MICROSAR ADC

Technical Reference

MCAL Emulation in VTT

Version 1.1.0

Authors	Christian Leder
Status	Released

Document Information

History

Author	Date	Version	Remarks
Christian Leder	2014-02-25	1.0.0	Initial creation of document
Christian Leder	2015-01-16	1.1.0	> Global renaming of Vip to Vtt > Usage of template 5.9.0 for the Technical reference

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_ADCDriver.pdf	V4.2.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.0.x	Initial version of the Vip ADC driver
2.0.x	Global renaming of Vip to Vtt

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module ADC as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	ADC_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	ADC_MODULE_ID	123 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The ADC module initializes and controls the internal Analog Digital Converter Unit(s) of the microcontroller. It provides services to start and stop a conversion respectively to enable and disable the trigger source for a conversion. Furthermore it provides services to enable and disable a notification mechanism and routines to query the status and result of a conversion. In this driver, the microcontroller's behavior is emulated by Vector's VTT framework.

The ADC module works on so called ADC channel groups. An ADC channel group contains one or more ADC channels. A conversion result for each of the channels can be set in the respective system variable of CANoe.

2.1 Architecture Overview

The following figure shows where the ADC is located in the AUTOSAR architecture.

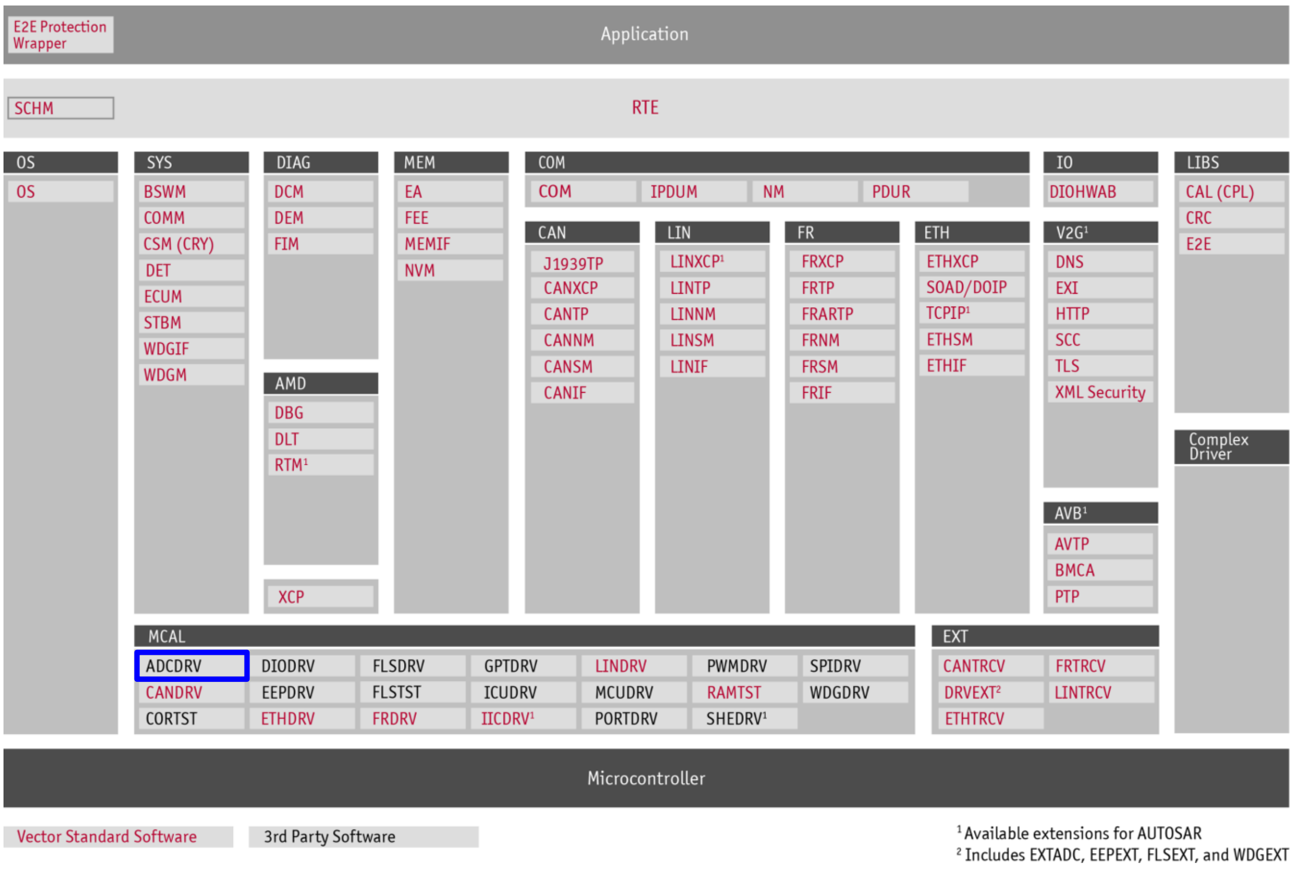


Figure 2-1 AUTOSAR 4.x Architecture Overview

Note
The Microcontroller in Figure 2-1 is emulated by the VTT framework.

The next figure shows the interfaces to adjacent modules of the ADC. These interfaces are described in chapter 4.3.1.

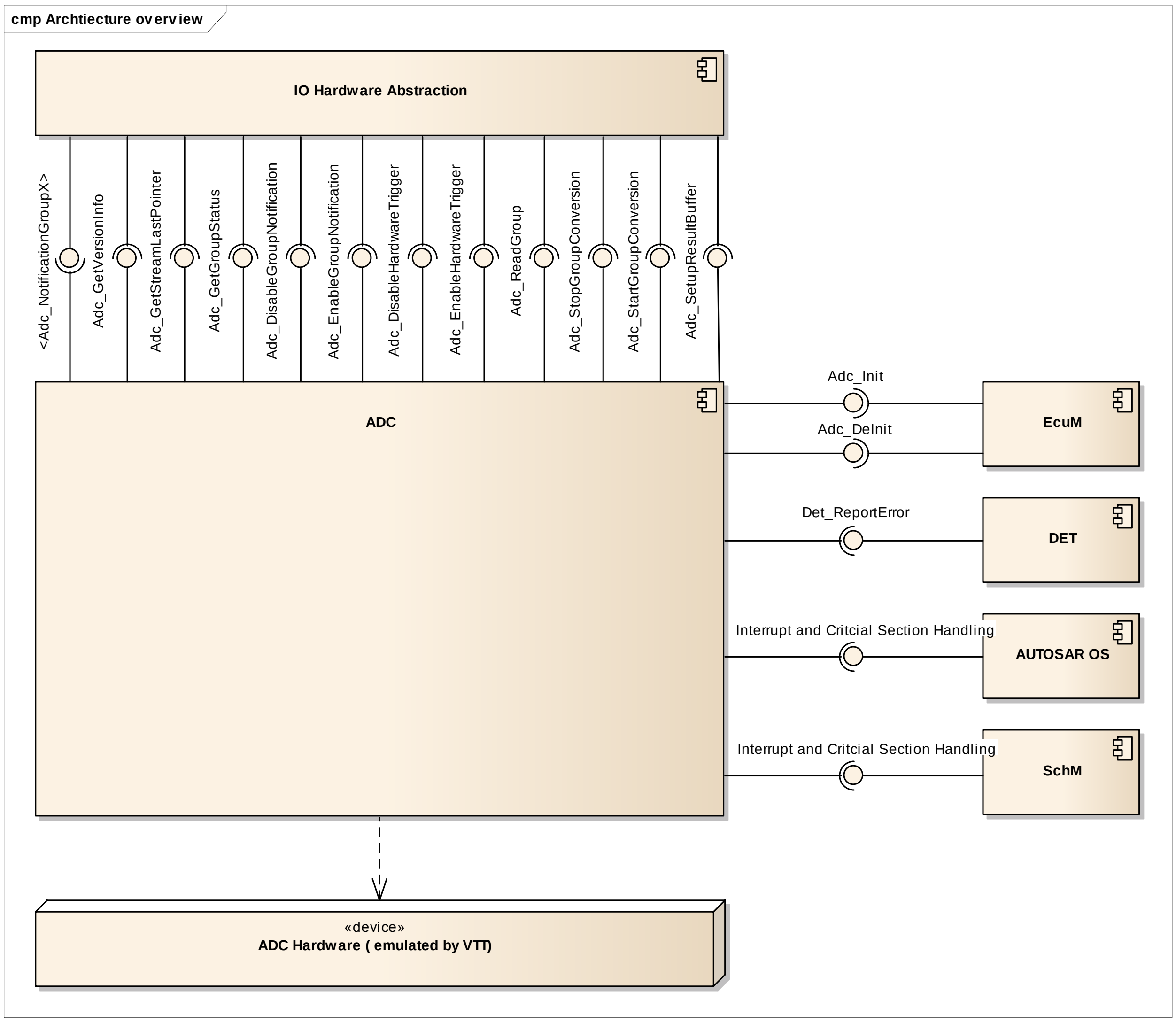


Figure 2-2 Interfaces to adjacent modules of the ADC

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the ADC.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further ADC functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Initialization and deinitialization of internal analog-to-digital conversion units
Grouping of ADC channels to so called ADC channel groups
Starting and stopping conversions of software triggered channel groups
Enabling and disabling hardware trigger of hardware triggered channel groups
Getting information about the status of a group
Reading converted analogue values

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
ADC does not detect hardware errors and therefore no DEM will be used.

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Additional parameter checks for the parameters `DataBufferPtr`, `PtrToSamplePtr` and `versionInfo`

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.2.1 Parameter Checking

For more details see Table 3-3 and chapter 3.15.1

3.1.3 Limitations

3.1.3.1 Diagnostic Event Manager

No DEM will be used, because this emulation does not support the detection of hardware errors.

3.1.3.2 Priority Implementation

Prioritization is not implemented due to not emulated hardware prioritization features.

3.1.3.3 Queuing

Conversion requests cannot be queued in a software queue. This feature is not implemented.

