Document Title	Specification of PWM Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	037
Document Classification	Standard
Document Version	3.2.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	3.2.0	AUTOSAR Release Management	• Introduction of McuClockReferencePoint • Editorial changes
31.10.2013	3.1.0	AUTOSAR Release Management	• Updated requirements related to PwmPowerStateAsynchTransitionMode • Updated Scheduled Functions chapter • Editorial changes • Removed chapter(s) on change documentation
21.02.2013	3.0.0	AUTOSAR Administration	• Added ECU degradation concept • Adapted to new SWS BSW General Split memory map header
04.10.2011	2.5.0	AUTOSAR Administration	• Re-formulated SWS_Pwm_00045
29.10.2010	2.4.0	AUTOSAR Administration	• New Error symbol: PWM_E_PARAM_POINTER, shall be reported if API Pwm_GetVersionInfo service is called with a NULL parameter • Updated the chapter Version Check • Maintenance in phrasing and explaining
07.12.2009	2.3.0	AUTOSAR Administration	• The behavior of the function Pwm_SetPeriodAndDuty is explained in case of an input value of zero period. • Added the chapter Debug Support • Splitted some requirements so each ID is unique. • Legal disclaimer revised
23.06.2008	2.2.1	AUTOSAR Administration	• Legal disclaimer revised

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
20.12.2007	2.2.0	AUTOSAR Administration	• Tables generated from UML-models and UML-diagrams linked to UML-model • General improvements of requirements in preparation of CT-development • Reactivation concept for IDLE PWM channels adapted • Development error in case of already initialized module added • Document meta information extended • Small layout adaptations made
30.01.2007	2.1.0	AUTOSAR Administration	• Updated file include structure • Added configuration macros ON/OFF for PWM APIs • Renamed configuration parameter PWM_PERIOD_UPDATED_ENDPERIOD to PwmPeriodUpdatedEndperiod • Updated PWM signal description figure • Legal disclaimer revised • “Advice for users” revised • “Revision Information” added
25.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template. • Modify abstraction level of PWM channel • Notifications are configurable • Update the configuration of the module
23.06.2005	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module PWM driver.

Each PWM channel is linked to a hardware PWM which belongs to the microcontroller. The type of the PWM signal (for example center Align, left Align, Etc..) is not defined within this specification and is left up to the implementation.

The driver provides functions for initialization and control of the microcontroller internal PWM stage (pulse width modulation). The PWM module generates pulses with variable pulse width. It allows the selection of the duty cycle and the signal period time.

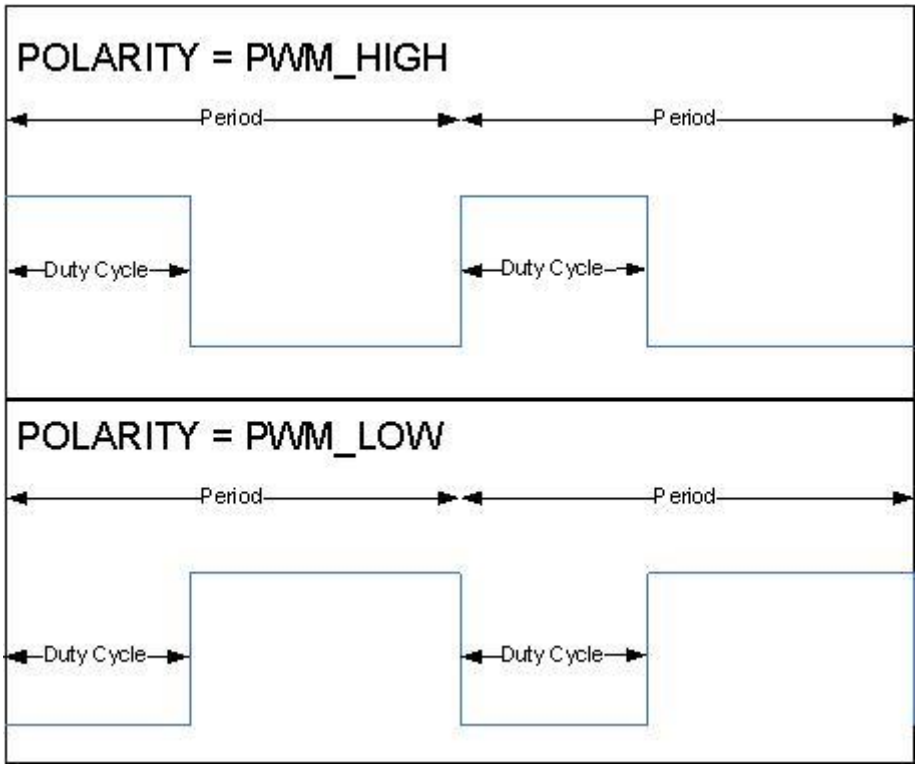


Figure 1: PWM signal description

2 Acronyms and abbreviations

Acronyms and abbreviations that have a local scope are not contained in the AUTOSAR glossary. These must appear in a local glossary.

Acronym:	***Description:***
PWM Channel	Numeric identifier linked to a hardware PWM.
PWM Output State	Defines the output state for a PWM signal. It could be: ▪ High. ▪ Low.
PWM Idle State	The idle state represents the output state of the PWM channel after the call of Pwm_SetOutputToIdle or Pwm_DeInit
PWM Polarity	Defines the starting output state of each PWM channel
PWM Duty cycle	Defines a percentage of the starting level (could be high or low) related to the period.
PWM period	Defines the period of the PWM signal.

Abbreviation:	***Description:***
PWM	Pulse Width Modulation.
DEM	Diagnostic Event Manager.
DET	Development Error Tracer.
MCU	Microcontroller Unit.
PLL	Phase Locked Loop.
ISR	Interrupt Service Routine.

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

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

[4] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[5] Specification of MCU Driver
AUTOSAR_SWS_MCUDriver.pdf

[6] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

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

[8] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[9] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [9] (SWS BSW General), which is also valid for PWM Driver.

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

4 Constraints and assumptions

4.1 Limitations

[SWS_Pwm_00001] ⌈The Pwm SWS does not cover PWM emulation on general purpose I/O.⌋ (SRS_Pwm_12386)

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

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

The PWM depends on the system clock. Thus, changes of the system clock (e.g. PLL on → PLL off) also affect the clock settings of the PWM hardware.

The PWM Driver depends on the following modules:

- PORT Driver: To set the port pin functionality. **PWM141**
- MCU Driver: To set prescaler, system clock and PLL. **PWM142**
- DET: Development Error Tracer in Development mode. **PWM143**

The document 087_AUTOSAR_ECU_Configuration contains a chapter 4.6 - *Clock Tree Configuration*, which details the mechanism to deliver reference clock signals to peripherals.

5.1 File structure

5.1.1 Code file structure

[SWS_Pwm_00065] ⌈The Pwm SWS shall not define the code file structure.⌋ (SRS_BSW_00380, SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00314, SRS_BSW_00370)

5.1.2 Header file structure

[SWS_Pwm_10075] ⌈Pwm.h shall include Pwm_Cfg.h.⌋ ()

[SWS_Pwm_40075] ⌈Pwm_Lcfg.c shall include Pwm.h and Pwm_Memmap.h.⌋ ()

[SWS_Pwm_50075] ⌈Pwm.c shall include Pwm.h, Pwm_MemMap.h, Det.h and SchM_Pwm.h.⌋ ()

[SWS_Pwm_60075] ⌈Pwm_PBcfg.c shall include Pwm_MemMap.h and Pwm.h.⌋ ()

[SWS_Pwm_70075] ⌈Pwm_Irq.c shall include Pwm_MemMap.h and Pwm.h.⌋ ()

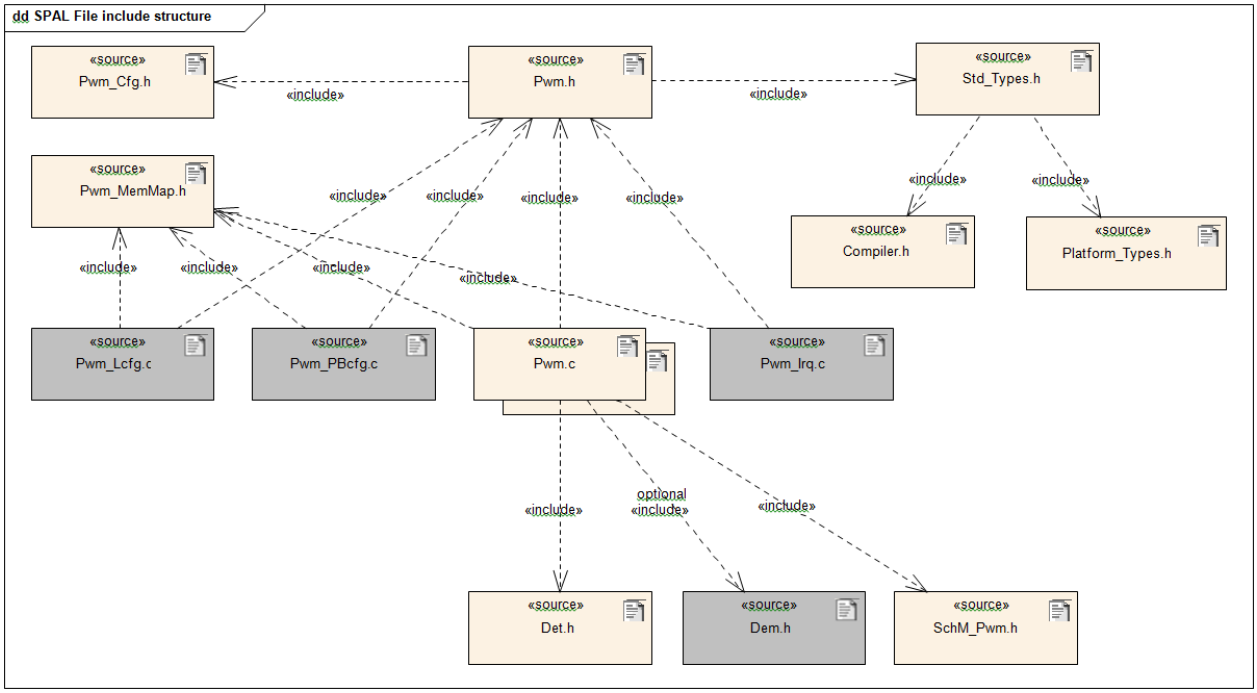


Figure 2: Header file structure

6 Requirements traceability

Requirement	Description	Satisfied by
-	-	SWS_Pwm_00014
-	-	SWS_Pwm_00016
-	-	SWS_Pwm_00018
-	-	SWS_Pwm_00020
-	-	SWS_Pwm_00061
-	-	SWS_Pwm_00076
-	-	SWS_Pwm_00077
-	-	SWS_Pwm_00079
-	-	SWS_Pwm_00081
-	-	SWS_Pwm_00088
-	-	SWS_Pwm_00089
-	-	SWS_Pwm_00093
-	-	SWS_Pwm_00094
-	-	SWS_Pwm_00095
-	-	SWS_Pwm_00096
-	-	SWS_Pwm_00097
-	-	SWS_Pwm_00098
-	-	SWS_Pwm_00099
-	-	SWS_Pwm_00100
-	-	SWS_Pwm_00101
-	-	SWS_Pwm_00102
-	-	SWS_Pwm_00104
-	-	SWS_Pwm_00105
-	-	SWS_Pwm_00106
-	-	SWS_Pwm_00107
-	-	SWS_Pwm_00108
-	-	SWS_Pwm_00109
-	-	SWS_Pwm_00110
-	-	SWS_Pwm_00111
-	-	SWS_Pwm_00116
-	-	SWS_Pwm_00118
-	-	SWS_Pwm_00119
-	-	SWS_Pwm_00121
-	-	SWS_Pwm_00150
-	-	SWS_Pwm_00151
-	-	SWS_Pwm_00154

