Document Title	Specification of Memory Access
Document Category	Software specification (SWS)
Document UID	1017
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	• Restructured the API specification chapter • Removed the Extended Production Errors chapter
2025-11-27	R25-11	AUTOSAR Release Management	• Clarification of the usage of the terms "direct" and "indirect"
2024-11-27	R24-11	AUTOSAR Release Management	• Removed all draft markings • Added SynchronizationGroups • Moved MemAccUseMemFuncPtrTable, MemAccMemNamePrefix and MemAccMemInvocation to MemDrv • Minor corrections and bugfixes
2023-11-23	R23-11	AUTOSAR Release Management	• Fixed inconsistencies
2022-11-24	R22-11	AUTOSAR Release Management	• Renamed MemAcc_MemApiType in MemAcc_MemBinaryHeaderType • Additional DET checks added • Updated Configurable Interface chapter • Corrected Service IDs • Minor corrections and bugfixes • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• 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 Introduction and Functional Overview 7
1.1 Supported Use-Cases . . . 8
2 Acronyms and Abbreviations 9
2.1 Physical Segmentation . . . 10
3 Related Documentation 11
3.1 Input Documents & Related Standards and Norms . . . 11
3.2 Related Specification . . . 11
4 Constraints and Assumptions 12
4.1 Limitations . . . 12
4.1.1 General Limitations . . . 12
4.1.2 Memory Mapped Access . . . 12
4.2 Applicability to Car Domains . . . 12
5 Dependencies to Other Modules 13
6 Requirements Tracing 14
7 Functional Specification 17
7.1 Overview . . . 17
7.2 Key Aspects . . . 17
7.3 Functional Elements . . . 17
7.3.1 Memory Address Translation . . . 17
7.3.1.1 Memory Mapping Constraints . . . 18
7.3.2 Memory Access Coordination . . . 19
7.3.3 Job Management . . . 19
7.3.4 Job Processing . . . 20
7.3.4.1 Job Status . . . 20
7.3.4.2 Job Result . . . 21
7.3.5 Hardware Specific Services . . . 22
7.3.6 Performance Optimization . . . 22
7.3.7 Generic Locking Mechanism . . . 23
7.3.8 Dynamic Memory Driver Handling . . . 23
7.3.8.1 Dynamic Memory Driver Activation . . . 23
7.3.8.2 Service Invocation . . . 24
7.4 Module Handling . . . 24
7.4.1 Initialization . . . 24
7.4.2 Scheduling . . . 24
7.5 General Design Rules . . . 24
7.5.1 Retry Mechanism . . . 25
7.5.2 Address Alignment . . . 25
7.5.3 64-Bit Support . . . 26

7.6 Error Classification . . . 27
7.6.1 Development Errors . . . 27
7.6.2 Runtime Errors . . . 27
7.6.3 Production Errors . . . 27
8 API Specification 28
8.1 Imported Types . . . 28
8.2 Type Definitions . . . 28
8.2.1 MemAcc_AddressAreaIdType . . . 28
8.2.2 MemAcc_AddressType . . . 29
8.2.3 MemAcc_ConfigType . . . 29
8.2.4 MemAcc_DataType . . . 29
8.2.5 MemAcc_JobResultType . . . 30
8.2.6 MemAcc_JobStatusType . . . 30
8.2.7 MemAcc_JobType . . . 31
8.2.8 MemAcc_LengthType . . . 31
8.2.9 MemAcc_MemoryInfoType . . . 32
8.2.10 MemAcc_JobInfoType . . . 33
8.2.11 MemAcc_HwIdType . . . 34
8.2.12 MemAcc_MemBinaryHeaderType . . . 34
8.2.13 MemAcc_MemAddressType . . . 36
8.2.14 MemAcc_MemConfigType . . . 36
8.2.15 MemAcc_MemDataType . . . 36
8.2.16 MemAcc_MemInstanceIdType . . . 37
8.2.17 MemAcc_MemJobResultType . . . 37
8.2.18 MemAcc_MemLengthType . . . 37
8.2.19 MemAcc_MemHwServiceIdType . . . 38
8.2.20 MemAcc_MemInitFuncType . . . 38
8.2.21 MemAcc_MemDeInitFuncType . . . 39
8.2.22 MemAcc_MemGetJobResultFuncType . . . 39
8.2.23 MemAcc_MemSuspendFuncType . . . 40
8.2.24 MemAcc_MemResumeFuncType . . . 40
8.2.25 MemAcc_MemPropagateErrorFuncType . . . 41
8.2.26 MemAcc_MemReadFuncType . . . 41
8.2.27 MemAcc_MemWriteFuncType . . . 42
8.2.28 MemAcc_MemEraseFuncType . . . 42
8.2.29 MemAcc_MemBlankCheckFuncType . . . 43
8.2.30 MemAcc_MemHwSpecificServiceFuncType . . . 44
8.2.31 MemAcc_MemMainFuncType . . . 44
8.2.32 MemAcc_ApplicationLockNotificationType . . . 45
8.2.33 Extension of Std_ReturnType . . . 45
8.3 Function Definitions . . . 45
8.3.1 Synchronous Functions . . . 46
8.3.1.1 MemAcc_Init . . . 46

8.3.1.2 MemAcc_DeInit . . . 46
8.3.1.3 MemAcc_GetJobResult . . . 47
8.3.1.4 MemAcc_GetJobStatus . . . 48
8.3.1.5 MemAcc_GetMemoryInfo . . . 49
8.3.1.6 MemAcc_GetProcessedLength . . . 50
8.3.1.7 MemAcc_GetJobInfo . . . 51
8.3.1.8 MemAcc_ActivateMem . . . 52
8.3.1.9 MemAcc_DeactivateMem . . . 53
8.3.1.10 MemAcc_ReleaseLock . . . 54
8.3.2 Asynchronous Functions . . . 55
8.3.2.1 MemAcc_Cancel . . . 55
8.3.2.2 MemAcc_Read . . . 57
8.3.2.3 MemAcc_Write . . . 58
8.3.2.4 MemAcc_Erase . . . 60
8.3.2.5 MemAcc_Compare . . . 61
8.3.2.6 MemAcc_BlankCheck . . . 63
8.3.2.7 MemAcc_HwSpecificService . . . 64
8.3.2.8 MemAcc_RequestLock . . . 65
8.4 Scheduled Functions . . . 67
8.4.1 MemAcc_MainFunction . . . 67
8.5 Expected Interfaces . . . 68
8.5.1 Mandatory Interfaces . . . 68
8.5.2 Optional Interfaces . . . 69
8.5.3 Configurable Interfaces . . . 69
8.5.3.1 <AddressAreaJobEndNotification> . . . 70
8.6 Service Interfaces . . . 70
9 Sequence Diagrams 71
9.1 Job Handling with Result Polling . . . 71
9.2 Job Handling with Job End Notification . . . 72
9.3 Mem Driver Initialization by MemAcc . . . 72
9.4 Mem Driver Initialization by EcuM . . . 73
9.5 Mem Driver Scheduling by MemAcc . . . 74
9.6 Mem Driver Scheduling by SchM . . . 75
9.7 Generic Lock Sequence . . . 76
10 Configuration Specification 77
10.1 How to Read this Chapter . . . 77
10.2 Containers and Configuration Parameters . . . 77
10.2.1 MemAcc . . . 78
10.2.2 MemAccAddressAreaConfiguration . . . 82
10.2.3 MemAccSubAddressAreaConfiguration . . . 84
10.2.4 MemAccSynchronizationGroupConfiguration . . . 88
10.3 Published Information . . . 89

A Change history of AUTOSAR traceable items 90
A.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 90
A.1.1 Added Specification Items in R26-11 . . . 90
A.1.2 Changed Specification Items in R26-11 . . . 90
A.1.3 Deleted Specification Items in R26-11 . . . 91
A.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 91
A.2.1 Added Specification Items in R25-11 . . . 91
A.2.2 Changed Specification Items in R25-11 . . . 92
A.2.3 Deleted Specification Items in R25-11 . . . 92
A.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 92
A.3.1 Added Specification Items in R24-11 . . . 92
A.3.2 Changed Specification Items in R24-11 . . . 92
A.3.3 Deleted Specification Items in R24-11 . . . 92
A.4 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 93
A.4.1 Added Specification Items in R23-11 . . . 93
A.4.2 Changed Specification Items in R23-11 . . . 93
A.4.3 Deleted Specification Items in R23-11 . . . 93
B Not applicable requirements 94

1 Introduction and Functional Overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module Memory Access (MemAcc).

The Memory Access module provides access to different memory technology devices by an address-based API.
The Memory Access module is always complemented by one or more Memory Driver (Mem). The Memory Access module is memory device technology agnostic and can be used with typical memory devices such as flash, EEPROM, RAM or phase change memory.

The Memory Access module and Memory Driver are located in the same layer of the AUTOSAR architecture as Fls and Eep Driver but split these modules into a hardware independent part (MemAcc) and a hardware dependent part (Mem).

Figure 1.1 shows an example architectural overview with different Memory Drivers and upper layers:

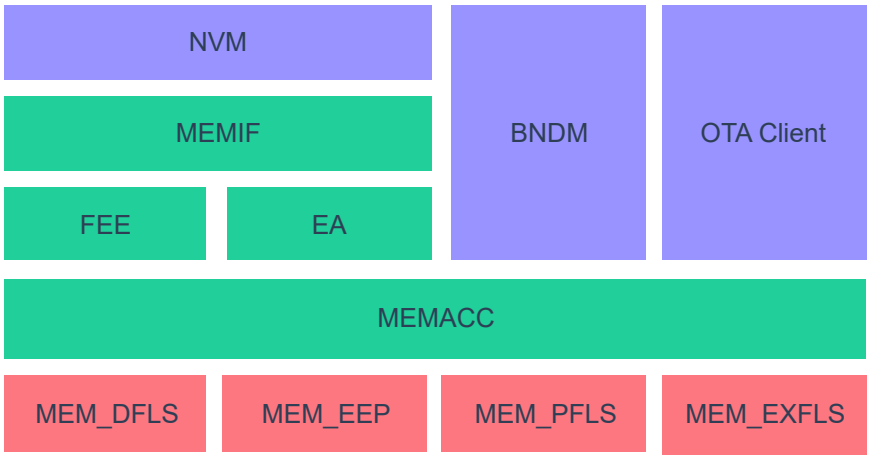


Figure 1.1: MemAcc Architecture Example

1.1 Supported Use-Cases

The combination of MemAcc module and Mem driver supports the following use cases:

- Block based non-volatile memory access for data storage using NvM and Fee or Ea
- OTA software update
- General address-based memory access, e.g. for BndM or flash bootloader usage

Combinations of these use cases are also supported.

2 Acronyms and Abbreviations

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

Abbreviation / Acronym	Description
BndM	Bulk Non-Volatile Data Manager
ECC	Error Correction Code
FOTA	Firmware Over The Air - remote firmware update using wireless communication
HSM	Hardware Security Module - dedicated security MCU core
OTA	Over The Air - general term for wireless communication between OEM backend and vehicle
RWW	Read While Write - capability of a memory device to perform a read operation in one memory bank while at the same time a write/erase operation takes place in another bank
SOTA	Software Over The Air - remote software update using wireless communication

Terms	Description
Address Area	Contiguous memory area in the logical address space Typically multiple physical memory sectors are combined to one logical address area.
Bank	Group of sector batches In case a memory technology is segmented in sectors, a bank is an instance of a sector batch group in which no read-while-write operation is permitted. In case of a flash memory device, this typically maps to an individual flash controller.
Job Request	Memory access request by an upper layer module for an address area.
Memory Device	Group of banks
Page Burst	Aggregated access of memory pages for improved performance In case a memory device technology has a physical segmentation, some memory devices provide an optimized access method to read or write multiple pages at a time. Page burst denotes the aggregation of memory pages used for the access optimization.
Sector	Smallest erasable memory unit (in bytes) Some memory device technologies require an explicit physical erase operation before the memory can be written. A sector defines the minimum size of such an erase unit. Depending on the memory device, sectors can be either uniform- or variable-sized.
Sector Batch	Aggregation of sectors with uniform size Logical aggregation of contiguous sectors with the same size.
Sector Burst	Aggregation of sectors for improved erase performance In case a memory technology needs a physical erase operation, some devices provide an erase performance optimization by erasing an aggregation of sectors in one step.
Sub Address Area	Contiguous memory area in the logical address space mapped to a sector batch of one memory device.

Terms	Description
Write Page	Smallest writeable unit of a memory device (in bytes) Some memory device technologies must be accessed considering a physical segmentation. Hence a byte-wise access is not possible for all memory device technologies. This term defines the minimum size that needs to be written in one access.

2.1 Physical Segmentation

Figure 2.1 gives an overview of the physical segmentation and the according technical terms:

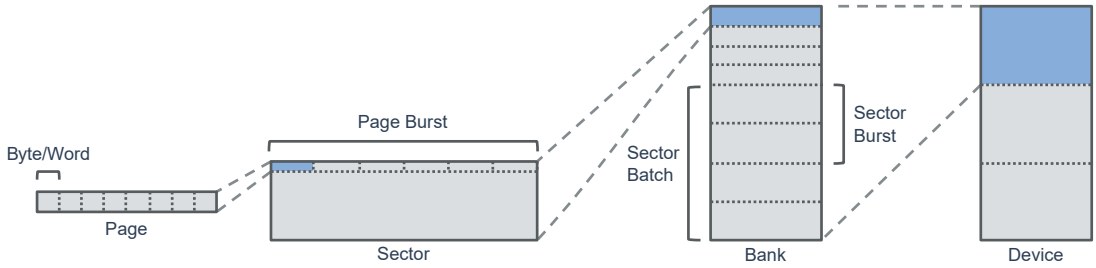


Figure 2.1: Overview of Physical Segmentation

3 Related Documentation

3.1 Input Documents & Related Standards and Norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_CP_RS_MemoryHWAbstractionLayer

[4] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

3.2 Related Specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for MemAcc.

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

4 Constraints and Assumptions

To be able to control and coordinate the access of shared memory resources, all memory upper layers shall use the MemAcc module to access these shared memory resources. The only exception to this constraint are exclusive memory accesses at discrete points in time.

4.1 Limitations

4.1.1 General Limitations

The MemAcc module is targeted for address based memory access. File based access is not considered.

Block based memory devices like NAND flash devices which require an explicit bad block management are out of scope of this specification.

4.1.2 Memory Mapped Access

It's not possible to perform a memory-mapped access on a shared memory resource while at the same time, the AUTOSAR memory stack performs an access on a shared memory resource.
This restriction applies to memory devices like flash or EEPROM where the memory must be put into a special programming mode in which a concurrent read access is not possible. This restriction applies to internal and external shared memory devices and also affects hardware-based flash EEPROM emulations.
In case a memory-mapped access is needed, MemAcc coordination must be implemented at the application level. The application must ensure that no concurrent access is performed on the shared memory.

4.2 Applicability to Car Domains

The MemAcc module can be used in any domain application that needs MemAcc to either store data or perform a software update.

5 Dependencies to Other Modules

The MemAcc module has interfaces towards the Flash EEPROM Emulation (Fee), the EEPROM Abstraction (Ea), Bulk Nv Data Manager (BndM), Memory Drivers (Mem), the Default Error Tracer (DET) and, in case of a OTA software update client, also to Complex Device Drivers (CDD).
The MemAcc module includes header files of DET and MemMap.

