Document Title	Specification of I/O Hardware Abstraction
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	047
Document Classification	Auxiliary

Document Version	2.1.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	2.1.0	AUTOSAR Release Management	• Revised requirement IDs • Editorial changes • Removed chapter(s) on change documentation
23.03.2011	2.0.3	AUTOSAR Administration	Legal disclaimer revised
23.06.2008	2.0.1	AUTOSAR Administration	Legal disclaimer revised
13.12.2007	2.0.0	AUTOSAR Administration	• Auto generation of chapters 8 and 10 with the Metamodel • Update of tables and some chapters of the document to stay compliant with correlated documents • Document meta information extended • Small layout adaptations made
14.02.2007	1.1.1	AUTOSAR Administration	• Various images corrected in PDFversion (printing problems)

Document Change History

Date	Version	Changed by	Change Description
31.01.2007	1.1.0	AUTOSAR Administration	• File structure updated • Traceability matrix corrected • Restriction for the usage of the SW-C template • Chapter about IOHWAB Runnable concept reworked • Chapter about IOHWAB description reworked • Adjustments in the configuration chapter • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
27.04.2006	1.0.0	AUTOSAR Administration	Initial Release

Table of Contents

1 Introduction and functional overview

This specification specifies the functionality and the configuration of the AUTOSAR Basic Software IO Hardware Abstraction. IO Hardware Abstraction is part of the ECU Abstraction Layer.

The IO Hardware Abstraction shall not be considered as a single module, as it can be designed as more than one module.This specification for the IO Hardware Abstraction is not intended to standardize this module or group of modules, but only its functional interfaces with other modules. Instead it is intended to be a guideline for the implementation.

Aim of the IO Hardware Abstraction is to make data transiting through the RTE fully independent of the ECU hardware. This means that the Software Component designer doesn’t need anymore to have the know-how how signals are affected on the physical level. Thus, IO Hardware Abstraction is ECU specific.

This will be mainly realized through the mapping of ECU signals on IO Hardware Abstraction (considered as a Software Component) ports. This document presents therefore the best way to map ECU signals to ports.

The IO Hardware Abstraction shall provide the service for initializing the whole IO Hardware Abstraction.

The intention of this document is:
- to explain which part of the Software Component template shall be used when defining an IO Hardware Abstraction.
- to give the way to define generic ports, where ECU signals are mapped.

The intention of this document is not:
- to provide C-APIs
- to provide a specific formalization for every ECU signal, like it is done via the standardization of functional data (body domain, powertrain, chassis domain)

Requirements in the SRS are referenced using [BSWxxx] where <xxx> is the requirement number. For example [BSW13905].

Requirements in the SWS are marked with [IoHwAb<n>] as first text in a paragraph. The scope of the requirement is the entire paragraph.

2 Acronyms and abbreviations

Abbreviation / Acronym:	*Description:*
AUTOSAR	AUTomotive Open System ARchitecture
BSW	Basic SoftWare
BSWMDT	Basic SoftWare Module DescripTion
DTD	Document Type Definition
ECU	Electronic Control Unit
HW	HardWare
Io Hw Ab	Input Output Hardware Abstraction
ISR	Interrupt Service Routine
MCAL	MicroController Abstraction Layer
OS	Operating System
RTE	RunTime Environment
SW	SoftWare
SWCT	SoftWare Component Template
XML	eXtensible Markup Language

Expressions used in this document

Expression	*Description*	*Example*
Callback	Within this document, the term 'callback' is used for API services which are intended for notifications to other BSW modules.	
Callout	This definition comes from the ECU state manager SWS "the term 'callout' is used for function stubs which can be filled by the system designer, usually at configuration time, with the purpose to add functionality to the ECU State Manager. Callouts are separated into two classes, where one class is optional to be filled. The other class is mandatory and serves as a Hardware Abstraction"	
Class	A class represents a kind of electrical connection to the ECU. It could be for example an analogue, a discrete,…	Analogue class, Discrete class, …
Client / Server communication	This definition is an extract from [9]: Client-server communication involves two entities, the client which is the requirer (or user) of a service and the server that provides the service. The client initiates the communication, requesting that the server performs a service, transferring a parameter set if necessary. The server, in the form of the RTE, waits for incoming communication requests from a client, performs the requested service and dispatches a response to the client's request. So, the direction of initiation is used to categorize whether an AUTOSAR Software Component is a client or a server.	
Electrical signal	It is the electrical signal on the pin of the ECU	Physical input voltage at an ECU-Pin

ECU pin	It is an hardware electrical connection of the ECU with the rest of the electronic system	
ECU Signal	It is the **software representation** of an electrical signal. An ECU signal has attributes and a symbolic name	Input voltage ,Discrete Output, PWM Input
ECU Signal group	It is the **software representation** of a group of electrical signals. A group is included in the class "discrete"	Only for discrete Inputs and discrete Outputs
Attributes	Characteristics that can be Software (SW) and Hardware (HW) for each kind of ECU Signals existing in a ECU	Range, Lifetime / delay
Sender-receiver communication	This definition is an extract from [9]: Sender-receiver communication involves the transmission and reception of signals consisting of atomic data elements that are sent by one component and received by one or more components. A sender-receiver interface can contain multiple data elements. Sender-receiver communication is one-way - any reply sent by the receiver is sent as a separate sender-receiver communication. A port of a component that requires an AUTOSAR sender-receiver interface can read the data elements described in the interface and a port that provides the interface can write the data elements.	
Symbolic name	The symbolic name of a ECU signal is used by the IO Hardware Abstraction to make a link (function, pin)	

ECU Signal attributes

Expression	*Description*	*Example*
Range	This is a functional range and not an electrical range. All the range is used either for functional needs or for diagnosis detections For analogue ECU signals [lowerLimit...upperLimit] (Voltage, current). For the particular case of a resistance signal and a timing signal (period), the lowerLimit value can not be negative.	[-12Volts...+12Volts] (voltage) [0,1] (discrete signals) [0…upperLimit] (period timing signal) [-100…100%] (Duty Cycle based timing signal)
Resolution	This attribute is for many Classes dependent on the range and the Data Type. Example: (upperLimit - lowerLimit) / ($2^{datatypelength}$ -1) For the others classes, it is known and defined.	[-12 Volts…+12Volts] Data Type : 16 bits Resolution => 24 / 65535
Accuracy	It depends of hardware peripheral used for acquisition and/or generation.	ADC converter could be a 8/10/12/16 bits converter
Inversion	Inversion between the physical value and the logical value. This attribute is not visible but done by IO Hardware Abstraction to deliver expected values to users.	Physical HighState → (Signal=False) Physical LowState → (Signal=True)

Expression	*Description*	*Example*
Sampling rate	Time period required to get a Signal value.	Sampling rate for a sampling windows (burst)

3 Related documentation

3.1 Input documents

[1] AUTOSAR List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_General.pdf

[4] Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

[5] AUTOSAR Glossary
AUTOSAR_Glossary.pdf

[6] Requirements on SPAL
AUTOSAR_SRS_SPAL.pdf

[7] Requirements on I/O Hardware Abstraction
AUTOSAR_SRS_IOHW_Abstraction.pdf

[8] Software Component Template
AUTOSAR_SoftwareComponentTemplate.pdf

[9] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[10] Specification of ECU State Manager
AUTOSAR_SWS_ECU_StateManager.pdf

[11] Specification of ECU Resource Template
AUTOSAR_ECU_ResourceTemplate.pdf

[12] Specification of ADC Driver
AUTOSAR_SWS_ADC_Driver.pdf

[13] Specification of DIO Driver
AUTOSAR_SWS_DIO_Driver.pdf

[14] Specification of ICU Driver
AUTOSAR_SWS_ICU_Driver.pdf

[15] Specification of PWM Driver
AUTOSAR_SWS_PWM_Driver.pdf

[16] Specification of PORT Driver
AUTOSAR_SWS_PORT_Driver.pdf

[17] Specification of GPT Driver
AUTOSAR_SWS_GPT_Driver.pdf

[18] Specification of SPI Handler/Driver
AUTOSAR_SWS_SPI_HandlerDriver.pdf

[19] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[20] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

None

4 Constraints and assumptions

4.1 Limitations

No limitations

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 Interface with MCAL drivers

5.1.1 Overview

The following picture shows the IO Hardware Abstraction. It is located above MCAL drivers. That means the IO Hardware Abstraction will call the drivers API for managing on chip devices. The configuration of MCAL drivers depends on the quality of the ECU signals to be provided by the IO Hardware Abstraction. For instance, it could be required to have notification when a relevant change occurs on the pin level (rising edge, falling edge). The system designer has to configure the MCAL driver to allow notification for a given signal. Notifications come from drivers and are handled within the IO Hardware Abstraction.
Please notice that IO Hardware Abstraction is not intended to abstract GPT functionalities, but rather to use them to perform its own functionalities. The interfacing with GPT driver is shown because it is part of MCAL drivers.

The following picture shows all interfaces with MCAL drivers:

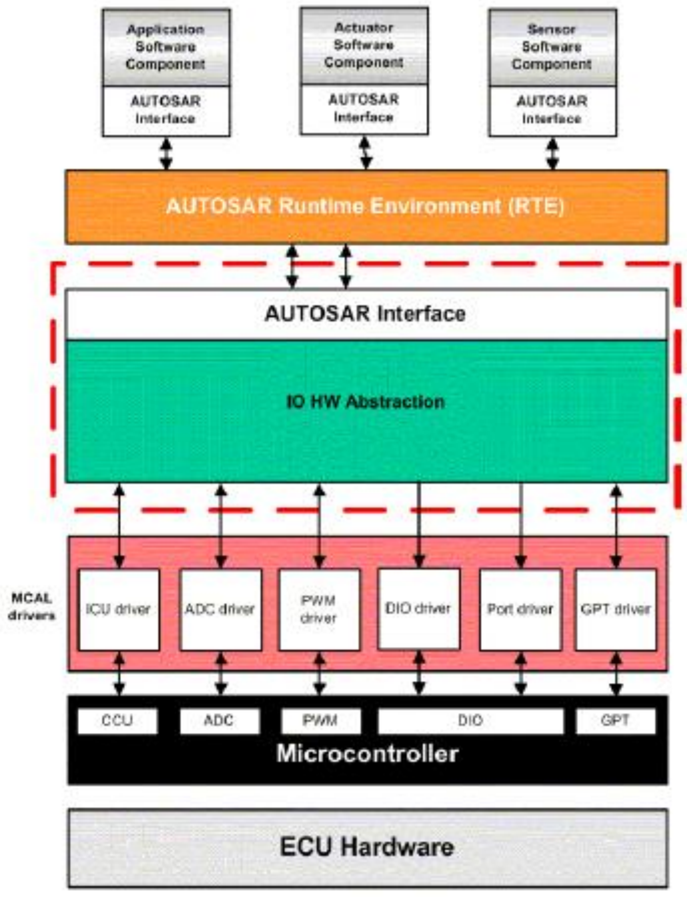

Figure 5.1: Interfaces with MCAL drivers

5.1.2 Summary of interfaces with MCAL drivers

IoHwAb078: IO Hardware Abstraction implementation has interfaces with all IO MCAL drivers listed below, and with the GPT driver:

	MCAL drivers					
IoHwAb	**ADC driver**	**PWM driver**	**ICU driver**	**DIO driver**	**PORT driver**	**GPT driver**
Calls API of	**X**	**X**	**X**	**X**	**X**	**X**
Receives notifications from	**X**	**X**	**X**	**-**	**-**	**X**

The table above must be red as following:
- The IO Hardware Abstraction calls API of the ADC driver
- The IO Hardware Abstraction receives notifications from the ADC driver.
- The IO Hardware Abstraction does not receive notifications from the DIO driver.

A complete list of all API is given in chapter 8.6.1

5.2 Interface with the communication drivers

IoHwAb079: IO Hardware Abstraction implementation has interfaces with some communication drivers, if on-board devices are managed (for instance by SPI).

The following picture shows the IO Hardware Abstraction, where some signals come from / are set via the SPI handler / driver.
According to the Layered Software Architecture [2] (*ID03-16*), the IO Hardware Abstraction contains dedicated drivers to manage external devices for instance:
- A driver for external ADC driver, connected via SPI.
- A driver for external IO realized on an Asic device, connected via SPI.

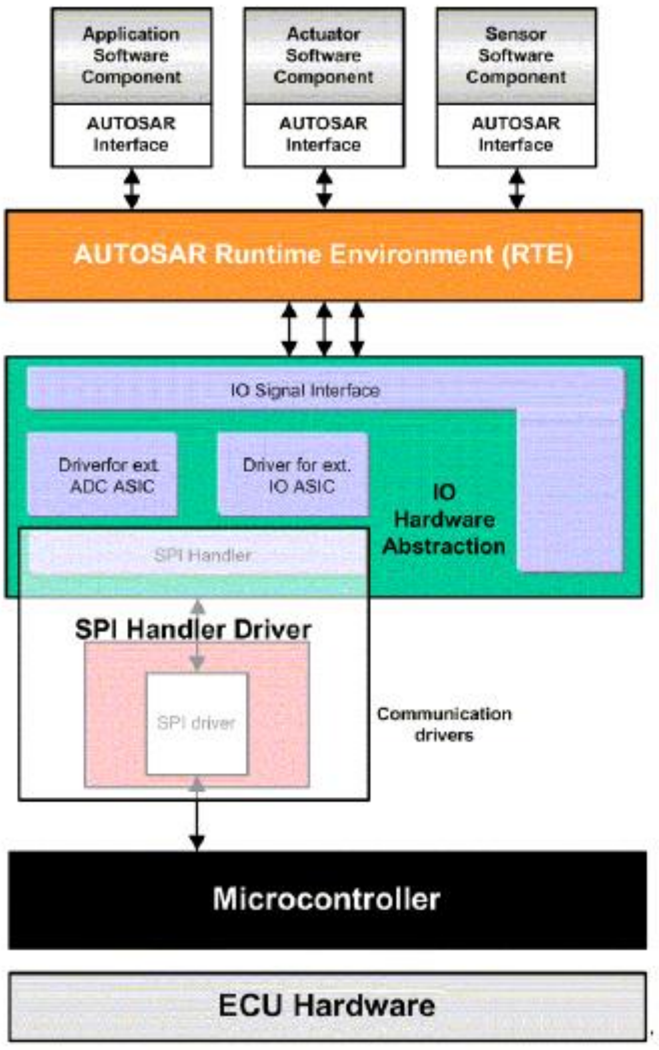

Figure 5.2: Interfaces with communication drivers

5.3 Interface with System Services

IoHwAb044: The IO Hardware Abstraction implementation has some interfaces with system services:

- ECU state manager (init function, shutdown function).
- DEM: Diagnostic Event Manager
- DET: Development Error Tracer
- BSW Scheduler (BSW-runnable entities will be members of OS tasks)

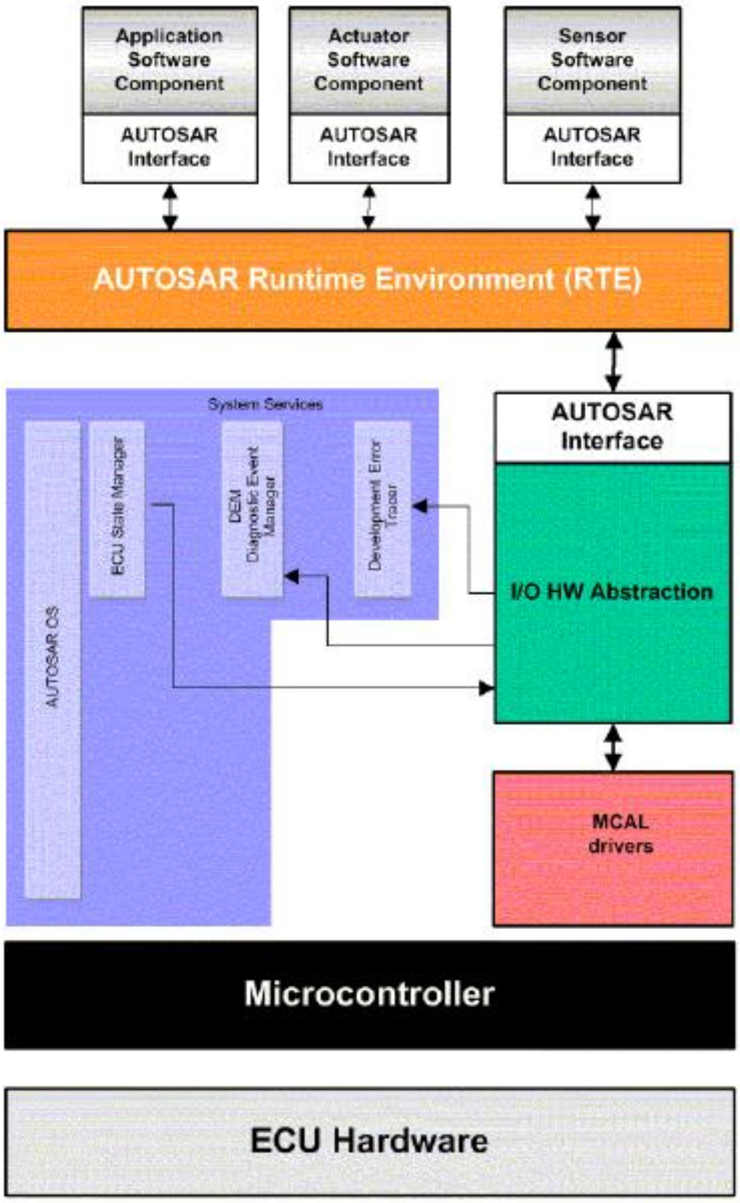

Figure 5.3: Interfaces with system services

5.4 File structure

5.4.1 Code file structure

IoHwAb097: The code file structure shall not be defined within this specification.

