Document Title	Requirements on Memory Hardware Abstraction Layer
Document Category	Requirement specification (RS)
Document UID	116
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History

Date	Release	Changed by	Description
2026-11-26	R26-11	AUTOSAR Release Management	• No content changes
2025-11-27	R25-11	AUTOSAR Release Management	• No content changes
2024-11-27	R24-11	AUTOSAR Release Management	• No content changes
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• Set items from draft to valid: SRS_MemHwAb_14034, SRS_MemHwAb_14035, SRS_MemHwAb_14036, SRS_MemHwAb_14037, SRS_MemHwAb_14038, SRS_MemHwAb_14039, SRS_MemHwAb_14040, SRS_MemHwAb_14041, SRS_MemHwAb_14042, SRS_MemHwAb_14043, SRS_MemHwAb_14044, SRS_MemHwAb_14045, SRS_MemHwAb_14046, SRS_MemHwAb_14047, SRS_MemHwAb_14048, SRS_MemHwAb_14049, SRS_MemHwAb_14050, SRS_MemHwAb_14051, SRS_MemHwAb_14052, SRS_MemHwAb_14053, SRS_MemHwAb_14054, SRS_MemHwAb_14055, SRS_MemHwAb_14056, SRS_MemHwAb_14057

△

2021-11-25	R21-11	AUTOSAR Release Management	• Added MemAcc and Mem related requirements (SRS_MemHwAb_14033 to SRS_MemHwAb_14056) due to Memory stack rework concept
2020-11-30	R20-11	AUTOSAR Release Management	• No content changes
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Added Requirements Tracing chapter
2015-07-31	4.2.2	AUTOSAR Release Management	• Requirements linked to BSW features
2014-10-31	4.2.1	AUTOSAR Release Management	• Requirements linked to BSW features
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• formal rework for requirements tracing • requirements reworked according to TPS_STDT_00078 • requirements linked to BSW and RTE features
2010-02-02	3.1.4	AUTOSAR Administration	• Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added

▽

△

2006-11-28	2.1.0	AUTOSAR Administration	• Legal disclaimer revised
2006-05-16	2.0.0	AUTOSAR Administration	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Scope of Document 7

2 How to read this document 8
2.1 Conventions used . . . 8
2.2 Requirements structure . . . 9

3 Acronyms and abbreviations 10

4 Functional Overview 11
4.1 Memory Access Module . . . 11
4.2 Memory Driver . . . 11
4.3 EEPROM Abstraction Layer . . . 11
4.4 Flash EEPROM Emulation . . . 11
4.5 Memory Abstraction Interface . . . 12

5 Requirements Specification 13
5.1 Functional Requirements . . . 13
5.1.1 Memory Abstraction Modules . . . 13
5.1.1.1 Configuration . . . 13
5.1.1.2 Initialization . . . 17
5.1.1.3 Normal Operation . . . 17
5.1.1.4 Shutdown Operation . . . 28
5.1.1.5 Fault Operation . . . 29
5.1.2 Memory Abstraction Interface . . . 30
5.1.2.1 General . . . 30
5.1.2.2 Configuration . . . 31
5.1.2.3 Normal Operation . . . 32
5.1.2.4 Fault Operation . . . 32
5.1.3 Onboard Device Abstraction . . . 32
5.2 Non-Functional Requirements (Qualities) . . . 33
5.2.1 Memory Abstraction Modules . . . 33
5.2.2 Memory Abstraction Interface . . . 33
5.2.2.1 Timing Requirements . . . 33
5.2.3 Onboard Device Abstraction . . . 34

6 References 35

A Change history of AUTOSAR traceable items 36
A.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 36
A.1.1 Added Requirements in R26-11 . . . 36
A.1.2 Changed Requirements in R26-11 . . . 36
A.1.3 Deleted Requirements in R26-11 . . . 36

A.2 Traceable item history of this document according to AUTOSAR Release R25-11 36
A.2.1 Added Requirements in R25-11 36
A.2.2 Changed Requirements in R25-11 36
A.2.3 Deleted Requirements in R25-11 36
A.3 Traceable item history of this document according to AUTOSAR Release R24-11 36
A.3.1 Added Requirements in R24-11 36
A.3.2 Changed Requirements in R24-11 37
A.3.3 Deleted Requirements in R24-11 37

1 Scope of Document

This document specifies requirements on the modules making up the Memory Hardware Abstraction Layer (MemHwA). The picture below shows the architecture and context of this Memory Hardware Abstraction Layer.

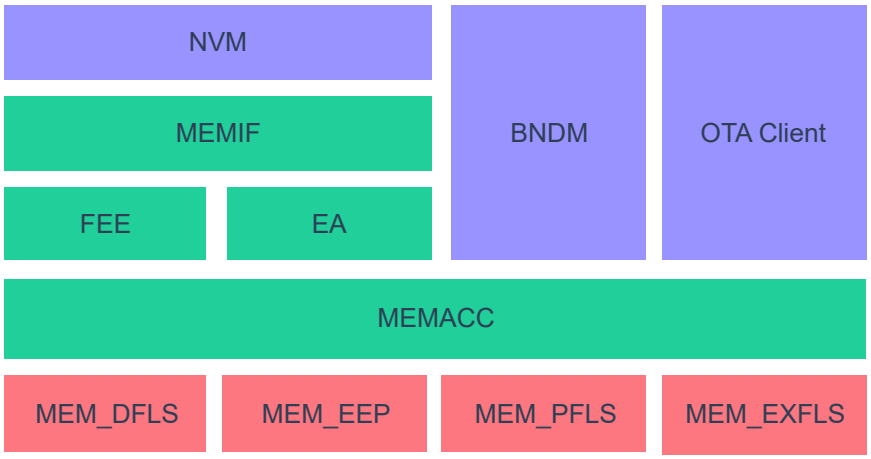


Figure 1.1: Components and Interfaces of the Memory Hardware Abstraction Layer

The Flash EEPROM Emulation (FEE) module and EEPROM Abstraction (EA) module shall provide a block based addressing scheme and a configurable, "virtually unlimited" number of erase-write-cycles. Thus, the upper layer module (the NVRAM manager) needs not be changed if the underlying memory driver and device is changed.

