ADC (EXpansion pack) 4.0 driver user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This guide describes the architecture, configuration and use of the analog digital converter (ADC) driver. It will help you to understand the functionality of the driver and will provide a reference to the driver's API.

The installation, build process, and general information about the use of the EB tresos Studio are not within the scope of this document. See the *EB tresos Studio for ACG8 user's guide* [8] for detailed information of these topics.

Intended audience

This document is intended for anyone who uses the analog digital converter (ADC) (expansion pack) driver of the TRAVEO™ T2G family.

Document Structure

Chapter 1 General overview gives a brief introduction to the ADC driver, explains the embedding in the AUTOSAR environment and describes the supported hardware and development environment.

Chapter 2 Using the ADC driver details the steps required to use the ADC driver in your application.

Chapter 3 Structure and dependencies describes the file structure and the dependencies for the ADC driver.

Chapter 4 EB tresos Studio configuration interface describes the driver's configuration with the EB tresos Studio software.

Chapter 5 Functional description gives a functional description of all services offered by the ADC driver.

Chapter 6 Hardware resources gives a description of all hardware resources used.

The Appendix A and Appendix B provides a complete API reference and access register table.

Abbreviations and definitions

Table 1 Abbreviation

Abbreviation	Description
ADC	Analog Digital Converter
Alternate calibration	Alternate calibration is used to quietly run calibration algorithm (in the background) for all groups except groups which are used for regular calibration.
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
Basic Software	Standardized part of software which does not fulfill a vehicle functional job.

Abbreviation	Description
BSW	Basic Software. Standardized part of software which does not fulfill a vehicle functional job.
Channel	Represents analog inputs. These analog inputs are analog signal pin and internal signals. Channel means ADC physical channel.
Channel Group	A set of one or more channels, which can be accessed as a single entity by its group name.
CPU	Central Processing Unit
DEM	Diagnostic Event Manager
DET	Default Error Tracer
DMA	Direct Memory Access
DW	Data Wire, this is CPU feature. DW is used for peripheral-to-memory and memory-to-peripheral data transfers. DW is also called Peripheral-DMA (P-DMA) controller. Generically, this feature is called “DMA”.
EB tresos Studio	Elektrobit Automotive configuration framework
GCE	Generic Configuration Editor
GPT	General Purpose Timer
HW	Hardware
HW trigger	HW trigger signal that starts one conversion of an ADC channel group. HW trigger signals are generated by timer, GPIO signal, etc.
HW unit	Represents a microcontroller input electronic device that includes all parts which are needed to perform an “analog to digital conversion”.
ISR	Interrupt Service Routine
Logical channel	SAR ADC has some logical channels depend on hardware. The group consists of several consecutive logical channels. SAR ADC samples the analog input (channel) mapped to logical channel.
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
OCU	Output Compare Unit
OS	Operating System
PASS	Programmable Analog Subsystem, this is hardware subsystem. This subsystem includes “SAR ADC”.
Preconditioning	Enabling broken wire detection by charging or discharging the ADC sampling capacitor before sampling the input signal.
PWM	Pulse Width Modulation
Regular calibration	Regular calibration is used to run calibration algorithm for all channel groups except groups which are used for alternate calibration.
Result (Stream) Buffer	The user of the ADC driver must 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. (Result buffer is called stream buffer in general when the channel group is handled in streaming access mode.)

Abbreviation	Description
SAR ADC	Successive Approximation Register analog-to-digital converter, this is hardware peripheral. SAR ADC means ADC HW unit.
SARMUX	SARMUX is the analog multiplexer that routes the signal to be converted to the ADC core input. SARMUXn is an analog multiplexer of SAR ADCn, where n indicates the SAR ADC number.
SelfDiag	Self diagnostic
SW	Software
SW trigger	Starting an ADC group conversion by an API call
TCPWM	16-bit and four 32-bit Timer/Counter Pulse-Width Modulator, this is hardware peripheral.
UTF-8	8-Bit Universal Character Set Transformation Format

Related documents

AUTOSAR requirements and specifications

Bibliography

[1] General specification of basic software modules, AUTOSAR release 4.2.2.

[2] Specification of ADC driver, AUTOSAR release 4.2.2.

[3] Specification of standard types, AUTOSAR release 4.2.2.

[4] Specification of ECU configuration parameters, AUTOSAR release 4.2.2.

[5] Specification of default error tracer, AUTOSAR release 4.2.2.

[6] Specification of diagnostic event manager, AUTOSAR release 4.2,2.

[7] Specification of memory mapping, AUTOSAR release 4.2.2

Elektrobit automotive documentation

Bibliography

[8] EB tresos Studio for ACG8 user’s guide.

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

[9] Layered software architecture, AUTOSAR release 4.2.2.

Table of contents

1 General overview

1.1 Introduction to the ADC driver

The ADC driver is a set of software routines, which enable the requested analog-to-digital conversions of ADC channels.

For this purpose, the ADC driver provides services for:

- Initializing and de-initializing the driver
- Starting and stopping a group conversion
- Reading group conversion results
- Enabling and disabling group notification
- Enabling and disabling power regarding ADC hardware
- Changing sampling time for each ADC channel

The driver conforms to the AUTOSAR standard and is implemented according to the *Specification of ADC driver* [2].

The ADC driver is delivered with a plugin for the EB tresos Studio, which allows you to statically configure the driver. The driver provides an interface to enable ADC hardware channels, to define symbolic names for hardware units, channel groups, channels and to create channel groups.

1.2 User profile

This guide is intended for users with a least basic knowledge of the following domains:

- Embedded systems
- C programming language
- AUTOSAR standard
- Target hardware architecture

1.3 Embedding in the AUTOSAR environment

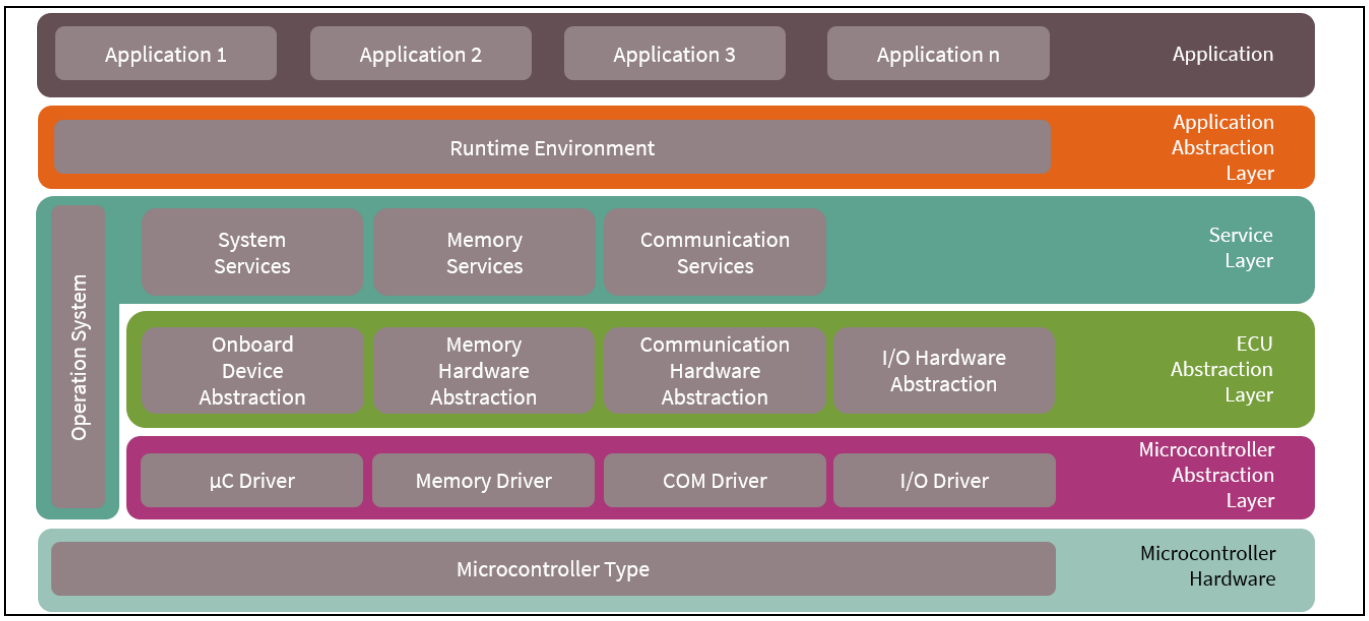

Figure 1 Overview of AUTOSAR software layers

Figure 1 depicts the layered AUTOSAR software architecture. The ADC driver (Figure 2) is part of the microcontroller abstraction layer (MCAL), the lowest layer of basic software in the AUTOSAR environment.

As an internal I/O driver, it provides a standardized and microcontroller-independent interface to higher software layers for starting/stopping ADC conversion of single/multiple channels of the ECU hardware.

For an overview of the AUTOSAR layered software architecture, see the *layered software architecture* [9].

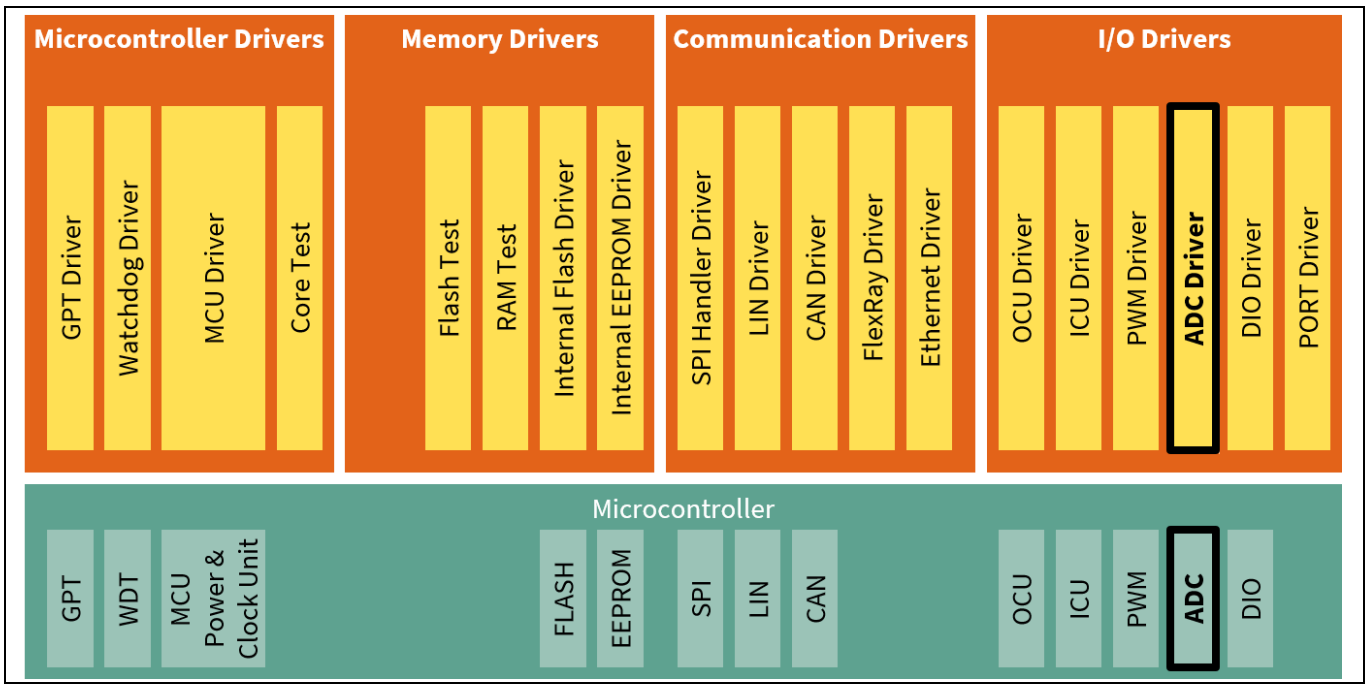

Figure 2 ADC driver in MCAL layer

1.4 Supported hardware

This version of the ADC driver supports the TRAVEO™ T2G microcontroller family. The supported derivatives are listed in the release notes.

Additional derivatives that contain only a subset of the capabilities of one derivative mentioned above can be implemented / supported by providing a resource file with its properties.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The BASE, MAKE, PORT, MCU, and RESOURCE module are required for the proper functionality of the ADC driver. PWM, GPT, and OCU module are not mandatory, and they can be used to trigger a group conversion by external signals (HW trigger) if necessary.

1.6 Character set and encoding

All source code files of the ADC driver are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 … 0x7F) being used.

1.7 Multicore support

The ADC driver supports multicore type II. The driver also supports multicore type III for some APIs (for example, read-only API or atomic-write API). For each multicore type, see the following sections.

Note: *If multicore type III is required, the section including data related to read-only API or atomic write API must be allocated to the memory which can be read from any cores.*

1.7.1 Multicore type

In this section, type I, type II, and type III are defined as multicore characteristics.

1.7.1.1 Single core only (Multicore type I)

For this multicore type, the driver is available on a single core. This type is referred as “Multicore Type I”.

Multicore type I has the following characteristic:

- The peripheral channels are accessed by only one core.

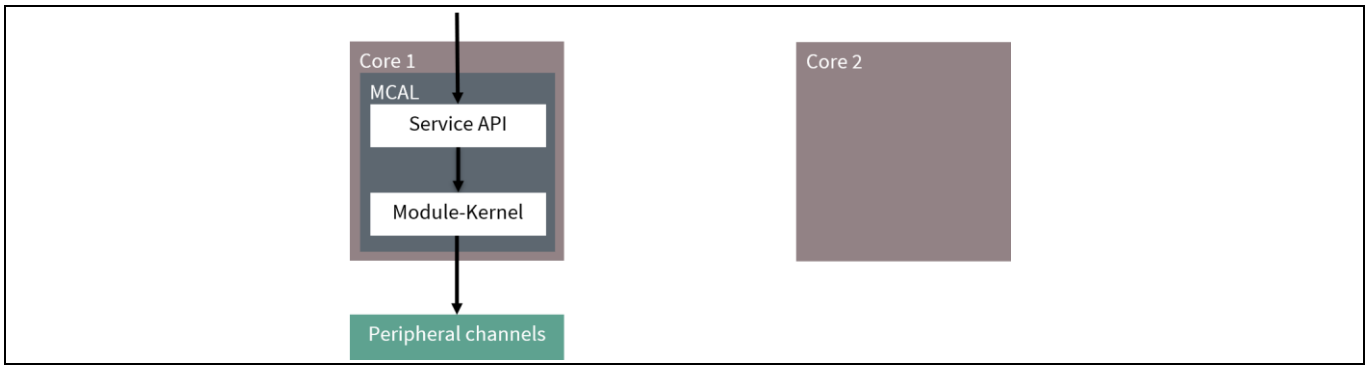

Figure 3 Overview of the multicore type I

1.7.1.2 Core dependent instances (Multicore type II)

For this multicore type, the driver has core-dependent instances with individually allocable hardware. This type is referred as “Multicore Type II”.

Multicore type II has the following characteristics:

- The driver code is shared among all cores
  - A common binary is used for all cores
  - A configuration is common for all cores
- Each core runs an instance of the driver
- Peripheral channels and their data are individually allocable to cores but cannot be shared among cores
- One core will be the master; and the master core must be initialized first
  - Cores other than the master core are called satellite cores.

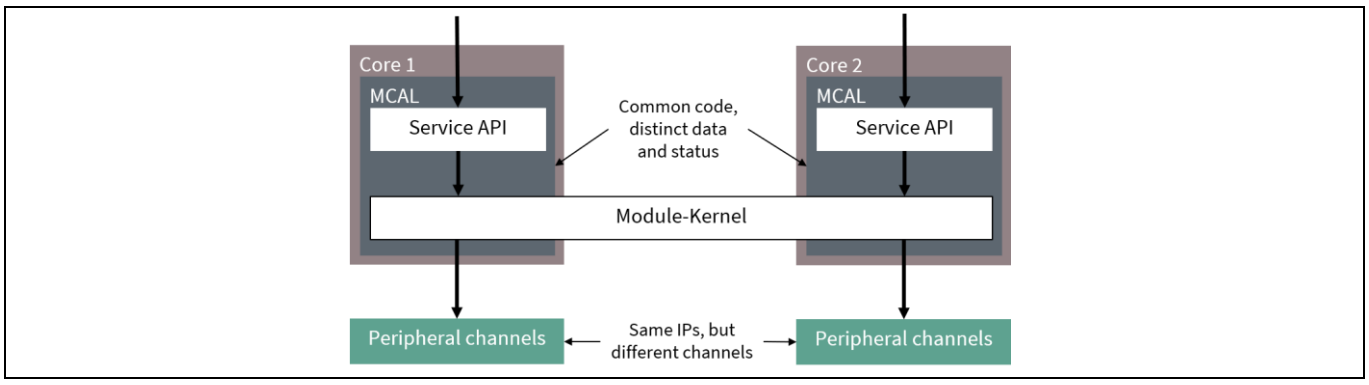

Figure 4 Overview of the multicore type II

1.7.1.3 Core independent instances (Multicore type III)

For this multicore type, the driver has core independent instances with globally available hardware. This type is referred as “Multicore Type III”.

Multicore type III has the following characteristics:

- The code of the driver is shared among all cores
  - A common binary is used for all cores
  - A configuration is common for all cores
- Each core runs an instance of the driver
- Peripheral channels are globally available for all cores

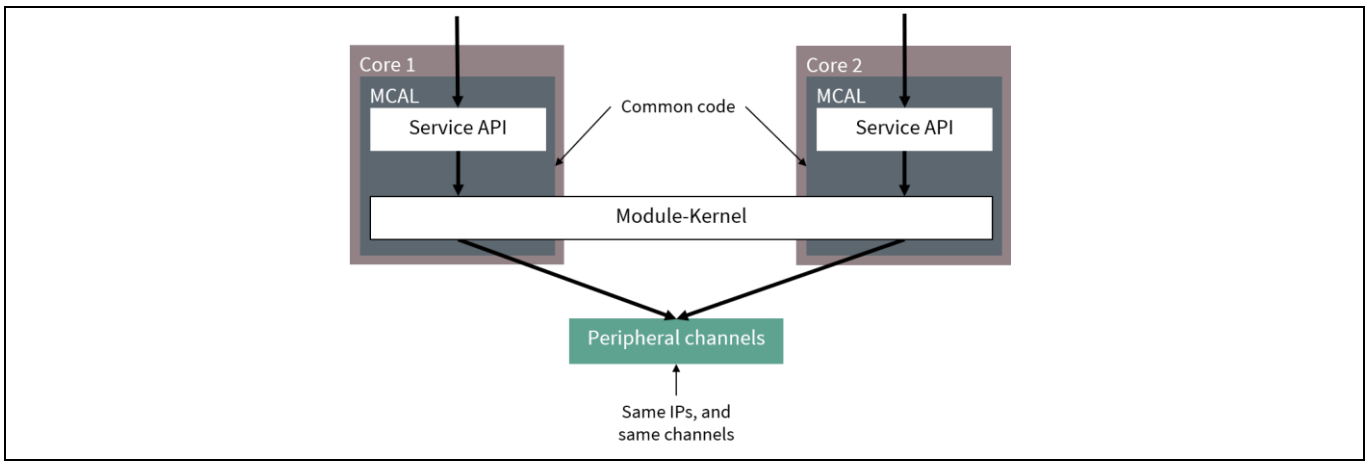

Figure 5 Overview of the multicore type III

1.7.2 Virtual core support

The ADC driver supports a number of cores. The configured cores need not to be equal to the physical cores.

The ADC driver calls a configurable callout function (`AdcGetCoreIdFunction`) to identify the core that is currently executing the code. This function can be implemented in the integration scope. The function can be written such that it does not return the physical core, but instead returns the SW partition ID, OS application ID, or any attribute/parameter. By interpreting these as the core, the ADC driver can support multiple SW partitions on a single physical core.

2 Using the ADC driver

This chapter describes all necessary steps to incorporate the ADC driver into your application.

2.1 Installation and prerequisites

Note: Before you start, see the EB tresos Studio for ACG8 user's guide [8] for the following information.

1. *The installation procedure of EB tresos ECU AUTOSAR components.*
2. *The usage of the EB tresos Studio.*
3. *The usage of the EB tresos ECU AUTOSAR build environment (It includes the steps to setup and integrate the own application within the EB tresos ECU AUTOSAR build environment).*

The installation of the ADC driver complies with the general installation procedure for EB tresos ECU AUTOSAR components given in the documents mentioned above. If the driver has been successfully installed, the driver will appear in the module list of the EB tresos Studio (see the *EB tresos Studio for ACG8 user's guide* [8]).

This guide assumes that the project is properly set up and is using the application template as described in the *EB tresos Studio for ACG8 user's guide* [8]. This template provides the necessary folder structure, project and makefiles needed to configure and compile your application within the build environment. You also must be familiar with the use of the command shell.

2.2 Configuring the ADC driver

The ADC driver can be configured with any AUTOSAR compliant GCE tool. Save the configuration in a separate file (for example, *Adc.xdm*). More information about the ADC driver configuration can be found in 4 EB tresos Studio configuration interface.

2.2.1 Architecture specifics

ADC driver supports not only AUTOSAR specification but also vendor- and driver-specific features. Therefore vendor- and driver-specific parameters are also added. All vendor- and driver-specific configuration parameters are listed as below:

- `AdcDemEventParameterRefs`
  - `ADC_E_HARDWARE_ERROR`
- `AdcConfigSet`
  - `AdcSupplyMonitorEnabledA`
  - `AdcSupplyMonitorLevelA`
  - `AdcSupplyMonitorEnabledB`
  - `AdcSupplyMonitorLevelB`
- `AdcHwUnit`
  - `AdcDiagnoseEnable`
  - `AdcDiagnosticReference`
  - `AdcPreconditionCycle`
  - `AdcSarMux1ConnectToAdc0`
  - `AdcSarMux1DiagnoseEnable`
  - `AdcSarMux1DiagnosticReference`
  - `AdcSarMux2ConnectToAdc0`

  - AdcSarMux2DiagnoseEnable
  - AdcSarMux2DiagnosticReference
  - AdcSarMux3ConnectToAdc0
  - AdcSarMux3DiagnoseEnable
  - AdcSarMux3DiagnosticReference
  - AdcCoreAssignment
- AdcChannel
  - AdcUseExternalMultiplexer
  - AdcChannelResultSigned
  - AdcChannelPulseDetect
  - AdcChannelPulsePositiveCount
  - AdcChannelPulseNegativeCount
  - AdcDiagnosisMode
- AdcGroup
  - AdcGroupHwTriggSrc
  - AdcFirstLogicalChannel
  - AdcInterruptMode
  - AdcUseDma
  - AdcUseAlternateCalibration
  - AdcSampleMode
  - AdcUseRedundancy
  - AdcLimitCheckNotification
  - AdcUseDynamicAllocate
  - AdcSWPriority
- AdcGeneral
  - AdcCalibrationApi
  - AdcErrorCalloutFunction
  - AdcIncludeFile
  - AdcCoreConsistencyCheckEnable
  - AdcGetCoreIdFunction
  - AdcMasterCoreReference
- AdcCustomFunction
  - AdcSelfDiagApi
  - AdcVoltageDeviation
  - AdcDiagConvertTimeout
- AdcGenericHWTriggerSelect
  - AdcGroupGenericHwTriggSrc
  - AdcGenericTriggerSelect
- AdcCoreConfiguration
  - AdcCoreConfigurationId
  - AdcCoreId

2.3 Adapting an application

To use the ADC driver in your application, you first must include the header files of ADC, PORT, and MCU driver by adding the following lines of code to your source file:

```
#include "Mcu.h" /* AUTOSAR MCU Driver */
#include "Port.h" /* AUTOSAR PORT Driver */
#include "Adc.h" /* AUTOSAR ADC Driver */
```

This publishes all required function and data prototypes and symbolic names of the configuration into the application. In addition, you must implement the error callout function for ASIL safety extension.

To use the ADC driver, the appropriate port pins, SAR clock setting, triggers (HW trigger and trigger to DMA), and ADC interrupts must be configured in PORT driver, MCU driver, and OS. For detailed information see Hardware resources.

Initialization of MCU, PORT, and ADC driver needs to be done in the following order:

For the master core:

```
Mcu_Init(&Mcu_Config[0]);
Port_Init(&Port_Config[0]);
Adc_Init(&Adc_Config[0]);
```

For the satellite core:

```
Mcu_Init(&Mcu_Config[0]);
Adc_Init(&Adc_Config[0]);
```

Note: *As a reference, the symbolic name can also be specified (for instance, `AdcConf_AdcConfigSet_AdcConfigSet_0`).*

The function `Mcu_Init()` is called with a pointer to a structure of type `Mcu_ConfigType`, which is published by the MCU driver itself.