5.4.2 Header file structure

IoHwAb064: The include file structure shall be as follows:

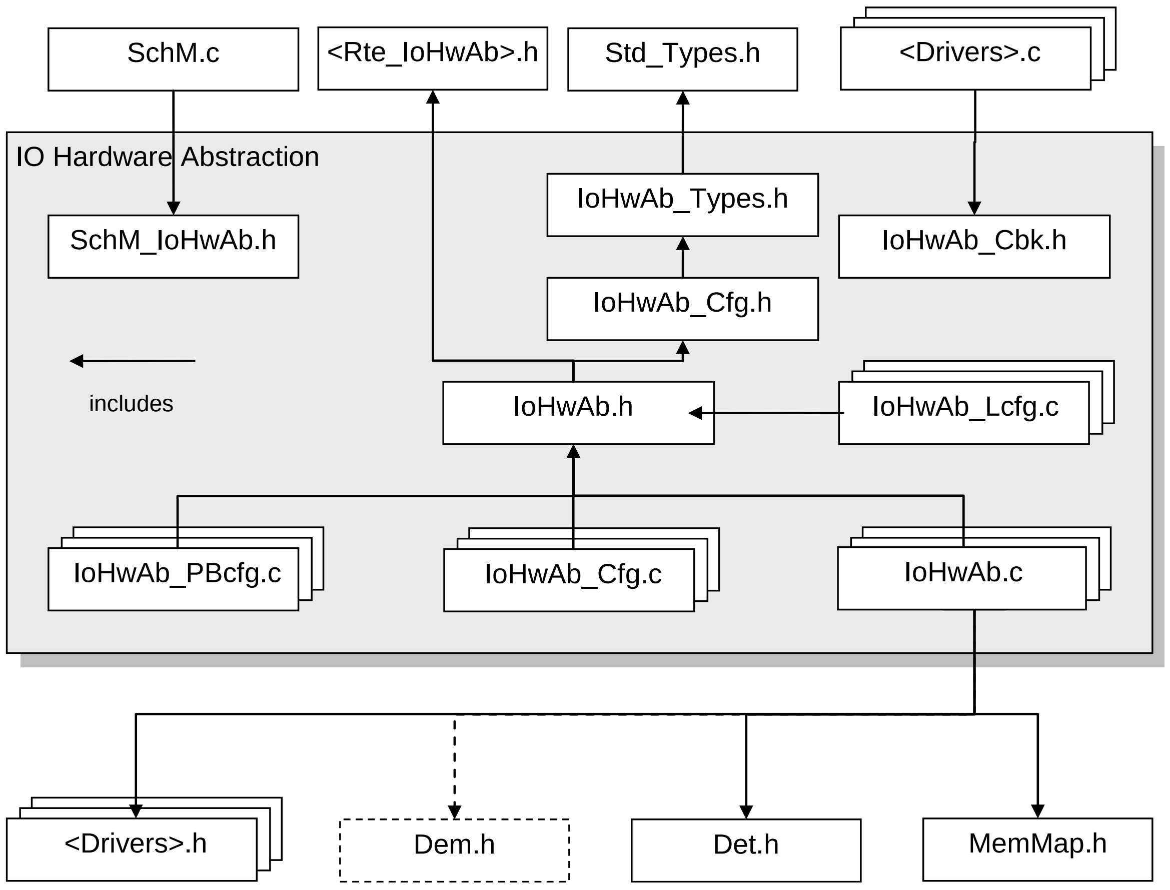

Figure 5.4: File structure

The files "<Drivers>.h" represent the different header files of the driver which will be actually interfaced to IO Hardware Abstraction. In the same way, the files "<Drivers>.c" represent the code files of the drivers which will be actually interfaced and need to call IO Hardware Abstraction callbacks.

The IO Hardware Abstraction C files (represented with name "IoHwAb.c") shall optionally include the Dem.h file if any production error will be issued by the implementation. By this inclusion, the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.

IoHwAb095: The pre-compile time parameters shall be placed in IoHwAb_Cfg.h

IoHwAb062: Additional header files can be added to the structure shown in the picture. This shall be defined in the design document of the IO Hardware Abstraction document.

IoHwAb112: The IO Hardware Abstraction shall not be considered as a single module. IO Hardware Abstraction can be designed as more than one module.source and header file. The name pattern shall not be fully standardized, however file names shall be prefixed with "IoHwAb_<reference>", where field <reference> could be the proprietary reference, in order to avoid name clashes.

6 Requirements traceability

Document: AUTOSAR General requirements on Basic Software Modules, [3]

Requirement	***Satisfied by***
[BSW003] Version identification	Requirement to be taken into account during implementation
[BSW00300] Module naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00301] Limit imported information	Requirement to be taken into account during implementation
[BSW00302] Limit exported information	Requirement to be taken into account during implementation
[BSW00304] AUTOSAR integer data types	Requirement to be taken into account during implementation
[BSW00305] Self-defined data types naming convention	[IoHwAb065]
[BSW00306] Avoid direct use of compiler and platform specific keywords	Requirement to be taken into account during implementation
[BSW00307] Global variables naming convention	Requirement on naming rules to be taken into account during implementation
[BSW00308] Definition of global data	Requirement to be taken into account during implementation
[BSW00309] Global data with read-only constraint	Requirement to be taken into account during implementation
[BSW00310] API naming convention	Requirement on naming rules to be taken into account during implementation
[BSW00312] Shared code shall be reentrant	Requirement to be taken into account during implementation
[BSW00314] Separation of interrupt frames and service routines	[IoHwAb097] Requirement to be taken into account during implementation
[BSW00318] Format of module version numbers	[IoHwAb056]
[BSW00321] Enumeration of module version numbers	Not applicable (requirement on implementation, not for specification)
[BSW00323] API parameter checking	[IoHwAb067]
[BSW00325] Runtime of interrupt service routines	Not applicable (this module does not implement any interrupt service routines)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (requirement on implementation, not for specification)
[BSW00327] Error values naming convention	Requirement on naming rules to be taken into account during implementation
[BSW00328] Avoid duplication of code	Requirement to be taken into account during implementation
[BSW00329] Avoidance of generic interfaces	Not Applicable: (requirement on software architecture, not for a single module)
[BSW00330] Usage of macros / inline functions instead of functions	Requirement to be taken into account during implementation
[BSW00331] Separation of error and status values	Requirement to be taken into account during implementation

[BSW00333] Documentation of callback function context	[IoHwAb033] Requirement to be taken into account during implementation
[BSW00334] Provision of XML file	Not applicable (requirement on documentation, not on specification)
[BSW00335] Status values naming convention	Requirement on naming rules to be taken into account during implementation
[BSW00336] Shutdown interface	[IoHwAb044], [IoHwAb036] (but no API provided by IO Hardware abstraction for deinit)
[BSW00337] Classification of errors	[IoHwAb067]
[BSW00338] Detection and Reporting of development errors	[IoHwAb051], [IoHwAb108]
[BSW00339] Reporting of production relevant error status	[IoHwAb052], [IoHwAb055]
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on documentation, not on specification)
[BSW00342] Usage of source code and object code	Not applicable (requirement on software architecture, not for a single module)
[BSW00343] Specification and configuration of time	Not applicable (no timings configurable)
[BSW00344] Reference to link time configuration	[IoHwAb060]
[BSW00345] Pre-compile-time configuration	[IoHwAb096], [IoHwAb064]
[BSW00346] Basic set of module files	[IoHwAb064], [IoHwAb097] Requirement to be taken into account during implementation
[BSW00347] Naming separation of different instances of BSW drivers	Implementation specific : Requirement to be taken into account during implementation
[BSW00348] Standard type header	[IoHwAb064] Requirement to be taken into account during implementation
[BSW00350] Development error detection keyword	[IoHwAb053], [IoHwAb108]
[BSW00353] Platform specific type header	Requirement to be taken into account during implementation
[BSW00355] Do not redefine AUTOSAR integer data types	Requirement to be taken into account during implementation
[BSW00357] Standard API return type	Requirement on naming rules to be taken into account during implementation
[BSW00358] Return type of `init()` functions	Requirement to be taken into account during implementation
[BSW00359] Return type of callback functions	Requirement to be taken into account during implementation
[BSW00360] Parameters of callback functions	Requirement to be taken into account during implementation
[BSW00361] Compiler specific language extension header	Requirement to be taken into account during implementation
[BSW00369] Do not return development error codes via API	[IoHwAb054]
[BSW00370] Separation of callback interface from API	[IoHwAb097] Requirement to be taken into account during implementation
[BSW00371] Do not pass function pointers via API	Requirement to be taken into account during implementation

[BSW00373] Main processing function naming convention	Requirement on naming rules to be taken into account during implementation
[BSW00374] Module vendor identification	[IoHwAb056] Element `IOHWAB_VENDOR_ID`
[BSW00375] Notification of wake-up reason	[IoHwAb047]
[BSW00376] Return type and parameters of main processing functions	Not Applicable: (No Main Function)
[BSW00377] Module specific API return types	Requirement on naming rules to be taken into account during implementation
[BSW00378] AUTOSAR boolean type	Requirement to be taken into account during implementation
[BSW00379] Module identification	[IoHwAb056] Element `IOHWAB_MODULE_ID`
[BSW00380] Separate C-Files for configuration parameters	[IoHwAb064] Requirement to be taken into account during implementation
[BSW00381] Separate configuration header file for pre-compile time parameters	[IoHwAb064] Requirement to be taken into account during implementation
[BSW00383] List dependencies of configuration files	[IoHwAb064]
[BSW00384] List dependencies to other modules	See chapter 5 [IoHwAb078], [IoHwAb079], [IoHwAb044]
[BSW00385] List possible error notifications	[IoHwAb051]
[BSW00386] Configuration for detecting an error	Requirement to be taken into account during implementation
[BSW00387] Specify the configuration class of callback function	Chapter 8.6
[BSW00388] Introduce containers	See chapter 10.2
[BSW00389] Containers shall have names	See chapter 10.2
[BSW00390] Parameter content shall be unique within the module	See chapter 10.2
[BSW00391] Parameter shall have unique names	See chapter 10.2
[BSW00392] Parameters shall have a type	See chapter 10.2
[BSW00393] Parameters shall have a range	See chapter 10.2
[BSW00394] Specify the scope of the parameters	See chapter 10.2
[BSW00395] List the required parameters (per parameters)	See chapter 10.2
[BSW00396] Configuration classes	See chapter 10.2
[BSW00397] Pre-compile-time parameters	See chapter 10.2
[BSW00398] Link-time parameters	Not applicable (no link-time configuration parameters)
[BSW00399] Loadable Post-build time parameters	Not applicable (no post build time configuration parameters)
[BSW004] Version check	[IoHwAb066]
[BSW00400] Selectable Post-build time parameters	Not applicable (no post build time configuration parameters)
[BSW00401] Documentation of multiple instances of configuration parameters	Requirement to be taken into account during implementation
[BSW00402] Published information	[IoHwAb056]
[BSW00404] Reference to post build time configuration	Not applicable (no post build time configuration for IO Hardware Abstraction)
[BSW00405] Reference to multiple configuration sets	Not applicable (no multiple configuration sets for IO Hardware

	Abstraction)
[BSW00406] Check module initialization	[IoHwAb102]
[BSW00407] Function to read out published parameters	[IoHwAb057], [IoHwAb058]
[BSW00408] Configuration parameter naming convention	See chapter 10.2
[BSW00409] Header files for production code error IDs	[IoHwAb064]
[BSW00410] Compiler switches shall have defined values	Requirement to be taken into account during implementation
[BSW00411] Get version info keyword	[IoHwAb110]
[BSW00412] Separate H-File for configuration parameters	[IoHwAb095]
[BSW00413] Accessing instances of BSW modules	Implementation specific : Requirement to be taken into account during implementation
[BSW00414] Parameter of init function	Requirement to be taken into account during implementation
[BSW00415] User dependent include files	Implementation specific : Requirement to be taken into account during implementation
[BSW00416] Sequence of Initialization	Not applicable: (Software integration requirement)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not Applicable: Module is a BSW
[BSW00419] Separate C-Files for pre-compile time configuration parameters	The code file structure is not defined within this specification and is left up to the implementer. Requirement to be taken into account during implementation
[BSW00422] Debouncing of production relevant error status	Implementation specific : Requirement to be taken into account during implementation
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	[IoHwAb001]
[BSW00424] BSW main processing function task allocation	Not applicable (this is a general integration requirement)
[BSW00425] Trigger conditions for schedulable objects	Implementation specific : Requirement to be taken into account during implementation and integration
[BSW00426] Exclusive areas in BSW modules	Implementation specific : Requirement to be taken into account during implementation
[BSW00427] ISR description for BSW modules	Implementation specific : Requirement to be taken into account during implementation
[BSW00428] Execution order dependencies of main processing functions	Not applicable: No Main function in this module
[BSW00429] Restricted BSW OS functionality access	Implementation specific : Requirement to be taken into account during implementation
[BSW00431] The BSW Scheduler module implements task bodies	Implementation specific : Requirement to be taken into account during implementation and Integration
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable: (No Main function in this module)
[BSW00433] Calling of main processing functions	Implementation specific : Requirement to be taken into account during implementation/Integration
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Implementation specific : Requirement to be taken into account during implementation/Integration
[BSW00435] Module Header File Structure for the Basic Software Scheduler	[IoHwAb064]
[BSW00436] Module Header File Structure for the	[IoHwAb064]

Memory Mapping	
[BSW00437] NoInit--Area in RAM	Implementation specific : Requirement to be taken into account during implementation/Integration
[BSW00438] Post Build Configuration Data Structure	Implementation specific : Requirement to be taken into account during implementation/Integration
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable: (requirement on MCAL drivers, IO Hardware Abstraction belongs to the ECU abstraction)
[BSW006] Platform independency	Requirement to be taken into account during implementation
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW009] Module User Documentation	Requirement to be taken into account during implementation
[BSW010] Memory resource documentation	Requirement to be taken into account during implementation
[BSW101] Initialization interface	[IoHwAb044], [IoHwAb036], [IoHwAb059] [IoHwAb060], [IoHwAb061]
[BSW158] Separation of configuration from implementation	[IoHwAb097] Requirement to be taken into account during implementation
[BSW159] Tool-based configuration	[IoHwAb040], [IoHwAb045]
[BSW160] Human-readable configuration data	Not applicable (requirement on documentation, not on specification)
[BSW161] Microcontroller abstraction	Not applicable (requirement on software architecture, not for a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on software architecture, not for a single module)
[BSW164] Implementation of interrupt service routines	Not applicable (this module does not implement any interrupt service routines)
[BSW167] Static configuration checking	Not applicable (requirement on configuration tool)
[BSW168] Diagnostic interface of SW components	Not applicable (this module does not provide special diagnostic features)
[BSW170] Data for reconfiguration of AUTOSAR SW-components	Not applicable (no reconfiguration for IO Hardware Abstraction)
[BSW171] Configurability of optional functionality	[IoHwAb100]
[BSW172] Compatibility and documentation of scheduling strategy	Requirement to be taken into account during implementation

Document: AUTOSAR requirements on Basic Software, cluster SPAL [6]

Requirement	***Satisfied by***
[BSW12056] Configuration of notification mechanisms	[IoHwAb032], [IoHwAb033], [IoHwAb034]
[BSW12057] Driver module initialisation	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12063] Raw value mode	Not applicable

	(requirement not explicitly for IO Hardware Abstraction)
[BSW12064] Change of operation mode during running operation	Not applicable (There is no operation mode defined for IO Hardware Abstraction)
[BSW12067] Setting of wake-up conditions	Not applicable (Requirement only for drivers, and IO Hardware Abstraction is not in charge of interrupt handling)
[BSW12068] MCAL initialization sequence	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12069] Wake-up notification of ECU State Manager	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12075] Use of application buffers	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12077] Non-blocking implementation	Not applicable (requirement for implementation, not for specification)
[BSW12078] Runtime and memory efficiency	Not applicable (requirement for implementation, not for specification)
[BSW12092] Access to drivers	Not applicable (requirement for implementation, not for specification)
[BSW12125] Initialization of hardware resources	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12129] Resetting of interrupt flags	Not applicable (IO Hardware Abstraction is not in charge of interrupt handling)
[BSW12163] Driver module deinitialization	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12169] Control of operation mode	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12263] Object code compatible configuration concept	Not applicable
[BSW12264] Specification of configuration items	Not applicable
[BSW12265] Configuration data shall be kept constant	Not applicable (requirement for implementation, not for specification)
[BSW12267] Configuration of wakeup sources	[IoHwAb047]
[BSW12448] Behavior after development error detection	[IoHwAb051], [IoHwAb054]
[BSW12461] Responsibility for register initialization	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12462] Provide settings for register initialization	Not applicable (IO Hardware Abstraction is not a driver)
[BSW12463] Combine and forward settings for register initialization	Not applicable (IO Hardware Abstraction is not a driver)
[BSW157] Notification mechanisms of drivers and handlers	Not applicable (IO Hardware Abstraction is neither a driver nor an handler)

Document: AUTOSAR requirements on IO Hardware Abstraction, [7]

Requirement	***Satisfied by***
[BSW12232] Symbolic Name for each Signal	[IoHwAb045]
[BSW12242] Onboard peripherals abstraction	[IoHwAb030]
[BSW12248] ECU Hardware protection	[IoHwAb038]
[BSW12319] Independency between physical and logical Level	[IoHwAb046]

[BSW12323] Simultaneous update of several discrete outputs	TO BE DEFINED
[BSW12324] Simultaneous Get / Read of several discrete Inputs	TO BE DEFINED
[BSW12338] Synchronous interface for signal access	TO BE DEFINED
[BSW12339] Guarantee worst case delay times	Requirement on implementation, not on specification
[BSW12409] Values within one static range for each signal	[IoHwAb014]
[BSW12410] Measurement of input voltage	[IoHwAb005]
[BSW12411] Control of output voltage	[IoHwAb005]
[BSW12412] Get / Read a discrete input	[IoHwAb006]
[BSW12413] Measurement of input current	[IoHwAb005]
[BSW12414] Control of Duty Cycle for a periodic Signal	[IoHwAb009]
[BSW12415] Measurement of connected resistance	[IoHwAb005]
[BSW12416] Control the period time of a signal	[IoHwAb009]
[BSW12417] Measurement of the period time of signals	[IoHwAb009]
[BSW12418] Control of discrete powered outputs	[IoHwAb008]
[BSW12419] Failure Monitoring	[IoHwAb008]
[BSW12445] Measurement of Duty Cycle of a periodic Signal	[IoHwAb009]
[BSW12449] Signal groups	[IoHwAb007]
[BSW12450] Report discrete input changes	[IoHwAb022]
[BSW12451] No hardware failure recovery	[IoHwAb039]
[BSW12452] Failure management: test pulse	[IoHwAb023], [IoHwAb083], [IoHwAb086] [IoHwAb091], [IoHwAb094], [IoHwAb111]
[BSW13900] Diagnostic of output signal, detection of short-circuit to the ground	[IoHwAb008]
[BSW13901] Diagnostic of Discrete signal, detection of short-circuit to +Ubat	[IoHwAb008]
[BSW13902] Diagnostic of Discrete signal, detection of open circuit	[IoHwAb008]
[BSW13903] Diagnostic of Discrete output signal, detection of overload	[IoHwAb008]
[BSW13904] Diagnostic of Discrete signal, detection of over temperature	[IoHwAb008]
[BSW13905] Uniqueness of the IO Hardware Abstraction	[IoHwAb025]