The Memory Access (MemAcc) module shall abstract from the addressing scheme of the underlying memory (Mem) drivers and provide an address based addressing scheme. Also, it provides a device-agnostic address-based memory interface to maintain the access coordination of different upper layer modules like NvM, BndM or OTA client compenent and the synchronization of the hardware access. Thus, the upper layer module (FEE, EA, BndM, etc.) needs not be changed if the underlying Mem drivers and devices are changed.

The Memory Abstraction Interface (MemIf) shall replace the driver interface layers (EEPROM and flash interface) and allow the NVRAM manager to access several memory abstraction modules (FEE and EA modules).

Instead of the combination of FEE / flash driver and / or EA / EEPROM driver, a vendor specific library might be used that provides the same functionality and API as those memory abstraction modules. The internals of such a library are of no concern as long as the functionality and API are supported. In case the vendor library replaces all needed FEE and EA modules, the Memory Abstraction Interface shall only be a bunch of macros.

2 How to read this document

Each requirement has its unique identifier starting with the prefix "BSW" (for "Basic Software"). For any review annotations, remarks or questions, please refer to this unique ID rather than chapter or page numbers!

2.1 Conventions used

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as follows.

Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the adjective “LEGALLY REQUIRED”, means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase, or the phrase “MUST NOT”, means that the definition is an absolute prohibition of the specification due to legal issues.
- SHALL: This phrase, or the adjective “REQUIRED”, means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective “RECOMMENDED”, means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase “NOT RECOMMENDED”, means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective “OPTIONAL”, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

An implementation, which does not include a particular option, SHALL be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, SHALL be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides).

2.2 Requirements structure

Each module specific chapter contains a short functional description of the Basic Software Module. Requirements of the same kind within each chapter are grouped under the following headlines (where applicable):

Functional Requirements:

- Configuration (which elements of the module need to be configurable)
- Initialization
- Normal Operation
- Shutdown Operation
- Fault Operation
- ...

Non-Functional Requirements:

- Timing Requirements
- Resource Usage
- Usability
- Output for other WPs (e.g. Description Templates, Tooling,...)
- ...

3 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to this document that are not included in the [1, AUTOSAR Glossary].

Abbreviation / Acronym:	Description:
(Logical) Block	Continuous area of memory that can be individually addressed by the module user (i.e. for read / write / erase / compare operations). The block size is statically configurable (pre-compile time).
Page	Smallest amount of memory that can be written in one pass.
Sector	Smallest amount of memory that can be erased in one pass.
FEE	Flash EEPROM Emulation
EA	EEPROM Abstraction Layer
MemIf	Memory Abstraction Interface
Mem	Memory Driver
MemAcc	Memory Access module
BndM	Bulk non-volatile data Manager
OTA client	Over The Air software update client
Sector Batch	Combination of multiple consecutive sectors of the same size
Sub Address Area	Combination of multiple non-contiguous sectors of the same size; used by Mem Acc
Address Area	Combination of multiple Sub Address Area
MCU	Microcontroller unit
MPU	Microprocessor unit
AP	AUTOSAR Adaptive Platform
CP	AUTOSAR Classic Platform

Table 3.1: Acronyms and abbreviations used in the scope of this Document

As this is a document from professionals for professionals, all other terms are expected to be known.

4 Functional Overview

4.1 Memory Access Module

The Memory Access (MemAcc) Module shall abstract any hardware dependency to the upper layer module which makes the memory access completely technology independent.

By abstracting the memory mapping in the Memory Access Module, the upper layer module doesn't need to know the physical segmentation of the underlying memory technology because the Memory Access module provides a contiguous logical memory area for the upper layer module.

The Memory Access Module shall handle all hardware independent functionality, such as iteration over multiple flash pages to program large memory areas. Apart from that it shall provide access coordination of different upper layer modules like NvM, BndM or OTA client compenent and the synchronization of the hardware access.

4.2 Memory Driver

The Memory Driver (Mem) shall provide a memory device agnostic interface to support all kinds of memory devices like flash, EEPROM, phase change memory (PCM), RAM, etc.

It supports basic services for reading, writing, and erasing of memory devices based on the physical segmentation.

In contrast to the FLS and EEP Driver, the Memory Driver works on physical addresses and supports data and code memory access.

4.3 EEPROM Abstraction Layer

The EEPROM Abstraction Layer (EA) shall extend the EEPROM driver such that it provides upper layer modules with a virtual segmentation on a linear address space and a "virtually limitless" number of erase / write cycles. Apart from that it shall provide the same functionality as an EEPROM driver.

4.4 Flash EEPROM Emulation

The Flash EEPROM Emulation (FEE) shall emulate the behavior of the EEPROM Abstraction Layer on flash memory technology. Thus it shall have the same functional scope and API as the EEPROM Abstraction Layer and allow for a similar configuration based on that of the underlying flash driver and flash device.

4.5 Memory Abstraction Interface

The Memory Abstraction Interface (MemIf) shall abstract from the number of underlying FEE or EA modules and provide upper layer modules with a virtual segmentation on a uniform linear address space.