3.1.3.4 Limit Check

The limit checking feature according to AUTOSAR is not supported by this driver.

3.1.3.5 Result Alignment

The ADC conversion alignment is currently not supported. All conversion values are right - aligned all the time.

3.2 Emulation

As described in chapter 2.1, the behavior of the microcontroller's ADC peripheral is emulated by the VTT framework. Parts of the emulation are:

> Emulation of conversions (including selectable conversion result values and selectable conversion and sample time)

> Emulation of hardware triggers (group can be triggered by setting system variables in CANoe)

> Five independent hardware units are emulated (one group per hardware unit can be converted simultaneously).

3.3 Initialization

The ADC driver is initialized by calling `Adc_Init(&AdcConfigSet)`, where `AdcConfigSet` is the name of the runtime configuration set.

After deinitializing the module by calling `Adc_DeInit()`, it can be initialized again.

3.4 States

Each ADC channel group has up to four different states, dependent on its configuration:

> `ADC_IDLE`: Group is idling; no conversion is ongoing.

> `ADC_BUSY:` Group is actively converting or is waiting for a hardware trigger event. Buffer is empty or contains no complete sample (a set of results, see info box below).

> `ADC_COMPLETED`: Group is actively converting with at least one valid sample in buffer. This status is only applied on groups that are configured with streaming access mode.

> `ADC_STREAM_COMPLETED`: Buffer is filled completely. Groups that stop implicitly (see chapter 3.10) are not actively converting in this state anymore. Groups that have to be stopped by the user (explicit) are still converting (buffer wrap around).

Note
A sample describes a set of results which contains one result for each channel of an ADC channel group.

The following graphics out of [1] show state transitions for different function calls and events.

3.4.1 One-shot, SW-triggered, single access

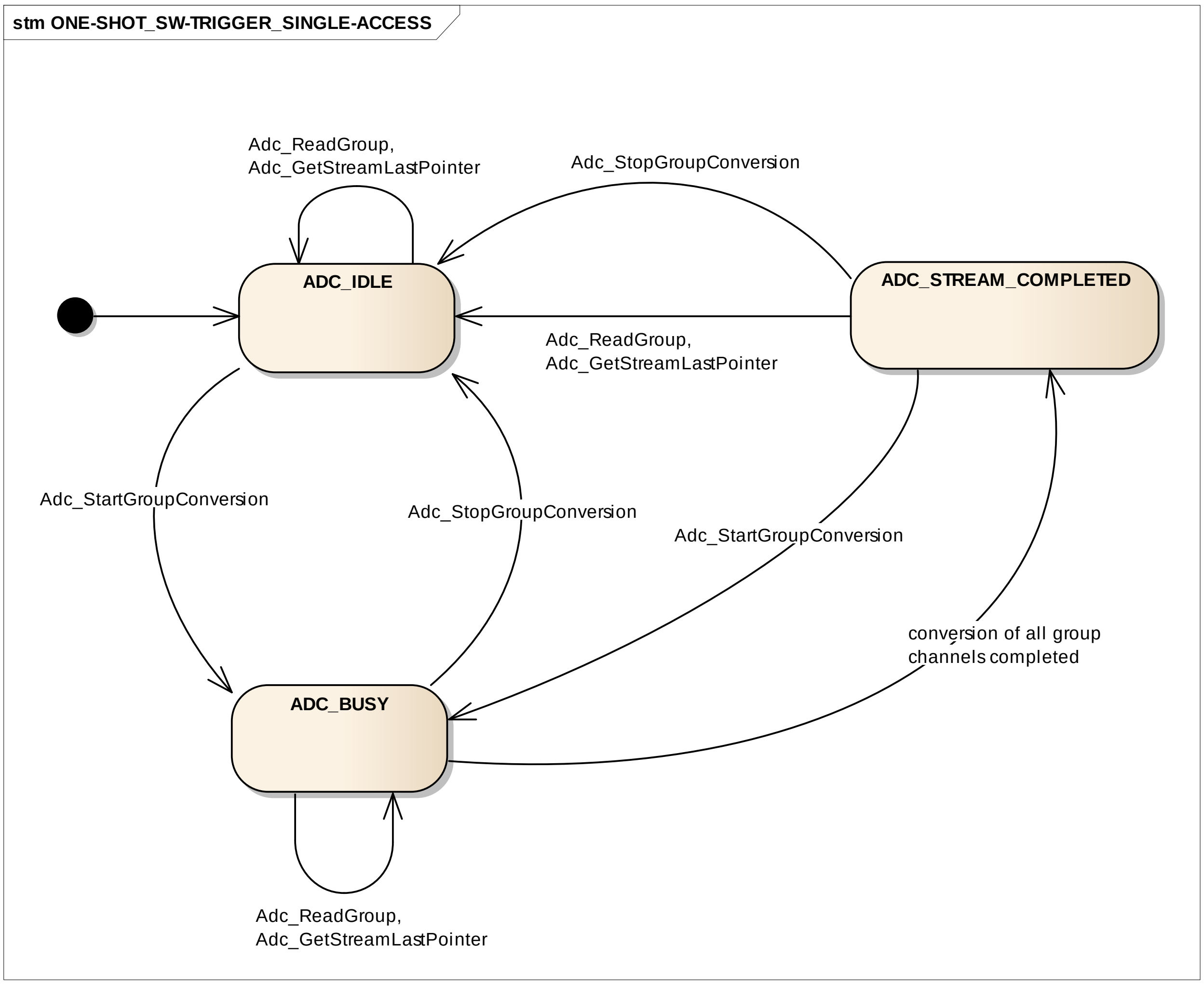


Figure 3-1 State transitions for a one-shot, SW-triggered, single access group

3.4.2 One-shot, HW-triggered, single access group

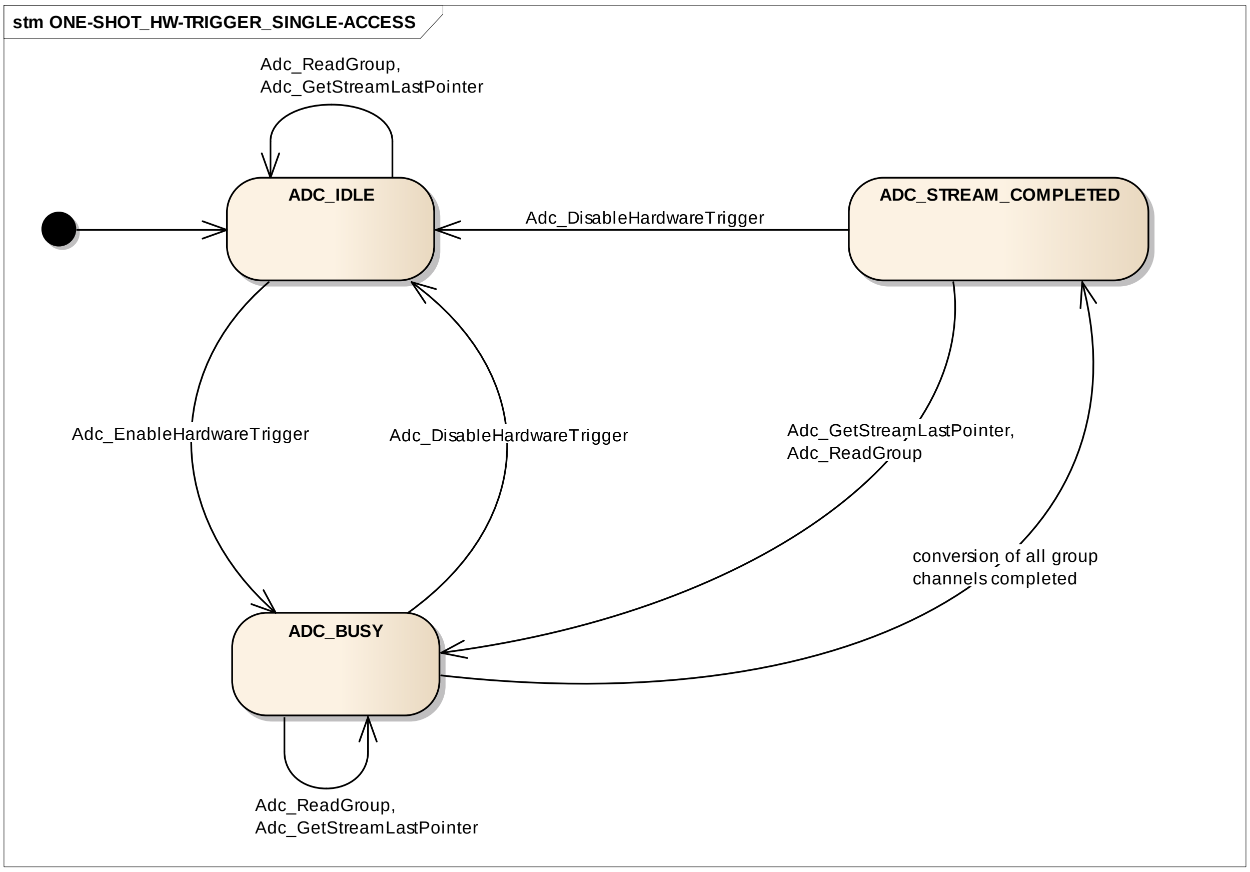


Figure 3-2 State transitions for a one-shot, HW-triggered, single access group

Caution
There is one big difference between the configurations shown in Figure 3-1 and Figure 3-2 above. Figure 3-1 shows a state machine for a group that **stops implicitly** when status `ADC_STREAM_COMPLETED` is reached. Figure 3-2 shows a group that **has to be stopped by the user (explicitly).** This kind of group is still active, even if the group status is `ADC_STREAM_COMPLETED`. For more information on group stop behavior see chapter 3.10.