-	-	SWS_Pwm_00155
-	-	SWS_Pwm_00156
-	-	SWS_Pwm_00157
-	-	SWS_Pwm_00158
-	-	SWS_Pwm_00159
-	-	SWS_Pwm_00160
-	-	SWS_Pwm_00161
-	-	SWS_Pwm_00162
-	-	SWS_Pwm_00163
-	-	SWS_Pwm_00164
-	-	SWS_Pwm_00165
-	-	SWS_Pwm_00166
-	-	SWS_Pwm_00167
-	-	SWS_Pwm_00168
-	-	SWS_Pwm_00169
-	-	SWS_Pwm_00170
-	-	SWS_Pwm_00171
-	-	SWS_Pwm_00172
-	-	SWS_Pwm_00173
-	-	SWS_Pwm_00174
-	-	SWS_Pwm_00175
-	-	SWS_Pwm_00176
-	-	SWS_Pwm_00177
-	-	SWS_Pwm_00178
-	-	SWS_Pwm_00179
-	-	SWS_Pwm_00180
-	-	SWS_Pwm_00181
-	-	SWS_Pwm_00182
-	-	SWS_Pwm_00183
-	-	SWS_Pwm_00184
-	-	SWS_Pwm_00185
-	-	SWS_Pwm_00186
-	-	SWS_Pwm_00187
-	-	SWS_Pwm_00188
-	-	SWS_Pwm_00189
-	-	SWS_Pwm_00190
-	-	SWS_Pwm_00191
-	-	SWS_Pwm_00192
-	-	SWS_Pwm_00193

-	-	SWS_Pwm_00194
-	-	SWS_Pwm_00195
-	-	SWS_Pwm_00196
-	-	SWS_Pwm_00198
-	-	SWS_Pwm_00199
-	-	SWS_Pwm_00200
-	-	SWS_Pwm_10075
-	-	SWS_Pwm_10086
-	-	SWS_Pwm_10112
-	-	SWS_Pwm_10113
-	-	SWS_Pwm_10115
-	-	SWS_Pwm_10120
-	-	SWS_Pwm_20086
-	-	SWS_Pwm_20112
-	-	SWS_Pwm_20113
-	-	SWS_Pwm_20115
-	-	SWS_Pwm_20120
-	-	SWS_Pwm_30115
-	-	SWS_Pwm_40075
-	-	SWS_Pwm_50075
-	-	SWS_Pwm_60075
-	-	SWS_Pwm_70075
BSW00431	-	SWS_Pwm_00153
BSW00434	-	SWS_Pwm_00153
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_Pwm_00153
SRS_BSW_00005	Modules of the æC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_Pwm_00153
SRS_BSW_00006	The source code of software modules above the æC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_Pwm_00153
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	SWS_Pwm_00153
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_Pwm_00153
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported	SWS_Pwm_00153

	platforms.	
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_Pwm_00007
SRS_BSW_00158	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation	SWS_Pwm_00065
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_Pwm_00153
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_Pwm_00153
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_Pwm_00153
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_Pwm_00153
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_Pwm_00153
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_Pwm_00153
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Pwm_00153
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_Pwm_00153
SRS_BSW_00171	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	SWS_Pwm_10080, SWS_Pwm_10082, SWS_Pwm_10083, SWS_Pwm_10084, SWS_Pwm_10085, SWS_Pwm_20080, SWS_Pwm_20082, SWS_Pwm_20083, SWS_Pwm_20084, SWS_Pwm_20085
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_Pwm_00153
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_Pwm_00153
SRS_BSW_00301	All AUTOSAR Basic Software	SWS_Pwm_00153

	Modules shall only import the necessary information	
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_Pwm_00153
SRS_BSW_00304	-	SWS_Pwm_00153
SRS_BSW_00305	Data types naming convention	SWS_Pwm_00153
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Pwm_00153
SRS_BSW_00307	Global variables naming convention	SWS_Pwm_00153
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_Pwm_00153
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_Pwm_00153
SRS_BSW_00310	API naming convention	SWS_Pwm_00153
SRS_BSW_00312	Shared code shall be reentrant	SWS_Pwm_00153
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_Pwm_00065
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_Pwm_00045, SWS_Pwm_00046, SWS_Pwm_00047, SWS_Pwm_00117, SWS_Pwm_10051, SWS_Pwm_20051, SWS_Pwm_30051
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_Pwm_00153
SRS_BSW_00326	-	SWS_Pwm_00153
SRS_BSW_00327	Error values naming convention	SWS_Pwm_00153
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_Pwm_00153
SRS_BSW_00329	-	SWS_Pwm_00153
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_Pwm_00153
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_Pwm_00153
SRS_BSW_00333	For each callback function it shall be specified if it is called from	SWS_Pwm_00153

	interrupt context or not	
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_Pwm_00153
SRS_BSW_00335	Status values naming convention	SWS_Pwm_00153
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_Pwm_00010
SRS_BSW_00337	Classification of development errors	SWS_Pwm_10002, SWS_Pwm_20002, SWS_Pwm_30002, SWS_Pwm_40002, SWS_Pwm_50002
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_Pwm_00153
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_Pwm_00153
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_Pwm_00070
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	SWS_Pwm_00065
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_Pwm_00153
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_Pwm_00153
SRS_BSW_00350	All AUTOSAR Basic Software Modules shall apply a specific naming rule for enabling/disabling the detection and reporting of development errors	SWS_Pwm_00153
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_Pwm_00153
SRS_BSW_00355	-	SWS_Pwm_00153
SRS_BSW_00357	For success/failure of an API call a standard return type shall be defined	SWS_Pwm_00153
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_Pwm_00153
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_Pwm_00153

SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_Pwm_00153
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_Pwm_00153
SRS_BSW_00370	-	SWS_Pwm_00065
SRS_BSW_00371	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	SWS_Pwm_00153
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_Pwm_00153
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_Pwm_00153
SRS_BSW_00376	-	SWS_Pwm_00153
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_Pwm_00153
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_Pwm_00153
SRS_BSW_00380	Configuration parameters being stored in memory shall be placed into separate c-files	SWS_Pwm_00065
SRS_BSW_00383	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	SWS_Pwm_00153
SRS_BSW_00385	List possible error notifications	SWS_Pwm_10002, SWS_Pwm_20002, SWS_Pwm_30002, SWS_Pwm_40002, SWS_Pwm_50002
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_Pwm_00045, SWS_Pwm_00046, SWS_Pwm_00047, SWS_Pwm_00117, SWS_Pwm_10002, SWS_Pwm_10051, SWS_Pwm_20002, SWS_Pwm_20051, SWS_Pwm_30002, SWS_Pwm_30051, SWS_Pwm_40002, SWS_Pwm_50002
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_Pwm_00153
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_Pwm_00117
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated	SWS_Pwm_20069

	module implementation	
SRS_BSW_00408	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	SWS_Pwm_00153
SRS_BSW_00410	Compiler switches shall have defined values	SWS_Pwm_00153
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_Pwm_00153
SRS_BSW_00414	The init function may have parameters	SWS_Pwm_00153
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_Pwm_00153
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_Pwm_00153
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_Pwm_00153
SRS_BSW_00419	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	SWS_Pwm_00153
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_Pwm_00153
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_Pwm_00153
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_Pwm_00153
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_Pwm_00153
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_Pwm_00153
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_Pwm_00153
SRS_BSW_00429	BSW modules shall be only allowed to use OS objects and/or related OS services	SWS_Pwm_00153

SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_Pwm_00153
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_Pwm_00153
SRS_Pwm_12293	The PWM driver shall allow the static configuration of PWM channel properties	SWS_Pwm_00197
SRS_Pwm_12295	The PWM driver shall provide a service for setting the duty cycle of a selected channel	SWS_Pwm_00013
SRS_Pwm_12297	The PWM driver shall provide a service for setting the period of a selected channel	SWS_Pwm_00019
SRS_Pwm_12299	The PWM driver shall allow to enable/disable the PWM edges notification during runtime	SWS_Pwm_00023, SWS_Pwm_00024
SRS_Pwm_12358	The PWM driver shall be capable to set the output of selected channel to a given state immediately	SWS_Pwm_00021
SRS_Pwm_12378	The PWM driver shall be able to assign notification to each edges of the PWM-signal	SWS_Pwm_00023, SWS_Pwm_00024, SWS_Pwm_00197
SRS_Pwm_12379	All PWM Channels which work with the same MCU Timer shall have either the same frequency or independent frequencies	SWS_Pwm_00153
SRS_Pwm_12381	By de-initializing the PWM driver, all PWM-channels shall be stop	SWS_Pwm_00010
SRS_Pwm_12382	The PWM Driver shall wait to the end of the signal period to update the duty cycle of a PWM signal	SWS_Pwm_00017
SRS_Pwm_12383	The PWM driver shall provide a 16 bit interface to set the duty cycle	SWS_Pwm_00058
SRS_Pwm_12385	The PWM driver shall provide a service to get the state of a PWM channel output	SWS_Pwm_00022
SRS_Pwm_12386	The PWM driver shall not cover a PWM emulation on general purpose I/O	SWS_Pwm_00001
SRS_Pwm_12389	The PWM driver shall allow only static configuration of the frequency for some PWM channels	SWS_Pwm_00041
SRS_Pwm_12459	The PWM Driver shall provide a scaling scheme for duty cycle	SWS_Pwm_00059

SRS_SPAL_00157	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	SWS_Pwm_00025
SRS_SPAL_12057	All driver modules shall implement an interface for initialization	SWS_Pwm_00007, SWS_Pwm_00052, SWS_Pwm_00062, SWS_Pwm_10009, SWS_Pwm_20009, SWS_Pwm_30009
SRS_SPAL_12064	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	SWS_Pwm_00153
SRS_SPAL_12067	All driver modules shall set their wake-up conditions depending on the selected operation mode	SWS_Pwm_00153
SRS_SPAL_12068	The modules of the MCAL shall be initialized in a defined sequence	SWS_Pwm_00153
SRS_SPAL_12069	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	SWS_Pwm_00153
SRS_SPAL_12075	All drivers with random streaming capabilities shall use application buffers	SWS_Pwm_00153
SRS_SPAL_12077	All drivers shall provide a non blocking implementation	SWS_Pwm_00153
SRS_SPAL_12078	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	SWS_Pwm_00153
SRS_SPAL_12092	The driver's API shall be accessed by its handler or manager	SWS_Pwm_00153
SRS_SPAL_12125	All driver modules shall only initialize the configured resources	SWS_Pwm_00062
SRS_SPAL_12129	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	SWS_Pwm_00026
SRS_SPAL_12163	All driver modules shall implement an interface for de-initialization	SWS_Pwm_00010, SWS_Pwm_00011, SWS_Pwm_00012
SRS_SPAL_12169	All driver modules that provide different operation modes shall provide a service for mode selection	SWS_Pwm_00153
SRS_SPAL_12265	Configuration data shall be kept constant	SWS_Pwm_00153
SRS_SPAL_12267	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	SWS_Pwm_00153
SRS_SPAL_12461	Specific rules regarding	SWS_Pwm_00153

	initialization of controller registers shall apply to all driver implementations	
SRS_SPAL_12462	The register initialization settings shall be published	SWS_Pwm_00153
SRS_SPAL_12463	The register initialization settings shall be combined and forwarded	SWS_Pwm_00153

7 Functional specification

7.1 General behavior

[SWS_Pwm_00088] ⌈All functions from the PWM module except `Pwm_Init`, `Pwm_DeInit` and `Pwm_GetVersionInfo` shall be re-entrant for different PWM channel numbers.