5 Requirements Specification

5.1 Functional Requirements

5.1.1 Memory Abstraction Modules

5.1.1.1 Configuration

[SRS_MemHwAb_14057] MemAcc module shall allow the configuration of the non-contiguous physical memory areas of different memory devices to a logical address area ⌈

Description:	MemAcc module shall allow the configuration of non-contiguous physical memory areas of different memory devices to a logical address area. The configuration parameters shall be used by the configuration tool to generate the memory areas allocated to each upper layer module. The following constraints shall be considered: 1. An address area can only be assigned to one upper layer module 2. Address areas can span multiple memory devices 3. Start address and length of memory access requests need to be aligned to the according physical memory segmentation 4. Within a sub-address area, only one sector size is allowed 5. Only one job per address area is allowed
Rationale:	1. Encapsulate hardware dependencies from upper layer modules 2. Simplify the memory acess by providing a logical address space 3. Enable merging non-contiguous physical address areas to a contiguous logical memory area 4. Enable merging of memory areas from different memory devices
Use Case:	1. Combination of internal and external memory devices to one address area for the OTA software update use case. The combination of the different physical areas to one logical address area simplifies the OTA client implementation. 2. The OTA software update client use case may need one address area for active software and one address area for inactive software. 3. BndM with non-contiguous physical memory.
Dependencies:	–
Supporting Material:	–

⌋

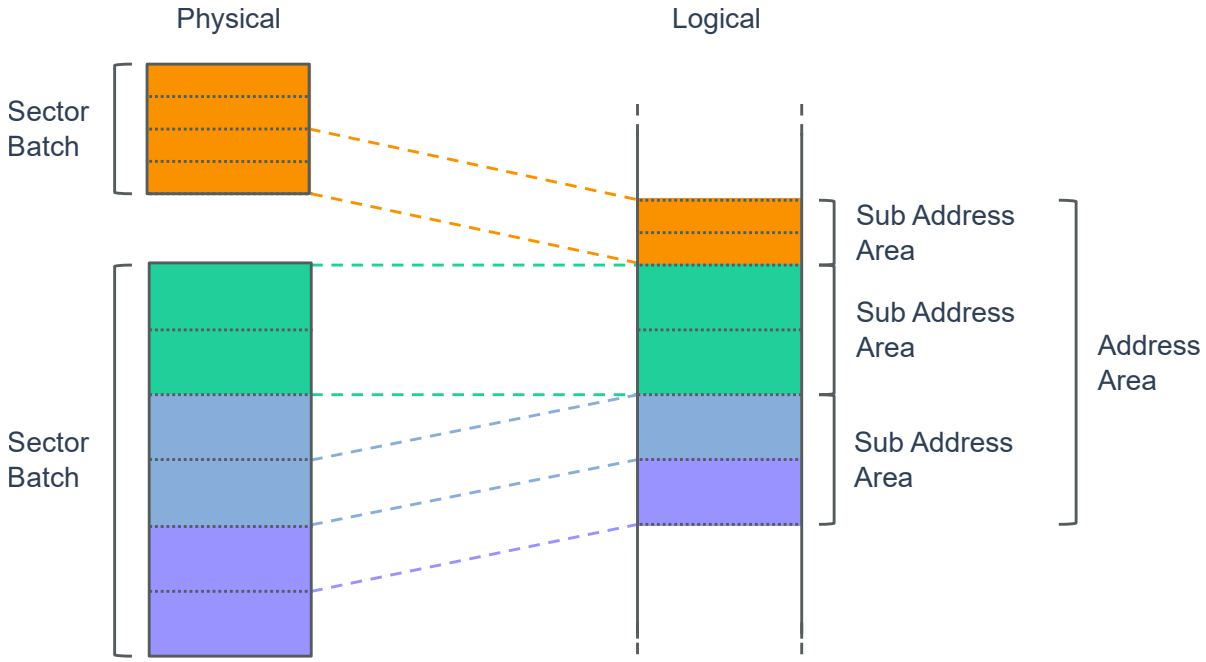


Figure 5.1: Overview of Address Translation/Mapping

[SRS_MemHwAb_14034] MemAcc module shall allow the configuration of the priority for different logical address areas ⌈

Description:	MemAcc module shall allow the configuration of the priority for different logical address areas. This configuration parameter shall be used by the configuration tool to generate the assigned priority for each upper layer module (address area).
Rationale:	1. Prioritization of writing crash data while OTA update running in the background. 2. Typically, code- and data flash share the same flash controller, therefore the write access of different upper layer modules or different CPUs needs to be prioritized/synchronized.
Use Case:	1. Shared data flash access of BNDM and FEE. 2. OTA software update in combination with FEE and shared hardware resources between code and data flash 3. FEE and HSM with shared data flash
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14035] MemAcc module shall support variant mapping ⌈

Description:	MemAcc module shall support variant mapping of two physical address areas to one virtual address area at initialization time.
Rationale:	For OTA software update use cases with an active and inactive memory area, the memory access from the OTA software update client shall always work with the same address area. Therefore, a variant mapping of two physical memory areas is necessary to one logical address area is needed. The variant selection shall be done at startup time.
Use Case:	OTA software update use case with active/inactive memory areas.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14036] Mem driver shall be statically configurable ⌈

Description:	The Mem driver shall allow the configuration of the physical attributes of a memory device like the memory segmentation or any memory device technology specific attributes.
Rationale:	Basic configuration
Use Case:	Physical segmentation needs to be considered by upper layer modules to align memory access requests.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14001] The FEE and EA modules shall allow the configuration of the alignment of the start and end addresses of logical blocks ⌈

Description:	The FEE and EA modules shall allow the configuration of the alignment of the start and end addresses of logical blocks. This configuration parameter shall be used by the configuration tool to generate the block numbers according to the block start addresses.
Rationale:	1) Ease handling of blocks inside the FEE and EA modules by aligning logical blocks to the underlying physical memory technology. 2) Allow for FEE and EA modules to calculate block start addresses instead of requiring a lookup table to map logical to physical addresses.