3.4.3 One-shot, HW-triggered, linear streaming access group

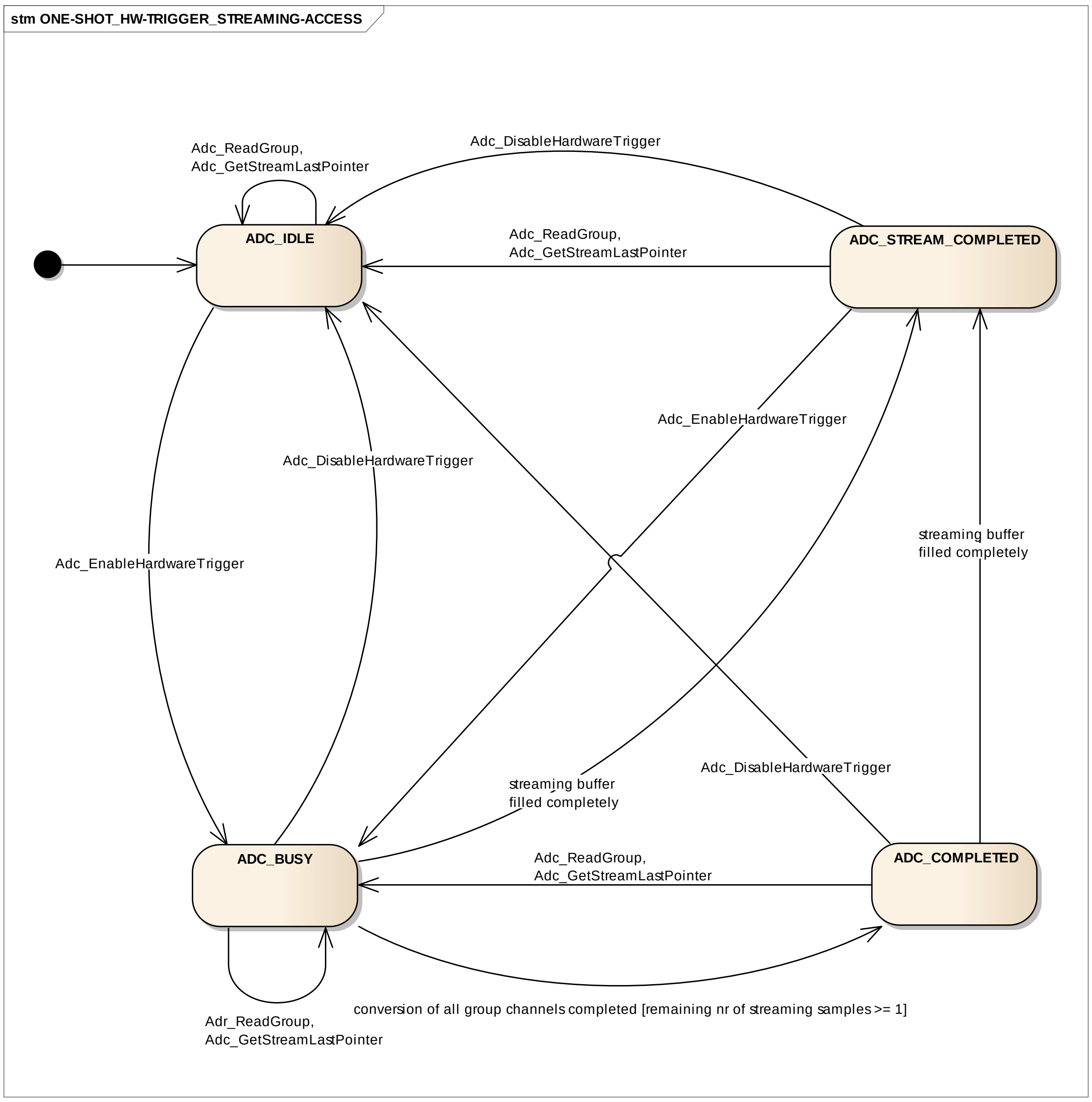

stm ONE-SHOT_HW-TRIGGER_STREAMING-ACCESS
ADC_IDLE
ADC_STREAM_COMPLETED
ADC_BUSY
ADC_COMPLETED
Adc_ReadGroup, Adc_GetStreamLastPointer
Adc_DisableHardwareTrigger
Adc_ReadGroup, Adc_GetStreamLastPointer
Adc_EnableHardwareTrigger
Adc_DisableHardwareTrigger
Adc_EnableHardwareTrigger
streaming buffer filled completely
Adc_DisableHardwareTrigger
streaming buffer filled completely
Adc_ReadGroup, Adc_GetStreamLastPointer
conversion of all group channels completed [remaining nr of streaming samples >= 1]
Adr_ReadGroup, Adc_GetStreamLastPointer

Figure 3-3 State transitions for a one-shot, HW-triggered, linear streaming access group

3.4.4 One-shot, HW-triggered, circular streaming access group

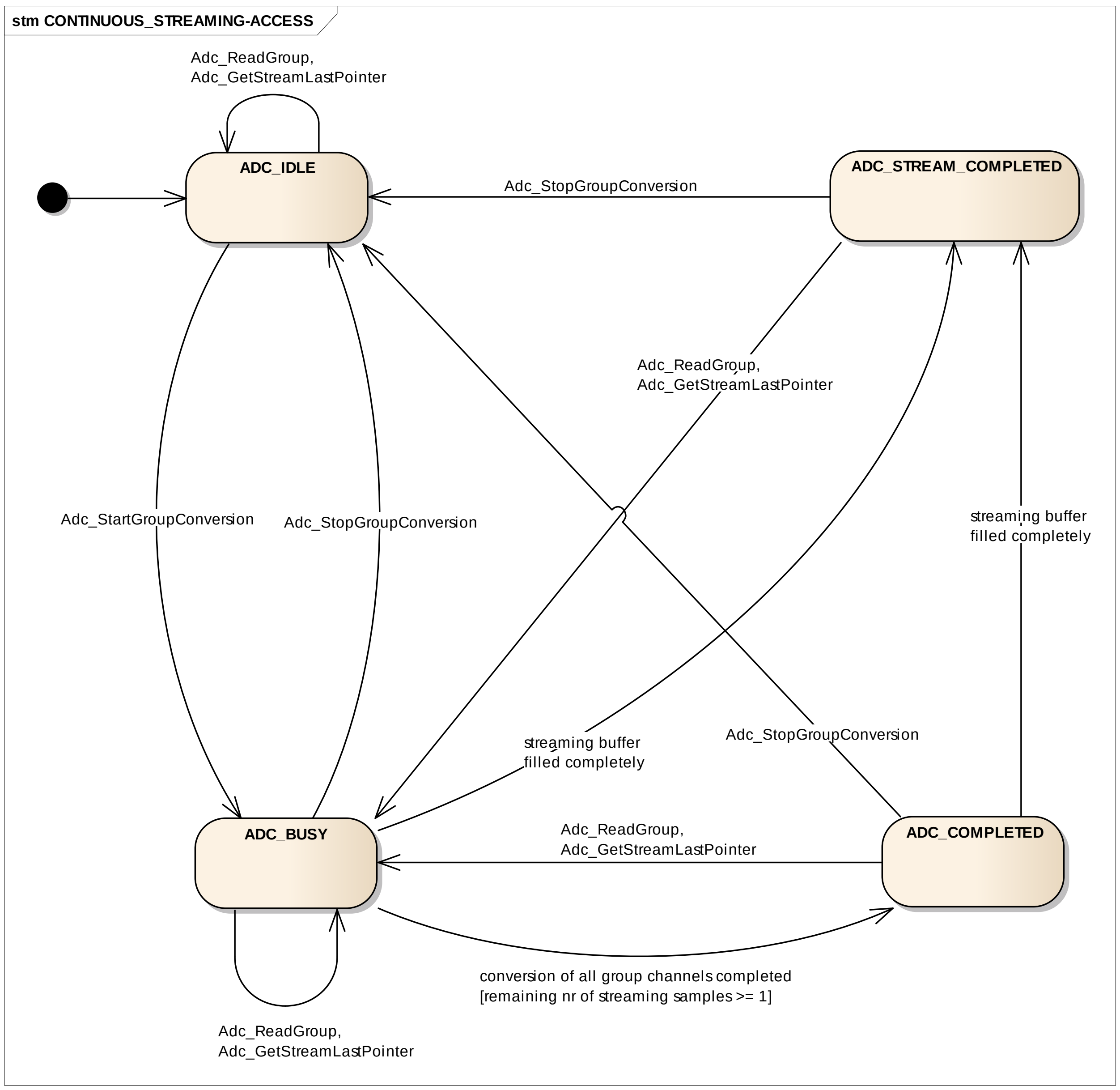


Figure 3-4 State transitions for a one-shot, HW-triggered, circular streaming access group

