Document Title	Requirements on Memory Hardware Abstraction Layer
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	116
Document Classification	Auxiliary
Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.0

Document Change History

Date	Release	Changed by	Change Description
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 & 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

Document Change History

Date	Release	Changed by	Change Description
2006-11-28	2.1	AUTOSAR Administration	• Legal disclaimer revised
2006-05-16	2.0	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 Content

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.

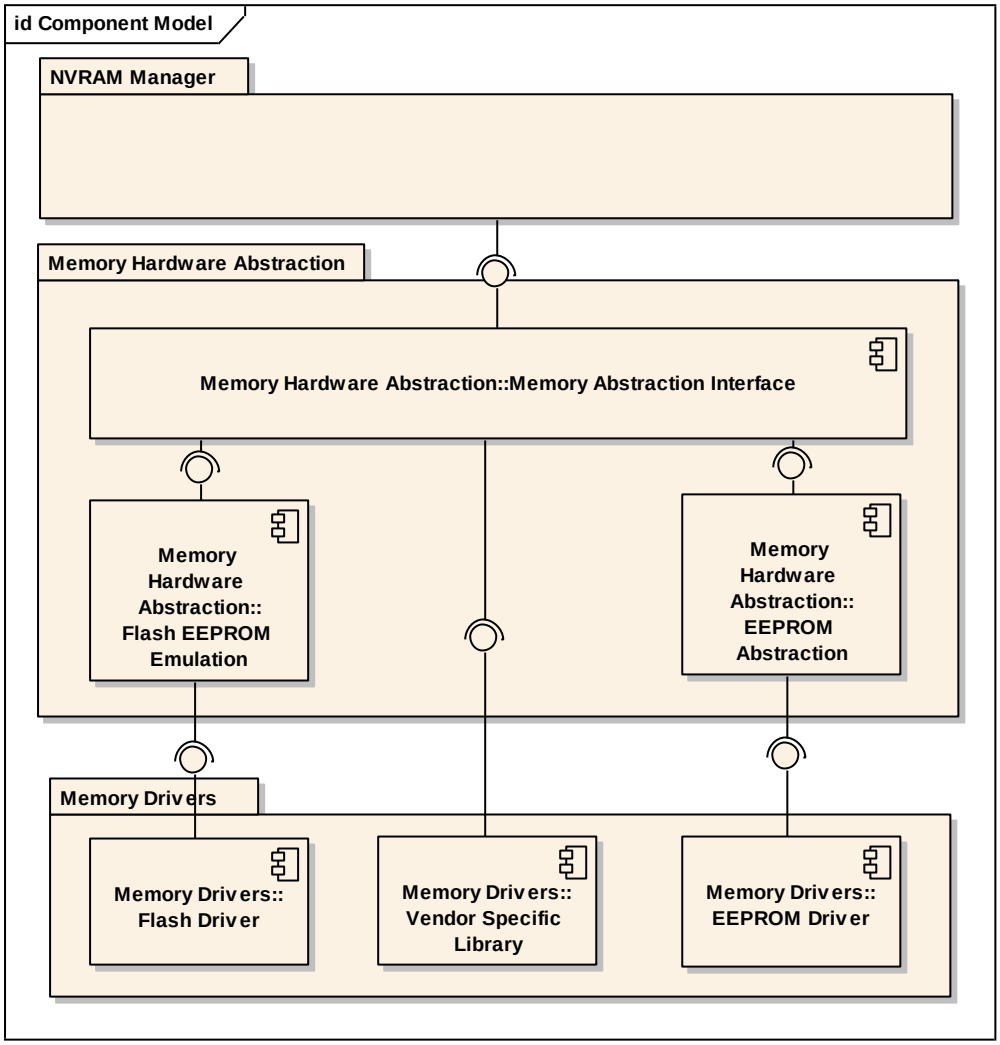


Figure 1: Components and Interfaces of the Memory Hardware Abstraction Layer

The EEPROM Abstraction (EA) module shall abstract from the addressing scheme of the underlying EEPROM driver and provide a uniform addressing scheme. Also it shall allow for a configurable, “virtually unlimited” number of erase-write-cycles. Thus the upper layer (the NVRAM manager) needs not be changed if the underlying EEPROM driver and device is changed.

The Flash EEPROM Emulation (FEE) module shall abstract from the addressing scheme of the underlying flash driver and provide a uniform addressing scheme and a configurable, “virtually unlimited” number of erase-write-cycles. Thus the upper layer (the NVRAM manager) needs not be changed if the underlying flash driver and device is changed.

The driver interface layers (EEPROM and flash interface) have been skipped in order to allow for an efficient implementation of the memory abstraction modules (FEE and EA modules). The FEE and EA directly interface to the underlying memory drivers. The requirements set forth for those interface layers shall instead apply to the memory abstraction interface.