▽

△

Use Case:	1) The Freescale Star12 has an internal EEPROM with 4 byte sector and 2 byte page size. By aligning the block start and end addresses to 4 byte boundaries the handling of blocks can be simplified since read-modify-write behavior is no longer needed. 2) Example: The address alignment is set to 4 (bytes). The first logical block gets the block number 1, its start address is 0 (a device specific base address is added by the underlying memory driver). The block size is 22 bytes, so it takes up 6 4-byte "pages". The next logical block should then get not the number 2 but the number 7, thus allowing the memory abstraction module to deduce that its start address is 24 ((block number -1) * page size).
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14002] The FEE and EA modules shall allow the configuration of a required number of write cycles for each logical block ⌈

Description:	The FEE and EA modules shall allow the configuration of a required number of write cycles for each logical block.
Rationale:	Abstract from hardware properties of underlying physical devices.
Use Case:	An external flash device is specified for 10.000 erase cycles per erase unit. A logical block is configured that requires 50.000 erase cycles. The FEE has to make sure that this logical block can be written 50.000 times while at the same time no flash cell must be erased more than 10.000 times.
Dependencies:	[SRS_MemHwAb_14012] Spreading of write access
Supporting Material:	–

⌋

[SRS_MemHwAb_14026] The block numbers 0x0000 and 0xFFFF shall not be used ⌈

Description:	The block numbers 0x0000 and 0xFFFF shall not be used by the memory abstraction module / generated by the configuration tool.
Rationale:	These numbers can not be distinguished from the erased value of a flash or EEPROM device.
Use Case:	The implementation stores the block number in non-volatile memory e.g. to mark the start or end of a logical block. When these numbers would be used, that marker could not be found / distinguished from an empty EEPROM or flash memory.
Dependencies:	–
Supporting Material:	–

⌋

5.1.1.2 Initialization

[SRS_MemHwAb_14037] MemAcc module and Mem driver shall provide an interface for initialization ⌈

Description:	MemAcc module and Mem driver shall provide an interface for initialization of all states and all global variables of the module. Before initialization, MemAcc module and Mem driver are inactive.
Rationale:	Basic functionality
Use Case:	ECU initialization.
Dependencies:	–
Supporting Material:	–

⌋

5.1.1.3 Normal Operation

[SRS_MemHwAb_14038] MemAcc module and Mem driver shall provide asynchronous memory access functions ⌈

Description:	MemAcc module and Mem driver shall provide asynchronous functions for accessing memory devices.
Rationale:	Basic functionality
Use Case:	Memory access functions must be non-blocking since the upper layer modules expect an asynchronous interface.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14039] MemAcc module and Mem driver shall support optional services ⌈

Description:	MemAcc module and Mem driver shall provide measures to make Mem driver services optional and indicate to the upper layer module that a specific service is not available.
Rationale:	The erase service is not needed for all memory device technologies, e.g., phase change memory (PCM).
Use Case:	1. Memory device technologies which don't need an erase service 2. Read-only Mem drivers
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_MemHwAb_14040] MemAcc module and Mem driver shall provide a synchronous status function ⌈

Description:	MemAcc module and Mem driver shall provide a synchronous function which returns the job processing status.
Rationale:	Provide memory job processing status to the upper layer module.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14041] MemAcc module shall provide a job notification mechanism for the upper layer modules ⌈

Description:	MemAcc module and Mem driver shall provide a notification mechanism to notify the upper layer module about the completion of a memory job request.
Rationale:	Provide memory job processing status to the upper layer module.
Use Case:	Reduce runtime overhead for upper layer modules by providing a job nontification mechanism.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14042] MemAcc module shall support multiple Mem drivers for different types of memory ⌈

Description:	MemAcc module shall support multiple memory drivers for different types of memory (internal/external program flash, data flash, RAM, etc).
Rationale:	Different memory device technologies require different memory driver implementations
Use Case:	1. OTA software requires code memory access as well as data memory access with different memory drivers 2. Usage of internal and external memory for OTA software updates
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14043] Mem driver and shall support multiple instances of the same memory device ⌈

Description:	Mem driver shall support multiple instances of the same memory device.
Rationale:	Memory instance handling enables the usage of the same driver for multiple memory devices of the same type.
Use Case:	The OTA software update use case requires multiple memory devices of the same type to expand the memory resources.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14044] MemAcc module shall manage the memory job requests from different upper layer modules ⌈

Description:	MemAcc module shall manage the memory job requests from different upper layer modules. The MemAcc job management includes 1. Splitting of access request according to the physical memory segmentation 2. Processing of parallel job requests of distinct memory sub address areas from different upper layer modules 3. Synchronization of conflicting hardware access requests 4. Prioritization of conflicting memory job requests from different upper layer modules 5. Cancellation of job requests based on the physical memory segmentation, i.e. flash page/sector
Rationale:	The MemAcc job management reduces the impact on upper layer modules and simplifies the implementation of the Mem drivers.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14045] MemAcc module and Mem driver shall provide measures for dynamic driver activation ⌈

Description:	The Mem driver shall provide measures for dynamic driver activation.
Rationale:	For some safety use-cases, it is undesirable that the Mem driver is available in an executable form because the Mem driver might be accidently called and overwrites the applications memory. Therefore, the Mem driver needs to be dynamically downloaded to RAM or stored in encrypted form and just be decrypted in RAM as needed.

