Document Title	Specification of OCU Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	615
Document Classification	Standard
Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History		
Release	**Changed by**	**Change Description**
4.2.2	AUTOSAR Release Management	• DET has been renamed. • SWS_Ocu_00041 and SWS_Ocu_00042 requirements are removed. • OCU_E_PARAM_CONFIG is removed. Added OCU_E_INIT_FAILED • Invalid requirement IDs: Updated SWS_Ocu_156, SWS_Ocu_169
4.2.1	AUTOSAR Release Management	• Set the postBuildVariantValue and postBuildVariantMultiplicity to false and also set the valueConfigClass and the multiplicityConfigClass for all variants to preCompile. • Removal of automatically supported BSW requirement. Reference to SWS_BSW_00380 is removed.
4.1.2	AUTOSAR Release Management	• Minor update of the document structure • Editorial changes • Removed chapter(s) on change documentation
4.1.1	AUTOSAR Administration	• Initial Release

Disclaimer

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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 OCU driver.

Each OCU software channel is linked to a hardware OCU peripheral which belongs to the microcontroller. An output pin can be optionally attached to this channel.

The driver provides functions for initialization and control of the microcontroller internal OCU functionality (Output Compare Unit). The OCU driver allows comparing and acting automatically when the value of a counter matches a defined threshold.The OCU driver provides services and configuration parameters for:

- Starting and stopping a comparison process
- Setting comparison threshold
- Enabling and disabling notification mechanisms
- Getting counter values
- Changing output pin states
- Triggering some hardware resources (ADC, DMA) if available.

The tick duration of a channel counter depends on the channel specific settings (part of OCU driver) as well as on the system clock and settings of the clock tree controlled by the MCU module. The tick duration is not limited by this specification.

Some microcontrollers don't have a dedicated OCU hardware cell, but instead a generic timer module that can be configured to provide the OCU functionality and other timer functionalities as well. This specification does not assume the hardware architecture. Instead; it defines parameters and APIs so that they can be implemented on any suitable hardware architecture. The picture below shows a typical representation of an OCU channel.
The 'output' is the action that is actually done upon compare match.

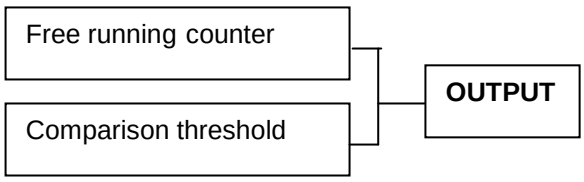


Figure 1: Abstract view of an OCU channel

2 Acronyms and abbreviations

Acronyms and abbreviations that have a local scope appear in the glossary below. Those that have a global scope are contained in the AUTOSAR glossary.

Acronym/Abbreviation	***Description***
OCU	Output Compare Unit
DMA	Direct Memory Access
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
DEM	Diagnostic Event Manager.
DET	Default Error Tracer.
SPAL	Standard Peripheral Abstraction Layer
MCU	Microcontroller Unit.
ISR	Interrupt Service Routine.

Term definition:	***Description:***
OCU channel	Represents a logical entity composed of a free running counter a comparison threshold and the action that is done as a result of the comparison process.
Compare threshold.	Target value that is compared with the content of the counter each time the counter is increased by one unit.
Free running counter	A counter that runs from a minimum (respectively a maxium) to a maximum (respectively a minimum) value and restarts automatically from the minimum (respectively a maximum) after reaching the maximum (respectively the minimum) value.
Reference Interval	Interval (in ticks) given by the caller of the *Ocu_SetAbsolutThreshold* API, and used as base to compute the return information.

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 Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.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.pdf

4 Constraints and assumptions

4.1 Assumptions

4.1.1 Clock

The driver does not support dynamic changes of the clock.
Since the system clock is fully managed by the MCU module, any dynamic change in the system clock settings will impact this module.
The module does not run in the sleep mode.

4.1.2 Resources

The allocation of resources is made exclusively by SW or HW to avoid shared resource issues.
e.g: usage of the API `Ocu_SetPinState`. This API cannot be called to change the state of a pin for a channel that is in the RUNNING state, otherwise there might be a conflict between the state set automatically by the hardware upon compre match and the one set by the API.

4.1.3 Counting and comparing

Our assumption is that the hardware that will operate this driver has the following counter abstraction model (example for an eight-bit counter).

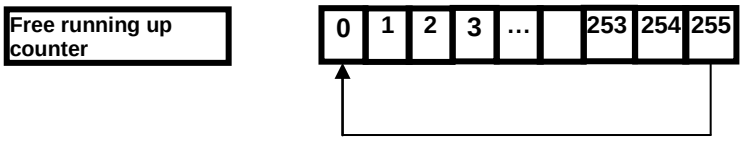


Figure 2: Abstraction model of the free running counter for this driver

Minimum value is 0
Maximum value is 255
The counter is reloaded with 0 when it exceeds the maximum value. That means it has 256 count steps.

Due to the quantization of counting, two different cases are possible when comparing the content of the counter with the threshold. The comparaison can occur when entering a state of the counter or while exiting from a state, as shown in the picture below

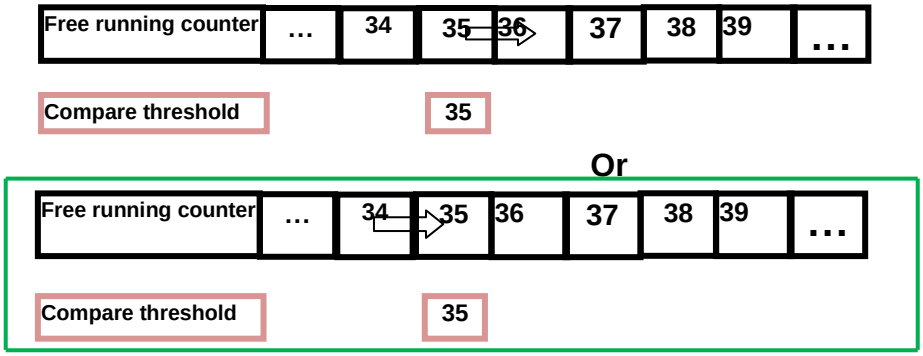


Figure 3: Abstraction model of the comparison process expected in driver.

The expected behavior of this driver is to have the comparison done **on entering** the state represented by the threshold.

4.2 Limitations

No limitations.

4.3 Applicability to car domains

No restrictions.

5 Dependencies to other modules

Module DET
If default error detection is enabled for the OCU driver, then the driver shall raise errors to the Default Error Tracer (DET) whenever a development error is encountered by this module.

Module DEM
The OCU driver shall report production errors to the Diagnostic Event Manager (DEM).

Module MCU Driver
The Microcontroller Unit Driver (MCU Driver) is primarily responsible for initializing and controlling the chip internal clock sources and clock prescalers. The OCU depends on the system clock. Thus, changes of the system clock (e.g. PLL on → PLL off) also affect the clock settings of the OCU hardware.
The MCU driver will set global prescalers, and the OCU clock.The OCU driver will not take care of setting the registers that configure the global clock, global prescalers and PLL in its initialization function. This has to be done by the MCU module. The OCU driver only configures local (OCU peripheral specific) resources.
Document [6] AUTOSAR_TPS_ECUConfiguration contains a chapter '4.8 - *Clock Tree Configuration'*, which details the mechanism to deliver reference clock signals to peripherals.

Module PORT
The configuration of port pins used for the OCU as outputs is done by the PORT driver. Hence the PORT driver has to be initialized prior to the use of OCU functions.

5.1 File structure

5.1.1 Code file structure

[SWS_Ocu_00001] ⌈The code file structure shall not be defined completely within this specification. At this point it shall be pointed out that the code-file structure shall include the following files
- Ocu_Lcfg.c – for link time configurable parameters and
- Ocu_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.⌋
(SRS_BSW_00419, SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00314, SRS_BSW_00370)

5.1.2 Header file structure

[SWS_Ocu_00002] ⌈Ocu.h shall include Ocu_Cfg.h.⌋ ()

[SWS_Ocu_00003] ⌈Ocu.h shall include Std_Types.h.⌋ ()

[SWS_Ocu_00004] ⌈Std_Types.h shall include Compiler.h and Platform_Types.h.⌋ ()

[SWS_Ocu_00005] ⌈Ocu_Lcfg.c shall include Ocu.h and Ocu_MemMap.h.⌋ ()

[SWS_Ocu_00006] ⌈Ocu.c shall include Ocu.h, Ocu_MemMap.h, Det.h and SchM_Ocu.h.⌋ ()

[SWS_Ocu_00007] ⌈Ocu_PBcfg.c shall include Ocu_MemMap.h and Ocu.h.⌋ ()

[SWS_Ocu_00008] ⌈Ocu_Irq.c shall include Ocu_MemMap.h and Ocu.h.⌋ ()

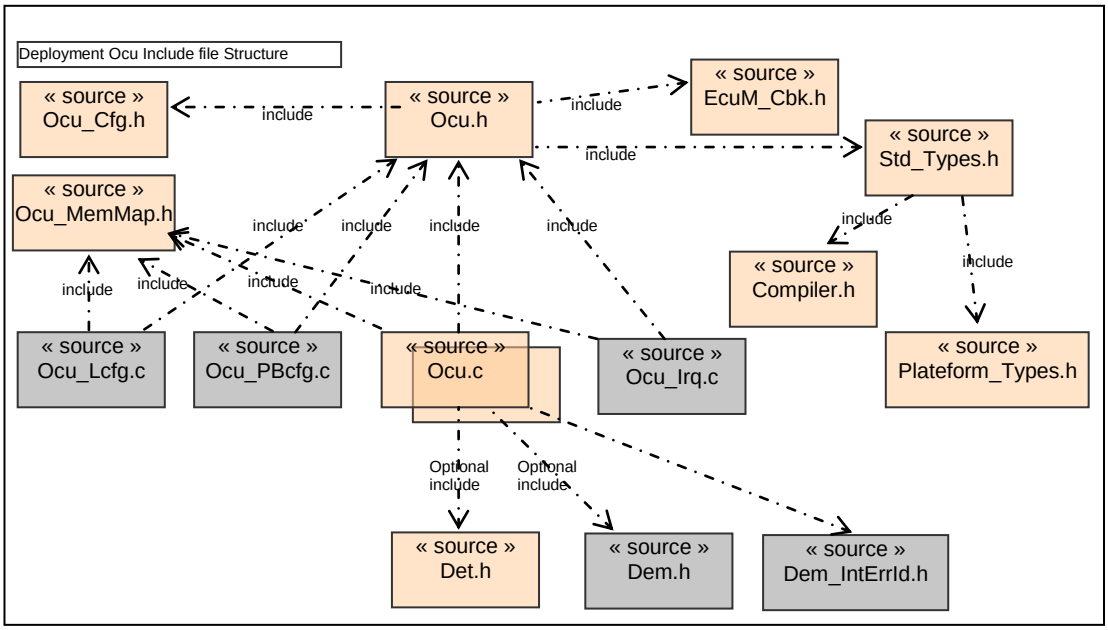


Figure 4: Header file structure

[SWS_Ocu_00009] ⌈The OCU Driver module shall optionally include the Dem.h file if any production error will be issued by the implementation.⌋ (SRS_BSW_00339)

6 Requirements traceability

Requirement	Description	Satisfied by
-	-	SWS_Ocu_00002
-	-	SWS_Ocu_00003
-	-	SWS_Ocu_00004
-	-	SWS_Ocu_00005
-	-	SWS_Ocu_00006
-	-	SWS_Ocu_00007
-	-	SWS_Ocu_00008
-	-	SWS_Ocu_00023
-	-	SWS_Ocu_00024
-	-	SWS_Ocu_00025
-	-	SWS_Ocu_00026
-	-	SWS_Ocu_00027
-	-	SWS_Ocu_00028
-	-	SWS_Ocu_00031
-	-	SWS_Ocu_00032
-	-	SWS_Ocu_00035
-	-	SWS_Ocu_00044
-	-	SWS_Ocu_00048
-	-	SWS_Ocu_00053
-	-	SWS_Ocu_00054
-	-	SWS_Ocu_00060
-	-	SWS_Ocu_00061
-	-	SWS_Ocu_00062
-	-	SWS_Ocu_00063
-	-	SWS_Ocu_00068
-	-	SWS_Ocu_00069
-	-	SWS_Ocu_00078
-	-	SWS_Ocu_00084
-	-	SWS_Ocu_00087
-	-	SWS_Ocu_00093
-	-	SWS_Ocu_00097
-	-	SWS_Ocu_00098
-	-	SWS_Ocu_00101
-	-	SWS_Ocu_00102
-	-	SWS_Ocu_00106
-	-	SWS_Ocu_00107

-	-	SWS_Ocu_00110
-	-	SWS_Ocu_00114
-	-	SWS_Ocu_00117
-	-	SWS_Ocu_00121
-	-	SWS_Ocu_00122
-	-	SWS_Ocu_00125
-	-	SWS_Ocu_00127
-	-	SWS_Ocu_00134
-	-	SWS_Ocu_00135
-	-	SWS_Ocu_00137
-	-	SWS_Ocu_00138
-	-	SWS_Ocu_00169
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_Ocu_00156
SRS_BSW_00004	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	SWS_Ocu_00012
SRS_BSW_00005	Modules of the ÂµC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_Ocu_00156
SRS_BSW_00006	The source code of software modules above the ÂµC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_Ocu_00156
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	SWS_Ocu_00156
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_Ocu_00156
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_Ocu_00156
SRS_BSW_000386	-	SWS_Ocu_00050, SWS_Ocu_00056, SWS_Ocu_00057, SWS_Ocu_00064, SWS_Ocu_00065, SWS_Ocu_00071, SWS_Ocu_00072, SWS_Ocu_00080
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_Ocu_00036
SRS_BSW_00157	-	SWS_Ocu_00133