The function `Port_Init()` is called with a pointer to a structure of type `Port_ConfigType`, which is published by the PORT driver itself. This function must be called on the master core only.

The function `Adc_Init()` is called with a pointer to a structure of type `Adc_ConfigType`, which is published by the ADC driver itself.

The master core must be initialized prior to the satellite core. All cores must be initialized with the same configuration.

After initialization of the ADC driver all groups are stopped and notifications are disabled. Initialization can be done by calling the `Adc_Init()` API.

A channel can only be sampled if it is part of or included in an ADC group. The group needs to be started to convert the channel values.

Each group has its individual state, which can be either of the following:

- `ADC_IDLE`: The group is inactive. The group was neither started nor enabled.

  The group will not convert any samples as long as it is in this state.

- ADC_BUSY: The group is active. The group was started or enabled and has not finished conversion yet.

  The group has not yet generated new sample data since it was started/enabled or since sample data was read by the ADC driver's environment the previous time.

- ADC_COMPLETED: The group is active. The group was started or enabled and has already finished at least one conversion round, but has not filled up the result buffer completely.
  The group has already generated new sample data. Sample data is available in the result buffer that was not yet read by the ADC driver's environment.
- ADC_STREAM_COMPLETED: The group completed conversion and has filled up the result buffer completely. Whether the group is active or inactive depends on its mode. SW trigger continuous conversion and single access group, HW trigger one-shot conversion and single access group, or circular streaming access group keep active even though the result buffer is fulfilled. SW trigger one-shot conversion and single access group, or linear streaming access group would be inactive immediately after the result buffer is fulfilled.
  The group has already generated new sample data. Sample data is available in the result buffer that was not read yet by the ADC module's environment.
- ADC_ERROR: The group is inactive, that is, a DMA transmission error occurred. The result for current group conversion is not available.

After the group state reaches ADC_COMPLETED or ADC_STREAM_COMPLETED, the sample data may be read. In state ADC_BUSY, reading will return no result.

Example: Group (MY_ADC_GROUP_1) is configured for single access mode and references MY_ADC_CHANNEL_0 and MY_ADC_CHANNEL_2 to define the channel group.

```
/* Allocate a buffer to receive results for each channel. */
Adc_ValueGroupType sampleBuffer[2];

Adc_SetupResultBuffer(MY_ADC_GROUP_1, sampleBuffer)

Adc_StartGroupConversion(MY_ADC_GROUP_1);

/* Try to read until group is finished. */
while (Adc_ReadGroup(MY_ADC_GROUP_1, sampleBuffer) != E_OK);
```

The above code shows that Adc_ReadGroup() can be used to wait until the completion of the conversion. It is also possible to wait until the completion of the conversion by calling Adc_GetGroupStatus() instead Adc_ReadGroup().

```
/* Try to check status until group is finished. */
while (Adc_GetGroupStatus(MY_ADC_GROUP_1) != ADC_STREAM_COMPLETED);
```

For more information on Adc_StartGroupConversion() and other services, see

Functions.

A notification function can be called to announce the completion of the ADC channel group conversion if the notification function is configured in advance. In this case, your application must provide the functions and its declarations that you configured. The file containing the declarations must be included using the `AdcIncludeFile` parameter which is located in container `AdcGeneral`. The notification functions, which are end of conversion, take no parameters and have void return type:

```
void MyNotificationFunction(void)
{
/* Insert your code here */
}
```

The limit check notification functions take one parameter for limit check information and have void return type. `LimitCheckState` indicates the channel in the group that corresponds to the condition of limit check. 0 bit of `LimitCheckState` indicates the first channel in the group:

```
void MyLimitCheckNotificationFunction(UINT32 LimitCheckState)
{
/* Insert your code here */
}
```

The notification functions are called from an interrupt context.

2.4 Starting the build process

Do the following to build your application.

Note: *For a clean build, you should use the build command with target `clean_all` before (`make clean_all`).*

1. Type the following in the command shell to generate the necessary dependent files:

```
> make generate
```

The details of the generated files are described in Generated files.

2. Then, type the following command to resolve the required file dependencies:

```
> make depend
```

3. Finally, you can compile and link the application with the following command:

```
> make (optional target: all)
```

The application is now built. All files are compiled and linked to a binary file, which can be downloaded to the target CPU cores.

2.5 Measuring the stack consumption

Do the following to measure stack consumption. It requires the Base module for proper measurement.

Note: *All files (including library files) should be rebuilt with the dedicated compiler option. The executable file built by this step must be used only to measure stack consumption.*

1. Add the following compiler option to the Makefile to enable stack consumption measurement:

   ```
   -DSTACK_ANALYSIS_ENABLE
   ```

2. Type the following command to clean library files:

   ```
   > make clean_lib
   ```

3. Follow the build process described in Starting the build process.
4. Measure the stack consumption by following the instructions given in the release notes.

2.6 Memory mapping

The *Adc_MemMap.h* file in the *$(TRESOS_BASE)/plugins/MemMap_TS_T40D13M0I0R0/include* directory is a sample. This file is replaced by the file generated by MEMMAP module. Input to the MEMMAP module is generated as *Adc_Bswmd.arxml* in the *$(PROJECT_ROOT)/output/generated/swcd* directory of your project folder.

2.6.1 Memory allocation keyword

- `ADC_START_SEC_CODE_ASIL_B /ADC_STOP_SEC_CODE_ASIL_B`

  The memory section type is CODE. All executable code is allocated in this section.

- `ADC_START_SEC_CONST_ASIL_B_UNSPECIFIED /ADC_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

  The memory section type is CONST. The following constants are allocated in this section:

  - ADC hardware configuration setting
  - ADC channel configuration setting
  - ADC channel group configuration setting
  - Lookup table to search a specific ADC channel group by interrupt resources index
  - DMA configuration setting
  - DMA channel configuration setting
  - ADC whole configuration setting
  - Pointer to driver status
  - Pointer to target power status

- `ADC_CORE[AdcCoreConfigurationId]_START_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED /ADC_CORE[AdcCoreConfigurationId]_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED`

  The memory section type is VAR. The following variables are allocated in this section:

  - Pointer to the configuration data set by `Adc_Init()`. (Initial value: NULL)
  - Information for driver status

- `ADC_CORE[AdcCoreConfigurationId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED /ADC_CORE[AdcCoreConfigurationId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED`

  The memory section type is VAR. The following variable is allocated in this section:

  - Information for ADC DMA descriptors
  - Information for target power state
  - Information for stream setting
  - Information for ADC channel group status
  - SW priority management information

- `ADC_CORE[MasterCoreId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32 /ADC_CORE[MasterCoreId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32`

  MasterCoreId means the AdcCoreConfigurationId specified in AdcMasterCoreReference.

  The memory section type is VAR. The following variable is allocated in this section:

  - Customer special flag

- `ADC_CORE[AdcCoreConfigurationId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_16`
  `/ADC_CORE[AdcCoreConfigurationId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_16`

  The memory section type is VAR. The following variable is allocated in this section:

  - First/Last enable channel information
- `ADC_CORE[AdcCoreConfigurationId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32`
  `/ADC_CORE[AdcCoreConfigurationId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32`

  The memory section type is VAR. The following variables are allocated in this section:

  - Enable/disable group channel information
  - Unread/read information
  - Limit check enable point information
- `ADC_CORE[AdcCoreConfigurationId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_BOOLEAN`
  `/ADC_CORE[AdcCoreConfigurationId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_BOOLEAN`

  The memory section type is VAR. The following variable is allocated in this section:

  - Notification enable setting

2.6.2 Memory allocation and constraints

All the memory sections that store init or uninit status must be zero-initialized before any driver function is executed on any core. If core consistency checks are disabled, inconsistent parameters would be detected and reported by PPU and SMPU.

- `ADC_CORE[AdcCoreConfigurationId]_START_VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]`
  `/`
  `ADC_CORE[AdcCoreConfigurationId]_STOP_VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]`

  This section is read/write accessed from the core represented by `AdcCoreConfigurationId` and read accessed from the other cores. Therefore, this section must not be allocated to TCRAM. For the core represented by `AdcCoreConfigurationId`, this section must be allocated to either non-cache or write-through cache SRAM area. For other cores, this section must be allocated to non-cache SRAM area.

  For multicore type III, this section is read accessed from other cores. Therefore, this section must not be allocated to TCRAM. For the core represented by `AdcCoreConfigurationId`, this section must be allocated to either non-cache or write-through cache SRAM area. For other cores, this section must be allocated to non-cache SRAM area.

For the details of `INIT_POLICY` and `ALIGNMENT`, see the *specification of memory mapping* [7].

3 Structure and dependencies

The ADC driver consists of static, configuration, and generated files.

3.1 Static files

- $(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/Adc_TS_* is the path to the ADC driver plugin.
- *$(PLUGIN_PATH)/lib_src* contains all static source files of the ADC driver. These files contain the functionality of the driver, which does not depend on the current configuration. The files are grouped into a static library.
- *$(PLUGIN_PATH)/lib_include* contains all the internal header files for the ADC driver.
- *$(PLUGIN_PATH)/src* comprises configuration dependent source files or special derivative files. Each file will be rebuilt when the configuration is changed.

All necessary source files will automatically be compiled and linked during the build process and all include paths will be set if the ADC driver is enabled.

- *$(PLUGIN_PATH)/include* is the basic public include directory needed by the user and should be included in Adc.h.
- *$(PLUGIN_PATH)/autosar* directory contains the AUTOSAR ECU parameter definition with vendor, architecture and derivative specific adaptations to create a correct matching parameter configuration for the ADC driver.

3.2 Configuration files

The configuration of the ADC driver is done via EB tresos Studio. The file containing the ADC driver's configuration is named *Adc.xdm* and is located in the directory *$(PROJECT_ROOT)/config*. This file serves as input for the generation of the configuration-dependent source and header files during the build process.

3.3 Generated files

During the build process the following files are generated on the basis of the current configuration description. They are located in the sub folder *output/generated* of your project folder:

- *include/Adc_Cfg.h* provides settings of configurations with pre-compiled attribute, for example, all symbolic names required by the API. In addition, it defines a `DemEventId` parameter of the DEM module, which is referred in configuration. It will be included in *Adc.h*.
- *include/Adc_ExternalInclude.h* provides the external include file information of configuration.
- *include/Adc_Irq.h* provides generated declaration of interrupt service routine.
- *include/Adc_PBcfg.h* provides settings of configurations with post-build attribute.
- *src/Adc_Irq.c* contains the interrupt service routine.
- *src/Adc_PBcfg.c* contains the constants for the ADC configuration.

Note: *Generated source files do not need to be added to your application make file. They will be compiled and linked automatically during the build process.*

- *swcd/Adc_Bswmd.arxml* contains BSW module description.

Note: *Additional steps are required for the generation of BSW module description. In EB tresos Studio, follow the menu path **Project** > **Build Project** and click **generate_swcd**.*

3.4 Dependencies

3.4.1 MCU driver

Although the ADC driver can be successfully compiled and linked without an AUTOSAR compliant MCU driver, the latter is required to configure and initialize SAR clock. Otherwise, the ADC driver does not work expectedly. The MCU driver needs to be initialized before the ADC driver is initialized. `Mcu_GetCoreID` can optionally be set to the `AdcGetCoreIdFunction` configuration parameter. See the MCU driver's user guide for details.

3.4.2 PORT driver

Although the ADC driver can be successfully compiled and linked without an AUTOSAR compliant PORT driver, the latter is required to configure and initialize all ports, HW trigger route, and DMA route (connection between IPs). Otherwise, the ADC driver will does not work expectedly. The PORT driver needs to be initialized before the ADC driver is initialized.

3.4.3 AUTOSAR OS

The AUTOSAR operating system handles the interrupts used by the ADC driver. See Interrupts for further information. `GetCoreID` can optionally be set to the `AdctGetCoreIdFunction` configuration parameter.

3.4.4 DET

If the default error detection feature is enabled in the ADC driver configuration, the DET must be installed, configured, and integrated into the application also.

3.4.5 DEM

If the diagnostic event manager is enabled in the ADC driver configuration, the DEM must be installed, configured, and integrated into the application.

3.4.6 GPT, PWM, and OCU driver (hardware trigger sources)

An ADC conversion can be triggered by hardware, but configuration and controlling of the hardware trigger source is not within scope of the ADC driver. The following hardware trigger options are available:

- GPT driver

  TCPWM can generate toggle signals to trigger an ADC channel group conversion when the timer expires. You are responsible for configuring and controlling the timer corresponding to ADC first channel in group by using GPT driver. Trigger functionality is enabled when the `GptHwTriggerOutputLine` configuration parameter is configured in GPT driver.

- PWM driver

  TCPWM can generate toggle signals to trigger an ADC channel group conversion when the timer counter matches the comparison value which is set in advance. You are responsible for configuring and controlling the timer corresponding to ADC first channel in group by using PWM driver. Trigger functionality is enabled, if the `PwmHwTriggerOutputLine` configuration parameter is configured in PWM driver. Configuration parameters `PwmHwTriggerOutputFactor` and, `PwmHwTriggerOutputDefaultTime` or `PwmHwTriggerOutputDefaultTick` can be used to set the trigger timing.

- OCU driver

  TCPWM can generate toggle signals to trigger an ADC channel group conversion when the timer counter matches the threshold value which is set in advance. You are responsible for configuring and controlling the timer corresponding to ADC first channel in group by using OCU driver. Trigger functionality is enabled when the `OcuHwTriggerOutputLine` configuration parameter is configured in OCU driver.

3.4.7 Error callout handler

The error callout handler is called on every error that is detected, regardless of whether default error detection is enabled or disabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via the `AdcErrorCalloutFunction` configuration parameter.

3.4.8 BSW scheduler

The ADC driver uses the following services of the BSW scheduler (originally named SchM, now BswM) to enter and leave critical sections:

- `SchM_Enter_Adc_ADC_EXCLUSIVE_AREA_[AdcCoreConfigurationId](void)`
- `SchM_Exit_Adc_ADC_EXCLUSIVE_AREA_[AdcCoreConfigurationId](void)`

You must ensure that the BSW scheduler is properly configured and initialized before using the ADC driver services.

4 EB tresos Studio configuration interface

The GUI is not part of this delivery. For further information, see the *EB tresos Studio for ACG8 user's guide* [8].

4.1 General configuration

This container has the following parameters to configure the general functions of ADC driver:

- `ADC_E_HARDWARE_ERROR` is a reference to the configured DEM event to report "Hardware failure". If the reference is not configured, the error will not be reported.
- `AdcDevErrorDetect` enables or disables the development error notification for the ADC driver.

  Setting this parameter to FALSE will disable the notification of development errors via DET. However, in contrast to AUTOSAR specification, detection of development errors is still enabled and errors will be reported via `AdcErrorCalloutFunction`.
- `AdcDeInitApi` adds or removes the service `Adc_DeInit()` from the code.
- `AdcEnableLimitCheck` enables or disables the limit checking feature in the ADC driver.
- `AdcEnableQueuing` determines if the queuing mechanism is active in case the priority mechanism is disabled.

Note: *This parameter is not evaluated by the ADC driver because the prioritization mechanism `ADC_PRIORITY_NONE` is not supported.*

- `AdcEnableStartStopGroupApi` adds or removes the services `Adc_StartGroupConversion()` and `Adc_StopGroupConversion()` from the code.
- `AdcGrpNotifCapability` determines if the group notification mechanism (the functions to enable and disable the notifications) is available at runtime.
- `AdcHwTriggerApi` adds or removes the services `Adc_EnableHardwareTrigger()` and `Adc_DisableHardwareTrigger()` from the code.
- `AdcCalibrationApi` adds or removes the services `Adc_ChangeCalibrationChannel()`, `Adc_SetCalibrationValue()`, `Adc_GetCalibrationAlternateValue()`, and `Adc_GetCalibrationValue()` from the code.
- `AdcLowPowerStatesSupport` adds or removes all power state management related APIs (`Adc_SetPowerState()`, `Adc_GetCurrentPowerState()`, `Adc_GetTargetPowerState()`, `Adc_PreparePowerState()`, `Adc_Main_PowerTransitionManager()`).
- `AdcPowerStateAsynchTransitionMode` enables or disables support of the ADC driver to the asynchronous power state transition.

Note: *As there is no preparation period in the hardware feature, only synchronous power state transition mode is supported. Therefore, this parameter is not used by the ADC driver and is not being evaluated.*

- `AdcPriorityImplementation` 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.
  - `ADC_PRIORITY_HW`: Only hardware priority mechanism is available.
  - `ADC_PRIORITY_NONE`: Priority mechanism is not available.
  - `ADC_PRIORITY_HW_SW`: Hardware and software priority mechanism are available.

Note: *The HW priority mechanism is used for all groups (including SW triggered SW priority group).*

- `AdcReadGroupApi` adds or removes the service `Adc_ReadGroup()` from the code.
- `AdcResultAlignment` selects alignment of ADC raw results in the ADC result buffer (left/right alignment).
  - `ADC_ALIGN_LEFT`: Left alignment.
  - `ADC_ALIGN_RIGHT`: Right alignment.
- `AdcVersionInfoApi` adds or removes the service `Adc_GetVersionInfo()` from the code.
- `AdcErrorCalloutFunction` is used to specify the error callout function name. The function is called on every error. The ASIL level of this function limits the ASIL level of the ADC driver.

Note: *This parameter must have a valid C function name; otherwise an error would occur in the configuration phase.*

- `AdcIncludeFile` is a list of filenames that will be included within the driver. Any application-specific symbol that is used by the ADC configuration (such as error callout function) should be included by configuring this parameter.

Note: *This parameter must have a unique filename with extension .h; otherwise some errors will occur in the configuration phase.*

4.2 AdcPublishedInformation configuration

This container has the following parameters, but these parameters cannot be changed. The parameters are used to publish common information about ADC driver.

- `AdcChannelValueSigned` informs whether the result value of the ADC driver has sign information (TRUE) or not (FALSE).

Note: *The `AdcChannelResultSigned` parameter specifies whether the result has signed information. Therefore, this `AdcChannelValueSigned` configuration parameter set to TRUE has no meaning.*

- `AdcGroupFirstChannelFixed` informs whether the first channel of an ADC channel group can be configured (FALSE) or is fixed (TRUE) to a value determined by the ADC HW unit.

Note: *This parameter is fixed to FALSE.*

- `AdcMaxChannelResolution` is maximum channel resolution in bits (does not specify accuracy).

Note: *This parameter is fixed to 12.*

4.3 AdcCustomFunction

This container has the following parameters to configure SelfDiag.

- `AdcSelfDiagApi` adds or removes the `Adc_StartDiagnosticFull()`, `Adc_GetDiagnosticResult()`, and `Adc_StartDiagnostic()` services from the code.
- `AdcVoltageDeviation` is a configuration parameter to set the deviation value for `SelfDiag`. This parameter specifies a value as a percentage relative to 2.5 V. For example, if the value of '10' is set for this

parameter, 0.25 V is an acceptable range. It means that the acceptable range for 2.5 V would be 2.25 to 2.75 V.

Note: *This parameter can be used when the AdcSelfDiagApi configuration parameter is enabled. Otherwise, this parameter is disabled.*

- `AdcDiagConvertTimeout` specifies the maximum count of checks to determine whether the conversion is finished. The `SelfDiag` function waits for the conversion done for self-diag. If conversion is not finished up to this count which this parameter shows, the function might check the result before the conversion is actually completed. In this case, the `SelfDiag` API does not work correctly. Therefore, this parameter should be greater than the actual conversion time. One checking process would take almost 10 cycles; however, note that this would depend on the compile option.

Note: *This parameter can be used when the AdcSelfDiagApi configuration parameter is enabled. Otherwise, this parameter is disabled.*

4.4 AdcPowerStateConfig configuration

This container has the following parameters to define a power state and initiate a callback when the power state is reached.

- `AdcPowerState` describes a different power state supported by the ADC HW. It should be defined by the HW supplier and used by the ADC driver 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 will always be configured.

Note: *Valid range is 0 (`ADC_FULL_POWER`) or 1 (`ADC_OFF_POWER`).*

- `AdcPowerStateReadyCbkRef` contains a reference to a power mode callback defined in a CDD or IoHwAbs component.

Note: *As there is no preparation period in the hardware feature, only synchronous power state transition mode is supported. Therefore, this parameter is not used by the ADC driver and is not being evaluated.*

4.5 AdcConfigSet configuration

This container contains the following configuration (parameters) and sub containers of the AUTOSAR ADC module:

- `AdcSupplyMonitorEnabledA` enables or disables the supply monitor for `AMUXBUS_A` (`amuxbus_a_mon`).

Note: *This parameter can be used when at least one of `AdcChannelId` is set to `AmuxbusA` in the `AdcConfigSet` container at least. Otherwise, this parameter is disabled.*

- `AdcSupplyMonitorLevelA` selects supply monitor level for `AMUXBUS_A`.
  - `ADC_SUPP_VREFL`: AMUXBUS_a_mon = VRL
  - `ADC_SUPP_VREFH`: AMUXBUS_a_mon = VRH

Note: *This parameter can be used when the `AdcSupplyMonitorEnabledA` configuration parameter is set to TRUE. Otherwise, this parameter is disabled.*

- `AdcSupplyMonitorEnabledB` enables or disables the supply monitor for `AMUXBUS_B` (`amuxbus_b_mon`).

Note: *This parameter can be used when at least one of `AdcChannelId` is set to `AmuxbusB` in the `AdcConfigSet` container. Otherwise, this parameter is disabled.*

- `AdcSupplyMonitorLevelB` selects the supply monitor level for `AMUXBUS_B`.
  - `ADC_SUPP_VREFL`: AMUXBUS_b_mon = VRL.
  - `ADC_SUPP_VREFH`: AMUXBUS_b_mon = VRH.

Note: *This parameter can be used when the `AdcSupplyMonitorEnabledB` configuration parameter is set to TRUE. Otherwise, this parameter is disabled.*

4.6 AdcHwUnit configuration

This container contains the HW unit configuration (parameters):

- `AdcClockSource` specifies the clock frequency for ADC hardware.

Note: *This parameter is not used by the ADC driver and therefore is not being evaluated. The clock source inputted into each ADC hardware unit is fixed. The ADC hardware does not support setting of the clock source. This feature can be made available using the MCU module.*

- `AdcHwUnitId` is the numeric ID of the HW unit.
  - `ADC_SAR_<PASS number>_<SAR ADC number>`: The range of this enumeration parameter depends on the hardware (for example, `ADC_SAR_0_0`, `ADC_SAR_0_1`, and so on).

Note: *PASS number is fixed to 0.*

- `AdcCoreAssignment` specifies the reference to `AdcCoreConfiguration` for the HwUnit core assignment.

Note: *`AdcCoreAssignment` must have the target's `AdcCoreConfiguration` setting. The same resource cannot be allocated to multiple cores.*

- `AdcPrescale` specifies optional ADC module-specific clock prescale factor, if supported by hardware.

Note: *This parameter is not used by the ADC driver and therefore is not being evaluated. The ADC hardware does not support setting of frequency. This feature can be made available using the MCU module.*

- `AdcDiagnoseEnable` is an optional parameter to enable or disable diagnostic reference for HW unit.

Note: *This parameter must be enabled when the channel in the group connects to the diagnostic reference through SARMUX, which is used in this hardware unit (see Diagnostic feature). Otherwise, this parameter should be disabled.*

- `AdcDiagnosticReference` is used to select diagnostic reference for hardware unit. The following diagnostic references are available:
  - `ADC_DIAG_VREFL`: Diagnostic reference is VREFL.
  - `ADC_DIAG_VREFH_1DIV8`: Diagnostic reference is VREFH_1DIV8.
  - `ADC_DIAG_VREFH_2DIV8`: Diagnostic reference is VREFH_2DIV8.
  - `ADC_DIAG_VREFH_3DIV8`: Diagnostic reference is VREFH_3DIV8.
  - `ADC_DIAG_VREFH_4DIV8`: Diagnostic reference is VREFH_4DIV8.
  - `ADC_DIAG_VREFH_5DIV8`: Diagnostic reference is VREFH_5DIV8.
  - `ADC_DIAG_VREFH_6DIV8`: Diagnostic reference is VREFH_6DIV8.
  - `ADC_DIAG_VREFH_7DIV8`: Diagnostic reference is VREFH_7DIV8.
  - `ADC_DIAG_VREFH`: Diagnostic reference is VREFH.
  - `ADC_DIAG_VREFX`: Diagnostic reference is VREFH.
  - `ADC_DIAG_VBG`: Diagnostic reference is VBG.
  - `ADC_DIAG_VIN1`: Diagnostic reference is VIN1.
  - `ADC_DIAG_VIN2`: Diagnostic reference is VIN2.
  - `ADC_DIAG_VIN3`: Diagnostic reference is VIN3.
  - `ADC_DIAG_I_SOURCE`: Diagnostic reference is I_SOURCE.
  - `ADC_DIAG_I_SINK`: Diagnostic reference is I_SINK.