In order to keep a simple module implementation, no check of SWS_Pwm_00088 must be performed by the module. ⌋ ()

[SWS_Pwm_00089] ⌈The Pwm module's user shall ensure the integrity if several function calls are made during run time in different tasks or ISRs for the same PWM channel.⌋()

7.2 Time Unit Ticks

7.2.1 Background & Rationale

To get times out of register values it is necessary to know the oscillator frequency, prescalers and so on. Since these settings are made in MCU and/or in other modules it is not possible to calculate such times.
Hence the conversions between time and ticks shall be part of an upper layer.

7.2.2 Requirements

[SWS_Pwm_00070] ⌈All time units used within the API services of the PWM module shall be of the unit ticks. ⌋(SRS_BSW_00343)

7.3 Support and management of HW low power states

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

7.3.1 Background

The PWM Driver offers power state control APIs and a background elaboration mechanism to handle asynchronous power state change processes (i.e. power state

changes which are not immediately complete as the they are requested, but need some longer operations).

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

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

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

7.3.2 Requirements

[SWS_Pwm_00154] ⌈The PwmDriver shall support power state changes and its APIs when the corresponding configuration parameter `PwmLowPowerStatesSupport` is set to TRUE.⌋()

[SWS_Pwm_00155] ⌈If the parameter `PwmLowPowerStatesSupport` is enabled then the APIs `Pwm_PreparePowerState`, `Pwm_SetPowerState`, `Pwm_GetCurrentPowerState`, `Pwm_GetTargetPowerState` shall be generated and shall be used to manage and get informations on power state transitions.⌋()

[SWS_Pwm_00156] ⌈The APIs `Pwm_GetTargetPowerState` and `Pwm_GetCurrentPowerState` shall be respectively used to gather information on the requested and the target Pwm power states.⌋()

[SWS_Pwm_00157] ⌈The `API Pwm_PreparePowerState` shall be used to start a power state transition. ⌋()

[SWS_Pwm_00158] ⌈After preparation for a power state is achieved by (**[SWS_Pwm_00157]**) then the API `Pwm_SetPowerState` shall be used to achieve the requested power state of the Pwm module.

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

[SWS_Pwm_00159] ⌈The Pwm Driver shall keep track of the call order of the APIs `Pwm_SetPowerState` and `Pwm_PreparePowerState`. `In case the first one is called before the second one is called`, a DET entry shall be stored and the action shall not be executed.⌋()

[SWS_Pwm_00160] ⌈The Pwm Module shall keep track of the current and of the target powerstate if the parameter `PwmLowPowerStatesSupport` is set to TRUE⌋().

[SWS_Pwm_00161] ⌈After the Initiliazation the power state of the module shall be always `FULL POWER` if the `PwmLowPowerStatesSupport` is set to TRUE.⌋()

[SWS_Pwm_00162] ⌈The Pwm Driver shall support synchronuous and asynchronous power state transitions, depending on the value of the configuration parameter `PwmPowerStateAsynchTransitionMode`. ⌋()

[SWS_Pwm_00163] ⌈In case the configuration parameter PwmPowerStateAsynchTransitionMode is set to FALSE, the preparation process and the setting process shall be considered concluded as soon as the respective APIs return. ⌋()

[SWS_Pwm_00164] ⌈In case the configuration parameter PwmPowerStateAsynchTransitionMode is set to TRUE, the preparation process shall continue in background after the relative API returns and its completion shall be notified by means of the configured callback.⌋()

7.4 Error classification

[SWS_Pwm_10002] ⌈The PWM Driver module shall report the development error "PWM_E_PARAM_CONFIG (0x10)", when API Pwm_Init service is called with wrong parameter. ⌋ (SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386)

[SWS_Pwm_20002] ⌈The PWM Driver module shall report the development error "PWM_E_UNINIT (0x11)", when API service is used without module initialization. ⌋ (SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386)

[SWS_Pwm_30002] ⌈The PWM Driver module shall report the development error "PWM_E_PARAM_CHANNEL (0x12)", when API service is used with an invalid channel Identifier. ⌋ (SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386)

[SWS_Pwm_40002] ⌈The PWM Driver module shall report the development error "PWM_E_PERIOD_UNCHANGEABLE (0x13)", on usage of unauthorized PWM service on PWM channel configured a fixed period. ⌋ (SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386)

[SWS_Pwm_50002] ⌈The PWM Driver module shall report the development error "PWM_E_ALREADY_INITIALIZED(0x14)", when API Pwm_Init service is called while the PWM driver has already been initialized. ⌋ (SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386)

[SWS_Pwm_00151] ⌈The PWM Driver module shall report the development error "PWM_E_PARAM_POINTER (0x15)", when API Pwm_GetVersionInfo service is called with a NULL parameter. ⌋ ()

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API Pwm_Init service called with wrong parameter	Development	PWM_E_PARAM_CONFIG	0x10
API service used without module initialization	Development	PWM_E_UNINIT	0x11
API service used with an invalid channel Identifier	Development	PWM_E_PARAM_CHANNEL	0x12
Usage of unauthorized PWM service on PWM channel configured a fixed period	Development	PWM_E_PERIOD_UNCHANGEABLE	0x13
API Pwm_Init service called while the PWM driver has already been initialised	Development	PWM_E_ALREADY_INITIALIZED	0x14
API Pwm_GetVersionInfo is called with a NULL parameter.	Development	PWM_E_PARAM_POINTER	0x15
API Pwm_SetPowerState is called while the PWM module is still in use.	Development	PWM_E_NOT_DISENGAGED	0x16
The requested power state is not supported by the PWM module.	Development	PWM_E_POWER_STATE_NOT_SUPPORTED	0x17
The requested power state is not reachable from the current one.	Development	PWM_E_TRANSITION_NOT POSSIBLE	0x18
API Pwm_SetPowerState has been called without having called the API Pwm_PreparePowerState before.	Development	PWM_E_PERIPHERAL_NOT_PREPARED	0x19
--	Production	--	Assigned externally

[[SWS_Pwm_00200]]

⌈The API shall report the DET error **PWM_E_NOT_DISENGAGED** in case this API is called when one or more HW channels (where applicable) are in a state different

than IDLE (or similar non-operational states) and/or there are still notification registered for the HW module channels.」()

[SWS_Pwm_00174]
「The API shall report the DET error **PWM_E_POWER_STATE_NOT_SUPPORTED** in case this API is called with an unsupported power state or the peripheral does not support low power states at all.」()

[SWS_Pwm_00175]
「The API shall report the DET error **PWM_E_TRANSITION_NOT POSSIBLE** in case the requested power state cannot be directly reached from the current power state.」()

[[SWS_Pwm_00176]]
「The API shall report the DET error **PWM_E_PERIPHERAL_NOT_PREPARED** in case the HW unit has not been previously prepared for the target power state by use of the API Pwm_PreparePowerState(). 」()

To get more details concerning error detection, refer to chapter API parameter checking.

7.5 Error Detection

For details refer to the chapters 7.2 “Error classification” & 7.3 “Error Detection” in *SWS_BSWGeneral.*

7.6 Error Notification

For details refer to the chapters 7.2 “Error classification” & 7.3 “Error Detection” in *SWS_BSWGeneral.*

7.7 Duty Cycle Resolution and scaling

[SWS_Pwm_00058] 「The width of the duty cycle parameter is 16 Bits. 」(SRS_Pwm_12383)

[SWS_Pwm_00059] ⌈The Pwm module shall comply with the following scaling scheme for the duty cycle:

- 0x0000 means 0%.
- 0x8000 means 100%. 0x8000 gives the highest resolution while allowing 100% duty cycle to be represented with a 16 bit value.

As an implementation guide, the following source code example is given:

```
AbsoluteDutyCycle =
((uint32)AbsolutePeriodTime * RelativeDutyCycle) >> 15;
```

⌋(SRS_Pwm_12459)

7.8 Version check

For details refer to the chapter 5.1.8 “Version Check” in *SWS_BSWGeneral.*

7.9 Debug Support

For details refer to the chapter 7.1.17 “Debugging support” in *SWS_BSWGeneral.*

8 API specification

8.1 Imported types

This chapter lists all types included from other modules.

[SWS_Pwm_00094]

⌈

Module	*Imported Type*
Dem	Dem_EventIdType
	Dem_EventStatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋()

8.2 Type definitions

8.2.1 Pwm_ChannelType

[SWS_Pwm_00106]

⌈

Name:	`Pwm_ChannelType`		
Type:	`uint`		
Range:	`8..32 bit`	--	This is implementation specific but not all values may be valid within the type. This type shall be chosen in order to have the most efficient implementation on a specific microcontroller platform.
Description:	Numeric identifier of a PWM channel.		

⌋()

8.2.2 Pwm_PeriodType

[SWS_Pwm_00107]

⌈

Name:	`Pwm_PeriodType`		
Type:	`uint`		
Range:	`8..32 bit`	--	Implementation specific. This type shall be chosen in order to have the most efficient implementation on a specific microcontroller platform.
Description:	Definition of the period of a PWM channel.		

⌋()

8.2.3 Pwm_OutputStateType

[SWS_Pwm_00108]

⌈

Name:	Pwm_OutputStateType	
Type:	Enumeration	
Range:	PWM_HIGH	The PWM channel is in high state.
	PWM_LOW	The PWM channel is in low state.
Description:	Output state of a PWM channel.	

⌋()

8.2.4 Pwm_EdgeNotificationType

[SWS_Pwm_00109]

⌈

Name:	Pwm_EdgeNotificationType	
Type:	Enumeration	
Range:	PWM_RISING_EDGE	Notification will be called when a rising edge occurs on the PWM output signal.
	PWM_FALLING_EDGE	Notification will be called when a falling edge occurs on the PWM output signal.
	PWM_BOTH_EDGES	Notification will be called when either a rising edge or falling edge occur on the PWM output signal.
Description:	Definition of the type of edge notification of a PWM channel.	

⌋()

8.2.5 Pwm_ChannelClassType

[SWS_Pwm_00110]

⌈

Name:	Pwm_ChannelClassType	
Type:	Enumeration	
Range:	PWM_VARIABLE_PERIOD	The PWM channel has a variable period. The duty cycle and the period can be changed.
	PWM_FIXED_PERIOD	The PWM channel has a fixed period. Only the duty cycle can be changed.
	PWM_FIXED_PERIOD_SHIFTED	The PWM channel has a fixed shifted period. Impossible to change it (only if supported by hardware)
Description:	Defines the class of a PWM channel	

⌋()

8.2.6 Pwm_ConfigType

[SWS_Pwm_00111]

⌈

Name:	Pwm_ConfigType	
Type:	Structure	
Range:	Hardware dependent	The contents of the initialization data structure are hardware specific.

	structure.	
Description:	This is the type of data structure containing the initialization data for the PWM driver.	

」()

[SWS_Pwm_00061] 「Pwm_ConfigType is a type of data structure containing the initialization data for the PWM driver.」()

8.2.7 Pwm_PowerStateRequestResultType

[SWS_Pwm_00165]

「

Name:	Pwm_PowerStateRequestResultType	
Type:	Enumeration	
Range:	PWM_SERVICE_ACCEPTED	Power state change executed.
	PWM_NOT_INIT	PWM Module not initialized.
	PWM_SEQUENCE_ERROR	Wrong API call sequence.
	PWM_HW_FAILURE	The HW module has a failure which prevents it to enter the required power state.
	PWM_POWER_STATE_NOT_SUPP	PWM Module does not support the requested power state.
	PWM_TRANS_NOT_POSSIBLE	PWM Module cannot transition directly from the current power state to the requested power state or the HW peripheral is still busy.
Description:	Result of the requests related to power state transitions.	