SRS_BSW_00158	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation	SWS_Ocu_00001
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_Ocu_00156
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_Ocu_00156
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_Ocu_00156
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_Ocu_00156
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_Ocu_00156
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_Ocu_00156
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Ocu_00156
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_Ocu_00156
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_Ocu_00049, SWS_Ocu_00070, SWS_Ocu_00079, SWS_Ocu_00088, SWS_Ocu_00094, SWS_Ocu_00103, SWS_Ocu_00111, SWS_Ocu_00118
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_Ocu_00156
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_Ocu_00156

SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_Ocu_00156
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_Ocu_00156
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	SWS_Ocu_00156
SRS_BSW_00305	Data types naming convention	SWS_Ocu_00156
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Ocu_00156
SRS_BSW_00307	Global variables naming convention	SWS_Ocu_00156
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_Ocu_00156
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_Ocu_00156
SRS_BSW_00310	API naming convention	SWS_Ocu_00156
SRS_BSW_00312	Shared code shall be reentrant	SWS_Ocu_00156
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_Ocu_00001
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_Ocu_00016, SWS_Ocu_00056, SWS_Ocu_00064, SWS_Ocu_00071, SWS_Ocu_00072, SWS_Ocu_00073, SWS_Ocu_00075, SWS_Ocu_00080, SWS_Ocu_00081, SWS_Ocu_00082, SWS_Ocu_00089, SWS_Ocu_00096, SWS_Ocu_00105, SWS_Ocu_00113, SWS_Ocu_00120, SWS_Ocu_00126
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_Ocu_00156
SRS_BSW_00326	-	SWS_Ocu_00156
SRS_BSW_00327	Error values naming convention	SWS_Ocu_00016, SWS_Ocu_00156
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_Ocu_00156

SRS_BSW_00329	-	SWS_Ocu_00156
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_Ocu_00156
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_Ocu_00016, SWS_Ocu_00156
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_Ocu_00156
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_Ocu_00156
SRS_BSW_00335	Status values naming convention	SWS_Ocu_00156
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_Ocu_00046
SRS_BSW_00337	Classification of development errors	SWS_Ocu_00014, SWS_Ocu_00015, SWS_Ocu_00016, SWS_Ocu_00017
SRS_BSW_00338	-	SWS_Ocu_00018, SWS_Ocu_00019, SWS_Ocu_00021
SRS_BSW_00339	Reporting of production relevant error status	SWS_Ocu_00009, SWS_Ocu_00017, SWS_Ocu_00019, SWS_Ocu_00020, SWS_Ocu_00021, SWS_Ocu_00022
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_Ocu_00156
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_Ocu_00156
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_Ocu_00013
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_Ocu_00036
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	SWS_Ocu_00001
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_Ocu_00156
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_Ocu_00156
SRS_BSW_00350	All AUTOSAR Basic Software	SWS_Ocu_00156

	Modules shall apply a specific naming rule for enabling/disabling the detection and reporting of development errors	
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_Ocu_00156
SRS_BSW_00355	-	SWS_Ocu_00156
SRS_BSW_00357	For success/failure of an API call a standard return type shall be defined	SWS_Ocu_00156
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_Ocu_00156
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_Ocu_00128, SWS_Ocu_00156
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_Ocu_00128, SWS_Ocu_00156
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_Ocu_00156
SRS_BSW_00369	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	SWS_Ocu_00018, SWS_Ocu_00019
SRS_BSW_00370	-	SWS_Ocu_00001
SRS_BSW_00371	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	SWS_Ocu_00156
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_Ocu_00156
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_Ocu_00156
SRS_BSW_00376	-	SWS_Ocu_00156
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_Ocu_00156
SRS_BSW_00378	AUTOSAR shall provide a	SWS_Ocu_00156

	boolean type	
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_Ocu_00156
SRS_BSW_00385	List possible error notifications	SWS_Ocu_00016, SWS_Ocu_00017
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_Ocu_00016, SWS_Ocu_00017, SWS_Ocu_00018, SWS_Ocu_00019, SWS_Ocu_00022, SWS_Ocu_00043, SWS_Ocu_00073, SWS_Ocu_00074, SWS_Ocu_00075, SWS_Ocu_00081, SWS_Ocu_00082, SWS_Ocu_00083, SWS_Ocu_00089, SWS_Ocu_00090, SWS_Ocu_00095, SWS_Ocu_00096, SWS_Ocu_00104, SWS_Ocu_00105, SWS_Ocu_00112, SWS_Ocu_00113, SWS_Ocu_00119, SWS_Ocu_00120, SWS_Ocu_00126
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_Ocu_00156
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_Ocu_00036
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_Ocu_00033
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_Ocu_00043, SWS_Ocu_00050, SWS_Ocu_00055, SWS_Ocu_00057, SWS_Ocu_00065, SWS_Ocu_00074, SWS_Ocu_00083, SWS_Ocu_00090, SWS_Ocu_00095, SWS_Ocu_00104, SWS_Ocu_00112, SWS_Ocu_00119
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Ocu_00123, SWS_Ocu_00124
SRS_BSW_00408	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	SWS_Ocu_00156
SRS_BSW_00410	Compiler switches shall have defined values	SWS_Ocu_00156
SRS_BSW_00411	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	SWS_Ocu_00124
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_Ocu_00156
SRS_BSW_00414	Init functions shall have a	SWS_Ocu_00156

	pointer to a configuration structure as single parameter	
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_Ocu_00156
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_Ocu_00156
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_Ocu_00156
SRS_BSW_00419	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	SWS_Ocu_00001
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_Ocu_00022
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_Ocu_00156
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_Ocu_00156
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_Ocu_00156
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_Ocu_00156
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_Ocu_00156
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_Ocu_00156
SRS_BSW_00429	BSW modules shall be only allowed to use OS objects and/or related OS services	SWS_Ocu_00156
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_Ocu_00156

SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_Ocu_00156
SRS_BSW_00434	-	SWS_Ocu_00156
SRS_BSW_00438	Configuration data shall be defined in a structure	SWS_Ocu_00033
SRS_BSW_0431	-	SWS_Ocu_00156
SRS_Ocu_00002	The OCU driver shall support the following basic static configurations per channel	SWS_Ocu_00033, SWS_Ocu_00034
SRS_Ocu_00005	The OCU Driver shall provide the functionality to de-initialize OCU driver	SWS_Ocu_00045
SRS_Ocu_00007	The OCU driver shall allow enabling /disabling notifications for an OCU channel during runtime	SWS_Ocu_00108, SWS_Ocu_00109, SWS_Ocu_00115, SWS_Ocu_00116
SRS_Ocu_00008	The OCU driver shall provide services for starting and stopping a channel	SWS_Ocu_00051, SWS_Ocu_00052, SWS_Ocu_00058, SWS_Ocu_00059
SRS_Ocu_00009	The OCU driver shall provide a synchronous service for reading the value of the counter	SWS_Ocu_00085, SWS_Ocu_00086
SRS_Ocu_00010	The OCU driver shall provide services to modify the value of the threshold of a channel	SWS_Ocu_00091, SWS_Ocu_00092, SWS_Ocu_00100
SRS_Ocu_00011	The OCU driver shall provide a synchronous service to set the state of the output pin attached to a channel	SWS_Ocu_00066, SWS_Ocu_00067
SRS_Ocu_00012	The OCU driver shall provide a service to set the action that will be performed by the pin attached to a channel upon comparison match	SWS_Ocu_00076, SWS_Ocu_00077
SRS_SPAL12448	-	SWS_Ocu_00113
SRS_SPAL_00157	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	SWS_Ocu_00128, SWS_Ocu_00129
SRS_SPAL_12056	All driver modules shall allow the static configuration of notification mechanism	SWS_Ocu_00131, SWS_Ocu_00132
SRS_SPAL_12057	All driver modules shall implement an interface for initialization	SWS_Ocu_00036, SWS_Ocu_00037, SWS_Ocu_00039, SWS_Ocu_00040
SRS_SPAL_12063	All driver modules shall only support raw value mode	SWS_Ocu_00029

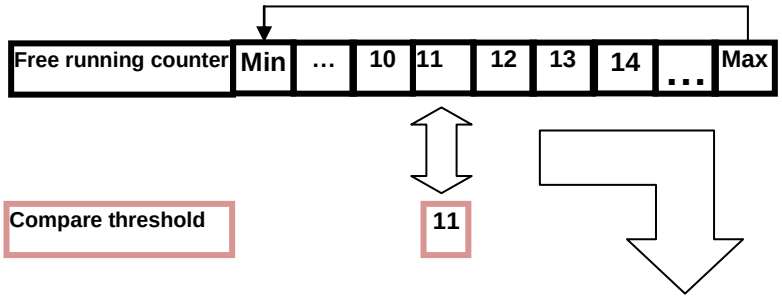
SRS_SPAL_12064	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	SWS_Ocu_00156
SRS_SPAL_12067	All driver modules shall set their wake-up conditions depending on the selected operation mode	SWS_Ocu_00156
SRS_SPAL_12068	The modules of the MCAL shall be initialized in a defined sequence	SWS_Ocu_00156
SRS_SPAL_12069	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	SWS_Ocu_00156
SRS_SPAL_12075	All drivers with random streaming capabilities shall use application buffers	SWS_Ocu_00156
SRS_SPAL_12077	All drivers shall provide a non blocking implementation	SWS_Ocu_00156
SRS_SPAL_12078	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	SWS_Ocu_00156
SRS_SPAL_12092	The driver's API shall be accessed by its handler or manager	SWS_Ocu_00156
SRS_SPAL_12125	All driver modules shall only initialize the configured resources	SWS_Ocu_00010, SWS_Ocu_00011, SWS_Ocu_00037, SWS_Ocu_00136
SRS_SPAL_12129	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	SWS_Ocu_00130
SRS_SPAL_12163	All driver modules shall implement an interface for de-initialization	SWS_Ocu_00046, SWS_Ocu_00047
SRS_SPAL_12169	All driver modules that provide different operation modes shall provide a service for mode selection	SWS_Ocu_00156
SRS_SPAL_12263	The implementation of all driver modules shall allow the configuration of specific module parameter types at link time	SWS_Ocu_00033
SRS_SPAL_12265	Configuration data shall be kept constant	SWS_Ocu_00156
SRS_SPAL_12267	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	SWS_Ocu_00156

SRS_SPAL_12448	All driver modules shall have a specific behavior after a development error detection	SWS_Ocu_00043, SWS_Ocu_00050, SWS_Ocu_00055, SWS_Ocu_00056, SWS_Ocu_00057, SWS_Ocu_00064, SWS_Ocu_00065, SWS_Ocu_00071, SWS_Ocu_00072, SWS_Ocu_00073, SWS_Ocu_00074, SWS_Ocu_00075, SWS_Ocu_00080, SWS_Ocu_00081, SWS_Ocu_00082, SWS_Ocu_00083, SWS_Ocu_00089, SWS_Ocu_00090, SWS_Ocu_00095, SWS_Ocu_00096, SWS_Ocu_00104, SWS_Ocu_00105, SWS_Ocu_00112, SWS_Ocu_00119, SWS_Ocu_00120, SWS_Ocu_00126
SRS_SPAL_12461	Specific rules regarding initialization of controller registers shall apply to all driver implementations	SWS_Ocu_00034, SWS_Ocu_00038, SWS_Ocu_00156
SRS_SPAL_12462	The register initialization settings shall be published	SWS_Ocu_00156
SRS_SPAL_12463	The register initialization settings shall be combined and forwarded	SWS_Ocu_00156

7 Functional specification

7.1 General behavior

The OCU channel is composed of two main elements: a free running counter and a compare threshold. These elements act together to generate actions required by the user**.**The free running counter can be provided by hardaware or software whereas the threshold is a value set by the user. It is then compared with the current content of the counter each time the counter is increased by one unit.



The driver compares both values each time the counter is increased by one unit. In case of equality, two different types of action can be done:

-report the information to the upper layer through a notification function.
-act on a configured output pin

Figure 5: General behavior of the driver

The OCU driver provides the following services for managing a channel:

- Starting a channel
- Stopping a channel
- Setting the comparison threshold value
- Enabling and disabling a notification function for a channel
- Getting counter values
- Changing output pin states

The states and the state transitions of an output compare channel are shown in the figure below.

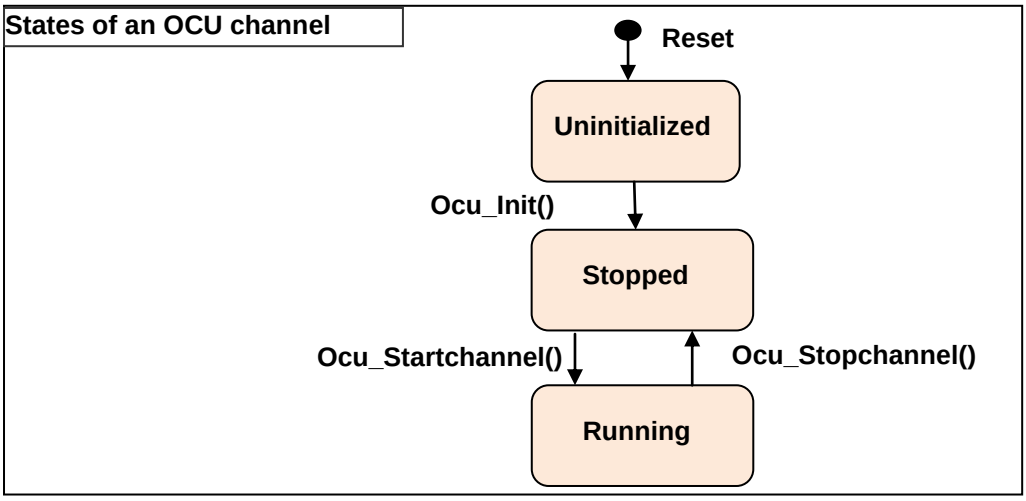


Figure 6: State diagram of an OCU channel

An Ocu channel has a simple state diagram with the states shown above. All the channels of the driver are initialized at once with the API Ocu_Init(). There's no API to inialize individually each channel.