3.4.5 Continuous, SW-triggered, single access group

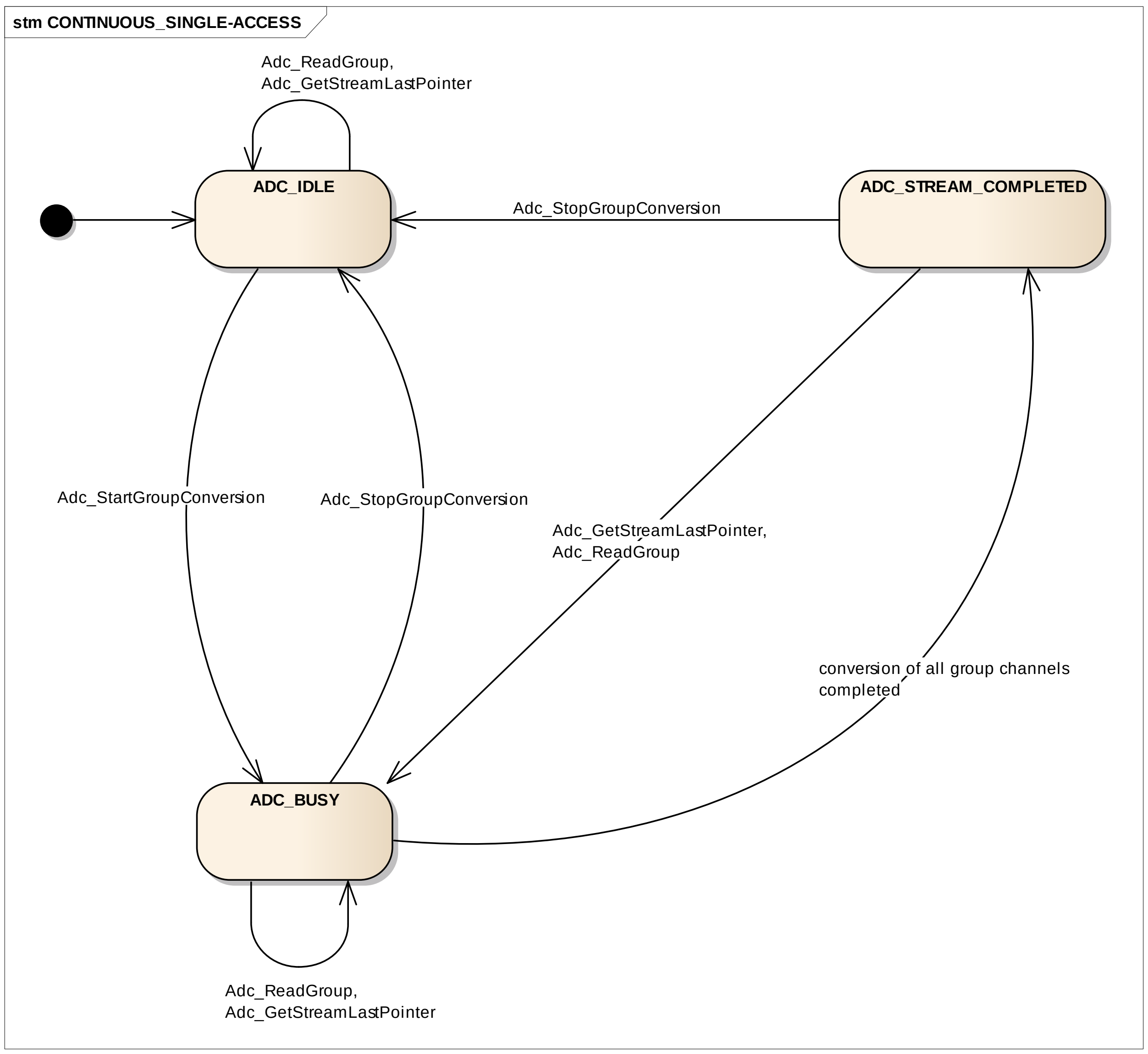


Figure 3-5 State transitions for a continuous, SW-triggered, single access group

3.4.6 Continuous, SW-triggered, linear streaming access group

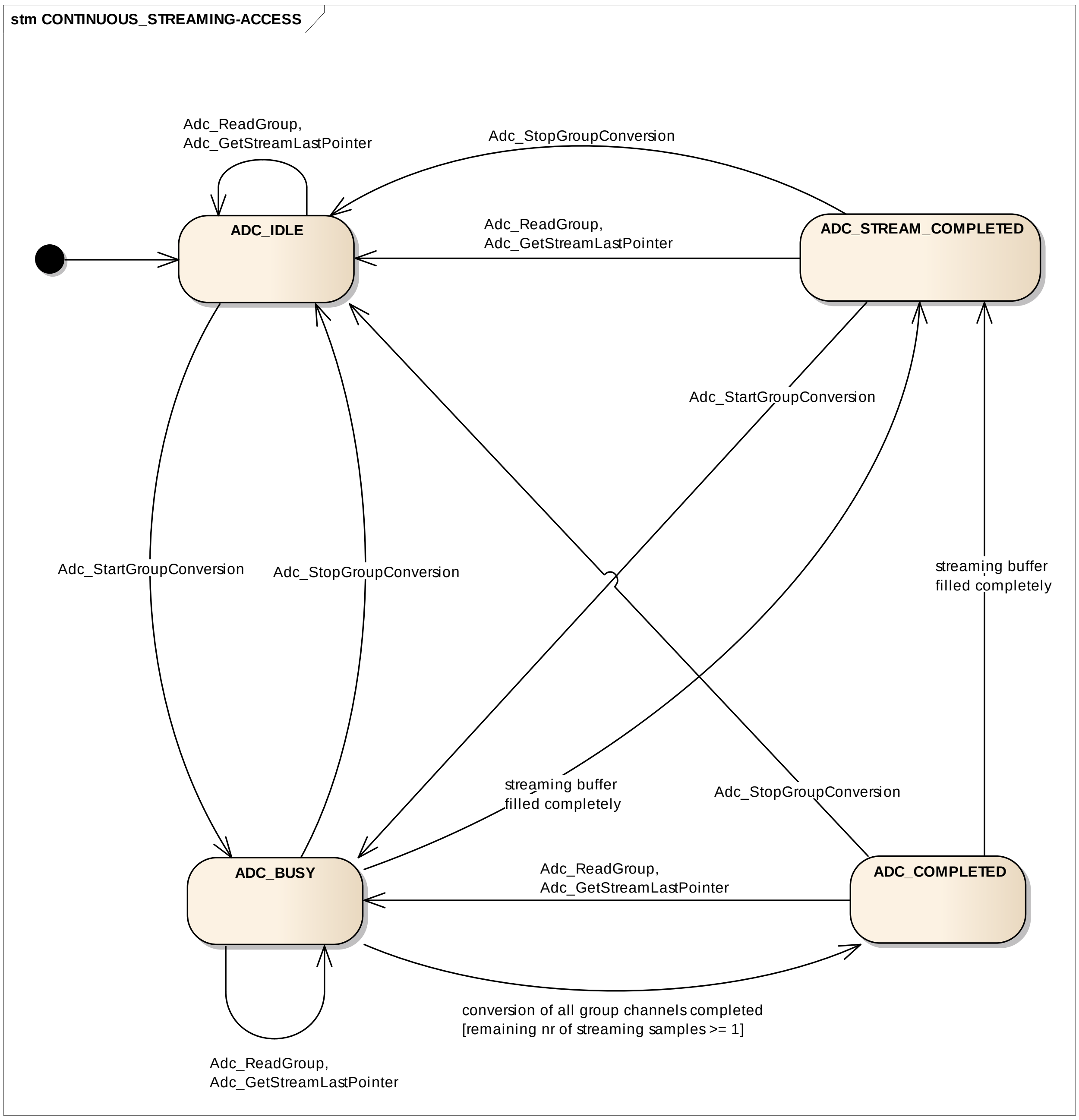


Figure 3-6 State transitions for a continuous, SW-triggered, linear streaming access group

3.4.7 Continuous, SW-triggered, circular streaming access group

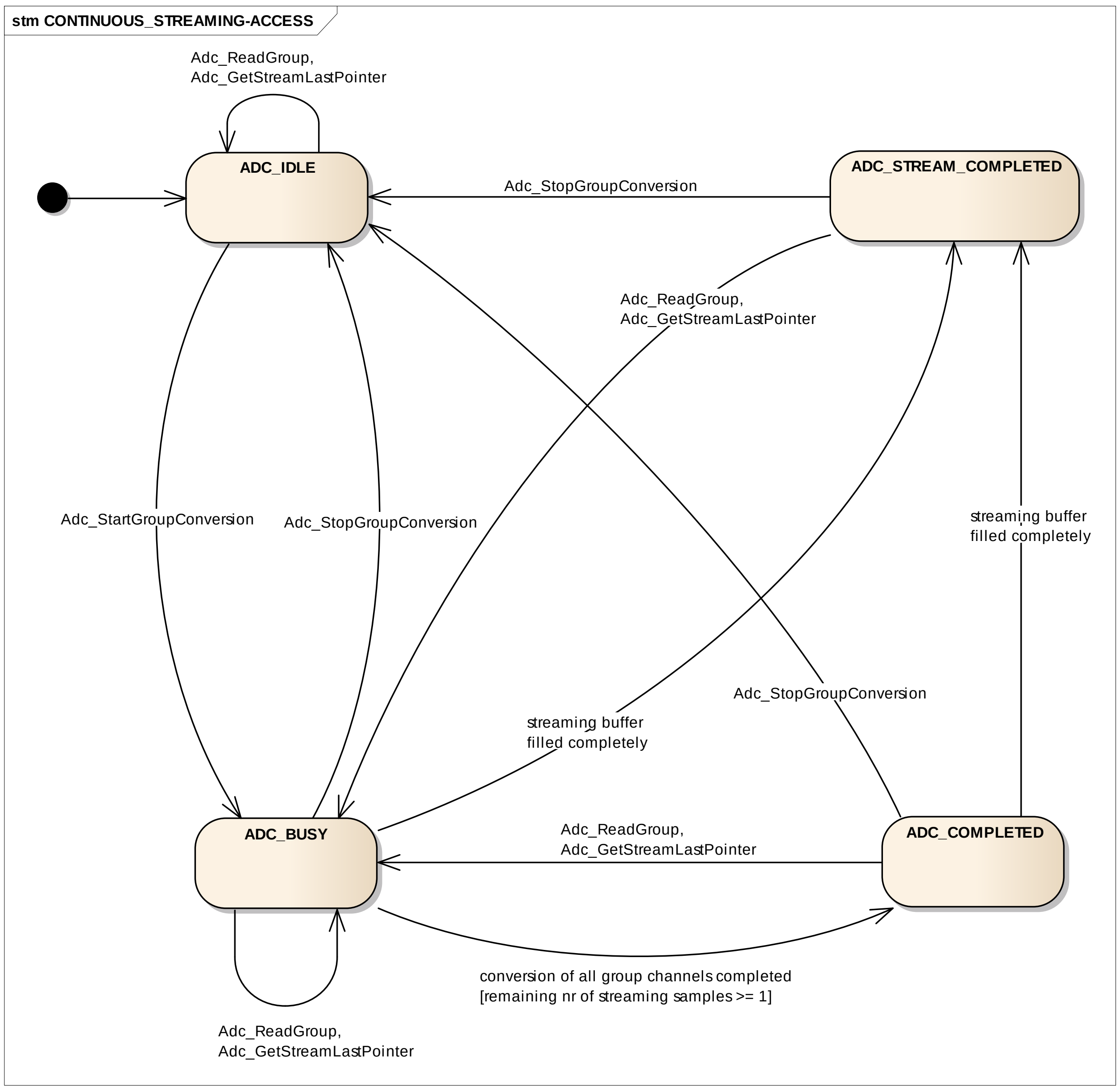


Figure 3-7 State transitions for a continuous, SW-triggered, circular streaming access group

3.5 Group Conversion Modes

The ADC differs between two different conversion modes. These are

> One-shot conversion and

> Continuous conversion

The difference is that a group which is configured in one-shot conversion mode converts each channel once, when the group is activated (triggered by hardware or started with an API call) and a group which is configured in continuous conversion mode executes the conversion several times.