」()

8.2.8 Pwm_PowerStateType

[SWS_Pwm_00197]

「

Name:	Pwm_PowerStateType	
Type:	Enumeration	
Range:	PWM_FULL_POWER	Full Power (0)
	1..255	power modes with decreasing power consumptions.
Description:	Power state currently active or set as target power state.	

Mandatory parameters:

- Assigned HW channel
- Default value for period
- Default value for duty cycle
- Polarity (high or low)
- Idle state high or low
- Channel class:
  - Fixed period
  - Fixed period, shifted (if supported by hardware)
  - Variable period

Optional parameters (if supported by hardware):

- Channel phase shift
- Reference channel for phase shift
- Microcontroller specific channel properties

」(SRS_Pwm_12293, SRS_Pwm_12378)

8.3 Function definitions

8.3.1 Pwm_Init

[SWS_Pwm_00095]

「

Service name:	Pwm_Init	
Syntax:	`void Pwm_Init( const Pwm_ConfigType* ConfigPtr )`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to configuration set
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service for PWM initialization.	

」()

[SWS_Pwm_00007] 「The function Pwm_Init shall initialize all internals variables and the used PWM structure of the microcontroller according to the parameters specified in ConfigPtr.」(SRS_BSW_00101, SRS_SPAL_12057)

[SWS_Pwm_00062] 「The function Pwm_Init shall only initialize the configured resources and shall not touch resources that are not configured in the configuration file. 」(SRS_SPAL_12057, SRS_SPAL_12125)

[SWS_Pwm_10009] ⌈The function Pwm_Init shall start all PWM channels with the configured default values.⌋(SRS_SPAL_12057)
If the duty cycle parameter equals:

- **[SWS_Pwm_20009]** ⌈0% or 100% : Then the PWM output signal shall be in the state according to the configured polarity parameter⌋(SRS_SPAL_12057)
- **[SWS_Pwm_30009]** ⌈>0% and <100%: Then the PWM output signal shall be modulated according to parameters period, duty cycle and configured polarity.⌋(SRS_SPAL_12057)

[SWS_Pwm_00052] ⌈The function Pwm_Init shall disable all notifications.⌋(SRS_SPAL_12057)

The reason is that the users of these notifications may not be ready. They can call Pwm_EnableNotification to start notifications.

[SWS_Pwm_00093] ⌈The users of the Pwm module shall not call the function Pwm_Init during a running operation.⌋()

[SWS_Pwm_00046] ⌈If development error detection is enabled for the Pwm module, the function Pwm_Init shall raise development error `PWM_E_PARAM_CONFIG` if `ConfigPtr` is a null pointer.

Regarding error detection, the requirement SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 is applicable to the function Pwm_Init.⌋(SRS_BSW_00323, SRS_BSW_00386)

[SWS_Pwm_00116] ⌈The Pwm module's environment shall not call any function of the Pwm module before having called Pwm_Init. .⌋()

[SWS_Pwm_00118] ⌈If development error detection is enabled, calling the routine Pwm_Init while the PWM driver and hardware are already initialized will cause a development error PWM_E_ALREADY_INITIALIZED. The desired functionality shall be left without any action.⌋()

[SWS_Pwm_10120] ⌈For pre-compile and link time configuration variants, a NULL pointer shall be passed to the initialization routine.⌋()
[SWS_Pwm_20120] ⌈In this case the check for this NULL pointer has to be omitted.⌋()

[SWS_Pwm_00121] ⌈A re-initialization of the Pwm driver by executing the Pwm_Init() function requires a de-initialization before by executing a Pwm_DeInit().⌋()

8.3.2 Pwm_DeInit

[SWS_Pwm_00096]

⌈

Service name:	Pwm_DeInit
Syntax:	`void Pwm_DeInit( void )`
Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for PWM De-Initialization.

⌋()

[SWS_Pwm_00010] ⌈The function Pwm_DeInit shall de-initialize the PWM module. ⌋ (SRS_BSW_00336, SRS_SPAL_12163, SRS_Pwm_12381)

[SWS_Pwm_00011] ⌈The function Pwm_DeInit shall set the state of the PWM output signals to the idle state. ⌋ (SRS_SPAL_12163)

[SWS_Pwm_00012] ⌈The function Pwm_DeInit shall disable PWM interrupts and PWM signal edge notifications. ⌋ (SRS_SPAL_12163)

[SWS_Pwm_10080] ⌈The function Pwm_DeInit shall be pre compile time configurable On/Off by the configuration parameter: PwmDeInitApi. ⌋ (SRS_BSW_00171)

[SWS_Pwm_20080] ⌈The function Pwm_DeInit shall be configurable On/Off by the configuration parameter PwmDeInitApi {PWM_DE_INIT_API}.

Regarding error detection, the requirements SWS_Pwm_00117, SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 are applicable to the function Pwm_DeInit. ⌋ (SRS_BSW_00171)

8.3.3 Pwm_SetDutyCycle

[SWS_Pwm_00097] ⌈

Service name:	Pwm_SetDutyCycle
Syntax:	`void Pwm_SetDutyCycle( Pwm_ChannelType ChannelNumber, uint16 DutyCycle )`

Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the PWM
	DutyCycle	Min=0x0000 Max=0x8000
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service sets the duty cycle of the PWM channel.	

⌋ ()

[SWS_Pwm_00013] ⌈The function Pwm_SetDutyCycle shall set the duty cycle of the PWM channel. ⌋ (SRS_Pwm_12295)

[SWS_Pwm_00014] ⌈When the requested duty cycle is either 0% or 100%, the function
Pwm_SetDutyCycle shall set the PWM output state to either PWM_HIGH or PWM_LOW, with regard to both the configured polarity parameter and the requested duty cycle.
Thus for 0% requested Duty Cycle the output will be the inverse of the configured polarity parameter, and for 100% Duty Cycle the output will be equal to the configured polarity parameter. ⌋ ()

[SWS_Pwm_00016] ⌈The function Pwm_SetDutyCycle shall modulate the PWM output signal according to parameters period, duty cycle and configured polarity, when the duty cycle > 0 % and < 100%.⌋ ()

[SWS_Pwm_00017] ⌈The function Pwm_SetDutyCycle shall update the duty cycle always at the end of the period if supported by the implementation and configured with PwmDutycycleUpdatedEndperiod. ⌋ (SRS_Pwm_12382)

Regarding format definition of duty cycle parameter, the requirement SWS_Pwm_00058 is applicable to the function Pwm_SetDutyCycle.

Regarding scaling definition of duty cycle parameter, the requirement SWS_Pwm_00059 is applicable to the function Pwm_SetDutyCycle.

[SWS_Pwm_00018] ⌈The driver shall forbid the spike on the PWM output signal. ⌋ ()

Regarding error detection, the requirements SWS_Pwm_00117, SWS_Pwm_00047, SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 are applicable to the function Pwm_SetDutyCycle.

[SWS_Pwm_10082] ⌈The function Pwm_SetDutyCycle shall be pre compile time configurable On/Off by the configuration parameter: PwmSetDutyCycle. .⌋ (SRS_BSW_00171)

[SWS_Pwm_20082] ⌈The function Pwm_SetDutyCycle shall be configurable On/Off by the configuration parameter: PwmSetDutyCycle {PWM_SET_DUTY_CYCLE_API}.⌋ (SRS_BSW_00171)

8.3.4 Pwm_SetPeriodAndDuty

[SWS_Pwm_00098] ⌈

Service name:	Pwm_SetPeriodAndDuty	
Syntax:	`void Pwm_SetPeriodAndDuty( Pwm_ChannelType ChannelNumber, Pwm_PeriodType Period, uint16 DutyCycle )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the PWM
	Period	Period of the PWM signal
	DutyCycle	Min=0x0000 Max=0x8000
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service sets the period and the duty cycle of a PWM channel	

⌋ ()

[SWS_Pwm_00019] ⌈The function Pwm_SetPeriodAndDuty shall set the period and the duty cycle of a PWM channel. ⌋ (SRS_Pwm_12297)

[SWS_Pwm_00076] ⌈The function Pwm_SetPeriodAndDuty shall update the period always at the end of the current period if supported by the implementation and configured with PwmPeriodUpdatedEndperiod. ⌋ ()

[SWS_Pwm_00020] ⌈When updating the PWM period and duty, the driver shall repress any spikes on the PWM output signal. ⌋ ()

The PWM duty cycle parameter is necessary to maintain the consistency between frequency and duty cycle. Refer to SWS_Pwm_00058 and SWS_Pwm_00059 to know the scaling and format definition of duty cycle parameter

Regarding error detection, the requirements SWS_Pwm_00117, SWS_Pwm_00045, SWS_Pwm_00047, SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 are applicable to the function Pwm_SetPeriodAndDuty.

[SWS_Pwm_00041] ⌈The function Pwm_SetPeriodAndDuty shall allow changing the period only for the PWM channel declared as variable period type. ⌋ (SRS_Pwm_12389)

[SWS_Pwm_10083] ⌈The function Pwm_SetPeriodAndDuty shall be pre compile time configurable On/Off by the configuration parameter: PwmSetPeriodAndDuty. ⌋ (SRS_BSW_00171)

[SWS_Pwm_20083] ⌈The function Pwm_SetPeriodAndDuty shall be configurable On/Off by the configuration parameter: PwmSetPeriodAndDuty {PWM_SET_PERIOD_AND_DUTY_API}.⌋ (SRS_BSW_00171)

[SWS_Pwm_00150] ⌈If the period is set to zero the setting of the duty-cycle is not relevant. In this case the output shall be zero (zero percent duty-cycle). ⌋ ()

8.3.5 Pwm_SetOutputToIdle

[SWS_Pwm_00099] ⌈

Service name:	Pwm_SetOutputToIdle	
Syntax:	`void Pwm_SetOutputToIdle( Pwm_ChannelType ChannelNumber )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the PWM
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service sets the PWM output to the configured Idle state.	

⌋ ()

[SWS_Pwm_00021] ⌈The function Pwm_SetOutputToIdle shall set immediately the PWM output to the configured Idle state. ⌋ (SRS_Pwm_12358)

Regarding error detection, the requirements SWS_Pwm_00117, SWS_Pwm_00047, SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 are applicable to the function Pwm_SetOutputToIdle.

[SWS_Pwm_10084] ⌈The function Pwm_SetOutputToIdle shall be pre compile time configurable On/Off by the configuration parameter: PwmSetOutputToIdle. ⌋ (SRS_BSW_00171)

[SWS_Pwm_20084] ⌈The function Pwm_SetOutputToIdle shall be configurable On/Off by the configuration parameter: PwmSetOutputToIdle {PWM_SET_OUTPUT_TO_IDLE_API}.⌋ (SRS_BSW_00171)

[SWS_Pwm_10086] ⌈After the call of the function Pwm_SetOutputToIdle, variable period type channels shall be reactivated using the Api Pwm_SetPeriodAndDuty( ) to activate the PWM channel with the new passed period. ⌋ ()

[[SWS_Pwm_20086]] ⌈After the call of the function Pwm_SetOutputToIdle, channels shall be reactivated using the Api Pwm_SetDutyCycle( ) to activate the PWM channel with the old period.⌋()

[[SWS_Pwm_00119]] ⌈After the call of the function Pwm_SetOutputToIdle, fixed period type channels shall be reactivated using only the API Pwm_SetDutyCycle() to activate the PWM channel with the old period. ⌋()

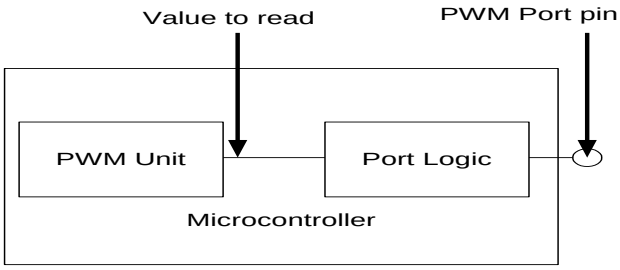
8.3.6 Pwm_GetOutputState

[SWS_Pwm_00100] ⌈

Service name:	Pwm_GetOutputState	
Syntax:	`Pwm_OutputStateType Pwm_GetOutputState( Pwm_ChannelType ChannelNumber )`	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the PWM
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Pwm_OutputStateType	PWM_HIGH The PWM output state is high PWM_LOW The PWM output state is low
Description:	Service to read the internal state of the PWM output signal.	