Depending on the hardware architecture, the hardware tied to an Ocu channel may be managed by the OCU cell or any other timer module in the microcontroller.

7.2 Version check

7.2.1 Background & Rationale

The integration of incompatible files is to be avoided. Minimum implementation is the version check of the header file inside the .c file (version numbers of .c and .h files must be identical).

[SWS_Ocu_00012] ⌈The OCU driver shall perform Inter Module Checks to avoid integration of incompatible files.The imported included files shall be checked by preprocessing directives.
The following version numbers shall be verified:
- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION
Where <MODULENAME> is the module short name of the other (external) modules, which provide header files included by the OCU driver.

If the values are not identical to the expected values, an error shall be reported.⌋
(SRS_BSW_00004)

7.3 Time Unit Ticks

7.3.1 Background & Rationale

To get times out of register values it is necessary to know the oscillator frequency, prescalers and some other settings of the whole system clock. 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.

7.3.2 Requirements

[SWS_Ocu_00013] ⌈All time units used within the API services of the OCU driver shall be of the unit ticks.⌋ (SRS_BSW_00343)

7.4 Error classification

7.4.1 Development Errors

[SWS_Ocu_00014] ⌈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.⌋ (SRS_BSW_00337)

[SWS_Ocu_00015] ⌈Development error values are of type uint8. ⌋ (SRS_BSW_00337)

[SWS_Ocu_00016] ⌈The following errors shall be detectable by the OCU driver depending on its build version (development / production mode).⌋ (SRS_BSW_00337, SRS_BSW_00323, SRS_BSW_00327, SRS_BSW_00331, SRS_BSW_00385, SRS_BSW_00386)

Type of error	*Relevance*	*Related error code*	*Value [hex]*	*Requirement*
API services other than **Ocu_GetVersionInfo()** and **Ocu_init()** used without module initialization	Development	OCU_E_UNINIT	0x02	#SWS_Ocu_00050 SWS_Ocu_00057 SWS_Ocu_00065 SWS_Ocu_00074 SWS_Ocu_00083 SWS_Ocu_00090 SWS_Ocu_00095 SWS_Ocu_00104 SWS_Ocu_00112 SWS_Ocu_00119
API service used with an invalid channel Identifier.	Development	OCU_E_PARAM_INVALID_CHANNEL	0x03	SWS_Ocu_00056 SWS_Ocu_00064 SWS_Ocu_00071 #SWS_Ocu_00080 SWS_Ocu_00089 SWS_Ocu_00096 SWS_Ocu_001052
API **Ocu_SetPinState()** called with an invalid pin state or when the channel is in the RUNNING state..	Development	OCU_E_PARAM_INVALID_STATE	0x04	SWS_Ocu_00073 SWS_Ocu_00075 SWS_Ocu_00137
API **Ocu_SetPinAction()** called with an invalid pin action.	Development	OCU_E_PARAM_INVALID_ACTION	0x05	SWS_Ocu_00082
Usage of **Ocu_DisableNotification()** or **Ocu_EnableNotification()** on a channel where a NULL pointer is configured as the notification function.	Development	OCU_E_NO_VALID_NOTIF	0x06	SWS_Ocu_00114 SWS_Ocu_00121
API **Ocu_Init()** called while the OCU driver has already been initialized	Development	OCU_E_ALREADY_INITIALIZED	0x07	SWS_Ocu_00043
API **Ocu_GetVersionInfo()** is called with a NULL parameter.	Development	OCU_E_PARAM_POINTER	0x08	SWS_Ocu_00126
API **Ocu_StartChannel()** called on a channel that is in state RUNNING.	Development	OCU_E_BUSY	0x09	SWS_Ocu_00055
Ocu_SetPinState() or **Ocu_SetPinAction()** called for a channel that doesn't have an associated output pin.	Development	OCU_E_PARAM_NO_PIN	0x0A	SWS_Ocu_00072 SWS_Ocu_00081
OCU initialization has been failed, e.g. selected configuration set doesn't exist.	Development	OCU_E_INIT_FAILED	0x0B	
--	Production	--	Assigned externally	

Table 1: Error Classification

[SWS_Ocu_00017] ⌈Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the OCU device specific implementation specification. The classification and enumeration shall be compatible to the errors listed above.⌋ (SRS_BSW_00337, SRS_BSW_00339, SRS_BSW_00385, SRS_BSW_00386)

7.4.2 Runtime Errors

<In case there are no runtime errors, please state the following sentence:
< There are no runtime errors.>

[SWS_Ocu_XXXXX] Runtime Error Types

Type of error	*Related error code*	*Value [hex]*

⌋ ( )

7.4.3 Transient Faults

<In case there are no transient faults, please state the following sentence:
< There are no transient faults.>

[SWS_Ocu_XXXXX] Transient Faults Types

Type of error	*Related error code*	*Value [hex]*

⌋ ( )

7.4.4 Production Errors

This module does not specify any production errors.

7.5 Error Detection

[SWS_Ocu_00018] ⌈The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time. The switch `OcuDevErrorDetectApi` shall activate or deactivate the detection of all development errors.⌋ (SRS_BSW_00338, SRS_BSW_00369, SRS_BSW_00386)

[SWS_Ocu_00019] ⌈If the switch `OcuDevErrorDetectApi` is enabled, then API parameter checking is enabled. The detailed description of the detected errors can

be found in chapter Error classification and chapter API specification.⌋
(SRS_BSW_00386, SRS_BSW_00338, SRS_BSW_00369, SRS_BSW_00339)

[**SWS_Ocu_00020]** ⌈The detection of production errors cannot be switched off.⌋
(SRS_BSW_00339)

7.6 Error Notification

[SWS_Ocu_00021] ⌈Detected development errors shall be reported with the service *Det_ReportError* of the Default Error Tracer (DET) if the pre-processor switch *OcuDevErrorDetectApi* is set.⌋ (SRS_BSW_00338, SRS_BSW_00339)

[SWS_Ocu_00022] ⌈Production errors shall be reported to Diagnostic Event Manager via the API *Dem_ReportErrorStatus*. ⌋ (SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00386)

7.7 Debug Support

[SWS_Ocu_00023] ⌈Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable. ⌋ ()

[SWS_Ocu_00024] ⌈All type definitions of variables which shall be debugged shall be accessible by the header file Ocu.h. ⌋ ()

[SWS_Ocu_00025] ⌈The declaration of variables in the header file shall be such that it is possible to calculate the size of the variables by C-"sizeof".⌋ ()

[SWS_Ocu_00026] ⌈Variables available for debugging shall be described in the respective OCU driver Description. ⌋ ()

8 API specification

8.1 Imported types

This chapter lists all types included from other modules.

[**SWS_Ocu_00027**] ⌈

Module	*Imported Type*
Dem	Dem_EventIdType
	Dem_EventStatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋ ()

8.2 Type definitions

8.2.1 Ocu_ChannelType

[SWS_Ocu_00028] ⌈

Name:	`Ocu_ChannelType`		
Type:	`uint`		
Range:	`8 / 16 / 32 bits`	--	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 an OCU channel.		

⌋ ()

8.2.2 Ocu_ValueType

[SWS_Ocu_00029] ⌈

Name:	`Ocu_ValueType`		
Type:	`uint`		
Range:	`8 / 16 / 32 bits`	--	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:	Type for reading the counter and writing the threshold values (in number of ticks).		

⌋ (SRS_SPAL_12063)

8.2.3 Ocu_PinStateType

[SWS_Ocu_00031] ⌈

Name:	`Ocu_PinStateType`	
Type:	`Enumeration`	
Range:	`OCU_HIGH`	The pin associated to an OCU channel is in high state.
	`OCU_LOW`	The pin associated to an OCU channel is in low state.
Description:	Output state of the pin linked to an OCU channel.	

⌋ ()

8.2.4 Ocu_PinActionType

[SWS_Ocu_00032] ⌈

Name:	Ocu_PinActionType	
Type:	Enumeration	
Range:	OCU_SET_HIGH	The channel pin will be set HIGH upon compare match.
	OCU_SET_LOW	The channel pin will be set LOW upon compare match.
	OCU_TOGGLE	The channel pin will be set to the opposite of its current level HIGH upon compare match.
	OCU_DISABLE	The channel pin will remain at its current level upon compare match.
Description:	Automatic action (by hardware) to be performed on a pin attached to an OCU channel.	

⌋ ()

8.2.5 Ocu_ConfigType

[SWS_Ocu_00033] ⌈

Name:	Ocu_ConfigType	
Type:	Structure	
Range:	Hardware dependent	The contents of the initialization data structure are hardware specific.
Description:	This is the type of the data structure containing the initialization data for the OCU driver.	

⌋ (SRS_Ocu_00002, SRS_SPAL_12263, SRS_BSW_00405, SRS_BSW_00438)

[SWS_Ocu_00034] ⌈Ocu_ConfigType is a type of data structure containing the initialization data for the OCU driver.

Mandatory parameters:

- Symbolic name for channel / channel ID
- maximum value of the counter
- Time resolution in number of ticks
- Notification function
- Default value of the threshold
- Minimum value of the counter

Optional parameters (if supported by hardware):

- count direction
- Ouput pin (levels, and possible automatic actions).
- Hardware triggered events (ADC or DMA).
- Microcontroller OCU-specific HW properties (optional prescaler, clock settings if supported by hardware).⌋ (SRS_Ocu_00002, SRS_SPAL_12461)

8.2.6 Ocu_ReturnType

[SWS_Ocu_00138] ⌈

Name:	`Ocu_ReturnType`	
Type:	`Enumeration`	
Range:	`OCU_CM_IN_REF_INTERVAL`	The compare match will occur inside the current Reference Interval.
	`OCU_CM_OUT_REF_INTERVAL`	The compare match will not occur inside the current Reference Interval.
Description:	Return information after setting a new threshold value.	

⌋ ()

8.3 Function definitions

8.3.1 Ocu_Init

[SWS_Ocu_00035] ⌈

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

⌋ ()

[SWS_Ocu_00036] ⌈The function `Ocu_Init` shall initialize all internals variables and the used Ocu structure of the microcontroller according to a configuration set referenced by *ConfigPtr*.⌋ (SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00101, SRS_SPAL_12057)

Note: All the channels are initialized at once by the API `Ocu_Init`. There's no API to individually initialize each channel.

[SWS_Ocu_00010] ⌈If a free-running counter of the OCU cell can be used by another timer module then the Ocu driver must not start nor stop the free-running counter.⌋ (SRS_SPAL_12125)

[SWS_Ocu_00011] ⌈The API Ocu_Init shall start all free-running counters, which are exclusively used by this driver.⌋(SRS_SPAL_12125)

[SWS_Ocu_00037] ⌈the function `Ocu_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)

The following rules regarding initialization of controller registers shall apply to this driver implementation:

- **[SWS_Ocu_00038]** ⌈If the hardware allows for only one usage of the register (register dedicated only to the OCU resource), then the OCU driver is responsible for initializing the register.⌋ (SRS_SPAL_12461)
- **Note1:** If the register can affect several hardware modules and if it is not an I/O register it shall be initialized by the MCU driver. (SRS_SPAL_12461)
- **Note2:** One-time writable registers that require initialization directly after reset shall be initialized by the start-up code. (SRS_SPAL_12461)
- **Note3:** All other registers shall be initialized by the startup code. (SRS_SPAL_12461).
- **Note4:** If a register can affect several hardware modules and if it is an I/O register it shall be initialized by the PORT driver. (SRS_SPAL_12461)

[SWS_Ocu_00039] ⌈The function `Ocu_Init shall` stop all channels.⌋ (SRS_SPAL_12057).

[SWS_Ocu_00040] ⌈The function `Ocu_Init` shall disable all notifications.⌋ (SRS_SPAL_12057)

The reason is that the users of these notifications may not be ready. They can call Ocu_EnableNotification() to start getting notifications.

[SWS_Ocu_00043] ⌈If default error detection is enabled for the OCU driver and the function `Ocu_Init` is called when the OCU driver and hardware are already initialized, the function `Ocu_Init` shall raise development error `OCU_E_ALREADY_INITIALIZED` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448)

[SWS_Ocu_00044] ⌈A re-initialization of the OCU driver by executing the function `Ocu_Init` requires a de-initialization before by executing the function `Ocu_DeInit`.⌋ ()

8.3.2 Ocu_DeInit

[SWS_Ocu_00045] ⌈

Service name:	Ocu_DeInit
Syntax:	`void Ocu_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:	This function de-initializes the OCU module.

⌋ (SRS_Ocu_00005)

[SWS_Ocu_00046] ⌈The function `Ocu_DeInit` shall deinitialize the OCU variables and registers that were initialized by `Ocu_Init` to a state comparable to their power on reset state. Values of registers which are not writeable are excluded.⌋ (SRS_BSW_00336, SRS_SPAL_12163)

Note: It's the responsibility of the hardware design that the state does not lead to undefined activities in the µC.

[SWS_Ocu_00047] ⌈The function `Ocu_DeInit` shall disable all used interrupts and notifications.⌋ (SRS_SPAL_12163)

[SWS_Ocu_00048] ⌈The function `Ocu_DeInit` shall influence only the peripherals which are allocated by static configuration and/or the runtime configuration set passed by the previous call of `Ocu_Init()`.⌋ ()

[SWS_Ocu_00136] ⌈The API Ocu_DeInit shall stop all free-running counters, which are exclusively used by this driver.⌋ (SRS_SPAL_12125)

Note: To prevent undefined behaviour during de-initialization, the user must stop all RUNNING channels (by calling the function `Ocu_StopChannel`) before calling the API `Ocu_DeInit`. Hence the requirement below.

[SWS_Ocu_00137] ⌈If default error detection is enabled for the OCU driver: if a channel is still in the RUNNING state when the function `Ocu_DeInit` is called, then the function shall raise the development error '`OCU_E_PARAM_INVALID_STATE`' and return without any action.⌋ ()

[SWS_Ocu_00049] ⌈The function `Ocu_DeInit` shall be pre compile time configurable On/Off by the configuration parameter: OcuDeInitApi {OCU_DE_INIT_API}.⌋ (SRS_BSW_00171).