Caution
Continuous conversion is only available for software triggered groups.

3.6 Group Trigger Sources

An ADC conversion can be initiated by two different events:

> A software API call (software triggered) and

> By hardware trigger (a system variable for each hardware triggered group is set up in the Vectors VTT framework)

Note
Each hardware triggered group provides a system variable in CANoe which is responsible for triggering this group. The name of the variable is the group's short name <AdcGroup>

Caution
HW trigger functionality is only available for groups configured in one-shot conversion mode.

3.7 Group Access Modes

The ADC differs between two, so called Group Access Modes. These are

> Single Access and

> Streaming Access

These modes describe how the application result buffer is written and how many samples (a set of all channels belonging to this ADC channel group) can be stored in the buffer.

In Single Access Mode, only one sample is stored in application result buffer. The user has to provide a buffer, which contains as many elements as channels belonging to the corresponding ADC channel group. This buffer gets overwritten each time a conversion is started. The results are stored in that order as configured with DaVinci Configurator Pro.

In Streaming Access Mode, several samples of an ADC channel group can be stored in the application result buffer. The number of samples can be configured in DaVinci Configurator Pro. Thus, the buffer has to provide m*n elements, where m is the number of channels belonging to this ADC channel group and n is the number of samples. The results are stored in the following order: the first n elements contain the results of the first channel; the second n elements contain the results of the second channel; and so on. These sets are stored in that order as configured with DaVinci Configurator Pro.

Caution
Streaming Access is not available in combination with one-shot, software triggered conversions.

3.8 Combinations of Operation Modes

The following combinations of Group Conversion Mode, Group Trigger Source and Group Access Mode are possible, according to [1].

Group Conversion Mode	Group Trigger Source	Group Access Mode
One-shot conversion		
One-shot conversion	Software triggered	Single Access
One-shot conversion	Hardware triggered	Single Access
One-shot conversion	Hardware triggered	Streaming Access
Continuous conversion		
Continuous	Software triggered	Single Access
Continuous	Software triggered	Streaming Access

Table 3-4 Combinations of Operation Modes

3.9 Streaming Buffer Modes

There are two different modes for a group that is configured with Streaming Access buffer:

> Linear Streaming Buffer and

> Circular Streaming Buffer

The difference between these modes is that a group that is configured with linear streaming buffer access stops converting automatically (implicitly), when the buffer is filled completely. A group that is configured with circular streaming buffer access does not stop when the buffer is filled completely; it just performs a so called wrap around and starts overwriting the buffer, beginning at the first element.

3.10 Conversion Stop Behavior

The ADC differs between two kinds of conversions:

> Conversions that stop implicitly and

> Conversions that have to be stopped by the user (explicitly)

Some groups (see table below) do stop implicitly (automatically) after a defined number of conversions, others have to be stopped by the user.

Groups that have to be stopped explicitly are cyclically overwriting their application result buffers when they are completely filled (`ADC_STREAM_COMPLETED`). In this case the user has to call either `Adc_StopGroupConversion` or `Adc_DisableHardwareTrigger`.

Example

Group1 (configured as one-shot, software-triggered, single access group):

Enable group-end notification by calling
`Adc_EnableGroupNotification(Group1)`

Call `Adc_StartGroupConversion(Group1)` to start the group. If the configured group-end notification successfully occurred, any other group can be started on the same HW unit, because the group stopped **implicitly** after the one-shot conversion.

Group2 (configured as continuous, software-triggered, single access group):

Enable group-end notification by calling
`Adc_EnableGroupNotification(Group2)`.

Call `Adc_StartGroupConversion(Group2)` to start the group.

The group is converting continuously now and notifications are sent after each conversion. It has to be stopped by the user (**explicitly**) before any other group can be started on the same HW unit.

Call `Adc_StopGroupConversion(Group2)` to stop the group again.

Group Conv. Mode	Group Trigger Source	Group Access Mode	Streaming Buffer Mode	Group Stop Behavior
Explicit stop required				
One-shot conversion	Hardware triggered	Single Access	---	explicit
One-shot conversion	Hardware triggered	Streaming Access	Circular	explicit
Continuous conv.	Software triggered	Streaming Access	Circular	explicit
Continuous conv.	Software triggered	Single Access	---	implicit
Stops implicitly				
One-shot conversion	Software triggered	Single Access	---	implicit
One-shot conversion	Hardware triggered	Streaming Access	Linear	implicit
Continuous conv.	Software triggered	Streaming Access	Linear	implicit

Table 3-5 Group stop behavior

Caution
The Conversion Stop Behavior has effect on state changes, for example when reading group results. Always consider if a group stops implicitly or has to be stopped explicitly.

3.11 Buffer setup and buffer access

The following graphics out of [1] show step by step:

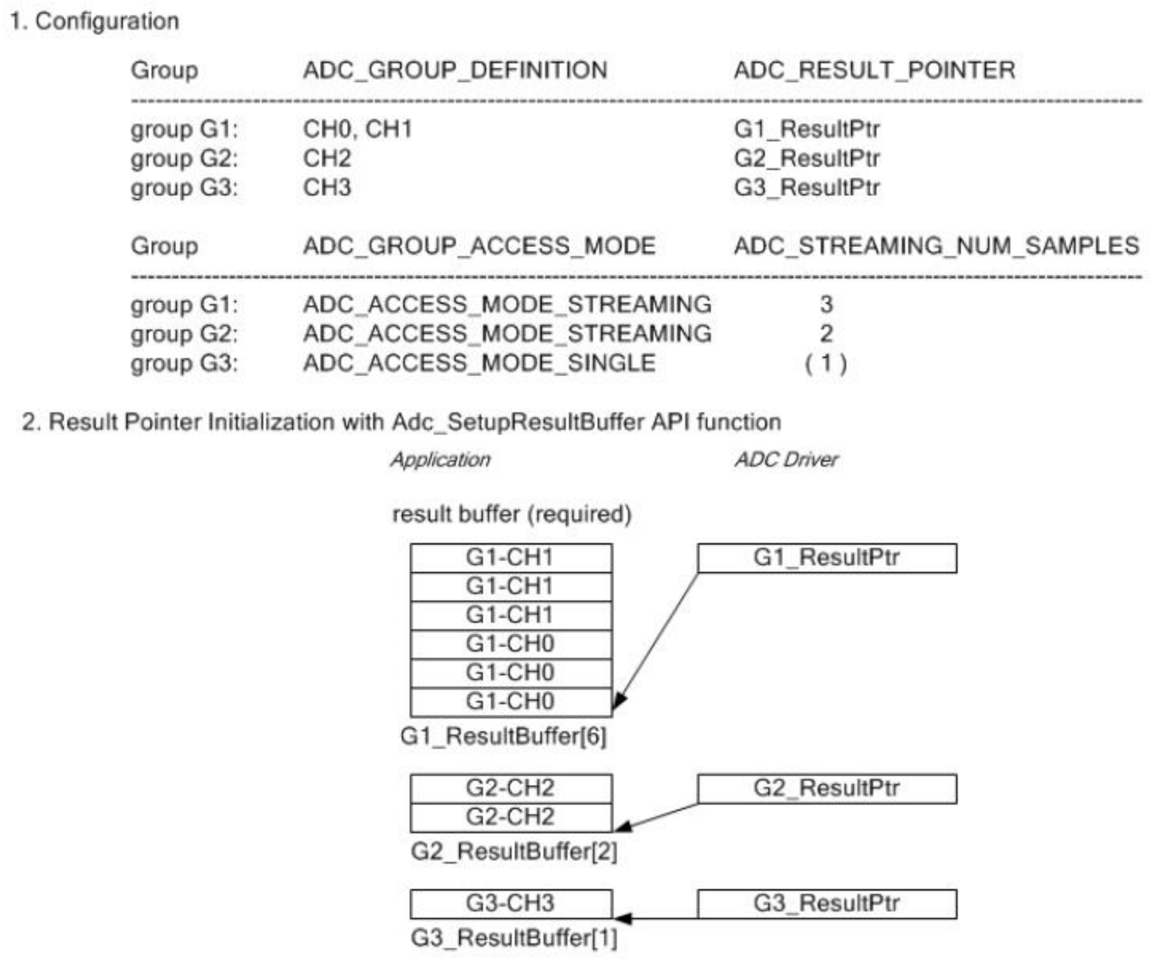
> a sample configuration for three groups (step 1),

> how the corresponding result buffers are initialized (step 2),

> result access by `Adc_GetStreamLastPointer` (step 3) and

> result access by `Adc_ReadGroup` (step 4)

Caution
The user has to provide an application result buffer for each of the ADC channel groups.
In case, the function `Adc_ReadGroup` is used for buffer access, the user has to provide an additional buffer.
Refer to the graphics below for an example. Have a look at the headlines (Application, ADC Driver and ADC HW) in the graphics to see the responsibilities for the single components.