7 Functional specification

7.1 ECU firmware software

The IO Hardware Abstraction, as a part of the ECU abstraction, has been defined as **ECU firmware.**

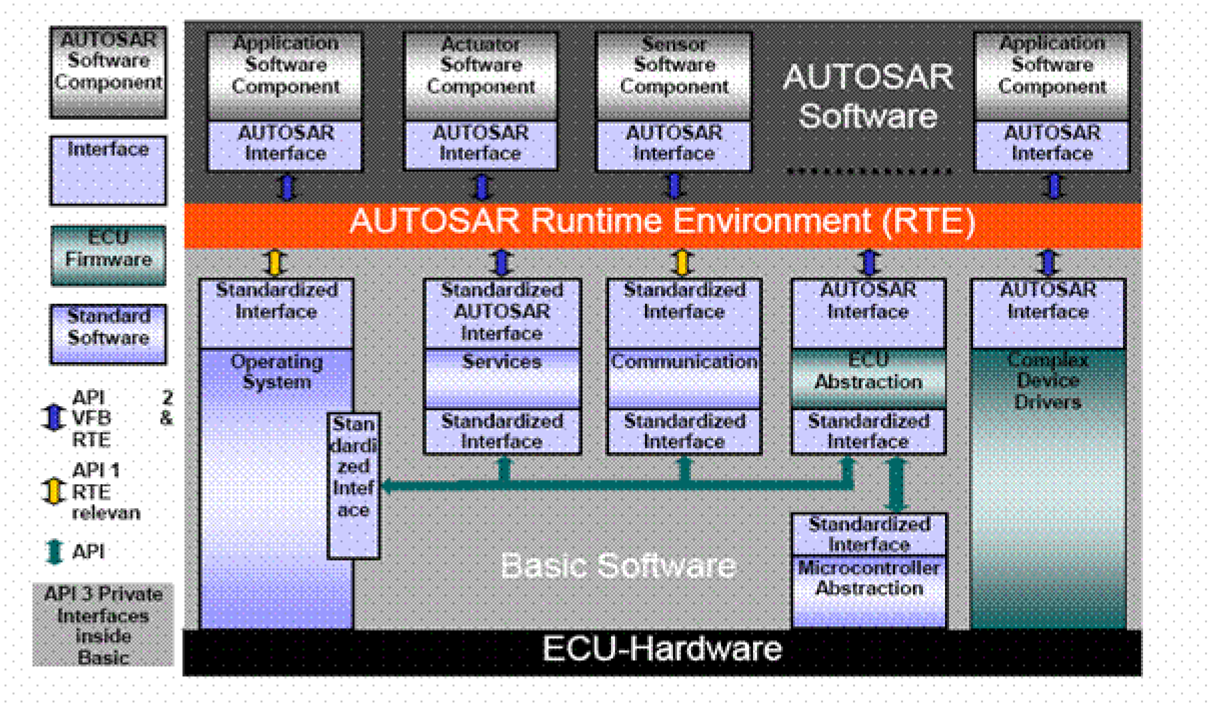

Figure 7.1: Autosar architecture

7.1.1 Background & Rationale

According to the AUTOSAR glossary [5], ECU firmware is ECU schematic dependent software located below the AUTOSAR RTE.

7.1.2 Requirements for firmware implementation

General requirements for IO Hardware Abstraction are related to hardware protection.

IoHwAb038: ECU firmware mainly means that this software is compatible and adapted to the ECU-Hardware. All strategies to protect the hardware must be included in this software. This document does not intend to standardize or give a recommendation for such hardware protection.

IoHwAb039: The IO Hardware Abstraction contains strategies to protect the hardware, but does not include hardware failure recovery strategies. The IO Hardware Abstraction shall not decide alone to switch on again an output, that has been switched off for hardware protection reasons. Such a strategy to recover a failure shall be defined in a Software Component.

Only the interfaces to the customer (other Software Components) are specified by the usage of the Software Component template.
That also means that the internal behavior of the IO Hardware Abstraction will never be standardized. The implementation cannot be specified.

That includes the usage of the lower layer (MCAL).

There is no IO Hardware Abstraction scalability. The customer Software Component specifies what it is needed (Quality of signal) and the IO Hardware Abstraction has to realize it.

7.2 ECU Signals Concept

7.2.1 Background & Rationale

One goal of AUTOSAR is to standardize interfaces. This is true only for SW-C located above the RTE. Below the RTE, services interfaces are standardized, but interfaces linked to the abstraction inputs/outputs can not be standardized.

Interfaces below the RTE represent an abstraction of electrical signal (The type of the data is specified at this level):

– either coming from others ECU / addressed to others ECU (e.g. via a CAN network)
– or coming from the ECU Inputs / addressed to ECU Outputs

Ports are entry points of an AUTOSAR components. They are typed by an interface. These interfaces correspond to “ECU signals”.

The concept of ECU signal comes from the necessity to guarantee the interchangeability of the BSW platforms.

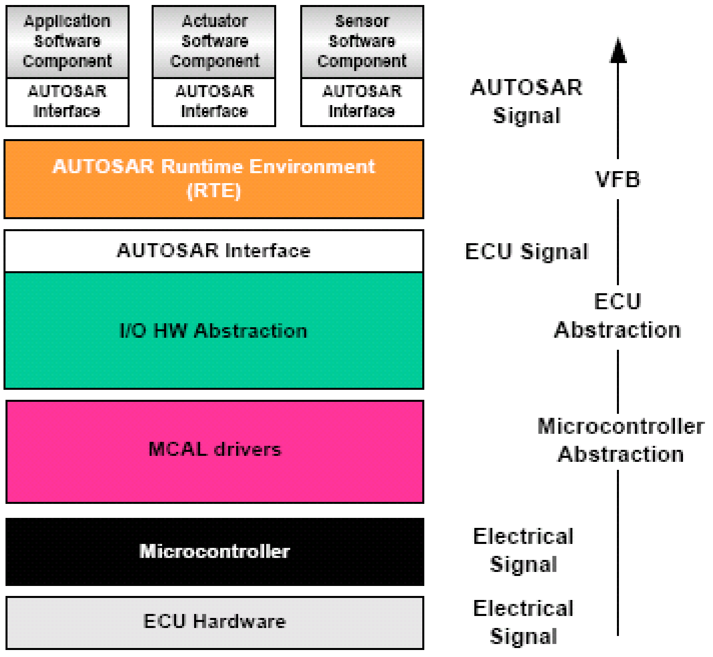

Figure 7.2: ECU Signal

7.2.2 Requirements about ECU signals

IoHwAb030: The IO Hardware Abstraction handles all Inputs and Outputs directly connected to the ECU (except those that have a dedicated driver, like CAN, see requirement [IoHwAb063]).
It includes all Inputs and Outputs, directly mapped on microcontroller ports, or on an on-board peripheral. All communications between the microcontroller and peripherals (excepting sensor and actuators, and peripherals managed by complex drivers) are hidden by the IO Hardware Abstraction, while considering the provided interfaces.

IoHwAb063: An ECU is connected to the rest of the system through networks and inputs and outputs pins. Networks are out of scope of this document and each ECU Signal represents one electrical signal, which means at least one input or output ECU pin.

The software at this layer shall abstract the ECU pins. Looking from this place (for example using an oscilloscope), Inputs and Outputs are only Electrical Signals. Hence, all that is defined in this document is related to this concept of Electrical Signal. One extension of this concept concerns Diagnosis (electrical failure status). Diagnosis are not visible from ECU connectors but are provided by the IO Hardware Abstraction.

Electrical signals with similar behavior shall form a class. Therefore, ECU signals, which denote the software representation of electrical signals shall have an association to a class and shall have characteristic attributes.
Possible classes and their attributes are listed below in this Chapter.

7.3 ECU signal classes

7.3.1 Background & Rationale

From the point of view of the ECU, all ECU Signals connected have not the same electrical behavior. Thus, they cannot be characterized using the same Attributes listed further in the document. To represent those behaviors in a model based schematic, the term Class is defined to group together ECU Signals with the same ECU electrical behavior.

7.3.2 Requirements about ECU signal classes

7.3.2.1 Analogue Class

[**IoHwAb005**]: The Analogue Class aggregates all ECU Signals, corresponding to Electrical Signals that are used as an analogue Input or Output Signal by the ECU Software. ECU Analogue Signals provided by the IO Hardware Abstraction could be used as three electrical types:

- Voltage Type
- Current Type
- Resistance Type

7.3.2.2 Discrete Class

IoHwAb006: The Discrete Class aggregates all ECU Signals, corresponding to electrical signal that are used as a discrete Input or Output Signal by the ECU Software. ECU Discrete Signals provided by IO Hardware Abstraction are booleans.
That means, only the following values are authorized:

- “1” to symbolise an electrical high level on the ECU pin
- “0” to symbolise an electrical low level on the ECU pin

IoHwAb007: An ECU Signals Group is composed by multiple ECU Signals belonging to this Discrete Class and with the same Direction Attribute. The definition of a signal group is done during the configuration step.

Note: On the level of MCAL drivers, it is allowed to change the direction of an ECU pin. The user has to define two different ECU signals with different directions. Basically, they will be associated to the same pin.
Thus, it is possible to use the whole functionality of PORT driver (especially the API PortSwitchDirection).

The usage of ECU Discrete Signals needs to use the concept of ECU Signals Group. That means for instance; Grp_A is composed only by ECU Discrete Input Signals while Grp_B is composed only by ECU Discrete Output Signals.

The main advantage of this ECU Signals Group concept is that IO Hardware Abstraction can access all members of a signal group through one service call. The call shall ensure data consistency across all members. The number of inputs (Input group) or of outputs (Output group) shall be configurable. All members (inputs or outputs) have to be located on the same port.

7.3.2.3 Diagnosis Class

IoHwAb008: The Diagnosis Class aggregates all ECU Signals, corresponding to Electrical Signals that are used as a diagnosis Input Signal by the ECU Software. An ECU Diagnosis Signal is always coupled to an ECU Output Signal and it indicates the electrical status of this ECU Output Signal. The different states available to this kind of ECU Signal are defined as following:

- No Valid Information available
- Short to Power Supply
- Short to Ground
- Open Load
- Over Temperature (see note below)
- Diagnosis OK

<u>Note about over temperature diagnosis:</u>
The over temperature detection can be realized inside a sensor component.
The over temperature detection is done inside the IO Hardware Abstraction if an external device driver is encapsulated inside the IO Hardware Abstraction.

The state of an ECU Diagnosis Signal is obtained by the electrical failure monitoring of the coupled ECU Output Signal. This failure monitoring is done in this case by software. Strategies to obtain the diagnoses as well as rules to validate them are proposed further in this document.

7.3.2.4 Pulse Width Modulation Class

IoHwAb009: The Pulse Width Modulation Class aggregates all ECU Signals, corresponding to Electrical Signals that are used as a pulse width demodulated Input or modulated ECU Output Signal by the ECU software. An ECU Pulse Width Signal, contrary to the others previously outlined, is mainly characterized by the following two temporal characteristics:

- The Period; it is the time required to complete a full cycle and begin to repeat another one.
- The Duty Cycle; it is the time of the cycle during which the signal is considered as active.

These two characteristics shall be accessible either separately or together consistently. That is why the IO Hardware Abstraction has different classes for PWM signals, allowing a specific behaviour for access operations according to the actual class. These classes and access operations are more detailed further in this document, however we can distinguish three main subclasses, and the following main behaviour for the access operations:

- A PWM sub-class mainly characterized by Period, itself derived in input and output subclasses. The access operations are designed, in this case, to access the Period (for example, an OP_SET(IN IoHwAb_PwxPeriodType Period) operation);

- A PWM sub-class mainly characterized by DutyCycle, itself derived in input and output subclasses. The access operations are designed, in this case, to access the DutyCycle (for example, an OP_SET(IN IoHwAb_PwxDutyCycleType DutyCycle) operation);
- A PWM class characterized by both Period and DutyCycle, itself derived in input and output subclasses. The access operations are designed, in this case, to access both characteristics simultaneously consistently (for example, an OP_SET(IN IoHwAb_PwxPeriodType Period, IN IoHwAb_PwxDutyCycleType DutyCycle) operation).

Remark:
It is up to the implementer of the IO hardware abstraction to configure the ICU module to trigger a demodulation, with either a falling edge, or a raising edge. This is done through the configuration parameter "Default Start Edge".

7.4 Attributes

7.4.1 Background & Rationale

Even the concept of Class allows to gather Signals having a similar electrical behavior, it is not yet sufficient for the users of IO Hardware Abstraction. The chains of acquisition (hardware and software) of each Signal are different and must be detailed because it is the heart of the characterization of Signals.

7.4.2 Requirements about ECU signal attributes

To detail these chains of acquisition, a list of Attributes is defined to identify configurable characteristics of ECU signals.

IoHwAb010: All ECU signals shall have a set of attributes specified and implemented according to the ECU electrical behavior and the software standardization. Some attributes are fixed, such as type, which is determined by the hardware. Other attributes like period and duty cycle, debouncing, may be configurable by the integrator.

7.4.2.1 Signal Data Type Attribute

IoHwAb011: All ECU Signals provided by the IO Hardware Abstraction shall have a Data Type attribute assigned. This attribute shall be configured with one of the available types presented here and defined in chapter 8.2:

Type:	`VoltageType`
Description:	This kind of data type shall be used for ECU Signals of the Analogue Class that depict voltage information to get and / or voltage information to set.

Type:	`CurrentType`

Description:	This kind of data type shall be used for ECU Signals of the Analogue Class that depict current information to get and / or current information to set.

Type:	`ResistanceType`
Description:	This kind of data type shall be used for ECU Signals of the Analogue Class that depict resistance information to get.

Type:	`DiscreteType`
Description:	This kind of data type shall be used for ECU Signals of the Discrete Class that depict logical information to get and / or logical information to set.

Type:	`DiagnosisType`
Description:	This kind of data type shall be used for ECU Signals of the Diagnosis Class that depict the electrical failure state of an Output Signal.

Type:	`PwxPeriodType`
Description:	This kind of data type shall be used for ECU Signals of the “Pulse Width Modulation” Class. That means this type is either used for modulation output or for demodulation input

Type:	`PwxDutyCycleType`
Description:	This kind of data type shall be used for ECU Signals of the “Pulse Width Modulation” Class. That means this type is either used for modulation output or for demodulation input

7.4.2.2 Access Attribute

IoHwAb012: All ECU Signals handled by the IO Hardware Abstraction have only one direction and the two possible directions for it are Input or Output. Therefore a configurable Access Attribute has been defined for this feature.

This attribute is relevant for configuring the connections between SW-Cs and IoHwAb, for it indicates to SW-Cs how the signal will behave in terms of direction. An ECU signal is mapped, like an interface, on IO Hardware Abstraction port. Note that all descriptions concerning Software Component Ports and interfaces are given later in this document.

To access the data of each ECU Signal handled, Software Component above RTE have to use an operation through the port protocol. This access operation is gathered from this Attribute such as introduces here:

Type:	`AccessType`	
Range:	*Input*	*The ECU Signal is an Input and the operation available through the RTE is a Get.*
	Output	*The ECU Signal is an Output and the operation available through the RTE is a Set.*
Description:	This kind of type shall be used for all ECU Signals to define statically the main port operation authorized according of the direction of ECU Signals.	

IoHwAb013: There is only one exception concerning ECU Signals belonging Diagnosis Class that are linked to an ECU Output Signal Port. These interface have

a specific operation to access these Diagnosis Signals. This particularity is more detailed further in this document.

IoHwAb080: the access attribute cannot be changed during runtime. It is up to the implementer to defined two different ECU signals, to get a value from an ECU pin or to set a value on this same ECU pin.

7.4.2.3 BSW-Range Attribute

IoHwAb014: All ECU Signals provided by the IO Hardware Abstraction have a Data Type Attribute defined but the scale of values for this type is not known and it is dependent on each Signal. This is why the BSW-Range attribute is defined and shall be configured for each Signal of the IO Hardware Abstraction.

This Attribute is composed by two parameters which contain the values of the range limits. These values are not only electrical values; they are software boundaries incoming from the Data Type range.

- Min Value : minimum valid value allowed for this Signal
- Max Value : maximum valid value allowed for this Signal

This is the range of the ECU-signal data available just below the RTE. On the contrary of the electrical range on the level of the hardware, BSW range gives the usable data range, just below the RTE.

Examples: they are based on Input Signal from Analogue Class with the `VoltageType` as Data Type Attribute.

- [Min , Max] => [-24, 24]; [0, 42000]; [0, 65535]…

This Attribute influences directly the value of others attributes (described in further chapters) which characterize also the Signals. It shall be configured only once for each Signal.

IoHwAb046: In the case of discrete ECU signals, the BSW-range attribute is defined with the allowed states (enumeration).
The IO Hardware Abstraction realizes the independency between the interface values used by users (Software-Components) and the physical level. For instance, doors could be OPEN/CLOSE and these states are independent of the real hardware input state (0V, 5V, 12V).
The BSW-range of an ECU Signal will be defined during configuration:

- by the Software Component which used the signal above the RTE.
- by all Software Component which use the signal above the RTE. They have to agree on the ECU Signal quality (also for a same BSW-range). The configuration shall fit to this expected quality.

7.4.2.4 Unit Attribute

IoHwAb015: All ECU Signals provided by the IO Hardware Abstraction could take any value of their BSW-Range Attribute but these values shall have one unit defined to be used in the right way by their clients. This is why the Unit attribute is defined and shall be configured for each Signal of the IO Hardware Abstraction.

The values authorized to this Attribute are only electrical units and they shall be independent of the functional related sensor or/and actuator used. The following lists of units for each Class are just examples and are not limited.

