUFFER_UNINIT.] ()

[SWS_Adcd_00371]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Adc_DisableHardwareTrigger</th>
</tr>
</thead>
</table>
| Syntax:      | void Adc_DisableHardwareTrigger(
|              |     Adc_GroupType Group
|              | )                          |
| Service ID[hex]: | 0x06                      |
| 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: | Disables the hardware trigger for the requested ADC Channel group. |

[SWS_Adcd_00116] [The function Adc_DisableHardwareTrigger shall disable the hardware trigger for the requested ADC Channel group.] (SRS_Adcd_12823)

[SWS_Adcd_00429] [The function Adc_DisableHardwareTrigger shall remove any queued start/restart request for the requested ADC Channel group if queuing is enabled.] ()

[SWS_Adcd_00145] [The function Adc_DisableHardwareTrigger shall abort an ongoing conversion, if applicable (supported by the hardware).] (SRS_Adcd_12364)

[SWS_Adcd_00157] [If enabled, the function Adc_DisableHardwareTrigger shall disable the notification mechanism for the requested group.] (SRS_Adcd_12317, SRS_Adcd_12318, SRS_Adcd_12364)

[SWS_Adcd_00361] [The function Adc_DisableHardwareTrigger shall set the group status to state ADC_IDLE.] ()

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

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)

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)

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.

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.]

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

Service name: Adc_EnableGroupNotification
Syntax:
void Adc_EnableGroupNotification(
    Adc_GroupType Group
)
### Service ID[hex]: 0x07

**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:** Enables the notification mechanism for the requested ADC Channel group.

---

[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_00373] [Service name: Adc_DisableGroupNotification  
Syntax:  
```c
void Adc_DisableGroupNotification(
  Adc_GroupType Group
)
```  
**Service ID[hex]: 0x08**  
**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: Disables the notification mechanism for the requested ADC Channel group.

[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] [ ]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Adc_GetGroupStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>Adc_StatusType Adc_GetGroupStatus( Adc_GroupType Group )</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x09</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>Group Numeric ID of requested ADC Channel group.</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>Adc_StatusType Conversion status for the requested group.</td>
</tr>
<tr>
<td>Description:</td>
<td>Returns the conversion status of the requested ADC Channel group.</td>
</tr>
</tbody>
</table>
[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)
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).]

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.]

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.
The function Adc_GetGroupStatus shall provide atomic access to the status data by the use of atomic instructions. [SRS_Adc_12291]

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. [SRS_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]

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]

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. [SRS_Adc_00436]

8.3.12 Adc_GetStreamLastPointer

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). [SRS_Adc_00375]
[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)
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.

If development error detection for the ADC module is 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 (PtrToSamplePtr), to NULL and return 0 without any further action.

(SRS_BSW_00386)

If development error detection for the ADC module is 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

<table>
<thead>
<tr>
<th>SWS_Ad_00376</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service name:</strong></td>
<td>Adc_GetVersionInfo</td>
</tr>
</tbody>
</table>
| **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. |

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

<table>
<thead>
<tr>
<th>SWS_Ad_00475</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service name:</strong></td>
<td>Adc_SetPowerState</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td>Std_ReturnType Adc_SetPowerState(</td>
</tr>
</tbody>
</table>
### Adc_PowerStateRequest

<table>
<thead>
<tr>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>This API configures the Adc module so that it enters the already prepared power state, chosen between a predefined set of configured ones.</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Parameters (in):</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>If the API returns E_OK:</td>
</tr>
</tbody>
</table>

- **ADC_SERVICE_ACCEPTED**: Power state change executed.
- **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.

<table>
<thead>
<tr>
<th>Parameters (out):</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>If the API returns E_NOT_OK:</td>
</tr>
</tbody>
</table>

- **E_OK**: Power Mode changed
- **E_NOT_OK**: request rejected

<table>
<thead>
<tr>
<th>Return value:</th>
<th>E_OK: Power Mode changed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E_NOT_OK: request rejected</td>
</tr>
</tbody>
</table>

### Description

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_Adcd_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_Adcd_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_Adcd_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_Adcd_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_Adcd_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]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Adc_GetCurrentPowerState</th>
</tr>
</thead>
</table>
| 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                     |
### 8.3.16 Adc_GetTargetPowerState