▽

△

Use Case:	Safety use cases to prevent accidental overwriting of memory areas.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14046] MemAcc module and Mem driver shall provide support for 64-Bit address range ⌈

Description:	MemAcc module and Mem driver shall provide the support for 64-Bit address range.
Rationale:	64-Bit address range is required to access more than 4GBytes memory.
Use Case:	Even though the typical CP ECUs don't need to address more than 4GBytes, for the OTA software update use case also CP MCUs need to be able to handle more than 4GBytes if the memory is shared with a POSIX/AP MPU.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14047] MemAcc module shall provide optional support for the initialization and main function triggering of memory drivers ⌈

Description:	MemAcc module shall provide optional support for the initialization and main function triggering of Mem drivers.
Rationale:	Since not all Mem drivers might be available all the time for some safety usecases, the Mem drivers cannot directly be initialized/triggered by ECUM/ SCHM.
Use Case:	For some safety use-cases, it is not desired that the Mem driver is available in an executable form as the memory driver might be accidently called and overwrites the applications memory. In this case, the Mem driver needs to be either downloaded dynamically to RAM or stored in encrypted form and just be decrypted in RAM as needed.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14048] Mem driver shall operate on physical segmentation/ physical addresses ⌈

Description:	The Mem driver shall only operate on the physical segmentation/physical addresses defined by the memory device technology i.e., pages and sectors for flash memory. Operations on larger areas than the physical segmentation shall be handled by MemAcc module.
Rationale:	Simplify Mem driver implementation.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14049] Mem driver shall use a standard binary format for dynamic driver activation ⌈

Description:	The Mem driver shall use a standard binary format for dynamic driver activation.
Rationale:	Since the MemAcc module shall not be hardware dependent, the Mem driver shall follow a standardized binary format so MemAcc can perform consistency checks for the activation of Mem drivers and provide a standardized method to call the Mem driver service functions.
Use Case:	Safety use cases to prevent accidental overwriting of memory areas.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14050] Mem driver shall handle only one job at one time ⌈

Description:	The Mem driver shall handle only one job (read, write or erase) at one time. Job requests during a running job shall be rejected.
Rationale:	Different operations like write and erase can't be handled at the same time and the results are dependent on the execution order.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14051] Mem driver shall not buffer data ⌈

Description:	The Mem driver shall not buffer data. The Mem driver services shall use the data buffers that are passed by the MemAcc module.
Rationale:	Avoid copy unnecessary copy operations.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14052] Mem driver multi-core type mapping ⌈

Description:	The Mem driver shall support multi-core type II requirements
Rationale:	To provide the most flexibility and to enable usage of hardware protection mechanisms (safety use cases), Mem driver shall support multi-core type II requirements.
Use Case:	Multi-core and safety use cases
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14053] Mem driver shall provide a function to a system ECC handle to propagate ECC errors ⌈

Description:	Mem driver shall provide a function to a system ECC handle to propagate non-correctable memory ECC errors to the Mem drive.
Rationale:	Dealing with ECC errors needs to be done on a system level as the error reaction needs to be handled on system level as well.
Use Case:	Typically, the Mem driver cannot detect an ECC error, thus cannot indicate an error to the upper layer module. Calling the Mem driver propagate ECC error API from a system ECC handler provides a way to propagate an ECC error using the normal fault handling mechanism to the Mem upper layer modules.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14054] MemAcc module shall provide a function to retrieve memory segmentation information ⌈

Description:	MemAcc module shall provide a function to retrieve memory segment information
Rationale:	Upper layer modules need to know segmentation of physical memory to align MemAcc access requests. No reference in the configuration required by upper layer modules.
Use Case:	OTA software update client with non-uniform sector layout
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14055] MemAcc module shall provide a lock function to enable/disable the direct memory access from application ⌈

Description:	MemAcc module shall provide lock function to enable/disable the direct memory access from application.
Rationale:	Lock functionality is required to avoid the parallel access of the same memory through MemAcc (i.e., from FEE, BNDM & OTA client etc.) and directly from application.
Use Case:	BNDM use case writes the memory through MemAcc and reads the data directly.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14056] MemAcc module and Mem driver shall provide a generic function to access the hardware specific functionalities ⌈

Description:	MemAcc module shall provide a generic function to access the hardware specific functionalities.
Rationale:	The generic function enables MemAcc to be hardware independent.
Use Case:	Hardware specific fault handling and additional hardware features not addressed by the standard MemAcc APIs.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14005] The FEE and EA modules shall provide upper layer modules with a virtual 32bit address space ⌈

Description:	The Flash EEPROM Emulation (FEE) and EEPROM Abstraction (EA) shall provide upper layer modules with a virtual 32bit address space. These 32 bit virtual (logical) addresses shall consist of a 16 bit logical block identifier and a 16 bit address offset within this logical block. Thus the memory abstraction layer shall support a (theoretical) number of 65534 logical (distinguishable) blocks per underlying physical device. Each block can have a (theoretical) size of 64 KBytes.
Rationale:	Abstract from hardware properties that would require changing the NVRAM manager if the underlying devices / drivers change.
Use Case:	1) Support systems with a high number of small blocks 2) Support systems with a few big blocks like e.g. MMI systems (fonts, speech) or navigation (maps, routes). 3) Allow NVRAM manager to encode block management information (e.g. block type) in the logical block identifier (by making it big enough)
Dependencies:	[SRS_MemHwAb_14026] Don't use certain block numbers
Supporting Material:	Figure 5.2: Virtual vs. physical address space