Examples: they are based on ECU Input Signal from Analogue Class and Pulse Width Modulation Class with different Data Type Attribute.

- Volts (V), milliVolts (mV), Ampers (A), micro-Ampers (µA), Ohms (Ω), kilo-Ohms (kΩ)…
- Seconds (s), microseconds (µs), milliseconds (ms)…

This Attribute influences directly the value of others attributes (described in this document) which characterize also the Signals. It shall be configured only once for each Signal.
The Unit of an ECU Signal will be defined during configuration:

- by the Software Component which used the signal above the RTE.
- by all Software Component which use the signal above the RTE. They have to agree on the ECU Signal quality (also for a same Unit). The configuration shall fit to this expected quality.

7.4.2.5 BSW-Resolution Attribute

IoHwAb016: This Attribute is gathered from the choice of the BSW-Range, the Unit and the Data Type.

7.4.2.6 BSW-Accuracy Attribute

IoHwAb098: This attribute gathered from the choice of the hardware accuracy parameter, the BSW range, the datatype. How to use this parameter is out of scope of this document.

7.4.2.7 Hardware Resolution Attribute

IoHwAb017: All ECU Signals delivered by the IO Hardware Abstraction have a BSW-Resolution gathered from other configured Attributes. But it will never be better than the resolution of the ECU acquisition chains so-called Hardware Resolution Attribute.

This Attribute gives the resolution of the complete acquisition chain (including microcontroller peripherals) for an ECU Signal. It is only available for information but it is not configurable. It is only applicable to ECU Signals that belong to Analogue Class and to Pulse Width Modulation Class.

The Attribute value should be extracted from an ECU Resource Template file. How to use this Attribute is not in the scope of this document.

7.4.2.8 Hardware Accuracy Attribute

IoHwAb018: All values of Analogue Class ECU Signals delivered by the IO Hardware Abstraction have accuracy depending on software computing but also depending on accuracy of the ECU acquisition chains so-called Hardware Accuracy Attribute.

This Attribute gives the accuracy of the complete acquisition chain (including microcontroller peripherals) for an ECU Signal. It is only available for information but it is not configurable. It is only applicable to Input ECU Signals that belong to Analogue Class.

The Attribute value should be extracted from an ECU Resource Template file. How to use this Attribute is not in the scope of this document.

7.4.2.9 Filtering/Debouncing Attribute

IoHwAb019: All values of ECU Signals delivered by the IO Hardware Abstraction have a different usage that means they are expected with different levels of abstraction. Hence the Filtering/Debouncing Attribute is defined and shall be configured for each Input ECU Signal of the IO Hardware Abstraction. By default, data are provided as “Raw” values.

This Attribute is only available to Input ECU Signals and it allows choosing the level of abstraction of each one. It influences the software strategy of those ECU Signals. This Attribute concerns two kinds of ECU Input Signals therefore it is a combination of names:

- Filtering concerns only Inputs from Analogue Class,
- Debouncing concerns only Inputs from Discrete Class and Diagnosis Class.

Possible values for this Attribute are proposed below just as examples. They are more detailed later in this document concerning parameters and strategies.

Type:	`FilterDebounceType`	
Range:	*RAW_DATA*	*Default configuration. This data has no specific method; the value get is directly delivered.*
	DEBOUNCE_DATA	*Available for Discrete and Diagnosis Classes. This data has a specific method; the value delivered is an average of X values get.*
	WAIT_TIME_DATA	*Available for Analogue, Diagnosis and Discrete Classes. This data has a specific method; the value delivered is get after a delay of X µs.*
Description:	This kind of type shall be used for allowed Input Signals to specify the internal IO Hardware Abstraction software acquisition method.	

7.4.2.10 Failure Monitoring Attribute

IO Hardware Abstraction allows to Software Components above RTE to set ECU Output Signals through Port concept (See chapter 7.5.2.2). But at present setting ECU Output Signals is not sufficient. Those Software Components need diagnoses information to take smart decisions. These diagnoses could directly concern actuators and/or wires connected to the ECU or they could only concern the electronic of the ECU.

The monitoring of actuators and wires connected to the ECU is fully out of scope. IO Hardware Abstraction only takes in account the electronic of the ECU thanks to the Failure Monitoring Attribute. This Attribute allows by configuration an electrical failure monitoring to ECU Output Signals.

IoHwAb020: The Failure Monitoring Attribute is only available to ECU Output Signals and it changes the behavior of Signals so it influences the configuration of connections between SW-Cs and IoHwAb. The value of this Attribute influences indeed directly the Port protocol for the Signal interface. Note that all descriptions concerning Port protocols are given later in this document.

- Failure Monitoring Attribute disabled (0): ECU Output Signal is used as a classic ECU signal through a Client / Server port protocol by a set operation.
- Failure Monitoring Attribute enabled (1): ECU Output Signal has a specific strategy to monitor electrical failures. An Electrical Signal from the Diagnosis Class shall be linked to this ECU Output Signal. This ECU Input Diagnosis Signal shall contain the result of the failure monitoring. The ECU diagnosis signal is available through the Client / Server port protocol of the Output Signal by a diagnosis operation.

Type:	`FailureMonitoringType`
Range:	*`[Disable, Enable]`*
Description:	This kind of type shall be used for ECU Output Signals to specify the behavior and the Port protocol used.

The Failure Monitoring attribute of an ECU Signal will be set during configuration:

- by the Software Component which used the signal above the RTE.
- by all Software Component which use the signal above the RTE. They have to agree on the functionality associated to an ECU Signal (also for the Failure Monitoring attribute). The configuration shall fit to this expected functionality.

7.4.2.11 Age Attribute

All ECU Signals handled by IO Hardware Abstraction depends on the ECU hardware design. This means that the time to set ECU Output Signals and the time to get ECU Input Signals could be different from one to other ECU Signal. So to guarantee a template behavior for all kind of ECU Signals (Input / Output) a common Age Attribute is defined and it shall be configured for each ECU Signal.

IoHwAb021: The Age Attribute has two specific names according to the direction of ECU Signal (Input / Ouput). Anyway, it always contains a maximum time value. Following descriptions explain the meaning of this Attribute for each kind of ECU Signals.

- ECU Input Signals: the specific name of this Attribute is Lifetime. The value defines the maximum allowed age for data of this Signal. If Lifetime is 0, then

the signal is directly get from the physical register. Example: Lifetime = 1000µs the value read is at maximum 1ms older.
- ECU Output Signals: the specific name of this Attribute is Delay. The value defines the maximum allowed time until this Signal is actually set. If Delay is 0, then the signal is immediately set to the physical register. Example: Delay = 100µs the command is set until 100µs elapse.

Type:	`AgeType`
Range:	*`unsigned integer of 16 bits`* *Proposal to be defined with IOHWa Group*
Description:	This kind of type shall be used to define the Age Attribute either Lifetime or Delay for ECU Signals. The time value shall be defined in microseconds (µs).

7.4.2.12 Reporting Feature Attribute

Mainly ECU Signals handled by IO Hardware Abstraction are accessed by Software Components above RTE using the classic Port concept choose (see more explanations in the further chapters of this document). But some Software Components using ECU Input Discrete Signals needs to be informed as soon as possible when the level of these ECU Signals changes. The Reporting Feature Attribute allows by configuration this capability to ECU Signals. This is also the case for some Analog Signals, to report the end of an ADC conversion for instance.

IoHwAb022: The Reporting Feature Attribute is only available to ECU Input Discrete and Analog Signals. This feature changes the behavior of ECU Signals so it influences the configuration of connections between SW-Cs and IoHwAb. The value of this Attribute influences indeed directly the Port protocol for the ECU Signal interface. The RTE is then responsible for transmitting the notification and the signal value from IO Hardware Abstraction to the consumer runnable entity.

Note that all descriptions concerning Port protocols are given further in this document.
- Reporting Feature Attribute disabled (0): ECU Input Discrete Signal is used as a classic signal through a Client / Server port protocol by a get operation.
- Reporting Feature Attribute enabled (1): ECU Input Discrete Signal is used as an event signal through a Sender / Receiver port protocol. Each ECU Signal level change is notified to RTE.

Type:	`ReportingFeatureType`
Range:	*`[Disable, Enable]`* *Proposal to be defined with IOHWa Group*
Description:	This kind of type shall be used for ECU Input Discrete Signals and ECU Input Analog Signals to specify the behavior and the Port protocol used.

Each ECU Signal of IO Hardware Abstraction must have only one Software Component user above the RTE with one dedicated Port protocol.

7.4.2.13 Pulse Test Attribute

Thanks to Failure Monitoring Attribute (previously described in this document), IO Hardware Abstraction allows detecting some electrical defaults but not all. In fact, there are cases where Software Component above the RTE knows better when to monitor Signals. This is why Pulse Test Attribute is defined to authorize the generation of a pulse test command when is required.

IoHwAb023: The Pulse Test Attribute is only available to ECU Output Signals and it allows by configuration an electrical test pulse command to ECU Signals so it influences the configuration of connections between SW-Cs and IoHwAb. The value of this Attribute influences indeed directly the Port protocol for the Signal interface. Note that all descriptions concerning Port protocols are given further in this document.

- Pulse Test Attribute disabled (0): For this ECU Output Signal, test pulse operation is not available through the Client / Server port protocol. In case of Failure Monitoring available, it is done while using the ECU Output Signal by a set operation.
- Pulse Test Attribute enabled (1): For this ECU Output Signal, test pulse operation is available through the Client / Server port protocol. When is required, IO Hardware Abstraction generates an electrical command and Failure Monitoring is done during this dedicated pulse. Characteristics of this pulse (level and duration) are described further in this document.

Type:	`PulseTestType`
Range:	*`[Disable, Enable]`*
Description:	This kind of type shall be used for ECU Output Signals to specify the capability to generate a test pulse command and the Port protocol used.

The Pulse test attribute of an ECU Signal will be set during configuration:

- by the Software Component which used the signal above the RTE.
- by all Software Component which use the signal above the RTE. They have to agree on the functionality associated to an ECU Signal (also for the Pulse Test property). The configuration shall fit to this expected functionality.

7.4.2.14 Wakeup Attribute

The ECU software architecture has defined the ECU State Manager module as responsible for managing ECU wakeups.

IoHwAb047: The Wakeup Attribute is only available to ECU Input Signals and it allows by configuration a callout function corresponding to this wakeup capability. The value of this Attribute does not influence directly the Port protocol for the Signal interface but only the number of runnable entities available in IO Hardware Abstraction. Note that all descriptions concerning runnable entities are given further in this document.

- Wakeup Attribute disabled (0): For this ECU Input Signal, there is not Wakeup capability expected.

- Wakeup Attribute enabled (1): For this ECU Input Signal, a Wakeup capability is expected that means the IO Hardware Abstraction shall provide a mechanism to inform the appropriated Software Component. Note that the reporting of wakeup on this input can be reported to the Software Component only if the ECU State manager has validated the wakeup reason.

Type:	`WakeupType`
Range:	`[Disable, Enable]`
Description:	This kind of type shall be used for ECU Input Signals to specify a triggering capability.

7.4.3 Overview of Attributes to qualify Signals

IoHwAb024: The following table summarizes the applicability of all previous defined Attributes to the possible IO Hardware Abstraction Signals (Inputs and Outputs).

- **X** means the Attribute is applicable to this Class of ECU Signal and shall be configured.
  - **Xl** means the Age Attribute applicable is the Lifetime.
  - **Xd** means the Age Attribute applicable is the Delay.
- **F** means the Attribute is applicable to this Class of ECU Signal but it is a fixed standard value.
- **O** means the Attribute is optional to this Class of ECU Signal and depends on a static configuration (disable/enable).
- **-** means the Attribute is not applicable or has no meaning to this Class of ECU Signal.

Signal \ **Attributes**	**Signal Data Type**	**Access**	**BSW-Range**	**Unit**	**BSW-Resolution**	**Failure Monitoring**	**Age (Lifetime/Delay)**	**Filtering / Debouncing**	**Sampling Rate**	**Report Feature**	**Pulse Test**	**Wakeup**
Analogue$_{in}$	X	X	X	X	X	-	Xl	X	X	O	-	-
Analogue$_{out}$	X	X	X	X	X	O	Xd	-	-	-	O	-
Discrete$_{in}$	X	X	F	-	-	-	Xl	X	X	O	-	O
Discrete$_{Status}$	X	-	F	-	-	-	Xl	X	-	-	-	O
Discrete$_{pow}$	X	X	F	-	-	O	Xd	-	-	-	O	-
PWx Period$_{in}$	X	X	X	X	X	-	Xl	-	-	-	-	O
PWx Period$_{out}$	X	X	X	X	X	O	Xd	-	-	-	O	-
PWx Duty Cycle$_{in}$	X	X	F	F	X	-	Xl	-	-	-	-	O
PWx Duty Cycle$_{out}$	X	X	F	F	X	O	Xd	-	-	-	O	-

Discrete powered outputs can have diagnosis.

7.5 IO Hardware Abstraction and Software Component Template

Note about this chapter: This chapter refers to document [8].
Changes inside this document may influence the content of this chapter.

7.5.1 Background & Rationale

The following picture is a part of the MetaModel and makes the progress from the Software Component template to the implementation easier to understand.

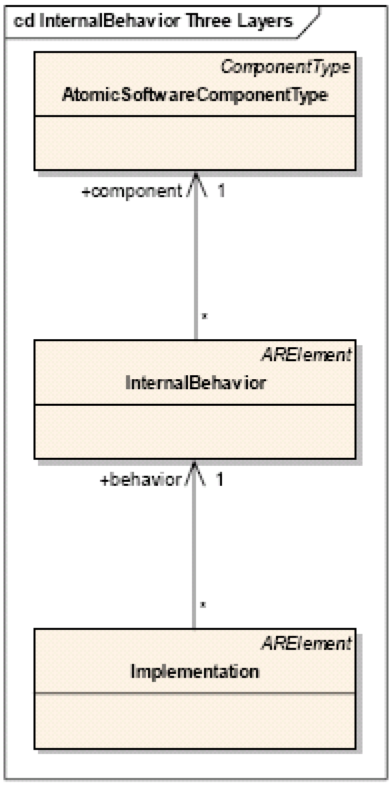

Figure 7.3: From the Software Component template to the implementation

This approach allows defining the standardization deepness. As explained previously, the implementation is ECU-firmware. Therefore, this chapter only summarizes how to define the IO Hardware Abstraction as a Software Component (SW-C), and gives a short overview of the internal behavior. The internal behavior description mainly covers BSW scheduling mechanisms.

7.5.2 Requirements about the usage of Software Component template

IoHwAb001: The IO Hardware Abstraction is based upon the Software Component Template as specified in document [8].
In the same manner than any other Software Component, the I/O Hardware Abstraction might by sub structured, depending on the complexity of an ECU.
Indeed, the IO Hardware Abstraction is a classical Component Prototype, that can be atomic or composed and that provides and requires interfaces. Moreover, I/O Hardware Abstraction may only interact by means of their PortPrototypes. Hidden dependencies that are not expressed by means of PortPrototypes are not allowed.

However, the I/O Hardware Abstraction interfaces on one side the MCAL drivers via Standardized Interfaces and on the other side the RTE. Hence, IO Hardware Abstraction shall respect the virtual port concepts.

IoHwAb025: An “EcuAbstractionComponentType” is a class that describes the IO Hardware Abstraction. This class shall be instantiated one or many time for each ECU. In this case the I/O Hardware Abstraction Layer is described by several different EcuAbstractionComponentTypes on M1.

An instantiation of “EcuAbstractionComponentType” provides a set of ports (). During RTE Generation, only those that are connected with Software Components would be taking into account.

This chapter gives an overview of virtual port concepts and runnable entities applied to the IO Hardware Abstraction needs. The following chapters of this document describe more in detail points set out here.

7.5.2.1 Ports concept and IO Hardware Abstraction

This is an overview of recommendations for defining Ports of IO Hardware Abstraction using Software Component template. Mainly, a port only exists if one ECU Signal[1] (at least) is allocated to it.
IO Hardware Abstraction has almost only provide-ports (P-port).

Almost all Ports of IO Hardware Abstraction have Client/Server interfaces. These ports have several operation prototypes available, configured on the basis of ECU Signals Attributes[2] configuration. Only ports dedicated to send notifications have Sender/Receiver interfaces. That means only event semantics is allowed for data elements of this interface in IO Hardware Abstraction.

- Further chapters in this document go deeper in usage of ports for IO Hardware Abstraction. But, it is advised to read the Software Component Template document [8] to be aware of all terms and all concepts used.

7.5.2.2 Software Component and Runnable concept

Software Components have functions to realize their strategies and internal behaviors. These are partly described using runnable entities. The former is contained in runnables and the latter depends of runnables design. Runnable entities are provided by the Atomic Software Component and are (at least indirectly) a subject for scheduling by the underlying operating system.

An implementation of an atomic Software Component has to provide an entry-point to code for each Runnable in its "InternalBehavior". For more information, please refer to the specification [8].

[1] ECU Signal as defined by IO Hardware Abstraction in this document in chapter [7.2]
[2] ECU Signal Attributes as defined by IO Hardware Abstraction in this document in chapter [7.4]

The runnable entities are the smallest code-fragments which can be activated independently. They are provided by the Atomic Software Component and are activated by the RTE. Runnables are for instance set up to respond to data exchange or operation invocation on a server.

The runnable entities have three possible states: Suspended, Enabled and Running. During run-time, each runnable of an atomic Software Component is (by being a member of an OS task) in one of these states.

For a sight of available choices and attributes to define each runnables of the Atomic Software Component, please refer to specification [8].

7.6 Scheduling concept for IO Hardware Abstraction

7.6.1 Background & Rationale

The IO Hardware Abstraction may consist of several BSW modules (e.g. onboard device driver).

Each of these BSW modules can provide BSW runnable entities (also called BswModuleEntity in the BSW Scheduler Specification (see [19]).

To make a parallel, a BswModuleEntity is the equivalent of SW-C runnable entities, for which the AUTOSAR glossary [5] gives the following definition: „"A Runnable Entity is a part of an Atomic Software-Component (→ definition) which can be executed and scheduled independently from the other Runnable Entities of this Atomic Software-Component".

In the case of SW-C runnables entities, these ones are called in AUTOSAR OS Tasks bodies. Runnables are given in the SW-C description. Activation of SW-C runnables strongly depends on RTE events.