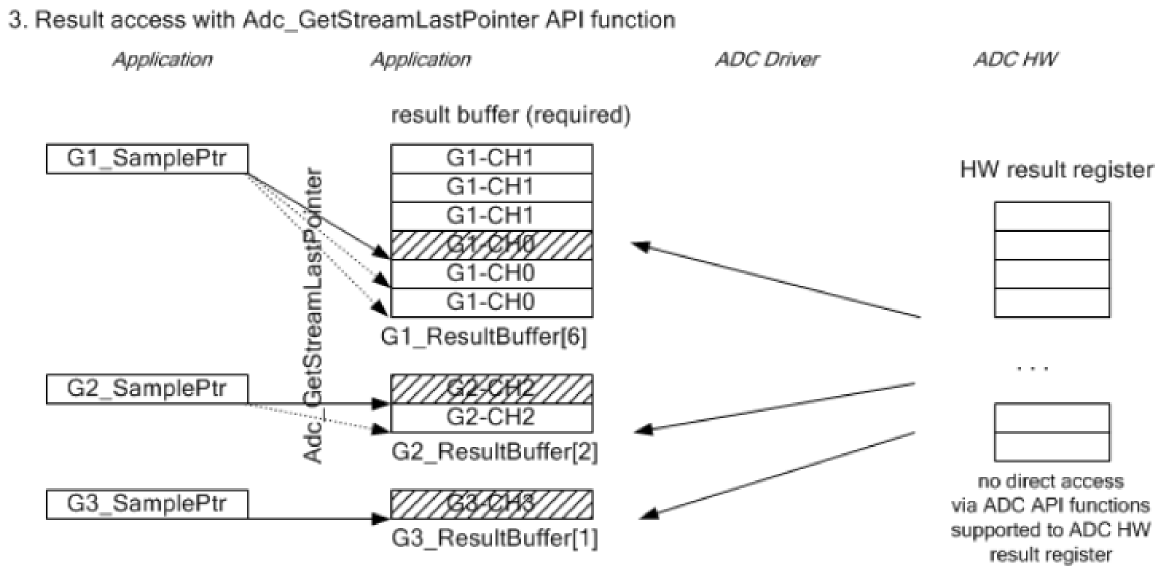
1. Configuration

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

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

2. Result Pointer Initialization with Adc_SetupResultBuffer API function

Figure 3-8 Buffer setup



3. Result access with Adc_GetStreamLastPointer API function

Figure 3-9 Buffer access

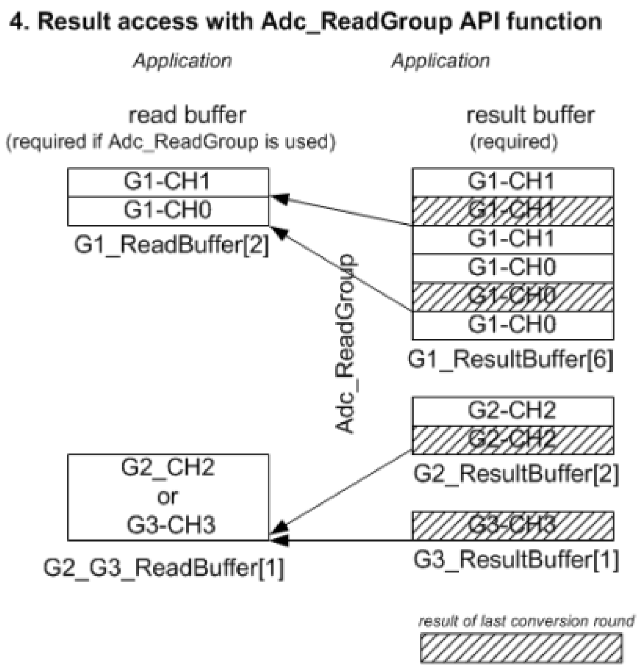


Figure 3-10 Access results using Adc_ReadGroup

Detailed explanation of the example above:

> Step 1: Group configuration

This example consists of three groups: `G1`, `G2` and `G3`. `G1` contains two channels, `G2` and `G3` one channel each. `G1` is configured for streaming access mode with three samples, `G2` for two samples and `G3` is configured for single access mode, which means, that only one sample is stored in application result buffer.

> Step 2: Setting up result buffers

The user has to provide one application result buffer per ADC channel group. The buffer has to have capacity for m*n elements, where m is the number of samples and n the number of channels belonging to that group. For allocating memory, e.g. call `Adc_SetupResultBuffer(G1, G1_ResultBuffer)` to assign the address of `G1_ResultBuffer` to group `G1_ResultPtr` (the module internal result buffer pointer). Do the analogue for `G2` and `G3`.

> Step 3: Result access by `Adc_GetStreamLastPointer`

For accessing results by `Adc_GetStreamLastPointer`, the user has to provide a pointer on `Adc_ValueGroupType`. In this example it is named `Gx_SamplePtr`. After calling `Adc_GetStreamLastPointer(G1, &G1_SamplePtr)` `G1_SamplePtr` references the latest entry of the first channel of a group. The function returns the number of valid samples stored in application result buffer. With `Gx_SamplePtr` and the return value, all results in buffer can be accessed (if the user takes the result buffer layout into account).

Caution
The ADC performs a wraparound when writing a circular buffer. The application has to take this into account when reading out results.

> Step 4: Result access by `Adc_ReadGroup`

Another way of getting conversion results is accessing group by `Adc_ReadGroup`. This function copies the latest sample (set of results) to application read buffer, which has to be allocated by the user. Due to the fact, that only one result per channel is copied, the size of the application read buffer has to be equal to the number of channels configured for that group. Call for example `Adc_ReadGroup(G2, G2_G3_ReadBuffer)` to store the latest result of group `G2` to the application read buffer `G2_G3_ReadBuffer`, which in this example is used for `G2` as well as for `G3`.

3.12 Simultaneous Operations

Only one group can be converted on a hardware unit at the same time. Another group can only be started if all groups are in state `ADC_IDLE` (or in state `ADC_STREAM_COMPLETED` in case group stops implicitly).

Nevertheless it is possible to simultaneously start two groups that are configured on different hardware units (if available).

3.13 Timings

This driver uses timers emulated by the VTT framework for simulation of group conversion and sampling time and for edge detection (trigger variables in CANoe). Therefore the user has to ensure, that the application does not block timer handling.

The following sample application **may not work**:

```
Adc_Init(&AdcConfigSet);
Adc_SetupResultBuffer(AdcGroup, &Appl_ResultBuffer);
Adc_StartGroupConversion(AdcGroup);
while(1)
{
  Appl_Status = Adc_GetGroupStatus(AdcGroup);

  if(ADC_STREAM_COMPLETED == Appl_Status)
  {
    break;
  }
}
```

The problem is, that the timer, that is started in `Adc_StartGroupConversion`, expires but cannot be processed, because the program stays in the while-loop forever. Timer processing is done in the OS' IDLE loop or when tasks are reschuled. Therefore the user has to ensure, that there is either a task rescheduling or the OS enters the IDLE loop, while the group is converting. This has to be assured for each conversion sample. If for example a group with streaming access has five samples, the timer is started five times.

Correction of the example above:

```
Adc_Init(&AdcConfigSet);
Adc_SetupResultBuffer(AdcGroup, &Appl_ResultBuffer);
```

```
  Adc_StartGroupConversion(AdcGroup);
  while(1)
  {
    Appl_Status = Adc_GetGroupStatus(AdcGroup);

    if(ADC_STREAM_COMPLETED == Appl_Status)
    {
      break;
    }

    /* Call scheduler to process timer */
    Schedule();
  }
```

The same problem occurs when **waiting for a hardware trigger** that is set by toggling the respective system variable in CANoe. The ADC driver has a built-in timer for hardware trigger supervision. Every time the timer expires, the states of the trigger variables are compared to their last states. If a state has changed in the meantime, an edge is successfully detected. To ensure, that the trigger supervision works properly, a call of `Schedule()` has to occur frequently (see example above).

3.14 Main Functions

Module ADC does not provide any cyclic main functions.

3.15 Error Handling

3.15.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `ADC_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported ADC ID is 123.

The reported service IDs identify the services which are described in 5.3. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Adc_Init
0x01	Adc_DeInit
0x02	Adc_StartGroupConversion
0x03	Adc_StopGroupConversion
0x04	Adc_ReadGroup
0x05	Adc_EnableHardwareTrigger
0x06	Adc_DisableHardwareTrigger
0x07	Adc_EnableGroupNotification
0x08	Adc_DisableGroupNotification
0x09	Adc_GetGroupStatus
0x0A	Adc_GetVersionInfo

Service ID	Service
0x0B	Adc_GetStreamLastPointer
0x0C	Adc_SetupResultBuffer

Table 3-6 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
0x0A	ADC_E_UNINIT	Adc_Init has not been called prior to another function call.
0x0B	ADC_E_BUSY	Adc_StartGroupConversion was called while another conversion is already running on the same HW unit or a HW trigger is already enabled. Adc_EnableHardwareTrigger was called while a conversion is ongoing on the same HW unit or a HW trigger is already enabled. Adc_DeInit was called while a conversion is still ongoing.
0x0C	ADC_E_IDLE	Adc_StopGroupConversion was called while no conversion was running. Adc_DisableHardwareTrigger was called while group is not enabled.
0x0D	ADC_E_ALREADY_INITIALIZED	Adc_Init has been called while ADC is already initialized.
0x0E	ADC_E_PARAM_CONFIG	Adc_Init has been called with configuration pointer (parameter ConfigPtr) referencing NULL_PTR.
0x15	ADC_E_PARAM_GROUP	Invalid group ID requested.
0x16	ADC_E_WRONG_CONV_MODE	Adc_EnableHardwareTrigger or Adc_DisableHardwareTrigger called on a group with conversion mode configured as continuous.
0x17	ADC_E_WRONG_TRIGG_SRC	Adc_StartGroupConversion or Adc_StopGroupConversion called on a group with trigger source configured as hardware. Adc_EnableHardwareTrigger or Adc_DisableHardwareTrigger called on a group with trigger source configured as software API.
0x18	ADC_E_NOTIF_CAPABILITY	Adc_EnableNotification or Adc_DisableNotification called for a group whose configuration has no notification available.
0x19	ADC_E_BUFFER_UNINIT	Conversion started but result buffer pointer has not been initialized before.
0x1A	ADC_E_PARAM_READ_BUFFER	This error is raised when Adc_ReadGroup is called with parameter DataBufferPtr referencing NULL_PTR. This error is additional to AUTOSAR specification.