⌋

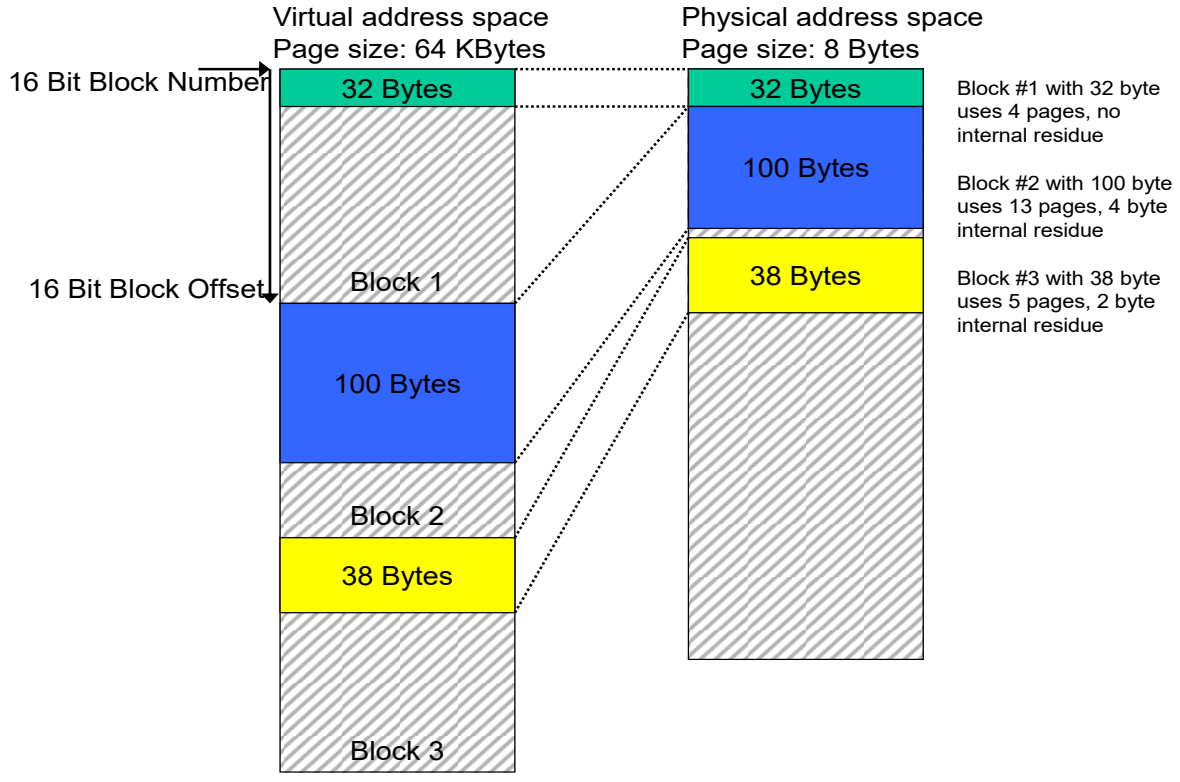


Figure 5.2: Virtual vs. physical address space

[SRS_MemHwAb_14006] The start address for a block erase or write operation shall always be aligned to the virtual 64K boundary ⌈

Description:	The start address for a block erase or write operation shall always be aligned to the virtual 64K boundary. In other words: The offset shall be ignored for block erase / write requests, every block erase / write request starts at address offset zero.
Rationale:	Allow optimized erase / write operations in underlying emulation modules and drivers if virtual 64K boundaries are mapped to physical sector / page boundaries.
Use Case:	Optimization of FEE and EA, simplify configuration and implementation.
Dependencies:	–
Supporting Material:	Just to make this clear: you can not erase or write only parts of the configured block, it's either all or nothing.

⌋

[SRS_MemHwAb_14007] The start address and length for reading a block shall not be limited to a certain alignment ⌈

Description:	The start address and length for reading a block shall not be limited to a certain alignment, i.e. it shall be possible to read one byte starting from any memory address.
Rationale:	Byte-wise reading of flash / EEPROM.
Use Case:	CRC calculation in the NVRAM manager.
Dependencies:	–
Supporting Material:	This allows reading a logical block in several passes, e.g. needed for CRC calculation. Note 1: If there are certain hardware properties that require an alignment of the read address, e.g. only 32bit aligned read possible, this shall be handled by the underlying driver. Note 2: This requirement shall allow the NVRAM manager to do a byte-wise read access on a logical block, it does not require the NVRAM manager to do so.

⌋

[SRS_MemHwAb_14009] The FEE and EA modules shall provide a conversion between the logical linear addresses and the physical memory addresses ⌈

Description:	The FEE and EA modules shall provide an unambiguous conversion between the logical linear addresses and the addresses used to access the underlying flash memory or EEPROM.
Rationale:	The physical device and the start address of a logical block shall be derived from the logical block identifier.
Use Case:	Transparent mapping of logical blocks to several physical non-volatile memory devices.

▽

△

Dependencies:	–
Supporting Material:	The memory addresses obtained by that conversion are address offsets to a device specific base address as described in the flash and EEPROM driver specifications.

⌋

[SRS_MemHwAb_14010] The FEE and EA modules shall provide a write service that operates only on complete configured logical blocks ⌈

Description:	The FEE and EA modules shall provide a write service that operates only on complete configured logical blocks.
Rationale:	Decouple the upper layer modules from driver internals.
Use Case:	The upper layer module shall only make one call to the Memory Abstraction Interface to write a logical block to non-volatile memory. If there are several passes needed to write all of the addressed memory area, this shall be handled internally in the FEE or EA modules or the underlying device drivers.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14029] The FEE and EA modules shall provide a read service that allows reading all or part of a logical block ⌈

Description:	The FEE and EA modules shall provide a read service that allows reading all or part of a logical block.
Rationale:	Allow for reading of NV memory.
Use Case:	Read functionality of the NVRAM manager.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14031] The FEE and EA modules shall provide a service that allows canceling an ongoing asynchronous operation ⌈

Description:	The FEE and EA modules shall provide a service that allows canceling an ongoing asynchronous operation like e.g. a read, write, erase or compare operation.
Rationale:	Needed for writing "immediate" data.
Use Case:	Immediate data (crash data) has to be written, while a read operation is currently in process.
Dependencies:	[SRS_MemHwAb_14013] Writing of "immediate" data must not be delayed

▽

△

Supporting Material:	–

⌋

[SRS_MemHwAb_14028] The FEE and EA modules shall provide a service to invalidate a logical block ⌈

Description:	The FEE and EA modules shall provide a service to invalidate a logical block. This shall be done by setting the module internal block management data appropriately. Note: Erasing the contents of the physical memory is an implementation option but not required.
Rationale:	To enable a data block to be marked as invalid by the upper layer.
Use Case:	Allow an application to mark data as outdated or no longer valid when physically erasing the data is not possible or not desirable (e.g. on flash memory technology).
Dependencies:	–
Supporting Material:	–