**Service name:** Adc_GetTargetPowerState  
**Syntax:**  
```c
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.

---

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.

---

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_Adcc_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_Adcc_00478]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Adc_PreparePowerState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>Std_ReturnType Adc_PreparePowerState(</td>
</tr>
<tr>
<td></td>
<td>Adc_PowerStateType PowerState,</td>
</tr>
<tr>
<td></td>
<td>Adc_PowerStateRequestResultType* Result</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x13</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Non Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>PowerState The target power state intended to be attained</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out): Result</td>
<td>If the API returns E_OK:</td>
</tr>
<tr>
<td></td>
<td>ADC_SERVICE_ACCEPTED: ADC Module power state</td>
</tr>
<tr>
<td></td>
<td>preparation was started.</td>
</tr>
<tr>
<td></td>
<td>If the API returns E_NOT_OK:</td>
</tr>
<tr>
<td></td>
<td>ADC_NOT_INIT: ADC Module not initialized.</td>
</tr>
<tr>
<td></td>
<td>ADC_SEQUENCE_ERROR: wrong API call sequence (Current</td>
</tr>
<tr>
<td></td>
<td>Power State = Target Power State).</td>
</tr>
<tr>
<td></td>
<td>ADC_POWER_STATE_NOT_SUPP: ADC Module does not</td>
</tr>
<tr>
<td></td>
<td>support the requested power state.</td>
</tr>
<tr>
<td></td>
<td>ADC_TRANS_NOT_POSSIBLE: ADC Module cannot transition</td>
</tr>
<tr>
<td></td>
<td>directly from the current power state to the requested power state</td>
</tr>
<tr>
<td></td>
<td>or the HW peripheral is still busy.</td>
</tr>
<tr>
<td>Return value:</td>
<td>Std_ReturnType</td>
</tr>
<tr>
<td></td>
<td>E_OK: Preparation process started</td>
</tr>
<tr>
<td></td>
<td>E_NOT_OK: Service is rejected</td>
</tr>
<tr>
<td>Description:</td>
<td>This API starts the needed process to allow the ADC HW module to enter the</td>
</tr>
<tr>
<td></td>
<td>requested power state.</td>
</tr>
</tbody>
</table>

[SWS_Adcc_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.

] ()
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.

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.

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.

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 Call-back 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

<table>
<thead>
<tr>
<th>Service name</th>
<th>Adc_Main_PowerTransitionManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>void Adc_Main_PowerTransitionManager(</td>
</tr>
<tr>
<td></td>
<td>void</td>
</tr>
<tr>
<td>Service ID[hex]</td>
<td>0x14</td>
</tr>
<tr>
<td>Description</td>
<td>This API is cyclically called and supervises the power state transitions, checking for the readiness of the module and issuing the callbacks IoHwAb_Adc_NotifyReadyForPowerState&lt;Mode&gt; (see</td>
</tr>
</tbody>
</table>
This API executes any non-immediate action needed to finalize a power state transition requested by Adc_PreparePowerState().

The rate of scheduling shall be defined by Adc_MainSchedulePeriod and shall be variable, as the function only needs to be called if a transition has been requested.

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

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]

<table>
<thead>
<tr>
<th>API function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det_ReportRuntimeError</td>
<td>Service to report runtime errors. If a callout has been configured then this callout shall be called.</td>
</tr>
</tbody>
</table>

8.6.2 Optional Interfaces

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

[SWS_AdC_00377]

<table>
<thead>
<tr>
<th>API function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det_ReportError</td>
<td>Service to report development errors.</td>
</tr>
</tbody>
</table>

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]
### 8.6.3.1 IoHwAb_AdcNotification<#groupID>

<table>
<thead>
<tr>
<th><strong>Service name:</strong></th>
<th>IoHwAb_AdcNotification&lt;#groupID&gt;</th>
</tr>
</thead>
</table>
| **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>. |

**[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. |

**[SWS_Adc_00083]**

| When the notification mechanism is disabled, the ADC module shall send no notification. |

**[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. |

**[SWS_Adc_00080]**

| If for a notification call-back the NULL pointer is configured, no call-back shall be executed. |

**[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. |

### 8.6.3.2 IoHwAb_Adc_NotifyReadyForPowerState<#Mode>

<table>
<thead>
<tr>
<th><strong>Service name:</strong></th>
<th>IoHwAb_Adc_NotifyReadyForPowerState&lt;#Mode&gt;</th>
</tr>
</thead>
</table>
| **Syntax:**       | void IoHwAb_Adc_NotifyReadyForPowerState<#Mode>(
|                   | void )                           |

**[SWS_Adc_00480]**
### Specification of ADC Driver

<table>
<thead>
<tr>
<th>Sync/Async:</th>
<th>Synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reentrancy:</td>
<td>Non Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>The API shall be invoked by the ADC Driver when the requested power state preparation for mode <code>&lt;#Mode&gt;</code> is completed.</td>
</tr>
</tbody>
</table>

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

![Diagram of Initialization of the ADC Driver]

Status: proposed by TO as per SWS Adc 2.0.1
Description: ADC hardware and driver initialization
Comments: 

Figure 13: Initialization of the ADC Driver