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

In requirements, the following specific semantics are 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 . Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase, or the phrase „SHALL NOT“, means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", mean 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" mean 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, MUST 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, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

The representation of requirements in AUTOSAR documents follows the table specified in [5].

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

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

Acronyms / abbreviations	***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

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

4 Functional Overview

4.1 EEPROM Abstraction Layer

The EEPROM Abstraction Layer (EA) shall extend the EEPROM driver such that it provides upper layers 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.

4.2 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.

4.3 Memory Abstraction Interface

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

5 Requirements Tracing

Requirement	Description	Satisfied by
RS_BFR_01816	-	SRS_MemHwAb_14013, SRS_MemHwAb_14026
RS_BRF_00129	AUTOSAR shall support data corruption detection and protection	SRS_MemHwAb_14014, SRS_MemHwAb_14015, SRS_MemHwAb_14016
RS_BRF_01000	AUTOSAR architecture shall organize the BSW in a hardware independent and a hardware dependent layer	SRS_MemHwAb_14017, SRS_MemHwAb_14018, SRS_MemHwAb_14019, SRS_MemHwAb_14022, SRS_MemHwAb_14024
RS_BRF_01800	AUTOSAR non-volatile memory functionality shall be divided into a hardware dependent and independent layer	SRS_MemHwAb_14017, SRS_MemHwAb_14018, SRS_MemHwAb_14019, SRS_MemHwAb_14022, SRS_MemHwAb_14024
RS_BRF_01808	AUTOSAR non-volatile memory handling shall support different kinds of memory hardware	SRS_MemHwAb_14019, SRS_MemHwAb_14020, SRS_MemHwAb_14021
RS_BRF_01812	AUTOSAR non-volatile memory functionality shall support the prioritization and asynchronous execution of jobs	SRS_MemHwAb_14031
RS_BRF_01816	AUTOSAR non-volatile memory functionality shall organize persistent data based on logical memory blocks	SRS_MemHwAb_14001, SRS_MemHwAb_14002, SRS_MemHwAb_14010, SRS_MemHwAb_14028, SRS_MemHwAb_14029, SRS_MemHwAb_14032
RS_BRF_01832	AUTOSAR non-volatile memory shall handle logical memory blocks independent of its physical address	SRS_MemHwAb_14005, SRS_MemHwAb_14006, SRS_MemHwAb_14007, SRS_MemHwAb_14009
RS_BRF_01840	AUTOSAR non-volatile memory functionality shall secure integrity of memory blocks	SRS_MemHwAb_14014, SRS_MemHwAb_14015, SRS_MemHwAb_14016
RS_BRF_01848	AUTOSAR non-volatile memory functionality shall provide mechanisms to enhance hardware reliability	SRS_MemHwAb_14002, SRS_MemHwAb_14012
RS_BRF_01850	AUTOSAR non-volatile memory functionality shall be able to cope with hardware lifetime constraints	SRS_MemHwAb_14002, SRS_MemHwAb_14012
RS_BRF_02040	AUTOSAR BSW and RTE shall ensure data consistency	SRS_MemHwAb_14015
RS_BRF_02232	AUTOSAR shall support development with run-time assertion checks	SRS_MemHwAb_14023

6 Requirements Specification

6.1 Functional Requirements

6.1.1 Memory Abstraction Modules

6.1.1.1 Configuration

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

[

Type:	Valid
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:	--

](RS_BRF_01816)

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

[

Type:	Valid
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:	--

](RS_BRF_01848, RS_BRF_01850, RS_BRF_01816)

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

[

Type:	Valid
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:	--

](RS_BFR_01816)

6.1.1.2 Initialization

No additional requirements, the “standard” requirements from the general section of the SPAL SRS regarding initialization shall be applied.