⌋

[SRS_MemHwAb_14012] Spreading of write access ⌈

Description:	If the configured number of write cycles for a logical block exceeds the number provided by the underlying physical device, the FEE or EA module has to provide sufficient mechanisms to spread the write requests for that logical block over a bigger memory area.
Rationale:	Allow for "unlimited" number of write cycles while simultaneously preventing memory cells from being erased more often than specified by the hardware vendor.
Use Case:	An external flash device is specified for 10.000 erase cycles per erase unit. A logical block is configured that requires 50.000 write cycles. The FEE has to make sure that this logical block can be written 50.000 times while at the same time no flash cell must be erased more than 10.000 times.
Dependencies:	[SRS_MemHwAb_14002] Configuration of number of required write cycles
Supporting Material:	This requirement replaces [BSW032] Spreading of write access and [SRS_LIBS_08530] NVRAM block type - walking from MemSvc SRS.

⌋

[SRS_MemHwAb_14013] Writing of immediate data shall not be delayed by internal management operations nor by erasing the memory area to be written to ⌈

Description:	Writing of immediate data shall not be delayed by internal management operations nor by erasing the memory area to be written to. If internal management operations are under way when immediate data has to be written, they have to be interrupted until the data has been written to non-volatile memory. There has to be a pre-erased memory area for writing of immediate data available at all times.
Rationale:	Immediate data has to be written immediately (that's what the name implies) that is as fast as the underlying hardware allows.
Use Case:	The FEE is reorganizing the blocks currently stored in flash when crash data has to be written.
Dependencies:	If an ongoing hardware access, e.g. an erase operation, can not be aborted its runtime has to be taken into account as the maximum allowable delay for immediate write operations.
Supporting Material:	–

⌋

[SRS_MemHwAb_14032] The FEE and EA modules shall provide an erase service that operates only on complete logical blocks containing immediate data ⌈

Description:	The FEE and EA modules shall provide an erase service that operates only on complete logical blocks containing immediate data.
Rationale:	SRS_MemHwAb_14013 requires pre-erased memory, therefore this memory areas have to be somehow erasable.
Use Case:	–
Dependencies:	[SRS_MemHwAb_14013] Writing of "immediate" data must not be delayed
Supporting Material:	• This service should only be called by a special application like e.g. diagnostics. • A possible implementation would be to invalidate the block containing immediate data and subsequently force a re-organization of blocks. During this re-organization invalidated blocks shall not be copied to the new memory location, thus the memory area for the immediate data will be (left) erased.

⌋

5.1.1.4 Shutdown Operation

The modules of the Memory Abstraction Layer don't need any shutdown capabilities (also there are no shutdown capabilities in the flash or EEPROM driver).

5.1.1.5 Fault Operation

[SRS_MemHwAb_14014] The FEE and EA modules shall detect possible data inconsistencies due to aborted / interrupted write operations ⌈

Description:	The FEE and EA modules shall detect possible data inconsistencies due to aborted / interrupted write operations.
Rationale:	The "user" shall not work on inconsistent data therefore it has to be recognized.
Use Case:	1) A write operation is interrupted by a loss of power, after power-on-reset the possible inconsistency of data shall be detected upon the next read access to the affected memory area. 2) A write operation is cancelled by the upper layer. Upon next read access to the affected memory area the possible data inconsistency shall be detected.
Dependencies:	–
Supporting Material:	Depending on the implementation, the physical device and the point in the write operation at which the interrupt occurs the FEE or EA module might be able to determine that the operation has failed but not which was the block that should have been written.

⌋

[SRS_MemHwAb_14015] The FEE and EA modules shall report possible data inconsistencies ⌈

Description:	The FEE and EA modules shall report possible data inconsistencies due to aborted / interrupted write operations to the DEM exactly once. After that the inconsistent memory area has to be marked such that no further errors are reported for that block.
Rationale:	Avoid "endless loops" in error reporting on every block read operation.
Use Case:	A write operation is interrupted or cancelled, the inconsistency is detected and reported upon the next read access to the affected memory area.
Dependencies:	[SRS_MemHwAb_14014] Detection of data inconsistencies
Supporting Material:	Depending on the implementation and the point in the write operation at which the interrupt occurs the FEE or EA module might be able to determine that the operation has failed but not which was the block that should have been written. In this case a read operation on that block might return old (outdated) data to the caller if such data is available. If this is not desired from the application, the block has to be explicitly invalidated before it is overwritten.

⌋

[SRS_MemHwAb_14016] The FEE and EA modules shall not return inconsistent data to the caller ⌈

Description:	The FEE and EA modules shall not return inconsistent data to the caller.
Rationale:	The "user" shall not work on inconsistent data.
Use Case:	A write operation is interrupted or cancelled, the data of that block thus is inconsistent. This inconsistency is detected on the next read access to that block, the data shall then not be returned to the caller.
Dependencies:	[SRS_MemHwAb_14014] Detection of data inconsistencies
Supporting Material:	Depending on the implementation and the point in the write operation at which the interrupt occurs the FEE or EA module might be able to determine that the operation has failed but not which was the block that should have been written. In this case a read operation on that block might return old (outdated) data to the caller if such data is available. If this is not desired from the application, the block has to be explicitly invalidated before it is overwritten. Providing default data for an inconsistent block is the job of the NVRAM manager.

⌋

5.1.2 Memory Abstraction Interface

The following requirements have been taken over from the SPAL SRS on Memory Abstraction and have been adapted (in wording only) to the architectural concept shown in Figure 1.1.

5.1.2.1 General

[SRS_MemHwAb_14019] The Memory Abstraction Interface shall provide uniform access to the API services of the underlying memory abstraction modules ⌈

Description:	The Memory Abstraction Interface shall provide uniform access to those API services of the underlying memory abstraction modules that are required for usage within the NVRAM manager. Further comments: The initialization routines and the job processing functions are not mapped by the memory abstraction interface.
Rationale:	Allow usage of memory abstraction modules by one uniform interface.
Use Case:	Allow the upper layer module access to internal and external memory devices without any difference.
Dependencies:	–

▽

△

Supporting Material:	This requirement shall replace [BSW12172].