[SWS_Ocu_00050] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function `Ocu_DeInit` shall raise the error `OCU_E_UNINIT`.⌋ (SRS_BSW_00406, SRS_BSW_000386, SRS_SPAL_12448)

8.3.3 Ocu_StartChannel

[SWS_Ocu_00051] ⌈

Service name:	Ocu_StartChannel	
Syntax:	`void Ocu_StartChannel( Ocu_ChannelType ChannelNumber )`	
Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the OCU
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to start an OCU channel.	

⌋ (SRS_Ocu_00008)

[SWS_Ocu_00052] ⌈The function Ocu_StartChannel shall start an OCU channel by allowing all compare match configured actions to be performed.⌋ (SRS_Ocu_00008)

[SWS_Ocu_00053] ⌈The function Ocu_StartChannel shall be reentrant if it is called for different channels.⌋ ()

[SWS_Ocu_00054] ⌈The state of the selected channel shall be set to "RUNNING" If the function Ocu_StartChannel has been successfully performed.⌋ ()

[SWS_Ocu_00055] ⌈If default error detection is enabled for the OCU driver: If the function Ocu_StartChannel is called on a channel in the state "RUNINNG", then the function shall raise the error `OCU_E_BUSY` and return without any action.⌋ (SRS_BSW_00406, SRS_SPAL_12448).

[SWS_Ocu_00056] ⌈If default error detection is enabled for the OCU driver: If the parameter `ChannelNumber` is invalid (not within the range specified by the configuration), the function Ocu_StartChannel shall raise the error `OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_000386, SRS_SPAL_12448).

[SWS_Ocu_00057] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function Ocu_StartChannel shall raise the error `OCU_E_UNINIT` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_000386, SRS_SPAL_12448).

8.3.4 Ocu_StopChannel

[SWS_Ocu_00058] ⌈

Service name:	Ocu_StopChannel	
Syntax:	`void Ocu_StopChannel( Ocu_ChannelType ChannelNumber )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the OCU
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to stop an OCU channel.	

⌋ (SRS_Ocu_00008)

[SWS_Ocu_00059] ⌈The function Ocu_StopChannel shall stop an OCU channel by halting compare match configured actions for this channel.⌋ (SRS_Ocu_00008)

[SWS_Ocu_00060] ⌈The function Ocu_StopChannel shall not stop the free-running counter associated with a channel.⌋ ()

Note: This is due to the fact that a free-running counter can be associated with more than one Ocu channel. Therefore, stopping that counter will harm the operation of the other channel(s).

[SWS_Ocu_00061] ⌈The function Ocu_StopChannel shall be reentrant if it is called for different channels.⌋ ()

[SWS_Ocu_00062] ⌈The state of the selected channel shall be set to “STOPPED”if the function Ocu_StopChannel is successfully performed.⌋ ()

[SWS_Ocu_00063] ⌈ If the function Ocu_StopChannel is called on a channel in the state "STOPPED", then the function shall leave without any action (no change of the channel state), and shall **not** raise a development error.⌋ ()

[SWS_Ocu_00064] ⌈If default error detection is enabled for the OCU driver: If the parameter `ChannelNumber` is invalid (not within the range specified by the configuration), the function Ocu_StopChannel shall raise the error `OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_000386, SRS_SPAL_12448).

[SWS_Ocu_00065] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function Ocu_StopChannel shall raise the error `OCU_E_UNINIT` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_000386, SRS_SPAL_12448).

8.3.5 Ocu_SetPinState

[SWS_Ocu_00066] ⌈

Service name:	Ocu_SetPinState	
Syntax:	`void Ocu_SetPinState( Ocu_ChannelType ChannelNumber, Ocu_PinStateType PinState )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the OCU
	PinState	OCU_LOW, OCU_HIGH
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to set immediately the level of the pin associated to an OCU channel.	

⌋ (SRS_Ocu_00011)

[SWS_Ocu_00067] ⌈The function Ocu_SetPinState shall set the pin associated with the channel to the level indicated by “PinState”.⌋ (SRS_Ocu_00011)

[SWS_Ocu_00068] ⌈The fuction Ocu_SetPinState shall be reentrant if it is called for different channels.⌋ ()

[SWS_Ocu_00069] ⌈The function Ocu_SetPinState shall be used only if the channel is not in the RUNNING state.⌋ ()

Note: The previous requirerment also means that it shall be possible to alter the state of a STOPPED channel by this API.

[SWS_Ocu_00070] ⌈The function Ocu_SetPinState shall be pre compile time configurable On/Off by the configuration parameter: OcuSetPinStateApi {OCU_SET_PIN_STATE_API}.⌋ (SRS_BSW_00171)

[SWS_Ocu_00071] ⌈If default error detection is enabled for the OCU driver: If the parameter `ChannelNumber` is invalid (not within the range specified by the configuration), the function Ocu_SetPinState shall raise the error `OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_000386, SRS_SPAL_12448)

[SWS_Ocu_00072] ⌈If default error detection is enabled for the OCU driver: If a pin is not associated with the channel (not defined in the configuration of the channel), the function Ocu_SetPinState shall raise the error `OCU_E_PARAM_NO_PIN` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_000386, SRS_SPAL_12448)

[SWS_Ocu_00073] ⌈If default error detection is enabled for the OCU driver: If the parameter `PinState` is invalid (not within the range specified by the configuration), the function Ocu_SetPinState shall raise the error `OCU_E_PARAM_INVALID_STATE` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)

[SWS_Ocu_00074] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function Ocu_SetPinState shall raise the error `OCU_E_UNINIT` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448).

[SWS_Ocu_00075] ⌈If default error detection is enabled for the OCU driver: If the channel is in the RUNNING state, the function Ocu_SetPinState shall raise the error `OCU_E_PARAM_INVALID_STATE` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448).

8.3.6 Ocu_SetPinAction

[SWS_Ocu_00076] ⌈

Service name:	Ocu_SetPinAction
Syntax:	`void Ocu_SetPinAction( Ocu_ChannelType ChannelNumber, Ocu_PinActionType PinAction )`
Service ID[hex]:	0x05

Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the OCU
	PinAction	OCU_SET_LOW, OCU_SET_HIGH, OCU_TOGGLE, OCU_DISABLE
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to indicate the driver what shall be done automatically by hardware (if supported) upon compare match.	

⌋(SRS_Ocu_00012)

[SWS_Ocu_00077] ⌈The function Ocu_SetPinAction shall set the action to be performed by hardware automatically, at the next compare match in the corresponding OCU channel. ⌋(SRS_Ocu_00012)

[SWS_Ocu_00078] ⌈The fuction OCU Ocu_SetPinAction shall be reentrant if it is called for different channels.⌋()

[SWS_Ocu_00079] ⌈The function Ocu_SetPinAction shall be pre compile time configurable by the configuration parameter: OcuSetPinActionApi {OCU_SET_PIN_ACTION_API}.⌋(SRS_BSW_00171)

[SWS_Ocu_00080] ⌈If default error detection is enabled for the OCU driver: If the parameter `ChannelNumber` is invalid (not within the range specified by the configuration), the function Ocu_SetPinAction shall raise the error `OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋(SRS_BSW_00323, SRS_BSW_000386, SRS_SPAL_12448)

[SWS_Ocu_00081] ⌈If default error detection is enabled for the OCU driver: If a pin is not associated with the channel (not defined in the configuration of the channel), the function Ocu_SetPinAction shall raise the error `OCU_E_PARAM_NO_PIN` and return without any action.⌋(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)

[SWS_Ocu_00082] ⌈If default error detection is enabled for the OCU driver: If the parameter PinAction is invalid (not within the range specified by the type), the function Ocu_SetPinAction shall raise the error `OCU_E_PARAM_INVALID_ACTION` and return without any action.⌋(SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)

[SWS_Ocu_00083] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function Ocu_SetPinAction shall raise the error

OCU_E_UNINIT and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448)

[SWS_Ocu_00084] ⌈If a pin is associated with a channel; the relevant action with this pin shall be performed upon compare match.⌋ ()

8.3.7 Ocu_GetCounter

[SWS_Ocu_00085] ⌈

Service name:	Ocu_GetCounter	
Syntax:	Ocu_ValueType Ocu_GetCounter(Ocu_ChannelType ChannelNumber)	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ChannelNumber	Numeric identifier of the OCU channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Ocu_ValueType	Content of the counter in ticks
Description:	Service to read the current value of the counter.	

⌋ (SRS_Ocu_00009)

[SWS_Ocu_00086] ⌈The function Ocu_GetCounter shall read and return the value of the counter of the channel indicated by ChannelNumber.⌋ (SRS_Ocu_00009)

[SWS_Ocu_00087] ⌈The function Ocu_GetCounter shall be re-entrant.⌋ ()

[SWS_Ocu_00088] ⌈The function Ocu_GetCounter shall be pre compile time configurable by the configuration parameter: OcuGetCounterApi {OCU_GET_COUNTER_API}.⌋ (SRS_BSW_00171)

[SWS_Ocu_00089] ⌈If default error detection is enabled for the OCU driver: If the parameter ChannelNumber is invalid (not within the range specified by the configuration), the function Ocu_GetCounter shall raise the error OCU_E_PARAM_INVALID_CHANNEL and shall return the value **“0”.**⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448).

[SWS_Ocu_00090] ⌈If default error detection is enabled for the OCU driver: if the driver is not initialized, then the function Ocu_GetCounterValue shall raise the error OCU_E_UNINIT and shall return the value **“0”.**⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448).

8.3.8 Ocu_SetAbsoluteThreshold

[SWS_Ocu_00091] ⌈

Service name:	Ocu_SetAbsoluteThreshold	
Syntax:	`Ocu_ReturnType Ocu_SetAbsoluteThreshold( Ocu_ChannelType ChannelNumber, Ocu_ValueType ReferenceValue, Ocu_ValueType AbsoluteValue )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the OCU channel
	ReferenceValue	Value given by the upper layer and used as a base to determine whether to call the notification before the function exits or not.
	AbsoluteValue	Value to compare with the content of the counter. This value is in ticks.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Ocu_ReturnType	Tells the caller whether the compare match will occur (or has already occured) during the current Refence Interval, as a result of setting the new threshold value.
Description:	Service to set the value of the channel threshold using an absolute input data.	

⌋ (SRS_Ocu_00010)

[SWS_Ocu_00092] ⌈The function `Ocu_SetAbsoluteThreshold` shall set the channel threshold (the compare value) to the value given by `AbsoluteValue`.⌋ (SRS_Ocu_00010)

[SWS_Ocu_00093] ⌈The fuction `Ocu_SetAbsoluteThreshold` shall be reentrant if it is called for different channels.⌋ ()

[SWS_Ocu_00094] ⌈The function Ocu_SetAbsoluteThreshold shall be pre compile time configurable On/Off by the configuration parameter: OcuSetAbsoluteThresholdApi {OCU_SET_ABSOLUTE_THRESHOLD_API}.⌋ (SRS_BSW_00171)

[SWS_Ocu_00095] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function `Ocu_SetAbsoluteThreshold` shall raise the error `OCU_E_UNINIT` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448).

[SWS_Ocu_00096] ⌈If default error detection is enabled for the OCU driver: If the parameter `ChannelNumber` is invalid (not within the range specified by the configuration), the function Ocu_SetAbsoluteThreshold shall raise the error

`OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448).

Note: ReferenceValue is information from the upper layer. With the combination of the ReferenceValue and the AbsoluteValue an interval (defined as 'Reference Interval', green area in the pictures below) is provided to take into account the fact that the counter is running continuously and there might be a delay between the request from a caller to update the compare threshold and the actual modification of this threshold.

To simplify the description here, we postulate that due to internal MCU and peripheral timings the write action to a HW compare register is always done:
- before the actual compare is made, this might even be within the same clock cycle (**case1**).
- after the actual compare is made, this might even be within the same clock cycle. (**case2**).
As shown with the following example *`Ocu_SetAbsoluteThreshold(1,30,35)`*; in the pictures below.

Case 1: the threshold is actually written before the target compare match occurs.

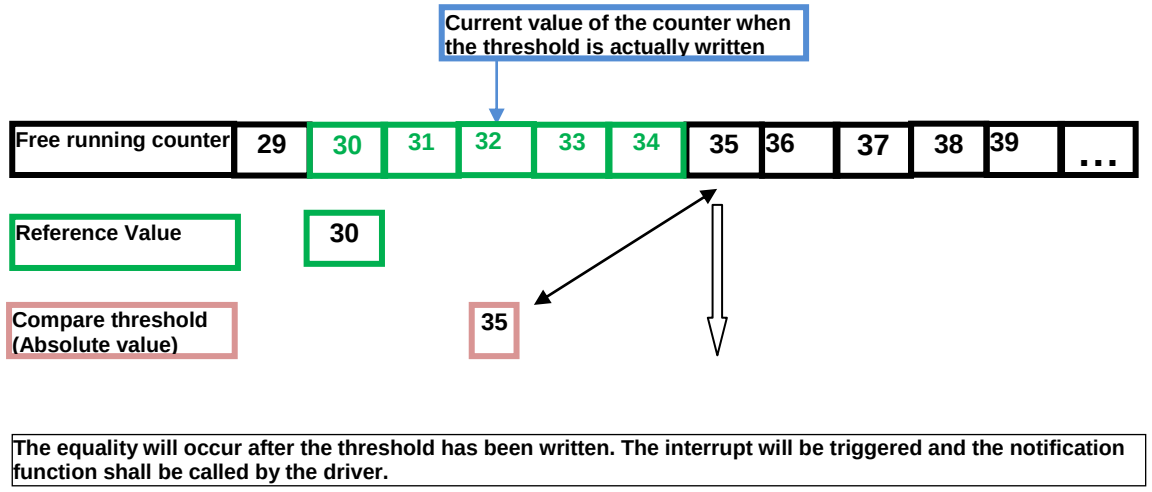


Figure 7: threshold actually written before the target compare match occurs

Case 2: the threshold is written after the targeted compare match has occurred

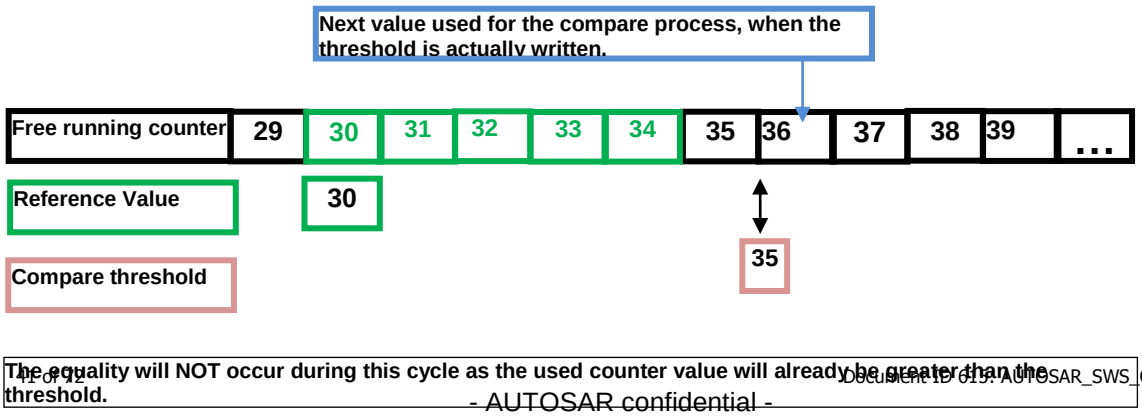


Figure 8: threshold actually written after the target compare match has occured

The Reference Interval takes into account the possible rollover of the counter as shown in the figure below.