However, BSW runnables are not activated by the RTE, but by the BSW Scheduler. To achieve this, IO Hardware Abstraction must be interfaced with BSW Scheduler, which activates them. In the same way than SW-C are most often activated by RTEEvents, the schedulables BswModuleEntities can be activated by BswEvent. There is also a kind of BswModuleEntity which can be activated in interrupt context. This leads to two sub-classes: BswSchedulableEntity and BswInterruptEntity.

The BSW Scheduler specification (document [19]) covers the requirement of BSW modules to schedule recurrently or sporadically BSW modules entities.

7.6.2 Requirements about IO Hardware Abstraction Scheduling concept

7.6.2.1 Operations for interfaces provided by Ports

IoHwAb031: The IO Hardware Abstraction, described from the interfaces point of view, shall define runnable entities to exchange data with Software Components through the RTE. The number of those runnables that are provided by each Port depends on ECU Signal attributes values.

The next Operations could exist or not depending on value of the attribute that is associated to them. Those Operations are part of the "PortInterface" configuration of either *require*- or *provide*-ports.

IoHwAb068: The implementation of the service behind these operations is ECU specific and the mapping to the corresponding "PortInterface" shall be documented in the Software Component description.

7.6.2.1.1 Gets operation, OP_GET

IoHwAb069: OP_GET depends on "Access Attribute" (§7.4.2.2). Considering an ECU Signal associated to a Port. This Port shall provide an OP_GET if this ECU Signal "Access Attribute" is configured as "Input".

–

IoHwAb113: In the case where the ECU Signal associated to this port is a Discrete Signal, the port shall provide an OP_GET anyway.

Otherwise, if this ECU Signal "Access Attribute" is not configured as "Input", the OP_GET is not required.

7.6.2.1.2 Sets operation, OP_SET

IoHwAb070: OP_SET depends on "Access Attribute" (§7.4.2.2). Considering an ECU Signal associated to a Port. This Port shall provide an OP_SET if this ECU Signal "Access Attribute" is configured as "Output".

–

IoHwAb114: In the case where the ECU Signal associated to this port is a Discrete Signal, the port shall provide an OP_SET anyway.

Otherwise, if this ECU Signal "Access Attribute" is not configured as "Output", the OP_SET is not required.

7.6.2.1.3 Diagnosis operation, OP_DIAG

IoHwAb071: OP_DIAG depends on "Failure Monitoring Attribute" (§7.4.2.10). Considering an ECU Signal associated to a Port. This Port shall provide an OP_DIAG if this ECU Signal "Failure Monitoring Attribute" is configured as "Enable".

Otherwise, if this ECU Signal “Failure Monitoring Attribute” is configured as “Disable”, the OP_DIAG is not required.

7.6.2.1.4 Test Pulse operation, OP_TEST

IoHwAb072: OP_TEST depends on “Pulse Test Attribute” (§7.4.2.13). Considering an ECU Signal associated to a Port. This Port shall provide an OP_TEST if this ECU Signal “Pulse Test Attribute” is configured as “Enable”.

Otherwise, if this ECU Signal “Pulse Test Attribute” is configured as “Disable”, the OP_TEST is not required.

7.6.2.1.5 Reporting operation, OP_REPORT

IoHwAb073: OP_REPORT depends on “Reporting Feature Attribute” (§7.4.2.12). Considering an ECU Signal associated to a Port. This Port shall provide an OP_REPORT if this ECU Signal “Reporting Feature Attribute” is configured as “Enable”.

Otherwise, if this ECU Signal “Reporting Feature Attribute” is configured as “Disable”, the OP_REPORT is not required.

7.6.2.2 Notification and/or Callback

IoHwAb032: The IO Hardware Abstraction shall define BswInterruptEntities (a sub-class class of BswModuleEntity by opposition to BswSchedulableEntity) to fulfill notification and/or callback mechanisms to exchange data with other modules below the RTE within an interrupt context.

The IO Hardware Abstraction may contain one or several callback functions. The available callback functions need to be hooked up to the notification interfaces of the MCAL drivers. Therefore, they have to respect the prototype definition of the MCAL drivers (no passing parameter, no return parameter).

IoHwAb033: The implementation has to take into consideration, that the callback functions will be executed in interrupt context.
Callback function can additionally provide the capability to trigger Software Components outside of the IO Hardware Abstraction. These notifications need to be handled through the RTE (sender port).

IoHwAb034: The number of available callback functions and the order of execution will be implementation dependent and must be documented in the IO Hardware Abstraction description.

7.6.2.3 Main function / job processing function

IoHwAb035: The IO Hardware Abstraction may contain one or several job processing functions that are BswSchedulableEntities (a sub-class class of

BswModuleEntity by opposition to BswInterruptEntity, e.g. one for each device driver). They shall be activated according to their use.
They will be time-triggered by the BSW Scheduler. They could be synchronized to the execution of the other runnable entities.
The number of BswSchedulableEntities and their order of execution will be implementation dependent and must be documented in the IO Hardware Abstraction description.

7.6.2.4 Initialization, Deinitialization and/or Callout

IoHwAb036: The IO Hardware Abstraction shall define BswModuleEntries to exchange data with other Software Component below the RTE outside an interrupt context, for example in case of BSW initialization/deinitialization.

These BswModuleEntries are linked to a dedicated BswModuleEntity, which will be called to perform the service / exchange the data.

The IO Hardware Abstraction may contain one or several initialization and deinitialization functions (e.g. one for each device driver). Similar to the MCAL drivers the initialization functions shall contain a parameter to be able to pass different configurations to the device drivers. This function shall initialize all local and global variables used by the IO Hardware Abstraction driver to an initial state.

IoHwAb037: The initialization/deinitialization functions shall be exclusively used/ handled by the ECU State Manager. For more information, refer to [10].
The number of available functions and the order of execution will be implementation dependent and must be documented in the IO Hardware Abstraction description.

7.6.2.4.1 Wakeup specific Callout

IoHwAb074: Wakeup services depend on “Wakeup Attribute” (§7.4.2.14). Considering an ECU Signal, IO Hardware Abstraction shall provide a specific wakeup callout service if this ECU Signal “Wakeup Attribute” is configured as “Enable”. These callouts are called either by EcuM or by driver, depending on the wakeup mode (interrupt or polling, see chapter 9 sequences), and the EcuM handles after that the wakeup validation.

Otherwise, if this ECU Signal “Wakeup Attribute” is not configured as “Enable”, this specific wakeup callout service is not required.

An overview of what is a BswModuleEntity can be found in the document [19].

7.6.2.5 IO Hardware Abstraction scheduling examples

7.6.2.5.1 Interface provided by ADC and IO Hardware Abstraction

The following example shows a scheduling example for an ADC conversion.
The IO Hardware Abstraction shall provide two P-ports.

The Software Component interface in this example is af_pressure.

The ECU state manager is able to trigger a BswModuleEntry for initialization of the ADC driver (Call of Adc_Init() with the Adc_ConfigType structure).

Use Case: The software component needs the af_pressure value.
1 – RTE triggers the OP_GET operation of the dedicated P-Port.
2 – R1 is a runnable entity and it allows to call the appropriated ADC driver services
ADC_EnableNotification
ADC_StartGroupConversion
3 – At the end of conversion, the ADC triggers the BswModuleEntry R2, within interrupt context. This is possible since the notification is allowed for this interface.
The ADC_NotificationGroup() function is specified in the ADC driver
4 – The notification is then "sent" to the Software Component via a RTEevent.

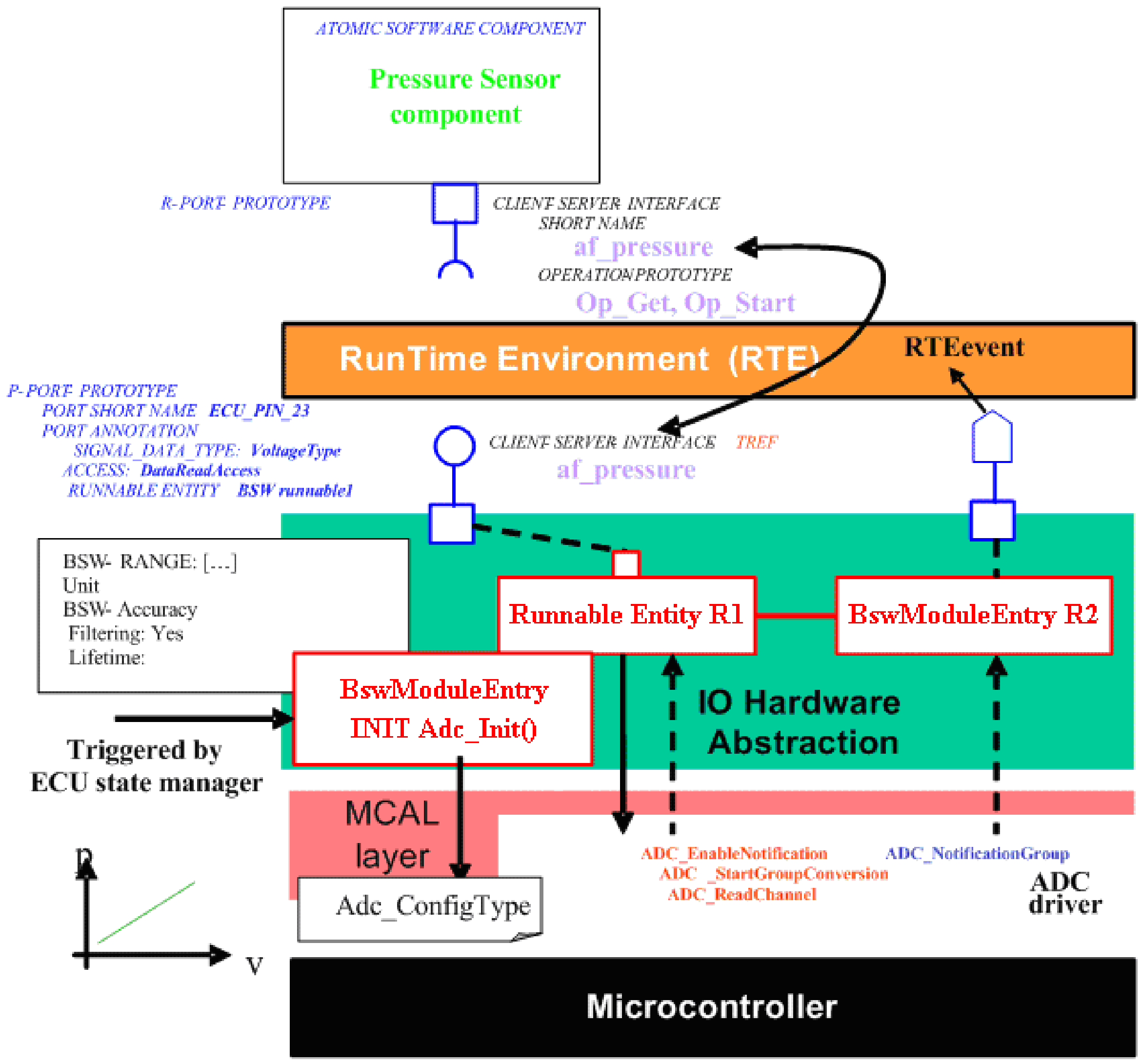

Figure 7.4: Example of IoHwAb runnables

The sequence diagram of this example is in chapter 9

7.6.2.5.2 Synchronous scheduling with Runnable Entities and BswSchedulableEntities

The following example shows a scheduling example for setting a Lamp linked to a SMART power.
The SMART power is connected to the microcontroller by SPI bus. Hence, the dedicated piece of code uses the SPI Handler/Driver.

The FrontLeftLamp value to be set by the RTE is in an IO Hardware Abstraction buffer.
An output line to another SMART power is set synchronously to trigger an ADC conversion of the same electrical signal by the ADC driver.

At the end of conversion, the converted result is available and the notification is set to the Analog input manager to store the value inside a buffer, available for diagnosis purpose.

In this example, the periodical treatment is realized by a BswSchedulableEntity.

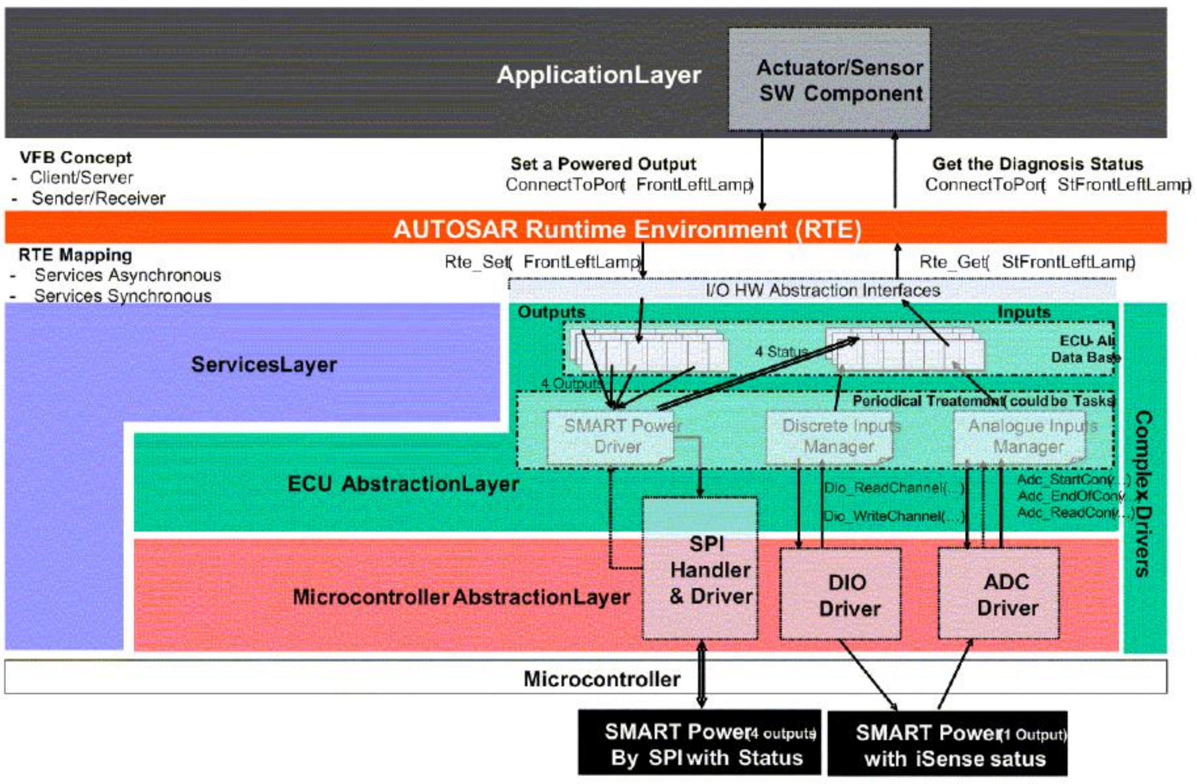

Figure 7.5: Example of IoHwAb runnable – cyclic setting of output and diagnosis

7.7 Other requirements

IoHwAb066: The configuration parameters shall be checked statically (at the latest during compile time) for correctness. The version information in the header and source files shall be validated and consistent (e.g. by comparing the version information in the header and source files with a pre-processor macro).

7.8 Error classification

Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.

IoHwAb067: Development error values are of type uint8.

Type or error	***Relevance***	***Related error code***	***Value [hex]***
Up to the implementer to define error he wants to report	Development	`Up to the implementer`	`0x01`
Up to the implementer to define error he wants to report	Production	`Up to the implementer`	`Assigned by DEM`

7.9 Error detection

IoHwAb053: The detection of development errors is configurable (*STD_ON* / *STD_OFF*) at pre-compile time.
The switch *`IoHwAbDevErrorDetect`* (see chapter 10) shall activate or deactivate the detection of all development errors.

IoHwAb054: If the *`IoHwAbDevErrorDetect`* switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.7 and chapter 8.

IoHwAb055: The detection of production code errors cannot be switched off.

7.10 Error notification

IoHwAb051: Detected development errors will be reported to the error hook of the Development Error Tracer (DET) if the pre-processor switch *`IoHwAbDevErrorDetect`* is set (see chapter 10).

IoHwAb052: Production errors shall be reported to Diagnostic Event Manager.

7.11 IO Hardware Abstraction layer description

7.11.1 Background & Rationale

The IO Hardware Abstraction layer has some analogies with a Software Component, especially regarding port definition for communication through the RTE.
Main difference is that IO Hardware Abstraction is below the RTE, within the ECU Abstraction Layer, and is unique within an ECU. IO Hardware Abstraction makes a kind of interface between Basic Software modules and Application Software.

For the IO Hardware Abstraction, but also for Services, the current methodology requires to fill out two different templates. For example, in order to integrate an NVRAM Manager on an AUTOSAR ECU one would use the BSWMDT to document its needs for the BSW Scheduler, OS Resources and so on. In addition, one would use the SWCT to describe the ports towards the RTE.

The IO Hardware Abstraction is a part of BSW. It could be considered as a group of modules. Although IOHWAB is ECU-firmware, each module of IOHWAB could fit to the BSWDT. Today, it is known that this point is not sufficiently documented in the current specification.

However, it is agreed that ECU Signal will be mapped to a VFB Port (See chapter 7.2 and chapter 7.5). Moreover, to describe the interfaces between an IO Hardware Abstraction implementation and applicative Software Components implementations (above RTE), one shall use the Software Component Template.

The intention of this chapter is to summarize all recommendations to define Ports, Interfaces and all other Software Component like elements during configuration process.

7.11.2 Requirements

7.11.2.1 IO Hardware Abstraction Ports definition

IoHwAb075: The IO Hardware Abstraction specification defines only recommendations for the Port usage. Their instantiation shall be done during the configuration process and is specific to the ECU electronic design.

The IO Hardware Abstraction proposes to create one Port for each ECU Signal identified, exception made for ECU Diagnosis Signals that are connected to ECU Output Signals. A relationship between this ECU Signal and the Port shall be created.

Example:
The ECU has 10 Analog input pins, 15 PWM output pins, 15 Digital output pins.
The IO Hardware Abstraction defines at least one Port for each ECU Signal. In this simple example, Ports are instantiated 40 times.