⌋ ()

[SWS_Pwm_00022] ⌈The function Pwm_GetOutputState shall read the internal state of the PWM output signal and return it as defined in the diagram below



Regarding error detection, the requirements SWS_Pwm_00117, SWS_Pwm_00047, SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 are applicable to the function Pwm_GetOutputState. ⌋ (SRS_Pwm_12385)

[SWS_Pwm_10085] ⌈The function Pwm_GetOutputState shall be pre compile time configurable On/Off using the configuration parameter: PwmGetOutputState. ⌋ (SRS_BSW_00171)

[SWS_Pwm_20085] ⌈The function Pwm_GetOutputState shall be configurable On/Off by the configuration parameter: PwmGetOutputState {PWM_GET_OUTPUT_STATE_API}.

Due to real time constraint and setting of the PWM channel (project dependant), the output state can be modified just after the call of the service Pwm_GetOutputState. ⌋ (SRS_BSW_00171)

8.3.7 Pwm_DisableNotification

[SWS_Pwm_00101] ⌈

Service name:	Pwm_DisableNotification
Syntax:	`void Pwm_DisableNotification( Pwm_ChannelType ChannelNumber )`
Service ID[hex]:	0x06
Sync/Async:	Synchronous
Reentrancy:	Reentrant for different channel numbers
Parameters (in):	ChannelNumber — Numeric identifier of the PWM
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service to disable the PWM signal edge notification.

⌋ ()

[SWS_Pwm_00023] ⌈The function Pwm_DisableNotification shall disable the PWM signal edge notification. ⌋ (SRS_Pwm_12378, SRS_Pwm_12299)

[SWS_Pwm_10112] ⌈The function Pwm_DisableNotification shall be pre compile time configurable On/Off using the configuration parameter: PwmNotificationSupported. ⌋ ()

[SWS_Pwm_20112] ⌈The function Pwm_DisableNotification shall be configurable On/Off by the configuration parameter: PwmNotificationSupported {PWM_NOTIFICATION_SUPPORTED}.

Regarding error detection, the requirements SWS_Pwm_00117, SWS_Pwm_00047, SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 are applicable to the function Pwm_DisableNotification. ⌋ ()

8.3.8 Pwm_EnableNotification

[SWS_Pwm_00102] ⌈

Service name:	Pwm_EnableNotification	
Syntax:	`void Pwm_EnableNotification( Pwm_ChannelType ChannelNumber, Pwm_EdgeNotificationType Notification )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the PWM
	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:	Service to enable the PWM signal edge notification according to notification parameter.	

⌋ ()

[SWS_Pwm_00024] ⌈The function Pwm_EnableNotification shall enable the PWM signal edge notification according to notification parameter. ⌋ (SRS_Pwm_12378, SRS_Pwm_12299)

[SWS_Pwm_00081] ⌈The function Pwm_EnableNotification shall cancel pending interrupts. ⌋ ()

[SWS_Pwm_10113] ⌈The function Pwm_EnableNotification shall be pre compile time configurable On/Off using the configuration parameter: PwmNotificationSupported. ⌋ ()

[SWS_Pwm_20113] ⌈The function Pwm_EnableNotification shall be configurable On/Off by the configuration parameter: PwmNotificationSupported {PWM_NOTIFICATION_SUPPORTED}.

Regarding error detection, the requirements SWS_Pwm_00117, SWS_Pwm_00047, SWS_Pwm_10051, SWS_Pwm_20051 and SWS_Pwm_30051 are applicable to the function Pwm_EnableNotification. ⌋ ()

8.3.9 Pwm_SetPowerState

[[SWS_Pwm_00166]]

⌈

Service name:	Pwm_SetPowerState
Syntax:	`Std_ReturnType Pwm_SetPowerState( Pwm_PowerStateRequestResultType* Result )`

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	Result	If the API returns E_OK: PWM_SERVICE_ACCEPTED:Power state change executed. If the API returns E_NOT_OK: PWM_NOT_INIT: PWM Module not initialized. PWM_SEQUENCE_ERROR: wrong API call sequence. PWM_HW_FAILURE: the HW module has a failure which prevents it to enter the required power state.
Return value:	Std_ReturnType	E_OK: Power Mode changed E_NOT_OK: request rejected
Description:	This API configures the Pwm module so that it enters the already prepared power state, chosen between a predefined set of configured ones.	

」()

[[SWS_Pwm_00167]

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

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

[SWS_Pwm_00168]

「In case the target power state is the same as the current one, no action is executed and the API returns immediately with an E_OK result.」()

[SWS_Pwm_00169]

「In case the normal Power State is requested, the API shall refer to the necessary parameters contained in the same containers used by Pwm_Init.

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

[SWS_Pwm_00170]

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

[SWS_Pwm_00171]

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

In case development error reporting is activated:

[SWS_Pwm_00172]

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

[SWS_Pwm_00173]

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

[SWS_Pwm_00194]

⌈The API shall report the DET error **PWM_E_POWER_STATE_NOT_SUPPORTED** in case this API is called with an unsupported power state or the peripheral does not support low power states at all.

⌋()

[SWS_Pwm_00195]

⌈The API shall report the DET error **PWM_E_TRANSITION_NOT POSSIBLE** in case the requested power state cannot be directly reached from the current power state.⌋()

[SWS_Pwm_00196]

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

8.3.10 Pwm_GetCurrentPowerState

[[SWS_Pwm_00177]

]⌈

Service name:	Pwm_GetCurrentPowerState
Syntax:	`Std_ReturnType Pwm_GetCurrentPowerState( Pwm_PowerStateType* CurrentPowerState, Pwm_PowerStateRequestResultType* Result )`
Service ID[hex]:	0x0a
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	CurrentPowerState	The current power mode of the PWM HW Unit is returned in this parameter
	Result	If the API returns E_OK: PWM_SERVICE_ACCEPTED: Current power mode was returned. If the API returns E_NOT_OK: PWM_NOT_INIT: PWM Module not initialized.
Return value:	Std_ReturnType	E_OK: Mode could be read E_NOT_OK: Service is rejected
Description:	This API returns the current power state of the PWM HW unit.	

⌋()

[[SWS_Pwm_00178]
⌈The API returns the power state of the HW unit.

In case development error reporting is activated:⌋()

[[SWS_Pwm_00179]
⌈The API shall report the DET error **PWM_E_UNINIT** in case this API is called before having initialized the HW unit.⌋()

8.3.11 Pwm_GetTargetPowerState

[[SWS_Pwm_00180]
⌈]

Service name:	Pwm_GetTargetPowerState	
Syntax:	`Std_ReturnType Pwm_GetTargetPowerState( Pwm_PowerStateType* TargetPowerState, Pwm_PowerStateRequestResultType* Result )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	TargetPowerState	The Target power mode of the PWM HW Unit is returned in this parameter
	Result	If the API returns E_OK: PWM_SERVICE_ACCEPTED:Target power mode was returned. If the API returns E_NOT_OK: PWM_NOT_INIT: PWM Module not initialized.
Return value:	Std_ReturnType	E_OK: Mode could be read E_NOT_OK: Service is rejected
Description:	This API returns the Target power state of the PWM HW unit.	

⌋()

[[SWS_Pwm_00181]
⌈The API returns the requested power state of the HW unit. This shall coincide with the current power state if no transition is ongoing.

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

In case development error reporting is activated:⌋()

[[SWS_Pwm_00182]
⌈The API shall report the DET error **PWM_E_UNINIT** in case this API is called before having initialized the HW unit.⌋()

8.3.12 Pwm_PreparePowerState

[[SWS_Pwm_00183]

⌈

Service name:	Pwm_PreparePowerState	
Syntax:	`Std_ReturnType Pwm_PreparePowerState( Pwm_PowerStateType PowerState, Pwm_PowerStateRequestResultType* Result )`	
Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	PowerState	The target power state intended to be attained
Parameters (inout):	None	
Parameters (out):	Result	If the API returns E_OK: PWM_SERVICE_ACCEPTED: PWM Module power state preparation was started. If the API returns E_NOT_OK: PWM_NOT_INIT: PWM Module not initialized. PWM_SEQUENCE_ERROR: wrong API call sequence (Current Power State = Target Power State). PWM_POWER_STATE_NOT_SUPP: PWM Module does not support the requested power state. PWM_TRANS_NOT_POSSIBLE: PWM Module cannot transition directly from the current power state to the requested power state or the HW peripheral is still busy.
Return value:	Std_ReturnType	E_OK: Preparation process started E_NOT_OK: Service is rejected
Description:	This API starts the needed process to allow the PWM HW module to enter the requested power state.	

⌋()

[[SWS_Pwm_00184]

⌜This API initiates all actions needed to enable a HW module to enter the target power state.

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

[[SWS_Pwm_00185]
⌜In case the target power state is the same as the current one, no action is executed and the API returns immediately with an E_OK result.

The responsibility of the preconditions is left to the environment.

In case development error reporting is activated.⌟()

[[SWS_Pwm_00186]
⌜The API shall report the DET error **PWM_E_UNINIT** in case this API is called before having initialized the HW unit.⌟()

[[SWS_Pwm_00187]
⌜The API shall report the DET error **PWM_E_POWER_STATE_NOT_SUPPORTED** in case this API is called with an unsupported power state is requested or the peripheral does not support low power states at all.⌟()

[[SWS_Pwm_00188]
⌜The API shall report the DET error **PWM_E_TRANSITION_NOT POSSIBLE** in case the requested power state cannot be directly reached from the current power state.

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

8.3.13 Pwm_GetVersionInfo

[SWS_Pwm_00103]

Service name:	Pwm_GetVersionInfo	
Syntax:	`void Pwm_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	

Description:	Service returns the version information of this module.

」()

[SWS_Pwm_20069] 「The function Pwm_GetVersionInfo shall be configurable On/Off by the configuration parameter: PwmVersionInfoApi {PWM_VERSION_INFO_API}.」(SRS_BSW_00407)

8.4 Callback notifications

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

8.5 Scheduled functions

All services offered by the PWM Driver are of synchronous nature, with the exception of the asynchronous power transition management, if so configured.
In case the synchronous power transition management is configured, no scheduled API is generated.

8.5.1 Pwm_Main_PowerTransitionManager

[[SWS_Pwm_00189]

「]

Service name:	Pwm_Main_PowerTransitionManager
Syntax:	`void Pwm_Main_PowerTransitionManager( void )`
Service ID[hex]:	0x0d
Description:	This API is cyclically called and supervises the power state transitions, checking for the readiness of the module and issuing the callbacks IoHwAb_Pwm_NotifyReadyForPowerState<Mode> (see PwmPowerStateReadyCbkRef configuration parameter).