Ocu_SetAbsoluteThreshold(1,70,20);
Example for a counter that runs from 0 to 255.

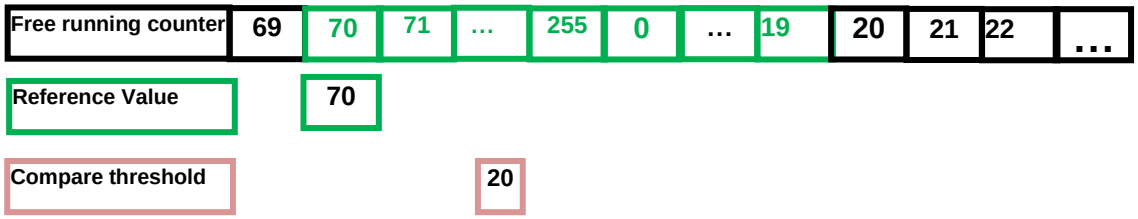


Figure 9: definition of a Reference Interval

As a result of the cases explained above, the expected behaviour of the driver is as follows.

- Notification to the upper layer is done only upon Compare Match (hardware): therefore there shall be a unique (at most, see further below about how to manage written threshold values) notification for each written value of the threshold during each Reference Interval.
- The API '`Ocu_SetAbsoluteThreshold`' shall return a status to inform the caller whether:
    - the writing was done inside the current Reference Interval (before actual compare match, it is even possible that the Compare Match might have already happened before the API returns)(**Case 1)**
    - or the writing was done outside the current Reference Interval. (**Case2)**

This status will help the caller (application) decide on how to proceed.

[SWS_Ocu_00098] ⌈After setting a new threshold value, the API `Ocu_SetAbsoluteThreshold` shall return a status to inform the caller whether the compare match will occur (or has already occured) during the current Reference Interval, as a result of setting the new threshold value. ⌋ ()

For the threshold value written during the previous call of the API `Ocu_SetAbsoluteThreshold`, the expected behaviour of the driver is as follows:
The previously written threshold value is erased by the current call.

Note: due to real time behaviour, the previously written threshold value might still produce a compare match; after the API has been called but the threshold value is not yet actually changed.

[SWS_Ocu_00097] ⌈Upon actual setting of a new threshold value, the previous threshold value (written during the last call of this API) shall no longer produce a compare match. ⌋ ()

8.3.9 Ocu_SetRelativeThreshold

[SWS_Ocu_00100] ⌈

Service name:	Ocu_SetRelativeThreshold	
Syntax:	`Ocu_ReturnType Ocu_SetRelativeThreshold( Ocu_ChannelType ChannelNumber, Ocu_ValueType RelativeValue )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different channel numbers	
Parameters (in):	ChannelNumber	Numeric identifier of the OCU channel
	RelativeValue	Value to use for computing the new threshold.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Ocu_ReturnType	Tells the caller whether the compare match will occur (or has already occured) during the current Refence Interval, as a result of setting the new threshold value.
Description:	Service to set the value of the channel threshold relative to the current value of the counter.	

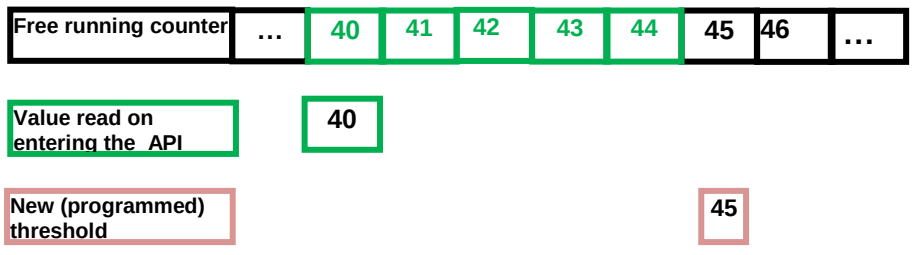
⌋ (SRS_Ocu_00010)

The behaviour of this API is as follows.

- On entry, the API reads the counter value (*ReadValue*). Then the new threshold value is computed and written according to the following formula: NewThresholdValue = *ReadValue* + RelativeValue.

The rest of the behaviour is then the same as for the API `Ocu_SetAbsoluteThreshold` where the reference value is now ReadValue, and the Reference Interval is between Readvalue and the new programmed threshold (NewThresholdValue) as shown in the picture below.

Exemple with Ocu_SetRelativeThreshold(1,5);



Note: As for the API Ocu_SetAbsoluteThreshold, the possible rollover of the counter is also included in the Reference Interval as shown in the figure below.
Exemple with Ocu_SetRelativeThreshold(1,20), with ReadValue equals to 253.

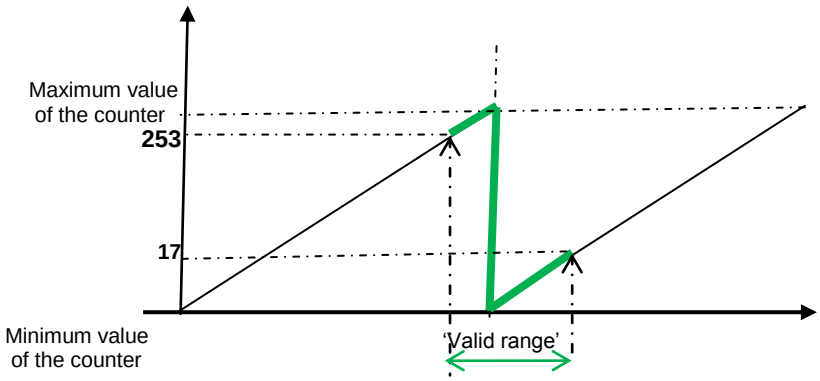


Figure 10: Taking into account the roll over of the counter

As a result, this API behaves like Ocu_SetAbsoluteThreshold , hence the requirements below.

[SWS_Ocu_00101] ⌈The function Ocu_SetRelativeThreshold shall add RelativeValue to the value of the counter on entering the function to compute the new threshold relative to the counter.⌋ ()

[SWS_Ocu_00106] ⌈After setting a new threshold value, the API `Ocu_SetRelativeThreshold` shall return a status to inform the caller whether the compare match will occur (or has already occured) during the current Reference Interval, as a result of setting the new threshold value.⌋ ()

[SWS_Ocu_00107] ⌈Upon actual setting of a new threshold value (absolute or relative), the previous threshold value shall no longer produce a compare match.⌋ ()

[SWS_Ocu_00102] ⌈The fuction OCU Ocu_SetAbsoluteThreshold shall be reentrant if it is called for different channels.⌋ ()

[SWS_Ocu_00103] ⌈The function Ocu_SetRelativeThreshold shall be pre compile time configurable On/Off by the configuration parameter: OcuSetRelativeThresholdApi {**OCU_SET_RELATIVE_THRESHOLD_API**}.⌋ (SRS_BSW_00171)

[SWS_Ocu_00104] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function Ocu_SetRelativeThreshold shall raise the error `OCU_E_UNINIT` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448).

[SWS_Ocu_00105] ⌈If default error detection is enabled for the OCU driver: if the parameter `ChannelNumber` is invalid (not within the range specified by the configuration), the function Ocu_SetRelativeThreshold shall raise the error `OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448).

8.3.10 Ocu_DisableNotification

[SWS_Ocu_00108] ⌈

Service name:	Ocu_DisableNotification
Syntax:	`void Ocu_DisableNotification( Ocu_ChannelType ChannelNumber )`
Service ID[hex]:	0x0a
Sync/Async:	Synchronous
Reentrancy:	Reentrant for different channel numbers
Parameters (in):	ChannelNumber — Numeric identifier of the OCU channel
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service is used to disable notifications from an OCU channel.

⌋ (SRS_Ocu_00007)

[SWS_Ocu_00109] ⌈The function Ocu_DisableNotification shall disable the OCU compare match notification.⌋ (SRS_Ocu_00007)

[SWS_Ocu_00110] ⌈The fuction OCU Ocu_DisableNotification shall be reentrant if it is called for different channels.⌋ ()

[SWS_Ocu_00111] ⌈The function Ocu_DisableNotification shall be pre compile time configurable On/Off by the configuration parameter: OcuNotificationSupported {OCU_NOTIFICATION_SUPPORTED}.⌋ (SRS_BSW_00171)

[SWS_Ocu_00112] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function `Ocu_DisableNotification` shall raise the error `OCU_E_UNINIT` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448).

[SWS_Ocu_00113] ⌈If default error detection is enabled for the OCU driver: If the parameter `Channel` is invalid (not within the range specified by configuration), the function `Ocu_DisableNotification` shall raise the error `OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL12448)

[SWS_Ocu_00114] ⌈If default error detection is enabled for the OCU driver: If the notification function is the NULL pointer, the function `Ocu_DisableNotification` shall raise the error `OCU_E_NO_VALID_NOTIF` and return without any action.⌋ () (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448).

8.3.11 Ocu_EnableNotification

[SWS_Ocu_00115] ⌈

Service name:	Ocu_EnableNotification
Syntax:	`void Ocu_EnableNotification( Ocu_ChannelType ChannelNumber )`
Service ID[hex]:	0x0b
Sync/Async:	Synchronous
Reentrancy:	Reentrant for different channel numbers
Parameters (in):	ChannelNumber — Numeric identifier of the OCU channel
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service is used to enable notifications from an OCU channel.

⌋ (SRS_Ocu_00007)

[SWS_Ocu_00116] ⌈The function Ocu_EnableNotification shall enable the OCU compare match notification of the indexed channel.⌋ (SRS_Ocu_00007)

[SWS_Ocu_00117] ⌈The function Ocu_EnableNotification shall be reentrant if it is called for different channels.⌋ ()

[SWS_Ocu_00118] ⌈The function Ocu_EnableNotification shall be pre compile time configurable On/Off by the configuration parameter: OcuNotificationSupported {OCU_NOTIFICATION_SUPPORTED}.⌋ (SRS_BSW_00171)

[SWS_Ocu_00119] ⌈If default error detection is enabled for the OCU driver: If the driver is not initialized, the function `Ocu_EnableNotification` shall raise the error `OCU_E_UNINIT` and return without any action.⌋ (SRS_BSW_00406, SRS_BSW_00386, SRS_SPAL_12448).

[SWS_Ocu_00120] ⌈If default error detection is enabled for the OCU driver: If the parameter `Channel` is invalid (not within the range specified by configuration), then the function `Ocu_EnableNotification` shall raise the error `OCU_E_PARAM_INVALID_CHANNEL` and return without any action.⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448)

[SWS_Ocu_00121] ⌈If default error detection is enabled for the OCU driver: If the notification function is the NULL pointer, the function `Ocu_EnableNotification` shall raise the error `OCU_E_NO_VALID_NOTIF` and return without any action.⌋ () (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448).

8.3.12 Ocu_GetVersionInfo

[SWS_Ocu_00122] ⌈

Service name:	Ocu_GetVersionInfo	
Syntax:	`void Ocu_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x09	
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:	This service returns the version information of this module.	

⌋ ()

[SWS_Ocu_00123] ⌈The function Ocu_GetVersionInfo shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers.⌋ (SRS_BSW_00407)

[SWS_Ocu_00124] ⌈The function Ocu_GetVersionInfo shall be pre compile time configurable On/Off by the configuration parameter: OcuVersionInfoApi {OCU_VERSION_INFO_API}.⌋ (SRS_BSW_00407, SRS_BSW_00411)

[SWS_Ocu_00125] ⌈If source code for caller and callee of Ocu_GetVersionInfo is available; the OCU driver should realize Ocu_GetVersionInfo as a macro, defined in the module's header file.⌋ ()

[SWS_Ocu_00126] ⌈If default error detection is enabled for the OCU driver, the function `Ocu_GetVersionInfo` shall raise development error `OCU_E_PARAM_POINTER` if parameter versioninfo is a null pointer, and return without any action⌋ (SRS_BSW_00323, SRS_BSW_00386, SRS_SPAL_12448).

8.4 Callback notifications

Since the OCU 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

The OCU driver offers only synchronous services and therefore doesn't need any scheduled functions.

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

This module does not require any mandatory interfaces.