9.2 De-Initialization of the ADC Driver

![Diagram of De-Initialization of the ADC Driver]

Status: Proposed by TO as per SWS ADC 2.0.1
Description: ADC driver and hardware deinitialization.
Comments: It's the responsibility of the HW design that this state doesn't lead to undefined activities in the uC
a) all notifications are disabled
b) all used interrupts are disabled

Figure 14: De-Initialization of the ADC Driver

9.3 Software triggered One-Shot conversion without notification
**Figure 15: Software triggered one-shot conversion without notification**

Status: proposed by S. Baenkel for SWS ADC 3.0.0

Description: Software triggered One-Shot Conversion without Notification

Configuration:
- Group 1: Channel 1, Channel 2
- One-Shot Conversion Mode
- Single Access Mode
9.4 Software triggered continuous conversion with notification

Status: proposed by S. Barnikol as per SWS ADC 3.0.0
Description: Software triggered continuous conversion with notification

Figure 16: Software triggered continuous conversion with notification
9.5 Hardware triggered One-Shot conversion with notification

Figure 17: Hardware triggered one-shot conversion with notification
9.6 HW Trigger - One-Shot conversion - Linear Streaming

Figure 18: Hardware triggered one-shot conversion – linear streaming
9.7 No Priority Mechanism – No Queuing

Note: Det_ReportError(ADC_E_BUSY) needs to be exchanged in this diagram with Det_ReportRuntimeError(ADC_E_BUSY)

Figure 19: No priority mechanism – no queuing
9.8 No Priority Mechanism – SW Queuing

Figure 20: No priority mechanism – software queuing
9.9 HW_SW Priority Mechanism – SW Queuing

Status proposed by S.Bamikel for ADC SWS V3.0.0
Description: SW queue, priority mechanism HW_SW
Configuration: - priority mechanism : HW_SW
- SW conversion requests
- priorities: Group1 > Group2
- queuing in SW queue
- one-shot conversion mode

Figure 20: Hardware/software priority mechanism – SW queuing
9.10 HW Priority Mechanism – HW Queuing

Status: proposed by S.Barnikel for SWS ADC 3.0.0
Description: HW queuing of aborted SW conversion
Configuration: - priority mechanism: HW
- priorities: Group1 > Group2
- queuing in HW queue
- SW Group2 conversion aborted / suspended from HW Group1 conversion
- SW Group2 conversion restarted / resumed after HW Group1 conversion completed

Figure 22: Hardware priority mechanism – HW queuing
9.11 HW_Sw Priority Mechanism – HW/SW Queuing

Figure 23: 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 10.2.3 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 Configuration and configuration parameters

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

10.2.1 Adc

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>Module Name</th>
<th>Module Description</th>
<th>Post-Build Variant Support</th>
<th>Supported Config Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECUC_Adc_00462</td>
<td>Adc</td>
<td>Configuration of the Adc (Analog Digital Conversion) module.</td>
<td>true</td>
<td>VARIANT-POST-BUILD, VARIANT-PRE-COMPILE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Included Containers</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdcConfigSet</td>
<td>1</td>
<td>This container contains the configuration parameters and sub containers of the AUTOSAR Adc module.</td>
</tr>
<tr>
<td>AdcGeneral</td>
<td>1</td>
<td>General configuration (parameters) of the ADC Driver software module.</td>
</tr>
<tr>
<td>AdcPublishedInformation</td>
<td>1</td>
<td>Additional published parameters not covered by &quot;Common&quot; Published Information. Note that these parameters have &quot;PUBLISHED-INFORMATION&quot; configuration class setting, since they are published information.</td>
</tr>
</tbody>
</table>
10.2.2 AdcGeneral

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00027</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Name</td>
<td>AdcGeneral</td>
</tr>
<tr>
<td>Description</td>
<td>General configuration (parameters) of the ADC Driver software module.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00405</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AdcDevErrorDetect</td>
</tr>
<tr>
<td>Parent Container</td>
<td>AdcGeneral</td>
</tr>
</tbody>
</table>
| 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 |

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00452</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AdcEnableLimitCheck</td>
</tr>
<tr>
<td>Parent Container</td>
<td>AdcGeneral</td>
</tr>
<tr>
<td>Description</td>
<td>Enables or disables limit checking feature in the ADC driver.</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucBooleanParamDef</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>false</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td>Pre-compile time: X All Variants</td>
</tr>
<tr>
<td></td>
<td>Link time: --</td>
</tr>
<tr>
<td></td>
<td>Post-build time: --</td>
</tr>
<tr>
<td>Scope / Dependency</td>
<td>scope: local</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00391</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AdcEnableQueuing</td>
</tr>
</tbody>
</table>
### Specification of ADC Driver

#### AUTOSAR CP Release 4.3.1

<table>
<thead>
<tr>
<th>Parent Container</th>
<th>AdcGeneral</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcucBooleanParamDef</td>
</tr>
<tr>
<td><strong>Default value</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Post-Build Variant Value</strong></td>
<td>false</td>
</tr>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td>Pre-compile time</td>
</tr>
<tr>
<td></td>
<td>Link time</td>
</tr>
<tr>
<td></td>
<td>Post-build time</td>
</tr>
<tr>
<td><strong>Scope / Dependency</strong></td>