Note: *This parameter can be used when `AdcDiagnoseEnable` is enabled. Otherwise, this parameter is disabled.*

- `AdcPreconditionCycle` is used to specify the duration of preconditioning in SAR clock cycles.
- `AdcSarMux1ConnectToAdc0`, `AdcSarMux2ConnectToAdc0`, and `AdcSarMux3ConnectToAdc0` determine if SARMUX1, SARMUX2, and SARMUX3 are connected to the hardware unit of SAR ADC0.
- `AdcSarMux1DiagnoseEnable`, `AdcSarMux2DiagnoseEnable`, and `AdcSarMux3DiagnoseEnable` enable or disable the diagnostic reference for SARMUX1, SARMUX2, and SARMUX3.

Note: *These parameters can be used when SARMUXes other than SAR ADC0 are connected to SAR ADC0.*

These parameters must be enabled when the group that included some input signals from each SARMUXes uses the diagnostic reference. Otherwise, these parameters should be disabled.

- `AdcSarMux1DiagnosticReference`, `AdcSarMux2DiagnosticReference`, and `AdcSarMux3DiagnosticReference` select the diagnostic reference output for SARMUX1, SARMUX2, and SARMUX3. The available diagnostic reference outputs are the same as the `AdcDiagnosticReference` configuration parameter.

Note: *These parameters can be used when the AdcSarMux1DiagnoseEnable, AdcSarMux2DiagnoseEnable, and AdcSarMux3DiagnoseEnable configuration parameters are enabled. Otherwise, these parameters are disabled.*

4.7 AdcChannel configuration

This container contains the channel configuration (parameters):

- `AdcChannelConvTime` is the configuration of conversion time, that is, the time during which the analog value is converted into digital (in clock cycles) for each channel.

Note: Valid range is only 14. HW sets it from 13 to 15, although the range of this parameter is fixed to 14. The conversion time depends on the hardware setting which is used; it cannot be specified by SW explicitly.

- `AdcChannelHighLimit` is the high limit used for limit checking.

Note: If the `AdcResultAlignment` configuration parameter is set to `ADC_ALIGN_LEFT`, the configured value which is shifted to 4 bits towards the left needs to be set. If the `AdcChannelResultSigned` configuration parameter is enabled, signed value needs to be set.

This parameter can be used when the `AdcChannelLimitCheck` configuration parameter is enabled. Otherwise, this parameter is disabled.

- `AdcUseExternalMultiplexe`r is a configuration that indicates whether to use the output for the external multiplexer.
  If `AdcUseExternalMultiplexer` is enabled, ADC outputs the value of bit 10:8 of `AdcChannelId` to external multiplexer while the channel is being converted.
- `AdcChannelId` defines the assignment of the channel to the ADC physical channel. This parameter is the symbolic name to be used as argument for certain APIs. This symbolic name allows accessing channel data. This value will be assigned to the symbolic name derived from the short name of the `AdcChannel` container.
  If `AdcUseExternalMultiplexer` is enabled, 10: 8 bit indicates the value to be output to the external multiplexer.

Note: Actual numeric of ADC channel ID can be calculated by using the following formula; the available channel depends on the device used: ADC channel ID = `n + (64 * m) + (256 * e)`
Where,
m is the SAR ADC number or SARMUX number
n is the address of the analog signal
e is the output of signal pattern to external multiplexer (0 – 7)
*For example, ADC channel ID for AN31 on ADC_SAR_0_1 or SARMUX1 and output to external multiplexer is 7 is 1887 (31 + 64 * 1 + 256 * 7).*

Table 2 Available ADC channels

AdcChannelId	AdcHwUnitId	Description
0 .. 31 + (64 * m) + (256 * e)	ADC_SAR_x_m	AN0..31, select corresponding analog input
32 + (64 * m) + (256 * e)	ADC_SAR_x_m	Internal special signal (Vmotor, select motor input)
33 + (64 * m)	ADC_SAR_x_m	Internal special signal (Vaux, select auxiliary input)
34 + (64 * m)	ADC_SAR_x_m	Internal special signal (AmuxbusA)
35 + (64 * m)	ADC_SAR_x_m	Internal special signal (AmuxbusB)

AdcChannelId	AdcHwUnitId	Description
36 + (64 * m)	ADC_SAR_x_m	Internal special signal (Vccd)
37 + (64 * m)	ADC_SAR_x_m	Internal special signal (Vdda)
38 + (64 * m)	ADC_SAR_x_m	Internal special signal (Vbg, bandgap voltage from SRSS)
39 + (64 * m)	ADC_SAR_x_m	Internal special signal (Vtemp, select temperature sensor)
62 + (64 * m)	ADC_SAR_x_m	VrefL
63 + (64 * m)	ADC_SAR_x_m	VrefH

“x” represents PASS number and “m” represents SAR ADC number and “e” represents output of signal pattern to external multiplexer.

- AdcChannelLimitCheck enables or disables limit checking for an ADC channel.

Note: *This parameter can be used when the AdcEnableLimitCheck configuration parameter is enabled. Otherwise, this parameter is disabled.*

- AdcChannelLowLimit is the low limit used for limit checking.

Note: *If the AdcResultAlignment configuration parameter is set to ADC_ALIGN_LEFT, the configured value which is shifted to 4 bits towards the left needs to be set. If the AdcChannelResultSigned configuration parameter is enabled, signed value needs to be set.*

This parameter can be used when the AdcChannelLimitCheck configuration parameter is enabled. Otherwise, this parameter is disabled.

- AdcChannelPulseDetect enables or disables the range detection of an ADC channel.

Note: *This parameter can be used when the AdcChannelLimitCheck configuration parameter is enabled. Otherwise, this parameter is disabled.*

- AdcChannelRangeSelect defines which conversion values are considered related to the borders defined with AdcChannelLowLimit and AdcChannelHighLimit configuration parameters.
  - ADC_RANGE_ALWAYS: Complete range – independent of channel limit settings.
  - ADC_RANGE_BETWEEN: Range between low and high limits with the high limit value included.
  - ADC_RANGE_NOT_BETWEEN: Range above high limit or below low limit with the low limit value included.
  - ADC_RANGE_NOT_OVER_HIGH: Range below high limit with the 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 low limit, with the low limit value included.

Note: *This parameter can be used when the AdcChannelLimitCheck configuration parameter is enabled. Otherwise, this parameter is disabled.*

- AdcChannelRefVoltsrcHigh is the upper reference voltage source for each channel.

Note: *This parameter is not used by the ADC driver and therefore is not being evaluated.*

- `AdcChannelRefVoltsrcLow` is the lower reference voltage source for each channel.

Note: *This parameter is not used by the ADC driver and therefore is not being evaluated.*

- `AdcChannelResolution` is the channel resolution in bits.

Note: *Only 12-bit resolution is available.*

- `AdcChannelSampTime` is the configuration of sampling time, that is, the time during which the value is sampled, (in clock cycles) for each ADC channel group.
- `AdcChannelResultSigned` determines the conversion result data is signed or unsigned. Signed value is used if the parameter is TRUE, otherwise unsigned value is used.

Note: *This parameter can be used when the `AdcResultAlignment` configuration parameter is set to `ADC_ALIGN_RIGHT`. Otherwise, this parameter is disabled.*

In case of unsigned value, conversion data is effectively a 12-bit value zero-extended (for example, VrefL: 0x0000, VrefH/2: 0x0800, VrefH: 0x0FFF). In case of signed value, the MSB (bit 11) of conversion data is inverted and sign extended (for example, VrefL: 0xF800, VrefH/2: 0x0000, VrefH: 0x07FF).

- `AdcChannelPulsePositiveCount` is the default counter value for the events resulting from range detection. If the resulting events occur, the counter decreases. When this counter is 0, a relevant interruption will happen and the counter will reset to the default value configured for `AdcChannelPulsePositiveCount`. The detection settings of the resulting events follow `AdcChannelRangeSelect`.

Note: *This parameter can be used when the AdcChannelPulseDetect configuration parameter is enabled. Otherwise, this parameter is disabled.*

- `AdcChannelPulseNegativeCount` is the default counter value for the events resulting from range detection. If the resulting events do not occur, the counter decreases. When this counter is 0, the counter will reset to the default value configured for `AdcChannelPulseNegativeCount`. The detection settings of the resulting events follow `AdcChannelRangeSelect`.

Note: *This parameter can be used when the `AdcChannelPulseDetect` configuration parameter is enabled. Otherwise, this parameter is disabled.*

- `AdcDiagnosisMode` specifies how verification is done for SelfDiag.
  - `ADC_DIAGNOSIS_NO`: No SelfDiag. The SelfDiag APIs return `E_OK` for the specified channel when the API is called in this mode.
  - `ADC_DIAGNOSIS_SIMPLE`: Connectivity would be verified by doing and checking the conversions of the channel related to SelfDiag 2 times.
  - `ADC_DIAGNOSIS_FULL`: Correctness would be verified through conversions of the channel related to SelfDiag for 9 times.

Note: *This parameter can be used when the AdcSelfDiagApi configuration parameter is enabled. Otherwise, this parameter is disabled.*

4.8 AdcGroup configuration

This container contains the group configuration (parameters):

- `AdcGroupAccessMode` is the type of access mode to group conversion results.
  - `ADC_ACCESS_MODE_SINGLE`: Single value access mode.
  - `ADC_ACCESS_MODE_STREAMING`: Streaming access mode.
- `AdcGroupConversionMode` is the type of conversion mode supported by the driver.
  - `ADC_CONV_MODE_CONTINUOUS`: Conversions of an ADC channel group are performed continuously after a software API call (start). The run automatically (no additional software or hardware trigger is needed).
  - `ADC_CONV_MODE_ONESHOT`: The conversion of an ADC channel group is performed once after a trigger.
- `AdcGroupId` is the numeric ID of the group. This parameter is the symbolic name to be used as argument for all APIs. This symbolic name allows accessing channel group data. This value will be assigned to the symbolic name derived of the short name of the `AdcGroup` container. The configured values are zero-based and consecutively numbered.
- `AdcGroupPriority` is the priority level of the group.

Note: *Groups with higher values are converted before groups with lower numbers.*

Note: *If the AdcPriorityImplementation parameter is set to ADC_PRIORITY_HW, the AdcGroupPriority parameter should be set.*

For details, see SelfDiag feature.

- `AdcGroupReplacement` is the replacement mechanism used on ADC group level if a group conversion is interrupted by a group that has a higher priority. There are three types of behavior available and one of them can be specified at a time for each ADC channel group.
  - `ADC_GROUP_REPL_ABORT_RESTART`: Abort/Restart mechanism is used on group level, if a group is interrupted by a higher priority group. The complete conversion round of the interrupted group (all group channels) is restarted after the higher priority group conversion is finished. If the group is configured in streaming access mode, only the results of the interrupted conversion round are discarded. Results of previous conversion rounds which are already written to the result buffer are not affected. Immediately abort the ongoing acquisition and on return restart the group scan from the first ADC channel of the group.
  - `ADC_GROUP_REPL_SUSPEND_RESUME`: Suspend/Resume mechanism is used on group level, if a group is interrupted by a higher priority group. The conversion 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. Before preempting, complete the ongoing acquisition and on return resume the group scan starting with the next channel.
  - `ADC_GROUP_REPL_ABORT_RESUME`: Abort/Resume mechanism is used on group level, if a group is interrupted by a higher priority group. The conversion 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. Immediately abort the ongoing acquisition and on return resume the group scan starting with the aborted channel.

Note: *ADC_GROUP_REPL_ABORT_RESUME is vendor specific replacement function.*

- AdcGroupTriggSrc is the type of source event that starts a group conversion.
  - ADC_TRIGG_SRC_HW: Group is triggered by a hardware event.
  - ADC_TRIGG_SRC_SW: Group is triggered by a software API call.
- AdcHwTrigSignal configures the edge of the hardware trigger signal on which the driver should react, that is, start the conversion.
  - ADC_HW_TRIG_BOTH_EDGES: React on both edges of the hardware trigger signal (only if supported by the ADC hardware).
  - ADC_HW_TRIG_FALLING_EDGE: React on the falling edge of the hardware trigger signal (only if supported by the ADC hardware).
  - ADC_HW_TRIG_RISING_EDGE: React on the rising edge of the hardware trigger signal (only if supported by the ADC hardware).

Note: *This parameter is not used by the ADC driver and therefore is not being evaluated. The ADC hardware does not support specifying the edge of the hardware signal. A similar feature is supported by other modules (for example, PORT driver).*

- AdcHwTrigTimer is the reload value of the embedded timer of the ADC module.

Note: *This parameter is not used by the ADC driver and therefore is not being evaluated. The ADC hardware does not have the embedded timer. A similar feature is supported by other modules (for example, PWM driver).*

- AdcNotification is the callback function for this group.

Note: *This parameter can be used when the AdcInterruptMode configuration parameter is enabled. Otherwise, this parameter is disabled.*

- AdcStreamingBufferMode configures the streaming buffer as “linear buffer” (the ADC driver stops the conversion as soon as the stream buffer is full) or “ring buffer” (wraps around if the end of the stream buffer is 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.
  - ADC_STREAM_BUFFER_LINEAR: The ADC driver stops the conversion as soon as the stream buffer is full (number of samples reached).
- AdcStreamingNumSamples is the number of ADC values to be acquired per channel in streaming access mode.

Note: *In single access mode, this parameter assumes a value of 1, since only one sample per channel is processed.*

- AdcGroupDefinition assigns AdcChannels to an AdcGroup.
- AdcFirstLogicalChannel is the first logical channel of the group, that is, the logical channel that should be configured in the HW for the first channel in AdcGroupDefinition. This logical channel may be triggered by a configured HW trigger.
- AdcGroupHwTriggSrc determines hardware trigger event of the group.

  - ADC_HWTRIGG_SRC_TCPWM: Trigger from corresponding TCPWM channel.
  - ADC_HWTRIGG_SRC_GENERIC0: Trigger from generic trigger input 0.
  - ADC_HWTRIGG_SRC_GENERIC1: Trigger from generic trigger input 1.
  - ADC_HWTRIGG_SRC_GENERIC2: Trigger from generic trigger input 2.
  - ADC_HWTRIGG_SRC_GENERIC3: Trigger from generic trigger input 3.
  - ADC_HWTRIGG_SRC_GENERIC4: Trigger from generic trigger input 4.

Note: *This parameter can be used when the AdcGroupTriggSrc configuration parameter is set to ADC_TRIGG_SRC_HW. Also, to set ADC_HWTRIGG_SRC_GENERIC0/1/2/3/4, it is necessary to configure the container of AdcGenericHWTriggerSelectConfiguration.*

- AdcInterruptMode enables or disables interrupt mode for each group.
- AdcUseDma enables or disables the DMA for each group.

Note: *This parameter cannot be enabled when this group contains a channel in which AdcChannelPulseDetect is enabled.*

- AdcUseAlternateCalibration determines whether to use alternate calibration values for each group.
- AdcSampleMode selects the sampling mode for group (see Port selection for further information).
  - ADC_SAMPLE_NORMAL: PRECOND_MODE is OFF and OVERLAP_DIAG is OFF.
  - ADC_SAMPLE_NORMAL_HALF: PRECOND_MODE is OFF and OVERLAP_DIAG is HALF.
  - ADC_SAMPLE_NORMAL_FULL: PRECOND_MODE is OFF and OVERLAP_DIAG is FULL.
  - ADC_SAMPLE_NORMAL_MUX: PRECOND_MODE is OFF and OVERLAP_DIAG is MUX_DIAG.
  - ADC_SAMPLE_VREFL: PRECOND_MODE is VREFL and OVERLAP_DIAG is OFF.
  - ADC_SAMPLE_VREFL_HALF: PRECOND_MODE is VREFL and OVERLAP_DIAG is HALF.
  - ADC_SAMPLE_VREFL_FULL: PRECOND_MODE is VREFL and OVERLAP_DIAG is FULL.
  - ADC_SAMPLE_VREFL_MUX: PRECOND_MODE is VREFL and OVERLAP_DIAG is MUX_DIAG.
  - ADC_SAMPLE_VREFH: PRECOND_MODE is VREFH and OVERLAP_DIAG is OFF.
  - ADC_SAMPLE_VREFH_HALF: PRECOND_MODE is VREFH and OVERLAP_DIAG is HALF
  - ADC_SAMPLE_VREFH_FULL: PRECOND_MODE is VREFH and OVERLAP_DIAG is FULL
  - ADC_SAMPLE_VREFH_MUX: PRECOND_MODE is VREFH and OVERLAP_DIAG is MUX_DIAG.
  - ADC_SAMPLE_DIAG: PRECOND_MODE is DIAG and OVERLAP_DIAG is OFF.
  - ADC_SAMPLE_DIAG_MUX: PRECOND_MODE is DIAG and OVERLAP_DIAG is MUX_DIAG.
- AdcUseRedundancy enables or disables result buffer redundancy. To enable AdcUseRedundancy, it is necessary to prepare twice the buffer size specified by Adc_SetupResultBuffer(). This is to secure the redundancy buffer after the result buffer.
- AdcLimitCheckNotification is limit check callback function for this group.

Note: *This parameter can be used when AdcChannelLimitCheck and AdcNotification configuration parameters are enabled. Otherwise, this parameter is disabled.*

- AdcUseDynamicAllocate enables or disables dynamic allocation for this group. Groups with AdcUseDynamicAllocate enabled share the same logical channels.

Note: *This parameter can be used when the `AdcGroupTriggSrc` configuration parameter is `ADC_TRIGG_SRC_SW`. Otherwise, this parameter is disabled. Groups with `AdcUseDynamicAllocate` enabled must have the same `AdcFirstLogicalChannel`.*

- `AdcSWPriority` indicates the priority of groups with `AcUseDynamicAllocate` enabled. 255 is the highest priority. The same priority as other groups are also allowed.

Note: *This parameter can be used when the `AdcUseDynamicAllocate` configuration parameter is enabled. Otherwise, this parameter is disabled.*

4.9 AdcGenericHWTriggerSelectConfiguration

This container contains the generic hardware trigger configuration (parameters):

- `AdcGroupGenericHwTriggSrc` has the value of `ADC_HWTRIGG_SRC_GENERIC 0 – 4`. The trigger configured here is used to configure `AdcGroupHwTriggSrc`.
- `AdcGenericTriggerSelect` selects the generic trigger for SAR generic trigger input.

4.10 AdcMulticore

`AdcMulticore` defines the multicore functional configuration of the ADC driver.

- `AdcCoreConsistencyCheckEnable` enables core consistency check during runtime. If enabled, the ADC function checks if the provided parameter (channel) is allowed on the current core.

Note: *Development error detect is enabled in ADC driver to enable this parameter.*

- `AdcGetCoreIdFunction` specifies the API to be called to get the core ID, for example, `GetCoreId()`

Note: *`AdcGetCoreIdFunction` must be a valid C function name.*

- `AdcMasterCoreReference` specifies the reference to the master core configuration.

Note: *AdcMasterCoreReference must have the target's AdcCoreConfiguration setting.*

4.11 AdcCoreConfiguration

`AdcCoreConfiguration` defines the core configuration of the ADC driver. `AdcCoreConfiguration` can also be configured without ADC channel assignment.

- `AdcCoreConfigurationId` is a zero-based, consecutive integer value. This is used as a logical core ID.

Note: *AdcCoreConfigurationId must be unique across AdcCoreConfiguration.*

- `AdcCoreId` is a core ID assigned to ADC HwUnits. This ID is returned from the configured `AdcGetCoreIdFunction` execution to identify the executing core.

Note: *`AdcCoreId` must be unique across `AdcCoreConfiguration`.*

The combination of `AdcCoreConfigurationId` and `AdcCoreId` must be unique across `AdcCoreConfiguration`.

5 Functional description

5.1 Module function

The ADC driver offers an API interface for enabling hardware triggered one-shot conversions and starting a software triggered one-shot or continuous conversion of all channels within a channel group. A channel group may consist of one or more ADC channels. Continuous or one-shot group conversions can be stopped by an API call (all channels within the channel group will be stopped). A start/stop or enable/disable operation is always performed on all channels of a channel group. However, starting or stopping an individual channel can be achieved by specifying a group consisting of exactly one channel.

The conversion result for a channel can be read as soon as the conversion is completed. Each ADC group that is in single access mode consists of a single buffer for each channel. The buffer is overwritten as soon as the next conversion result for that specific channel is available. The result buffer which is used in streaming access mode can store an arbitrary number of conversion results per channel. To synchronize read access, notification functions for every group can be enabled (and disabled) during runtime.

Two or more groups software triggered can be started at the same time by calling an API `Adc_StartGroupConversion()` multiple times with different groups. The requests will be accepted and converted after all groups with higher priority have finished conversion.

The ADC driver also offers and API to temporarily disable or enable individual channels in each group, a function to double hold the conversion result, and a function to switch the value of threshold dynamically.

5.2 Inclusion

The file *Adc.h* includes all necessary external identifiers. Thus, your application only needs to include *Adc.h* to make all API functions and data types available.

5.3 Initialization and de-initialization

The ADC driver provides functions for initialization and de-initialization. Initialization by calling an API `Adc_Init()` is mandatory once on each core before the use of ADC driver.

Note: *This ADC driver supports post-build-time configuration, thus different configuration set pointers can be passed to the `Adc_Init()` function.*
`Adc_Init()` must be called on the master core before any cores are initialized. If `Adc_Init()` is called on the satellite core, the master core must be already initialized. The same configuration set must be specified on all cores during initialization. If no HwUnit is assigned to the satellite core, `Adc_Init()` is not required on that core.

The driver is de-initialized by using `Adc_DeInit()` once on each core after use. The function resets all ADC registers to their hardware power-on-reset values. Usage of `Adc_DeInit()` is prohibited while any conversion is ongoing.

Note: *`Adc_DeInit()` must be called on the master core after all satellite cores are de-initialized. If `Adc_DeInit()` is called on the satellite core, the master core must be already initialized. The integrated system must prevent other cores from calling the ADC API while `Adc_DeInit()` is being called.*

5.4 Runtime reconfiguration

All configuration parameters cannot be changed at runtime except the `AdcChannelSampTime`, `AdcChannelHighLimit` and `AdcChannelLowLimit`. These parameters are changeable by calling an API `Adc_ChangeSamplingTime()` or `Adc_SelectChannelThreshold()`. The option to change these parameters by using `Adc_ChangeSamplingTime()` or `Adc_SelectChannelThreshold()` is only available when the ADC channel group to which the ADC channel belongs to is not running.

Example using the `Adc_ChangeSamplingTime()` and `Adc_SelectChannelThreshold()`:

```
#include "Adc.h"
/* ... */
/* initialize ADC Driver */
/* .... */
/* change sampling time */
Adc_ChangeSamplingTime(AdcConf_AdcGroup_AdcGroup_MY_GROUP,
AdcConf_AdcChannel_AdcChannel_MY_CHANNEL, my_sampling_time);


Adc_SelectChannelThreshold(AdcConf_AdcGroup_AdcGroup_MY_GROUP,
AdcConf_AdcChannel_AdcChannel_MY_CHANNEL, my_upper_limit_value,
my_lower_limit_value);
```

5.5 Channels and channel groups

A channel represents a single analog input signal. The value (not conversion result) of a single channel is of the type `Adc_ChannelType`. They represent ADC physical channels. These values correspond to GPIO or internal special signals. GPIO signals are connected to the hardware analog input pins directly, whereas internal signals are not connected to the pins (See Analog input signals for details). If diagnostic reference is used, the selected diagnostic reference signal is sampled instead of analog signal.

A channel group consists of several channels. Its value (not conversion result) is of the type `Adc_GroupType`.

Starting analog-to-digital conversion for a channel is only possible if the channel is included in a channel group.

The symbolic names of the groups will be used for the API calls.

Note: *The value of a symbolic name is implementation/architecture optimized. The configured groups are zero based and consecutively numbered because this index is used by the implementation for a fast array access. The configured channels would be assigned to groups in ascending order. There is no requirement for the channel numbers in a group to be consecutive; gaps between the channel numbers are possible.*

5.6 Start/stop SW-triggered group conversion

For accessing SW triggered group conversion, the ADC driver provides the following services:

- `void Adc_StartGroupConversion(Adc_GroupType Group);`
- `void Adc_StopGroupConversion(Adc_GroupType Group);`

`Adc_StartGroupConversion()` starts the conversion of all channels inside the specified group, if the group is configured for SW triggered group. Depending on the group configuration, one-shot or continuous conversion will be started.

If there is a conversion of some other group that is already running, the request will be accepted but will not start. The queued conversion requests are executed based on priority (highest first). A high priority group can interrupt an ongoing low priority group. The interrupted group will be aborted or suspended, and this can be selected using the `AdcGroupReplacement` configuration parameter.