7.12 Examples

7.12.1 EXAMPLE 1: Use case of on-board hardware

This example is derived from a power supplier ECU.

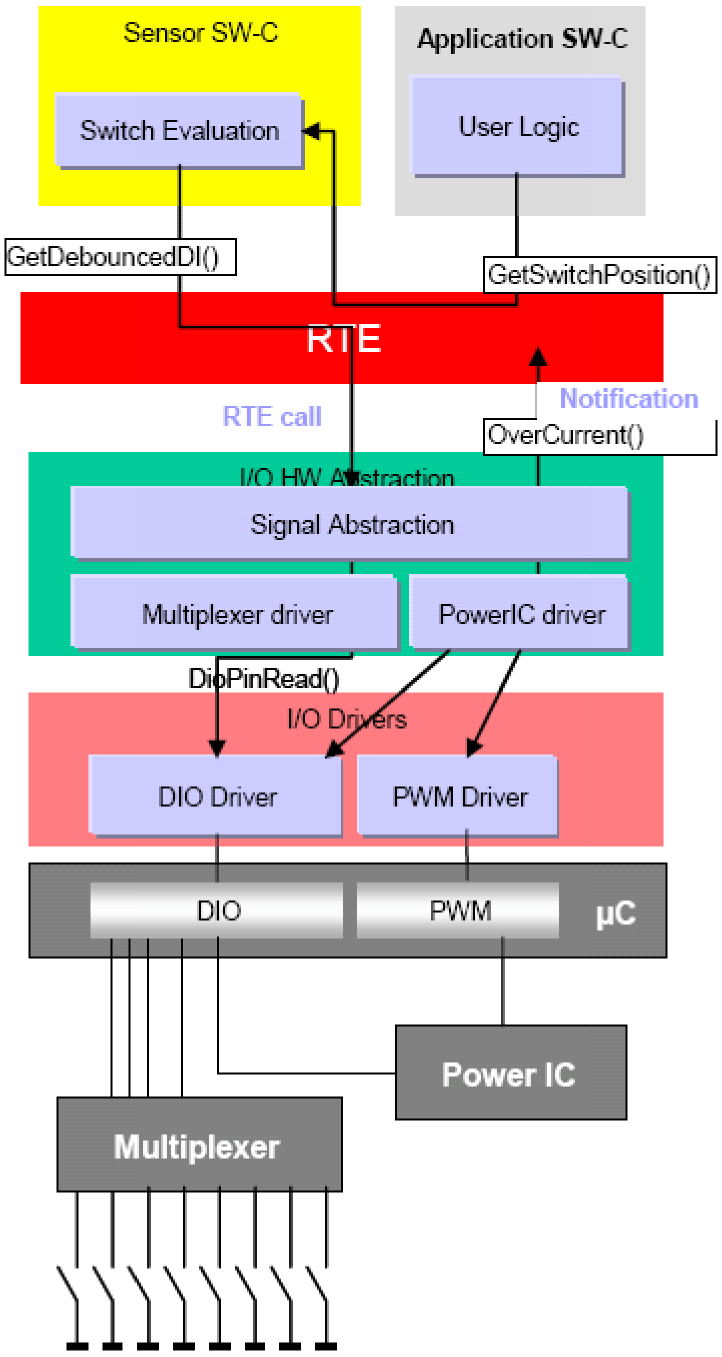

Figure 7.6: Use case of on-board hardware

- The ECU has a high number of Digital Inputs (DI).
- One main group is the "**slow DI's**" for mechanical switches
- The second main group is the "**fast DI's**" for the diagnosis of the Power IC (this DI indicates that the output current is to high "over current", these DI's are not led out of the ECU)
- The MCU has not enough PIN's -> the slow DI's are connected to 8 bit multiplexers (**3 address lines and 1 data line** for each multiplexer)
- the maximum time between the occurrence of an "over current" and the switch of the Power IC is 1 ms
- One OEM requirement is that the reaction of a switch must be not later than **100 ms**
- One other OEM requirement is that each DI must be debounced by **3 of 5 voting.** However the practise shows that the kind of debouncing is not really important because the mechanical switches and the power IC do not generate disturbing signals

The **solution** today is that
all DI (slow and fast) are read **every 0,8 ms (cyclic task)** (The scan rate for the slow DI could be lower but the overhead for an additional task is higher than the runtime savings)

- The debouncing for the slow DI's is **1 time in every loop** (so the worst cast delay to the debounced value is 3,2 ms)
- If an overcurrent is detected the pin will read again several times but in the same loop and the power IC will switched off immediately
- The application runs **every 10 ms** and reads the debounced DI for the switches and the diagnosis information's

Decomposition on the AUTOSAR architecture:

Layer	***Multiplexed IO***	***Power IC***
Application	Runnable reads the data every 10 ms	gets a notification if the power IC detects overcurrent.
RTE	Handles runnables	
IO Hardware Abstraction	8 signal mapped on ports, definition of port feature and Client/Server interface Signal abstraction gives the debounce time (better than a debounce voting rule) A cyclic task performs a reading of input via DIO service call	IO Hardware Abstraction makes decision to switch off the Power IC if an overcurrent is detected (in the driver of the external ASIC) a cyclic task performs a reading of input via DIO service call.
MCAL driver	<u>*DIO driver*</u>: adress lines, 1 data line	<u>*DIO driver*</u>: 1 feedback line from power IC <u>*PWM driver*</u>: 1 line to the power IC
ECU hardware	<u>*Multiplexer*</u>: Mapping of 8 electrical signal	<u>*Power IC*</u>: Controls the power supply of the multiplexer

7.12.2 EXAMPLE 2: Use case of failure monitoring managed by SPI

In this example, an diagnostic output signal shall be defined with the diagnosis attribute on the level of the IO Hardware Abstraction.
Therefore, an input is used to perform the diagnosis of the output.

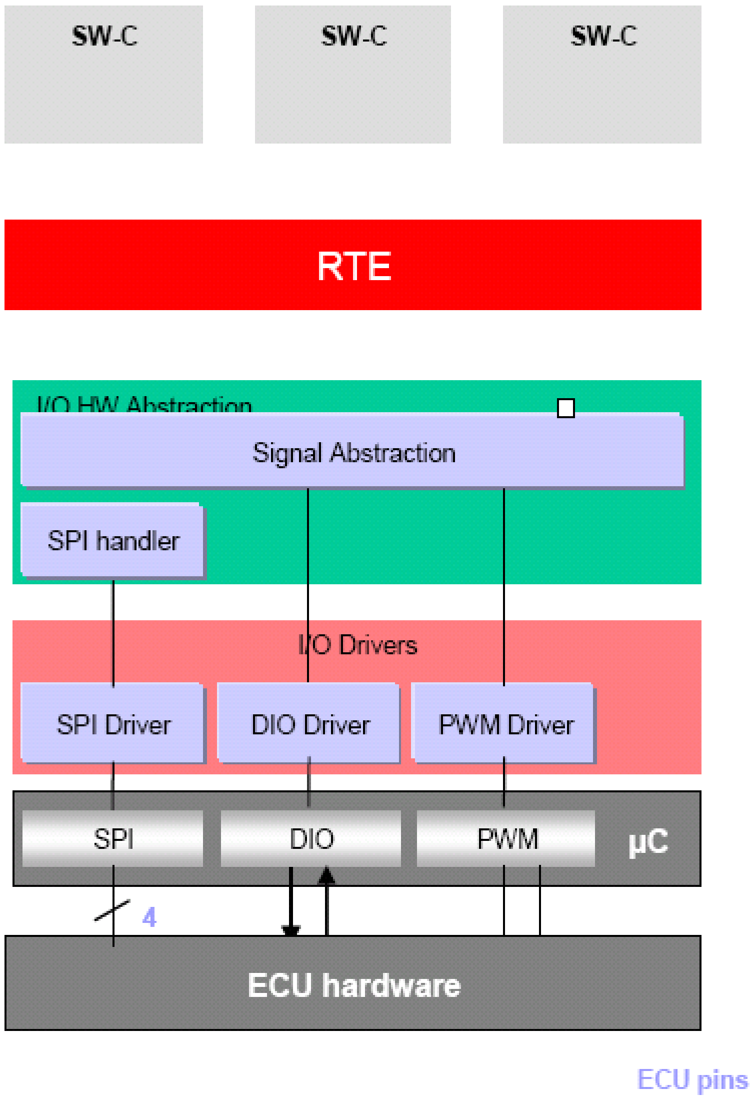

Figure 7.7: Use case of failure monitoring managed by SPI

When the IO Hardware Abstraction asks for positioning one output (Dio_WriteChannel), a readout of the channel is done via a ECU pin configured as input.

The ICU driver sends a notification to the IO Hardware Abstraction.
The protection strategy is located in the ECU firmware.

Software Component can get the diagnosis value through the port using the diagnosis operation.

Example: Short-circuit to ground
IO Hardware Abstraction detects 0x00 (short-circuit to ground)
It is done via a periodically check of outputs, and storing of results
ECU firmware switchs an internal variable to failure status.
ECU firmware makes the decision to protect the output. The IO Hardware Abstraction provides at the moment the diagnostic information to the Software Component.

7.12.3 EXAMPLE 5: Output power stage

The ECU hardware has a power stage ASIC.
Therefore, all ECU pins shall be available as “signals” at the level on the IO Hardware Abstraction, just below the RTE.

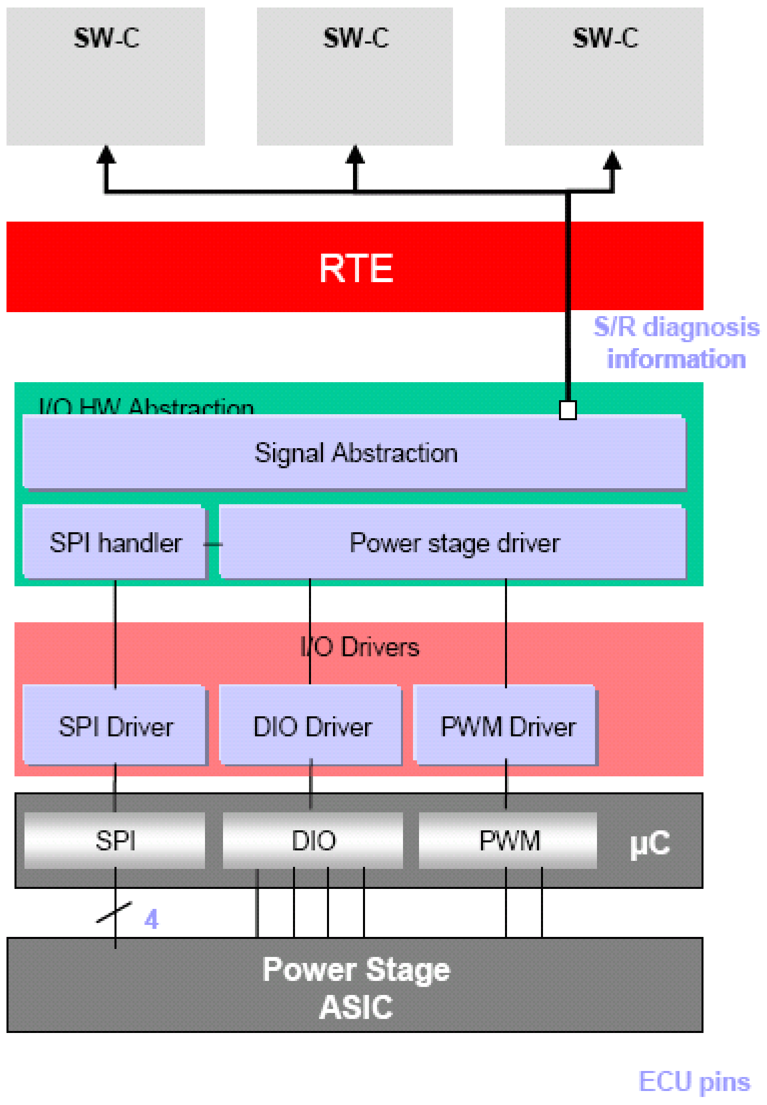

Figure 7.8: Use case of output power stage

Some outputs are controlled via the SPI driver/handler.

Some inputs are directly controlled via the DIO driver.
Some voltages, frequencies are set via the PWM driver.

A **power stage driver** provides the view of all outputs. It calls services of PWM, DIO drivers and SPI handler. The signal abstraction makes all these outputs “visible” from the point of view of Software Component (signals are mapped on Ports).

- The “Power stage driver” can be configurable.

Diagnosis:

- Every failure can be detected on the level of the power stage. The diagnosis data flow goes through the SPI communication to the Power stage driver
- Then, the diagnosis is provided to all Software Component via a S/R interface.
- The diagnosis information can also be sent to the DEM

8 API specification

8.1 Imported types

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

IoHwAb118:

Module	*Imported Type*
Adc	Adc_GroupType
	Adc_StatusType
	Adc_StreamNumSampleType
	Adc_ValueGroupType
	Adc_ConfigType
Dem	Dem_EventIdType
Dio	Dio_ChannelType
	Dio_LevelType
	Dio_PortLevelType
	Dio_PortType
	Dio_ChannelGroupType
EcuM	EcuM_WakeupSourceType
Gpt	Gpt_ChannelType
	Gpt_ModeType
	Gpt_ValueType
Icu	Icu_ActivationType
	Icu_ChannelType
	Icu_DutyCycleType
	Icu_EdgeNumberType
	Icu_IndexType
	Icu_InputStateType
	Icu_ValueType
	Icu_ConfigType
Port	Port_PinDirectionType
	Port_PinModeType
	Port_PinType
	Port_ConfigType
Pwm	Pwm_ChannelType
	Pwm_EdgeNotificationType
	Pwm_OutputStateType
	Pwm_PeriodType
	Pwm_ConfigType
Spi	Spi_AsyncModeType
	Spi_ChannelType
	Spi_DataType
	Spi_HWUnitType
	Spi_JobResultType
	Spi_JobType
	Spi_NumberOfDataType
	Spi_SeqResultType
	Spi_SequenceType
	Spi_StatusType
	Spi_ConfigType

Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

IoHwAb065: Following types shall be defined for the IO Hardware Abstraction implementation.

8.2.1 IoHwAb_ConfigType

Name:	`IoHwAb_ConfigType`	
Type:	`Structure`	
Range:	`none`	The contents of the initialization data structure are specific.
Description:	This is the type of the external data structure containing the initialization data for the IO Hardware Abstraction driver.	

8.2.2 IoHwAb_SignalType

Name:	`IoHwAb_SignalType`		
Type:	`uint16, uint32`		
Range:	`16...32 bits`	--	-
Description:	This is the type of the external data structure containing the initialization data for the signal handled by the IO Hardware Abstraction		

8.2.3 IoHwAb_DiscreteGroupType

Name:	`IoHwAb_DiscreteGroupType`		
Type:	`uint8`		
Range:	`0...255`	--	This type is used for Read / Write operation on a group of discrete signals. This type shall be independent of the target platform.
Description:	This is the type used for handling the signal value read on a group of discrete inputs, and to write the signal value on a group of discrete outputs.		

8.2.4 IoHwAb_SignalDiagnosisType

Name:	`IoHwAb_SignalDiagnosisType`		
Type:	`uint8`		
Range:	`0bxx1x xx00`	--	Over Temperature
	`0bxxx1 xx00`	--	Open Load
	`0bxxxx 1x00`	--	Short to the ground
	`0bxxxx x100`	--	Short to the Power Supply
	`0bxxxx xx10`	--	No valid information available
	`0b0000 0001`	--	Diagnosis not supported (could be a static check)

	`0b0000 0000`	--	Diagnosis OK
	–	--	This type is a bit field. Several errors could be present at the same time. Following mapping can be used.
Description:	This is the type used for handling diagnosis information.		

8.2.5 IoHwAb_VoltageType

Name:	`IoHwAb_VoltageType`		
Type:	`uint16, uint32`		
Range:	`16...32 bit`	--	Shall cover all available voltage range The best type should be chosen for the specific MCU platform (best performance).
Description:	This is a type of the variable used to store the value of a voltage read on an analogue input		

8.2.6 IoHwAb_CurrentType

Name:	`IoHwAb_CurrentType`		
Type:	`uint16, uint32`		
Range:	`16...32 bit`	--	Shall cover all available current range The best type should be chosen for the specific MCU platform (best performance).
Description:	This is a type of the variable used to store the value of a current read on a input		

8.2.7 IoHwAb_ResistanceType

Name:	`IoHwAb_ResistanceType`		
Type:	`uint16, uint32`		
Range:	`16...32 bit`	--	Shall cover all available resistance range The best type should be chosen for the specific MCU platform (best performance).
Description:	This is a type of the variable used to store the value of a resistance read on a input		

8.2.8 IoHwAb_PwxPeriodType

Name:	`IoHwAb_PwxPeriodType`		
Type:	`uint16, uint32`		
Range:	`16...32 bit`	`--`	--
Description:	This kind of data type shall be used for ECU Signals of the “Pulse Width Modulation” Class. That means this type is either used for modulation output or for demodulation input		

8.2.9 IoHwAb_PwxDutyCycleType

Name:	`IoHwAb_PwxDutyCycleType`		
Type:	`uint16, uint32`		
Range:	`16...32 bit`	--	--
Description:	This kind of data type shall be used for ECU Signals of the “Pulse Width Modulation” Class. That means this type is either used for modulation output or for demodulation input		

8.3 Function definitions

This is a list of functions provided for upper layer modules.

NOTE FOR IO HARDWARE ABSTRACTION:

As explained in the previous chapters, no functional API will be specified for the IO Hardware Abstraction.

8.3.1 IoHwAb_Init<_Init_Id>

IoHwAb119:

Service name:	IoHwAb_Init<Init_Id>

Syntax:	`void IoHwAb_Init<Init_Id>(`
	
	`)`

Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Initializes either all the IO Hardware Abstraction software or is a part of the IO Hardware Abstraction.

IoHwAb059: This kind of function initializes either all the IO Hardware Abstraction software, or a part of the IO Hardware Abstraction.

IoHwAb060: The multiplicity of IO devices managed by the IO Hardware Abstraction software is handled via several init functions. Each init function is tagged with a <_Init_ID>. Therefore, an external device, having its driver encapsulated inside the IO Hardware Abstraction, can be separately initialized.