⌋

[SRS_MemHwAb_14020] The Memory Abstraction Interface shall allow the selection of an underlying memory abstraction module by using a device index ⌈

Description:	The Memory Abstraction Interface shall allow the selection of an underlying memory abstraction module (FEE or EA module) by using a device index.
Rationale:	Requirement of the NVRAM Manager
Use Case:	The NVRAM Manager uses a device index for selecting the appropriate memory abstraction module.
Dependencies:	–
Supporting Material:	SWS NVRAM Manager This requirement shall replace [BSW12173].

⌋

5.1.2.2 Configuration

[SRS_MemHwAb_14021] The Memory Abstraction Interface shall allow the pre-compile time configuration of the number of underlying memory abstraction modules ⌈

Description:	The Memory Abstraction Interface shall allow the pre-compile time configuration of the number of underlying memory abstraction modules.
Rationale:	Flexibility
Use Case:	One ECU only uses internal EEPROM (thus needing one EA module), another ECU uses both internal plus external EEPROM (thus needing two EA modules).
Dependencies:	–
Supporting Material:	WP Architecture This requirement shall replace [BSW12174].

⌋

5.1.2.3 Normal Operation

[SRS_MemHwAb_14022] The Memory Abstraction Interface shall preserve the functionality of the underlying memory abstraction module ⌈

Description:	The Memory Abstraction Interface shall preserve the functionality of the underlying memory abstraction module. It shall not provide additional functionality.
Rationale:	Simplicity, efficiency
Use Case:	The memory abstraction modules abstract from all hardware properties, the Memory Abstraction Interface does not need to add anything (it only is needed to access more than one memory abstraction module).
Dependencies:	–
Supporting Material:	This requirement shall replace [BSW12175].

⌋

5.1.2.4 Fault Operation

[SRS_MemHwAb_14023] The Memory Abstraction Interface shall only check those parameters that are used within the interface itself ⌈

Description:	The Memory Abstraction Interface shall only check those parameters that are used within the interface itself and that are not passed to the underlying memory abstraction modules.
Rationale:	Simplicity, efficiency: avoid double checking of parameters.
Use Case:	The device index may be checked (depending on the setting of the development error detection switch). The block address shall not be checked.
Dependencies:	–
Supporting Material:	This requirement shall replace [BSW12176].

⌋

5.1.3 Onboard Device Abstraction

For the Onboard Device Abstraction the same requirements like for the Memory Hardware Abstraction apply. One member of the Onboard Device Abstraction is the Watchdog Interface.

5.2 Non-Functional Requirements (Qualities)

5.2.1 Memory Abstraction Modules

[SRS_MemHwAb_14017] The EA module shall extend the functional scope of an EEPROM driver ⌈

Description:	The EEPROM Abstraction Layer (EA) shall extend the functional scope of an EEPROM driver. In addition to the properties of an EEPROM driver, the EA shall work on a virtual 32bit address space and it shall abstract completely from the limitation of erase / write cycles given by the underlying device.
Rationale:	Uniform handling of all EEPROM devices.
Use Case:	The NVRAM manager shall not need to be changed if the underlying EEPROM drivers and devices change.
Dependencies:	–
Supporting Material:	AUTOSAR SRS EEPROM driver

⌋

[SRS_MemHwAb_14018] The FEE module shall extend the functional scope of an internal flash driver ⌈

Description:	The Flash EEPROM Emulation (FEE) shall extend the functional scope of an internal flash driver. It shall have the same functional scope and API as an EA module.
Rationale:	Uniform handling of all flash devices.
Use Case:	The NVRAM manager shall not need to be changed if the underlying flash drivers and devices change.
Dependencies:	[SRS_MemHwAb_14017] Scope of EEPROM Abstraction Layer
Supporting Material:	AUTOSAR SRS EEPROM driver AUTOSAR SRS Flash driver

⌋

5.2.2 Memory Abstraction Interface

5.2.2.1 Timing Requirements

[SRS_MemHwAb_14024] The Memory Abstraction Interface shall preserve the timing behavior of the underlying memory abstraction modules and their APIs ⌈

Description:	The Memory Abstraction Interface shall preserve the timing behavior of the underlying memory abstraction modules and their APIs by 1:1 mapping of the Memory Abstraction Interface API to the memory abstraction modules' API

▽

△

Rationale:	Simplicity, efficiency
Use Case:	Example: The write service of the Memory Abstraction Interface is directly mapped to the write service of an underlying memory abstraction module (FEE or EA).
Dependencies:	–
Supporting Material:	WP Architecture This requirement shall replace [BSW12177].

⌋

5.2.3 Onboard Device Abstraction

For the Onboard Device Abstraction the same requirements like for the Memory Hardware Abstraction apply. One member of the Onboard Device Abstraction is the Watchdog Interface.

6 References

[1] Glossary
AUTOSAR_FO_TR_Glossary

A Change history of AUTOSAR traceable items

A.1 Traceable item history of this document according to AUTOSAR Release R26-11

A.1.1 Added Requirements in R26-11

none

A.1.2 Changed Requirements in R26-11

none

A.1.3 Deleted Requirements in R26-11

none

A.2 Traceable item history of this document according to AUTOSAR Release R25-11

A.2.1 Added Requirements in R25-11

none

A.2.2 Changed Requirements in R25-11

none

A.2.3 Deleted Requirements in R25-11

none

A.3 Traceable item history of this document according to AUTOSAR Release R24-11

A.3.1 Added Requirements in R24-11

none

A.3.2 Changed Requirements in R24-11

none

A.3.3 Deleted Requirements in R24-11

none