<td>scope: local</td>
</tr>
</tbody>
</table>

#### SWS Item

<table>
<thead>
<tr>
<th>Name</th>
<th>AdcEnableStartStopGroupApi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent Container</strong></td>
<td>AdcGeneral</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Adds / removes the services Adc_StartGroupConversion() and Adc_StopGroupConversion() from the code. true: Adc_StartGroupConversion() and Adc_StopGroupConversion() can be used. false: Adc_StartGroupConversion() and Adc_StopGroupConversion() can not be used.</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcucBooleanParamDef</td>
</tr>
<tr>
<td><strong>Default value</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Post-Build Variant Value</strong></td>
<td>false</td>
</tr>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td>Pre-compile time</td>
</tr>
<tr>
<td></td>
<td>Link time</td>
</tr>
<tr>
<td></td>
<td>Post-build time</td>
</tr>
<tr>
<td><strong>Scope / Dependency</strong></td>
<td>scope: local</td>
</tr>
</tbody>
</table>

#### SWS Item

<table>
<thead>
<tr>
<th>Name</th>
<th>AdcGrpNotifCapability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent Container</strong></td>
<td>AdcGeneral</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Determines, if the group notification mechanism (the functions to enable and disable the notifications) is available at runtime. true: Enabled. false: Disabled.</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcucBooleanParamDef</td>
</tr>
<tr>
<td><strong>Default value</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Post-Build Variant Value</strong></td>
<td>false</td>
</tr>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td>Pre-compile time</td>
</tr>
<tr>
<td></td>
<td>Link time</td>
</tr>
<tr>
<td></td>
<td>Post-build time</td>
</tr>
<tr>
<td><strong>Scope / Dependency</strong></td>
<td>scope: local</td>
</tr>
</tbody>
</table>

#### SWS Item

<table>
<thead>
<tr>
<th>Name</th>
<th>AdcHwTriggerApi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent Container</strong></td>
<td>AdcGeneral</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Adds / removes the services Adc_EnableHardwareTrigger() and</td>
</tr>
</tbody>
</table>
Adc_DisableHardwareTrigger() from the code.
true: Adc_EnableHardwareTrigger() and Adc_DisableHardwareTrigger() can be used.
false: Adc_EnableHardwareTrigger() and Adc_DisableHardwareTrigger() 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_Ad_00457 :
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 Value false

Value Configuration Class
Pre-compile time X All Variants
Link time --
Post-build time --

Scope / Dependency scope: local

SWS Item ECUC_Ad_00458 :
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 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_00393

**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 (AdcPriorityHwSw) 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.

**Implementation Type**: Adc_PriorityImplementationType

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
</tr>
</tbody>
</table>
| 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 |
| Value Configuration Class | Pre-compile time | X | All Variants |
|               | Link time       | -- |
|               | Post-build time | -- |
| Scope / Dependency | scope: local |

### SWS Item ECUC_Adc_00394

**Name**: AdcReadGroupApi  
**Parent Container**: AdcGeneral  
**Description**: Adds / removes the service Adc_ReadGroup() and from the code.  
true: Adc_ReadGroup() can be used.  
false: Adc_ReadGroup() can not be used.

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>EcucBooleanParamDef</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>false</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td>Pre-compile time</td>
</tr>
<tr>
<td></td>
<td>Link time</td>
</tr>
<tr>
<td></td>
<td>Post-build time</td>
</tr>
<tr>
<td>Scope / Dependency</td>
<td>scope: local</td>
</tr>
</tbody>
</table>

### SWS Item ECUC_Adc_00444

**Name**: AdcResultAlignment  
**Parent Container**: AdcGeneral  
**Description**: Alignment of ADC raw results in ADC result buffer (left/right alignment).  
**Implementation Type**: Adc_ResultAlignmentType

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
</tr>
</tbody>
</table>
Specification of ADC Driver
AUTOSAR CP Release 4.3.1

<table>
<thead>
<tr>
<th>Range</th>
<th>ADC_ALIGN_LEFT</th>
<th>left alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_ALIGN_RIGHT</td>
<td></td>
<td>right alignment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-Build Variant Value</th>
<th>Pre-compile time</th>
<th>X</th>
<th>All Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-build time</td>
<td>--</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Scope Dependency       | scope: local    |

SWS Item: ECUC_Ad_00409:
Name: AdcVersionInfoApi
Parent Container: AdcGeneral
Description:
Adds / removes the service Adc_GetVersionInfo() from the code.
true: Adc_GetVersionInfo() can be used.
false: Adc_GetVersionInfo() can not be used.

Multiplicity: 1
Type: EcucBooleanParamDef
Default value: false

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>X</th>
<th>All Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-build time</td>
<td>--</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scope / Dependency: scope: local

Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdcPowerStateConfig</td>
<td>0..*</td>
<td>Each instance of this parameter defines a power state and the callback to be called when this power state is reached.</td>
</tr>
</tbody>
</table>

10.2.3 AdcPowerStateConfig

SWS Item: ECUC_Ad_00459:
Name: AdcPowerStateConfig
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_Ad_00461:
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: EcuIntegerParamDef (Symbolic Name generated for this parameter)
Range: 0 .. 18446744073709551615
Default value: --
### Specification of ADC Driver

**AUTOSAR**

**CP Release 4.3.1**

---

**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**

**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**
- --

**maxLength**
- --

**minLength**
- --

**regularExpression**
- --

**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

**Description**
- This container contains the configuration parameters and sub containers of the AUTOSAR Adc module.

**Configuration Parameters**

---

#### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdcHwUnit</td>
<td>1..*</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

#### 10.2.5 AdcChannel

**SWS Item**
- **ECUC_Adc_00268**

**Container Name**
- AdcChannel

**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

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>Name</th>
<th>Parent Container</th>
<th>Description</th>
<th>Multiplicity</th>
<th>Type</th>
<th>Range</th>
<th>Default value</th>
<th>Post-Build Variant Value</th>
<th>Post-Build Variant Value</th>
<th>Multiplicity Configuration</th>
<th>Value Configuration Class</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECUC_Adc_00011</td>
<td>AdcChannelConvTime</td>
<td>AdcChannel</td>
<td>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</td>
<td>0..1</td>
<td>EcuIntegerParamDef</td>
<td>0 .. 18446744073709551615</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>Pre-compile time</td>
<td>X All Variants</td>
<td>scope: local</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Link time</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Post-build time</td>
<td>X All Variants</td>
<td></td>
</tr>
<tr>
<td>ECUC_Adc_00455</td>
<td>AdcChannelHighLimit</td>
<td>AdcChannel</td>
<td>High limit - used for limit checking.</td>
<td>0..1</td>
<td>EcuIntegerParamDef</td>
<td>0 .. 18446744073709551615</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>Pre-compile time</td>
<td>X All Variants</td>
<td>scope: local</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Link time</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Post-build time</td>
<td>X All Variants</td>
<td></td>
</tr>
</tbody>
</table>

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_Adcc_00392:

**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  
- Post-build time: X VARIANT-POST-BUILD  
**Scope / Dependency**: scope: local

### SWS Item ECUC_Adcc_00453:

**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 Value**: false  
**Multiplicity Configuration Class**:
- Pre-compile time: X All Variants  
- 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.  
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_Adcc_00454:

**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 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: --
### Value Configuration Class

| Pre-compile time | Link time | Post-build time |
|------------------|-----------|-----------------
| X                | --        | --              |

#### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>ECUC_Adc_00456 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Container</td>
<td>AdcChannel</td>
</tr>
<tr>
<td>Description</td>
<td>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</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>0..1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>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.</td>
</tr>
<tr>
<td>Post-Build Variant</td>
<td>false</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td>Pre-compile time: X All Variants</td>
</tr>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>--</td>
</tr>
</tbody>
</table>

#### SWS Item

<table>
<thead>
<tr>
<th>Name</th>
<th>ECUC_Adc_00089 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Container</td>
<td>AdcChannel</td>
</tr>
<tr>
<td>Description</td>
<td>Upper reference voltage source for each channel. Enumeration literals are defined vendor specific.</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>0..1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>--</td>
</tr>
<tr>
<td>Post-Build Variant</td>
<td>true</td>
</tr>
</tbody>
</table>
### Post-Build Variant Value
| true |
### Multiplicity Configuration Class Pre-compile time
<p>| X VARIANT-PRE-Compile |</p>
<table>
<thead>
<tr>
<th>Link time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-build time</td>
</tr>
<tr>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>
### Value Configuration Class Pre-compile time
<p>| X VARIANT-PRE-Compile |</p>
<table>
<thead>
<tr>
<th>Link time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-build time</td>
</tr>
<tr>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>
### Scope / Dependency
| scope: local |

---

### SWS Item
**ECUC_Ad_00023** :
**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 Value** true
### Multiplicity Configuration Class Pre-compile time
<p>| X VARIANT-PRE-Compile |</p>
<table>
<thead>
<tr>
<th>Link time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-build time</td>
</tr>
<tr>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>
### Value Configuration Class Pre-compile time
<p>| X VARIANT-PRE-Compile |</p>
<table>
<thead>
<tr>
<th>Link time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-build time</td>
</tr>
<tr>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>
### Scope / Dependency
| scope: local |