6 Requirements Tracing

The following tables reference the requirements specified in [3] and [4] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_MemAcc_00002]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_MemAcc_00002]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_MemAcc_00002] [SWS_MemAcc_00031] [SWS_MemAcc_00034] [SWS_MemAcc_00036] [SWS_MemAcc_00037] [SWS_MemAcc_00039] [SWS_MemAcc_00041] [SWS_MemAcc_00042] [SWS_MemAcc_00044] [SWS_MemAcc_00045] [SWS_MemAcc_00046] [SWS_MemAcc_00047] [SWS_MemAcc_00049] [SWS_MemAcc_00050] [SWS_MemAcc_00051] [SWS_MemAcc_00052] [SWS_MemAcc_00054] [SWS_MemAcc_00055] [SWS_MemAcc_00056] [SWS_MemAcc_00058] [SWS_MemAcc_00059] [SWS_MemAcc_00060] [SWS_MemAcc_00061] [SWS_MemAcc_00063] [SWS_MemAcc_00064] [SWS_MemAcc_00065] [SWS_MemAcc_00067] [SWS_MemAcc_00068] [SWS_MemAcc_00070] [SWS_MemAcc_00071] [SWS_MemAcc_00072] [SWS_MemAcc_00073] [SWS_MemAcc_00077] [SWS_MemAcc_00093] [SWS_MemAcc_00124]
[SRS_BSW_00327]	Error values naming convention	[SWS_MemAcc_10038]
[SRS_BSW_00337]	Classification of development errors	[SWS_MemAcc_10038]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_MemAcc_10038]
[SRS_BSW_00406]	API handling in uninitialized state	[SWS_MemAcc_00030] [SWS_MemAcc_00033] [SWS_MemAcc_00035] [SWS_MemAcc_00038] [SWS_MemAcc_00040] [SWS_MemAcc_00043] [SWS_MemAcc_00048] [SWS_MemAcc_00053] [SWS_MemAcc_00057] [SWS_MemAcc_00062] [SWS_MemAcc_00066] [SWS_MemAcc_00076] [SWS_MemAcc_00088] [SWS_MemAcc_00090] [SWS_MemAcc_00099] [SWS_MemAcc_00117]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_MemAcc_10002] [SWS_MemAcc_91012]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_MemAcc_10035] [SWS_MemAcc_10036]
[SRS_MemHwAb_-14002]	The FEE and EA modules shall allow the configuration of a required number of write cycles for each logical block	[SWS_MemAcc_00005] [SWS_MemAcc_00100]



△

Requirement	Description	Satisfied by
[SRS_MemHwAb_-14031]	The FEE and EA modules shall provide a service that allows canceling an ongoing asynchronous operation	[SWS_MemAcc_00029] [SWS_MemAcc_00123]
[SRS_MemHwAb_-14034]	MemAcc module shall allow the configuration of the priority for different logical address areas	[SWS_MemAcc_00014] [SWS_MemAcc_00017] [SWS_MemAcc_00024]
[SRS_MemHwAb_-14035]	MemAcc module shall support variant mapping	[SWS_MemAcc_10015]
[SRS_MemHwAb_-14037]	MemAcc module and Mem driver shall provide an interface for initialization	[SWS_MemAcc_00025] [SWS_MemAcc_10015] [SWS_MemAcc_10041]
[SRS_MemHwAb_-14038]	MemAcc module and Mem driver shall provide asynchronous memory access functions	[SWS_MemAcc_00028] [SWS_MemAcc_00076] [SWS_MemAcc_00115] [SWS_MemAcc_10018] [SWS_MemAcc_10023] [SWS_MemAcc_10024] [SWS_MemAcc_10025] [SWS_MemAcc_10026] [SWS_MemAcc_10027] [SWS_MemAcc_10028] [SWS_MemAcc_10030]
[SRS_MemHwAb_-14039]	MemAcc module and Mem driver shall support optional services	[SWS_MemAcc_10018] [SWS_MemAcc_10023] [SWS_MemAcc_10024] [SWS_MemAcc_10025] [SWS_MemAcc_10026] [SWS_MemAcc_10027] [SWS_MemAcc_10028] [SWS_MemAcc_10030]
[SRS_MemHwAb_-14040]	MemAcc module and Mem driver shall provide a synchronous status function	[SWS_MemAcc_00020] [SWS_MemAcc_00021] [SWS_MemAcc_00092] [SWS_MemAcc_00104] [SWS_MemAcc_00105] [SWS_MemAcc_00106] [SWS_MemAcc_00107] [SWS_MemAcc_00108] [SWS_MemAcc_00109] [SWS_MemAcc_00112] [SWS_MemAcc_00113] [SWS_MemAcc_00114] [SWS_MemAcc_00118] [SWS_MemAcc_00119] [SWS_MemAcc_00120] [SWS_MemAcc_10009] [SWS_MemAcc_10011] [SWS_MemAcc_10013] [SWS_MemAcc_10019] [SWS_MemAcc_10021] [SWS_MemAcc_10022] [SWS_MemAcc_10037] [SWS_MemAcc_10039] [SWS_MemAcc_10040] [SWS_MemAcc_91016]
[SRS_MemHwAb_-14041]	MemAcc module shall provide a job notification mechanism for the upper layer modules	[SWS_MemAcc_00015] [SWS_MemAcc_10029]
[SRS_MemHwAb_-14042]	MemAcc module shall support multiple Mem drivers for different types of memory	[SWS_MemAcc_00098] [SWS_MemAcc_10010]
[SRS_MemHwAb_-14043]	Mem driver and shall support multiple instances of the same memory device	[SWS_MemAcc_91011]
[SRS_MemHwAb_-14044]	MemAcc module shall manage the memory job requests from different upper layer modules	[SWS_MemAcc_00006] [SWS_MemAcc_00007] [SWS_MemAcc_00008] [SWS_MemAcc_00028]
[SRS_MemHwAb_-14045]	MemAcc module and Mem driver shall provide measures for dynamic driver activation	[SWS_MemAcc_00085] [SWS_MemAcc_00089] [SWS_MemAcc_00121] [SWS_MemAcc_00122] [SWS_MemAcc_10014] [SWS_MemAcc_10033] [SWS_MemAcc_10034] [SWS_MemAcc_91000] [SWS_MemAcc_91001] [SWS_MemAcc_91002] [SWS_MemAcc_91003] [SWS_MemAcc_91004] [SWS_MemAcc_91005] [SWS_MemAcc_91006] [SWS_MemAcc_91007] [SWS_MemAcc_91008] [SWS_MemAcc_91009] [SWS_MemAcc_91010] [SWS_MemAcc_91018]

▽

△

Requirement	Description	Satisfied by
[SRS_MemHwAb_-14046]	MemAcc module and Mem driver shall provide support for 64-Bit address range	[SWS_MemAcc_00081] [SWS_MemAcc_00082] [SWS_MemAcc_10001] [SWS_MemAcc_10007] [SWS_MemAcc_91013] [SWS_MemAcc_91014]
[SRS_MemHwAb_-14047]	MemAcc module shall provide optional support for the initialization and main function triggering of memory drivers	[SWS_MemAcc_00025] [SWS_MemAcc_00084] [SWS_MemAcc_00111] [SWS_MemAcc_00121] [SWS_MemAcc_00122] [SWS_MemAcc_10017] [SWS_MemAcc_10033] [SWS_MemAcc_10034]
[SRS_MemHwAb_-14048]	Mem driver shall operate on physical segmentation/physical addresses	[SWS_MemAcc_00003] [SWS_MemAcc_00004] [SWS_MemAcc_00087] [SWS_MemAcc_00101] [SWS_MemAcc_00102] [SWS_MemAcc_00125] [SWS_MemAcc_00126] [SWS_MemAcc_00127]
[SRS_MemHwAb_-14049]	Mem driver shall use a standard binary format for dynamic driver activation	[SWS_MemAcc_00089]
[SRS_MemHwAb_-14051]	Mem driver shall not buffer data	[SWS_MemAcc_10004] [SWS_MemAcc_91020]
[SRS_MemHwAb_-14054]	MemAcc module shall provide a function to retrieve memory segmentation information	[SWS_MemAcc_10012] [SWS_MemAcc_10020]
[SRS_MemHwAb_-14055]	MemAcc module shall provide a lock function to enable/disable the direct memory access from application	[SWS_MemAcc_00116] [SWS_MemAcc_10030] [SWS_MemAcc_10031]
[SRS_MemHwAb_-14056]	MemAcc module and Mem driver shall provide a generic function to access the hardware specific functionalities	[SWS_MemAcc_00083] [SWS_MemAcc_10008] [SWS_MemAcc_10028]
[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	[SWS_MemAcc_00012] [SWS_MemAcc_00018] [SWS_MemAcc_00078] [SWS_MemAcc_00079] [SWS_MemAcc_00080] [SWS_MemAcc_10000]

Table 6.1: Requirements Tracing

7 Functional Specification

This chapter defines the behavior of the MemAcc module.
The API of the module is defined in chapter 8, while the configuration is defined in 10.

7.1 Overview

The MemAcc module provides a memory device agnostic address-based memory access for different upper layers modules. It implements all high level functionality like job management, access coordination and allocation of Memory Driver access requests according to the physical segmentation as the Memory Drivers expect all memory accesses aligned to physical segments.

7.2 Key Aspects

- Access coordination of different upper layers like Fee, Ea and OTA software update client
- Memory technology device agnostic API supporting all kinds of memories including code memories
- Coordination of CPU cores like host- and HSM core
- Memory access job management
- Virtualization of memory areas to support mapping of non-contiguous areas as well as spanning areas across different memory devices
- Mapping of memory locations to memory devices
- Splitting of Memory Driver access requests according to physical segments like pages and sectors for flash memories

7.3 Functional Elements

7.3.1 Memory Address Translation

The MemAcc module abstracts the physical memory addressing scheme of the Mem driver to the upper layer by means of a logical address space.

Figure 7.1 provides an overview of the memory address translation/mapping scheme:

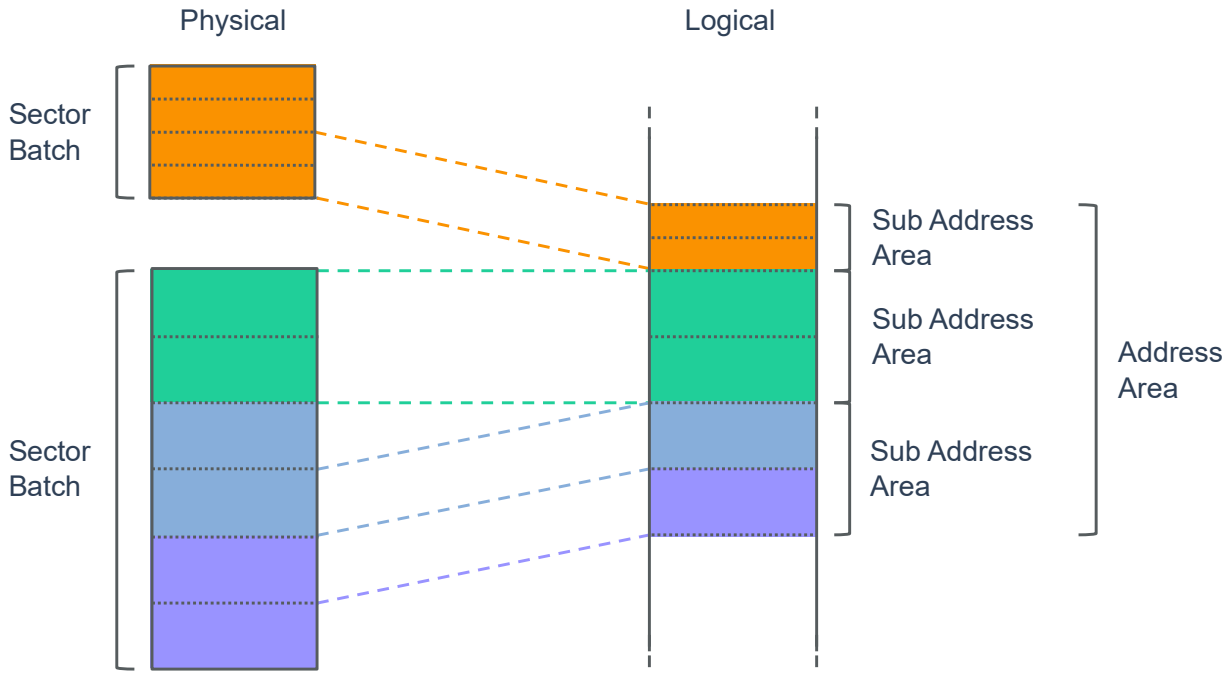


Figure 7.1: Overview Memory Address Translation Scheme

[SWS_MemAcc_00012]

Upstream requirements: SRS_MemHwAb_14057

⌈All MemAcc services, i.e. erase, read and write, shall support access requests which cross memory device boundaries based on the logical/physical memory mapping.⌋

7.3.1.1 Memory Mapping Constraints

[SWS_MemAcc_00078]

Upstream requirements: SRS_MemHwAb_14057

⌈An address area shall only be assigned to one upper layer module.⌋

[SWS_MemAcc_00079]

Upstream requirements: SRS_MemHwAb_14057

⌈Within a sub-address area, only a uniform sector size is allowed.⌋

Note: A sub-address area maps to a sector batch.

[SWS_MemAcc_00080]

Upstream requirements: SRS_MemHwAb_14057

⌈Start address and length of address areas shall be aligned to the physical sectors.⌋

Note: Start address and length of memory accesses have to be aligned to the physical segmentation.

7.3.2 Memory Access Coordination

[SWS_MemAcc_00006]

Upstream requirements: SRS_MemHwAb_14044

⌈The MemAcc module shall coordinate conflicting memory accesses by multiple upper layers.⌋

Note: Typically, code- and data flash share the same flash controller and therefore it's not possible to perform a write access at the same time. Since code- and data flash write access might happen at the same time for the software update use case, MemAcc needs to coordinate these accesses.

[SWS_MemAcc_00007]

Upstream requirements: SRS_MemHwAb_14044

⌈The MemAcc module shall only coordinate conflicting resource accesses. The access dependencies shall be configurable in the configuration tool.⌋

Note: Only relevant resource conflicts shall be coordinated to prevent any performance impact.

[SWS_MemAcc_00008]

Upstream requirements: SRS_MemHwAb_14044

⌈The MemAcc module shall support multiple memory access requests from different upper layers for distinct memory areas at the same time. In case there is a hardware resource conflict, the memory stack shall still accept the access request and process it once the resource is free.⌋

Note: The AUTOSAR BSW upper layers shall not have to deal with any retry mechanisms as this would affect every upper layer.

7.3.3 Job Management

In general, all MemAcc services that need a significant amount of time to process an operation are defined as asynchronous services. Therefore, MemAcc job requests get only queued by the asynchronous services like `MemAcc_Read` and the processing of the queued jobs happen in the `MemAcc_MainFunction`.

[SWS_MemAcc_00018]

Upstream requirements: SRS_MemHwAb_14057

⌈The MemAcc module shall allow only one job request per address area.⌋

Note: Simplification of job management since one address area can only have one upper layer and job request are typically requested sequentially from the upper layer.

[SWS_MemAcc_00014]

Upstream requirements: SRS_MemHwAb_14034

⌈Based on MemAccAddressAreaPriority, MemAcc shall prioritize memory access requests from AUTOSAR BSW upper layer modules.⌋

Note: Writing crash non-volatile data shall have priority over background software update tasks.

[SWS_MemAcc_00024]

Upstream requirements: SRS_MemHwAb_14034

⌈The prioritization of memory operations shall use the Mem_Suspend and Mem_Resume service if the memory hardware supports this functionality.
If the memory hardware does not support a suspend/resume functionality, the prioritization shall be implemented on a page/page burst, respectively sector/sector burst basis.⌋

7.3.4 Job Processing

[SWS_MemAcc_00015]

Upstream requirements: SRS_MemHwAb_14041

⌈If a job end notification function is configured by MemAccJobEndNotificationName, MemAcc shall notify the upper layer BSW module by calling the configured notification function.⌋

Note: In case no notification function is configured, the upper layer BSW module has to poll the MemAcc job status.

[SWS_MemAcc_00017]

Upstream requirements: SRS_MemHwAb_14034

⌈If the MemAcc module is not able to process a job request, e.g. due to a pending request on the same address area or due to an invalid parameter, the job request shall be rejected by an E_NOT_OK return code.⌋

7.3.4.1 Job Status

The MemAcc module provides the current job processing status information via the MemAcc_GetJobStatus service.

[SWS_MemAcc_00113]

Upstream requirements: SRS_MemHwAb_14040

⌈After initialization via the MemAcc_Init service, the job precessing status shall be set to MEMACC_JOB_IDLE.⌋

[SWS_MemAcc_00104]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the job processing was completed or no job is currently pending, the job processing status shall be set to `MEMACC_JOB_IDLE`.⌋

[SWS_MemAcc_00020]

Upstream requirements: SRS_MemHwAb_14040

⌈Once a job request was accepted, the job processing status shall be set to `MEMACC_JOB_PENDING`.⌋

7.3.4.2 Job Result

The results of the last MemAcc job is provided by the `MemAcc_GetJobResult` service. It can be used by upper layer modules to retrieve detailed information for fine-tuned fault handling.