Error Code		Description
0x1B	ADC_E_PARAM_RESULT_BUFFER	This error is raised when Adc_SetupResultBuffer is called with parameter DataBufferPtr referencing NULL_PTR. This error is additional to AUTOSAR specification.
0x1C	ADC_E_PARAM_STREAM_PTR	This error is raised when Adc_GetStreamLastPointer is called with parameter PtrToSamplePtr referencing NULL_PTR. This error is additional to AUTOSAR specification.
0x1D	ADC_E_PARAM_VINFO	This error is raised when Adc_GetVersionInfo is called with parameter versioninfo referencing NULL_PTR. This error is additional to AUTOSAR specification.

Table 3-7 Errors reported to DET

3.15.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-8 are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled.

The following table shows which parameter checks are performed on which services:

Check / Service	ADC_E_UNIT	ADC_E_BUSY	ADC_E_IDLE	ADC_E_ALREADY_INITIALIZED	ADC_E_PARAM_CONFIG	ADC_E_PARAM_GROUP	ADC_E_WRONG_CONV_MODE	ADC_E_WRONG_TRIGG_SRC	ADC_E_NOTIF_CAPABILITY	ADC_E_BUFFER_UNINIT	ADC_E_PARAM_READ_BUFFER	ADC_E_PARAM_RESULT_BUFFER	ADC_E_PARAM_STREAM_PTR	ADC_E_PARAM_VINFO
Adc_Init				■	■									
Adc_SetupResultBuffer	■	■				■						■		
Adc_DeInit	■	■												
Adc_StartGroupConversion	■	■				■		■		■				
Adc_StopGroupConversion	■		■			■		■						
Adc_ReadGroup	■		■			■					■			
Adc_EnableHardwareTrigger	■	■				■		■		■				
Adc_DisableHardwareTrigger	■		■			■		■						
Adc_EnableGroupNotification	■					■			■					
Adc_DisableGroupNotification	■					■			■					
Adc_GetGroupStatus	■					■								
Adc_GetStreamLastPointer	■		■			■							■	
Adc_GetVersionInfo														■

Table 3-8 Development Error Checking: Assignment of checks to services

4 Integration

This chapter gives necessary information for the integration of the MICROSAR ADC into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the ADC contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
Adc.h	The module header defines the interface of the ADC. This file must be included by upper layer software components
Adc.c	This C-source contains the implementation of the module's functionalities
Adc_Irq.h	The Adc_Irq header defines the interfaces for the emulated hardware
Adc_Irq.c	This C-Source contains the implementation of the interfaces for the emulated hardware
DrvAdc_VttCanoe01Asr.jar	This jar-file contains the generator and the validator for the DaVinci Configurator Pro
VTTAdc_bswmd.arxml	Basic Software Module Description according to AUTOSAR for VTT Emulation
Adc_bswmd.arxml	Optional Basic Software Module Description. Placeholder for real target (semiconductor manufacturer) in VTT only use case

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
Adc_Cfg.h	The configuration-header contains the static configuration part of this module
Adc_PBcfg.c	The configuration-source contains the object independent part of the runtime configuration

Table 4-2 Generated files

4.2 Include Structure

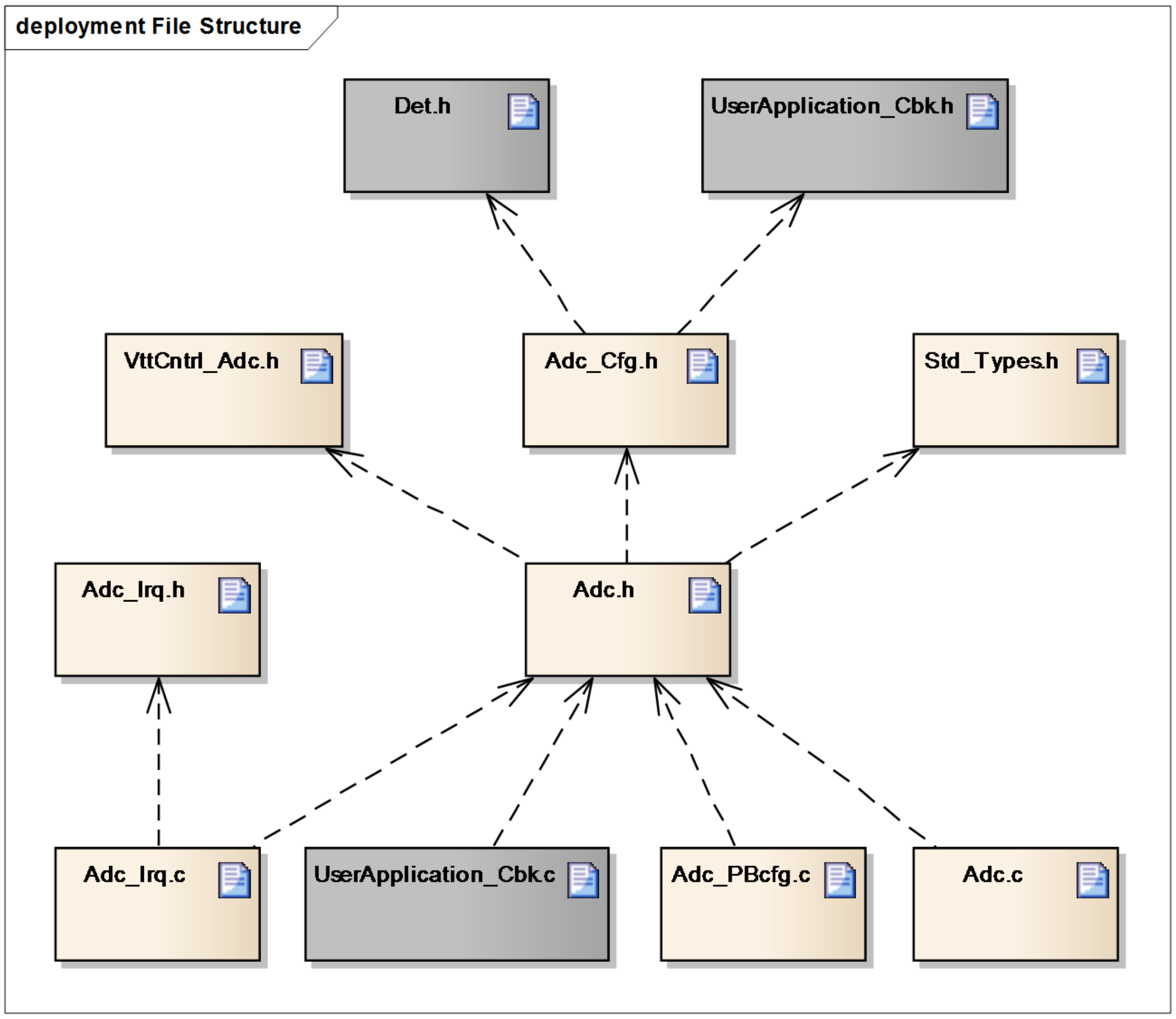


Figure 4-1 Include Structure

4.3 Dependencies on SW Modules

4.3.1 AUTOSAR OS (Optional)

An operating system can be used for task scheduling, interrupt handling, global suspend and restore of interrupts and creating of the Interrupt Vector Table.

4.3.2 DET (Optional)

The ADC module depends on the DET (by default) in order to report development errors. Detection and reporting of development errors can be enabled or disabled by the switch "Enable Development Error Detection".

4.3.3 SchM (Optional)

Beside the AUTOSAR OS the Schedule Manager provides functions that module ADC calls at begin and end of critical sections.

4.3.4 EcuM (Optional)

The module EcuM is responsible for the initialization of the ADC module.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

The types defined by the ADC are described in this chapter.

Type Name	C-Type	Description	Value Range
`Adc_ValueGroupType`	uint32	Type for reading the converted values of a channel group (raw, without further scaling, right aligned).	Limited by resolution (in this case 8 to 32 bit).
`Adc_GroupType`	uint8	Numeric ID of an ADC channel group. Use the symbolic group name on the API.	0 – 254. 255 is reserved

`Adc_StatusType`	enum	Current status of the conversion of the requested ADC Channel group.	`ADC_IDLE`
			`ADC_BUSY`
			`ADC_COMPLETED`
			`ADC_STREAM_COMPLETED`

5.2 Interrupt Service Routines provided by ADC

5.2.1 AdcIsr_<0…4>

Prototype	
`void` **`AdcIsr_<0…4>`** `( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Interrupt Service Routine for each Hardware Unit. The ISRs are called from the VTT Kernel if an ongoing conversion has finished. Within the function, a handler is called which do the state transitions, reads the samples, and calls the notification (and so on).	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	

Table 5-1 AdcIsr_<0…4>

5.3 Services provided by ADC

5.3.1 Adc_Init

Prototype	
void **Adc_Init** (const Adc_ConfigType* ConfigPtr)	
Parameter	
ConfigPtr	Pointer to the configuration struct of the ADC
Return code	
-	-
Functional Description	
This function initializes all internal variables. All group states are set to ADC_IDLE and all hardware triggers and group notifications are disabled.	
Particularities and Limitations	
> This function is synchronous. > This function is non reentrant. > This function is always available. > This service shall not be called during a running operation.	
Expected Caller Context	
> Expected to be called in application context, mostly during initialization phase.	