IoHwAb061: This kind of init function is called by the ECU State Manager. The ECU integrator is able to configure the init sequence order called by the ECU State manager

IoHwAb102: After having finished the module initialization, the IO Hardware Abstraction state shall be set to `IOHWAB_IDLE`, the job result shall be set to `IOHWAB_JOB_OK`.

8.3.2 IoHwAb_GetVersionInfo

IoHwAb120:

Service name:	IoHwAb_GetVersionInfo	
Syntax:	`void IoHwAb_GetVersionInfo(` `Std_VersionInfoType* versioninfo` `)`	
Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this implementation of IO Hardware Abstraction.
Return value:	None	
Description:	Returns the version information of this module.	

IoHwAb057: This service returns the version information of this implementation of IO Hardware Abstraction. The version information includes:
- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

IoHwAb058: This function shall be pre compile time configurable `On/Off` by the configuration parameter: `IoHwAbVersionInfoApi`

Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the header file.

Configuration of IoHwAb_GetVersionInfo: <Description of statically configurable attributes that affect this API call. Reference to configuration parameters described in chapter 10>

8.4 Call-back notifications

This is a list of functions provided for lower layer modules. The function prototypes of the callback functions shall be provided in the file `IoHwAb_Cbk.h`

8.4.1 IoHwAb_Adc_Notification

IoHwAb121:

Service name:	IoHwAb_Adc_Notification_<#group_ID>
Syntax:	`void IoHwAb_Adc_Notification_<#group_ID>(`

	)
Service ID[hex]:	0x20
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Will be called by the ADC Driver when a group conversion is completed for group <#group_ID>.

IoHwAb104: This function is called by the ADC driver when a group conversion is completed for group <#group_ID>

8.4.2 IoHwAb_Pwm_Notification

IoHwAb122:

Service name:	IoHwAb_Pwm_Notification_<#channel>	

Syntax:	`void IoHwAb_Pwm_Notification_<#channel>(`
	`    Pwm_EdgeNotificationType Notification`
	`)`	

Service ID[hex]:	0x30	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Notification	Type of notification PWM_RISING_EDGE or PWM_FALLING_EDGE or PWM_BOTH_EDGES
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Will be called by the PWM Driver when a signal edge occurs on channel <#channel>.	

IoHwAb105: This function is called by the PWM driver when a signal edge occurs on channel <#channel>

8.4.3 IoHwAb_Icu_Notification

IoHwAb123:

Service name:	IoHwAb_Icu_Notification_<#channel>

Syntax:	`void IoHwAb_Icu_Notification_<#channel>(`
	
	`)`

Service ID[hex]:	0x40
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	None
Return value:	None
Description:	Will be called by the ICU driver when a signal edge occurs on channel <#channel>.

IoHwAb106: This function is called by the ICU driver when a signal edge occurs on channel <#channel>

8.4.4 IoHwAb_Gpt_Notification

IoHwAb124:

Service name:	IoHwAb_Gpt_Notification_<#channel>

Syntax:	`void IoHwAb_Gpt_Notification_<#channel>(`
	
	 `)`

Service ID[hex]:	0x50
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Will be called by the GPT driver when a timer value expires on channel <#channel>.

IoHwAb107: This function is called by the GPT driver when a timer value expires on channel <#channel>

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1 <Name of scheduled function>

Service name:	<Name of API call>
Service ID [hex]:	<Number of service ID. This ID is used as parameter for the error report API of Development Error Tracer. The ID shall not be equal to an ID within chapter 8.3>
Description:	<Set of local software requirements including ID that define the operation of this API call.>
Timing:	<fixed cyclic / variable cyclic / on pre condition>
Pre condition:	<List of assumptions about the environment in which the API call must operate.>
Configuration:	<Description of statically configurable attributes that affect this API call. For instance cycle time(s) in case of fixed cyclic timing. Reference to chapter 10.>

Terms and definitions:

Fixed cyclic: Fixed cyclic means that one cycle time is defined at configuration and shall not be changed because functionality is requiring that fixed timing (e.g. filters).
Variable cyclic: Variable cyclic means that the cycle times are defined at configuration, but might be mode dependent and therefore vary during runtime.
On pre condition: On pre condition means that no cycle time can be defined. The function will be called when conditions are fulfilled. Alternatively, the function may be called cyclically however the cycle time will be assigned dynamically during runtime by other modules.

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

There is no mandatory interfaces for IO Hardware Abstraction.
It depends on the implemented ECU signals.

– For an implementation of ECU discrete signals, it is required to have the DIO driver and the PORT driver.
– For an implementation of ECU analog signals, it is required to have at least the ADC driver.
– For an implementation of ECU PWx signals, it is required to have at least the PWM and the ICU driver.

Example of an IO Hardware Abstraction using all MCAL drivers APIs :
Note that <module_name>_Init and <module_name>_DeInit functions are not listed bellow. The initialization sequence is called by the ECU state manager, and not by the IO Hardware Abstraction.
< module_name>_GetVersionInfo functions are also not listed here.

This table has been built according to following documents

– Driver ADC	document [12]
– Driver DIO	document [13]
– Driver ICU	document [14]
– Driver PWM	document [15]
– Driver PORT	document [16]
– Driver GPT	document [17]
– Driver SPI	document [18]

IoHwAb126:

API function	*Description*
Adc_DeInit	Returns all ADC HW Units to a state comparable to their power on reset state.
Adc_DisableGroupNotification	Disables the notification mechanism for the requested ADC Channel group.
Adc_DisableHardwareTrigger	Disables the hardware trigger for the requested ADC Channel group.
Adc_EnableGroupNotification	Enables the notification mechanism for the requested ADC Channel

	group.
Adc_EnableHardwareTrigger	Enables the hardware trigger for the requested ADC Channel group.
Adc_GetGroupStatus	Returns the conversion status of the requested ADC Channel group.
Adc_GetStreamLastPointer	Returns the number of valid samples per channel, stored in the result buffer. Reads a pointer, pointing to a position in the group result buffer. With the pointer position, the results of all group channels of the last completed conversion round can be accessed. With the pointer and the return value, all valid group conversion results can be accessed (the user has to take the layout of the result buffer into account).
Adc_GetVersionInfo	Returns the version information of this module.
Adc_Init	Initializes the ADC hardware units and driver.
Adc_ReadGroup	Reads the group conversion result of the last completed conversion round of the requested group and stores the channel values starting at the DataBufferPtr address. The group channel values are stored in ascending channel number order (in contrast to the storage layout of the result buffer if streaming access is configured).
Adc_SetupResultBuffer	Initializes the group specific ADC result buffer pointer as configured (see ADC291) to point to the DataBufferPtr address which is passed as parameter. The ADC driver stores all group conversion results to result buffer addressed with the result buffer pointer. Adc_SetupResultBuffer determines the address of the result buffer. After reset, before a group conversion can be started, an initialization of the ADC result buffer pointer is required.
Adc_StartGroupConversion	Starts the conversion of all channels of the requested ADC Channel group.
Adc_StopGroupConversion	Stops the conversion of the requested ADC Channel group.
Dio_GetVersionInfo	Service to get the version information of this module.
Dio_ReadChannel	Returns the value of the specified DIO channel.
Dio_ReadChannelGroup	This Service reads a subset of the adjoining bits of a port.
Dio_ReadPort	Returns the level of all channels of that port.
Dio_WriteChannel	Service to set a level of a channel.
Dio_WriteChannelGroup	Service to set a subset of the adjoining bits of a port to a specified level.
Dio_WritePort	Service to set a value of the port.
Gpt_DeInit	Deinitializes all hardware timer channels.
Gpt_DisableWakeup	Disables the wakeup interrupt invocation of a channel.
Gpt_EnableWakeup	Enables the wakeup interrupt invocation of a channel.
Gpt_GetTimeElapsed	Gets the time already elapsed.
Gpt_GetTimeRemaining	Gets the time remaining until the next timeout period will expire.
Gpt_GetVersionInfo	Returns the version information of this module.
Gpt_SetMode	Sets the operation mode of the GPT.
Icu_DeInit	This function de-initializes the ICU module.
Icu_DisableEdgeCount	This function disables the counting of edges of the given channel.
Icu_DisableNotification	This function disables the notification of a channel.
Icu_DisableWakeup	This function disables the wakeup capability of a single ICU channel.
Icu_EnableEdgeCount	This function enables the counting of edges of the given channel.
Icu_EnableNotification	This function enables the notification on the given channel.
Icu_EnableWakeup	This function (re-)enables the wakeup capability of the given ICU channel.
Icu_GetDutyCycleValues	This function reads the coherent active time and period time for the given ICU Channel.
Icu_GetEdgeNumbers	This function reads the number of counted edges.
Icu_GetInputState	This function returns the status of the ICU input.
Icu_GetTimeElapsed	This function reads the elapsed Signal Low Time for the given channel.
Icu_GetTimestampIndex	This function reads the timestamp index of the given channel.

Icu_GetVersionInfo	This function returns the version information of this module.
Icu_Init	This function initializes the driver.
Icu_ResetEdgeCount	This function resets the value of the counted edges to zero.
Icu_SetActivationCondition	This function sets the activation-edge for the given channel.
Icu_StartSignalMeasurement	This function starts the measurement of signals.
Icu_StartTimestamp	This function starts the capturing of timer values on the edges.
Icu_StopSignalMeasurement	This function stops the measurement of signals of the given channel.
Icu_StopTimestamp	This function stops the timestamp measurement of the given channel.
Port_GetVersionInfo	Returns the version information of this module.
Port_Init	Initializes the Port Driver module.
Port_RefreshPortDirection	Refreshes port direction.
Port_SetPinDirection	Sets the port pin direction
Port_SetPinMode	Sets the port pin mode.
Pwm_DeInit	Service for PWM De-Initialization.
Pwm_DisableNotification	Service to disable the PWM signal edge notification.
Pwm_EnableNotification	Service to enable the PWM signal edge notification according to notification parameter.
Pwm_GetOutputState	Service to read the internal state of the PWM output signal.
Pwm_GetVersionInfo	Service returns the version information of this module.
Pwm_Init	Service for PWM initialization.
Pwm_SetDutyCycle	Service sets the duty cycle of the PWM channel.
Pwm_SetOutputToIdle	Service sets the PWM output to the configured Idle state.
Pwm_SetPeriodAndDuty	Service sets the period and the duty cycle of a PWM channel
Spi_AsyncTransmit	Service to transmit data on the SPI bus.
Spi_Cancel	Service cancels the specified on-going sequence transmission.
Spi_DeInit	Service for SPI de-initialization.
Spi_GetHWUnitStatus	This service returns the status of the specified SPI Hardware microcontroller peripheral.
Spi_GetJobResult	This service returns the last transmission result of the specified Job.
Spi_GetSequenceResult	This service returns the last transmission result of the specified Sequence.
Spi_GetStatus	Service returns the SPI Handler/Driver software module status.
Spi_GetVersionInfo	This service returns the version information of this module.
Spi_Init	Service for SPI initialization.
Spi_MainFunction_Driving	--
Spi_MainFunction_Handling	--
Spi_ReadIB	Service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.
Spi_SetAsyncMode	Service to set the asynchronous mechanism mode for SPI busses handled asynchronously.
Spi_SetupEB	Service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.
Spi_SyncTransmit	Service to transmit data on the SPI bus
Spi_WriteIB	Service for writing one or more data to an IB SPI Handler/Driver Channel specified by parameter.

8.6.2 Optional Interfaces

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

IoHwAb125:

API function	***Description***
Dem_ReportErrorStatus	Reports errors to the DEM.
Det_ReportError	Service to report development errors.
EcuM_SetWakeupEvent	Sets the wakeup event.

8.6.3 Configurable interfaces

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

Name:	<Name of interface, no C syntax>	
Syntax:	`<Syntax of call including return type and parameters. Types must be defined using the type template. The real C name is not given, because it is configurable>`	
Reentrancy:	<reentrant / don't care>	
Parameters (in):	`<Parameter 1>`	<Description of parameter 1>
	`<Parameter 2>`	<Description of parameter 2>
Parameters (out):	`<Parameter 3>`	<Description of parameter 3>
Return value:	<Range of legal values>	<Description and the circumstances under which that value is returned, and the values of configuration attributes in which the value can be returned>
Description:	<Set of local software requirements including ID which define the operation of this API call.>	
Caveats:	<List of assumptions about the environment in which the API call must operate.>	
Configuration:	<Description of statically configurable attributes that affect this API call. Reference to configuration parameters described in chapter 10>	

Terms and definitions:
Reentrant: interface is expected to be reentrant
Don't care: reentrancy of interface not relevant for this module (in general it is in this case not reentrant).

8.6.4 Job End Notification

none

9 Sequence diagrams

9.1 ECU-signal provided by the IO Hardware Abstraction (example)

This sequence diagram explains the example of chapter 7.6.2.5.

In this example, the Sensor / Actuator Component is the client, the IO Hardware Abstraction is the server.

The Sensor/Actuator Component asks for a new value of the af_pressure AUTOSAR signal, that is an ECU signal on the level of the IO Hardware Abstraction.

After Adc conversion is finished, a notification coming from MCAL driver is converted into a RTE event for the Sensor / Actuator Component. Then, it can perform a synchronous read of the value present in the af_pressure signal buffer.

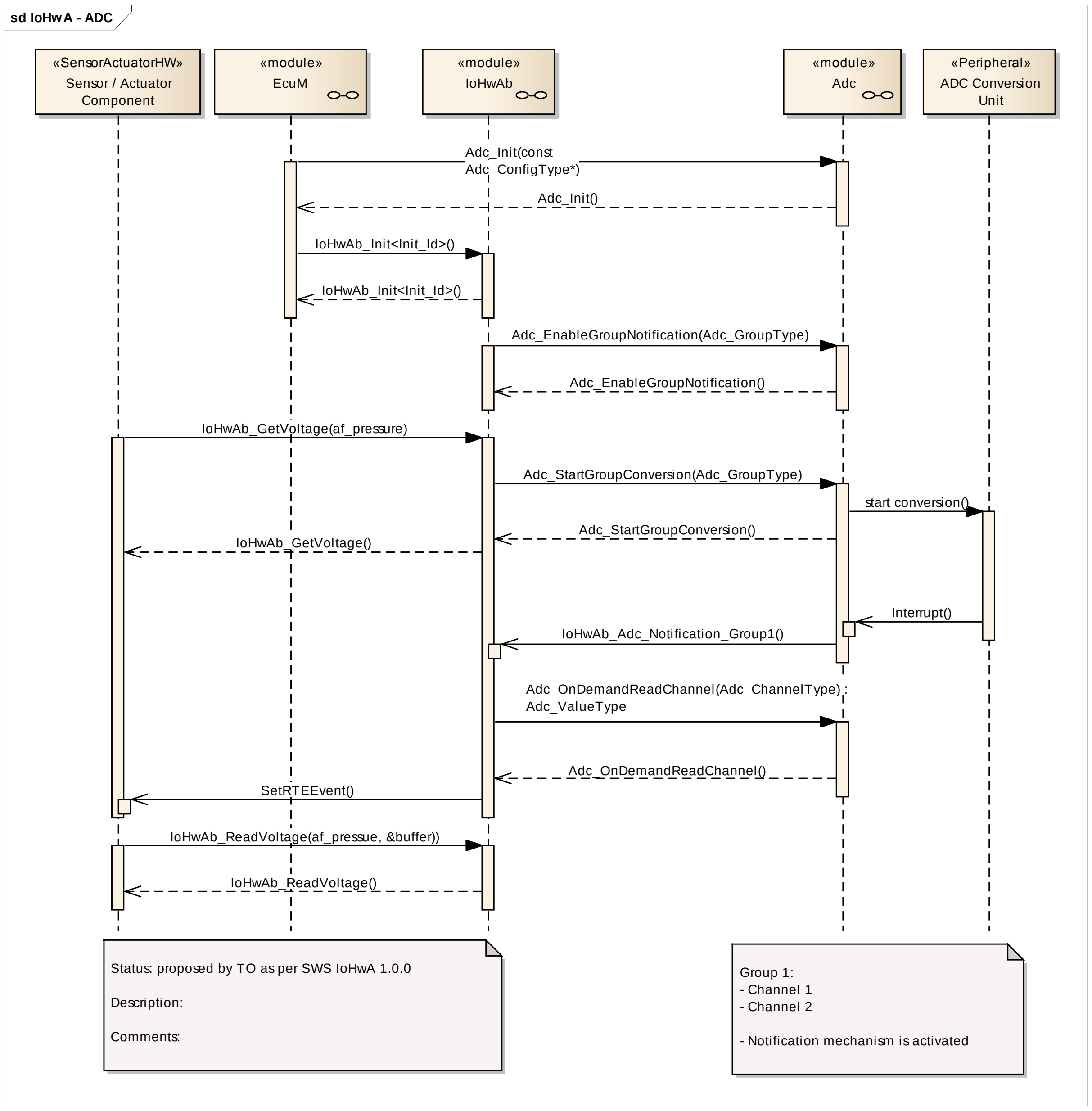
sd IoHwA - ADC
«SensorActuatorHW» Sensor / Actuator Component
«module» EcuM
«module» IoHwAb
«module» Adc
«Peripheral» ADC Conversion Unit
Adc_Init(const Adc_ConfigType*)
Adc_Init()
IoHwAb_Init<Init_Id>()
IoHwAb_Init<Init_Id>()
Adc_EnableGroupNotification(Adc_GroupType)
Adc_EnableGroupNotification()
IoHwAb_GetVoltage(af_pressure)
Adc_StartGroupConversion(Adc_GroupType)
start conversion()
Adc_StartGroupConversion()
IoHwAb_GetVoltage()
Interrupt()
IoHwAb_Adc_Notification_Group1()
Adc_OnDemandReadChannel(Adc_ChannelType) : Adc_ValueType
Adc_OnDemandReadChannel()
SetRTEEvent()
IoHwAb_ReadVoltage(af_pressue, &buffer))
IoHwAb_ReadVoltage()
Status: proposed by TO as per SWS IoHwA 1.0.0
Description:
Comments:
Group 1:
- Channel 1
- Channel 2
- Notification mechanism is activated

Figure 9.1: Sequence diagram - ADC conversion

Note: APIs `IoHwAb_GetVoltage(af_pressure)` and `IoHwAbReadVoltage(af_pressure, &buffer)` are not specified interfaces, and are given only for the example.