After the newly processed group is completed, the interrupted group might be restarted from a first channel in the channel group. Otherwise, it might be resumed from the aborted channel or the channel next to the aborted channel. It depends on the setting in the `AdcGroupReplacement` configuration parameter in the ADC channel group.

If dynamic allocate is enabled and all groups of `AdcUseDyamicAllocate` are enabled, ADC starts conversion for each group depending upon `AdcSWPriority` in SW priority order. If dynamic allocate is enabled and one group of `AdcUseDynamicAllocate` is enabled and the other group of `AdcUseDyanamicAllocate` is disabled, ADC starts conversion for each group depending on `AdcGroupPriority` in HW priority order.

`Adc_StartGroupConversion()` is not allowed on an ongoing group. However, it can be called in `ADC_STREAM_COMPLETED` when the channel group is configured for streaming access and linear buffer mode or for SW trigger and one-shot conversion mode.

`Adc_StopGroupConversion()` directly forces a group to stop if the conversion is running. If the requested group was started but is not running now (pending by hardware priority mechanism), then it will be stopped.

`Adc_StopGroupConversion()` disables notification.

`Adc_StopGroupConversion()` on an idle group is not allowed.

Note: *In a continuous mode of SW triggered group, the `AdcGroupAccessMode` configuration parameter controls two implementations with different advantages and drawbacks.*
If the `AdcGroupAccessMode` configuration parameter is configured for `ADC_ACCESS_MODE_SINGLE`, the feature is implemented in a way that the conversion of the group is triggered in a continuous manner by the hardware itself. This implementation minimizes interrupt load. Depending on timing conditions, the conversion result that triggered calling the group notification might already be overwritten at the time when conversion results are actually read out within the interrupt or in a later call to `Adc_ReadGroup()`. In addition, if ADC conversion is always completed faster than interrupt processing, it might cause unexpected behavior (for example, stack).
If the `AdcGroupAccessMode` configuration parameter is configured for `ADC_ACCESS_MODE_STREAMING`, the feature is implemented in a way that the conversion of the group is triggered in a continuous manner by the software. The next conversion is triggered by the interrupt of conversion completed. The next conversion does not start before ADC driver transfers the conversion result from the register to the result buffer; so, this implementation assures accurate conversion results.

5.7 Enable or disable hardware-triggered group conversion

The ADC driver provides the following services to access hardware triggered groups:

- `void Adc_EnableHardwareTrigger(Adc_GroupType Group);`
- `void Adc_DisableHardwareTrigger(Adc_GroupType Group);`
- `void Adc_EnableHwTrigger(Adc_GroupType Group, Adc_HwTriggerTimerType SelectTrigger);`
- `void Adc_DisableHwTrigger(Adc_GroupType Group);`

`Adc_EnableHardwareTrigger()` opens the hardware trigger window (that is, if the hardware trigger event occurs the conversion starts). Each hardware trigger event causes conversion of all channels of the group resulting in one conversion result per channel.

The conversion mode for hardware triggered groups needs to be one-shot conversion.

Notification remains active (if enabled) after conversion is done.

The hardware trigger window will be closed (disabled) using `Adc_DisableHardwareTrigger()`. This also disables the notification for this group.

`Adc_EnableHardwareTrigger()` is not allowed on an ongoing group.

`Adc_DisableHardwareTrigger()` closes the hardware trigger window (that is, if the hardware trigger event comes the conversion does not start).

`Adc_DisableHardwareTrigger()` disables notification.

`Adc_DisableHardwareTrigger()` on an idle group is not allowed.

Hardware triggers will be ignored if they appear before the previously triggered conversion has finished. Therefore, make sure that the hardware triggering period is longer than the conversion time of the whole group.

`Adc_EnableHwTrigger()` is almost the same as `Adc_EnableHardwareTrigger()`. In addition, SW can specify hardware trigger source with argument.

`DisableHwTrigger()` is the same as `Adc_DisableHardwareTrigger()`.

5.8 Read services

The ADC driver provides the following API services for reading the last valid conversion results of a group:

- `Std_ReturnType Adc_ReadGroup(Adc_GroupType Group, Adc_ValueGroupType* DataBufferPtr);`
- `Adc_StreamNumSampleType Adc_GetStreamLastPointer(Adc_GroupType Group, Adc_ValueGroupType** PtrToSamplePtr);`
- `Adc_DataReadType Adc_ReadChannelValue(Adc_GroupType Group, Adc_ChannelType Channel, uint16* Adc_ChannelDataPtr);`

`Adc_ReadGroup()` copies the results (if any) of the last conversion of all channels to the provided buffer and returns E_OK if the result is available. The result value is copied from the result buffer, if interrupt mode is used, DMA is enabled, or both. Otherwise, it is copied from the dedicated register.

If `Adc_ReadGroup()` is called while the ADC channel group conversion is not finished, the API will return `E_NOT_OK` without any actions.

`Adc_GetStreamLastPointer()` returns the number of samples that have been converted per channel and a result buffer pointer to the last valid conversion result of the requested group.

Asking for a result buffer pointer while there are no conversion results available will return zero and set a NULL pointer to the passed parameter.

`Adc_ReadChannelValue()` copies the result of the last conversion of one channel to the provided buffer and returns `ADC_DATA_UNREAD` if the result is not yet read. The result value is copied from the result buffer, if interrupt mode is used, DMA is enabled, or both. Otherwise, the result is copied from the dedicated register.

If `AdcUseRedundancy` is enabled, the redundancy buffer and result buffers are compared in the above functions. If there is a mismatch, these functions report an error code `ADC_E_REDUNDANCY_ERROR`.

5.9 Notification

To enable or disable the user defined notification functions, the ADC driver provides the following services:

- `void Adc_EnableGroupNotification(Adc_GroupType Group);`
- `void Adc_DisableGroupNotification(Adc_GroupType Group);`

These functions enable or disable the group notification during runtime. The user defined notification function will be called if the analog-to-digital conversion of the group is finished.

The notification function will be called every time the analog-to-digital conversion has finished once for all channels in the group.

The callback function that is specified by `AdcNotification` is called for each ADC channel group if it is used in interrupt mode.

If `AdcChannelLimitCheck` is enabled and falls within the condition of limit check, the function specified by `AdcLimitCheckNotification` will be also called.

The group notification will be automatically disabled if the group is explicitly stopped (calling by `Adc_StopGroupConversion()` and `Adc_DisableHardwareTrigger()`).

Group notification can be enabled before the group is started and while the group is running. This means the following is allowed:

```
Adc_EnableGroupNotification(MY_ADC_GROUP_1);
Adc_StartGroupConversion(MY_ADC_GROUP_1);
```

The function at the time of limit check has the following format:

```
typedef P2FUNC(void, TYPEDEF, Adc_LCNFctPtrType) (VAR(uint32, AUTOMATIC)
LimitCheckState);
```

The argument `LimitCheckState` indicates the channel in the group that corresponds to the limit check condition. Each bit corresponds to each channel, and when it is 1, it means that the limit check conditions are satisfied.

0 bit (LSb) represents the smallest channel number in the group. 31 bit (MSb) represents the 32nd channel in the group.

Note: *Notification can be disabled by calling Adc_DisableGroupNotification as mentioned in this chapter. However, it would not work for notifications that have already been handled, if Adc_DisableGroupNotification is called from a higher priority interruption during a few cycles*

before the user-defined notification function is called after the ADC group conversion. In this situation, the user-defined function might be called even though it is already disabled. To prevent the unexpected call, notification should not be enabled before starting the ADC group conversion (that is, before calling Adc_StartGroupConversion, Adc_EnableHardwareTrigger, and Adc_EnableHwTrigger.

5.10 Limit checking

To activate the limit checking feature in general, you need to set the `AdcEnableLimitCheck` configuration parameter to TRUE.

The result is notified by the callback function specified by `AdcLimitCheckNotification`.

The ADC conversion result is stored in the result buffer regardless of the limit check result.

5.11 Power management

The ADC driver supports power management feature to reduce the electricity related to ADC hardware. The following APIs are available to manage power. This feature is not vendor-specific and it comes from AUTOSAR specification. If power state needs to be changed, `Adc_PreparePowerState()` should be called before calling `Adc_SetPowerState()`.

- `Std_ReturnType Adc_GetCurrentPowerState(Adc_PowerStateType * CurrentPowerState, Adc_PowerStateRequestResultType * Result);`
- `Std_ReturnType Adc_GetTargetPowerState(Adc_PowerStateType * TargetPowerState, Adc_PowerStateRequestResultType * Result);`
- `Std_ReturnType Adc_PreparePowerState(Adc_PowerStateType PowerState, Adc_PowerStateRequestResultType * Result);`
- `Std_ReturnType Adc_SetPowerState(Adc_PowerStateRequestResultType * Result);`

Note: Asynchronous power state transition mode is not supported. It means that an API `Adc_Main_PowerTransitionManager()` is implemented as null function because there is no preparation period in the hardware feature.

5.12 Interrupt and polling mode

The ADC driver supports not only interrupt mode but also polling mode. When interrupt mode is used, an interruption occurs after the ADC conversion of all ADC channels in a group is completed (if DMA is disabled) or DMA transfer is completed (if DMA is enabled). The ADC driver copies a result into a result buffer in an interruption (if DMA is disabled). If DMA is enabled, the result is not copied in the interruption as the DW hardware copies the result.

When polling mode is used, an interruption never occurs after the ADC group conversion is completed.

If polling mode is used with no DMA, the conversion result would be read from a register. If polling mode is used with DMA, it would be read from a result buffer instead.

The `AdcInterruptMode` configuration parameter is intended to specify the interrupt or polling mode that is used for each ADC channel group. If DMA is enabled, a result is copied to a result buffer via DMA hardware instead of an interrupt handler.

Note: Only single access mode is supported in the polling mode. Streaming access mode cannot be supported in polling mode.

5.13 Triggered by HW

HW trigger is used to trigger an ADC channel group conversion. The `AdcGroupTriggSrc` configuration parameter needs to be set to `ADC_TRIGG_SRC_HW`, when HW trigger is used,

A group with trigger source hardware, whose trigger was enabled with `Adc_EnableHardwareTrigger()` or `Adc_EnableHwTrigger()`, will execute the channel group conversions whenever a trigger event occurs.

The HW trigger connects to the first logical channel of the group. This connection needs to be established in advance. This connection is supported by one-to-one trigger group or multiplexer-based trigger group. This is selectable via the `AdcGroupHwTriggSrc` configuration parameter. One-to-one trigger group is used if an ADC logical channel is connected from corresponding TCPWM directly. Multiplexer-based trigger group is used if an ADC channel is connected from HW trigger via generic input. A kind of HW trigger which is connected via generic input depends on which device is used.

In addition, if multiplexer-based trigger group is used, generic input needs to be specified via the `AdcGenericTriggerSelect` configuration parameter.

Note: *Trigger group needs to be configured by PORT driver but not ADC driver.*

5.14 DMA transfer

DMA can be used to copy the conversion result from the register to the result buffer, if the `AdcUseDma` configuration parameter is enabled in an ADC channel group. The transfer would reduce CPU load as it is done without CPU assistance. In addition, if `AdcUseRedundancy` is valid, DMA will copy data from register to result buffer and redundancy result buffer.

DW trigger connects to a last channel of the group to start DMA transfer. This connection needs to be established in advance. DMA transfer will be started immediately after all of ADC channels in a group are completed (if limit check is disabled). If limit check is enabled, it will be started immediately after all of ADC channels in a group are completed. This connection is supported by one-to-one trigger group or multiplexer-based trigger group. One-to-one trigger group is used if an ADC channel connects to DW directly. Multiplexer-based trigger group is used if it connects to DW via generic output.

The group status is changed to `ADC_ERROR` after a DMA error is detected. DEM will be reported (if configured) when `Adc_GetGroupStatus()` is called in this case. `Adc_StopGroupConversion()` or `Adc_DisableHardwareTrigger()` should be called before restarting the ADC channel group in which the error is detected.

Note: *Trigger group needs to be configured by the PORT driver, but not the ADC driver.*

Note: *The ADC driver's environment must guarantee that DMA is enabled (DW:CTL:ENABLED=1) when DMA is used. The ADC driver cannot access the global register (DW:CTL:ENABLED bit) directly because this setting affects other modules which access to the same register.*

Note: *If DMA and interrupt mode are disabled, `Adc_ReadGroup()` and `Adc_GetStreamLastPointer()` read the conversion result from the associated register directly. Therefore, if the call timing of `Adc_ReadGroup()` and `Adc_GetStreamLastPointer()` is after the next conversion result comes in, the conversion result, read by the APIs, is already overwritten by the next conversion. If DMA and interrupt mode are disabled, reading the conversion result from register directly might cause reading the result that it is out of range.*

Note: *For HW triggered group, the conversion result might be overwritten, when the trigger cycle is short.*

Note: *When DMA or interrupt is used, ADC driver transfers the conversion result from the register to the result buffer using DMA or interrupt. In this case, ADC driver might transfer the overwritten result if the next group conversion completes faster than the DMA transfer or interrupt processing. Therefore, it is recommended to use DMA that can operate at high speed.*

Note: *When DMA is used, the group cannot contain a channel in which* `AdcChannelPulseDetect` *is enabled.*

5.15 Changing the sampling time during runtime

The ADC driver supports change of sampling time for an ADC channel during runtime. Sampling time can be changed by calling an API `Adc_ChangeSamplingTime()`. The API has three parameters to specify the affected ADC channel group, affected ADC channel, and sampling duration. The third argument `SamplingTime` is represented in cycles but not time. Therefore, it should be specified in cycles according to current SAR clock frequency.

Note: *An error will occur if the API is called when the ADC channel group is not idle.*

5.16 Port selection

Each hardware unit is preceded by its own SARMUX, which connects to a distinct set of up to 32 analog pins. This means that hardware unit 1 cannot sample the analog pins connected to hardware unit 2.

In some cases, it may be desirable to have one hardware unit being able to reach all analog inputs of the chip.

The ADC driver provides the `AdcSarMux1ConnectToAdc0`, `AdcSarMux2ConnectToAdc0`, and `AdcSarMux3ConnectToAdc0` configuration parameters to support this use case.

When these configuration parameters are enabled, the corresponding SARMUXes are connected to the hardware unit of SAR ADC0.

Note: *SARMUXes of other hardware units can be connected only to the hardware unit of SAR ADC0. Hardware units of SARMUXes connected to the hardware unit of SAR ADC0 cannot be used.*

5.17 Sample mode

Sample mode is used to specify whether preconditioning, overlap diagnostic, or both is enabled. The `AdcSampleMode` configuration parameter is used to specify the sample mode that is used.

Preconditioning provides functionality to enable broken wire detection by charging or discharging the ADC sampling capacitor before sampling the input signal. If preconditioning is enabled, preconditioning time can be specified in cycle (SAR clock) by the `AdcPreconditionCycle` configuration parameter.

Overlap diagnostic allows diagnostic reference output and the analog input signal to connect the ADC sampling capacitor at the same time (see *TRAVEO™ T2G automotive body controller entry family architecture technical reference manual* for further information).

Table 3 lists the sample mode and whether it covers preconditioning, overlap diagnostic, or both. `ADC_SAMPLE_NORMAL` needs to be used if neither preconditioning nor overlap diagnostic are used.

Table 3 Available sample modes

AdcSampleMode	Preconditioning	Overlap diagnostic reference
ADC_SAMPLE_NORMAL	Not used	Not used
ADC_SAMPLE_NORMAL_HALF	Not used	Overlapping for the first half of the sample window
ADC_SAMPLE_NORMAL_FULL	Not used	Overlapping for the full sample window
ADC_SAMPLE_NORMAL_MUX	Not used	Measure diagnostic reference through SARMUX input
ADC_SAMPLE_VREFL	Discharge to VREFL	Not used
ADC_SAMPLE_VREFL_HALF	Discharge to VREFL	Overlapping for the first half of the sample window
ADC_SAMPLE_VREFL_FULL	Discharge to VREFL	Overlapping for the full sample window
ADC_SAMPLE_VREFL_MUX	Discharge to VREFL	Measure diagnostic reference through SARMUX input
ADC_SAMPLE_VREFH	Charge to VREFH	Not used
ADC_SAMPLE_VREFH_HALF	Charge to VREFH	Overlapping for the first half of the sample window
ADC_SAMPLE_VREFH_FULL	Charge to VREFH	Overlapping for the full sample window
ADC_SAMPLE_VREFH_MUX	Charge to VREFH	Measure diagnostic reference through SARMUX input
ADC_SAMPLE_DIAG	Connect to the diagnostic reference output during preconditioning	Not used
ADC_SAMPLE_DIAG_MUX	Connect to the diagnostic reference output during preconditioning	Measure diagnostic reference through SARMUX input

5.18 Diagnostic feature

The ADC driver provides hardware feature about diagnostic.

Diagnostic reference can be configured by the `AdcDiagnosticReference` configuration parameter. It is only available when the `AdcDiagnoseEnable` configuration parameter is enabled.

The `AdcSarMux1DiagnosticReference`, `AdcSarMux2DiagnosticReference`, `AdcSarMux3DiagnosticReference`, `AdcSarMux1DiagnoseEnable`, `AdcSarMux2DiagnoseEnable`, and `AdcSarMux3DiagnoseEnable` configuration parameters also operate in the same manner.

Table 4 lists the diagnostic reference available in each sample mode (that is, the `AdcSampleMode` configuration parameter).

Note: ADC driver does not conduct diagnostic itself. Therefore, application needs to perform diagnostics using the diagnostic feature.

Table 4 Available diagnostic references in each sample mode

AdcSampleMode	Diagnostic reference used?	Available diagnostic references (iAdcDiagnosticReference)
ADC_SAMPLE_NORMAL	Not used	–
ADC_SAMPLE_NORMAL_HALF	Used	ADC_DIAG_I_SOURCE or ADC_DIAG_I_SINK
ADC_SAMPLE_NORMAL_FULL	Used	ADC_DIAG_I_SOURCE or ADC_DIAG_I_SINK
ADC_SAMPLE_NORMAL_MUX	Used	ADC_DIAG_VREFL to ADC_DIAG_VIN3
ADC_SAMPLE_VREFL	Not used	–
ADC_SAMPLE_VREFL_HALF	Used	ADC_DIAG_I_SOURCE or ADC_DIAG_I_SINK
ADC_SAMPLE_VREFL_FULL	Used	ADC_DIAG_I_SOURCE or ADC_DIAG_I_SINK
ADC_SAMPLE_VREFL_MUX	Used	ADC_DIAG_VREFL to ADC_DIAG_VIN3
ADC_SAMPLE_VREFH	Not used	–
ADC_SAMPLE_VREFH_HALF	Used	ADC_DIAG_I_SOURCE or ADC_DIAG_I_SINK
ADC_SAMPLE_VREFH_FULL	Used	ADC_DIAG_I_SOURCE or ADC_DIAG_I_SINK
ADC_SAMPLE_VREFH_MUX	Used	ADC_DIAG_VREFL to ADC_DIAG_VIN3
ADC_SAMPLE_DIAG	Used	ADC_DIAG_VREFL to ADC_DIAG_VIN3
ADC_SAMPLE_DIAG_MUX	Used	ADC_DIAG_VREFL to ADC_DIAG_VIN3

5.19 Analog calibration feature

Analog calibration is used to make the actual ADC transfer curve move closer to the ideal transfer curve. Analog calibration can be conducted by correcting an offset and a gain error.

ADC driver provides the following APIs to correct an offset and a gain value (see

Functions for further information).

- `Std_ReturnType Adc_ChangeCalibrationChannel(Adc_GroupType Group, Adc_SignalType Signal);`
- `Std_ReturnType Adc_SetCalibrationValue(Adc_HwUnitType HwUnit, Adc_OffsetValueType Offset, Adc_GainValueType Gain, boolean Update);`
- `Std_ReturnType Adc_GetCalibrationAlternateValue(Adc_HwUnitType HwUnit, Adc_OffsetValueType * OffsetPtr, Adc_GainValueType * GainPtr);`
- `Std_ReturnType Adc_GetCalibrationValue(Adc_HwUnitType HwUnit, Adc_OffsetValueType * OffsetPtr, Adc_GainValueType * GainPtr);`

Hardware supports not only regular calibration but also alternate calibration to find new correction values during runtime. You should use alternate calibration instead of regular calibration when new correction values need to be found. The `AdcUseAlternateCalibration` configuration parameter in the AdcGroup which is used for finding new correction values needs to be set to TRUE, if alternate calibration is used.

The same APIs (`Adc_StartGroupConversion`, `Adc_StopGroupConversion`, `Adc_DisableHardwareTrigger`, `Adc_EnableHardwareTrigger`, `Adc_ReadGroup`, `Adc_GetStreamLastPointer`, `Adc_DisableGroupNotification`, `Adc_EnableGroupNotification`, `Adc_SetupResultBuffer`, `Adc_GetGroupStatus`, `Adc_SelectChannelThreshold`, `Adc_DisableChannel`, `Adc_EnableChannel`, `Adc_GetADCAddr`, `Adc_ReadChannelValue`, `Adc_GetGroupLimitCheckState`, `Adc_EnableHwTrigger`, `Adc_DisableHwTrigger`) are used for channel group conversion regardless of whether alternate calibration is used.

You need to find new correction values according to a calibration flow mentioned in Table 5 (see *TRAVEO™ T2G automotive body controller entry family architecture technical reference manual* for further information). An offset and a gain error depend on external factors (for example, temperature). Therefore, analog calibration needs to be conducted periodically.

Table 5 Calibration of the ADC hardware

No.	Calibration flow	API to be called at each stage
1	Set an analog gain correction value (ANA_CAL_ALT.AGAIN) to ‘0’.	This operation is conducted in calibration flow 3 and 5.
2	Configure a channel to convert VrefL.	Call `Adc_ChangeCalibrationChannel(` Group=’AlternateCalibrationGroup’, Signal=’VrefL’)
3	Do several software-triggered acquisitions using different AOFFSET values X (ANA_CAL_ALT.AOFFSET). Do this until the AOFFSET value X is found for which the converted value transitions from 0x001 to 0x000.	Call `Adc_SetCalibrationValue(` HwUnit=’TargetHwUnit’, Offset=X, Gain=’0’, Update=’FALSE’) Call Adc_StartGroupConversion(Group=’AlternateCalibrationGroup’) Call Adc_ReadGroup(Group=’AlternateCalibrationGroup’, DataBufferPtr=<DataBufferAddress>) This operation is repeated until the appropriate X is found.
4	Now change the channel configuration to convert VrefH.	Call `Adc_ChangeCalibrationChannel(` Group=’AlternateCalibrationGroup’, Signal=’VrefH’)

No.	Calibration flow	API to be called at each stage
5	Do several software triggered acquisitions using different AOFFSET values Y (ANA_CAL_ALT.AOFFSET). Do this until the AOFFSET value Y is found for which the converted value transitions from 0xFFE to 0xFFF.	Call `Adc_SetCalibrationValue(` HwUnit='TargetHwUnit', Offset=Y, Gain='0', Update='FALSE') Call Adc_StartGroupConversion(Group='AlternateCalibrationGroup') Call Adc_ReadGroup(Group='AlternateCalibrationGroup', DataBufferPtr=<DataBufferAddress>) This operation is repeated until the appropriate Y is found.
6	Calculate A = (X+Y) / 2 + 2 (by application code itself).	–
7	Set the analog offset value (ANA_CAL_ALT.AOFFSET) to A.	This operation is conducted in no.9 and no.11.
8	Change the channel configuration back to converting VrefL.	Call `Adc_ChangeCalibrationChannel(` Group='AlternateCalibrationGroup', Signal='VrefL')

No.	Calibration flow	API to be called at each stage
9	Do several software triggered acquisitions using different AGAIN values Z (ANA_CAL_ALT.AGAIN). Do this until the AGAIN value Z is found for which the converted value transitions from 0x001 to 0x000 (using averaging for the final acquisitions).	Call `Adc_SetCalibrationValue(` HwUnit='TargetHwUnit', Offset=A, Gain=Z, Update='FALSE') Call Adc_StartGroupConversion(Group='AlternateCalibrationGroup') Call Adc_ReadGroup(Group='AlternateCalibrationGroup', DataBufferPtr=<DataBufferAddress>) This operation is repeated until the appropriate Z is found.
10	Calculate B = Z + 1 (by application code itself).	–
11	Set the analog gain value (ANA_CAL_ALT_AGAIN) to B and update the regular calibration register with alternate calibration values.	Call `Adc_SetCalibrationValue(` HwUnit='TargetHwUnit', Offset=A, Gain=B, Update='TRUE')

After the correction values have been found by alternate calibration, the regular calibration register needs to be updated with these values. After hardware accepts the request to be updated, it will do the calibration update when the ADC is idle. In case of a continuous triggered group (that is, the channel group is configured as a continuous conversion group and configured to use single-access-mode or enable limit-check-feature), the channel group conversion is repeated. In this case, the HW will do the calibration update after the group completes and before the group starts automatically again.