Table 5-2 Adc_Init

5.3.2 Adc_DeInit

Prototype	
void **Adc_DeInit** (void)	
Parameter	
-	-
Return code	
-	-
Functional Description	
This function deinitializes the module. This function shall return all ADC HW Units to a state comparable to the power on reset state. All interrupts and notifications are disabled.	
Particularities and Limitations	
> This function is synchronous. > This function is non reentrant. > This function can be enabled or disabled. > This service shall not be called during a running operation.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-3 Adc_DeInit

5.3.3 Adc_SetupResultBuffer

Prototype	

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

Parameter	
`Group`	Identifier of the requested ADC channel group.
`DataBufferPtr`	Pointer to the application's result buffer.
Return code	
`Std_ReturnType`	Function returns `E_OK` in case setting up result buffer was successful, otherwise `E_NOT_OK` is returned.
Functional Description	
This function initializes the group specific result buffer pointer with `DataBufferPtr`. This pointer shall reference the application result buffer to which all conversion results are stored. The user has to provide one application result buffer for each of the ADC channel groups. The buffer has to have capacity for m*n elements, where m is the number of samples and n the number of channels belonging to that group. For further details, see chapter 3.11	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This service shall be called after module initialization and before starting group conversion.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-4 Adc_SetupResultBuffer

Info
In the implementation `Adc_ValueGroupRefType` is used instead of `Adc_ValueGroupType*`. These types are equal due to a typedef.

5.3.4 Adc_StartGroupConversion

Prototype	
`void` **`Adc_StartGroupConversion`** `(Adc_GroupType group)`	
Parameter	
`group`	Identifier of the requested ADC channel group.
Return code	
-	-
Functional Description	
This function starts the conversion of all channels of the requested software triggered ADC channel group.	

Particularities and Limitations
> This function is asynchronous. > This function is reentrant for simultaneous access to different groups. > This function can be enabled or disabled. > This service shall be called after module initialization and buffer setup. > This function shall only be used for starting software triggered conversions.
Expected Caller Context
> Expected to be called in application context.

Table 5-5 Adc_StartGroupConversion

5.3.5 Adc_StopGroupConversion

Prototype	
`void` **`Adc_StopGroupConversion`** `(Adc_GroupType Group)`	
Parameter	
`Group`	Identifier of the requested ADC channel group.
Return code	
-	-
Functional Description	
This function stops conversion of the requested software triggered ADC channel group immediately, sets the group state to `ADC_IDLE` and disables the group notification for the requested group.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This function can be enabled or disabled. > This service shall be called after conversion has been started. > This function shall only be used for stopping software triggered conversions.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-6 Adc_StopGroupConversion

5.3.6 Adc_ReadGroup

Prototype

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

Parameter	
`Group`	Identifier of the requested ADC channel group.

`DataBufferPtr`	ADC results of all channels of the selected group are stored in the data buffer addressed with the pointer..
Return code	
`Std_ReturnType`	`E_OK`: Results are available and written to the buffer. `E_NOT_OK`: No results are available or development error occurred.
Functional Description	
This function copies the latest sample (a set of all channels of an ADC channel group) from application result buffer to application read buffer, which is referenced in `DataBufferPtr`. The result values are unsigned, right aligned, without scaling and do not bring any status information with them. The group channel values are stored in the order they are configured in the DaVinci Configurator Pro. Calling `Adc_ReadGroup` causes a state transition because the buffer content is invalidated: > Calling Adc_ReadGroup while status is ADC_BUSY: no state transition. > Calling Adc_ReadGroup while status is ADC_COMPLETED: state transition to ADC_BUSY > Calling Adc_ReadGroup while status is ADC_STREAM_COMPLETED: > State transition to ADC_BUSY for groups that have to be stopped by the user (explicit stop) > State transition to ADC_IDLE for groups that stop implicitly, e.g. SW-triggered, one-shot conversions.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This function can be enabled or disabled. > This service shall be called after conversion has been started.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-7 Adc_ReadGroup

Note
In the implementation `Adc_ValueGroupRefType` is used instead of `Adc_ValueGroupType*`. These types are equal due to a typedef.

5.3.7 Adc_EnableHardwareTrigger

Prototype	
`void` **`Adc_EnableHardwareTrigger`** `(Adc_GroupType group)`	
Parameter	
`group`	Identifier of the requested ADC channel group.
Return code	
-	-

Functional Description
This function enables the hardware trigger for the requested hardware triggered ADC channel group. Each time a trigger event occurs, current channel values are read out of CANoe (after the emulated sampling and conversion times have expired). Refer to chapter 3.13 for important timing topics.
Particularities and Limitations
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This function can be enabled or disabled . > This service shall be called after module initialization and buffer setup.
Expected Caller Context
> Expected to be called in application context.

Table 5-8 Adc_EnableHardwareTrigger

5.3.8 Adc_DisableHardwareTrigger

Prototype	
`void` **`Adc_DisableHardwareTrigger`** `(Adc_GroupType group)`	
Parameter	
`group`	Identifier of the requested ADC channel group.
Return code	
-	-
Functional Description	
This function disables the hardware trigger for the requested ADC channel group immediately. The group state is set to `ADC_IDLE` and notifications are switched off.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This function can be enabled or. > This service shall be called after hardware trigger has been enabled.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-9 Adc_DisableHardwareTrigger

5.3.9 Adc_EnableGroupNotification

Prototype	
`void` **`Adc_EnableGroupNotification`** `(Adc_GroupType group)`	
Parameter	
`group`	Identifier of the requested ADC channel group.
Return code	
-	-

Functional Description
This function enables the notification mechanism for the requested ADC channel group. By default (after initialization), notifications are disabled. If group notification is configured and turned on, a user-configured notification function will be called, each time a conversion is done.
Particularities and Limitations
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This function can be enabled or disabled. > This service shall be called after module initialization.
Expected Caller Context
> Expected to be called in application context.

Table 5-10 Adc_EnableGroupNotification

5.3.10 Adc_DisableGroupNotification

Prototype	
`void` **`Adc_DisableGroupNotification`** `(Adc_GroupType group)`	
Parameter	
`group`	Identifier of the requested ADC channel group.
Return code	
-	-
Functional Description	
This function disables the group notification, so that no notification is sent, when a conversion of all channels of this ADC channel group is done.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This function can be enabled or disabled. > This service shall be called after group notification has been enabled.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-11 Adc_DisableGroupNotification

5.3.11 Adc_GetGroupStatus

Prototype	
`Adc_StatusType` **`Adc_GetGroupStatus`** `(Adc_GroupType group)`	
Parameter	
`group`	Identifier of the requested ADC channel group.
Return code	
`Adc_StatusType`	Conversion status of the requested ADC channel group.

Functional Description
This function returns the conversion status of the requested ADC channel group. See chapter 3.4.
Particularities and Limitations
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This service shall be called after module initialization.
Expected Caller Context
> Expected to be called in application context.

Table 5-12 Adc_GetGroupStatus

5.3.12 Adc_GetStreamLastPointer

Prototype	

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

Parameter	
`Group`	Identifier of the requested ADC channel group.
`PtrToSamplePtr`	Reference to a user-declared pointer, which is set to point to the latest result of the first channel in buffer. The user has to take the design of the application result buffer into account. (See chapter 3.11)
Return code	
`Adc_StreamNumSampleType`	Returns the number of valid samples.
Functional Description	
This function returns the number of valid samples that are currently in application result buffer of the requested ADC channel group. Furthermore it provides a pointer to the latest available results. With the pointer position, the results of all group channels of all valid samples can be accessed. Therefore the user has to take the layout of the application result buffer into account. > Calling `Adc_GetStreamLastPointer` causes a state transition because the buffer content is invalidated: > Calling Adc_GetStreamLastPointer while status is ADC_BUSY: no state transition. > Calling Adc_GetStreamLastPointer while status is ADC_COMPLETED: state transition to ADC_BUSY > Calling Adc_GetStreamLastPointer while status is ADC_STREAM_COMPLETED: > State transition to `ADC_BUSY` for groups that have to be stopped by the user (explicit stop) State transition to `ADC_IDLE` for groups that stop implicitly, e.g. SW-triggered, one-shot conversions.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for simultaneous access to different groups. > This service shall be called after group conversion has been started.	
Call context	
> Task Context	

Table 5-13 Adc_GetStreamLastPointer

5.3.13 Adc_GetVersionInfo

Prototype	
`void Adc_GetVersionInfo (Std_VersionInfoType* versioninfo)`	
Parameter	
`versioninfo`	Pointer to where the version information of this module has to be stored.
Return code	
-	-
Functional Description	
This function returns the version information of the module. The version information includes: > Module Id > Vendor Id > Software version numbers	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant. > This function can be enabled or disabled. > This service can be called previous to module initialization.	
Expected Caller Context	
> Expected to be called in application context.	

Table 5-14 Adc_GetVersionInfo

5.4 Services used by ADC

In the following table services provided by other components, which are used by the ADC are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError

Table 5-15 Services used by the ADC

5.5 Callback Functions

The ADC does not provide callback functions.

5.6 Configurable Interfaces

5.6.1 Notifications

At its configurable interfaces the ADC defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the ADC but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

5.6.1.1 <Adc_GroupEndNotification>

Prototype	
`void <`**`Adc_GroupEndNotification>`**` ( void )`	
Parameter	
`Void`	--
Return code	
`void`	--
Functional Description	
This function is called when group conversion of the specified group has finished and notification function is configured and enabled by `Adc_EnableGroupNotification`. An individual notification callback can be associated to each ADC group.	
Particularities and Limitations	
> This feature can be enabled or disabled.	
Call context	
> Interrupt Context	

Table 5-16 <Adc_GroupEndNotification>

6 Configuration

6.1 Configuration Variants

The ADC supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the ADC parameters depend on the supported configuration variants. For their definitions please see the VTTAdc_bswmd.arxml file.

6.2 Configuration with DaVinci Configurator 5

The ADC module is configured with the help of the configuration tool DaVinci Configurator 5 (CFG5). The definition of each parameter is given in the corresponding BSWMD file.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
CANoe	Tool for simulation and testing of networks and electronic control units.
DaVinci Configurator	Configuration and generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
ADC	Analog Digital Converter
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SWS	Software Specification
VTT	vVIRTUALtarget

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

> News
> Products
> Demo software
> Support
> Training data
> Addresses

www.vector.com