8.6.2 Optional Interfaces

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

[SWS_Ocu_00127] ⌈

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_Ocu_00128] ⌈

Service name:	Ocu_Notification_<Channel>
Syntax:	`void Ocu_Notification_<Channel>( void )`
Sync/Async:	Synchronous
Reentrancy:	Reentrancy of this API call depends on the user code

Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This notification function is called when a compare match occurs on the associated channel.

⌋(SRS_BSW_00359, SRS_BSW_00360, SRS_SPAL_00157)

The notification prototype `Ocu_Notification_<channel#>` is for the notification callback function provided by the upper layer and shall be implemented by the user.

[SWS_Ocu_00129] ⌈The OCU driver shall call the function Ocu_Notification_<Channel#> according to the last call of Ocu_EnableNotification/Ocu_DisableNotification for channel <Channel#>, if there's a compare match on that channel.⌋(SRS_SPAL_00157)

[SWS_Ocu_00130] ⌈The OCU driver shall reset the interrupt flag (if needed by hardware) associated with the notification Ocu_Notification_<Channel#>⌋(SRS_SPAL_12129)

[SWS_Ocu_00131] ⌈The Ocu notification functions shall be configurable as function pointers within the initialization data structure (`Ocu_ConfigType`).⌋(SRS_SPAL_12056)

[SWS_Ocu_00132] ⌈⌈If the NULL pointer is configured for a notification call-back, then no call-back shall be executed.⌋(SRS_SPAL_12056)

[SWS_Ocu_00133] ⌈When the notification mechanism is disabled, the OCU driver shall send no notification.⌋(SRS_BSW_00157)

9 Sequence and Timing diagrams

9.1 Initialization

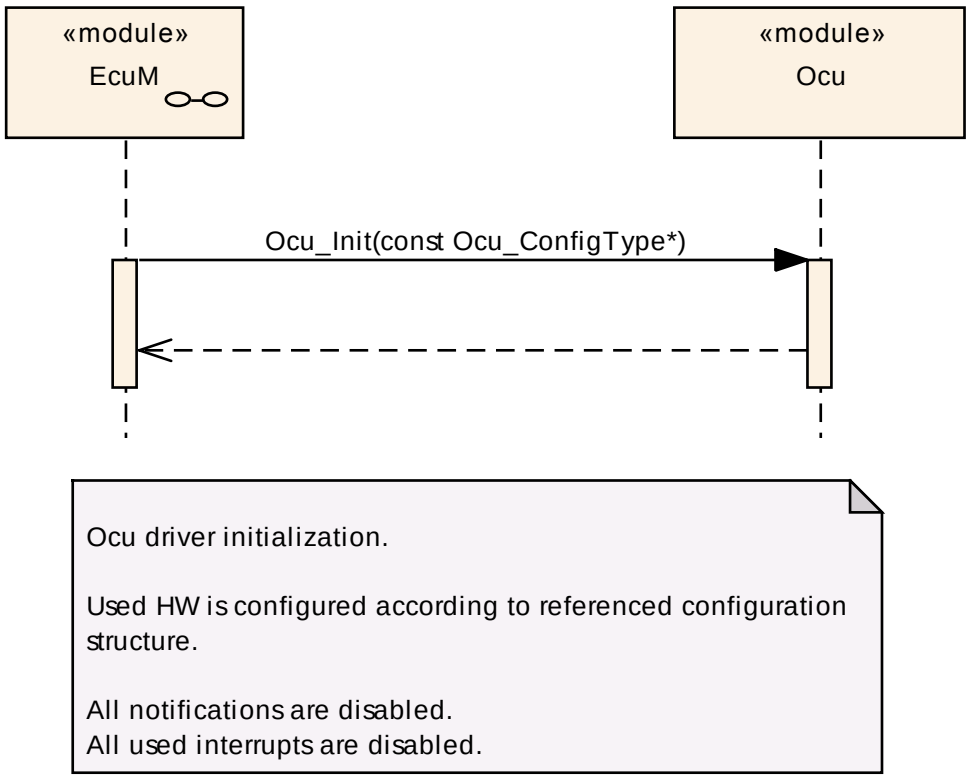


Figure 11: Ocu initialization

9.2 De-initialization

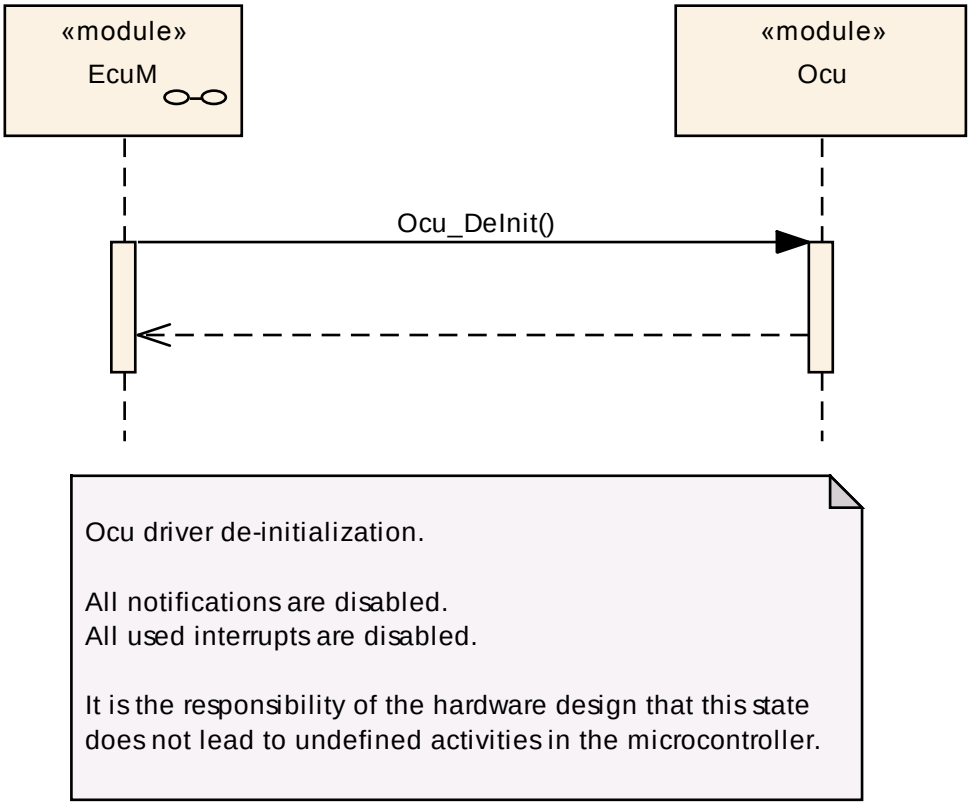


Figure 12: Ocu de-initialization

9.3 Using the Ocu Notifications

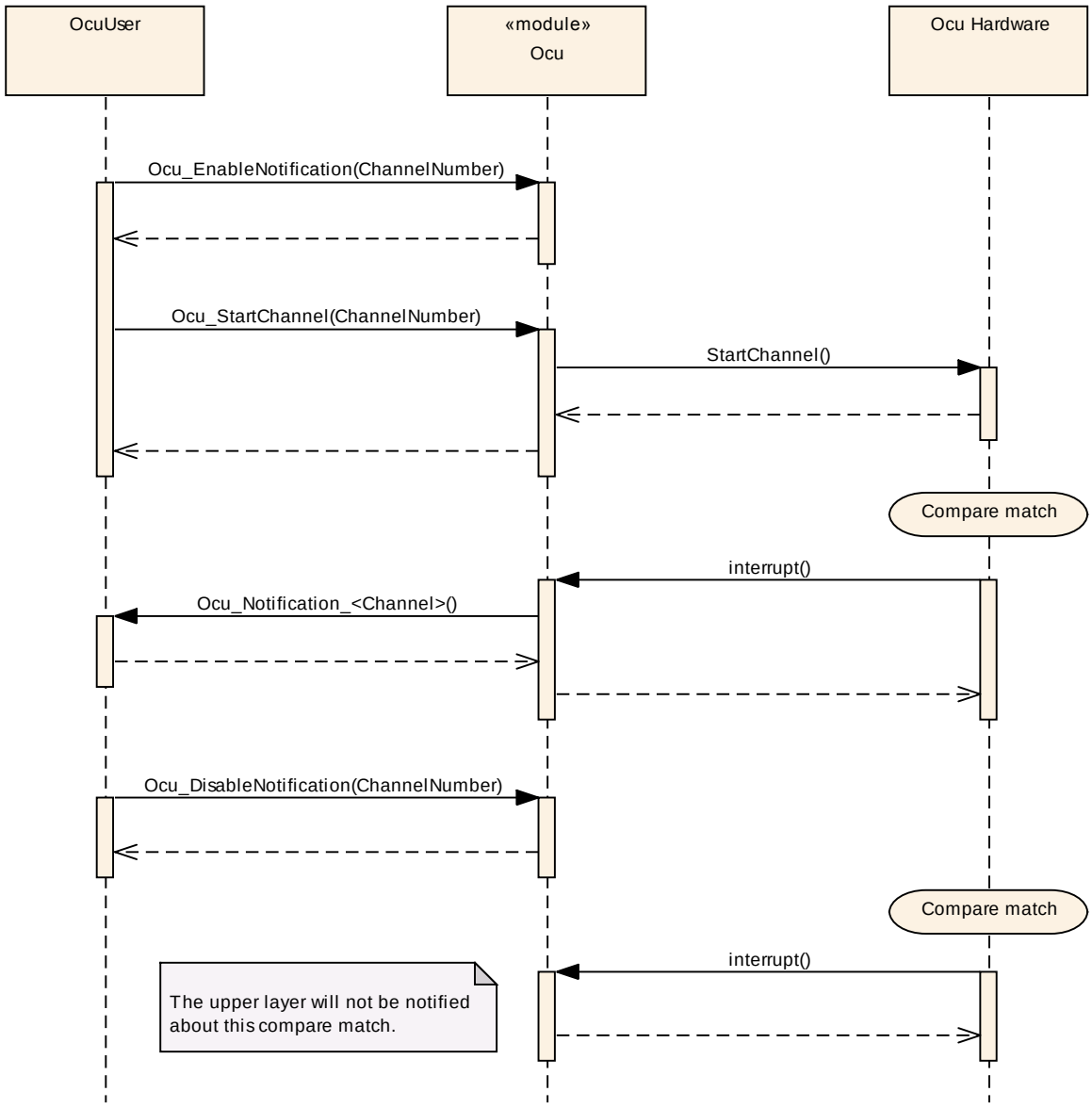


Figure 13: Enable/Disable Notifications

9.4 Ocu_SetPinState

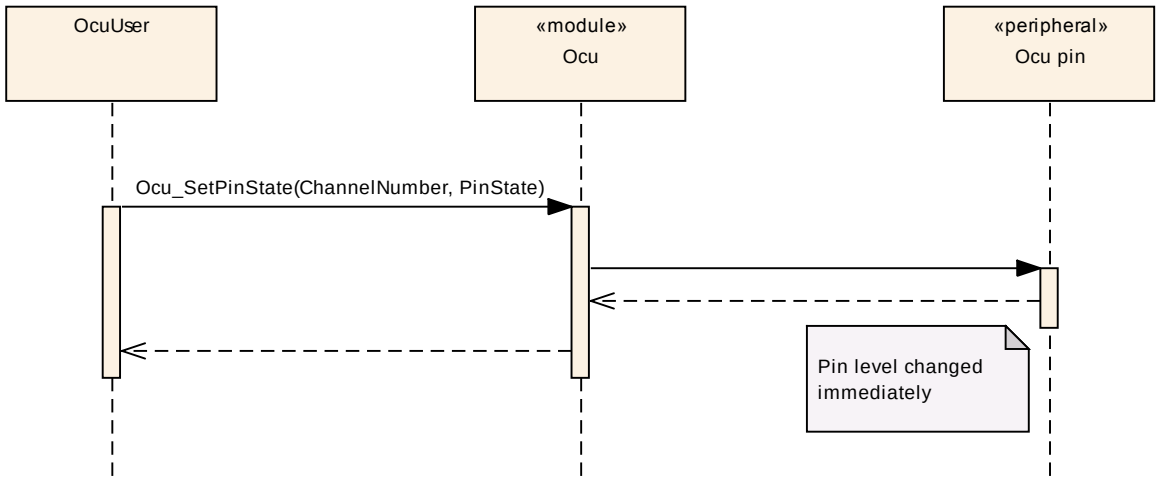


Figure 14: Ocu driver sets the pin state

9.5 Ocu_SetPinAction

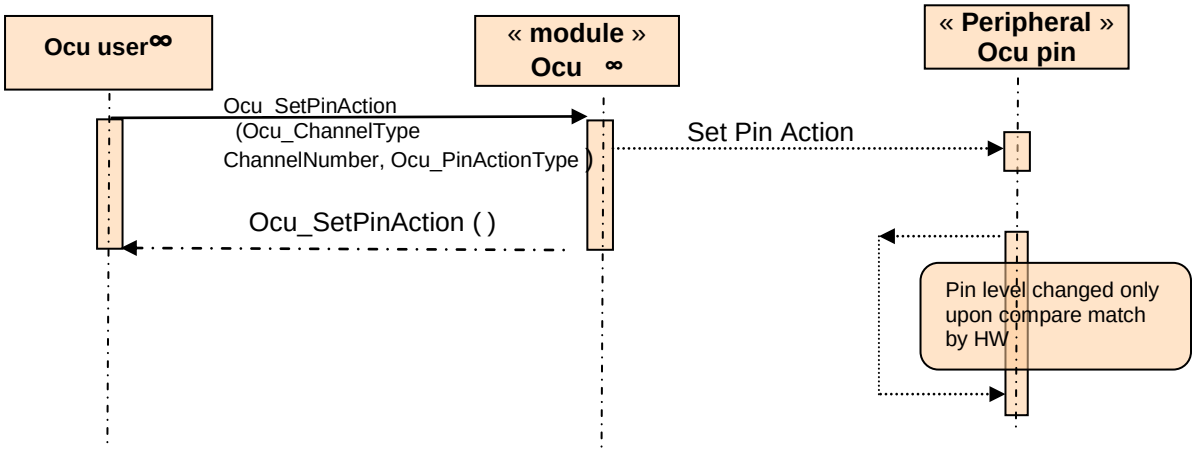


Figure 15: Change the pin state upon compare match

9.6 Setting a new compare threshold

Refer to the chapters Ocu_SetAbsoluteThreshold and Ocu_SetRelativeThreshold.