To check whether update is already finished, compare the regular calibration values read by `Adc_GetCalibrationValue()` with the new calibration values set by `Adc_SetCalibrationValue()` and confirm that the values are same.

Note: *When an alternate calibration value is changed by `Adc_SetCalibrationValue`, the channel group which uses the alternate calibration should be stopped in advance. If this procedure is not obeyed, it will result in undefined results for that acquisition.*

Note: *Selecting the mean of correction values helps you to find more accurate correction values (that is, an AOFFSET value X, an AOFFSET value Y, and an AGAIN value Z in Table 5).*

Here's an example to find more accurate correction values:

ADC channel group conversion needs to be repeated as described in Table 5 (3, 5, and 9) until each correction value is found. For example, in case of point 3, you need to find an AOFFSET value X for which the converted value transitions from 0x000 to 0x001. This conversion needs to be repeated while incrementing value X until the value is found. Finally, you can find the value and use the value as a correction value, however you can find the value by the same way again. If you repeat the operation to find an AOFFSET value X number of times, the value of X might be different for each operation. The mean of correction values would deem an accurate correction value. You can also find more accurate values of an AOFFSET value Y and an AGAIN value Z by using the same method.

5.20 SelfDiag feature

Connectivity or correctness can be verified by using the SelfDiag feature. Connectivity of internal line related to an ADC channel will be verified through SelfDiag APIs by checking the converted result to feed the internal Diag voltage VrefL and VrefH, if the channel is configured as `ADC_DIAGNOSIS_SIMPLE`.

The correctness of the AD converter related to an ADC channel will be checked through the converted result to feed the internal Diag VrefL, VrefH, VrefH/2, VrefH/3, VrefH/4, VrefH/5, VrefH/6, and VrefH/7, if the channel is configured as `ADC_DIAGNOSIS_FULL`.

The acceptable range for correctness can be set at the AdcVoltageDeviation configuration parameter on the **General** tab.

Note: *The SelfDiag feature can be used after initialization (calling Adc_Init()).*

Note: *When the SelfDiag APIs return FALSE via the OutputPtr output argument, it is recommended to do calibration.*

5.21 Hardware prioritization

The hardware supports two types of prioritization:

1. Explicit hardware prioritization
2. Implicit hardware prioritization

5.21.1 Explicit hardware prioritization

The ADC hardware allows to configure a priority for each ADC channel group in the range of 0 to 7, where 7 has the highest priority (converted first) and 0 has the lowest priority (converted last).

The ADC driver will always generate the same explicit hardware priority to all logical channels of the same group.

The explicit hardware priority is stored as an element `GroupHwPriority` in the type `Adc_GroupConfigType`.

Note: *When a group scan is ongoing and a new higher priority trigger arrives, then it can cause the preemption of the ongoing lower priority group scan. The trigger preemption type can be specified by the `AdcGroupReplacement` configuration parameter (see AdcChannel Configuration).*

Note: *The `AdcGroupPriority` configuration parameter and hardware have different priorities. For hardware, 0 is the highest priority, whereas 7 is the highest priority for `AdcGroupPriority`. `AdcGroupPriority` is changed to hardware priority inside the driver by inverting it (for instance, 0 to 7, 1 to 6, and so on).*

5.21.2 Implicit hardware prioritization

The ADC hardware prioritizes all channels that have the same explicit hardware priority according to their index of the logical channels, where lower index values take precedence.

Note: *The driver does not allow to configure same priorities, because the implicit hardware prioritization is always applied.*
The `AdcFirstLogicalChannel` configuration parameter allows to influence the implicit hardware priority.

Note: *The `AdcFirstLogicalChannel` configuration parameter also affects the available hardware triggers for the group.*

Note: *Implicit priority only determines which of the pending group scans will be executed first. It cannot cause the preemption of the ongoing lower priority group scan.*

5.22 Software prioritization

ADC provides software priority handling between `AdcUseDynamicAllocate` enabled groups. A maximum of 16 groups can have `AdcUseDynamicAllocate` enabled. For the group, `AdcGroupTriggerSrc` must be `ADC_TRIGG_SRC_SW`.

Software priority mechanism internally has a SW queue, and the group called with `Adc_StartGroupConversion()` is stored in the SW queue once. The groups with the highest priority in SW queue will be registered in the hardware register. The groups registered in the hardware register will be converted sequentially according to other group and hardware prioritization.

SW queue follows these rules:

- SW queue is processed in `AdcSWPriority` order
- `AdcSWPriority` ranges from 0 to 255. (255 is the highest priority)
- If `AdcSWPriority` is the same; groups registered by `Adc_StartGroupConversion()` will be processed first.
- If there is a group that has been converted first and if a group with higher `AdcSWPriority` is started by `Adc_StartGroupConversion()`, the group with higher `AdcSWPriority` will interrupt. Interrupted groups are pushed to the SW queue. The interrupt is performed within the interrupt processing that has been converted to the end of the group.

5.23 API parameter checking

The ADC driver's services perform regular error checks.

When an error occurs, the error hook routine (configured via the `AdcErrorCalloutFunction` parameter) is called and the error code, service ID, module ID, and instance ID are passed as parameters.

If a development error detection is enabled, all errors are also reported to DET, which is a central error hook function within the AUTOSAR environment. The checking itself cannot be deactivated for safety reasons.

The following development error checks are performed by the services of the ADC driver:

- The `Adc_Init()` function checks if the `ConfigPtr` parameter is within the scope of the post-build configuration. In the case of invalid configuration pointer, the error code `ADC_E_PARAM_CONFIG` will be reported.
- The `Adc_DeInit()`, `Adc_StartGroupConversion()`, `Adc_StopGroupConversion()`, `Adc_ReadGroup()`, `Adc_EnableHardwareTrigger()`, `Adc_DisableHardwareTrigger()`, `Adc_EnableGroupNotification()`, `Adc_DisableGroupNotification()`, `Adc_SetupResultBuffer()`, `Adc_GetStreamLastPointer()`, `Adc_GetGroupStatus()`, `Adc_GetCurrentPowerState()`, `Adc_GetTargetPowerState()`, `Adc_PreparePowerState()`, and `Adc_SetPowerState()` functions check if the driver has already been initialized. In case of an uninitialized driver, the error code `ADC_E_UNINIT` will be reported.
- The `Adc_Init()` function checks if the driver has already been initialized. In case of an already initialized driver, the error code `ADC_E_ALREADY_INITIALIZED` will be reported.
- The `Adc_DeInit()` function checks whether there is a running conversion. The `Adc_StartGroupConversion()`, `Adc_EnableHardwareTrigger()`, and `Adc_SetupResultBuffer()` check if the given group has been started for conversion and has not yet finished. If any of the above cases applies, the error code `ADC_E_BUSY` will be reported.
- The `Adc_StartGroupConversion()`, `Adc_StopGroupConversion()`, `Adc_ReadGroup()`, `Adc_EnableHardwareTrigger()`, `Adc_DisableHardwareTrigger()`, `Adc_EnableGroupNotification()`, `Adc_DisableGroupNotification()`, `Adc_GetStreamLastPointer()`, `Adc_SetupResultBuffer()`, and `Adc_GetGroupStatus()` functions check whether a valid group parameter is passed on as an input parameter. If an invalid group occurs, the error code `ADC_E_PARAM_GROUP` will be reported.
- The `Adc_StartGroupConversion()`, `Adc_StopGroupConversion()`, `Adc_EnableHardwareTrigger()`, and `Adc_DisableHardwareTrigger()` functions check whether a group with a valid trigger source is specified. If the group has an invalid trigger source, the error code `ADC_E_WRONG_TRIGG_SRC` will be reported.
- The `Adc_StopGroupConversion()`, `Adc_DisableHardwareTrigger()`, `Adc_ReadGroup()`, and `Adc_GetStreamLastPointer()` functions check whether a conversion has been started. If the group's status is `ADC_IDLE`, the error code `ADC_E_IDLE` will be reported.
- The `Adc_EnableHardwareTrigger()` and `Adc_DisableHardwareTrigger()` functions check whether the group is configured for one-shot conversion. If the group is configured for continuous conversion, the error code `ADC_E_WRONG_CONV_MODE` will be reported.
- The `Adc_EnableGroupNotification()` and `Adc_DisableGroupNotification()` functions check if a notification function is configured for the group. If not, the error code `ADC_E_NOTIF_CAPABILITY` will be reported.
- The `Adc_StartGroupConversion()` and `Adc_EnableHardwareTrigger()` functions check whether the result buffer for the group is valid (has been set up). If the group's result buffer is NULL, the error code `ADC_E_BUFFER_UNINIT` will be reported.
- The `Adc_GetVersionInfo()` and `Adc_SetupResultBuffer()` functions check if the function is called with NULL pointer. In case of NULL pointer, the error code `ADC_E_PARAM_POINTER` will be reported.
- The `Adc_PreparePowerState()` and `Adc_SetPowerState()` functions check if unsupported power state is required. In this case, the error code `ADC_E_POWER_STATE_NOT_SUPPORTED` will be reported.
- The `Adc_SetPowerState()` function checks if one or more ADC group/channel is not in IDLE state. If one or more ADC groups/channels are running, the error code `ADC_E_NOT_DISENGAGED` will be reported.

Also, see

Functions for a description of API functions and the associated error codes.

5.24 Vendor-specific error checking

The ADC driver also performs vendor-specific error checks.

When an error occurs, the same error hook routine (configured via the `AdcErrorCalloutFunction` parameter) is called as for other error checks.

The following vendor specific development error checks are performed by the services of the ADC driver:

- The `Adc_ChangeSamplingTime()`, `Adc_ChangeCalibrationChannel()`, `Adc_SetCalibrationValue()`, `Adc_GetCalibrationAlternateValue()`, `Adc_GetCalibrationValue()`, `Adc_GetDiagnosticResult()`, `Adc_SelectChannelThreshold()`, `Adc_StartDiagnostic()`, and `Adc_StartDiagnosticFull()`, `Adc_DisableChannel()`, `Adc_EnableChannel()`, `Adc_GetADCAddr()`, `Adc_ReadChannelValue()`, `Adc_GetGroupLimitCheckState()`, `Adc_EnableHwTrigger()` and `Adc_DisableHwTrigger()` functions check if the driver has already been initialized. In the case of an uninitialized driver, the `ADC_E_UNINIT` error code will be reported.
- The `Adc_ChangeSamplingTime()`, `Adc_ChangeCalibrationChannel()`, `Adc_SelectChannelThreshold()`, `Adc_DisableChannel()`, `Adc_EnableChannel()`, and `Adc_EnableHwTrigger()` functions check whether the given group is not running. If it's ongoing, the `ADC_E_BUSY` error code will be reported.
- The `Adc_StartDiagnosticFull()` function checks whether the all groups in the given HW unit are not running. If any group is ongoing, the `ADC_E_BUSY` error code will be reported.
- The `Adc_GetDiagnosticResult()` and `Adc_StartDiagnostic()` functions check whether the all groups in the HW unit, to which the given channel belongs, are not running. If any group is ongoing, the `ADC_E_BUSY` error code will be reported.
- The `Adc_ChangeSamplingTime()`, `Adc_ChangeCalibrationChannel()`, `Adc_SelectChannelThreshold()`, `Adc_DisableChannel()`, `Adc_EnableChannel()`, `Adc_GetADCAddr()`, `Adc_ReadChannelValue()`, `Adc_GetGroupLimitCheckState()`, `Adc_DisableHwTrigger()`, and `Adc_EnableHwTrigger()` functions check whether a valid group parameter is passed as an input parameter. If an invalid group is detected, the `ADC_E_PARAM_GROUP` error code will be reported.
- The `Adc_EnableHwTrigger()` and `Adc_DisableHwTrigger()` functions check whether a group with a valid trigger source is specified. If the group has an invalid trigger source, the `ADC_E_WRONG_TRIGG_SRC` error code will be reported.
- The `Adc_GetStreamLastPointer()`, `Adc_ReadGroup()`, `Adc_SetPowerState()`, `Adc_GetCurrentPowerState()`, `Adc_GetTargetPowerState()`, `Adc_PreparePowerState()`, `Adc_GetCalibrationAlternateValue()`, `Adc_GetCalibrationValue()`, `Adc_GetDiagnosticResult()`, `Adc_GetGroupLimitCheckState()`, `Adc_ReadChannelValue()`, `ADC_API_START_DIAGNOSTIC()`, and `ADC_API_START_DIAGNOSTIC_FULL()` functions check if the pointers passed as parameters are valid (not a NULL pointer). In the case of an invalid pointer, the `ADC_E_PARAM_POINTER` error code will be reported.
- The `Adc_ChangeSamplingTime()`, `Adc_ReadChannelValue()`, and `Adc_SelectChannelThreshold()` functions check if the ADC channel passed as parameter is included in the group. If the group does not include the ADC channel, the `ADC_E_PARAM_CHANNEL` error code will be reported.
- The `Adc_GetDiagnosticResult()` and `Adc_StartDiagnostic()` functions check whether the ADC channel passed as a parameter is currently on-going for diagnosis. If it is on-going, the `ADC_E_PARAM_CHANNEL` error code will be reported.

- The Adc_GetDiagnosticResult() and Adc_StartDiagnostic() functions check if the ADC channel passed as a parameter is included in the HW unit. If the ADC channel is not included in the HW unit, the ADC_E_PARAM_CHANNEL error code will be reported.
- The Adc_ChangeSamplingTime() function checks if the sampling time passed as parameter is valid. If it is out of range, the ADC_E_PARAM_SAMPLING_TIME error code will be reported.
- The Adc_GetGroupLimitCheckState(),Adc_GetStreamLastPointer(),Adc_ReadChannelValue() and Adc_ReadGroup() functions check if DMA error is detected during the ADC conversion. If the error is detected, the ADC_E_CONVERSION_ERROR error code will be reported.
- The Adc_ChangeCalibrationChannel() function checks whether the given signal is valid. If invalid, the ADC_E_PARAM_SIGNAL error code will be reported.
- The Adc_ChangeCalibrationChannel() function checks whether the alternate calibration is enabled in the given group. If it is disabled, the ADC_E_PARAM_GROUP error code will be reported.
- The Adc_SetCalibrationValue() function checks whether the given gain is within the range. If out of range, the ADC_E_PARAM_GAIN error code will be reported.
- The Adc_SetCalibrationValue() function checks whether the given update value is valid. If invalid, the ADC_E_PARAM_UPDATE error code will be reported.
- The Adc_GetCalibrationAlternateValue(),Adc_GetCalibrationValue(), Adc_SetCalibrationValue(), and Adc_StartDiagnosticFull() functions check whether the given HW unit is valid. If invalid, the ADC_E_PARAM_HWUNIT error code will be reported.
- The Adc_ReadChannelValue(),Adc_GetGroupLimitCheckState(), and Adc_DisableHwTrigger() functions check whether a conversion has been started. If the group's status is ADC_IDLE, the ADC_E_IDLE error code will be reported.
- The Adc_EnableHwTrigger() and Adc_DisableHwTrigger() functions check whether the group is configured for one-shot conversion. If the group is configured for continuous conversion, the ADC_E_WRONG_CONV_MODE error code will be reported.
- The Adc_EnableHwTrigger() function checks whether the result buffer for the group is valid (has been set up). If the group's result buffer is NULL, the ADC_E_BUFFER_UNINIT error code will be reported.
- The Adc_GetStreamLastPointer(),Adc_ReadGroup(), and Adc_ReadChannelValue() functions check if the result buffer and redundancy buffer match. If there is a mismatch, the ADC_E_REDUNDANCY_ERROR error code will be reported.
- The Adc_DisableChannel() and Adc_EnableChannel() functions check if the channel is already enabled or disabled. In this case, the ADC_E_CHANNEL_ID_NG error code will be reported.
- The Adc_EnableHwTrigger() function checks whether the argument is outside the range of generic trigger. In this case, the ADC_E_PARAM_SELECT_TRIGG error code will be reported.
- The Adc_SelectChannelThreshold() function checks if the threshold value passed as parameter is valid. If invalid, the ADC_E_PARAM_THRESHOLD_VALE error code will be reported.
- The Adc_Init() function checks whether the specified configuration is same between master core and satellite core. If the configuration is dirrerent, the ADC_E_DIFFERENT_CONFIG error code will be reported.
- The Adc_ChangeCalibrationChannel(),Adc_ChangeSamplingTime(),Adc_DeInit(), Adc_DisableChannel(),Adc_DisableGroupNotification(),Adc_DisableHardwareTrigger(), Adc_DisableHwTrigger(),Adc_EnableChannel(),Adc_EnableGroupNotification(), Adc_EnableHardwareTrigger(),Adc_EnableHwTrigger(),Adc_GetADCAddr(), Adc_GetCalibrationAlternateValue(),Adc_GetCalibrationValue(), Adc_GetCurrentPowerState(),Adc_GetDiagnosticResult(),Adc_GetGroupStatus(), Adc_GetGroupLimitCheckState(),Adc_GetStreamLastPointer(),Adc_GetTargetPowerState(), Adc_Init(),Adc_PreparePowerState(),Adc_ReadChannelValue(),Adc_ReadGroup(),

`Adc_SelectChannelThreshold()`, `Adc_SetCalibrationValue()`, `Adc_SetPowerState()`, `Adc_SetupResultBuffer()`, `Adc_StartDiagnostic()`, `Adc_StartDiagnosticFull()`, `Adc_StartGroupConversion()`, and `Adc_StopGroupConversion()` functions check whether the API is called in an expected core. If the API is called in unexpected core, the `ADC_E_INVALID_CORE` error code will be reported.

- The `Adc_DisableHardwareTrigger()`, `Adc_DisableHwTrigger()`, `Adc_GetLimitCheckState()`, `Adc_GetStreamLastPointer()`, `Adc_ReadChannelValue()`, `Adc_ReadGroup()`, and `Adc_StopGroupConversion()` functions check whether the given group status is the same as ADC_DIAG. In this case, the `ADC_E_DIAG` error code will be reported.

Also, see

Functions for a description of API functions and the associated error codes.

5.25 Reentrancy

`Adc_Init()`, `Adc_DeInit()`, `Adc_ChangeSamplingTime()`, `Adc_SetPowerState()`, `Adc_GetCurrentPowerState()`, `Adc_GetTargetPowerState()`, `Adc_PreparePowerState()`, `Adc_SelectChannelThreshold()`, `Adc_DisableChannel()`, `Adc_EnableChannel()`, `Adc_GetADCAddr()`, `Adc_ReadChannelValue()`, and `Adc_GetGroupLimitCheckState()` are not reentrant. `Adc_GetCalibrationAlternateValue()`, `Adc_GetCalibrationValue()`, `Adc_GetVersionInfo()`, `Adc_StartDiagnosticFull()`, `Adc_GetDiagnosticResult()` and `Adc_StartDiagnostic()` are reentrant.

`Adc_SetCalibrationValue()` is reentrant if called on different HW units. It is non-reentrant if called on the same HW unit. The service may be accessed by different tasks or interrupts simultaneously if the tasks or interrupts access disjunct sets of HW units.

All other services are reentrant if called on different groups. They are non-reentrant if called on the same group. The services may be accessed by different tasks or interrupts simultaneously as long as the tasks or interrupts access disjunct sets of groups.

5.26 Configuration checking

ADC groups are defined as a group of channels. All channels of the group must be configured.

Additional hardware dependent checks are implemented.

5.27 Sleep mode

The ADC driver and the hardware controlled by the ADC driver do not provide a dedicated Sleep mode.

Note: *After entering DeepSleep mode, the ADC hardware does not save the value of the result register. If a conversion has already been completed but the result has not yet been read, the user application must get the result before entering the DeepSleep mode.*

5.28 Debugging support

The ADC driver does not support debugging.

5.29 Execution-time dependencies

Table 6 lists the dependencies of API functions and ISRs.

Table 6 Execution-time dependencies

Affected function	Dependency
`Adc_Init()` `Adc_DeInit()`	Runtime depends on the number of hardware units, configured groups, and logical channels configured for each group. The runtime also depends on the usage of DMA feature.
`Adc_DmaDone_*()`	Runtime depends on whether the group is completed. If the group is finished, it needs more time because all logical channels need to be disabled. The duration depends on the number of logical channels configured for the group. The runtime also depends on the mode that is used (HW trigger or SW trigger, continuous conversion or one-shot conversion, single access or streaming access). Furthermore, if software priority is enabled, priority processing is performed, so the runtime depends on the groups in which software priority is enabled.
`Adc_IsrConversionDone_*()`	Runtime depends on the number of logical channels configured for the group, group access mode, and current group status. The runtime also depends on the mode that is used (HW trigger or SW trigger, continuous conversion or one-shot conversion, single access or streaming access). Furthermore, if software priority is enabled, priority processing is performed, so the runtime depends on the groups in which software priority is enabled.

`Adc_EnableHardwareTrigger()`
`Adc_DisableHardwareTrigger()`
 	
`Adc_EnableHwTrigger()`
`Adc_DisableHwTrigger()`
	Runtime depends on the number of logical channels configured for the group that is given as input parameter.
	The runtime also depends on the usage of DMA feature and the mode that is used (HW trigger or SW trigger, continuous conversion or one-shot conversion, single access or streaming access, and DMA).

`Adc_GetGroupStatus()`	Runtime depends on whether interrupt or polling mode is used. Interrupt mode is faster than polling mode. The access to a status register is necessary in case of polling mode, because no interrupt updates the current status. Whereas, the register access is not necessary in case of interrupt mode, because the status is always updated in the interrupts. The runtime also depends on the mode that is used (HW trigger or SW trigger, continuous conversion or one-shot conversion, single access or streaming access, and DMA).

Affected function	Dependency

`Adc_DisableGroupNotification()`
`Adc_EnableGroupNotification()`
`Adc_GetVersionInfo()`
`Adc_SetupResultBuffer()`
`Adc_Main_PowerTransitionManager()`
`Adc_GetCurrentPowerState()`
`Adc_GetTargetPowerState()`
`Adc_PreparePowerState()`
`Adc_GetADCAddr()`
`Adc_GetGroupLimitCheckState()`
	Runtime does not depend on any configuration setting.

`Adc_SetPowerState()`	Runtime depends on the number of configured hardware units and groups.

`Adc_ChangeSamplingTime()`
`Adc_ChangeCalibrationChannel()`
`Adc_SelectChannelThreshold()`
`Adc_DisableChannel()`
`Adc_EnableChannel()`
`Adc_GetDiagnosticResult()`
	Runtime depends on the number of logical channels included in the specified channel group.

`Adc_SetCalibrationValue()`
`Adc_GetCalibrationAlternateValue()`
`Adc_GetCalibrationValue()`
	Runtime depends on the number of HW units defined in the configuration set and the position of the HW unit in the configuration set.

`Adc_StartGroupConversion()` `Adc_StopGroupConversion()`	Runtime depends on the number of logical channels configured for the group that is given as input parameter. The runtime also depends on the usage of DMA feature and the mode that is used (HW trigger or SW trigger, continuous conversion or one-shot conversion, single access or streaming access, and DMA). Furthermore, if software priority is enabled, priority processing is performed, so the runtime depends on the groups in which software priority is enabled.

`Adc_ReadGroup()`
`Adc_GetStreamLastPointer()`
`Adc_ReadChannelValue()`
	Runtime depends on the number of logical channels configured for the group that is given as input parameter.
	The runtime also depends on the usage of DMA feature and the mode that is used (HW trigger or SW trigger, continuous conversion or one-shot conversion, single access or streaming access, and DMA).
	Furthermore, if `AdcUseRedundancy` is enabled, the runtime depends on it to compare the result buffer with the redundancy buffer.

`Adc_StartDiagnosticFull()` `Adc_StartDiagnostic()`	Runtime depends on the number of logical channels included in the specified HW unit or group.

Note: *`Adc_GetVersionInfo()`, `Adc_Init()`, `Adc_GetCurrentPowerState()`, `Adc_GetTargetPowerState()`, and `Adc_Main_PowerTransitionManager()` do not use a critical section inside. Other than these APIs (including ISRs) use a critical section inside. The duration of critical section is proportional to the execution time which is listed in Table 6. But it is*

not proportional in case of `Adc_GetCalibrationAlternateValue()` *and* `Adc_GetCalibrationValue()`.

5.30 Important notes on the ADC driver's environment

Table 7 lists some important notes for the usage of the ADC driver within the environment specified by AUTOSAR.