[SWS_MemAcc_00112]

Upstream requirements: SRS_MemHwAb_14040

⌈After initialization via the `MemAcc_Init` service, the job result shall be set to `MEMACC_OK`.⌋

[SWS_MemAcc_00105]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the job processing was completed successfully, the job result shall be set to `MEMACC_OK`.⌋

[SWS_MemAcc_00106]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the job processing was completed but the results of the last MemAcc job didn't meet the expected result, e.g. a blank check operation was applied on a non-blank memory area, the job result shall be set to `MEMACC_INCONSISTENT`.⌋

[SWS_MemAcc_00107]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the last memory operation was completed but the ECC circuit corrected an ECC error, the job result shall be set to `MEMACC_ECC_CORRECTED`.⌋

[SWS_MemAcc_00108]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the last memory operation didn't complete due to an uncorrectable ECC error, the job result shall be set to `MEMACC_ECC_UNCORRECTED`.⌋

[SWS_MemAcc_00021]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the last memory operation was canceled, the job result shall be set to `MEMACC_CANCELED`.⌋

[SWS_MemAcc_00109]

Upstream requirements: SRS_MemHwAb_14040

⌈In case the memory operation was not successfully completed for any other reason, the job result shall be set to `MEMACC_FAILED`.⌋

7.3.5 Hardware Specific Services

To support memory device specific services, the MemAcc module provides a generic API to call hardware specific Mem driver services - see `MemAcc_HwSpecificService`.
Each Mem driver can have multiple hardware specific services which are selected by the `hwServiceId` parameter.

Note: By providing a generic API for hardware specific services, the MemAcc module can be kept hardware independent.

7.3.6 Performance Optimization

Some Mem drivers provide burst capabilities, i.e. instead of writing/erasing one smallest possible unit, several of these units are written/erased at once to increase the write/erase throughput. Depending on the hardware capabilities, a Mem driver might offer two burst modes:

- Erase multiple sectors
- Write multiple pages

[SWS_MemAcc_00087]

Upstream requirements: SRS_MemHwAb_14048

⌈If enabled by `MemAccUseEraseBurst`, MemAcc shall align and split the Mem driver erase requests according to the erase burst size of the Mem driver.⌋

[SWS_MemAcc_00101]

Upstream requirements: SRS_MemHwAb_14048

⌈MemAcc shall split the Mem driver read requests according to the maximum read length defined by `MemMaxRead` if the length of the MemAcc read request is larger than the `MemMaxRead` value of the respective Mem driver.⌋

[SWS_MemAcc_00102]

Upstream requirements: SRS_MemHwAb_14048

⌈If enabled by `MemAccUseWriteBurst`, MemAcc shall align and split the Mem driver write requests according to the write burst size of the Mem driver.⌋

Note: Enabling burst mode also increases the latency when a job shall be processed with a higher priority. Therefore, system integrators have to consider the maximum latency when configuring the burst modes.

7.3.7 Generic Locking Mechanism

To support upper layers like the BndM, the MemAcc module provides a generic lock API which can be used to restrict the memory access by a certain Mem driver, e.g., if an upper layer wants to do a direct memory mapped access.

Figure 9.7 shows an example lock/unlock sequence.

Note: The application or upper layer module has to maintain the lock state and release the lock once the direct memory access was completed. For the sake of simplicity, nested locks are not supported.

7.3.8 Dynamic Memory Driver Handling

For some safety-relevant use cases, it is not desirable for the Mem driver to be permanently available in an executable form, e.g. to prevent accidental overwriting of memory areas. For these use cases, the Mem driver is compiled as a separate binary which contains a function pointer table to expose the Mem driver service functions to the MemAcc module and the MemAcc module calls the Mem driver service functions indirectly using the Mem driver function pointer table.

7.3.8.1 Dynamic Memory Driver Activation

For the dynamic memory driver activation, the upper layer module has to ensure that the Mem driver binary is available for execution, e.g. downloaded to RAM and initialized before any MemAcc job is requested for the according address area.

7.3.8.2 Service Invocation

[SWS_MemAcc_00085]

Upstream requirements: SRS_MemHwAb_14045

⌈If `MemInvocation` (see [ECUC_Mem_00025]) is set to `DIRECT_STATIC`, MemAcc shall call the Mem driver service functions. Otherwise the MemAcc shall call the Mem driver service functions via the Mem driver function pointer table.⌋

7.4 Module Handling

7.4.1 Initialization

The MemAcc module is initialized via `MemAcc_Init`. Except for `MemAcc_Init`, the API functions of the MemAcc module may only be called after the module has been properly initialized.
Depending on the `MemInvocation` (see [ECUC_Mem_00025]) attribute, MemAcc can also initialize the Mem driver's individual initialization functions.

Figure 9.3 shows the Mem driver initialization via MemAcc while figure 9.4 shows the Mem driver initialization via EcuM.

7.4.2 Scheduling

Since most of the MemAcc module services are asynchronous services, the `MemAcc_MainFunction` needs to be cyclically triggered.
Depending on the `MemInvocation` (see [ECUC_Mem_00025]) attribute, MemAcc can call all Mem main functions within `MemAcc_MainFunction`.

Note: In case Mem drivers shall be dynamically activated, the scheduling of the Mem driver main functions cannot be done via the SchM. Therefore, the MemAcc module has to take care of the main function triggering depending on the individual Mem driver state.

Figure 9.5 shows the Mem main function triggering via MemAcc while figure 9.6 shows the Mem main function triggering via SchM.

7.5 General Design Rules

[SWS_MemAcc_00083]

Upstream requirements: SRS_MemHwAb_14056

⌈The MemAcc module implementation shall be hardware independent.⌋

Note: The MemAcc module will be used with different kinds of Mem drivers, e.g., for internal and external memory devices. Thus, MemAcc has to be completely hardware independent.

[SWS_MemAcc_00098]

Upstream requirements: SRS_MemHwAb_14042

⌈The MemAcc module implementation shall support multiple Mem drivers.⌋

[SWS_MemAcc_00002]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00167, SRS_BSW_00004

⌈The MemAcc module shall check static configuration parameters statically (at the latest during compile time) for correctness.⌋

7.5.1 Retry Mechanism

[SWS_MemAcc_00005]

Upstream requirements: SRS_MemHwAb_14002

⌈If `MemAccNumberOfEraseRetries` is set to a value > 0, MemAcc shall retry the Mem driver erase operation according to the configured value if the Mem driver erase operation resulted in `MEM_JOB_FAILED`.⌋

[SWS_MemAcc_00100]

Upstream requirements: SRS_MemHwAb_14002

⌈If `MemAccNumberOfWriteRetries` is set to a value > 0, MemAcc shall retry the Mem driver write operation according to the configured value if the Mem driver write operation resulted in `MEM_JOB_FAILED`.⌋

Note: Upper layers shall not have to deal with transient memory write/erase issues. The retry mechanism strongly depends on the underlying memory technology/devices types. Not all memory devices support a write retry.

7.5.2 Address Alignment

The MemAcc module does not perform any kind of buffer alignment. Therefore, the buffers provided by the upper layers need to consider already the alignment requirements defined by the `MemAccBufferAlignmentValue` attribute.
`MemAccBufferAlignmentValue` must be configured to the least common multiple value needed by the underlying Mem drivers. The same applies to the write page requirements.

[SWS_MemAcc_00003]

Upstream requirements: SRS_MemHwAb_14048

⌈The MemAcc module shall split memory write access requests for the Mem driver layer according to page/page burst size defined by `MemWritePageSize`/`MemWriteBurstSize`.⌋

Note: Mem driver expects request aligned to the physical write segmentation.

[SWS_MemAcc_00004]

Upstream requirements: SRS_MemHwAb_14048

⌈If the start address or the length of a memory write request does not match the physical write segmentation of the device defined by `MemWritePageSize`, `MemAcc_Write` shall reject such job requests with `E_NOT_OK`.⌋

Note: Memory requests must be aligned to the physical memory segmentation.

[SWS_MemAcc_00125]

Upstream requirements: SRS_MemHwAb_14048

⌈If the start address or the length of a memory read request does not match the minimum read size of the device defined by `MemMinReadSize`, `MemAcc_Read` shall reject such job requests with `E_NOT_OK`.⌋

[SWS_MemAcc_00126]

Upstream requirements: SRS_MemHwAb_14048

⌈If the start address or the length of a memory erase request does not match the physical erase segmentation of the device defined by `MemEraseSectorSize`, `MemAcc_Erase` shall reject such job requests with `E_NOT_OK`.⌋

[SWS_MemAcc_00127]

Upstream requirements: SRS_MemHwAb_14048

⌈The MemAcc module shall split memory erase access requests for the Mem driver layer according to sector/sector burst sizes defined by `MemEraseSectorSize`/`MemEraseBurstSize`.⌋

Note: Mem driver expects request aligned to the physical erase segmentation.

7.5.3 64-Bit Support

[SWS_MemAcc_00081]

Upstream requirements: SRS_MemHwAb_14046

⌈The MemAcc module shall support address areas larger than 4GBytes, thus `MemAcc_AddressType` and `MemAcc_LengthType` shall be defined as a 64-Bit types in case the address area configuration of one address area exceeds 4GBytes.⌋

[SWS_MemAcc_00082]

Upstream requirements: SRS_MemHwAb_14046

⌈If all address areas don't exceed 4GBytes, `MemAcc_AddressType` and `MemAcc_LengthType` shall be defined as a 32-Bit types.⌋

Note: Avoid unnecessary overhead due to 64-Bit types.

7.6 Error Classification

7.6.1 Development Errors

[SWS_MemAcc_10038] Definition of development errors in module MemAcc

Upstream requirements: SRS_BSW_00337, SRS_BSW_00386, SRS_BSW_00327

⌈

Type of error	*Related error code*	*Error value*
API service called without module initialization	MEMACC_E_UNINIT	0x01
API service called with NULL pointer argument	MEMACC_E_PARAM_POINTER	0x02
API service called with wrong address area ID	MEMACC_E_PARAM_ADDRESS_AREA_ID	0x03
API service called with address and length not belonging to the passed address area ID	MEMACC_E_PARAM_ADDRESS_LENGTH	0x04
API service called with a hardware ID not belonging to the passed address area ID	MEMACC_E_PARAM_HW_ID	0x05
API service called for an address area ID with a pending job request	MEMACC_E_BUSY	0x06
Dynamic MEM driver activation failed due to inconsistent MEM driver binary	MEMACC_E_MEM_INIT_FAILED	0x07

⌋

7.6.2 Runtime Errors

There are no runtime errors.

7.6.3 Production Errors

There are no production errors.

8 API Specification

8.1 Imported Types

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

[SWS_MemAcc_10037] Definition of imported datatypes of module MemAcc

Upstream requirements: SRS_MemHwAb_14040

⌈

Module	*Header File*	*Imported Type*
Mem	Mem.h	Mem_AddressType
	Mem.h	Mem_ConfigType
	Mem.h	Mem_DataType
	Mem.h	Mem_HwServiceIdType
	Mem.h	Mem_InstanceIdType
	Mem.h	Mem_LengthType
Std	Std_Types.h	Std_ReturnType

⌋

8.2 Type Definitions

8.2.1 MemAcc_AddressAreaIdType

[SWS_MemAcc_10000] Definition of datatype MemAcc_AddressAreaIdType

Upstream requirements: SRS_MemHwAb_14057

⌈

Name	MemAcc_AddressAreaIdType
Kind	Type
Derived from	uint16
Description	Unique address area ID type.
Available via	MemAcc_GeneralTypes.h

⌋

8.2.2 MemAcc_AddressType

[SWS_MemAcc_10001] Definition of datatype MemAcc_AddressType

Upstream requirements: SRS_MemHwAb_14046

⌈

Name	MemAcc_AddressType	
Kind	Type	
Derived from	***Basetype***	***Variation***
	uint32	–
	uint64	–
Description	Logical memory address type.	
Available via	MemAcc_GeneralTypes.h	

⌋

8.2.3 MemAcc_ConfigType

[SWS_MemAcc_10002] Definition of datatype MemAcc_ConfigType

Upstream requirements: SRS_BSW_00414

⌈

Name	MemAcc_ConfigType
Kind	Structure
Description	Postbuild configuration structure type.
Available via	MemAcc.h

⌋

8.2.4 MemAcc_DataType

[SWS_MemAcc_10004] Definition of datatype MemAcc_DataType

Upstream requirements: SRS_MemHwAb_14051

⌈

Name	MemAcc_DataType
Kind	Type
Derived from	uint8
Description	General data type.
Available via	MemAcc_GeneralTypes.h

⌋

8.2.5 MemAcc_JobResultType

[SWS_MemAcc_10039] Definition of datatype MemAcc_JobResultType

Upstream requirements: SRS_MemHwAb_14040

⌈

Name	MemAcc_JobResultType		
Kind	Enumeration		
Range	MEMACC_OK	0x00	The last MemAcc job was finished successfully
	MEMACC_FAILED	0x01	The last MemAcc job resulted in an unspecific failure and the job was not completed
	MEMACC_INCONSISTENT	0x02	The results of the last MemAcc job didn't meet the expected result, e.g. a blank check operation was applied on a non-blank memory area
	MEMACC_CANCELED	0x03	The last MemAcc job was canceled
	MEMACC_ECC_ UNCORRECTED	0x04	The last memory operation returned an uncorrectable ECC error
	MEMACC_ECC_ CORRECTED	0x05	The last memory operation returned a correctable ECC error
	MEMACC_MEM_ SERVICE_NOT_AVAIL	0x06	The service requested is not available for this memory driver
Description	Asynchronous job result type.		
Available via	MemAcc_GeneralTypes.h		

⌋

8.2.6 MemAcc_JobStatusType

[SWS_MemAcc_10009] Definition of datatype MemAcc_JobStatusType

Upstream requirements: SRS_MemHwAb_14040

⌈

Name	MemAcc_JobStatusType		
Kind	Enumeration		
Range	MEMACC_JOB_IDLE	0x00	Job processing was completed or no job currently pending
	MEMACC_JOB_PENDING	0x01	A job is currently being processed
Description	Asynchronous job status type.		
Available via	MemAcc_GeneralTypes.h		

⌋

8.2.7 MemAcc_JobType

[SWS_MemAcc_10011] Definition of datatype MemAcc_JobType

Upstream requirements: SRS_MemHwAb_14040

⌈

Name	MemAcc_JobType		
Kind	Enumeration		
Range	MEMACC_NO_JOB	0x00	No job currently pending
	MEMACC_WRITE_JOB	0x01	Write job pending
	MEMACC_READ_JOB	0x02	Read job pending
	MEMACC_COMPARE_JOB	0x03	Compare job pending
	MEMACC_ERASE_JOB	0x04	Erase job pending
	MEMACC_ MEMHWSPECIFIC_JOB	0x05	Hardware specific job pending
	MEMACC_BLANKCHECK_ JOB	0x06	Blank check job pending
	MEMACC_ REQUESTLOCK_JOB	0x07	Request lock job pending
Description	Type for asynchronous jobs.		
Available via	MemAcc_GeneralTypes.h		

⌋

8.2.8 MemAcc_LengthType

[SWS_MemAcc_10007] Definition of datatype MemAcc_LengthType

Upstream requirements: SRS_MemHwAb_14046

⌈

Name	MemAcc_LengthType	
Kind	Type	
Derived from	***Basetype***	***Variation***
	uint32	–
	uint64	–
Description	Job length type.	
Available via	MemAcc_GeneralTypes.h	

⌋

8.2.9 MemAcc_MemoryInfoType

[SWS_MemAcc_10012] Definition of datatype MemAcc_MemoryInfoType

Upstream requirements: SRS_MemHwAb_14054

⌈

Name	MemAcc_MemoryInfoType	
Kind	Structure	
Elements	LogicalStartAddress	
	Type	MemAcc_AddressType
	Comment	Logical start address of sub address area
	PhysicalStartAddress	
	Type	MemAcc_AddressType
	Comment	Physical start address of sub address area
	MaxOffset	
	Type	MemAcc_LengthType
	Comment	Size of sub address area in bytes -1
	EraseSectorSize	
	Type	uint32
	Comment	Size of a sector in bytes
	EraseSectorBurstSize	
	Type	uint32
	Comment	Size of a sector burst in bytes. Equals SectorSize in case burst is disabled
	MinReadSize	
	Type	uint32
	Comment	Smallest readable unit in bytes
	WritePageSize	
	Type	uint32
	Comment	Write size of a page in bytes
	MaxReadSize	
	Type	uint32
	Comment	Largest readable unit in bytes
	WritePageBurstSize	
	Type	uint32
	Comment	Size of a page burst in bytes. Equals WritePageSize in case burst is disabled
	HwId	
	Type	MemAcc_HwIdType
	Comment	Referenced memory driver hardware identifier
Description	This structure contains information of Mem device characteristics. It can be accessed via the MemAcc_GetMemoryInfo() service.	
Available via	MemAcc_GeneralTypes.h	