」()

[[SWS_Pwm_00190]

「This API executes any non-immediate action needed to finalize a power state transition requested by Pwm_PreparePowerState().」()

[[SWS_Pwm_00191]

「The rate of scheduling shall be defined by Pwm MainSchedulePeriod and shall be variable, as the function only needs to be called if a transition has been requested.」()

[[SWS_Pwm_00192]

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

[[SWS_Pwm_00193]
⌈In case the PWM module is not initialized, this function shall simply return without any further elaboration. This is needed to avoid to elaborate uninitialized variables. No DET error shall be entered, because this condition can easily be verified during the startup phase (tasks started before the initialization is complete).

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

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

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

As this module is part of the MCAL layer, it access directly to the microcontroller registers and therefore doesn't need any lower interfaces.

8.6.2 Optional Interfaces

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

[SWS_Pwm_00104] ⌈

API function	***Description***
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function. OBD Events Suppression shall be ignored for this computation.
Det_ReportError	Service to report development errors.

⌋ ()

8.6.3 Configurable interfaces

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

[SWS_Pwm_00105] ⌈

Service name:	Pwm_Notification_<#Channel>
Syntax:	`void Pwm_Notification_<#Channel>( void )`
Sync/Async:	Synchronous
Reentrancy:	PWM user implementation dependant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The Pwm module shall call the function Pwm_Notification_<#Channel> accordingly to the last call of Pwm_EnableNotification for channel <#Channel>.

⌋ ()

[SWS_Pwm_00025] ⌈The Pwm module shall call the function Pwm_Notification_<#Channel> accordingly to the last call of Pwm_EnableNotification and Pwm_DisableNotification for channel <#Channel>.⌋ (SRS_SPAL_00157)

[SWS_Pwm_00026] ⌈The Pwm module shall reset the interrupt flag associated to the notification Pwm_Notification_<#Channel>⌋ (SRS_SPAL_12129)

[SWS_Pwm_10115] ⌈The Pwm module shall provide the functionality of Pwm_EnableNotification only when the configuration parameter PwmNotificationSupported is ON. ⌋ ()

[SWS_Pwm_20115] ⌈The Pwm module shall provide the functionality of Pwm_DisableNotification only when the configuration parameter PwmNotificationSupported is ON. ⌋ ()

[SWS_Pwm_30115] ⌈The Pwm module shall reset the interrupt flag associated to the notification only when the configuration parameter PwmNotificationSupported is ON. ⌋ ()

[[SWS_Pwm_00198]

]⌈

Service name:	IoHwAb_Pwm_NotifyReadyForPowerState<#Mode>
Syntax:	`void IoHwAb_Pwm_NotifyReadyForPowerState<#Mode>( void )`

Service ID[hex]:	0x60
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The API shall be invoked by the PWM Driver when the requested power state preparation for mode <#Mode> is completed.

」()

[[SWS_Pwm_00199]]

「In case the PWM Driver is configured to support power state management with asynchronous transitions, this API shall be called to signal completion of the power transition preparation phase to the IoHwAbs module.

This is a callback, this API is to be implemented in the IoHwAbs component.」()

8.7 API parameter checking

[SWS_Pwm_10051] 「If development error detection for the Pwm module is enabled, and a development error occurs, then the corresponding PWM function shall report the error to the Development Error Tracer. 」(SRS_BSW_00323, SRS_BSW_00386)

[SWS_Pwm_20051] 「If development error detection for the Pwm module is enabled, and a development error occurs, then the corresponding PWM function shall skip the desired functionality in order to avoid any corruptions of data or hardware registers leaving the function without any actions. 」(SRS_BSW_00323, SRS_BSW_00386)

[SWS_Pwm_30051] 「If development error detection for the Pwm module is enabled, and a development error occurs, then the corresponding PWM function shall return PWM_LOW for the function Pwm_GetOutputState. 」(SRS_BSW_00323, SRS_BSW_00386)

[SWS_Pwm_00117] 「If development error detection for the Pwm module is enabled: if any function (except Pwm_Init) is called before Pwm_Init has been called, the called function shall raise development error `PWM_E_UNINIT`. 」(SRS_BSW_00406, SRS_BSW_00323, SRS_BSW_00386)

[SWS_Pwm_00045] 「If development error detection for the Pwm module is enabled: The API `Pwm_SetPeriodAndDuty()` shall check if the given PWM channel is of the channel class type `PWM_VARIABLE_PERIOD`. If this is not the case the development error

PWM_E_PERIOD_UNCHANGEABLE shall be called. ⌋ (SRS_BSW_00323, SRS_BSW_00386)

[SWS_Pwm_00047] ⌈If development error detection for the Pwm module is enabled: the PWM functions shall check the parameter ChannelNumber and raise development error PWM_E_PARAM_CHANNEL if the parameter ChannelNumber is invalid. ⌋ (SRS_BSW_00323, SRS_BSW_00386)