Table 7 Important notes on the ADC driver's environment

No.	Short title	Description
1	Maintaining consistency	The ADC driver's environment guarantees the consistency of data that has been read by checking the return value of `Adc_GetGroupStatus()`.
2	Starting a conversion	To guarantee consistent values, it is assumed that you have started an ADC group conversion (or enabled in case of HW triggered group) successfully before status polling via `Adc_GetGroupStatus()` begins.
3	Preparing result buffer before starting a conversion	The ADC driver's environment ensures that the application buffer, whose address is passed as argument of `Adc_SetupResultBuffer()`, has sufficient size to hold all group channel conversion results, and if streaming access is selected, can hold these results multiple times as specified by the `AdcStreamingNumSamples` configuration parameter. In addition, if `AdcUseRedundancy` is enabled, the redundancy buffer is placed after the result buffer. It also guarantees that result buffer is aligned at memory address which is multiples of 2 bytes regardless of whether DMA is enabled. Regarding a sub-derivative which supports cache feature, the CPU has an individual cache that is not shared with the DMA bus master. Therefore, ensure that all result buffers used by the ADC channel groups in which DMA is enabled reside in a non-cacheable memory area. This can be achieved by placing the buffer in a user-specific memory region configured by the CPU's memory protection unit (MPU) as non-cache-able. The ADC driver does not support use of DMA for the result buffer placed in CPUs tightly coupled memories (TCMs). If used, the ADC driver reports to DEM error by DMA transfer.
4	Concurrent conversions on a same HW unit	The ADC driver's environment guarantees that no concurrent conversions take place on the same HW unit (that is, ADC hardware cannot handle more than two ADC channel groups in same HW unit concurrently). 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. If `AdcUseDynamicAllocate` is enabled, before a conversion request is issued to the hardware, the ADC module determines witch SW priority to convert a valid group to hardware based on software priority. This means priority control within SW queue using SW priority. The ADC module issues conversion requests to the hardware with the highest priority group within SW queue.

No.	Short title	Description
5	Capability of handling ADC group conversion	When single access mode and continuous conversion are used, next group conversion might be completed before the interruption of the current group conversion is finished. When HW trigger is used and the HW trigger frequently comes in for too short a period, the next group conversion might be completed before the interruption of the current group conversion is finished. It might cause unexpected behavior (for example, exception or hardware fault). The ADC driver's environment needs to take care of it.
6	Swap conversion order by SW priority	If there is a group of low priority of SW priority with `StreamingMode`(`AdcStreamingNumSamples` is more than 1) and a group of high priority of SW priority with `SingleMode`, even if the low priority group is undergoing conversion, the group with the highest priority that was `StartGroupConversoin` later will be finished first. This happens by exchanging SW queue every time stream ends.

5.31 Functions available without core dependency

Some APIs can be called on any core regardless of resource assignment.

The following function is available on any core without any restriction:

- `Adc_GetVersionInfo()`

The following function is available on any cores with a specific section allocation described in the Note:

- `Adc_GetADCAddr()`

  This function can get the address of the conversion result register in the first ADC channel of the requested ADC channel group.

Note: *The section `VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]` must be allocated to the memory which can be read from any core to call this API on any cores.*
For the details of INIT_POLICY and ALIGNMENT, see the specification of memory mapping [7].

6 Hardware resources

6.1 Peripheral clocks

The following peripheral clocks can be used to drive SAR ADCs.

SAR[n] clock (“n” represents SAR number and it depends on hardware which be used.)

6.2 Analog input signals

Table 8 shows analog input signals, which can be inputted into an ADC physical channel. GPIO and internal special signals are available. GPIO signals connect to certain pins, whereas internal special signals do not connect.

Make sure that pins to which GPIO signals connect are correctly set in the PORT driver’s configuration.

Table 8 Analog input signals

Signal type	Signals	Address of the analog signal	Needed to be initialized by PORT driver?	Description
GPIO signals	AN0…AN31	0…31	Yes	ADC analog input (32 inputs for signals from I/O pins)
Special signals	Vmotor	32	Yes	Motor input (This is an I/O pin.)
	Vaut	33	No	Auxiliary input
	AmuxbusA	34	Yes	“long-reach” signals to other input pads through GPIO AMUXBUS-A
	AmuxbusB	35	Yes	“long-reach” signals to other input pads through GPIO AMUXBUS-B
	Vccd	36	No	Digital power supply (Vccd)
	Vdda	37	No	Analog power supply (Vdda)
	Vdb	38	No	Bandgap voltage from SRSS (band-gap reference)
	Vtemp	39	No	Temperature sensor.
	VrefL	62	No	VREFL
	VrefH	63	No	VREFH

Note: One temperature sensor is shared by all ADCs. The temperature sensor must only be connected to one ADC at a time.

Note: For the temperature sensor, see the device datasheet for temperature sensor sampling time.

Note: For the temperature sensor, the following procedure is applicable for the TRAVEO™ T2G-B-E i.e., Body Entry devices in order to gain the accuracy of the temperature sensor. After a reset or DeepSleep wakeup, set bit 9, 8, and 6 of PASS_TEST_CTL register to 1 while keeping the other bits unchanged.

The ADC driver does not use any hardware timers, but can be triggered via hardware timers. TCPWM timers can be used to start HW triggered ADC channel group conversion. These timers are controlled by GPT, PWM, and OCU modules.

6.3 Interrupts

The ADC driver uses the interrupts associated with the configured hardware resource. The ISR should be allocated to the same core as allocated to HwUnit. The ISR must be declared in the AUTOSAR OS as Category 1 Interrupt or Category 2 Interrupt.

The interrupt that occurs after ADC conversion will appear after the last ADC channel in an ADC channel group is completed.

The interrupt that occurs after the completion of DMA transfer is only used when DMA feature is enabled. The interrupt will appear after DMA transfer, which copies the result to result buffer, is completed. The interrupt will also appear when DMA transfer has failed.

Note: *Vector numbers depend on the subderivative.*

You can define the ISR; The ISR and IRQ-Name for each ADC is specified as:

```
ISR_NATIVE(Adc_IsrConversionDone_<Interrupt vector>_Cat1)
ISR(Adc_IsrConversionDone_<Interrupt vector>_Cat2)
ISR_NATIVE(Adc_DmaDone_<Interrupt vector>_Cat1)
ISR(Adc_DmaDone_<Interrupt vector>_Cat2)
```

Note: *`<Interrupt vector>` represents numeric of interrupt vector, which depends on the logical channel of the group. If the last channel in the group is disabled by `Adc_DisableChannel()`, the last enable logical channel interrupt will occur.*
The interrupt that occurs after ADC conversion is related to `Adc_IsrConversionDone_<Interrupt vector>_Cat1/Cat2`, whereas the interrupt that occurs after DMA transfer is related to `Adc_DmaDone_<Interrupt vector>_Cat1/Cat2`.

Note: *Adc_IsrConversionDone_<Interrupt vector>_Cat2 and Adc_DmaDone_<Interrupt vector>_Cat2 must be called from the (OS) interrupt service routine.*
In the case of Category-1 usage, the address of `Adc_IsrConversionDone_<Interrupt vector>_Cat1` and `Adc_DmaDone_<Interrupt vector>_Cat1` must be the entry in the (OS) interrupt vector table.

Example: Category-1 ISR for ADC channel 3 is in the generated file *generate/src/Adc_Irq.c*:

```
ISR_NATIVE(Adc_IsrConversionDone_86_Cat1)
{
...
}
```

Example: Category-2 ISR for the ADC channel 3 is in the generated file *generate/src/Adc_Irq.c*:

```
ISR(Adc_IsrConversionDone_86_Cat2)
{
...
}
```

Example: Category-1 ISR for DMA channel 53 is in the generated file *generate/src/Adc_Irq.c*:

```
ISR_NATIVE(Adc_DmaDone_204_Cat1)
{
...
}
```

Example: Category-2 ISR for DMA channel 53 is in the generated file *generate/src/Adc_Irq.c*:

```
ISR_NATIVE(Adc_DmaDone_204_Cat2)
{
...
}
```

Note: *On the Arm® Cortex®-M4 CPU, priority inversion of interrupts may occur under specific timing conditions in the integrated system with TRAVEO™ T2G MCAL. For more details, see the following errata notice.*

Arm® Cortex®-M4 Software Developers Errata Notice - 838869:
"Store immediate overlapping exception return operation might vector to incorrect interrupt"

If the user application cannot tolerate the priority inversion, a DSB instruction should be added at the end of the interrupt function to avoid the priority inversion.

TRAVEO™ T2G MCAL interrupts are handled by an ISR wrapper (handler) in the integrated system. Thus, if necessary, the DSB instruction should be added just before the end of the handler by the integrator.

6.4 Triggers

There are two types of trigger, which might be used in ADC conversion. The type of trigger needs to be configured if HW trigger is used for a certain ADC channel group, DMA support is enabled, or both. These trigger settings are done by PORT driver.

Table 9 lists all triggers and their connections.

Table 9 All triggers related to ADC conversion

Connection	Trigger group
One-to-one trigger input from a corresponding TCPWM	One-to-one trigger group
Generic input trigger from timer, pins or system triggers	Multiplexer-based trigger group
One-to-one trigger output to a corresponding DW	One-to-one trigger group
Generic output trigger to DW	Multiplexer-based trigger group

Trigger group number depends on the subderivative. Generic trigger inputs are shared between ADCs.

7 Appendix A – API reference

7.1 Include files

The only file that needs to be included to use functions of the ADC driver is the *Adc.h* file.

7.2 Data types

7.2.1 Adc_ChannelType

Type

`uint16`

Description

Numeric ID of an ADC channel.

7.2.2 Adc_GroupType

Type

`uint16`

Description

Numeric ID of an ADC channel group.

7.2.3 Adc_ValueGroupType

Type

`uint16`

Description

This type represents the converted values for a channel group.

7.2.4 Adc_StreamNumSampleType

Type

`uint16`

Description

This type represents the number of group conversions in streaming access mode (in single access mode, value of relevant parameter, `AdcStreamingNumSamples` is 1).

7.2.5 Adc_StatusType

Type

```
typedef enum
{
    ADC_IDLE,
    ADC_BUSY,
    ADC_COMPLETED,
    ADC_STREAM_COMPLETED,
    ADC_ERROR,
    ADC_DIAG
}  Adc_StatusType;
```

Description

Current status of the conversion of the ADC channel group.

- `ADC_IDLE`: The conversion of the group has not been started, and results are unavailable.
- `ADC_BUSY`: The conversion of the group has been started and is ongoing, and results are unavailable.
- `ADC_COMPLETED`: A conversion round (not the final round) of the group has been finished. A result is available for all channels of the group.
- `ADC_STREAM_COMPLETED`: The result buffer is completely filled. For each channel of the group, the number of samples to be acquired is available.
- `ADC_ERROR`: A DMA transmission error occurred, and results are unavailable.
- `ADC_DIAG`: The conversion of the group has been on-going for doing SelfDiag.

7.2.6 Adc_SamplingTimeType

Type

```
uint16
```

Description

Type of sampling time, that is, the time during which the value is sampled (in clock-cycles).

7.2.7 Adc_PowerStateRequestResultType

Type

```
typedef enum adc_pwerstaterequestresulttype_enum
{
    ADC_SERVICE_ACCEPTED,
    ADC_NOT_INIT,
    ADC_SEQUENCE_ERROR,
    ADC_HW_FAILURE,
    ADC_POWER_STATE_NOT_SUPP,
    ADC_TRANS_NOT_POSSIBLE
}  Adc_PowerStateRequestResultType
```

Description

Result of the requests related to power state transitions.

- `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 from entering the required power state.
- `ADC_POWER_STATE_NOT_SUPP`: ADC module does not support the requested power state.
- `ADC_TRANS_NOT_POSSIBLE`: ADC module cannot transition directly from the current power state to the requested power state or the HW peripheral is still busy.

7.2.8 Adc_PowerStateType

Type

```
typedef enum adc_powerstatetype_enum
{
    ADC_FULL_POWER,
    ADC_OFF_POWER
}  Adc_PowerStateType;
```

Description

Power state currently active or set as target power state.

- `ADC_FULL_POWER`: Full Power (0).
- `ADC_OFF_POWER`: Off Power (1).

7.2.9 Adc_ConfigType

Type

```
typedef struct
```

Description

This type describes the driver initialization structure. Architecture specific and global driver settings are saved.

7.2.10 Adc_ChannelRangeSelectType

Type

```
typedef enum
{
    ADC_RANGE_UNDER_LOW,
    ADC_RANGE_BETWEEN,
    ADC_RANGE_OVER_HIGH,
    ADC_RANGE_ALWAYS,
    ADC_RANGE_NOT_UNDER_LOW,
    ADC_RANGE_NOT_BETWEEN,
    ADC_RANGE_NOT_OVER_HIGH
```

```
}  Adc_ChannelRangeSelectType;
```

Description

Type for configuring the range of limit checking and how the conversion result is taken into account for updating result buffer.

- `ADC_RANGE_UNDER_LOW`: Range below low limit – low limit value included.
- `ADC_RANGE_BETWEEN`: Range between low limit and high limit – high limit value included.
- `ADC_RANGE_OVER_HIGH`: Range above high limit.
- `ADC_RANGE_ALWAYS`: Complete range – independent of channel limit settings.
- `ADC_RANGE_NOT_UNDER_LOW`: Range above low limit.
- `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.

Note: *The configured range select is managed by a value to set directly to the hardware register instead of this type, so this type is not used by ADC driver.*

7.2.11 Adc_PrescaleType

Type

`uint32`

Description

Type of clock prescaler factor.

Note: *The hardware controlled by ADC module does not support the prescale feature, so this type is not used by ADC driver.*

7.2.12 Adc_ConversionTimeType

Type

`uint8`

Description

Type of conversion time, that is, the time during which the sampled analog value is converted into digital representation.

Note: *The conversion time is fixed by hardware, so this type is not used by ADC driver.*

7.2.13 Adc_ResolutionType

Type

`uint8`

Description

Type of channel resolution in number of bits.

Note: *Only 12-bit resolution is available, so this type is not used by ADC driver.*

7.2.14 Adc_TriggerSourceType

Type

```
typedef enum adc_triggersourcetype_enum
{
  ADC_TRIGG_SRC_SW,
  ADC_TRIGG_SRC_HW
} Adc_TriggerSourceType;
```

Description

Type for configuring the trigger source for an ADC channel group.

- `ADC_TRIGG_SRC_SW`: Group is triggered by a software API call.
- `ADC_TRIGG_SRC_HW`: Group is triggered by a hardware event.

7.2.15 Adc_GroupConvModeType

Type

```
typedef enum adc_groupconvmodetype_enum
{
  ADC_CONV_MODE_ONESHOT,
  ADC_CONV_MODE_CONTINUOUS
} Adc_GroupConvModeType;
```

Description

Type for configuring the conversion mode of an ADC channel group.

- `ADC_CONV_MODE_ONESHOT`: 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, an ongoing One-Shot conversion can be stopped by a software API call. In case of "group trigger source hardware", an ongoing One-Shot conversion can be stopped by disabling the trigger event (if supported by hardware).
- `ADC_CONV_MODE_CONTINUOUS`: Repeated conversions of each ADC channel in an ADC channel group are performed. "Continuous conversion mode" is only available for "group trigger source software". An ongoing "Continuous conversion" can be stopped by a software API call.

7.2.16 Adc_GroupPriorityType

Type

```
uint8
```

Description

Priority level of the channel.

7.2.17 Adc_GroupDefType

Type

`uint32`

Description

Type for assignment of channels to a channel group.

Note: *The ADC driver can control the assignment of channels to a group without using this type, so this type is not used by ADC Driver.*

7.2.18 Adc_StreamBufferModeType

Type

```
typedef enum adc_streambuffermodetype_enum
{
  ADC_STREAM_BUFFER_LINEAR,
  ADC_STREAM_BUFFER_CIRCULAR
} Adc_StreamBufferModeType;
```

Description

Type for configuring the streaming access mode buffer type.

- `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.

7.2.19 Adc_GroupAccessModeType

Type

```
typedef enum adc_groupaccessmodetype_enum
{
  ADC_ACCESS_MODE_SINGLE,
  ADC_ACCESS_MODE_STREAMING
} Adc_GroupAccessModeType;
```

Description

Type for configuring the access mode to group conversion results.

- `ADC_ACCESS_MODE_SINGLE`: Single value access mode.
- `ADC_ACCESS_MODE_STREAMING`: Streaming access mode.

7.2.20 Adc_HwTriggerSignalType

Type

```
typedef enum adc_hwtriggersignaltype_enum
{
  ADC_HW_TRIG_RISING_EDGE,
  ADC_HW_TRIG_FALLING_EDGE,
  ADC_HW_TRIG_BOTH_EDGES
} Adc_HwTriggerSignalType;
```

Description

Type for configuring the edge of the hardware trigger signal the driver that should react, that is, start the conversion.

- `ADC_HW_TRIG_RISING_EDGE`: React on the rising edge of the hardware trigger signal.
- `ADC_HW_TRIG_FALLING_EDGE`: React on the falling edge of the hardware trigger signal.
- `ADC_HW_TRIG_BOTH_EDGES`: React on both edges of the hardware trigger signal.

Note: The hardware controlled by ADC driver does not have this feature, so this type is not used by ADC Driver.

7.2.21 Adc_HwTriggerTimerType

Type

```
uint32
```

Description

Type for the reload value of the ADC module embedded timer.

Note: The hardware controlled by ADC driver does not have this feature, so this type is not used by ADC Driver.

7.2.22 Adc_PriorityImplementationType

Type

```
typedef enum adc_priorityimplementationtype_enum
{
  ADC_PRIORITY_NONE,
  ADC_PRIORITY_HW,
  ADC_PRIORITY_HW_SW
} Adc_PriorityImplementationType;
```

Description

Type for configuring the prioritization mechanism.

- `ADC_PRIORITY_NONE`: Priority mechanism is not available.
- `ADC_PRIORITY_HW`: Hardware priority mechanism is available only.

- ADC_PRIORITY_HW_SW: Hardware and software priority mechanism is available.

Note: *Only ADC_PRIORITY_HW is supported and HW priority mechanism is used for all groups (including SW triggered), so this type is not used by ADC Driver.*

7.2.23 Adc_GroupReplacementType

Type

```
typedef enum adc_groupreplacementtype_enum
{
  ADC_GROUP_REPL_ABORT_RESTART = 1U,
  ADC_GROUP_REPL_ABORT_RESUME,
  ADC_GROUP_REPL_SUSPEND_RESUME
} Adc_GroupReplacementType;
```

Description

Replacement mechanism used on ADC group level if a group conversion is interrupted by a group that has a higher priority.

- ADC_GROUP_REPL_ABORT_RESTART: Abort/Restart mechanism is used on group level, if a group is interrupted by a higher priority group. The complete conversion round of the interrupted group (all group channels) is restarted after the higher priority group conversion is finished. If the group is configured in streaming access mode, only the results of the interrupted conversion round are discarded. Results of previous conversion rounds, which are already written to the result buffer are not affected.
- ADC_GROUP_REPL_ABORT_RESUME: Abort/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. Immediately abort the ongoing acquisition and on return resume the group scan starting with the aborted channel.
- ADC_GROUP_REPL_SUSPEND_RESUME: 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. Before preempting, complete the ongoing acquisition and on return resume the group scan from the next channel.

7.2.24 Adc_ResultAlignmentType

Type

```
typedef enum adc_resultalignmenttype_enum
{
  ADC_ALIGN_LEFT,
  ADC_ALIGN_RIGHT
} Adc_ResultAlignmentType;
```

Description

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

- ADC_ALIGN_LEFT: Left alignment.

- `ADC_ALIGN_RIGHT`: Right alignment.

Note: *The configured result alignment is managed by a value to set directly to the hardware register instead of this type, so this type is not used by ADC driver.*

7.2.25 Adc_HwUnitType

Type

`uint8`

Description

This type represents a hardware unit ID.

7.2.26 Adc_OffsetValueType

Type

`sint8`

Description

This type represents an analog offset correction value and is used for calibration setting.

7.2.27 Adc_GainValueType

Type

`sint8`

Description

This type represents an analog gain correction value and is used for calibration setting.

7.2.28 Adc_SignalType

Type

`uint8`

Description

This type represents an input analog signal.

7.2.29 Adc_DataReadType

Type

```
typedef enum adc_datareadtype_enum
{
  ADC_DATA_READ,
  ADC_DATA_UNREAD
} Adc_DataReadType;
```

Description

This type represents "read" or "unread" from data of buffer.

- `ADC_DATA_READ`: Data from buffer was read.
- `ADC_DATA_UNREAD`: Data from buffer was unread.

7.2.30 Adc_GroupHwTriggSrcType

Type

```
typedef enum adc_grouphwtriggsrctype_enum
{
  ADC_HWTRIGG_SRC_OFF,
  ADC_HWTRIGG_SRC_TCPWM,
  ADC_HWTRIGG_SRC_GENERIC0,
  ADC_HWTRIGG_SRC_GENERIC1,
  ADC_HWTRIGG_SRC_GENERIC2,
  ADC_HWTRIGG_SRC_GENERIC3,
  ADC_HWTRIGG_SRC_GENERIC4,
  ADC_HWTRIGG_SRC_CONTINUOUS
} Adc_GroupHwTriggSrcType;
```

Description

Type for specifying which hardware trigger is used.

- `ADC_HWTRIG_SRC_OFF`: Not used as a hardware trigger. it is used in the case of software trigger.
- `ADC_HWTRIG_SRC_TCPWM`: Use TCPWM hardware trigger.
- `ADC_HWTRIG_SRC_GENERIC0`: Use GENERIC0 hardware trigger.
- `ADC_HWTRIG_SRC_GENERIC0`: Use GENERIC1 hardware trigger.
- `ADC_HWTRIG_SRC_GENERIC0`: Use GENERIC2 hardware trigger.
- `ADC_HWTRIG_SRC_GENERIC0`: Use GENERIC3 hardware trigger.
- `ADC_HWTRIG_SRC_GENERIC0`: Use GENERIC4 hardware trigger.
- `ADC_HWTRIG_SRC_CONTINUOUS`: Not used as a hardware trigger. It is used in the case of software trigger.

7.2.31 Adc_CoreIdType

Type

`Uint8`

Description

Numeric ID of a core.

7.2.32 Adc_DriverStatusType

Type

```
typedef enum adc_driverstatustype_enum
{
  ADC_DIVER_UNINIT,
  ADC_DIVER_INITIALIZED
} Adc_DriverStatusType;
```

Description

Type for specifying which driver status is used.

- `ADC_DRIVER_UNINIT`: Uninitialized state.
- `ADC_DRIVER_INITIALIZED`: Initialized state.

7.3 Constants

7.3.1 Error codes

A service may return one of the error codes, listed in Table 10, if development error detection is enabled.

Table 10 Error codes

Name	Value	Description
ADC_E_UNINIT	0x0A	Driver is not initialized.
ADC_E_BUSY	0x0B	Analog-to-digital conversion is already started.
ADC_E_IDLE	0x0C	Analog-to-digital conversion is not started.
ADC_E_ALREADY_INITIALIZED	0x0D	Driver is already initialized.
ADC_E_PARAM_CONFIG	0x0E	Configuration pointer is non-NULL pointer.
ADC_E_PARAM_POINTER	0x14	NULL pointer is given.
ADC_E_PARAM_GROUP	0x15	Invalid group ID is given.
ADC_E_WRONG_CONV_MODE	0x16	Group in continuous conversion mode is given.
ADC_E_WRONG_TRIGG_SRC	0x17	Group with wrong trigger source is given.
ADC_E_NOTIF_CAPABILITY	0x18	Group notification capability is not activated.
ADC_E_BUFFER_UNINIT	0x19	Buffer is not set up.
ADC_E_NOT_DISENGAGED	0x1A	ADC groups are not in IDLE state.
ADC_E_POWER_STATE_NOT_SUPPORTED	0x1B	Unsupported power state is requested.
ADC_E_TRANSITION_NOT_POSSIBLE	0x1C	Requested power state cannot be reached directly.
ADC_E_PERIPHERAL_NOT_PREPARED	0x1D	ADC is not prepared for target power state yet.
ADC_E_PARAM_CHANNEL	0x2A	Channel is not in the group.
ADC_E_PARAM_HWUNIT	0x2B	HW unit is out of valid range.
ADC_E_PARAM_SAMPLING_TIME	0x2C	Sampling time is not in valid range.
ADC_E_CONVERSION_ERROR	0x2D	Conversion is an error.
ADC_E_HARDWARE_ERROR_FOR_CALLOUT	0x2E	Hardware product error occurs (this id is for callout).