⌋

8.2.10 MemAcc_JobInfoType

[SWS_MemAcc_10013] Definition of datatype MemAcc_JobInfoType

Upstream requirements: SRS_MemHwAb_14040

⌈

Name	MemAcc_JobInfoType	
Kind	Structure	
Elements	LogicalAddress	
	Type	MemAcc_AddressType
	Comment	Address of currently active address area request
	Length	
	Type	MemAcc_LengthType
	Comment	Length of the currently active address area request
	HwId	
	Type	MemAcc_HwIdType
	Comment	Referenced memory driver hardware identifier
	MemInstanceId	
	Type	uint32
	Comment	Instance ID of the current memory request
	MemAddress	
	Type	uint32
	Comment	Physical address of the current memory driver request
	MemLength	
	Type	uint32
	Comment	Length of memory driver request
	CurrentJob	
	Type	MemAcc_JobType
	Comment	Currently active MemAcc job
	MemResult	
	Type	MemAcc_MemJobResultType
	Comment	Current or last Mem driver result
Description	This structure contains information the current processing state of the MemAcc module.	
Available via	MemAcc_GeneralTypes.h	

⌋

8.2.11 MemAcc_HwIdType

[SWS_MemAcc_10010] Definition of datatype MemAcc_HwIdType

Upstream requirements: SRS_MemHwAb_14042

⌈

Name	MemAcc_HwIdType		
Kind	Type		
Derived from	uint32		
Range	0 - 4294967295	–	The name of each enum parameter is constructed from the Mem module name and the Mem instance name
Description	Type for the unique numeric identifiers of all Mem hardware instances used for hardware specific requests.		
Available via	MemAcc_GeneralTypes.h		

⌋

8.2.12 MemAcc_MemBinaryHeaderType

[SWS_MemAcc_10014] Definition of datatype MemAcc_MemBinaryHeaderType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemBinaryHeaderType	
Kind	Structure	
Elements	UniqueId	
	Type	uint64
	Comment	Unique ID
	Flags	
	Type	uint64
	Comment	Header flags
	Header	
	Type	uint64
	Comment	Address of Mem driver header structure
	Delimiter	
	Type	uint64
	Comment	Address of Mem driver delimiter field
	InitFunc	
	Type	MemAcc_MemInitFuncType
	Comment	Mem_Init function pointer
	DeinitFunc	
	Type	MemAcc_MemDeInitFuncType
	Comment	Mem_Deinit function pointer
	MainFunc	

▽

△

	Type	MemAcc_MemMainFuncType
	Comment	Mem_Main function pointer
	GetJobResultFunc	
	Type	MemAcc_MemGetJobResultFuncType
	Comment	Mem_GetJobResult function pointer
	ReadFunc	
	Type	MemAcc_MemReadFuncType
	Comment	Mem_Read function pointer
	WriteFunc	
	Type	MemAcc_MemWriteFuncType
	Comment	Mem_Write function pointer
	EraseFunc	
	Type	MemAcc_MemEraseFuncType
	Comment	Mem_Erase function pointer
	PropagateErrorFunc	
	Type	MemAcc_MemPropagateErrorFuncType
	Comment	Mem_PropagateError function pointer
	BlankCheckFunc	
	Type	MemAcc_MemBlankCheckFuncType
	Comment	Mem_BlankCheck function pointer
	SuspendFunc	
	Type	MemAcc_MemSuspendFuncType
	Comment	Mem_Suspend function pointer
	ResumeFunc	
	Type	MemAcc_MemResumeFuncType
	Comment	Mem_Resume function pointer
	HwSpecificServiceFunc	
	Type	MemAcc_MemHwSpecificServiceFuncType
	Comment	Hardware specific service function pointer
Description	This structure contains elements for accessing the Mem driver service functions and consistency information.	
Available via	MemAcc_MemApi.h	

⌋

8.2.13 MemAcc_MemAddressType

[SWS_MemAcc_91013] Definition of datatype MemAcc_MemAddressType

Upstream requirements: SRS_MemHwAb_14046

⌈

Name	MemAcc_MemAddressType
Kind	Type
Derived from	MemAcc_AddressType
Description	Physical memory device address type
Available via	MemAcc_MemApi.h

⌋

8.2.14 MemAcc_MemConfigType

[SWS_MemAcc_91012] Definition of datatype MemAcc_MemConfigType

Upstream requirements: SRS_BSW_00414

⌈

Name	MemAcc_MemConfigType
Kind	Structure
Description	Memory driver configuration structure type
Available via	MemAcc_MemApi.h

⌋

8.2.15 MemAcc_MemDataType

[SWS_MemAcc_91020] Definition of datatype MemAcc_MemDataType

Upstream requirements: SRS_MemHwAb_14051

⌈

Name	MemAcc_MemDataType
Kind	Type
Derived from	uint8
Description	General data type
Available via	MemAcc_MemApi.h

⌋

8.2.16 MemAcc_MemInstanceIdType

[SWS_MemAcc_91011] Definition of datatype MemAcc_MemInstanceIdType

Upstream requirements: SRS_MemHwAb_14043

⌈

Name	MemAcc_MemInstanceIdType
Kind	Type
Derived from	uint32
Description	Memory driver instance ID type
Available via	MemAcc_MemApi.h

⌋

8.2.17 MemAcc_MemJobResultType

[SWS_MemAcc_91016] Definition of datatype MemAcc_MemJobResultType

Upstream requirements: SRS_MemHwAb_14040

⌈

Name	MemAcc_MemJobResultType		
Kind	Enumeration		
Range	MEM_JOB_OK	0x00	The last job has been finished successfully
	MEM_JOB_PENDING	0x01	A job is currently being processed
	MEM_JOB_FAILED	0x02	Job failed for some unspecific reason
	MEM_INCONSISTENT	0x03	The checked page is not blank
	MEM_ECC_ UNCORRECTED	0x04	Uncorrectable ECC errors occurred during memory access
	MEM_ECC_CORRECTED	0x05	Correctable ECC errors occurred during memory access
Description	Asynchronous job result type		
Available via	MemAcc_GeneralTypes.h		

⌋

8.2.18 MemAcc_MemLengthType

[SWS_MemAcc_91014] Definition of datatype MemAcc_MemLengthType

Upstream requirements: SRS_MemHwAb_14046

⌈

Name	MemAcc_MemLengthType
Kind	Type
Derived from	uint32
Description	Physical memory device length type
Available via	MemAcc_MemApi.h

⌋

8.2.19 MemAcc_MemHwServiceIdType

[SWS_MemAcc_10008] Definition of datatype MemAcc_MemHwServiceIdType

Upstream requirements: SRS_MemHwAb_14056

⌈

Name	MemAcc_MemHwServiceIdType
Kind	Type
Derived from	uint32
Description	Index type for Mem driver hardware specific service table.
Available via	MemAcc_MemApi.h

⌋

8.2.20 MemAcc_MemInitFuncType

[SWS_MemAcc_91000] Definition of datatype MemAcc_MemInitFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemInitFuncType	
Kind	Function Pointer	

Syntax	`void (*MemAcc_MemInitFuncType) (`
	`  MemAcc_MemConfigType* configPtr`
	`)`	

Parameters (in)	configPtr	Pointer to the Mem driver configuration data structure.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Function pointer for the Mem_Init service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.21 MemAcc_MemDeInitFuncType

[SWS_MemAcc_91018] Definition of datatype MemAcc_MemDeInitFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemDeInitFuncType
Kind	Function Pointer

Syntax	`void (*MemAcc_MemDeInitFuncType) (`
	`  void`
	`)`

Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Function pointer for the Mem_DeInit service for the invocation of the Mem driver API via function pointer interface.
Available via	MemAcc_MemApi.h

⌋

8.2.22 MemAcc_MemGetJobResultFuncType

[SWS_MemAcc_91002] Definition of datatype MemAcc_MemGetJobResultFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemGetJobResultFuncType	
Kind	Function Pointer	

Syntax	`MemAcc_MemJobResultType (*MemAcc_MemGetJobResultFuncType) (`
	`  MemAcc_MemInstanceIdType instanceId`
	`)`	

Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemAcc_MemJobResultType	Most recent job result.
Description	Function pointer for the Mem_JobResultType service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.23 MemAcc_MemSuspendFuncType

[SWS_MemAcc_91008] Definition of datatype MemAcc_MemSuspendFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemSuspendFuncType	
Kind	Function Pointer	
Syntax	`Std_ReturnType (*MemAcc_MemSuspendFuncType) ( MemAcc_MemInstanceIdType instanceId )`	
Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`:The requested job has not been accepted by the module. `E_MEM_SERVICE_NOT_AVAIL`: The underlying Mem driver service function is not available.
Description	Function pointer for the Mem_Suspend service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.24 MemAcc_MemResumeFuncType

[SWS_MemAcc_91009] Definition of datatype MemAcc_MemResumeFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemResumeFuncType	
Kind	Function Pointer	
Syntax	`Std_ReturnType (*MemAcc_MemResumeFuncType) ( MemAcc_MemInstanceIdType instanceId )`	
Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`:The requested job has not been accepted by the module. `E_MEM_SERVICE_NOT_AVAIL`: The underlying Mem driver service function is not available.
Description	Function pointer for the Mem_Resume service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.25 MemAcc_MemPropagateErrorFuncType

[SWS_MemAcc_91006] Definition of datatype MemAcc_MemPropagateErrorFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemPropagateErrorFuncType	
Kind	Function Pointer	

Syntax	`void (*MemAcc_MemPropagateErrorFuncType) (`
	`  MemAcc_MemInstanceIdType instanceId`
	`)`	

Parameters (in)	instanceId	ID of the related memory driver instance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Function pointer for the Mem_PropagateError service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.26 MemAcc_MemReadFuncType

[SWS_MemAcc_91003] Definition of datatype MemAcc_MemReadFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemReadFuncType	
Kind	Function Pointer	

Syntax	`Std_ReturnType (*MemAcc_MemReadFuncType) (`
	`  MemAcc_MemInstanceIdType instanceId,`
	`  MemAcc_MemAddressType sourceAddress,`
	`  MemAcc_MemDataType* destinationDataPtr,`
	`  MemAcc_MemLengthType length`
	`)`	

Parameters (in)	instanceId	ID of the related memory driver instance.
	sourceAddress	Physical address to read data from.
	length	Read length in bytes.
Parameters (inout)	None	
Parameters (out)	destinationDataPtr	Destination memory pointer to store the read data.
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has not been accepted by the module.
Description	Function pointer for the Mem_Read service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.27 MemAcc_MemWriteFuncType

[SWS_MemAcc_91004] Definition of datatype MemAcc_MemWriteFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemWriteFuncType	
Kind	Function Pointer	

Syntax	`Std_ReturnType (*MemAcc_MemWriteFuncType) (`
	`  MemAcc_MemInstanceIdType instanceId,`
	`  MemAcc_MemAddressType targetAddress,`
	`  const MemAcc_MemDataType* sourceDataPtr,`
	`  MemAcc_MemLengthType length`
	`)`	

Parameters (in)	instanceId	ID of the related memory driver instance.
	targetAddress	Physical write address (aligned to page size).
	sourceDataPtr	Source data pointer (aligned to page size).
	length	Write length in bytes (aligned to page size).
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has not been accepted by the module.
Description	Function pointer for the Mem_Write service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.28 MemAcc_MemEraseFuncType

[SWS_MemAcc_91005] Definition of datatype MemAcc_MemEraseFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemEraseFuncType	
Kind	Function Pointer	

Syntax	`Std_ReturnType (*MemAcc_MemEraseFuncType) (`
	`  MemAcc_MemInstanceIdType instanceId,`
	`  MemAcc_MemAddressType targetAddress,`
	`  MemAcc_MemLengthType length`
	`)`	

Parameters (in)	instanceId	ID of the related memory driver instance.
	targetAddress	Physical erase address (aligned to sector size).
	length	Erase length in bytes (aligned to sector size).
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description	Function pointer for the Mem_Erase service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.29 MemAcc_MemBlankCheckFuncType

[SWS_MemAcc_91007] Definition of datatype MemAcc_MemBlankCheckFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemBlankCheckFuncType	
Kind	Function Pointer	

Syntax	`Std_ReturnType (*MemAcc_MemBlankCheckFuncType) (`
	`  MemAcc_MemInstanceIdType instanceId,`
	`  MemAcc_MemAddressType targetAddress,`
	`  MemAcc_MemLengthType length`
	`)`	

Parameters (in)	instanceId	ID of the related memory driver instance.
	targetAddress	Physical blank check address.
	length	Blank check length.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description	Function pointer for the Mem_BlankCheck service for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.30 MemAcc_MemHwSpecificServiceFuncType

[SWS_MemAcc_91010] Definition of datatype MemAcc_MemHwSpecificServiceFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemHwSpecificServiceFuncType	
Kind	Function Pointer	

Syntax	`Std_ReturnType (*MemAcc_MemHwSpecificServiceFuncType) (`
	`  MemAcc_MemInstanceIdType instanceId,`
	`  MemAcc_MemHwServiceIdType hwServiceId,`
	`  MemAcc_MemDataType* dataPtr,`
	`  MemAcc_MemLengthType* lengthPtr`
	`)`	

Parameters (in)	instanceId	ID of the related memory driver instance.
	hwServiceId	Hardware specific service request identifier for dispatching the request.
	lengthPtr	Size pointer of the passed data.
Parameters (inout)	dataPtr	Request specific data pointer.
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has not been accepted by the module.
Description	Function pointer for the Mem_HwSpecificService function for the invocation of the Mem driver API via function pointer interface.	
Available via	MemAcc_MemApi.h	

⌋

8.2.31 MemAcc_MemMainFuncType

[SWS_MemAcc_91001] Definition of datatype MemAcc_MemMainFuncType

Upstream requirements: SRS_MemHwAb_14045

⌈

Name	MemAcc_MemMainFuncType
Kind	Function Pointer

Syntax	`void (*MemAcc_MemMainFuncType) (`
	`  void`
	`)`

Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Function pointer for the Mem_MainFunction service for the invocation of the Mem driver API via function pointer interface.
Available via	MemAcc_MemApi.h

⌋

8.2.32 MemAcc_ApplicationLockNotificationType

[SWS_MemAcc_91021] Definition of datatype MemAcc_ApplicationLockNotificationType ⌈

Name	MemAcc_ApplicationLockNotificationType
Kind	Function Pointer

Syntax	`void (*MemAcc_ApplicationLockNotificationType) (`
	`  void`
	`)`

Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Type of the function pointer to register an application callback to be notified once the address area lock was successfully acquired.
Available via	MemAcc.h

⌋

8.2.33 Extension of Std_ReturnType

[SWS_MemAcc_91022] Definition of Std_ReturnType-extension for module Mem Acc ⌈

Range	E_MEM_SERVICE_NOT_AVAIL	0x03	The underlying Mem driver service function is not available.
Description	Overlayed return value of Std_ReturnType for Mem driver API		
Available via	MemAcc_MemApi.h		

⌋

8.3 Function Definitions

This chapter contains all APIs provided by the module.

MemAcc does not provide any call-back notification functions.

8.3.1 Synchronous Functions

8.3.1.1 MemAcc_Init

[SWS_MemAcc_10015] Definition of API function MemAcc_Init

Upstream requirements: SRS_MemHwAb_14035, SRS_MemHwAb_14037

⌈

Service Name	MemAcc_Init	
Syntax	`void MemAcc_Init ( const MemAcc_ConfigType* configPtr )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	configPtr	Pointer to selected configuration structure.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initialization function - initializes all variables and sets the module state to initialized.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00025]

Upstream requirements: SRS_MemHwAb_14037, SRS_MemHwAb_14047

⌈The service `MemAcc_Init` shall initialize the MemAcc module internal states. If `MemInvocation` (see [ECUC_Mem_00025]) is set to `INDIRECT_DYNAMIC` or `INDIRECT_STATIC`, `MemAcc_Init` shall also initialize all available Mem drivers by calling the Mem driver's individual initialization functions.⌋

8.3.1.2 MemAcc_DeInit

[SWS_MemAcc_10041] Definition of API function MemAcc_DeInit

Upstream requirements: SRS_MemHwAb_14037

⌈

Service Name	MemAcc_DeInit
Syntax	`void MemAcc_DeInit ( void )`
Service ID [hex]	0x13
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None

▽

△

Parameters (out)	None
Return value	None
Description	Deinitialize module. If there are still access jobs pending, they are immediately terminated and the module state is set to unitialized. Therefore, MemAcc must be re-initialized before it will accept any new job requests after this service is processed.
Available via	MemAcc.h

⌋

[SWS_MemAcc_00111]

Upstream requirements: SRS_MemHwAb_14047

⌈The service `MemAcc_DeInit` shall de-initialize the MemAcc module internal states. If `MemInvocation` (see [ECUC_Mem_00025]) is set to `INDIRECT_DYNAMIC` or `INDIRECT_STATIC`, `MemAcc_DeInit` shall also de-initialize all available Mem drivers by calling the Mem driver's individual de-initialization functions.⌋

8.3.1.3 MemAcc_GetJobResult

[SWS_MemAcc_10019] Definition of API function MemAcc_GetJobResult

Upstream requirements: SRS_MemHwAb_14040

⌈

Service Name	MemAcc_GetJobResult	
Syntax	`MemAcc_JobResultType MemAcc_GetJobResult ( MemAcc_AddressAreaIdType addressAreaId )`	
Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemAcc_JobResultType	Most recent job result of the referenced address area.
Description	Returns the consolidated job result of the address area referenced by addressAreaId.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00092]

Upstream requirements: SRS_MemHwAb_14040

⌈The service `MemAcc_GetJobResult` shall return the consolidated result of the last MemAcc job.⌋

Note: If a MemAcc job is still pending, the API returns the result of the last MemAcc job.