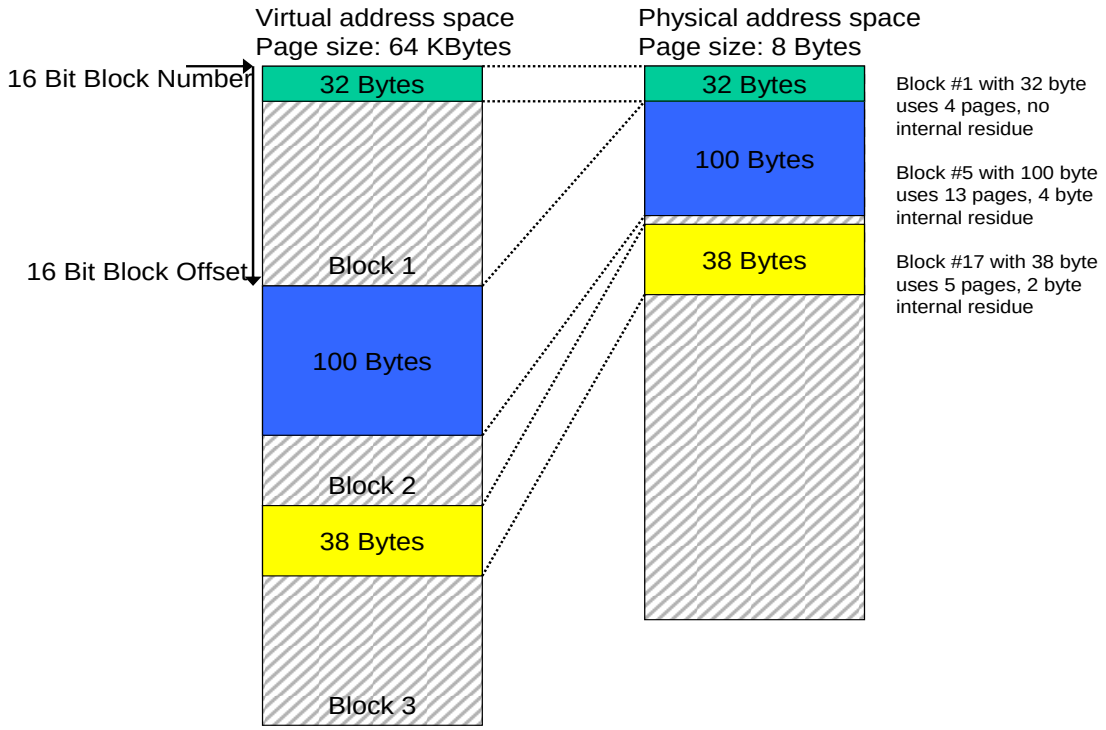
6.1.1.3 Normal Operation

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

[

Type:	Valid
Description:	The Flash EEPROM Emulation (FEE) and EEPROM Abstraction (EA) shall provide upper layers 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 2: Virtual vs. physical address space

](RS_BRF_01832)



Note: Sizes not shown to scale

Figure 2: Virtual vs. physical address space

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

⌈

Type:	Valid
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.

⌋(RS_BRF_01832)

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

⌈

Type:	Valid
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.

](RS_BRF_01832)

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

[

Type:	Valid
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.

](RS_BRF_01832)

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

[

Type:	Valid
Description:	The FEE and EA modules shall provide a write service that operates only on complete configured logical blocks.
Rationale:	Decouple the upper layer from driver internals.
Use Case:	The upper layer 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:	--

](RS_BRF_01816)

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

⌈

Type:	Valid
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:	--

⌋(RS_BRF_01816)

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

⌈

Type:	Valid
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:	--

⌋(RS_BRF_01812)

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

⌈

Type:	Valid
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:	--

⌋(RS_BRF_01816)

6.1.1.3.9 [SRS_MemHwAb_14012] Spreading of write access

⌈

Type:	Valid
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.

⌋(RS_BRF_01848, RS_BRF_01850)

6.1.1.3.10 [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

⌈

Type:	Valid
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:	--

⌋(RS_BFR_01816)

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

⌈

Type:	Valid
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.

](RS_BRF_01816)

6.1.1.4 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).

6.1.1.5 Fault Operation

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

[

Type:	Valid
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.

](RS_BRF_00129,RS_BRF_01840)

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

[

Type:	Valid
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.

](RS_BRF_00129,RS_BRF_01840,RS_BRF_02040)

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

[

Type:	Valid
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.

](RS_BRF_00129,RS_BRF_01840)

6.1.2 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.

6.1.2.1 General

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

⌈

Type:	Valid
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 access to internal and external memory devices without any difference.
Dependencies:	--
Supporting Material:	This requirement shall replace [BSW12172].

⌋(RS_BRF_01000,RS_BRF_01800,RS_BRF_01808)

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

⌈

Type:	Valid
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].

⌋(RS_BRF_01808)

6.1.2.2 Configuration

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

⌈

Type:	Valid
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].

⌋(RS_BRF_01808)

6.1.2.3 Normal Operation

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

⌈

Type:	Valid
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].

⌋(RS_BRF_01000,RS_BRF_01800)

6.1.2.4 Fault Operation

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

⌈

Type:	Valid
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].

](RS_BRF_02232)

6.1.3 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.

6.2 Non-Functional Requirements (Qualities)

6.2.1 Memory Abstraction Modules

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

[

Type:	Valid
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

](RS_BRF_01000,RS_BRF_01800)

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

[

Type:	Valid
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

](RS_BRF_01000,RS_BRF_01800)

6.2.2 Memory Abstraction Interface

6.2.2.1 Timing Requirements

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

[

Type:	Valid
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].

](RS_BRF_01000,RS_BRF_01800)

6.2.3 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.

7 References

7.1 Deliverables of AUTOSAR

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

[5] Software Standardization Template
AUTOSAR_TPS_StandardizationTemplate.pdf

7.2 Related standards and norms

None