Name	Value	Description
ADC_E_PARAM_GAIN	0x30	Analog gain correction value is out of range.
ADC_E_PARAM_UPDATE	0x31	Update value is incorrect.
ADC_E_PARAM_SIGNAL	0x32	Invalid input signal.
ADC_E_INVALID_CORE	0x33	Invalid core ID was passed on as parameter.
ADC_E_INIT_FAILED	0x34	Satellite core is initialized before master core.
ADC_E_DIFFERENT_CONFIG	0x35	Mismatch between the configuration of master core and satellite core.
ADC_E_REDUNDANCY_ERROR	0xA0	Mismatch between buffer and redundancy buffer
ADC_E_CHANNEL_ID_NG	0xA1	Invalid channel ID
ADC_E_PARAM_SELECT_TRIGG	0xA2	Invalid select trigger
ADC_E_PARAM_THRESHOLD_VALE	0xA3	Threshold value is not in valid range.
ADC_E_DIAG	0xA4	SelfDiag is already started.

7.3.2 Version information

Table 11 Version information

Name	Value	Description
ADC_SW_MAJOR_VERSION	See release notes	Vendor specific major version number.
ADC_SW_MINOR_VERSION	See release notes	Vendor specific minor version number.
ADC_SW_PATCH_VERSION	See release notes	Vendor specific patch version number.

7.3.2.1 Module information

Table 12 Module information

Name	Value	Description
ADC_MODULE_ID	123	Module ID (ADC).
ADC_VENDOR_ID	66	Vendor ID.

7.3.3 API service IDs

The API service IDs, listed in Table 13, are used when reporting errors via DET or via the error callout function:

Table 13 API service IDs

Name	Value	API name
ADC_API_INIT	0x00	Adc_Init
ADC_API_DEINIT	0x01	Adc_DeInit
ADC_API_START_GROUP_CONVERSION	0x02	Adc_StartGroupConversion
ADC_API_STOP_GROUP_CONVERSION	0x03	Adc_StopGroupConversion
ADC_API_READ_GROUP	0x04	Adc_ReadGroup
ADC_API_ENABLE_HARDWARE_TRIGGER	0x05	Adc_EnableHardwareTrigger
ADC_API_DISABLE_HARDWARE_TRIGGER	0x06	Adc_DisableHardwareTrigger

Name	Value	API name
ADC_API_ENABLE_GROUP_NOTIFICATION	0x07	Adc_EnableGroupNotification
ADC_API_DISABLE_GROUP_NOTIFICATION	0x08	Adc_DisableGroupNotification
ADC_API_GET_GROUP_STATUS	0x09	Adc_GetGroupStatus
ADC_API_GET_VERSION_INFO	0x0A	Adc_GetVersionInfo
ADC_API_GET_STREAM_LAST_POINTER	0x0B	Adc_GetStreamLastPointer
ADC_API_SETUP_RESULT_BUFFER	0x0C	Adc_SetupResultBuffer
ADC_API_SET_POWER_STATE	0x10	Adc_SetPowerState
ADC_API_GET_CURRENT_POWER_STATE	0x11	Adc_GetCurrentPowerState
ADC_API_GET_TARGET_POWER_STATE	0x12	Adc_GetTargetPowerState
ADC_API_PREPARE_POWER_STATE	0x13	Adc_PreparePowerState
ADC_API_MAIN_POWER_TRANSITION_MANAGER	0x14	Adc_Main_PowerTransitionManager
ADC_API_CHANGE_SAMPLING_TIME	0x30	Adc_ChangeSamplingTime
ADC_API_CHANGE_CALIBRATION_CHANNEL	0x31	Adc_ChangeCalibrationChannel
ADC_API_SET_CALIBRATION_VALUE	0x32	Adc_SetCalibrationValue
ADC_API_GET_CALIBRATION_ALTERNATE_VALUE	0x33	Adc_GetCalibrationAlternateValue
ADC_API_GET_CALIBRATION_VALUE	0x34	Adc_GetCalibrationValue
ADC_API_ISR	0x40	Interrupt service for ADC
ADC_API_DISABLE_CHANNEL	0x50	Adc_DisableChannel
ADC_API_ENABLE_CHANNEL	0x51	Adc_EnableChannel
ADC_API_GET_ADC_ADDR	0x52	Adc_GetADCAddr
ADC_API_READ_CHANNEL_VALUE	0x53	Adc_ReadChannelValue
ADC_API_GET_LIMIT_CHECK_STATE	0x54	Adc_GetGroupLimitCheckState
ADC_API_SELECT_CHANNEL_THRESHOLD	0x55	Adc_SelectChannelThreshold
ADC_API_ENABLE_HW_TRIGGER	0x56	Adc_EnableHwTrigger
ADC_API_DISABLE_HW_TRIGGER	0x57	Adc_DisableHwTrigger
ADC_API_START_DIAGNOSTIC_FULL	0x58	Adc_StartDiagnosticFull
ADC_API_GET_DIAGNOSTIC_RESULT	0x59	Adc_GetDiagnosticResult
ADC_API_START_DIAGNOSTIC	0x5A	Adc_StartDiagnostic

7.3.4 Invalid core ID value

Table 14 Invalid core ID

Name	Value	Description
ADC_INVALID_CORE	0xFF	Invalid core ID

7.4 Functions

7.4.1 Adc_Init

Syntax

```
void Adc_Init
(
    const Adc_ConfigType* ConfigPtr
)
```

Service ID

0x0

Parameters (in)

- `ConfigPtr` – Pointer to configuration set.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_ALREADY_INITIALIZED` – The ADC driver has already been initialized.
- `ADC_E_PARAM_CONFIG` – The configuration pointer points to an invalid configuration.
- `ADC_E_INIT_FAILED` –Satellite core is initialized before master core
- `ADC_E_DIFFERENT_CONFIG` – The configuration pointer is different from the initialized configuration of the master core.
- `ADC_E_INVALID_CORE` – The core ID is invalid.

DEM errors

None

Description

The function initializes the hardware ADC according to the post build configuration.

Note: *This service only affects ADC HwUnits assigned to the current core.*

7.4.2 Adc_DeInit

Syntax

```
void Adc_DeInit
(
    void
)
```

Service ID

0x1

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – driver is not initialized yet.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_INVALID_CORE` – The core ID is invalid.

DEM errors

None

Description

The function reinitializes all ADC hardware conversion units to a state comparable with the power-on reset state.

Note: *This service only affects ADC HwUnits assigned to the current core.*

7.4.3 Adc_StartGroupConversion

Syntax

```
void Adc_StartGroupConversion
(
    Adc_GroupType Group
)
```

Service ID

0x2

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_WRONG_TRIGG_SRC` – The group is not configured for SW triggered group.
- `ADC_E_BUFFER_UNINIT` – The group's result buffer is not set up.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

The function starts the conversion of all channels within the specified group, if the group is configured for SW triggered group.

7.4.4 Adc_StopGroupConversion

Syntax

```
void Adc_StopGroupConversion
(
    Adc_GroupType Group
)
```

Service ID

0x3

Parameters (in)

- `Group` – Numeric ID of required ADC channel group.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_IDLE` – No current ongoing conversion for the group.
- `ADC_E_WRONG_TRIGG_SRC` – The group is not configured for SW triggered group.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

This function stops the conversion of all channels within the specified group.

7.4.5 Adc_ReadGroup

Syntax

```
Std_ReturnType Adc_ReadGroup
(
    Adc_GroupType Group,
    Adc_ValueGroupType* DataBufferPtr
)
```

Service ID

0x4

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.
- `DataBufferPtr` – ADC results of all channels of the selected group are stored in the data buffer addressed by the pointer.

Parameters (out)

- None

Return value

- `E_OK` – Results are available and written to the data buffer.
- `E_NOT_OK` – No results are available or development error occurred.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_IDLE` – No current ongoing conversion for the group.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_CONVERSION_ERROR` – The group's status is `ADC_ERROR` (conversion is an error).
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.
- `ADC_E_REDUNDANCY_ERROR` – Result buffer and redundancy buffer do not match.

DEM errors

None

Description

Reads the group conversion result of the last completed conversion round of the requested group and stores the conversion result at the address pointed by `DataBufferPtr`.

7.4.6 Adc_EnableHardwareTrigger

Syntax

```
void Adc_EnableHardwareTrigger
(
    Adc_GroupType Group
)
```

Service ID

0x5

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_WRONG_CONV_MODE` – The group is configured in continuous mode.
- `ADC_E_WRONG_TRIGG_SRC` – The group is not configured for HW triggered group.
- `ADC_E_BUFFER_UNINIT` – The group's result buffer is not set up.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

The function enables the hardware trigger capability of a group configured in one-shot mode.

7.4.7 Adc_DisableHardwareTrigger

Syntax

```
void Adc_DisableHardwareTrigger
(
    Adc_GroupType Group
)
```

Service ID

0x6

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_WRONG_CONV_MODE` – The group is configured for continuous mode.
- `ADC_E_WRONG_TRIGG_SRC` – The group is not configured for HW triggered group.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.
- `ADC_E_IDLE` – No current ongoing conversion for the group.

DEM errors

None

Description

The function disables the hardware trigger capability of a group configured for one-shot mode.

7.4.8 Adc_EnableGroupNotification

Syntax

```
void Adc_EnableGroupNotification
(
    Adc_GroupType Group
)
```

Service ID

0x7

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_NOTIF_CAPABILITY` – The group notification function is NULL or notification capability is inactive.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

The function enables the group notification during runtime, if the notification capability is activated.

7.4.9 Adc_DisableGroupNotification

Syntax

```
void Adc_DisableGroupNotification
(
    Adc_GroupType Group
)
```

Service ID

0x8

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_NOTIF_CAPABILITY` – The group notification function is NULL or notification capability is inactive.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

The function disables the group notification during runtime, if the notification capability is activated.

7.4.10 Adc_GetGroupStatus

Syntax

```
Adc_StatusType Adc_GetGroupStatus
(
    Adc_GroupType Group
)
```

Service ID

0x9

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

Conversion status for the requested group. `ADC_IDLE` is always returned when DET error is detected.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

- `ADC_E_HARDWARE_ERROR` – Hardware product error occurs. `ADC_E_HARDWARE_ERROR_FOR_CALLOUT` will be passed instead of `ADC_E_HARDWARE_ERROR` in case of the error callout handler.

Description

Returns the conversion status of the requested group.

7.4.11 Adc_GetVersionInfo

Syntax

```
void Adc_GetVersionInfo
(
    Std_VersionInfoType* versioninfo
)
```

Service ID

0xA

Parameters (in)

None

Parameters (out)

- `versioninfo` – Pointer to store the version information of this module.

Return value

None

DET errors

- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.

DEM errors

None

Description

Returns the version of the ADC driver in a `Std_VersionInfoType` structure.

7.4.12 Adc_GetStreamLastPointer

Syntax

```
Adc_StreamNumSampleType Adc_GetStreamLastPointer
(
    Adc_GroupType Group,
    Adc_ValueGroupType** PtrToSamplePtr
)
```

Service ID

0xB

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.

Parameters (out)

- `PtrToSamplePtr` – Pointer to a pointer that will be set to point to the last sampled value.

Return value

Number of valid samples per channel. The value 0 is always returned when DET error is detected or the group is in `ADC_BUSY`.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_IDLE` – No current ongoing conversion for the group.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_CONVERSION_ERROR` – The group's status is ADC_ERROR (conversion is an error).
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.
- `ADC_E_REDUNDANCY_ERROR` – Result buffer and redundancy buffer do not match.

DEM errors

None

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 and the return value, all valid group conversion results can be accessed.

7.4.13 Adc_SetupResultBuffer

Syntax

```
Std_ReturnType Adc_SetupResultBuffer
(
    Adc_GroupType Group,
    Adc_ValueGroupType* DataBufferPtr
)
```

Service ID

0xC

Parameters (in)

- `Group` – Numeric ID of the requested ADC channel group.
- `DataBufferPtr` – Pointer to result buffer.

Parameters (out)

None

Return value

- `E_OK` – Result buffer pointer initialized successfully.
- `E_NOT_OK` – Operation failed or development error occurred.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Initializes the group-specific ADC result buffer pointer as configured to point to the `DataBufferPtr` address, which is passed as parameter. The ADC driver stores all group conversion results in the result buffer addressed by the result buffer pointer. `Adc_SetupResultBuffer()` determines the address of the result buffer. After reset, before a group conversion can be started, ADC result buffer pointer must be initialized.

7.4.14 Adc_SetPowerState

Syntax

```
Std_ReturnType Adc_SetPowerState
(
    Adc_PowerStateRequestResultType* Result
)
```

Service ID

0x10

Parameters (in)

None

Parameters (out)

- `Result` – Pointer to store the result (one of the following definitions is stored):
  - `ADC_SERVICE_ACCEPTED` – Power state change executed (return value is E_OK).
  - *`ADC_NOT_INIT`* – ADC driver is not initialized.
  - `ADC_POWER_STATE_NOT_SUPP` – ADC driver does not support the requested power state.
  - `ADC_TRANS_NOT_POSSBILE` – The ADC HW peripheral is still busy.

Return value

- `E_OK` – Power mode changed.
- `E_NOT_OK` – Request rejected.

DET errors

- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_POWER_STATE_NOT_SUPPORTED` – Unsupported power state request.
- `ADC_E_NOT_DISENGAGED` – ADC group/channel not in IDLE state.
- `ADC_E_INVALID_CORE` – The core ID is invalid.

DEM errors

None

Description

This API configures the ADC driver so that it enters the already prepared power state; chosen from a predefined set of configured states.

Note: *This service only affects ADC HwUnits assigned to the current core.*

7.4.15 Adc_GetCurrentPowerState

Syntax

```
Std_ReturnType Adc_GetCurrentPowerState
(
    Adc_PowerStateType* CurrentPowerState,
    Adc_PowerStateRequestResultType* Result
)
```

Service ID

0x11

Parameters (in)

None

Parameters (out)

- `CurrentPowerState` – Returns the current power mode of the ADC HW unit.

Parameters (out)

- `Result` – Pointer to store the result (one of the following definitions is stored):
  - `ADC_SERVICE_ACCEPTED` – Power state change executed (return value is E_OK).
  - `ADC_NOT_INIT` – ADC driver is not initialized or the given parameter is a NULL pointer.

Return value

- `E_OK` – Mode could be read.
- `E_NOT_OK` – Service is rejected.

DET errors

- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_INVALID_CORE` – The core ID is invalid.

DEM errors

None

Description

This API returns the current power state of the ADC HW unit.

Note: This service only affects ADC HwUnits assigned to the current core.

7.4.16 Adc_GetTargetPowerState

Syntax

```
Std_ReturnType Adc_GetTargetPowerState
(
    Adc_PowerStateType* TargetPowerState,
    Adc_PowerStateRequestResultType* Result
)
```

Service ID

0x12

Parameters (in)

None

Parameters (out)

- `TargetPowerState` – Returns the target power mode of the ADC HW unit

Parameters (out)

- `Result` – Pointer to store the result (one of the following definitions is stored):
  - `ADC_SERVICE_ACCEPTED` – Power state change executed (return value is E_OK).
  - `ADC_NOT_INIT` – ADC driver is not initialized and the given parameter is a NULL pointer.

Return value

- `E_OK` – Mode could be read.
- `E_NOT_OK` – Service is rejected.

DET errors

- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_INVALID_CORE` – The core ID is invalid.

DEM errors

None

Description

This API returns the target power state of the ADC HW unit.

Note: This service only affected ADC HwUnits assigned to the current core.

7.4.17 Adc_PreparePowerState

Syntax

```
Std_ReturnType Adc_PreparePowerState
(
    Adc_PowerStateType PowerState,
    Adc_PowerStateRequestResultType* Result
)
```

Service ID

0x13

Parameters (in)

- `PowerState` – The target power state intended to be attained.

Parameters (out)

- `Result` – Pointer to store the result (one of the following definitions is stored):
  - `ADC_SERVICE_ACCEPTED` – Power state change executed (return value is E_OK).
  - `ADC_NOT_INIT` – ADC driver is not initialized.
  - `ADC_POWER_STATE_NOT_SUPP` – ADC driver does not support the requested power state.

Return value

- `E_OK` – Preparation process started.
- `E_NOT_OK` – Service is rejected.

DET errors

- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_POWER_STATE_NOT_SUPPORTED` – Unsupported power state request.
- `ADC_E_INVALID_CORE` – The core ID is invalid.

DEM errors

None

Description

This API starts the process required to allow the ADC HW module to enter the requested power state.

Note: This service only affects ADC HwUnits assigned to the current core.

7.4.18 Adc_Main_PowerTransitionManager

Syntax

```
void Adc_Main_PowerTransitionManager
(
    void
)
```

Service ID

0x14

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

None

DEM errors

None

Description

This API is cyclically called to supervise the power state transitions, check for the readiness of the module, and issue the callbacks.

This service is not supported because hardware does not support it.

7.4.19 Adc_ChangeSamplingTime

Syntax

```
Std_ReturnType Adc_ChangeSamplingTime
(
    Adc_GroupType Group,
    Adc_ChannelType Channel,
    Adc_SamplingTimeType SamplingTime
)
```

Service ID

0x30

Parameters (in)

- `Group` – Numeric ID of the requested ADC channel group.
- `Channel` – Numeric ID of the requested ADC channel.
- `SamplingTime` – Sampling time in clock cycles that is set to the ADC channel.

Parameters (out)

None

Return value

- `E_OK` – Sampling time changed.
- `E_NOT_OK` – Request rejected.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_PARAM_SAMPLING_TIME` – Sampling time is not in valid range.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_PARAM_CHANNEL` – Channel is not in the group.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

This API is used to change a sampling time for ADC channel, if a group to which the channel belongs is not running.

7.4.20 Adc_ChangeCalibrationChannel

Syntax

```
Std_ReturnType Adc_ChangeCalibrationChannel
(
    Adc_GroupType Group,
    Adc_SignalType Signal
)
```

Service ID

0x31

Parameters (in)

- `Group` – Numeric ID of requested ADC channel group.
- `Signal` – Calibration measurement analog signal (ADC_PIN_VREFL or ADC_PIN_VREFH).

Parameters (out)

- None

Return value

- `E_OK` – Analog input signal is changed correctly.
- `E_NOT_OK` – Request is rejected.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_SIGNAL` – Invalid input signal.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Changes calibration measurement analog signal (VrefL / VrefH) for the group converted by using alternate calibration values.

7.4.21 Adc_SetCalibrationValue

Syntax

```
Std_ReturnType Adc_SetCalibrationValue
(
    Adc_HwUnitType HwUnit,
    Adc_OffsetValueType Offset,
    Adc_GainValueType Gain,
    boolean Update
)
```

Service ID

0x32

Parameters (in)

- `HwUnit` – Numeric ID of requested ADC HW unit.
- `Offset` – Analog offset correction value for alternate calibration.
- `Gain` – Analog gain correction value for alternate calibration.
- `Update` – Flag indicates whether to update regular calibration value with alternate calibration values (TRUE: update regular calibration values with alternate calibration values, FALSE: do not update regular calibration values).

Parameters (out)

- None

Return value

- `E_OK` – Alternate calibration values are set correctly, and regular calibration values are updated if the parameter update is set to true.
- `E_NOT_OK` – Request is rejected.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GAIN` – Analog gain correction value is out of range.
- `ADC_E_PARAM_UPDATE` – Update value is incorrect.
- `ADC_E_PARAM_HWUNIT` – HW unit ID is invalid.
- `ADC_E_INVALID_CORE` – The current core and HwUnit assignment core are different. The core ID is invalid.

DEM errors

None

Description

Sets alternate calibration values for the requested ADC HW unit. If the Update parameter is true, triggers update regular calibration values with alternate calibration values.

7.4.22 Adc_GetCalibrationAlternateValue

Syntax

```
Std_ReturnType Adc_GetCalibrationAlternateValue
(
    Adc_HwUnitType HwUnit,
    Adc_OffsetValueType* OffsetPtr,
    Adc_GainValueType* GainPtr
)
```

Service ID

0x33

Parameters (in)

- `HwUnit` – Numeric ID of requested ADC HW unit.

Parameters (out)

- `OffsetPtr` – A pointer to the buffer used to store analog offset correction value for alternate calibration.
- `GainPtr` – A pointer to the buffer used to store analog gain correction value for alternate calibration.

Return value

- `E_OK` – Alternate calibration values are read correctly.
- `E_NOT_OK` – Request is rejected.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_PARAM_HWUNIT` – HW unit ID is invalid.
- `ADC_E_INVALID_CORE` – The current core and HwUnit assignment core are different. The core ID is invalid.

DEM errors

None

Description

Gets alternate calibration values of the requested ADC HW unit.

7.4.23 Adc_GetCalibrationValue

Syntax

```
Std_ReturnType Adc_GetCalibrationValue
(
    Adc_HwUnitType HwUnit,
    Adc_OffsetValueType* OffsetPtr,
    Adc_GainValueType* GainPtr
)
```

Service ID

0x34

Parameters (in)

- `HwUnit` – Numeric ID of requested ADC HW unit.

Parameters (out)

- `OffsetPtr` – A pointer to the buffer used to store analog offset correction value for regular calibration.
- `GainPtr` – A pointer to the buffer used to store analog gain correction value for regular calibration.

Return value

- `E_OK` – Regular calibration values of the requested ADC HW unit.
- `E_NOT_OK` – Request is rejected.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_PARAM_HWUNIT` – HW unit ID is invalid.
- `ADC_E_INVALID_CORE` – The current core and HwUnit assignment core are different. The core ID is invalid.

DEM errors

None

Description

Gets regular calibration values of the requested ADC HW unit.

7.4.24 Adc_DisableChannel

Syntax

```
void Adc_DisableChannel
(
    Adc_GroupType Group,
    Adc_ChannelType Channel
)
```

Service ID

0x50

Parameters (in)

- Group – Numeric ID of requested ADC channel group.
- Channel – Numeric ID of requested ADC channel.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_CHANNEL_ID_NG` – Channel ID is invalid.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Disables one channel in group.

7.4.25 Adc_EnableChannel

Syntax

```
void Adc_EnableChannel
(
    Adc_GroupType Group,
    Adc_ChannelType Channel
)
```

Service ID

0x51

Parameters (in)

- Group – Numeric ID of requested ADC channel group.
- Channel – Numeric ID of requested ADC channel.

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_CHANNEL_ID_NG` – Channel ID is invalid.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Enables one channel in group.

7.4.26 Adc_GetADCAddr

Syntax

```
uint32 Adc_GetADCAddr
(
    Adc_GroupType Group
)
```

Service ID

0x52

Parameters (in)

- Group – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

Address of the conversion result register.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_INVALID_CORE` – The core ID is invalid.

DEM errors

None

Description

Returns the address of the conversion result register of the first ADC channel of the requested ADC channel group.

7.4.27 Adc_ReadChannelValue

Syntax

```
Adc_DataReadType Adc_ReadChannelValue
(
    Adc_GroupType Group,
    Adc_ChannelType Channel,
    uint16 Adc_ChannelDataPtr
)
```

Service ID

0x53

Parameters (in)

- Group – Numeric ID of requested ADC channel group.
- Channel – Numeric ID of requested ADC channel.

Parameters (out)

- Adc_ChannelDataPtr – Pointer to ADC channel result buffer.

Return value

- `ADC_DATA_READ` – Data from buffer was read.
- `ADC_DATA_UNREAD` – Data from buffer was unread.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.
- `ADC_E_IDLE` – No current ongoing conversion for the group.
- `ADC_E_CONVERSION_ERROR` – The group's status is `ADC_ERROR` (conversion is an error).
- `ADC_E_PARAM_CHANNEL` – Channel is not in the group.
- `ADC_E_REDUNDANCY_ERROR` – Mismatch between buffer and redundancy buffer

DEM errors

None

Description

Indicates whether the conversion result was already read.

7.4.28 Adc_GetGroupLimitCheckState

Syntax

```
Std_ReturnType Adc_GetGroupLimitCheckState
(
    Adc_GroupType Group,
    uint32 GroupIntrStatePtr
)
```

Service ID

0x54

Parameters (in)

- Group – Numeric ID of requested ADC channel group.

Parameters (out)

- GroupIntrStatePtr – Pointer to where to store the result.

Return value

- `E_OK` – Results are available and getting result of limit check from group.
- `E_NOT_OK` – No results are available, or development error occurred.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_IDLE` – No current ongoing conversion for the group.
- `ADC_E_CONVERSION_ERROR` – The group's status is `ADC_ERROR` (conversion is an error).
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Returns the result of limit check from group.

7.4.29 Adc_SelectChannelThreshold

Syntax

```
Std_ReturnType Adc_SelectChannelThreshold
(
    Adc_GroupType Group,
    Adc_ChannelType Channel,
    uint16 UpperLimit,
    uint16 LowerLimit
)
```

Service ID

0x55

Parameters (in)

- Group – Numeric ID of requested ADC channel group.
- Channel – Numeric ID of requested ADC channel.
- UpperLimit – Upper threshold for range comparator.
- LowerLimit – Lower threshold for range comparator.