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 OCU Driver.

Chapter 10.3 specifies published information of the module OCU Driver.

10.1 How to read this chapter

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

- AUTOSAR Layered Software Architecture [1]
- AUTOSAR ECU Configuration Specification [6]
  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 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.

10.1.3 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 the chapters: Functional specification and API specification.

10.2.1 Variants

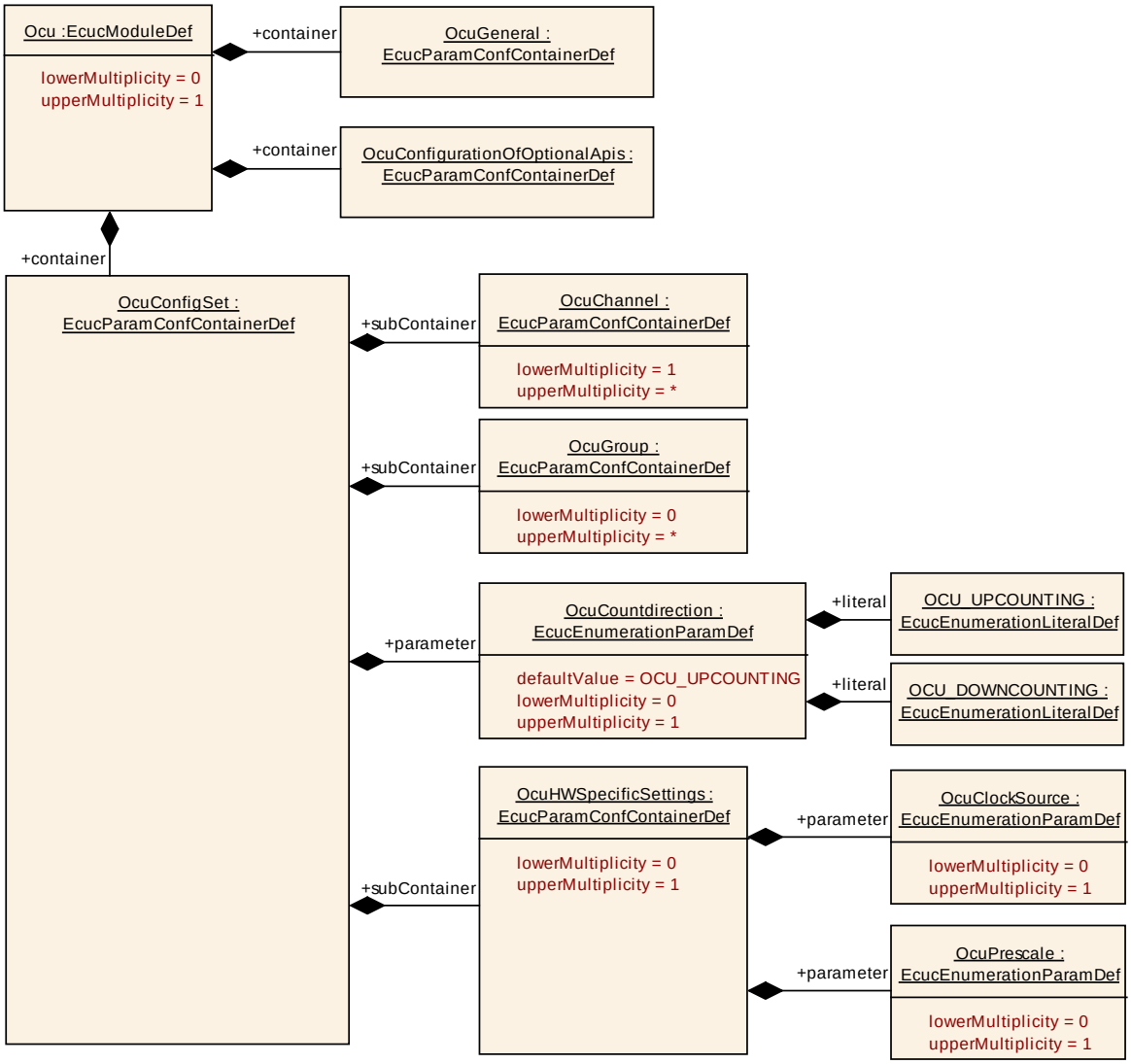
[SWS_Ocu_00134] ⌈VARIANT-PRE-COMPILE (Pre Compile) is limited to pre-compile configuration parameters only.⌋ ()

[SWS_Ocu_00135] ⌈VARIANT-POST-BUILD includes a mix of pre-compile, link time and post build configuration parameters.⌋ ()

10.2.2 Ocu

SWS Item	**ECUC_Ocu_00136 :**
Module Name	*Ocu*
Module Description	Configuration of Ocu (Output Compare Unit) module.
Post-Build Variant Support	true

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OcuConfigSet	1	This container is the base of a Configuration Set, which contains the configured OCU channels. This way, different configuration sets can be defined for post-build process.
OcuConfigurationOfOptionalApis	1	Configuration of optional APIs.
OcuGeneral	1	This container contains the module-wide configuration parameters of the OCU Driver.

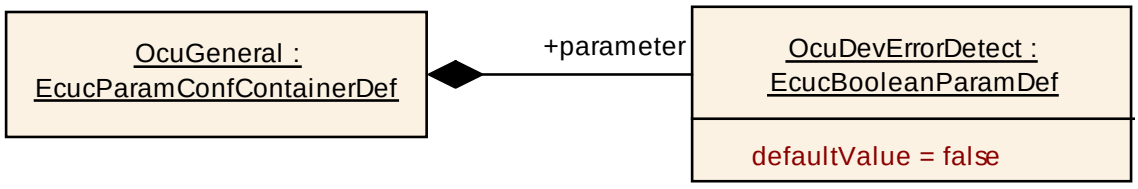


10.2.3 OcuGeneral

SWS Item	**ECUC_Ocu_00137 :**
Container Name	OcuGeneral
Description	This container contains the module-wide configuration parameters of the OCU Driver.
Configuration Parameters	

SWS Item	**ECUC_Ocu_00138 :**		
Name	OcuDevErrorDetect		
Description	Switches the Default Error Tracer (Det) detection and notification ON or OFF. • true: enabled (ON). • false: disabled (OFF).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.4 OcuConfigurationOfOptionalApis

SWS Item	**ECUC_Ocu_00139 :**		
Container Name	OcuConfigurationOfOptionalApis		
Description	Configuration of optional APIs.		
Configuration Parameters			

SWS Item	**ECUC_Ocu_00140 :**		
Name	OcuDeInitApi		
Description	Adds / removes the service Ocu_DeInit() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00141 :**		
Name	OcuGetCounterApi		
Description	Adds / removes the service Ocu_GetCounter() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00142 :**		
Name	OcuNotificationSupported		
Description	Adds / removes the services Ocu_EnableNotification() and Ocu_DisableNotification() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00143 :**		
Name	OcuSetAbsoluteThresholdApi		
Description	Adds / removes the service Ocu_SetAbsoluteThreshold() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

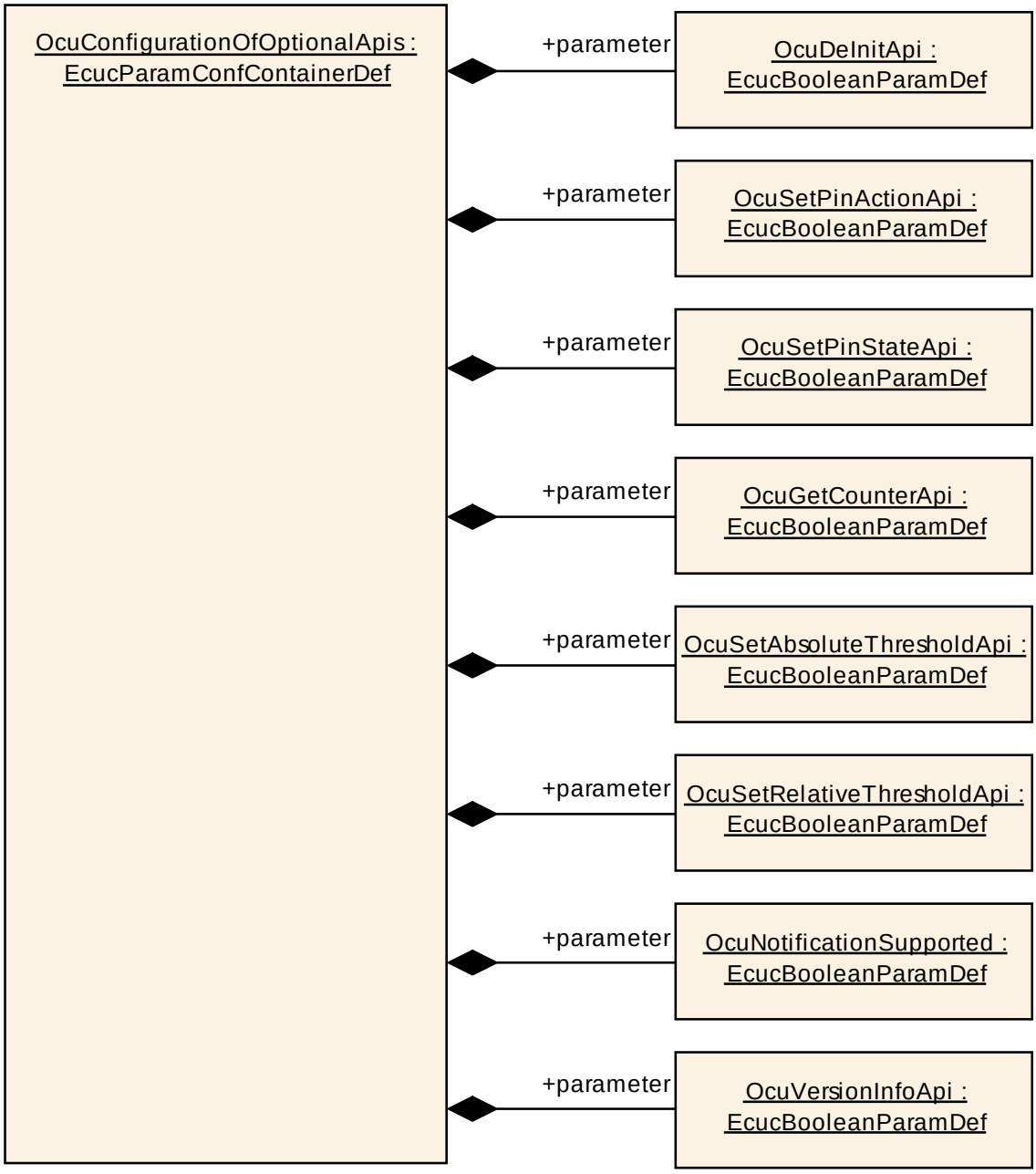
SWS Item	ECUC_Ocu_00144 :		
Name	OcuSetPinActionApi		
Description	Adds / removes the service Ocu_SetPinAction() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Ocu_00145 :		
Name	OcuSetPinStateApi		
Description	Adds / removes the service Ocu_SetPinState() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Ocu_00146 :		
Name	OcuSetRelativeThresholdApi		
Description	Adds / removes the service Ocu_SetRelativeThreshold() from the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Ocu_00147 :		
Name	OcuVersionInfoApi		
Description	Switch to indicate that the Ocu_GetVersionInfo() is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

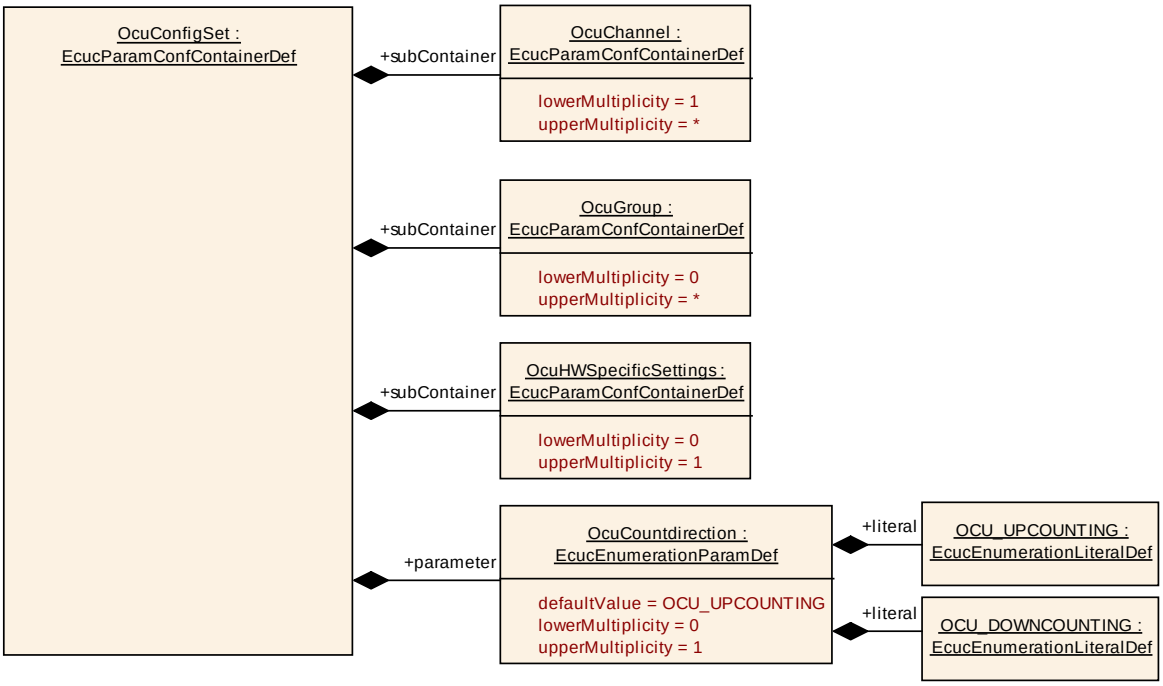

OcuConfigurationOfOptionalApis : EcucParamConfContainerDef
+parameter
OcuDeInitApi : EcucBooleanParamDef
+parameter
OcuSetPinActionApi : EcucBooleanParamDef
+parameter
OcuSetPinStateApi : EcucBooleanParamDef
+parameter
OcuGetCounterApi : EcucBooleanParamDef
+parameter
OcuSetAbsoluteThresholdApi : EcucBooleanParamDef
+parameter
OcuSetRelativeThresholdApi : EcucBooleanParamDef
+parameter
OcuNotificationSupported : EcucBooleanParamDef
+parameter
OcuVersionInfoApi : EcucBooleanParamDef

10.2.5 OcuConfigSet

SWS Item	**ECUC_Ocu_00148 :**
Container Name	OcuConfigSet
Description	This container is the base of a Configuration Set, which contains the configured OCU channels. This way, different configuration sets can be defined for post-build process.
Configuration Parameters	

SWS Item	**ECUC_Ocu_00149 :**		
Name	OcuCountdirection		
Description	This parameter indicates the count direction for the whole OCU driver.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	OCU_DOWNCOUNTING	The OCU counter will reckon from the maximum to the minimum value.	
	OCU_UPCOUNTING	The OCU counter will reckon from the minimum to the maximum value.	
Default value	OCU_UPCOUNTING		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OcuChannel	1..*	Configuration of an individual OCU channel.
OcuGroup	0..*	This container contains the parameters for configuring an OCU group.
OcuHWSpecificSettings	0..1	This container contains Ocu-specific parameters for selecting the clock source and setting optional prescalers if supported by hardware. Implementation is defined vendor specific.