9 Sequence diagrams

9.1 Initialization

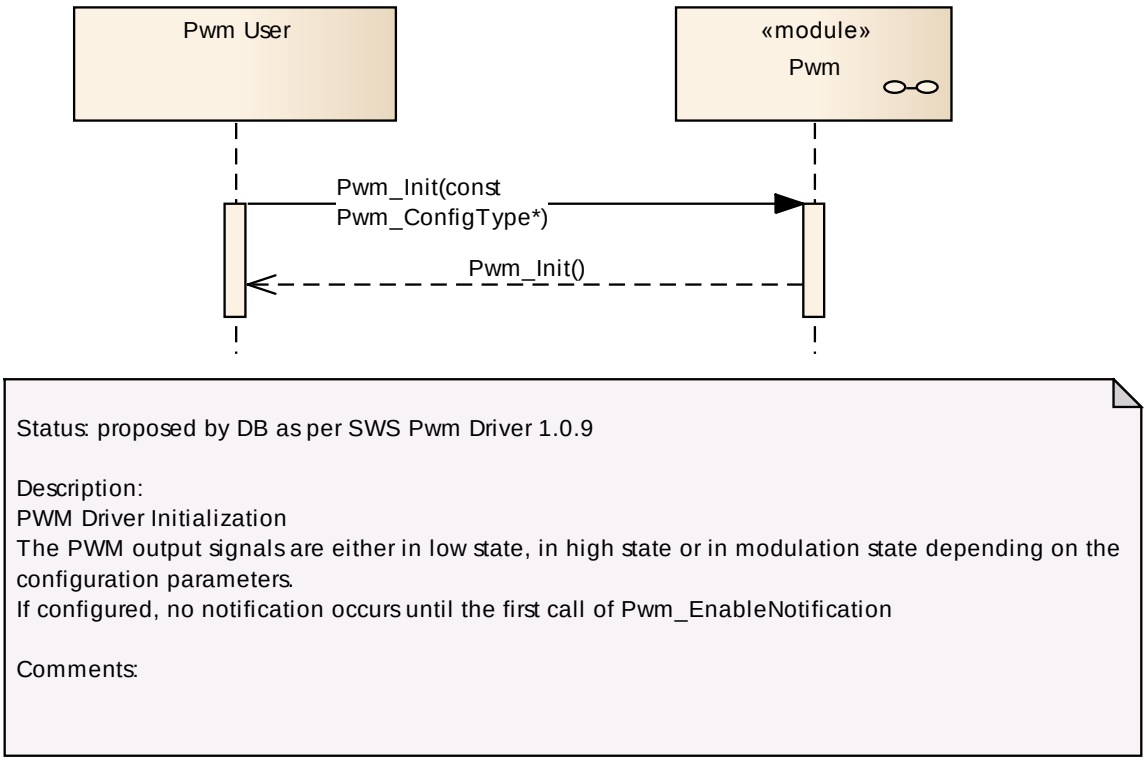


Figure 3: Pwm initialization

9.2 De-initialization

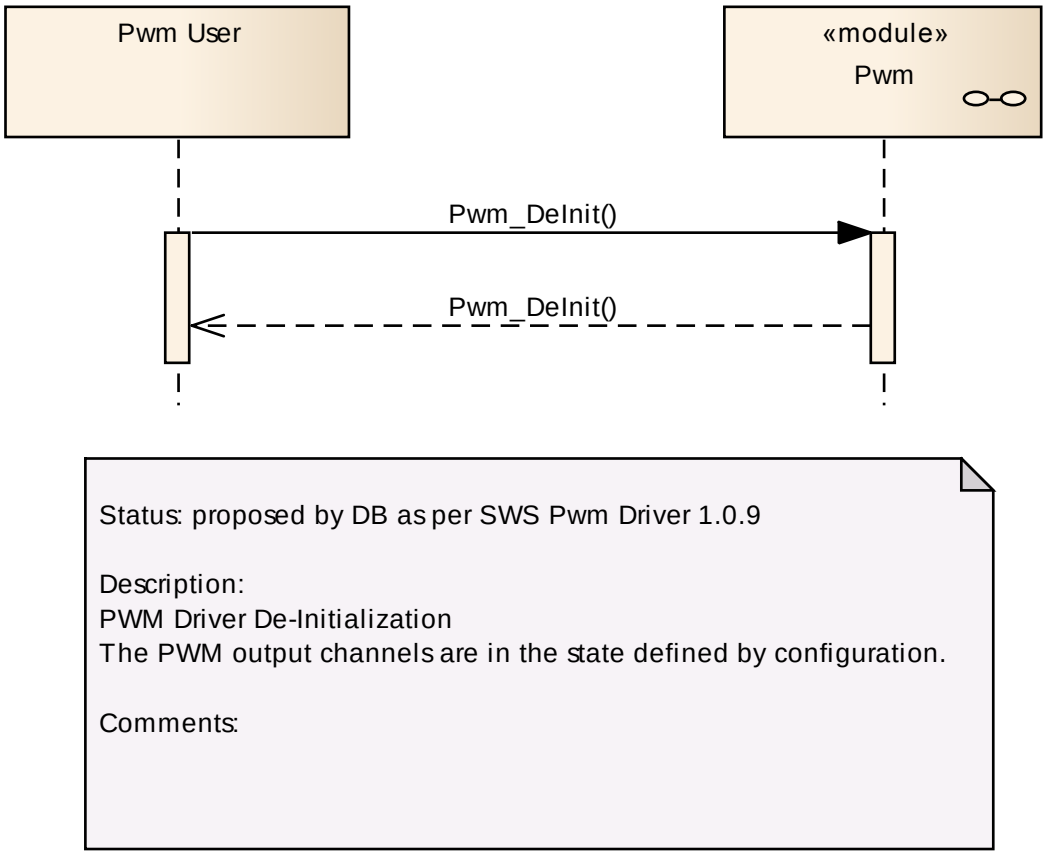


Figure 4: Pwm de-initialization

9.3 Setting the duty cycle

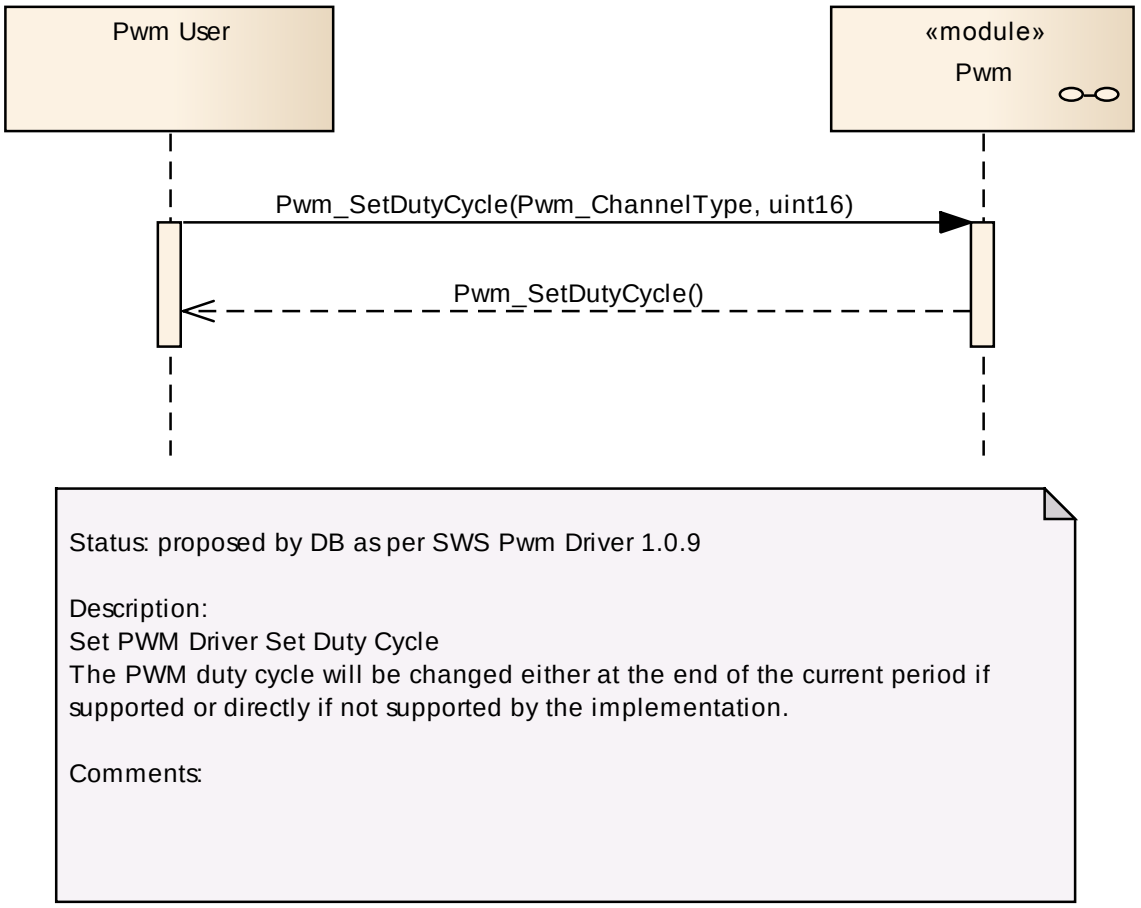


Figure 5: Setting the duty cycle

9.4 Setting the period and the duty

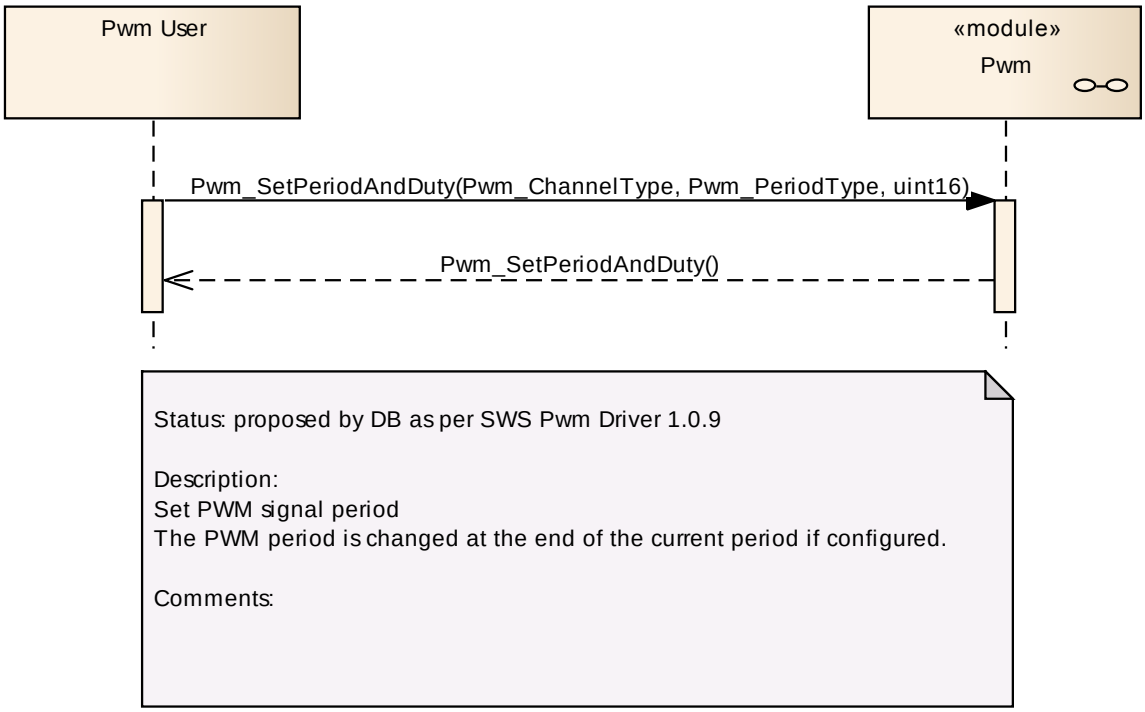


Figure 6: Setting period and duty cycle

9.5 Setting the PWM output to idle

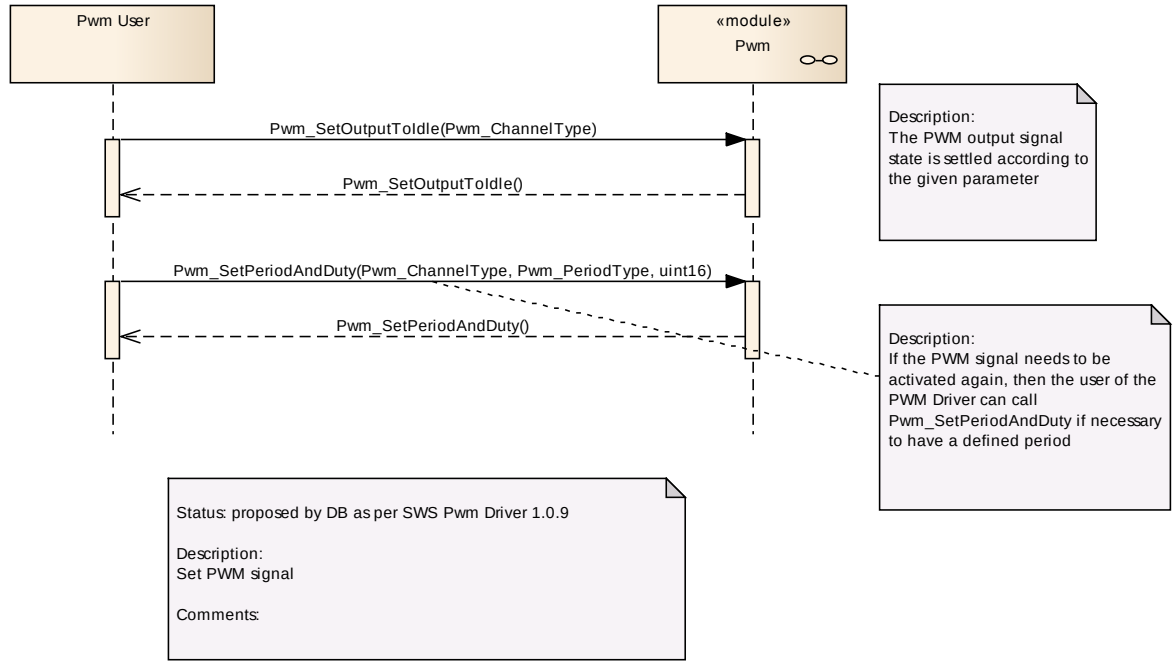


Figure 7: Setting Pwm output to idle

9.6 Getting the PWM Output state

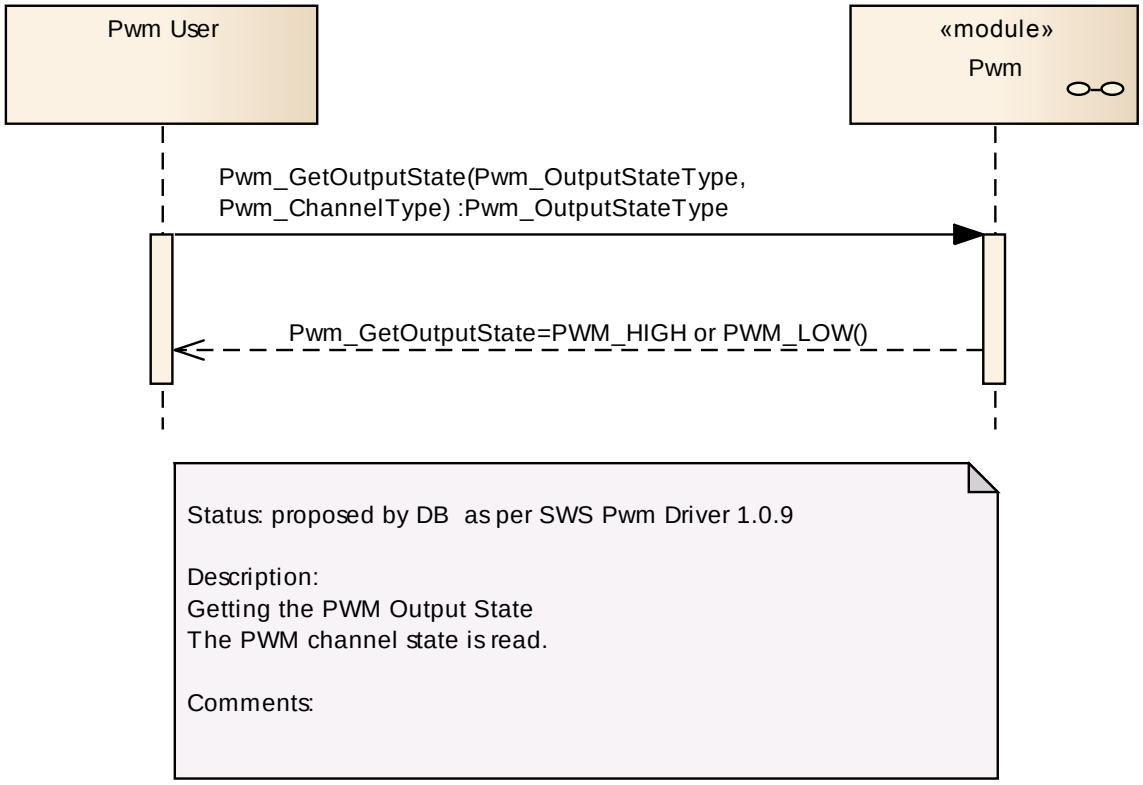


Figure 8: Getting Pwm output state

9.7 Using the PWM notifications

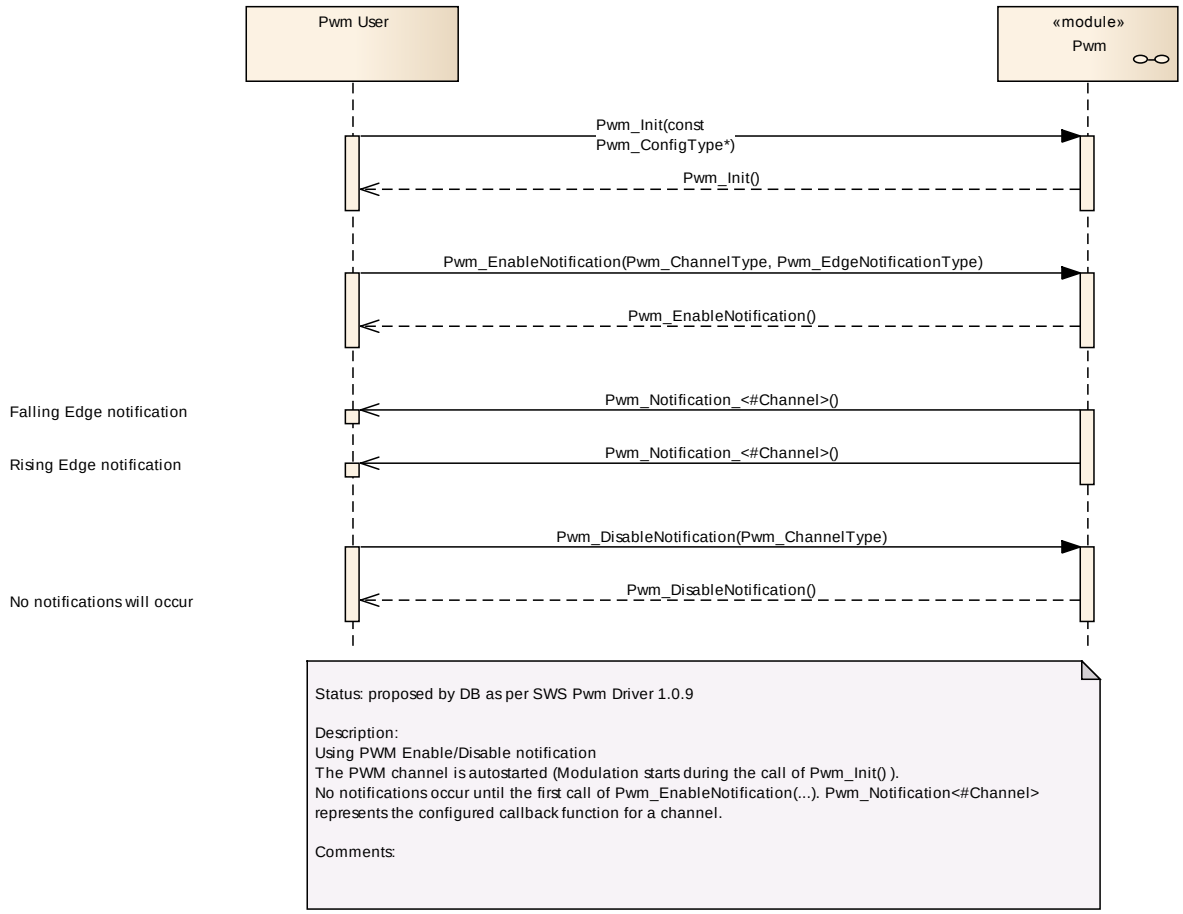


Figure 9: Using Pwm notifications

10 Configuration specification

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

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

Chapter 10.3 specifies published information of the module PWM Driver.

10.1 How to read this chapter

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

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters Functional specification and Chapter API specification.

10.2.1 Variants

[SWS_Pwm_00079] ⌈VARIANT-PRE-COMPILE (Pre Compile) is limited to pre-compile configuration parameters only.⌋()

[SWS_Pwm_00077] ⌈VARIANT-POST-BUILD includes a mix of pre-compile, link time and post build configuration parameters.⌋()

10.2.2 Pwm

Module Name	*Pwm*
Module Description	Configuration of Pwm (Pulse Width Modulation) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
PwmChannelConfigSet	1	Multiple Configuration Set Container
PwmConfigurationOfOptApiServices	1	--
PwmGeneral	1	--

10.2.3 PwmGeneral

SWS Item	**ECUC_Pwm_00004 :**		
Container Name	PwmGeneral{PwmModuleConfiguration}		
Description	--		
Configuration Parameters			

SWS Item	**ECUC_Pwm_00131 :**		
Name	PwmDevErrorDetect {PWM_DEV_ERROR_DETECT}		
Description	Switch for enabling the development error detection.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00132 :**		
Name	PwmDutycycleUpdatedEndperiod {PWM_DUTYCYCLE_UPDATED_ENDPERIOD}		
Description	Switch for enabling the update of the duty cycle parameter at the end of the current period. TRUE: update of duty cycle is done at the end of period of currently generated waveform (current waveform is finished). FALSE: update of duty cycle is done immediately (just after service call, current waveform is cut).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00139 :**		
Name	PwmIndex		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00142 :**		
Name	PwmLowPowerStatesSupport {PWM_LOW_PWR_STATES_SUPPORT}		
Description	Adds / removes all power state management related APIs (PWM_SetPowerState, PWM_GetCurrentPowerState, PWM_GetTargetPowerState, PWM_PreparePowerState, PWM_Main_PowerTransitionManager), indicating if the HW offers low power state management.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00133 :**		
Name	PwmNotificationSupported {PWM_NOTIFICATION_SUPPORTED}		
Description	Switch to indicate that the notifications are supported		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00134 :**		
Name	PwmPeriodUpdatedEndperiod {PWM_DUTY_PERIOD_UPDATED_ENDPERIOD}		
Description	Switch for enabling the update of the period parameter at the end of the current period. TRUE: update of period/duty cycle is done at the end of period of currently generated waveform (current waveform is finished). FALSE: update of period/duty cycle is done immediately (just after service call, current waveform is cut).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00143 :**		
Name	PwmPowerStateAsynchTransitionMode {PWM_ASYNCH_PWR_STATE_TRANS}		
Description	Enables / disables support of the PWM Driver to the asynchronous power state transition.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter shall only be configured if the parameter PwmLowPowerStatesSupport is set to true.		

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

10.2.4 PwmPowerStateConfig

SWS Item	**ECUC_Pwm_00144 :**

Container Name	PwmPowerStateConfig{PWM_PWR_STATE_CONFIG}
Description	Each instance of this parameter defines a power state and the callback to be called when this power state is reached.
Configuration Parameters	

SWS Item	**ECUC_Pwm_00146 :**		
Name	PwmPowerState {PWM_PWR_STATE}		
Description	Each instance of this parameter describes a different power state supported by the PWM HW. It should be defined by the HW supplier and used by the PWMDriver to reference specific HW configurations which set the PWM HW module in the referenced power state. At least the power mode corresponding to full power state shall be always configured.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 18446744073709551615		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter shall only be configured if the parameter PwmLowPowerStatesSupport is set to true.		

SWS Item	**ECUC_Pwm_00145 :**		
Name	PwmPowerStateReadyCbkRef {PWM_PWR_STATE_READY_CBK_REF}		
Description	Each instance of this parameter contains a reference to a power mode callback defined in a CDD or IoHwAbs component.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter shall only be configured if the parameter PwmLowPowerStatesSupport is set to true.		

No Included Containers

10.2.5 PwmChannel

SWS Item	**ECUC_Pwm_00027 :**