### SWS Item
**ECUC_Ad_00019** :
**Name** AdcChannelResolution
**Parent Container** AdcChannel
**Description** Channel resolution in bits. ImplementationType: Adc_ResolutionType
**Multiplicity** 0..1
**Type** EcuiIntegerParamDef
**Range** 1 .. 63
**Default value** --
**Post-Build Variant Value** true
### Multiplicity Configuration Class Pre-compile time
<p>| X VARIANT-PRE-Compile |</p>
<table>
<thead>
<tr>
<th>Link time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-build time</td>
</tr>
<tr>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>
### Value Configuration Class Pre-compile time
<p>| X VARIANT-PRE-Compile |</p>
<table>
<thead>
<tr>
<th>Link time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-build time</td>
</tr>
<tr>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>
### Scope / Dependency
| scope: local
dependency: AdcMaxChannelResolution: The actual resolution has to be less or equal than the maximum resolution. |

---

### SWS Item
**ECUC_Ad_00290** :
**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.
## Specification of ADC Driver

### AUTOSAR CP Release 4.3.1

**ImplementationType**: Adc_SamplingTimeType  
**Multiplicity**: 0..1  
**Type**: EcucIntegerParamDef  
**Range**: 0 .. 18446744073709551615  

**Default value**: --

### Post-Build Variant

**Multiplicity**: true

**Post-Build Variant Value**: true

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-compile time</td>
<td>X</td>
<td>VARIANT-PRE-COMPILE</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Post-build time</td>
<td>X</td>
<td>VARIANT-POST-BUILD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Value Configuration Class

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-compile time</td>
<td>X</td>
<td>VARIANT-PRE-COMPILE</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Post-build time</td>
<td>X</td>
<td>VARIANT-POST-BUILD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Scope / Dependency

scope: local

### No Included Containers

#### 10.2.6 AdcGroup

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Ad_00028 :</th>
<th>Container Name</th>
<th>AdcGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This container contains the Group configuration (parameters).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Configuration Parameters

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Ad_00317 :</th>
<th>Name</th>
<th>AdcGroupAccessMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Container</td>
<td>AdcGroup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Type of access mode to group conversion results.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImplementationType</td>
<td>Adc_GroupAccessModeType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>ADC_ACCESS_MODE_SINGLE</td>
<td>Single value access mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADC_ACCESS_MODE_STREAMING</td>
<td>Streaming access mode</td>
<td></td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>true</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Value Configuration Class

| | Pre-compile time | Link time | Post-build time |
| | | | |
| Pre-compile time | X | VARIANT-PRE-COMPILE | -- | |
| Post-build time | X | VARIANT-POST-BUILD | | |

### Scope / Dependency

dependency: AdcGroupTriggSrc / AdcGroupConvMode: streaming access mode is not available for one-shot conversion mode with software trigger source.

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Ad_00397 :</th>
<th>Name</th>
<th>AdcGroupConversionMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Container</td>
<td>AdcGroup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Type of conversion mode supported by the driver.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImplementationType</td>
<td>Adc_GroupConvModeType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>ADC_CONV_MODE_CONTINUOUS</td>
<td>Conversions of an ADC channel group are performed continuously after a software API call (start). The conversions itself are running</td>
<td></td>
</tr>
</tbody>
</table>

---

113 of 122

- AUTOSAR confidential -

Document ID 010: AUTOSAR_SWS_ADCDriver
**ADC_CONV_MODE_ONESHOT**

Automatically (no additional software or hardware trigger needed).

The conversion of an ADC channel group is performed once after a trigger.

<table>
<thead>
<tr>
<th>Post-Build Variant Value</th>
<th>true</th>
</tr>
</thead>
</table>

**Value Configuration Class**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X</th>
<th>VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Post-build time</td>
<td>X</td>
<td>VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

**Scope / Dependency**

/scope: local
dependency: AdcGroupTriggSrc: Continuous conversion mode only available for software triggered groups.

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_AdC_00398 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AdcGroupId</td>
</tr>
<tr>
<td>Parent Container</td>
<td>AdcGroup</td>
</tr>
<tr>
<td>Description</td>
<td>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</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcuUIntegerParamDef (Symbolic Name generated for this parameter)</td>
</tr>
<tr>
<td>Range</td>
<td>0 .. 1023</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-Build Variant Value</th>
<th>false</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>X</th>
<th>All Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Link time</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post-build time</td>
<td>X</td>
<td>VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

**Scope / Dependency**

scope: local

dependency: ADC_PRIORITY_IMPLEMENTATION