[SWS_MemAcc_00033]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_GetJobResult` shall check that the MemAcc module has been initialized. If this check fails, `MemAcc_GetJobResult` shall raise the development error `MEMACC_E_UNINIT`.⌋

[SWS_MemAcc_00034]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_GetJobResult` shall check that the provided `addressAreaId` is consistent with the configuration. If this check fails, `MemAcc_GetJobResult` shall raise the development error `MEMACC_E_PARAM_ADDRESS_AREA_ID`.⌋

8.3.1.4 MemAcc_GetJobStatus

[SWS_MemAcc_10040] Definition of API function MemAcc_GetJobStatus

Upstream requirements: SRS_MemHwAb_14040

⌈

Service Name	MemAcc_GetJobStatus	
Syntax	`MemAcc_JobStatusType MemAcc_GetJobStatus ( MemAcc_AddressAreaIdType addressAreaId )`	
Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemAcc_JobStatusType	Most recent job result of the referenced address area.
Description	Returns the status of the MemAcc job referenced by addressAreaId.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00118]

Upstream requirements: SRS_MemHwAb_14040

⌈The service `MemAcc_GetJobStatus` shall return `MEMACC_JOB_IDLE` for the referenced `addressAreaId` if MemAcc is not processing a job request.⌋

[SWS_MemAcc_00119]

Upstream requirements: SRS_MemHwAb_14040

⌈The service `MemAcc_GetJobStatus` shall return `MEMACC_JOB_PENDING` for the referenced `addressAreaId` if MemAcc is currently processing a job request.⌋

[SWS_MemAcc_00117]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJobStatus shall check that the MemAcc module has been initialized. If this check fails, MemAcc_GetJobStatus shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00124]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJobStatus shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_GetJobStatus shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

8.3.1.5 MemAcc_GetMemoryInfo

[SWS_MemAcc_10020] Definition of API function MemAcc_GetMemoryInfo

Upstream requirements: SRS_MemHwAb_14054

⌈

Service Name	MemAcc_GetMemoryInfo	
Syntax	`Std_ReturnType MemAcc_GetMemoryInfo ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_AddressType address, MemAcc_MemoryInfoType* memoryInfoPtr )`	
Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	address	Address in logical address space from which corresponding memory device information shall be retrieved.
Parameters (inout)	None	
Parameters (out)	memoryInfoPtr	Destination memory pointer to store the memory device information.
Return value	Std_ReturnType	E_OK: The requested addressAreaId and address are valid. E_NOT_OK: The requested addressAreaId and address are invalid.
Description	This service function retrieves the physical memory device information of a specific address area. It can be used by an upper layer to get all necessary information to align the start address and trim the length for erase/write jobs.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00035]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetMemoryInfo shall check that the MemAcc module has been initial-

ized. If this check fails, MemAcc_GetMemoryInfo shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00036]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetMemoryInfo shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_GetMemoryInfo shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

[SWS_MemAcc_00037]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetMemoryInfo shall raise the development error MEMACC_E_PARAM_POINTER if the memoryInfoPtr argument is a NULL pointer.⌋

8.3.1.6 MemAcc_GetProcessedLength

[SWS_MemAcc_10021] Definition of API function MemAcc_GetProcessedLength

Upstream requirements: SRS_MemHwAb_14040

⌈

Service Name	MemAcc_GetProcessedLength	
Syntax	`MemAcc_LengthType MemAcc_GetProcessedLength ( MemAcc_AddressAreaIdType addressAreaId )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
Parameters (inout)	None	
Parameters (out)	None	
Return value	MemAcc_LengthType	Processed length of current job (in bytes).
Description	Returns the accumulated number of bytes that have already been processed in the current job.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00120]

Upstream requirements: SRS_MemHwAb_14040

⌈The service MemAcc_GetProcessedLength shall return the processed length of the current MemAcc job referenced by addressAreaId. The processed length information shall only be updated if the underlying Mem job was successfully completed.⌋

[SWS_MemAcc_00038]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetProcessedLength shall check that the MemAcc module has been initialized. If this check fails, MemAcc_GetProcessedLength shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00039]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetProcessedLength shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_GetProcessedLength shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

8.3.1.7 MemAcc_GetJobInfo

[SWS_MemAcc_10022] Definition of API function MemAcc_GetJobInfo

Upstream requirements: SRS_MemHwAb_14040

⌈

Service Name	MemAcc_GetJobInfo	
Syntax	`void MemAcc_GetJobInfo ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_JobInfoType* jobInfoPtr )`	
Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
Parameters (inout)	None	
Parameters (out)	jobInfoPtr	Structure pointer to return the detailed processing information of the current job.
Return value	None	
Description	Returns detailed information about the current memory job like memory device ID, job type, job, processing state or job result, address area as well as address and length. In case no job is pending on the referenced address area, the function returns the information of the last job.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00040]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJobInfo shall check that the MemAcc module has been initialized. If this check fails, MemAcc_GetJobInfo shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00041]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJobInfo shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_GetJobInfo shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

[SWS_MemAcc_00042]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJobInfo shall raise the development error MEMACC_E_PARAM_POINTER if the jobInfoPtr argument is a NULL pointer.⌋

8.3.1.8 MemAcc_ActivateMem

[SWS_MemAcc_10033] Definition of API function MemAcc_ActivateMem

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14047

⌈

Service Name	MemAcc_ActivateMem	
Syntax	`Std_ReturnType MemAcc_ActivateMem ( MemAcc_AddressType headerAddress, MemAcc_HwIdType hwId )`	
Service ID [hex]	0x14	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	headerAddress	Physical start address of Mem driver header structure.
	hwId	Unique numeric memory driver identifier.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Mem driver activation successful. E_NOT_OK: Mem driver activation failed.
Description	Dynamic activation and initialization of a Mem driver referenced by hwId and headerAddress.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00121]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14047

⌈If MemInvocation (see [ECUC_Mem_00025]) is set to INDIRECT_DYNAMIC, the service MemAcc_ActivateMem shall initialize the Mem driver referenced by hwId and headerAddress and update the internal driver activation state.⌋

[SWS_MemAcc_00088]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_ActivateMem` shall check that the MemAcc module has been initialized. If this check fails, `MemAcc_ActivateMem` shall raise the development error `MEMACC_E_UNINIT`.⌋

[SWS_MemAcc_00089]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14049

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_ActivateMem` shall ensure the validity of the Mem driver binary by checks in the following sequence:

1. Comparing the header address field with the start address of the Mem driver binary (if the Mem driver was not compiled as a relocatable binary)
2. Unique ID validity
3. Availability and consistency of the delimiter field

If any of these checks fails `MemAcc_ActivateMem` shall raise the development error `MEMACC_E_MEM_INIT_FAILED`.⌋

8.3.1.9 MemAcc_DeactivateMem

[SWS_MemAcc_10034] Definition of API function MemAcc_DeactivateMem

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14047

⌈

Service Name	MemAcc_DeactivateMem	
Syntax	`Std_ReturnType MemAcc_DeactivateMem ( MemAcc_AddressType headerAddress, MemAcc_HwIdType hwId )`	
Service ID [hex]	0x15	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	headerAddress	Physical start address of Mem driver header structure.
	hwId	Unique numeric memory driver identifier.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Mem driver deactivation successful. `E_NOT_OK`: Mem driver deactivation failed.
Description	Dynamic deactivation of a Mem driver referenced by hwId and headerAddress.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00122]

Upstream requirements: SRS_MemHwAb_14045, SRS_MemHwAb_14047

⌈If `MemInvocation` (see [ECUC_Mem_00025]) is set to `INDIRECT_DYNAMIC`, the service `MemAcc_DeactivateMem` shall de-initialize the Mem driver referenced by `hwId` and `headerAddress` and update the internal driver activation state.⌋

[SWS_MemAcc_00090]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_DeactivateMem` shall check that the MemAcc module has been initialized. If this check fails, `MemAcc_DeactivateMem` shall raise the development error `MEMACC_E_UNINIT`.⌋

[SWS_MemAcc_00123]

Upstream requirements: SRS_MemHwAb_14031

⌈In case a MemAcc job is still pending, the service `MemAcc_DeactivateMem` shall return `E_NOT_OK` without any further action.⌋

Note: After calling the MemAcc_DeactivateMem service, the integration code shall also clear the memory area where the corresponding Mem driver is stored to prevent accidental execution of a Mem driver.

8.3.1.10 MemAcc_ReleaseLock

[SWS_MemAcc_10031] Definition of API function MemAcc_ReleaseLock

Upstream requirements: SRS_MemHwAb_14055

⌈

Service Name	MemAcc_ReleaseLock	
Syntax	`Std_ReturnType MemAcc_ReleaseLock ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_AddressType address, MemAcc_LengthType length )`	
Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	address	Logical start address to identify lock area.
	length	Length to identify lock area.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has been rejected by the module.
Description	Release access lock of provided address area.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00076]

Upstream requirements: SRS_BSW_00406, SRS_MemHwAb_14038

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_ReleaseLock shall check that the MemAcc module has been initialized. If this check fails, MemAcc_ReleaseLock shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00093]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_ReleaseLock shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_ReleaseLock shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

[SWS_MemAcc_00077]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_ReleaseLock shall raise the development error MEMACC_E_PARAM_ADDRESS_LENGTH if the address range defined by address and length is invalid, i.e. not mapped to a specific Mem driver.⌋

8.3.2 Asynchronous Functions

8.3.2.1 MemAcc_Cancel

[SWS_MemAcc_10018] Definition of API function MemAcc_Cancel

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	MemAcc_Cancel

Syntax	`void MemAcc_Cancel (`
	`  MemAcc_AddressAreaIdType addressAreaId`
	`)`

Service ID [hex]	0x04
Sync/Async	Asynchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	addressAreaId	Numeric identifier of address area.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Triggers a cancel operation of the pending job for the address area referenced by the address AreaId. Cancelling affects only jobs in pending state. For any other states, the request will be ignored.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00028]

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14044

⌈When the `MemAcc_Cancel` service is called by an upper layer, the MemAcc module shall wait for the completion of a pending Mem job and stop further processing of the current MemAcc job.⌋

Note: Not all memory devices support a cancel operation in hardware. To keep the behavior consistent, the cancel operation is only applied on the physical segmentation.

[SWS_MemAcc_00029]

Upstream requirements: SRS_MemHwAb_14031

⌈In case no MemAcc job is pending, the `MemAcc_Cancel` service shall just return without any further action, i.e., the result of the last MemAcc job shall not be affected.⌋

[SWS_MemAcc_00030]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Cancel` shall check that the MemAcc module has been initialized. If this check fails, `MemAcc_Cancel` shall raise the development error `MEMACC_E_UNINIT`.⌋

[SWS_MemAcc_00031]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Cancel` shall check that the provided `addressAreaId` is consistent with the configuration. If this check fails, `MemAcc_Cancel` shall raise the development error `MEMACC_E_PARAM_ADDRESS_AREA_ID`.⌋

8.3.2.2 MemAcc_Read

[SWS_MemAcc_10023] Definition of API function MemAcc_Read

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	MemAcc_Read	
Syntax	`Std_ReturnType MemAcc_Read ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_AddressType sourceAddress, MemAcc_DataType* destinationDataPtr, MemAcc_LengthType length )`	
Service ID [hex]	0x09	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	sourceAddress	Read address in logical address space.
	length	Read length in bytes (aligned to MemMinReadSize)
Parameters (inout)	None	
Parameters (out)	destinationDataPtr	Destination memory pointer to store the read data.
Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description	Triggers a read job to copy data from the source address into the referenced destination data buffer. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the read operation was successful, the result of the job is MEMACC_OK. If the read operation failed, the result of the job is either MEMACC_FAILED in case of a general error or MEMACC_ECC_CORRECTED/MEMACC_ECC_UNCORRECTED in case of a correctable/uncorrectable ECC error.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00043]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Read shall check that the MemAcc module has been initialized. If this check fails, MemAcc_Read shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00044]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Read shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_Read shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

[SWS_MemAcc_00045]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Read` shall raise the development error `MEMACC_E_PARAM_POINTER` if the `destinationDataPtr` argument is a NULL pointer.⌋

[SWS_MemAcc_00046]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Read` shall raise the development error `MEMACC_E_PARAM_ADDRESS_LENGTH` if the address range defined by `sourceAddress` and `length` is invalid, i.e. not aligned to `MemMinReadSize` or exceeds the configured area.⌋

[SWS_MemAcc_00047]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Read` shall raise the development error `MEMACC_E_BUSY` if a previous MemAcc job for the same `addressAreaId` is still being processed.⌋

8.3.2.3 MemAcc_Write

[SWS_MemAcc_10024] Definition of API function MemAcc_Write

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	MemAcc_Write	
Syntax	`Std_ReturnType MemAcc_Write ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_AddressType targetAddress, const MemAcc_DataType* sourceDataPtr, MemAcc_LengthType length )`	
Service ID [hex]	0x0a	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	targetAddress	Write address in logical address space.
	sourceDataPtr	Source data pointer (aligned to MemAccBufferAlignmentValue).
	length	Write length in bytes (aligned to page size).
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description	Triggers a write job to store the passed data to the provided address area with given address and length. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the write operation was successful, the job result is MEMACC_OK. If there was an issue writing the data, the result is MEMACC_FAILED.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00048]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Write shall check that the MemAcc module has been initialized. If this check fails, MemAcc_Write shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00049]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Write shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_Write shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

[SWS_MemAcc_00050]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Write shall raise the development error MEMACC_E_PARAM_POINTER if the sourceDataPtr argument is a NULL pointer.⌋

[SWS_MemAcc_00051]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Write shall raise the development error MEMACC_E_PARAM_ADDRESS_LENGTH if the address range defined by targetAddress and length is invalid, i.e. not aligned to MemWritePageSize or exceeds the configured area.⌋

[SWS_MemAcc_00052]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Write shall raise the development error MEMACC_E_BUSY if a previous MemAcc job for the same addressAreaId is still being processed.⌋

8.3.2.4 MemAcc_Erase

[SWS_MemAcc_10025] Definition of API function MemAcc_Erase

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	MemAcc_Erase	
Syntax	`Std_ReturnType MemAcc_Erase ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_AddressType targetAddress, MemAcc_LengthType length )`	
Service ID [hex]	0x0b	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	targetAddress	Erase address in logical address space (aligned to sector size).
	length	Erase length in bytes (aligned to sector size).
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description	Triggers an erase job of the given area. Triggers an erase job of the given area defined by targetAddress and length. The result of this service can be retrieved using the Mem_GetJobResult API. If the erase operation was successful, the result of the job is MEM_JOB_OK. If the erase operation failed, e.g. due to a hardware issue, the result of the job is MEM_JOB_FAILED.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00053]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Erase` shall check that the MemAcc module has been initialized. If this check fails, `MemAcc_Erase` shall raise the development error `MEMACC_E_UNINIT`.⌋

[SWS_MemAcc_00054]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Erase` shall check that the provided `addressAreaId` is consistent with the configuration. If this check fails, `MemAcc_Erase` shall raise the development error `MEMACC_E_PARAM_ADDRESS_AREA_ID`.⌋

[SWS_MemAcc_00055]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_Erase` shall raise the development error

MEMACC_E_PARAM_ADDRESS_LENGTH if the address range defined by targetAddress and length is invalid, i.e. not aligned to MemEraseSectorSize or exceeds the configured area.⌋

[SWS_MemAcc_00056]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Erase shall raise the development error MEMACC_E_BUSY if a previous MemAcc job for the same addressAreaId is still being processed.⌋

8.3.2.5 MemAcc_Compare

[SWS_MemAcc_10026] Definition of API function MemAcc_Compare

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	MemAcc_Compare	
Syntax	`Std_ReturnType MemAcc_Compare ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_AddressType sourceAddress, const MemAcc_DataType* dataPtr, MemAcc_LengthType length )`	
Service ID [hex]	0x0c	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	sourceAddress	Compare address in logical address space.
	dataPtr	Pointer to user data which shall be compared to data in memory.
	length	Compare length in bytes.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description	Triggers a job to compare the passed data to the memory content of the provided address area. The job terminates, if all bytes matched or a difference was detected. The result of this service can be retrieved using the MemAcc_GetJobResult() API. If the compare operation determined a mismatch, the result code is MEMACC_INCONSISTENT.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00057]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Compare shall check that the MemAcc module has been initialized. If this check fails, MemAcc_Compare shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00058]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Compare shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_Compare shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