Container Name	PwmChannel{PwmChannelConfiguration}
Description	Configuration of an individual PWM channel.
Configuration Parameters	

SWS Item	**ECUC_Pwm_00136 :**
Name	PwmChannelClass {PWM_CHANNEL_CLASS}
Description	Class of PWM Channel. ImplementationType: Pwm_ChannelClassType
Multiplicity	0..1

Type	EcucEnumerationParamDef		
Range	PWM_FIXED_PERIOD	Only the duty cycle can be changed.	
	PWM_FIXED_PERIOD_SHIFTED	Only the duty cycle can be changed. The period is shifted (only if supported by hardware)	
	PWM_VARIABLE_PERIOD	Duty Cycle and period can be changed.	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00137 :**		
Name	PwmChannelId		
Description	Channel Id of the PWM channel. This value will be assigned to the symbolic name derived of the PwmChannel container short name.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00138 :**		
Name	PwmDutycycleDefault {PWM_DUTYCYLE_DEFAULT}		
Description	Value of duty cycle used for Initialization 0, represents 0% 0x8000 represents 100%		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 32768		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00122 :**		
Name	PwmIdleState {PWM_IDLE_STATE}		
Description	The parameter PWM_IDLE_STATE represents the output state of the PWM after the signal is stopped (e.g. call of Pwm_SetOutputToIdle).		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	PWM_HIGH	The PWM channel output will be set to high (3 or 5 V) in idle state.	
	PWM_LOW	The PWM channel output will be set to low (0 V) in idle state.	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00123 :**
Name	PwmNotification {Pwm_Notification}

Description	Definition of the Callback function.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	"NULL"		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00124 :**		
Name	PwmPeriodDefault {PWM_PERIOD_DEFAULT}		
Description	Value of period used for Initialization.(in seconds).		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00125 :**		
Name	PwmPolarity {PWM_POLARITY}		
Description	Defines the starting polarity of each PWM channel.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	PWM_HIGH	The PWM channel output is high at the beginning of the cycle and then goes low when the duty count is reached.	
	PWM_LOW	The PWM channel output is low at the beginning of the cycle and then goes high when the duty count is reached.	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00147 :**		
Name	PwmMcuClockReferencePoint		
Description	This parameter contains reference to the McuClockReferencePoint		
Multiplicity	1		
Type	Reference to [McuClockReferencePoint]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.2.6 PwmChannelConfigSet

SWS Item	**ECUC_Pwm_00140 :**
Container Name	PwmChannelConfigSet [Multi Config Container]
Description	Multiple Configuration Set Container

Configuration Parameters

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
PwmChannel	1..*	Configuration of an individual PWM channel.

10.2.7 PwmConfigurationOfOptApiServices

SWS Item	**ECUC_Pwm_00126 :**
Container Name	PwmConfigurationOfOptApiServices
Description	--
Configuration Parameters	

SWS Item	**ECUC_Pwm_00141 :**		
Name	PwmDeInitApi {PWM_DE_INIT_API}		
Description	Adds / removes the service Pwm_DeInit() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00127 :**		
Name	PwmGetOutputState {PWM_GET_OUTPUT_STATE_API}		
Description	--		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00128 :**		
Name	PwmSetDutyCycle {PWM_SET_DUTY_CYCLE_API}		
Description	--		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00129 :**		
Name	PwmSetOutputToIdle {PWM_SET_OUTPUT_TO_IDLE_API}		
Description	--		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00130 :**		
Name	PwmSetPeriodAndDuty {PWM_SET_PERIOD_AND_DUTY_API}		
Description	--		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Pwm_00135 :**		
Name	PwmVersionInfoApi {PWM_VERSION_INFO_API}		
Description	Switch to indicate that the Pwm_ GetVersionInfo is supported		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.3 Published Information

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

11 Not applicable requirements

[SWS_Pwm_00153] ⌈These requirements are not applicable to this specification.⌋

(SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00170, SRS_BSW_00419, SRS_BSW_00383, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, BSW00431, SRS_BSW_00432, SRS_BSW_00433, BSW00434, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00326, SRS_BSW_00342, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00300, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00305, SRS_BSW_00307, SRS_BSW_00310, SRS_BSW_00373, SRS_BSW_00327, SRS_BSW_00335, SRS_BSW_00350, SRS_BSW_00408, SRS_BSW_00410, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00357, SRS_BSW_00377, SRS_BSW_00304, SRS_BSW_00355, SRS_BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00371, SRS_BSW_00358, SRS_BSW_00414, SRS_BSW_00376, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00329, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00009, SRS_BSW_00401, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00003, SRS_BSW_00341, SRS_BSW_00334, SRS_SPAL_12267, SRS_SPAL_12461, SRS_SPAL_12462, SRS_SPAL_12463, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_12169, SRS_SPAL_12075, SRS_SPAL_12064, SRS_SPAL_12067, SRS_SPAL_12077, SRS_SPAL_12078, SRS_SPAL_12092, SRS_SPAL_12265, SRS_Pwm_12379)