OcuConfigSet :
EcucParamConfContainerDef
+subContainer
OcuChannel :
EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+subContainer
OcuGroup :
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+subContainer
OcuHWSpecificSettings :
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
OcuCountdirection :
EcucEnumerationParamDef
defaultValue = OCU_UPCOUNTING
lowerMultiplicity = 0
upperMultiplicity = 1
+literal
OCU_UPCOUNTING :
EcucEnumerationLiteralDef
+literal
OCU_DOWNCOUNTING :
EcucEnumerationLiteralDef

10.2.6 OcuChannel

SWS Item	**ECUC_Ocu_00150 :**		
Container Name	OcuChannel		
Description	Configuration of an individual OCU channel.		
Configuration Parameters			

SWS Item	**ECUC_Ocu_00151 :**		
Name	OcuAssignedHardwareChannel		
Description	The physical hardware channel that is assigned to this logical channel.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00152 :**		
Name	OcuChannelId		
Description	Channel Id of the OCU channel. This value will be assigned to the symbolic name derived from the OcuChannel container short name. It defines the assignment of the channel to the physical OCU hardware channel.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00153 :**		
Name	OcuChannelTickDuration		
Description	Specifies the tick duration of the counter of the channel. This parameter is the number of the input clock edges (rising edges or falling edges exclusively) counted each time to increase the counter by one unit. The value range depends from the used HW, not all allowed values may be relevant.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 32768		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00154 :**
Name	OcuDefaultThreshold
Description	Value of comparison threshold used for Initialization.

Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00155 :**		
Name	OcuHardwareTriggeredAdc		
Description	This parameter is used to allow the OCU channel to trigger an ADC channel upon compare match, if this is supported by hardware. The value of the parameter represents the ADC physical channel to trigger.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	0		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00156 :**		
Name	OcuHardwareTriggeredDMA		
Description	This parameter is used to allow the OCU channel to trigger a DMA channel upon compare match, if this is supported by hardware. The value of the parameter represents the DMA physical channel to trigger.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	0		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00157 :**		
Name	OcuMaxCounterValue		
Description	Maximum value in ticks, the counter of the OCU channel is able to count.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		

Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

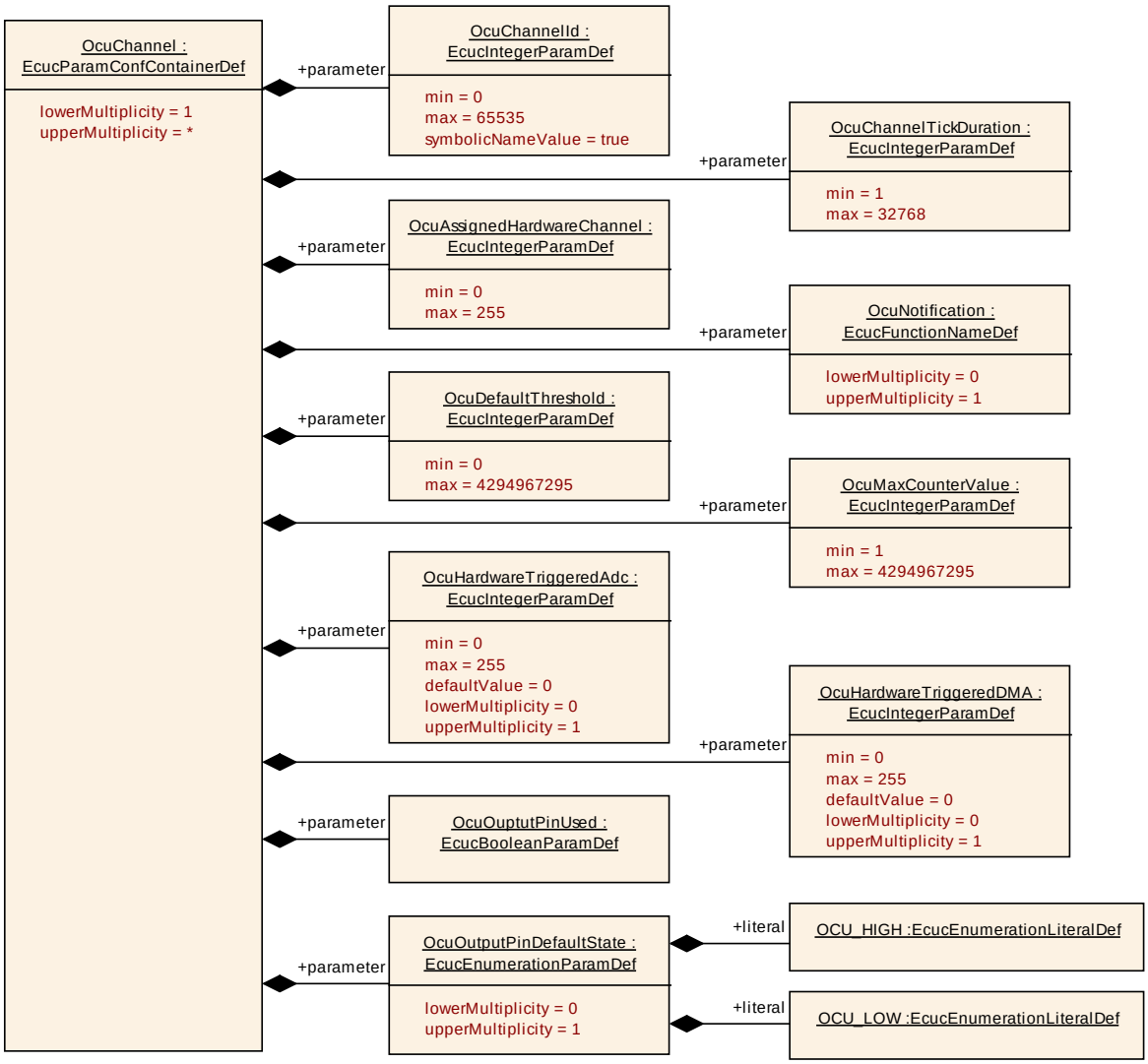
SWS Item	**ECUC_Ocu_00158 :**		
Name	OcuNotification		
Description	Definition of a function pointer to a Callback function.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00159 :**		
Name	OcuOuptutPinUsed		
Description	Information about the usage of an output pin on this channel. True: the channel uses an output pin. False: the channel does not use an output pin.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00160 :**		
Name	OcuOutputPinDefaultState		
Description	The parameter OcuOutputPinDefaultState represents the state that a pin associated with a channel shall be set to after initialisation.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	OCU_HIGH		The OCU channel output pin will be set to high (3 or 5 V) when requested.
	OCU_LOW		The OCU channel output pin will be set to low (0V) when requested.
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity	***Pre-compile time***	X	VARIANT-PRE-COMPILE

Configuration Class	***Link time***	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers



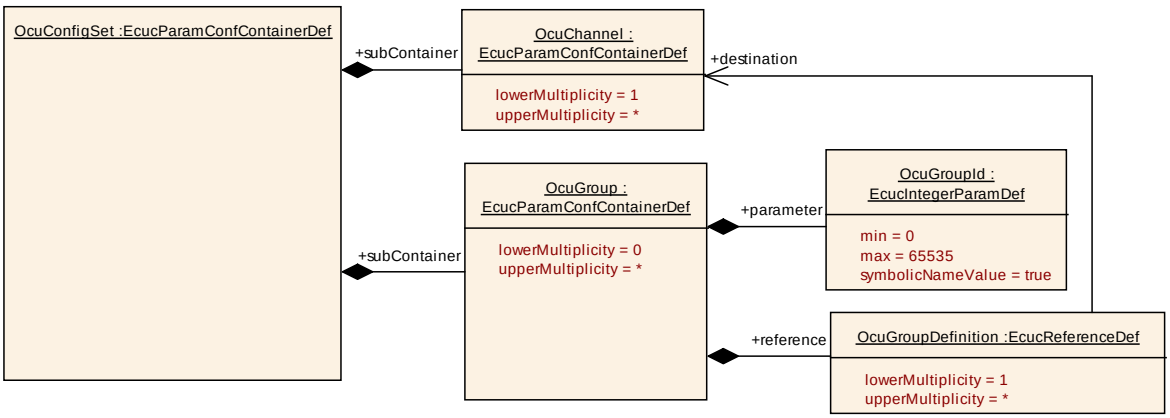
10.2.7 OcuGroup

SWS Item	**ECUC_Ocu_00161 :**		
Container Name	OcuGroup		
Description	This container contains the parameters for configuring an OCU group.		
Configuration Parameters			

SWS Item	**ECUC_Ocu_00162 :**		
Name	OcuGroupId		
Description	Numeric ID of the group. This parameter is the symbolic name of the group.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00163 :**		
Name	OcuGroupDefinition		
Description	Assignment of OcuChannels to an OcuGroup.		
Multiplicity	1..*		
Type	Reference to [OcuChannel]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers



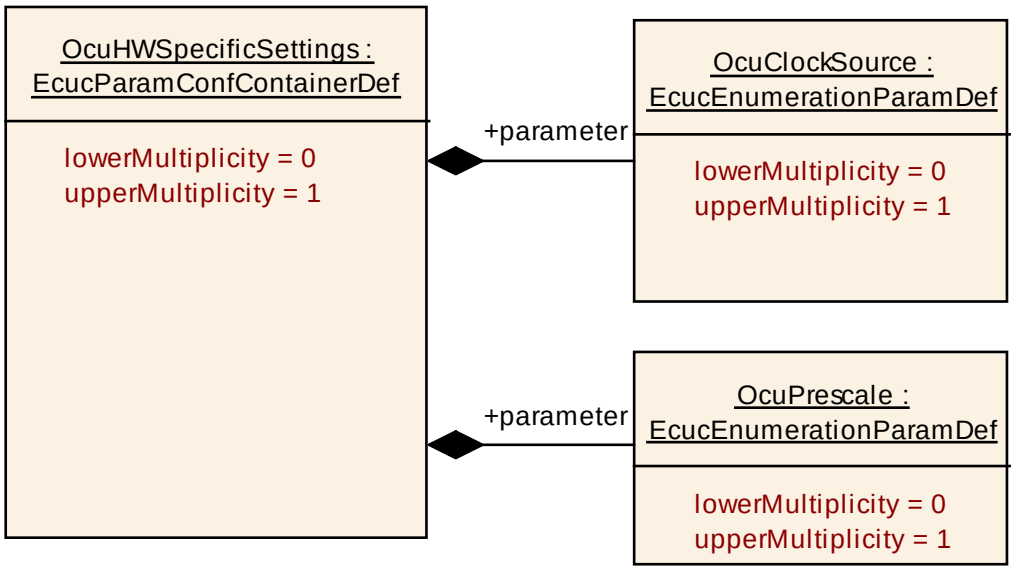
10.2.8 OcuHWSpecificSettings

SWS Item	**ECUC_Ocu_00164 :**
Container Name	OcuHWSpecificSettings
Description	This container contains Ocu-specific parameters for selecting the clock source and setting optional prescalers if supported by hardware. Implementation is defined vendor specific.
Configuration Parameters	

SWS Item	**ECUC_Ocu_00165 :**		
Name	OcuClockSource		
Description	The OCU driver specific clock input for the unit can statically be configured to select different clock sources if provided by hardware. Enumeration literals are defined vendor specific.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Ocu_00166 :**		
Name	OcuPrescale		
Description	Optional OCU driver specific clock prescale factor, if supported by hardware. Implementation is defined vendor specific.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers


OcuHWSpecificSettings :
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
OcuClockSource :
EcucEnumerationParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
OcuPrescale :
EcucEnumerationParamDef
lowerMultiplicity = 0
upperMultiplicity = 1

10.3 Published Information

[SWS_Ocu_00169] ⌈The standardized common published parameters as required by SRS_BSW_00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [8].⌋ ()

Additional module-specific published parameters are listed below if applicable.

11 Not applicable requirements

[SWS_Ocu_00156] ⌈These requirements are not applicable to this specification.⌋

(SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00170, 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, SRS_BSW_0431, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00434, 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)