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_AdC_00287 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AdcGroupPriority</td>
</tr>
<tr>
<td>Parent Container</td>
<td>AdcGroup</td>
</tr>
<tr>
<td>Description</td>
<td>Priority level of the AdcGroup. ImplementationType: Adc_GroupPriorityType</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>0..1</td>
</tr>
<tr>
<td>Type</td>
<td>EcuUIntegerParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>0 .. 255</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-Build Variant Value</th>
<th>true</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Multiplicity Configuration Class</th>
<th>Pre-compile time</th>
<th>X</th>
<th>VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-build time</td>
<td>X</td>
<td>VARIANT-POST-BUILD</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>X</th>
<th>VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-build time</td>
<td>X</td>
<td>VARIANT-POST-BUILD</td>
<td></td>
</tr>
</tbody>
</table>

**Scope / Dependency**

scope: local
dependency: ADC_PRIORITY_IMPLEMENTATION
<table>
<thead>
<tr>
<th><strong>Parent Container</strong></th>
<th>AdcGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>0..1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcucEnumerationParamDef</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>ADC_GROUP_REPL_ABORT_RESTART</td>
</tr>
</tbody>
</table>

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.

| **Range**            | ADC_GROUP_REPL_SUSPEND_RESUME |

Suspend/Resume mechanism is used on group level, if a group is interrupted by a higher priority group. The conversions 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.

<table>
<thead>
<tr>
<th><strong>Post-Build Variant</strong></th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Post-Build Variant Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>Pre-compile time</td>
</tr>
<tr>
<td><strong>Configuration Class</strong></td>
<td>Link time</td>
</tr>
<tr>
<td><strong>Post-build time</strong></td>
<td>X VARIANT-POST-BUILD</td>
</tr>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td>Pre-compile time</td>
</tr>
<tr>
<td><strong>Link time</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Post-build time</strong></td>
<td>X VARIANT-POST-BUILD</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>scope: local</td>
</tr>
</tbody>
</table>

**SWS Item**: ECUC_Adc_00399
**Name**: AdcGroupTriggSrc
**Parent Container**: AdcGroup
**Description**: Type of source event that starts a group conversion. ImplementationType: Adc_TriggerSourceType
### Multiplicity

<table>
<thead>
<tr>
<th>Type</th>
<th>EcucEnumerationParamDef</th>
</tr>
</thead>
</table>

### Range

<table>
<thead>
<tr>
<th>ADC_TRIGG_SRC_HW</th>
<th>Group is triggered by a hardware event.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_TRIGG_SRC_SW</td>
<td>Group is triggered by a software API call.</td>
</tr>
</tbody>
</table>

### Post-Build Variant Value

| true |

### Value Configuration Class

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

### Scope Dependency

| scope: local dependency: AdcGroupConvMode: Trigger source HW is not available for continuous conversion mode. |

### SWS Item ECUC_Adc_00400:

<table>
<thead>
<tr>
<th>Name</th>
<th>AdcHwTrigSignal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Container</td>
<td>AdcGroup</td>
</tr>
</tbody>
</table>

| 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 |

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>0..1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>EcucEnumerationParamDef</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Range</th>
<th>ADC_HW_TRIG_BOTH_EDGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_HW_TRIG_FALLING_EDGE</td>
<td>React on the falling edge of the hardware trigger signal (only if supported by the ADC hardware).</td>
</tr>
<tr>
<td>ADC_HW_TRIG_RISING_EDGE</td>
<td>React on the rising edge of the hardware trigger signal (only if supported by the ADC hardware).</td>
</tr>
</tbody>
</table>

### Post-Build Variant Value

| true |

### Value Configuration Class

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

### Value Configuration Class

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>AdcHwTrigTimer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Container</td>
<td>AdcGroup</td>
</tr>
</tbody>
</table>

| Description | Reload value of the ADC module embedded timer (only if supported by ADC hardware). ImplementationType: Adc_HwTriggerTimerType |

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>0..1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>EcucIntegerParamDef</th>
</tr>
</thead>
</table>

| Range         | 0..18446744073709551615 |

116 of 122
## Specification of ADC Driver

### AUTOSAR CP Release 4.3.1

#### Default value

<table>
<thead>
<tr>
<th>Post-Build Variant Multiplicity</th>
<th>true</th>
</tr>
</thead>
</table>

#### Post-Build Variant Value

true

<table>
<thead>
<tr>
<th>Multiplicity Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>X</td>
<td>--</td>
<td>X</td>
<td>--</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Value Configuration Class

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X</th>
<th>VARIANT-PRE-COMPILE</th>
</tr>
</thead>
</table>

#### Scope / Dependency

scope: local
dependency: AdcTriggSrcHw: Valid only if the group is configured to be triggered by a hardware event.

---

**SWS Item** ECUC_Ad_00402:

#### Name

AdcNotification

#### Parent Container

AdcGroup

#### Description

Callback function for each group

#### Multiplicity

0..1

#### Type

EcucFunctionNameDef