Parameters (out)

None

Return value

- `E_OK` – Threshold changed.
- `E_NOT_OK` – Request rejected.

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_PARAM_CHANNEL` – Channel is not in the group.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.
- `ADC_E_PARAM_THRESHOLD_VALUE` – UpperLimit, LowerLimit, or both are invalid.

DEM errors

None

Description

Changes a threshold for ADC channel, if a group belonging to the channel is not running.

7.4.30 Adc_EnableHwTrigger

Syntax

```
void Adc_EnableHwTrigger
(
    Adc_GroupType Group,
    Adc_HwTriggerTimerType SelectTrigger
)
```

Service ID

0x56

Parameters (in)

- Group – Numeric ID of requested ADC channel group.
- SelectTrigger – HW trigger source. The value to be set is a value that can be used as a hardware trigger in "Adc_GroupHwTriggSrcType".

Parameters (out)

None

Return value

None

DET errors

- `ADC_E_UNINIT` – Driver is not initialized yet.
- `ADC_E_PARAM_GROUP` – Group ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_WRONG_CONV_MODE` – The group is configured in continuous mode.
- `ADC_E_WRONG_TRIGG_SRC` – The group is not configured for HW triggered group.
- `ADC_E_BUFFER_UNINIT` – The group's result buffer is not set up.

- ADC_E_PARAM_SELECT_TRIGG – The select trigger is invalid.
- ADC_E_INVALID_CORE – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Enables the hardware trigger for the requested ADC channel group and uses specified HW trigger.

7.4.31 Adc_DisableHwTrigger

Syntax

```
void Adc_DisableHwTrigger
(
    Adc_GroupType Group
)
```

Service ID

0x57

Parameters (in)

- Group – Numeric ID of requested ADC channel group.

Parameters (out)

None

Return value

None

DET errors

- ADC_E_UNINIT – Driver is not initialized yet.
- ADC_E_PARAM_GROUP – Group ID is invalid.
- ADC_E_WRONG_CONV_MODE – The group is configured for continuous mode.
- ADC_E_WRONG_TRIGG_SRC – The group is not configured for HW triggered group.
- ADC_E_IDLE – No current ongoing conversion for the group.
- ADC_E_INVALID_CORE – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Disables the hardware trigger for the requested ADC channel group.

7.4.32 Adc_StartDiagnosticFull

Syntax

```
Std_ReturnType Adc_StartDiagnosticFull
(
    Adc_HwUnitType HwUnit,
    uint32 * DiagResultPtr,
    boolean * OutputPtr
)
```

Service ID

0x58

Parameters (in)

- HwUnit – Numeric ID of the requested ADC HW unit.

Parameters (out)

- DiagResultPtr – Pointer to the result of the DiagnosticFull function (for `Adc_GetDiagnosticResult`). Buffer needs 32bit * 6.
- OutputPtr – Pointer to the SelfDiag result buffer. TRUE means that all channels are correct. FALSE means that at least one channel is NG.

Return value

- `E_OK` – Service has been performed.
- `E_NOT_OK` – Service has been rejected for reasons such as a bad parameter or status.

DET errors

- `ADC_E_UNINIT` – Driver is not yet initialized.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_PARAM_HWUNIT` – HW unit ID is invalid.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Diagnostics for all channels of the HW unit. All channels dedicated with the group of the HW unit are verified through converting internal DIAG to determine whether connectivity and correctness are within acceptable deviation.

7.4.33 Adc_GetDiagnosticResult

Syntax

```
Std_ReturnType Adc_GetDiagnosticResult
(
    Adc_ChannelType Channel,
    uint32 * DiagResultPtr,
    boolean * OutputPtr
)
```

Service ID

0x59

Parameters (in)

- Channel – Numeric ID of the requested ADC channel.

Parameters (out)

- DiagResultPtr – Pointer to the result of the DiagnosticFull function.
- OutputPtr – Pointer to the SelfDiag result buffer. TRUE means that a specific channel is correct. FALSE means that a specific channel is NG.

Return value

- `E_OK` – Service has been performed.
- `E_NOT_OK` – Service has been rejected for reasons such as a bad parameter or status.

DET errors

- `ADC_E_UNINIT` – Driver is not yet initialized.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_PARAM_CHANNEL` – Channel is not in the group, or the channel is currently running for diagnosis.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Identify the ADC channel from the detailed information. The SelfDiag result of a specific channel of group is returned. If a specific channel is configured as `DIAGNOSIS_NO`, the return value is `E_NOT_OK`.

7.4.34 Adc_StartDiagnostic

Syntax

```
Std_ReturnType Adc_StartDiagnostic
(
    Adc_ChannelType Channel,
    boolean * OutputPtr
)
```

Service ID

0x5A

Parameters (in)

- Channel – Numeric ID of the requested ADC channel.

Parameters (out)

- OutputPtr – Pointer to SelfDiag result buffer. TRUE means that a specific channel is correct. FALSE means that a specific channel is NG.

Return value

- `E_OK` – Service has been performed.
- `E_NOT_OK` – Service has been rejected for reasons such as a bad parameter or status.

DET errors

- `ADC_E_UNINIT` – Driver is not yet initialized.
- `ADC_E_PARAM_POINTER` – The given parameter is a NULL pointer.
- `ADC_E_PARAM_CHANNEL` – Channel is not in the group, or the channel is currently running for diagnosis.
- `ADC_E_BUSY` – The conversion is currently ongoing.
- `ADC_E_INVALID_CORE` – The current core and group assignment core associated with the HwUnit are different. The core ID is invalid.

DEM errors

None

Description

Diagnostic-specific channel of the HW unit. A specific channel dedicated with the group is verified through converting internal DIAG to determine whether connectivity and correctness are within acceptable deviation.

7.5 Required callback functions

7.5.1 Default error tracer (DET)

If development error detection is enabled, the ADC driver uses the following callback function provided by DET. If you do not use DET, you must implement this function within your application.

7.5.1.1 Det_ReportError

Syntax

```
Std_ReturnType Det_ReportError
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` – Module ID of calling module.
- `InstanceId` – `AdcCoreConfigurationId` of the core that calls this function or ADC_INVALID_CORE.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected development error.

Return value

Returns always `E_OK` (is required for services).

Description

Service for reporting development errors.

7.5.2 Diagnostic event manager (DEM)

If DEM notifications are enabled, the ADC driver uses the following callback function provided by DEM. If you do not use DEM, you must implement this function within your application.

7.5.2.1 Dem_ReportErrorStatus

Syntax

```
void Dem_ReportErrorStatus
(
    Dem_EventIdType EventId,
    Dem_EventStatusType EventStatus
)
```

Reentrancy

Reentrant

Parameters (in)

- `EventId` – Identification of an event which is assigned by DEM.
- `EventStatus` – Monitor test result of given event.

Return value

None

Description

Service for reporting diagnostic events.

7.5.3 Callout functions

7.5.3.1 Error callout API

The AUTOSAR ADC module requires an error callout handler. Each error is reported to this handler, error checking cannot be switched OFF. The name of the function to be called can be configured by the `AdcErrorCalloutFunction` parameter.

Syntax

```
void Error_Handler_Name
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` – Module ID of calling module.
- `InstanceId` – AdcCoreConfigurationId of the core that calls this function or ADC_INVALID_CORE.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected error.

Return value

None

Description

Service for reporting errors.

7.5.3.2 Get core ID API

The AUTOSAR ADC module requires a function to get the valid core ID. This function is being used to determine the core from which the code is getting executed. The name of the function to be called can be configured by `AdcGetCoreIdFunction` parameter.

Syntax

`uint8 GetCoreID_Function_Name (void)`

Reentrancy

Reentrant

Parameters (in)

None

Return value

- `CoreId` - ID of the current core.

Description

Service for getting valid core ID.

Note: *This function will return the core ID configured in `AdcCoreId` on `AdcConreConfiguration` within `AdcMulticore`.*
For example, two cores are configured in `AdcCoreConfiguration`.

Executing core	AdcCoreConfigurationId	AdcCoreId
CM7_0	0	15
CM7_1	1	16

Upon calling this function from core CM7_0, it shall return 15.

Upon calling this function from core CM7_1, it shall return 16.

8 Appendix B – Access register table

8.1 SAR ADC

Table 15 SAR ADC access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SARn_CTL	31:0	Word (32 bits)	0x00000000 or 0xE0000400	This register globally controls the SAR setting.	Adc_Init Adc_DeInit Adc_SetPowerState	0x000007FF	0x*0000400 (After Adc_Init Digit * depends on configuration value.) 0x00000000 (After Adc_DeInit.)
SARn_DIAG_CTL	31:0	Word (32 bits)	0x00000000 | SAR IP enabled/disabled << 31 | diagnostic reference selection (These items depend on configuration.)	This register controls the diagnostic reference function.	Adc_Init Adc_SetPowerState Adc_DeInit	0x0000000F	0x0000000* (After Adc_Init Digit * depends on configuration value.) 0x00000000 (After Adc_DeInit.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SARn_PRECOND_CTL	31:0	Word (32 bits)	0x00000000 | preconditioning time (These items depend on configuration.)	This register sets preconditioning time in clock cycles.	Adc_Init Adc_DeInit	0x0000000F	0x0000000* (After Adc_Init Digit * depends on configuration value.) 0x00000000 (After Adc_DeInit.)
SARn_ANA_CAL	31:0	Word (32 bits)	0x00000000 | Gain value << 16 | Offset value<< 0	Current analog calibration values	Adc_Init Adc_DeInit Adc_SetCalibrationValue (not writing directly)	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_DIG_CAL	31:0	Word (32 bits)	0x00000000	Current digital calibration values	Adc_Init Adc_DeInit	0x003F0FFF	0x00000000
SARn_ANA_CAL_ALT	31:0	Word (32 bits)	0x00000000 | Gain value << 16 | Offset value << 0	Alternate analog calibration values	Adc_Init Adc_DeInit Adc_SetCalibrationValue	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_DIG_CAL_ALT	31:0	Word (32 bits)	0x00000000	Alternate digital calibration values	Adc_Init Adc_DeInit	0x003F0FFF	0x00000000
SARn_CAL_UPD_CMD	31:0	Word (32 bits)	0x00000001 or 0x00000000	Calibration update command	Adc_SetCalibrationValue	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_TR_PEND	31:0	Word (32 bits)	-	Trigger pending status	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SARn_WORK_VALID	31:0	Word (32 bits)	-	Channel working data register 'valid' bits	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_WORK_RANGE	31:0	Word (32 bits)	-	Range detected	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_WORK_RANGE_HI	31:0	Word (32 bits)	-	Range detect above Hi flag	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_WORK_PULSE	31:0	Word (32 bits)	-	Pulse detected	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_RESULT_VALID	31:0	Word (32 bits)	-	Channel result data register 'valid' bits	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_RESULT_RANGE_HI	31:0	Word (32 bits)	-	Channel range above Hi flags	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_STATUS	31:0	Word (32 bits)	-	Current status of internal SAR registers (mostly for debug)	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_AVG_STAT	31:0	Word (32 bits)	-	Current averaging status (for debug)	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SARn_CHx_STRUCT_TR_CTL	31:0	Word (32 bits)	0x00000000 | last channel of a group << 11 | preemption type << 8 | channel priority << 4 | trigger select (These items depend on configuration.)	This register controls the trigger function for the channel.	Adc_Init Adc_DeInit Adc_DisableChannel Adc_EnableChannel Adc_EnableHardwareTrigger Adc_EnableHwTrigger Adc_StartGroupConversion Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x80000B77	0x00000*** (After Adc_Init Digit * depends on configuration value.) 0x00000800 (After Adc_DeInit.)
SARn_CHx_STRUCT_SAMPLE_CTL	31:0	Word (32 bits)	0x00000000 | alternate calibration << 31 | sample time << 16 | overlap mode or SARMUX diagnostics << 14 | preconditioning mode << 12 | external_mul << 8 |external_mul_en << 11 | physical port (only ADC0) << 6 | address of the analog signal (pin)	This register controls the sampling function for the channel.	Adc_Init Adc_DeInit Adc_ChangeSamplingTime Adc_ChangeCalibrationChannel Adc_StartGroupConversion Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x8000F0FF	0x*000*0** (After Adc_Init Digit * depends on configuration value.) 0x00000000 (After Adc_DeInit.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
			(These items depend on configuration.)				
SARn_CHx_STRUCT_POST_CTL	31:0	Word (32 bits)	0x00000000 | range detect mode << 22 | sign extended << 7 | left or right align << 6 | post processing (These items depend on configuration.)	This register controls post processing.	Adc_Init Adc_DeInit Adc_StartGroupConversion Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00DFFFC7	0x00*000** (After Adc_Init Digit * depends on configuration value.) 0x00000000 (After Adc_DeInit.)
SARn_CHx_STRUCT_RANGE_CTL	32:0	Word (32 bits)	0x00000000 | range detect high thresholds << 15 | range detect low thresholds (These items depend on configuration.)	This register sets range detect thresholds.	Adc_Init Adc_DeInit	0xFFFFFFFF	0x******** (After Adc_Init Digit * depends on configuration value.) 0x00000000 (After Adc_DeInit.)
SARn_CHx_STRUCT_INTR	31:0	Word (32 bits)	0x00000707	This register indicates and clears interrupt request.	Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_Init Adc_DeInit Adc_StartGroupConversion Adc_DisableHardwareTrigger Adc_StopGroupConversion Adc_GetStreamLastPointer Adc_ReadGroup	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
					Adc_EnableHardwareTrigger Adc_SelectChannelThreshold Adc_DisableHardwareTrigger Adc_EnableHardwareTrigger Adc_EnableHwTrigger Adc_DisableHwTrigger Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2		
SARn_CHx_STRUCT_INTR_SET	31:0	Word (32 bits)	-	Interrupt set request register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_CHx_STRUCT_INTR_MASK	31:0	Word (32 bits)	0x00000000 | range detect interrupt mask << 8| done interrupt mask	This register enables/disables ADC group interrupt.	Adc_Init Adc_DisableHardwareTrigger Adc_StopGroupConversion Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_StartGroupConversion Adc_StopGroupConversion Adc_EnableHwTrigger Adc_DisableHwTrigger Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
SARn_CHx_STRUCT_INTR_MASKED	31:0	Word (32 bits)	-	Interrupt masked request register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SARn_CHx_STRUCT_WORK	31:0	Word (32 bits)	-	Working data register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_CHx_STRUCT_RESULT	31:0	Word (32 bits)	0x00000000 | conversion result	This register holds the conversion result of the channel.	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_CHx_STRUCT_GRP_STAT	31:0	Word (32 bits)	-	Group status register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SARn_CHx_STRUCT_ENABLE	31:0	Word (32 bits)	0x0000000 | channel enable/disable	This register enables/disables the corresponding channel.	Adc_DisableHardwareTrigger Adc_StopGroupConversion Adc_Init Adc_DeInit Adc_EnableHardwareTrigger Adc_DisableChannel Adc_EnableChannel Adc_StartGroupConversion Adc_StopGroupConversion Adc_GetStreamLastPointer Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SARn_CHx_STRUCT_TR_CMD	31:0	Word (32 bits)	0x00000001	This register starts software trigger.	Adc_StartGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PASS_CTL	31:0	Word (32 bits)	0xF0000000 | Reference mode << 21 | Supply monitor level for AMUXBUS_B << 5 | Supply monitor enable for AMUXBUS_B << 4 | Supply monitor level for AMUXBUS_A << 1 | Supply monitor enable for AMUXBUS_A (These items depend on configuration.)	This register controls supply monitoring function and enables/disables debug pause.	Adc_Init Adc_DeInit	0xF0600033	0xF0*000** (After Adc_Init Digit * depends on configuration value.) 0x00000000 (After Adc_DeInit.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
PASS_SARn_TR_IN_SEL	31:0	Word (32 bits)	0x00000000 | generic trigger 4 << 16 | generic trigger 3 << 12 | generic trigger 2 << 8 | generic trigger 1 << 4 | generic trigger 0 (These items depend on configuration.)	This register selects generic trigger for SAR generic trigger input.	Adc_Init Adc_DeInit	0x000FFFFF	0x000***** (After Adc_Init Digit * depends on configuration value.) 0x00043210 (After Adc_DeInit.)
PASS_SARn_TR_OUT_SEL	31:0	Word (32 bits)	0x00000000 | generic trigger output 1 << 8 | generic trigger output 0 (These items depend on configuration.)	This register selects SAR output trigger for generic trigger output.	Adc_StartGroupConversion Adc_EnableHardwareTrigger Adc_DeInit Adc_StopGroupConversion Adc_EnableHwTrigger Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Note: 'n' is SAR ADC number, and 'x' is logical channel number.

8.2 DW

Table 16 DW access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DWm_CTL	31:0	Word (32 bits)	-	Control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_STATUS	31:0	Word (32 bits)	-	Status	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ACT_DESCR_CTL	31:0	Word (32 bits)	-	Active descriptor control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ACT_DESCR_SRC	31:0	Word (32 bits)	-	Active descriptor source	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ACT_DESCR_DST	31:0	Word (32 bits)	-	Active descriptor destination	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ACT_DESCR_X_CTL	31:0	Word (32 bits)	-	Active descriptor X loop control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ACT_DESCR_Y_CTL	31:0	Word (32 bits)	-	Active descriptor Y loop control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ACT_DESCR_NEXT_PTR	31:0	Word (32 bits)	-	Active descriptor next pointer	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ACT_SRC	31:0	Word (32 bits)	-	Active source	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DWm_ACT_DST	31:0	Word (32 bits)	-	Active destination	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_ECC_CTL	31:0	Word (32 bits)	-	ECC control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CRC_CTL	31:0	Word (32 bits)	-	CRC control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CRC_DATA_CTL	31:0	Word (32 bits)	-	CRC data control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CRC_POL_CTL	31:0	Word (32 bits)	-	CRC polynomial control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CRC_LFSR_CTL	31:0	Word (32 bits)	-	CRC LFSR control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CRC_REM_CTL	31:0	Word (32 bits)	-	CRC remainder control	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CRC_REM_RESULT	31:0	Word (32 bits)	-	CRC remainder result	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DWm_CHi_STRUCT_CH_CTL	31:0	Word (32 bits)	0x80000002 or 0x00000002	This register globally controls DW channel.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_DisableHardwareTrigger Adc_StopGroupConversion Adc_Init Adc_EnableHwTrigger Adc_DisableHwTrigger Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CHi_STRUCT_CH_STATUS	31:0	Word (32 bits)	0x00000000 | source of the interrupt cause	This register specifies the source of the interrupt cause.	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CHi_STRUCT_CH_IDX	31:0	Word (32 bits)	0x00000000	This register specifies the X&Y loop index.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DWm_CHi_STRUCT_CH_CURR_PTR	31:0	Word (32 bits)	0x00000000 | address of current descriptor	This register sets the address of current descriptor.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
DWm_CHi_STRUCT_INTR	31:0	Word (32 bits)	0x00000001	This register indicates whether event is detected and clears interrupt flag.	Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2 Adc_Init Adc_DeInit Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_DisableHardwareTrigger Adc_StopGroupConversion Adc_ReadGroup Adc_GetStreamLastPointer Adc_EnableHwTrigger Adc_DisableHwTrigger Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
DWm_CHi_STRUCT_INTR_SET	31:0	Word (32 bits)	-	Interrupt set	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DWm_CHi_STRUCT_INTR_MASK	31:0	Word (32 bits)	0x00000001 0x00000000	This register enables/disables DW interrupt.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_Init Adc_DeInit Adc_DisableHardwareTrigger Adc_StopGroupConversion Adc_EnableHwTrigger Adc_DisableHwTrigger Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
DWm_CHi_STRUCT_INTR_MASKED	31:0	Word (32 bits)	-	Interrupt masked	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CHi_STRUCT_SRAM_DATA0	31:0	Word (32 bits)	0x00000000	SRAM data 0	Adc_Init Adc_DeInit	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_CHi_STRUCT_SRAM_DATA1	31:0	Word (32 bits)	0x00000000	SRAM data 1	Adc_Init Adc_DeInit	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DWm_DESCR_STRUCT_DESCR_CTL	31:0	Word (32 bits)	Depends on configuration value	This register sets DW descriptors.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DWm_DESCR_STRUCT_DESCR_SRC	31:0	Word (32 bits)	0x00000000 | base address of source location	This register sets the base address of source location.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
DWm_DESCR_STRUCT_DESCR_DST	31:0	Word (32 bits)	0x00000000 | base address of destination location	This register sets the base address of destination location.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
DWm_DESCR_STRUCT_DESCR_X_CTL	31:0	Word (32 bits)	Depends on configuration value	This register controls X loop.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DWm_DESCR_STRUCT_DESCR_Y_CTL	31:0	Word (32 bits)	Depends on configuration value	This register controls Y loop.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
DWm_DESCR_STRUCT_DESCR_NEXT_PTR	31:0	Word (32 bits)	0x00000000 | address of next descriptor in descriptor list	This register sets address of next descriptor in descriptor list.	Adc_EnableHardwareTrigger Adc_StartGroupConversion Adc_EnableHwTrigger Adc_StopGroupConversion Adc_IsrConversionDone_[ADC Irq number]_Cat1 Adc_IsrConversionDone_[ADC Irq number]_Cat2 Adc_DmaDone_[Dma Irq number]_Cat1 Adc_DmaDone_[Dma Irq number]_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Note: 'm' is DW controller number, and 'i' is DW channel number.

Revision history

Document revision	Date	Description of changes
**	2020-09-04	Initial release.
*A	2020-11-20	Changed a memmap file include folder (Memory Mapping). Updated information regarding memory section (Memory Allocation Keyword). MOVED TO INFINEON TEMPLATE.
*B	2021-05-21	Added new configuration parameters (AdcChannelPulseDetect, AdcChannelPulsePositiveCount, and AdcChannelPulseNegativeCount) in section 2.2.1 Architecture Specifics. Added a note in section 4.7 AdcGroup Configuration. Added a note in section 5.14 DMA Transfer. Added a note in section 5.9 Notification. Added SARMUX in Glossary. Added configuration parameters (AdcSarMux1ConnectToAdc0, AdcSarMux2ConnectToAdc0, AdcSarMux3ConnectToAdc0, AdcSarMux1DiagnosticReference, AdcSarMux2DiagnosticReference, AdcSarMux3DiagnosticReference, AdcSarMux1DiagnoseEnable, AdcSarMux2DiagnoseEnable, and AdcSarMux3DiagnoseEnable) in Architecture Specifics section and AdcHwUnit Configuration section. Modified configuration parameters (AdcDiagnoseEnable and AdcDiagnosticReference) in AdcHwUnit Configuration section. Modified configuration parameter (AdcChannelId) in section 4.6 AdcChannel Configuration. Added section 5.16 Port Selection. Added a comment in section 5.18 Diagnostic Feature. Modified a part of the description of SAR ADC in Access Register Table.
*C	2021-08-27	Added new configuration parameters (`AdcDiagnosisMode`, `AdcSelfDiagApi`, `AdcVoltageDeviation`, and `AdcDiagConvertTimeout`) in the "`AdcChannel`" section. Added new configuration parameters (`AdcSelfDiagApi`, `AdcVoltageDeviation`, and `AdcDiagConvertTimeout`) in the "`AdcCustomFunction`" section. Added a new configuration parameter (`AdcDiagnosisMode`) in the "`AdcChannel` configuration" section. Added a new section for the SelfDiag feature. Updated a new section for API parameter checking. Updated a new section for vendor-specific error checking. Updated a new section for reentrancy (added 3 new APIs). Updated a new section for execution-time dependencies (added 3 new APIs). Updated a new section for `Adc_StatusType` (added a status `ADC_DIAG`).

Document revision	Date	Description of changes
		Updated a new section for error codes (added an error code `ADC_E_DIAG`). Updated a new section for API service IDs (added `Adc_StartDiagnosticFull`, `Adc_GetDiagnosticResult`, and `Adc_StartDiagnostic`). Added a new section for `Adc_StartDiagnosticFull`. Added a new section for `Adc_GetDiagnosticResult`. Added a new section for `Adc_StartDiagnostic`. Added a note in the “Interrupts” section.
*D	2021-12-07	Updated to the latest branding guidelines
*E	2023-03-03	Updated the title. Updated the Hardware documentation. Updated analog calibration flow in 5.19 Analog calibration feature. Added a note in 5.27 Sleep mode.
*F	2023-06-06	Added a note in 6.2 Analog input signals. Updated the description in chapter 2.6.1.
*G	2023-10-06	Updated register information in Table 16. Corrected core identification keyword in section 2.6and 5.31.
*H	2023-12-08	Web release. No content updates.
*I	2025-08-12	Added a note under the description of AdcGroupPriority in 4.8 AdcGroup configuration

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-08-12

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-30828 Rev. *I

Important notice
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.