[SWS_MemAcc_00059]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Compare shall raise the development error MEMACC_E_PARAM_POINTER if the dataPtr argument is a NULL pointer.⌋

[SWS_MemAcc_00060]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Compare shall raise the development error MEMACC_E_PARAM_ADDRESS_LENGTH if the address range defined by sourceAddress and length is invalid, i.e. not aligned to MemMinReadSize.⌋

[SWS_MemAcc_00061]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_Compare shall raise the development error MEMACC_E_BUSY if a previous MemAcc job for the same addressAreaId is still being processed.⌋

[SWS_MemAcc_00114]

Upstream requirements: SRS_MemHwAb_14040

⌈If the compare operation determined a mismatch, the result code returned by the MemAcc_GetJobResult service shall be set to MEMACC_INCONSISTENT, otherwise MemAcc_GetJobResult shall return MEMACC_OK.⌋

8.3.2.6 MemAcc_BlankCheck

[SWS_MemAcc_10027] Definition of API function MemAcc_BlankCheck

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039

⌈

Service Name	MemAcc_BlankCheck	
Syntax	`Std_ReturnType MemAcc_BlankCheck ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_AddressType targetAddress, MemAcc_LengthType length )`	
Service ID [hex]	0x0d	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	targetAddress	Blank check address in logical address space.
	length	Blank check length in bytes.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has been rejected by the module.
Description	Checks if the passed address space is blank, i.e. erased and writeable. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the address area defined by targetAddress and length is blank, the result is MEMACC_OK, otherwise the result is MEMACC_INCONSISTENT.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00062]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_BlankCheck` shall check that the MemAcc module has been initialized. If this check fails, `MemAcc_BlankCheck` shall raise the development error `MEMACC_E_UNINIT`.⌋

[SWS_MemAcc_00063]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_BlankCheck` shall check that the provided `addressAreaId` is consistent with the configuration. If this check fails, `MemAcc_BlankCheck` shall raise the development error `MEMACC_E_PARAM_ADDRESS_AREA_ID`.⌋

[SWS_MemAcc_00064]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_BlankCheck` shall raise the development error

MEMACC_E_PARAM_ADDRESS_LENGTH if the address range defined by sourceAddress and length is invalid, i.e. not aligned to MemWritePageSize.⌋

[SWS_MemAcc_00065]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_BlankCheck shall raise the development error MEMACC_E_BUSY if a previous MemAcc job for the same addressAreaId is still being processed.⌋

8.3.2.7 MemAcc_HwSpecificService

[SWS_MemAcc_10028] Definition of API function MemAcc_HwSpecificService

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039, SRS_MemHwAb_14056

⌈

Service Name	MemAcc_HwSpecificService	
Syntax	`Std_ReturnType MemAcc_HwSpecificService ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_HwIdType hwId, MemAcc_MemHwServiceIdType hwServiceId, MemAcc_DataType* dataPtr, MemAcc_MemLengthType* lengthPtr )`	
Service ID [hex]	0x0e	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	hwId	Unique numeric memory driver identifier.
	hwServiceId	Hardware specific service request identifier for dispatching the request.
Parameters (inout)	dataPtr	Data pointer pointing to the job buffer. Value can be NULL_PTR, if not needed. If dataPtr is used by the hardware specific service, the pointer must be valid until the job completed.
	lengthPtr	Size pointer of the data passed by dataPtr. Can be NULL_PTR if dataPtr is also NULL_PTR.
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module.
Description	Triggers a hardware specific job request referenced by hwServiceId. Service specific data can be passed/retrieved by dataPtr. The result of this service can be retrieved using the MemAcc_GetJobResult API. If the hardware specific operation was successful, the result of the job is MEMACC_OK. If the hardware specific operation failed, the result of the job is MEMACC_FAILED.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00066]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_HwSpecificService shall check that the MemAcc module has been initialized. If this check fails, MemAcc_HwSpecificService shall raise the development error MEMACC_E_UNINIT.⌋

[SWS_MemAcc_00067]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_HwSpecificService shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_HwSpecificService shall raise the development error MEMACC_E_PARAM_ADDRESS_AREA_ID.⌋

[SWS_MemAcc_00068]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_HwSpecificService shall raise the development error MEMACC_E_PARAM_HW_ID if the Mem driver hardware identification given by hwId is invalid or not assigned to the passed addressAreaId.⌋

[SWS_MemAcc_00070]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_HwSpecificService shall raise the development error MEMACC_E_BUSY if a previous MemAcc job for the same addressAreaId is still being processed.⌋

8.3.2.8 MemAcc_RequestLock

[SWS_MemAcc_10030] Definition of API function MemAcc_RequestLock

Upstream requirements: SRS_MemHwAb_14038, SRS_MemHwAb_14039, SRS_MemHwAb_14055

⌈

Service Name	MemAcc_RequestLock

Syntax	`Std_ReturnType MemAcc_RequestLock (`
	`  MemAcc_AddressAreaIdType addressAreaId,`
	`  MemAcc_AddressType address,`
	`  MemAcc_LengthType length,`
	`  MemAcc_ApplicationLockNotificationType lockNotificationFctPtr`
	`)`

Service ID [hex]	0x11
Sync/Async	Asynchronous

▽

△

Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	address	Logical start address of the address area to identify the Mem driver to be locked.
	length	Length of the address area to identify the Mem driver to be locked.
	lockNotificationFctPtr	Pointer to the callback function to be notified once the address area lock was successfully acquired.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The requested job has been accepted by the module. `E_NOT_OK`: The requested job has been rejected by the module
Description	Request lock of an address area for exclusive access. Once the lock is granted, the referenced lock notification function is called by MemAcc.	
Available via	MemAcc.h	

⌋

[SWS_MemAcc_00115]

Upstream requirements: SRS_MemHwAb_14038

⌈`MemAcc_RequestLock` shall lock all memory accesses of the Mem driver referenced by the `addressAreaId`, `address` and `length` parameter.
If an upper layer calls a MemAcc service function for an address area which is locked for direct memory access, MemAcc shall still accept the memory access request for the address area but shall not forward the access request to the corresponding Mem driver until the lock request is released by the `MemAcc_ReleaseLock` service.⌋

[SWS_MemAcc_00116]

Upstream requirements: SRS_MemHwAb_14055

⌈MemAcc shall wait until the address area referenced by `addressAreaId` is idle before it calls the lock notification function referenced by `lockNotificationFctPtr` to notify the upper layer module that the lock of the address area was successfully acquired.⌋

[SWS_MemAcc_00099]

Upstream requirements: SRS_BSW_00406

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_RequestLock` shall check that the MemAcc module has been initialized. If this check fails, `MemAcc_RequestLock` shall raise the development error `MEMACC_E_UNINIT`.⌋

[SWS_MemAcc_00071]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by `MemAccDevErrorDetect`, the service `MemAcc_RequestLock` shall check that the provided `addressAreaId` is consistent with the configuration. If this check fails, `MemAcc_RequestLock` shall raise the development error `MEMACC_E_PARAM_ADDRESS_AREA_ID`.⌋

[SWS_MemAcc_00072]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_RequestLock shall raise the development error MEMACC_E_PARAM_POINTER if the lockNotificationFctPtr argument is a NULL pointer.⌋

[SWS_MemAcc_00073]

Upstream requirements: SRS_BSW_00323

⌈If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_RequestLock shall raise the development error MEMACC_E_PARAM_ADDRESS_LENGTH if the address range defined by address and length is invalid, i.e. not mapped to a specific Mem driver.⌋

8.4 Scheduled Functions

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

8.4.1 MemAcc_MainFunction

[SWS_MemAcc_10017] Definition of scheduled function MemAcc_MainFunction

Upstream requirements: SRS_MemHwAb_14047

⌈

Service Name	MemAcc_MainFunction
Syntax	`void MemAcc_MainFunction ( void )`
Service ID [hex]	0x03
Description	Service to handle the requested jobs and the internal management operations. Depending on the configuration MemAcc will call the Mem driver main functions.
Available via	SchM_MemAcc.h

⌋

[SWS_MemAcc_00084]

Upstream requirements: SRS_MemHwAb_14047

⌈If MemInvocation (see [ECUC_Mem_00025]) is set to INDIRECT_DYNAMIC or INDIRECT_STATIC, MemAcc shall call all Mem main functions within MemAcc_MainFunction.
MemAcc_MainFunction shall only call the Mem main function if there is a job request pending for the corresponding Mem driver.⌋

8.5 Expected Interfaces

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

8.5.1 Mandatory Interfaces

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

[SWS_MemAcc_10036] Definition of mandatory interfaces required by module MemAcc

Upstream requirements: SRS_BSW_00415

⌈

API Function	*Header File*	*Description*
Mem_BlankCheck	Mem.h	Triggers a job to check the erased state of the page which is referenced by targetAddress. The result of this service can be retrieved using the Mem_GetJob Result API. If the checked page is blank, the result of the job is MEM_JOB_OK. Otherwise, if the page is not blank, the result is MEM_INCONSISTENT.
Mem_DeInit	Mem.h	De-initialize module. If there is still an access job pending, it is immediately terminated (using hardware cancel operation) and the Mem driver module state is set to unitialized. Therefore, Mem must be re-initialized before it will accept any new job requests after this service is processed.
Mem_Erase	Mem.h	Triggers an erase job of the given sector/sector batch defined by targetAddress and length. The result of this service can be retrieved using the Mem_GetJobResult API. If the erase operation was successful, the result of the job is MEM_JOB_OK. If the erase operation failed, e.g. due to a hardware issue, the result of the job is MEM_JOB_FAILED.
Mem_GetJobResult	Mem.h	Service to return results of the most recent job.
Mem_HwSpecificService	Mem.h	Triggers a hardware specific memory driver job. dataPtr can be used to pass and return data to/from this service. This service is just a dispatcher to the hardware specific service implementation referenced by hwServiceId. The result of this service can be retrieved using the Mem_GetJob Result API. If the hardware specific operation was successful, the result of the job is MEM_JOB_OK. If the hardware specific operation failed, the result of the job is MEM_JOB_FAILED.
Mem_Init	Mem.h	Initialization function - initializes all variables and sets the module state to initialized.
Mem_MainFunction	SchM_Mem.h	Service to handle the requested jobs and the internal management operations.
Mem_PropagateError	Mem.h	This service can be used to report an access error in case the Mem driver cannot provide the access error information - typically for ECC faults. It is called by the system ECC handler to propagate an ECC error to the memory upper layers..

▽

△

API Function	Header File	Description
Mem_Read	Mem.h	Triggers a read job to copy the from the source address into the referenced destination data buffer. The result of this service can be retrieved using the Mem_GetJobResult API. If the read operation was successful, the result of the job is MEM_JOB_OK. If the read operation failed, the result of the job is either MEM_JOB_FAILED in case of a general error or MEM_ECC_CORRECTED/MEM_ECC_UNCORRECTED in case of a correctable/uncorrectable ECC error.
Mem_Resume	Mem.h	Resume suspended memory operation using hardware mechanism.
Mem_Suspend	Mem.h	Suspend active memory operation using hardware mechanism.
Mem_Write	Mem.h	Triggers a write job to store the passed data to the provided address area with given address and length. The result of this service can be retrieved using the Mem_GetJobResult API. If the write operation was successful, the job result is MEM_JOB_OK. If there was an issue writing the data, the result is MEM_FAILED.

⌋

8.5.2 Optional Interfaces

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

[SWS_MemAcc_10035] Definition of optional interfaces requested by module MemAcc

Upstream requirements: SRS_BSW_00415

⌈

API Function	Header File	Description
Det_ReportError	Det.h	Service to report development errors.

⌋

8.5.3 Configurable Interfaces

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

8.5.3.1 <AddressAreaJobEndNotification>

[SWS_MemAcc_10029] Definition of configurable interface <AddressAreaJobEndNotification>

Upstream requirements: SRS_MemHwAb_14041

⌈

Service Name	<AddressAreaJobEndNotification>	
Syntax	`void <AddressAreaJobEndNotification> ( MemAcc_AddressAreaIdType addressAreaId, MemAcc_JobResultType jobResult )`	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	addressAreaId	Numeric identifier of address area.
	jobResult	Result of the last MemAcc operation.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	MemAcc application job end notification callback. The function name is configurable.	
Available via	configurable	

⌋

8.6 Service Interfaces

The MemAcc module does not provide any service interfaces.