#### Default value

--

#### Post-Build Variant Multiplicity

true

#### Post-Build Variant Value

true

<table>
<thead>
<tr>
<th>Multiplicity Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>X</td>
<td>--</td>
<td>X</td>
<td>--</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Value Configuration Class

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X</th>
<th>VARIANT-PRE-COMPILE</th>
</tr>
</thead>
</table>

#### Scope / Dependency

scope: local
dependency: This parameter is only available, if notification capability is configured available by AdcGrpNotifCapability

---

**SWS Item** ECUC_Ad_00316:

#### 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 the stream buffer is full (number of samples reached).

#### Post-Build Variant

true
### Value

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X VARIANT-PRE-COMPILE</td>
<td>--</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

| Scope / Dependency | scope: local dependency: AdcGroupAccessMode: Valid only for streaming access mode. |

### SWS Item

**ECUC_Adc_00292** :

**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**: EcuIntegerParamDef

**Range**: 1 .. 255

**Default value**: 1

**Post-Build Variant Value**: true

**Value Configuration Class**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

**Scope / Dependency**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

**SWS Item**

**ECUC_Adc_00014** :

**Name**: AdcGroupDefinition

**Parent Container**: AdcGroup

**Description**: Assignment of AdcChannels to a AdcGroups.

**ImplementationType**: Adc_GroupDefType

**Multiplicity**: 1..*

**Type**: Reference to [ AdcChannel ]

**Post-Build Variant Value**: true

**Multiplicity**: true

**Post-Build Variant Value**: true

**Value Configuration Class**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

**Scope / Dependency**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>X VARIANT-PRE-COMPILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>X VARIANT-POST-BUILD</td>
</tr>
</tbody>
</table>

### No Included Containers

[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)
### 10.2.7 AdcHwUnit

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00242 :</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

### Configuration Parameters

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00087 :</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>AdcClockSource</td>
</tr>
<tr>
<td><strong>Parent Container</strong></td>
<td>AdcHwUnit</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>0..1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcucEnumerationParamDef</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Post-Build Variant Multiplicity</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Post-Build Variant Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Configuration Class</strong></td>
<td><strong>Pre-compile time</strong> : X VARIANT-PRE-COMPILE</td>
</tr>
<tr>
<td><strong>Link time</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Post-build time</strong></td>
<td>X VARIANT-POST-BUILD</td>
</tr>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td><strong>Pre-compile time</strong> : X VARIANT-PRE-COMPILE</td>
</tr>
<tr>
<td><strong>Link time</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Post-build time</strong></td>
<td>X VARIANT-POST-BUILD</td>
</tr>
<tr>
<td><strong>Scope / Dependency</strong></td>
<td>scope: local</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00088 :</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>AdcPrescale</td>
</tr>
<tr>
<td><strong>Parent Container</strong></td>
<td>AdcHwUnit</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Optional ADC module specific clock prescale factor, if supported by hardware. ImplementationType: Adc_PrescaleType</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>0..1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcucIntegerParamDef</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>0 .. 65535</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
<tr>
<td>---------------</td>
<td>----</td>
</tr>
<tr>
<td>Post-Build Variant Multiplicity</td>
<td>true</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>true</td>
</tr>
</tbody>
</table>

### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdcChannel</td>
<td>1..*</td>
<td>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.</td>
</tr>
<tr>
<td>AdcGroup</td>
<td>1..*</td>
<td>This container contains the Group configuration (parameters).</td>
</tr>
</tbody>
</table>

[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

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00030 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Name</td>
<td>AdcPublishedInformation</td>
</tr>
<tr>
<td>Description</td>
<td>Additional published parameters not covered by “Common” Published Information. Note that these parameters have “PUBLISHED-INFORMATION” configuration class setting, since they are published information.</td>
</tr>
</tbody>
</table>

### Configuration Parameters

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Adc_00410 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AdcChannelValueSigned</td>
</tr>
<tr>
<td>Parent Container</td>
<td>AdcPublishedInformation</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucBooleanParamDef</td>
</tr>
</tbody>
</table>
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)
11 Not applicable requirements

[SWS_Adc_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_00303,
SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00371, 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)