10 Configuration specification

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

Chapter 10.2 specifies the structure (containers) and the parameters of the IO Hardware Abstraction.

Chapter 10.3 specifies published information of the IO Hardware Abstraction.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [4]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

Not applicable

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.

- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.4 Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time	-	specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time	-	specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build	-	specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.2 Containers and configuration parameters

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

10.2.1 Variants

No variants specified

10.2.2 IoHwAbstraction

Module Name	*IoHwAbstraction*
Module Description	Configuration of IO HW Abstraction.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbEcuSignalGroup	0..*	This container contains the configuration (parameters) of all ECU signals groups implemented in the IO Hardware Abstraction Module
IoHwAbEcuSignals	0..*	This container contains the configuration (parameters) of all ECU signals implemented in the IO Hardware Abstraction Module. Remarks: - The IO Hardware Abstraction module is an ECU firmware module. That means that the implementation shall fit only to the ECU hardware. The implementer has to develop only the functionality needed (eg: if there is no pulse width ECU signal configurated, the pulse width abstraction will not be available. - At least one ECU signal shall be configurated to have the module.
IoHwAbGeneral	1	This container contains common description for the IO Hardware Abstraction module

10.2.3 IoHwAbGeneral

SWS Item	**IoHwAb096 :**
Container Name	IoHwAbGeneral{IoHwAb_Configuration}
Description	This container contains common description for the IO Hardware Abstraction module
Configuration Parameters	

SWS Item	:		
Name	IoHwAbDevErrorDetect		
Description	Switches the Development Error Detection and Notification ON or OFF		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbVersionInfoApi {IOHWAB_VERSION_INFO_API}		
Description	Pre-processor switch to enable / disable the API to read out the modules version information.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	

Scope / Dependency	

No Included Containers

10.2.4 IoHwAbEcuSignals

SWS Item	**IoHwAb087 :**
Container Name	IoHwAbEcuSignals{IoHwAb_ECUSignals_Configuration}
Description	This container contains the configuration (parameters) of all ECU signals implemented in the IO Hardware Abstraction Module. Remarks: - The IO Hardware Abstraction module is an ECU firmware module. That means that the implementation shall fit only to the ECU hardware. The implementer has to develop only the functionality needed (eg: if there is no pulse width ECU signal configurated, the pulse width abstraction will not be available. - At least one ECU signal shall be configurated to have the module.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbAnalogInput	0..*	This container contains the description of an analog ECU signal used as input. The operation available through the Port is a OP_GET.
IoHwAbAnalogOutput	0..*	This container contains the description of an analog ECU signal used as output. The operation available through the Port is a OP_SET.
IoHwAbDiscreteDiagnosis	0..*	This container contains the description of a discrete ECU signal used as diagnosis.
IoHwAbDiscreteInput	0..*	This container contains the description of a discrete ECU signal used as input. The operation available through the Port is an OP_GET.
IoHwAbDiscreteOutput	0..*	This container contains the description of a discrete ECU signal used as Output. The operation available through the Port is a OP_SET.
IoHwAbPwDutyCycleInput	0..*	This container contains specific description for an input ECU signal from class pulse width, handling especially the characteristic DutyCycle The operation available through the Port is a OP_GET
IoHwAbPwDutyCycleOutput	0..*	This container contains specific description for an output ECU signal from class pulse width, handling especially the characteristic DutyCycle. The operation available through the Port is a OP_SET.
IoHwAbPwPeriodInput	0..*	This container contains specific description for an input ECU signal from class pulse width, handling especially the characteristic period
IoHwAbPwPeriodOutput	0..*	This container contains specific description for an output ECU signal from class pulse width, handling especially the characteristic period The operation available through the Port is a OP_SET.

10.2.5 IoHwAbEcuSignalGroup

SWS Item	**IoHwAb127 :**
Container Name	IoHwAbEcuSignalGroup{IoHwAb_ECUSignalGroups_Configuration}
Description	This container contains the configuration (parameters) of all ECU signals groups implemented in the IO Hardware Abstraction Module
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbDiscSigGrpInput	0..*	This container contains the description of a discrete ECU signal group. The container IoHwAb_Discrete_SignalGroup contains several IoHwAb_Class_Discrete_Input sub-containers
IoHwAbDiscSigGrpOutput	0..*	This container contains the description of a discrete ECU signal group. The container IoHwAb_Discrete_SignalGroup contains several IoHwAb_Class_Discrete_Output sub-containers.

10.2.6 IoHwAbDiscSigGrpInput

SWS Item	**IoHwAb101 :**
Container Name	IoHwAbDiscSigGrpInput{IoHwAb_Discrete_SignalGroup_Inputs}
Description	This container contains the description of a discrete ECU signal group. The container IoHwAb_Discrete_SignalGroup contains several IoHwAb_Class_Discrete_Input sub-containers
Configuration Parameters	

SWS Item	**:**		
Name	IoHwAbDiscSigGrpInMember		
Description	Each member of the signal group is referenced by an instance of tthis element. In the IO HW Abstraction SWS, individual signals are aggregated by the signal group instead of referenced, but by using references it is still possible to have the same signal being par of several signal groups, as intended in the SWS.		
Multiplicity	1..*		
Type	Reference to [IoHwAbDiscreteInput]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.7 IoHwAbDiscSigGrpOutput

SWS Item	**IoHwAb103 :**
Container Name	IoHwAbDiscSigGrpOutput{IoHwAb_Discrete_SignalGroup_Outputs}
Description	This container contains the description of a discrete ECU signal group. The container IoHwAb_Discrete_SignalGroup contains several

	IoHwAb_Class_Discrete_Output sub-containers.
Configuration Parameters	

SWS Item	:		
Name	IoHwAbDiscSigGrpOutMember		
Description	Each member of the signal group is referenced by an instance of tthis element. In the IO HW Abstraction SWS, individual signals are aggregated by the signal group instead of referenced, but by using references it is still possible to have the same signal being par of several signal groups, as intended in the SWS.		
Multiplicity	1..*		
Type	Reference to [IoHwAbDiscreteOutput]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.8 IoHwAbDiscrete

SWS Item	**IoHwAb081 :**
Container Name	IoHwAbDiscrete{IoHwAb_Class_Discrete}
Description	This container contains common description of a all ECU signals from class discrete
Configuration Parameters	

SWS Item	:		
Name	IoHwAbDiscreteBswRange {IOHWAB_CLASS_DISCRETE_BSW_RANGE}		
Description	This parameter gives all possible values of an ECU discrete signal		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbDiscreteDatatype {IOHWAB_CLASS_DISCRETE_DATATYPE}		
Description	This parameter gives the DataType used for a discrete signal		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:
Name	IoHwAbDiscreteMaxAge {IOHWAB_CLASS_DISCRETE_MAX_AGE}

Description	This parameter gives the maximum age of an ECU discrete signal		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.9 IoHwAbDiscreteInput

SWS Item	**IoHwAb082 :**
Container Name	IoHwAbDiscreteInput{IoHwAb_Discrete_Input}
Description	This container contains the description of a discrete ECU signal used as input. The operation available through the Port is an OP_GET.
Configuration Parameters	

SWS Item	:		
Name	IoHwAbDiscInputDebounce {IOHWAB_DISCRETE_INPUT_DEBOUNCE}		
Description	This parameter specifies if the ECU discrete signal is filtered / debounced 0 represents Debouncing deactivated N represents Debouncing activated, N values sampled before value is available		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbDiscInputReport {IOHWAB_DISCRETE_INPUT_REPORT}		
Description	This parameter specifies if the report features is activated FALSE represents no report if a significant change occurs TRUE represents repot if a significant change occurs		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:
Name	IoHwAbDiscInputWakeup {IOHWAB_DISCRETE_INPUT_WAKEUP}
Description	This parameter specifies if the ECU pin is a wakeup input TRUE represents a wakeup Input. During sleep mode, a change is reported to the module. In case of a valid reason, the ECU state manager will be restarted to validate the wakeup.

	FALSE represents a simple discrete Input.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbDiscrete	1	This container contains common description of a all ECU signals from class discrete

10.2.10 IoHwAbDiscreteOutput

SWS Item	**IoHwAb083 :**
Container Name	IoHwAbDiscreteOutput{IoHwAb_Discrete_Output}
Description	This container contains the description of a discrete ECU signal used as Output. The operation available through the Port is a OP_SET.
Configuration Parameters	

SWS Item	**:**		
Name	IoHwAbDiscOutputFailureMoni {IOHWAB_DISCRETE_OUTPUT_FAILURE_MONITORING}		
Description	This parameter specifies if the ECU discrete Output is monitored, to detect failure TRUE represents that the ECU Signal is a monitored Output. Failures on this Output are detected and managed by the module FALSE represents that the ECU Signal is a simple discrete Output.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**:**		
Name	IoHwAbDiscOutputPulseTest {IOHWAB_DISCRETE_OUTPUT_PULSE_TEST}		
Description	This parameter specifies if an test pulse is used to detect failure on the ECU discrete Output TRUE represents that a Test Pulse is used for failure monitoring FALSE represents that no Test Pulse is used for failure monitoring		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbDiscrete	1	This container contains common description of a all ECU signals from class discrete

10.2.11 IoHwAbDiscreteDiagnosis

SWS Item	**IoHwAb099 :**
Container Name	IoHwAbDiscreteDiagnosis{IoHwAb_Discrete_Diagnosis}
Description	This container contains the description of a discrete ECU signal used as diagnosis.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbDiscrete	1	This container contains common description of a all ECU signals from class discrete

10.2.12 IoHwAbAnalog

SWS Item	**IoHwAb084 :**
Container Name	IoHwAbAnalog{IoHwAb_Class_Analog}
Description	This container contains common description of a all ECU signals from class analog.
Configuration Parameters	

SWS Item	:		
Name	IoHwAbAnalogBswRange {IOHWAB_CLASS_ANALOG_BSW_RANGE}		
Description	This parameter gives the BSW Range used for an analog signal, according to the datatype		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 ..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbAnalogBswResolution {IOHWAB_CLASS_ANALOG_BSW_RESOLUTION}		
Description	This parameter gives the resolution used for an analog signal		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbAnalogDataType {IOHWAB_CLASS_ANALOG_DATATYPE}		
Description	This parameter gives the DataType used for an analog signal		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbAnalogMaxAge {IOHWAB_CLASS_ANALOG_MAX_AGE}		
Description	This parameter gives the maximum age of an ECU analog signal		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbAnalogUnit {IOHWAB_CLASS_ANALOG_UNIT}		
Description	This parameter gives the unit used for an analog signal		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.13 IoHwAbAnalogInput

SWS Item	**IoHwAb085 :**
Container Name	IoHwAbAnalogInput{IoHwAb_Analog_Input}
Description	This container contains the description of an analog ECU signal used as input. The operation available through the Port is a OP_GET.
Configuration Parameters	

SWS Item	:
Name	IoHwAbAnalnputFilter {IOHWAB_ANALOG_INPUT_FILTER}

Description	This parameter specifies if the ECU analog signal is filtered		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbAnaInputReport {IOHWAB_ANALOG_INPUT_SAMPLING_RATE}		
Description	This parameter specifies if the report features is activated		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbAnaInputSamplingRate {IOHWAB_ANALOG_INPUT_SAMPLING_RATE}		
Description	This parameter specifies the sampling rate for an analog input		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbAnalog	1	This container contains common description of a all ECU signals from class analog.

10.2.14 IoHwAbAnalogOutput

SWS Item	**IoHwAb086 :**
Container Name	IoHwAbAnalogOutput{IoHwAb_Analog_Output}
Description	This container contains the description of an analog ECU signal used as output. The operation available through the Port is a OP_SET.
Configuration Parameters	

SWS Item	:
Name	IoHwAbAnaOutputFailureMoni {IOHWAB_ANALOG_OUTPUT_FAILURE_MONITORING}
Description	This parameter specifies if the ECU analog Output is monitored, to detect failure
Multiplicity	1
Type	BooleanParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbAnaOutputPulseTest {IOHWAB_ANALOG_OUTPUT_PULSETEST}		
Description	This parameter specifies if an test pulse is used to detect failure on the ECU analog Output		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbAnalog	1	This container contains common description of a all ECU signals from class analog.

10.2.15 IoHwAbPulseWidth

SWS Item	**IoHwAb088 :**
Container Name	IoHwAbPulseWidth{IoHwAb_Class_PulseWidth}
Description	This container contains common description of a all ECU signals from class pulse width
Configuration Parameters	

SWS Item	:		
Name	IoHwAbPwBswResolution {IOHWAB_CLASS_PULSEWIDTH_BSW_RESOLUTION}		
Description	This parameter gives the resolution used for a pulse width ECU signal		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:
Name	IoHwAbPwDatatype {IOHWAB_CLASS_PULSEWIDTH_BSW_RESOLUTION}
Description	This parameter gives the DataType used for a pulse width signal
Multiplicity	1
Type	StringParamDef
Default value	--

regularExpression	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**:**		
Name	IoHwAbPwMaxAge {IOHWAB_CLASS_PULSEWIDTH_MAX_AGE}		
Description	This parameter gives the maximum age for a pulse width ECU signal		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.16 IoHwAbPwPeriod

SWS Item	**IoHwAb089 :**
Container Name	IoHwAbPwPeriod{IoHwAb_Class_PulseWidthPeriod}
Description	This container contains common description of a all ECU signals from class pulse width, handling especially the characteristic period
Configuration Parameters	

SWS Item	**:**		
Name	IoHwAbPwpBswRange {IOHWAB_CLASS_PULSEWIDTHPERIOD_BSW_RANGE}		
Description	This parameter gives the BSW Range for a pulse width signal (in percent)		
Multiplicity	1		
Type	IntegerParamDef		
Range	-100 .. 100		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**:**		
Name	IoHwAbPwpUnit {IOHWAB_CLASS_PULSEWIDTHPERIOD_UNIT}		
Description	This parameter gives the unit for an ECU signal, handling especially the characteristic period		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	--	

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
IoHwAbPulseWidth	1	This container contains common description of a all ECU signals from class pulse width

10.2.17 IoHwAbPwPeriodInput

SWS Item	**IoHwAb090 :**
Container Name	IoHwAbPwPeriodInput{IoHwAb_PulseWidthPeriodInput}
Description	This container contains specific description for an input ECU signal from class pulse width, handling especially the characteristic period
Configuration Parameters	

SWS Item	:		
Name	IoHwAbPwpInputWakeup {IOHWAB_PulseWidthPeriod_Wakeup}		
Description	This parameter specifies if the ECU pin is a wakeup input		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	*Pre-compile time*	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
IoHwAbPwPeriod	1	This container contains common description of a all ECU signals from class pulse width, handling especially the characteristic period

10.2.18 IoHwAbPwPeriodOutput

SWS Item	**IoHwAb091 :**
Container Name	IoHwAbPwPeriodOutput{IoHwAb_PulseWidthPeriodOutput}
Description	This container contains specific description for an output ECU signal from class pulse width, handling especially the characteristic period The operation available through the Port is a OP_SET.
Configuration Parameters	

SWS Item	:
Name	IoHwAbPwpOutputFailureMoni {IOHWAB_PulseWidthPeriod_Output_Failure_Monitoring}
Description	This parameter specifies if the ECU Output is monitored, to detect failure
Multiplicity	1
Type	BooleanParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:		
Name	IoHwAbPwpOutputPulseTest {IOHWAB_PulseWidthPeriod_Output_PulseTest}		
Description	This parameter specifies if an test pulse is used to detect failure on the ECU Output		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbPwPeriod	1	This container contains common description of a all ECU signals from class pulse width, handling especially the characteristic period

10.2.19 IoHwAbPwDutyCycle

SWS Item	**IoHwAb092 :**
Container Name	IoHwAbPwDutyCycle{IoHwAb_Class_PulseWidthDutyCycle}
Description	This container contains common description of a all ECU signals from class pulse width, handling especially the characteristic duty cycle
Configuration Parameters	

SWS Item	:		
Name	IoHwAbPwdcBswRange {IOHWAB_CLASS_PULSEWIDTHDUTYCYCLE_BSW_RANGE}		
Description	This parameter gives the BSW Range for a pulse width signal		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 100		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	:
Name	IoHwAbPwdcUnit {IOHWAB_CLASS_PULSEWIDTHDUTYCYCLE_UNIT}
Description	This parameter gives the unit for an ECU signal, handling especially the characteristic period
Multiplicity	1

Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbPulseWidth	1	This container contains common description of a all ECU signals from class pulse width

10.2.20 IoHwAbPwDutyCycleInput

SWS Item	**IoHwAb093 :**
Container Name	IoHwAbPwDutyCycleInput{IoHwAb_PulseWidthDutyCycleInput}
Description	This container contains specific description for an input ECU signal from class pulse width, handling especially the characteristic DutyCycle The operation available through the Port is a OP_GET
Configuration Parameters	

SWS Item	:		
Name	IoHwAbPwdcInputWakeup {IOHWAB_PULSEWIDTHDUTYCYCLE_INPUT_WAKEUP}		
Description	This parameter specifies if the ECU pin is a wakeup input		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
IoHwAbPwDutyCycle	1	This container contains common description of a all ECU signals from class pulse width, handling especially the characteristic duty cycle

10.2.21 IoHwAbPwDutyCycleOutput

SWS Item	**IoHwAb094 :**
Container Name	IoHwAbPwDutyCycleOutput{IoHwAb_PulseWidthDutyCycleOutput}
Description	This container contains specific description for an output ECU signal from class pulse width, handling especially the characteristic DutyCycle. The operation available through the Port is a OP_SET.
Configuration Parameters	

SWS Item	**:**		
Name	IoHwAbPwdcOutputFailureMoni {IOHWAB_CLASS_PULSEWIDTHDUTYCYCLE_OUTPUT_FAILURE_MONITORING}		
Description	This parameter specifies if the ECU Output is monitored, to detect failure		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**:**		
Name	IoHwAbPwdcOutputPulseTest {IOHWAB_CLASS_PULSEWIDTHDUTYCYCLE_OUTPUT_PULSETEST}		
Description	This parameter specifies if an test pulse is used to detect failure on the ECU Output		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
IoHwAbPwDutyCycle	1	This container contains common description of a all ECU signals from class pulse width, handling especially the characteristic duty cycle

10.3 Published Information

Published information contains data defined by the implementer of the SW that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),

vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [20] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.