9 Sequence Diagrams

9.1 Job Handling with Result Polling

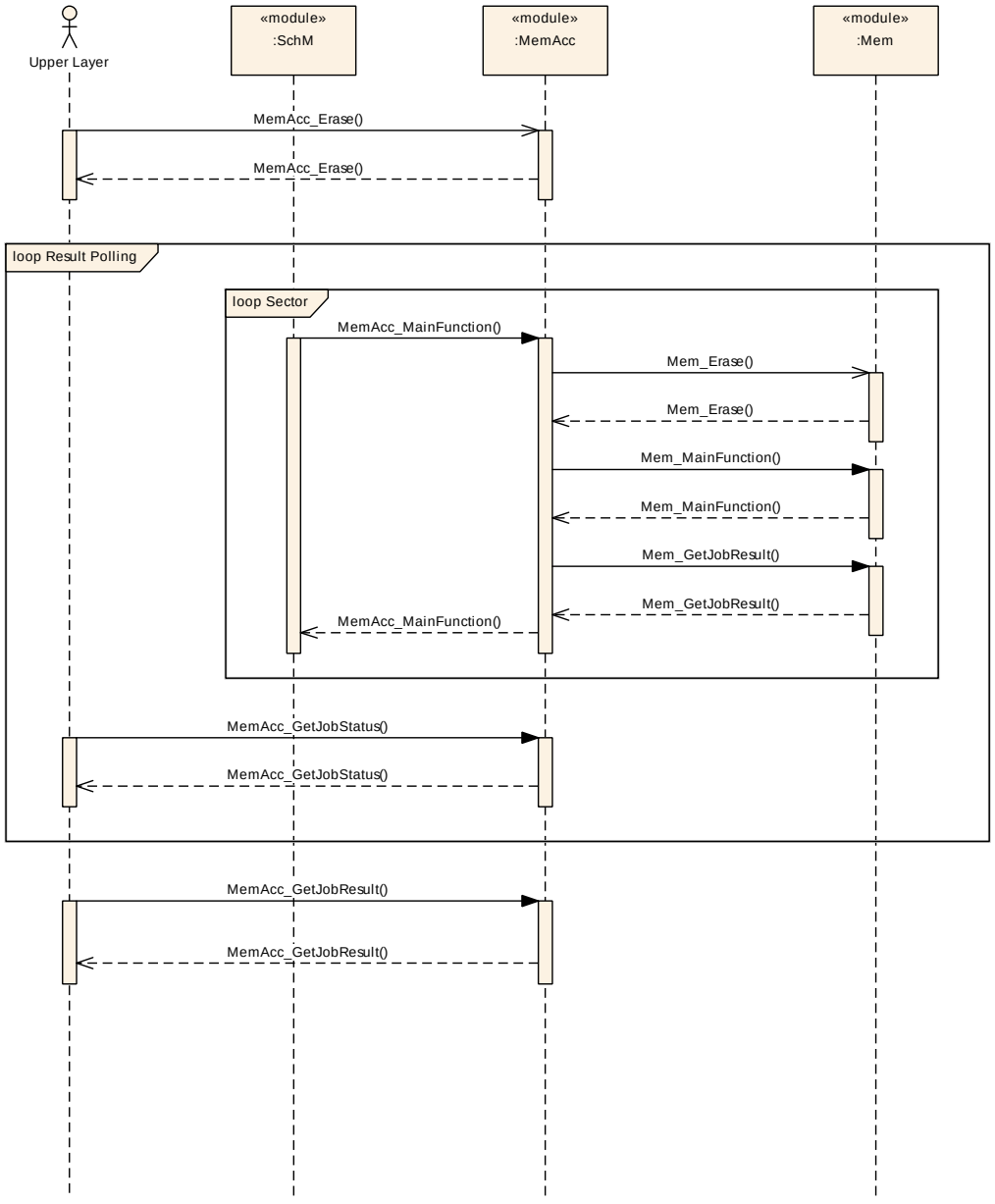


Figure 9.1: Job Handling with Result Polling

9.2 Job Handling with Job End Notification

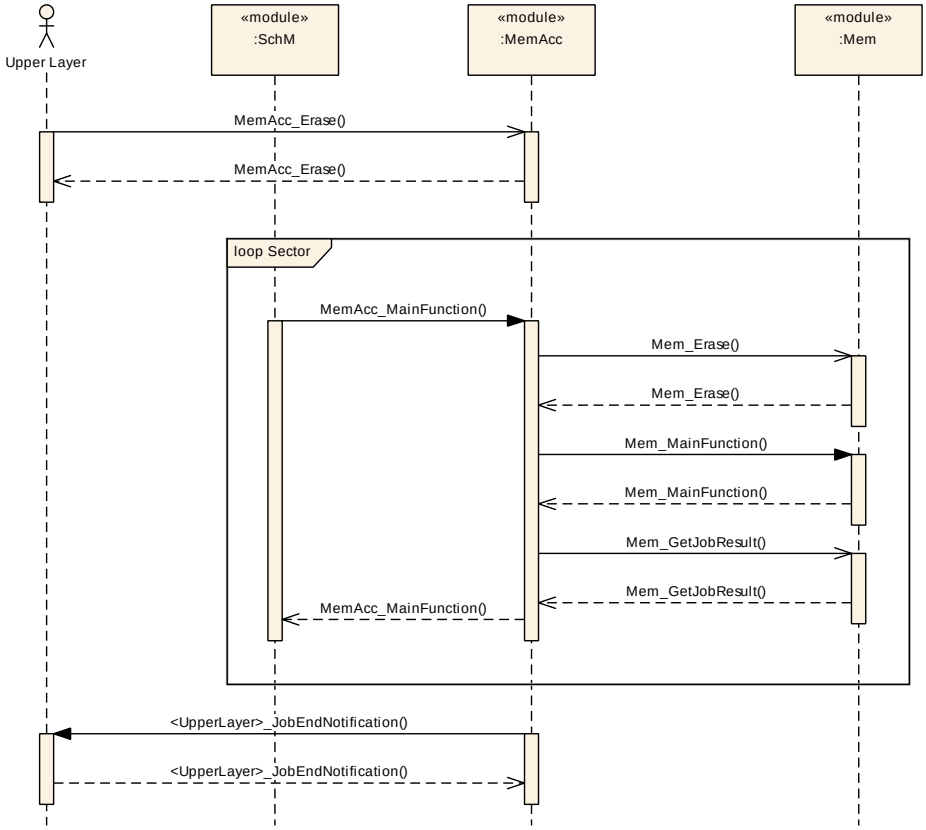


Figure 9.2: Job Handling with Job End Notification

9.3 Mem Driver Initialization by MemAcc

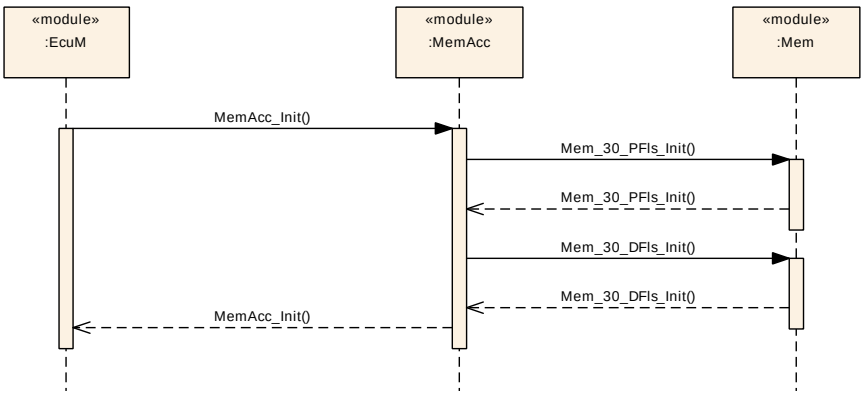


Figure 9.3: Mem Driver Initialization by MemAcc

9.4 Mem Driver Initialization by EcuM

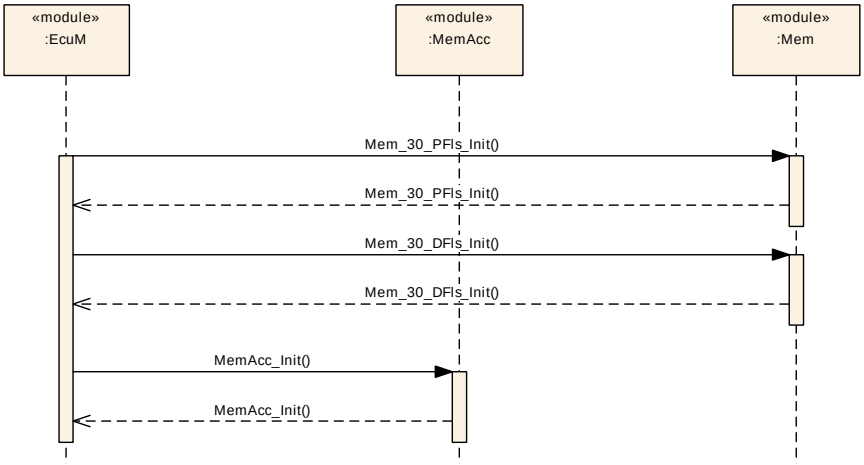


Figure 9.4: Mem Driver initialization by EcuM

9.5 Mem Driver Scheduling by MemAcc

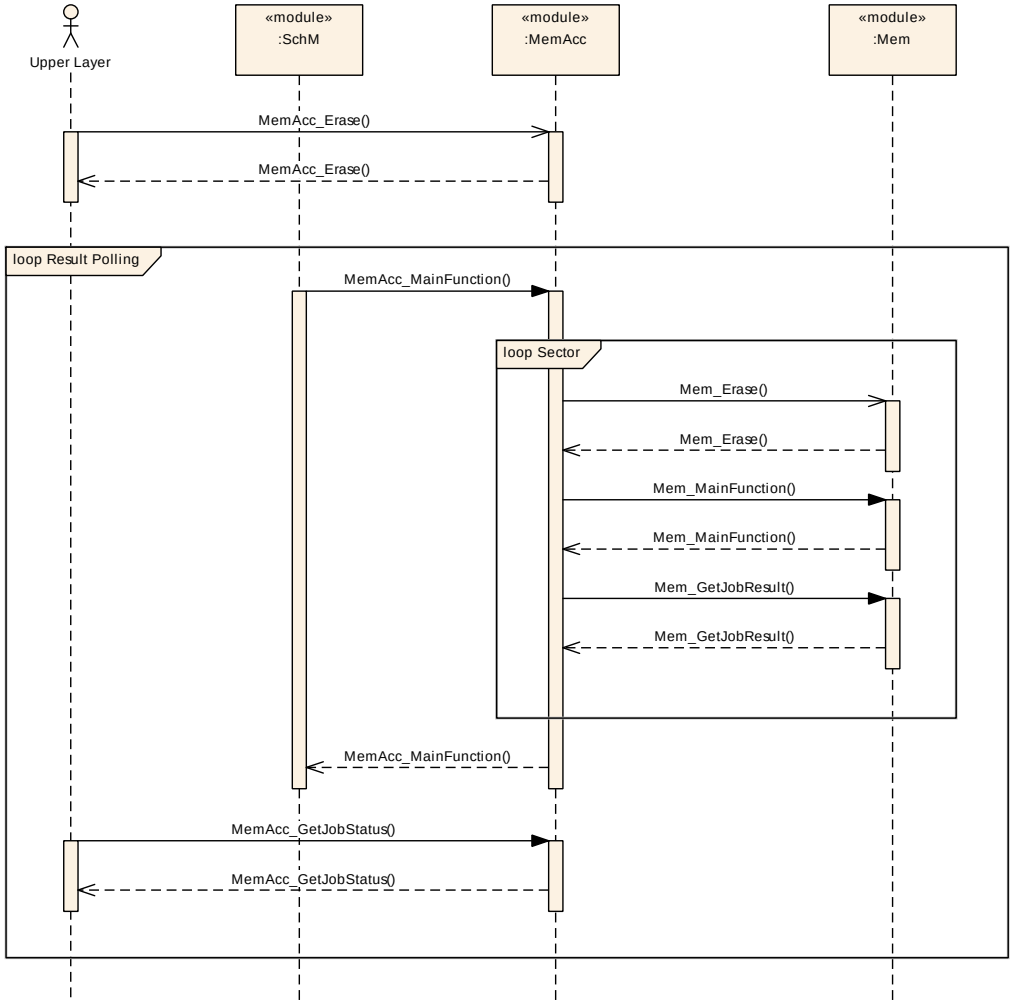


Figure 9.5: Mem Driver Scheduling by MemAcc

9.6 Mem Driver Scheduling by SchM

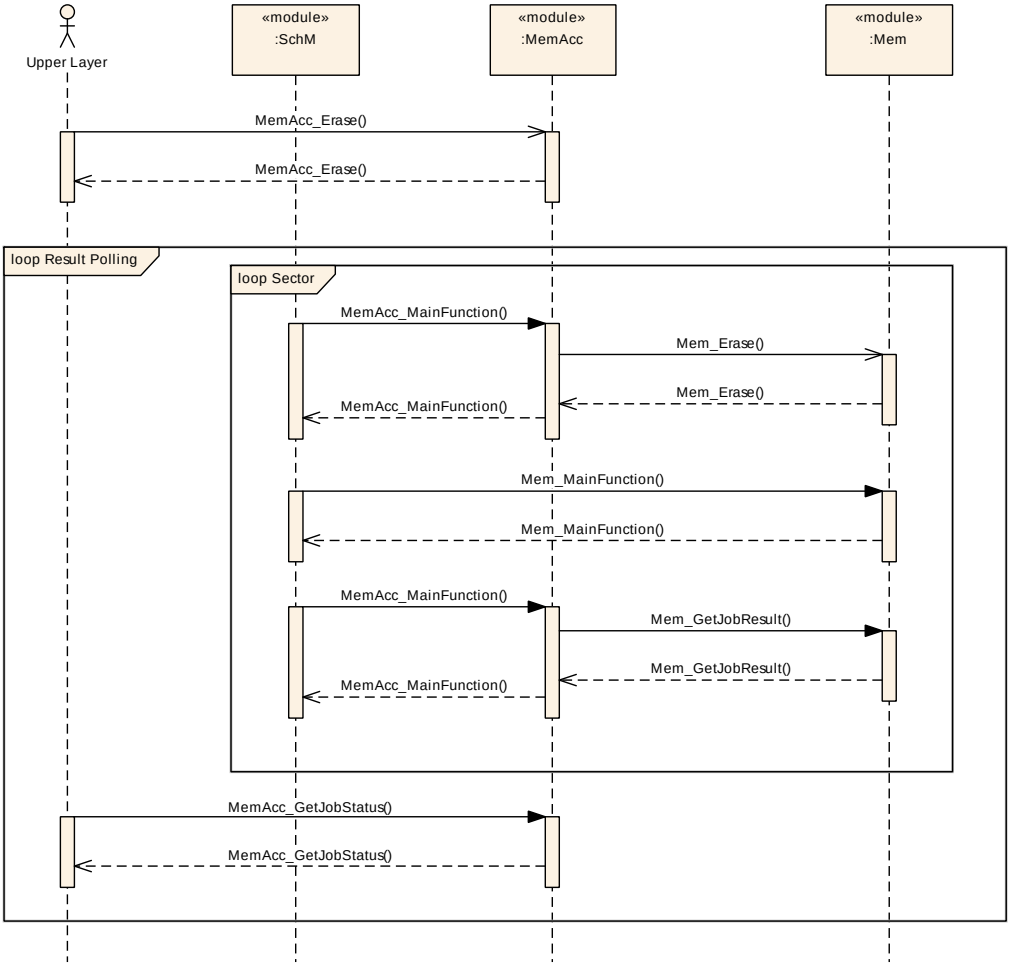


Figure 9.6: Mem Driver Scheduling by SchM

9.7 Generic Lock Sequence

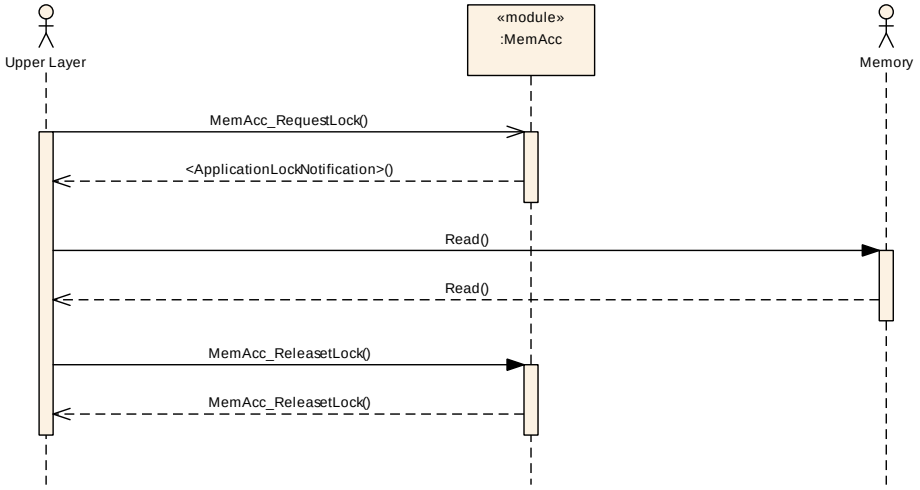


Figure 9.7: Example Lock/Unlock Sequence

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 MemAcc.

Chapter 10.3 specifies published information of the module MemAcc.

10.1 How to Read this Chapter

For details refer to [2] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Containers and Configuration Parameters

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

10.2.1 MemAcc

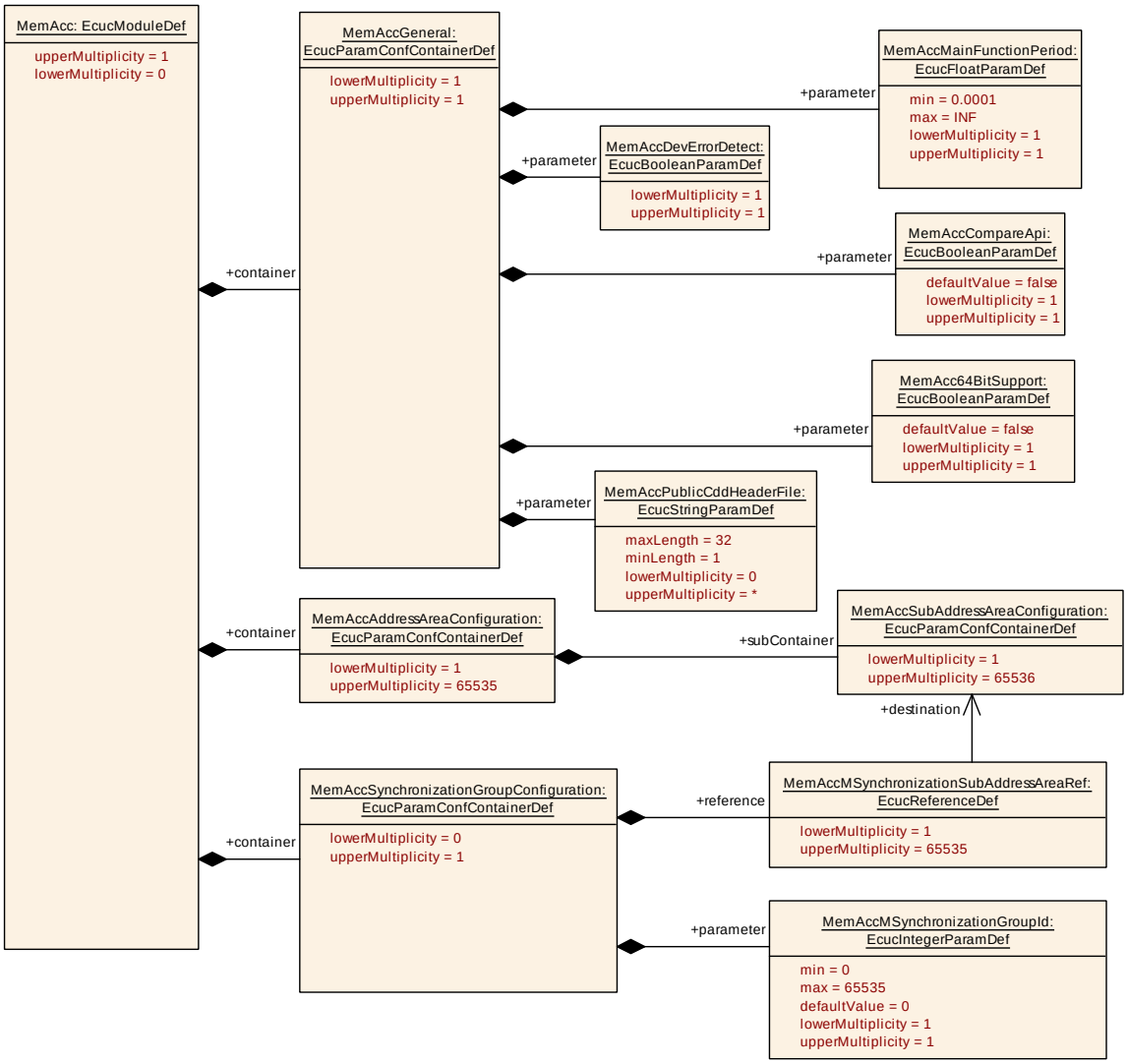


Figure 10.1: MemAcc

[ECUC_MemAcc_00001] Definition of EcucModuleDef MemAcc ⌈

Module Name	MemAcc
Description	The MemAcc module shall only coordinate conflicting resource accesses. The access dependencies shall be configured based on MemAccSynchronization GroupConfiguration. Note: Only relevant resource conflicts shall be coordinated to prevent any performance impact.
Post-Build Selectable Support	false
Post-Build Loadable Support	false

Included Containers		
Container Name	**Multiplicity**	**Description**
MemAccAddressAreaConfiguration	1..65535	This container includes the configuration of AddressArea specific parameters for the MemAcc module. An AddressArea is a logical area of memory. Upper layers only use logical addresses to access the address area. It is the job of MemAcc to map between logical and physical addresses. An AddressArea contains SubAddressAreas and each SubAddress Area is part of a physically continuous memory area (sector batch).
MemAccGeneral	1	General configuration parameters of the MemAcc.
MemAccSynchronizationGroup Configuration	0..1	This container represents one synchronization group. This container references sub address areas with same synchronization affiliation.

⌋

[ECUC_MemAcc_00002] Definition of EcucParamConfContainerDef MemAcc General ⌈

Container Name	MemAccGeneral
Parent Container	MemAcc
Upstream mapping defined	–
Description	General configuration parameters of the MemAcc.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemAcc64BitSupport	1	[ECUC_MemAcc_00024]
MemAccCompareApi	1	[ECUC_MemAcc_00006]
MemAccDevErrorDetect	1	[ECUC_MemAcc_00005]
MemAccMainFunctionPeriod	1	[ECUC_MemAcc_00004]
MemAccPublicCddHeaderFile	0..*	[ECUC_MemAcc_00028]

No Included Containers

⌋

[ECUC_MemAcc_00024] Definition of EcucBooleanParamDef MemAcc64BitSupport ⌈

Parameter Name	MemAcc64BitSupport
Parent Container	MemAccGeneral
Upstream mapping defined	–
Description	If this option is selected, the address type shall be implemented in 64Bit.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false

▽

△

Pre-Compile only	true

⌋

[ECUC_MemAcc_00006] Definition of EcucBooleanParamDef MemAccCompare Api ⌈

Parameter Name	MemAccCompareApi
Parent Container	MemAccGeneral
Upstream mapping defined	–
Description	This parameter enables/disables the function MemAcc_Compare(). This function allows to compare data stored in a buffer with data stored in memory.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_MemAcc_00005] Definition of EcucBooleanParamDef MemAccDevError Detect ⌈

Parameter Name	MemAccDevErrorDetect
Parent Container	MemAccGeneral
Upstream mapping defined	–
Description	Switches the development error detection and notification on or off.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_MemAcc_00004] Definition of EcucFloatParamDef MemAccMainFunction Period ⌈

Parameter Name	MemAccMainFunctionPeriod	
Parent Container	MemAccGeneral	
Upstream mapping defined	–	
Description	This value specifies the fixed call cycle for MemAcc_MainFunction(). Additionally, if a job is ongoing on a Mem, the underlying Mem_MainFunction will be triggered directly by MemAcc at this fixed call cycle. MemAcc does not depend on a fixed cycle time; in can be triggered at arbitrary rates.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[1E-4 .. INF]	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_MemAcc_00028] Definition of EcucStringParamDef MemAccPublicCdd HeaderFile ⌈

Parameter Name	MemAccPublicCddHeaderFile
Parent Container	MemAccGeneral
Upstream mapping defined	–
Description	Defines header files for callback functions which shall be included in case of CDDs. Range of characters is 1.. 32.
Multiplicity	0..*
Type	EcucStringParamDef
Default value	–
Length	1-32
Regular Expression	–
Pre-Compile only	true

⌋

10.2.2 MemAccAddressAreaConfiguration

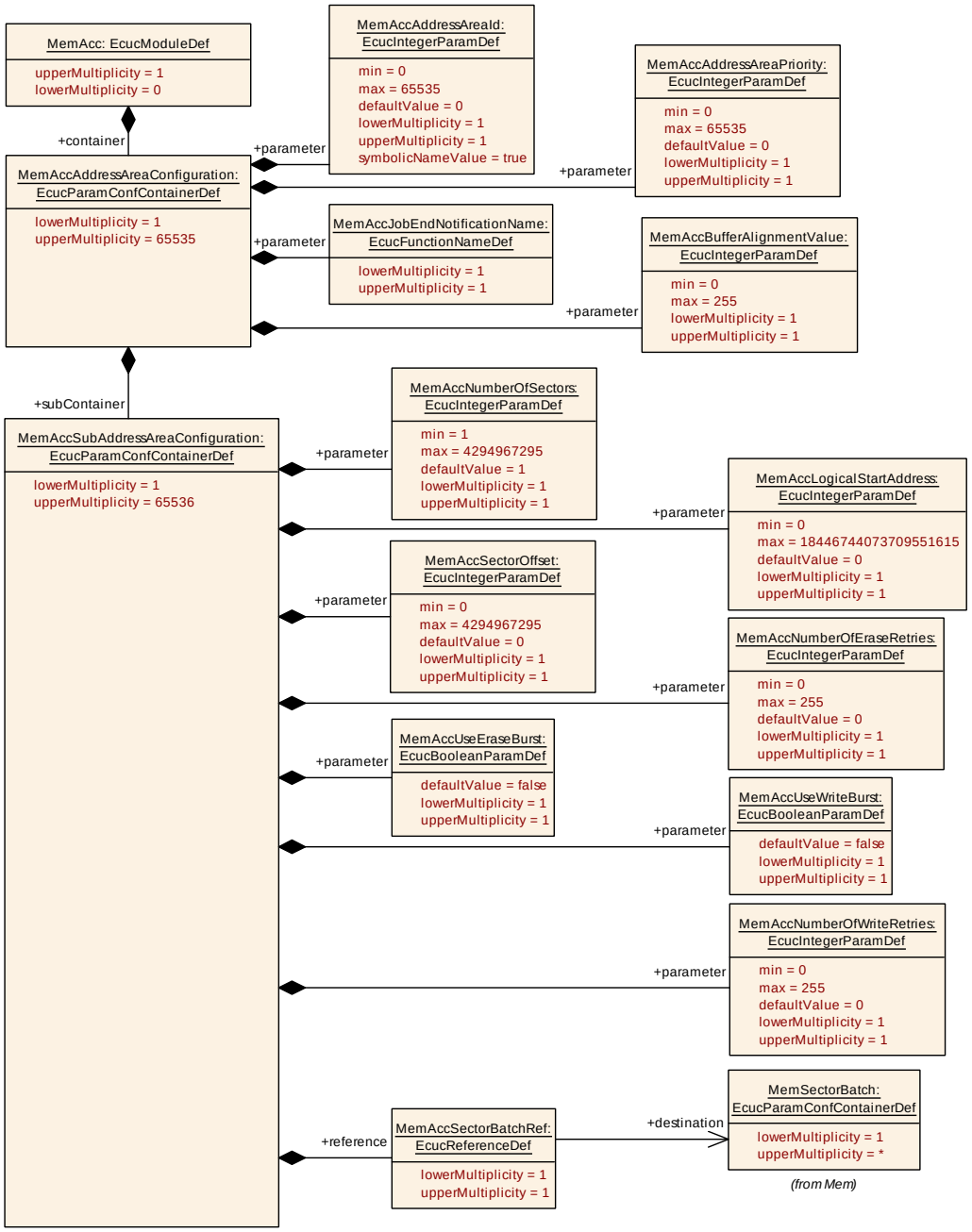


Figure 10.2: MemAccAddressAreaConfiguration

[ECUC_MemAcc_00010] Definition of EcucParamConfContainerDef MemAccAddressAreaConfiguration ⌈

Container Name	MemAccAddressAreaConfiguration
Parent Container	MemAcc
Upstream mapping defined	–

▽

△

Description	This container includes the configuration of AddressArea specific parameters for the MemAcc module. An AddressArea is a logical area of memory. Upper layers only use logical addresses to access the address area. It is the job of MemAcc to map between logical and physical addresses. An AddressArea contains SubAddressAreas and each Sub AddressArea is part of a physically continuous memory area (sector batch).	
Multiplicity	1..65535	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemAccAddressAreaId	1	[ECUC_MemAcc_00011]
MemAccAddressAreaPriority	1	[ECUC_MemAcc_00012]
MemAccBufferAlignmentValue	1	[ECUC_MemAcc_00025]
MemAccJobEndNotificationName	1	[ECUC_MemAcc_00027]

Included Containers		
Container Name	**Multiplicity**	**Description**
MemAccSubAddressArea Configuration	1..65536	This container includes the configuration parameters for a physically continuous area of memory.

⌋

[ECUC_MemAcc_00011] Definition of EcucIntegerParamDef MemAccAddressAreaId ⌈

Parameter Name	MemAccAddressAreaId	
Parent Container	MemAccAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This value specifies a unique identifier which is used to reference to an AddressArea. This identifier is used as parameter for MemAcc jobs in order to distinguish between several AddressAreas with the same logical addresses.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	0	
Pre-Compile only	true	

⌋

[ECUC_MemAcc_00012] Definition of EcucIntegerParamDef MemAccAddressAreaPriority ⌈

Parameter Name	MemAccAddressAreaPriority
Parent Container	MemAccAddressAreaConfiguration
Upstream mapping defined	–

▽

△

Description	This value specifies the priority of an AddressArea compared to other AddressAreas (0 = lowest priority, 65535 = highest priority). For each AddressArea only one job can be processed at a time. MemAcc processes the jobs priority based. In case a job with a higher priority is requested by an upper layer, the lower priority jobs are suspended until the higher priority job is completed.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00025] Definition of EcucIntegerParamDef MemAccBuffer AlignmentValue ⌈

Parameter Name	MemAccBufferAlignmentValue	
Parent Container	MemAccAddressAreaConfiguration	
Upstream mapping defined	–	
Description	Buffer alignment value inherited by MemAcc upper layer modules. The value must be a multiple of MinReadSize.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_MemAcc_00027] Definition of EcucFunctionNameDef MemAccJobEnd NotificationName ⌈

Parameter Name	MemAccJobEndNotificationName
Parent Container	MemAccAddressAreaConfiguration
Upstream mapping defined	–
Description	Job end notification function which is called after MemAcc job completion of the corresponding address area. If this parameter is left empty, no job end notification is triggered and the upper layer module needs to poll the job results.
Multiplicity	1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

10.2.3 MemAccSubAddressAreaConfiguration

[ECUC_MemAcc_00013] Definition of EcucParamConfContainerDef MemAccSub AddressAreaConfiguration ⌈

Container Name	MemAccSubAddressAreaConfiguration	
Parent Container	MemAccAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This container includes the configuration parameters for a physically continuous area of memory.	
Multiplicity	1..65536	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemAccLogicalStartAddress	1	[ECUC_MemAcc_00015]
MemAccNumberOfEraseRetries	1	[ECUC_MemAcc_00021]
MemAccNumberOfSectors	1	[ECUC_MemAcc_00014]
MemAccNumberOfWriteRetries	1	[ECUC_MemAcc_00020]
MemAccSectorOffset	1	[ECUC_MemAcc_00016]
MemAccUseEraseBurst	1	[ECUC_MemAcc_00018]
MemAccUseWriteBurst	1	[ECUC_MemAcc_00019]
MemAccSectorBatchRef	1	[ECUC_MemAcc_00023]

No Included Containers

⌋

[ECUC_MemAcc_00015] Definition of EcucIntegerParamDef MemAccLogical StartAddress ⌈

Parameter Name	MemAccLogicalStartAddress	
Parent Container	MemAccSubAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This value specifies the logical start address of the SubAddressArea.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00021] Definition of EcucIntegerParamDef MemAccNumberOf EraseRetries ⌈

Parameter Name	MemAccNumberOfEraseRetries
Parent Container	MemAccSubAddressAreaConfiguration
Upstream mapping defined	–

▽

△

Description	This value specifies the number of retries of a failed erase job. 0: No retry, a failed job will be aborted immediately > 0: Retry the number of times before aborting the job.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00014] Definition of EcucIntegerParamDef MemAccNumberOfSectors ⌈

Parameter Name	MemAccNumberOfSectors	
Parent Container	MemAccSubAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This value specifies the number of physical sectors of the SubAddressArea.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	1	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00020] Definition of EcucIntegerParamDef MemAccNumberOfWriteRetries ⌈

Parameter Name	MemAccNumberOfWriteRetries	
Parent Container	MemAccSubAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This value specifies the number of retries of a failed write job. 0: No retry, a failed job will be aborted immediately > 0: Retry the number of times before aborting the job.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00016] Definition of EcucIntegerParamDef MemAccSectorOffset ⌈

Parameter Name	MemAccSectorOffset	
Parent Container	MemAccSubAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This value specifies the sector offset of the SubAddressArea in case the SubAddress Area should not start with the first sector of the referenced MemSectorBatch.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00018] Definition of EcucBooleanParamDef MemAccUseEraseBurst ⌈

Parameter Name	MemAccUseEraseBurst	
Parent Container	MemAccSubAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This parameter enables erase bursting for the related sub address area.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00019] Definition of EcucBooleanParamDef MemAccUseWriteBurst ⌈

Parameter Name	MemAccUseWriteBurst	
Parent Container	MemAccSubAddressAreaConfiguration	
Upstream mapping defined	–	
Description	This parameter enables write bursting for the related sub address area.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00023] Definition of EcucReferenceDef MemAccSectorBatchRef ⌈

Parameter Name	MemAccSectorBatchRef
Parent Container	MemAccSubAddressAreaConfiguration
Upstream mapping defined	–

▽

△

Description	Reference to MemSectorBatch mapped to the SubAddressArea.	
Multiplicity	1	
Type	Reference to MemSectorBatch	
Post-Build Value	Loadable	Selectable

⌋

10.2.4 MemAccSynchronizationGroupConfiguration

[ECUC_MemAcc_00029] Definition of EcucParamConfContainerDef MemAccSynchronizationGroupConfiguration ⌈

Container Name	MemAccSynchronizationGroupConfiguration	
Parent Container	MemAcc	
Upstream mapping defined	–	
Description	This container represents one synchronization group. This container references sub address areas with same synchronization affiliation.	
Multiplicity	0..1	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemAccMSynchronizationGroupId	1	[ECUC_MemAcc_00030]
MemAccMSynchronizationSubAddressAreaRef	1..65535	[ECUC_MemAcc_00031]

No Included Containers

⌋

[ECUC_MemAcc_00030] Definition of EcucIntegerParamDef MemAccMSynchronizationGroupId ⌈

Parameter Name	MemAccMSynchronizationGroupId	
Parent Container	MemAccSynchronizationGroupConfiguration	
Upstream mapping defined	–	
Description	This value specifies an unique identifier which is used to reference to a synchronization group. The value is used within MemAcc internally.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_MemAcc_00031] Definition of EcucReferenceDef MemAccMSynchronizationSubAddressAreaRef ⌈

Parameter Name	MemAccMSynchronizationSubAddressAreaRef	
Parent Container	MemAccSynchronizationGroupConfiguration	
Upstream mapping defined	–	
Description	References to all SubAddressAreas relevant for this synchronization group.	
Multiplicity	1..65535	
Type	Reference to MemAccSubAddressAreaConfiguration	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

10.3 Published Information

For details refer to [2] Chapter 10.3 *“Published Information”*.

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 Specification Items in R26-11

none

A.1.2 Changed Specification Items in R26-11

Number	Heading
[ECUC_MemAcc_00001]	Definition of EcucModuleDef MemAcc
[ECUC_MemAcc_00002]	Definition of EcucParamConfContainerDef MemAccGeneral
[ECUC_MemAcc_00004]	Definition of EcucFloatParamDef MemAccMainFunctionPeriod
[ECUC_MemAcc_00005]	Definition of EcucBooleanParamDef MemAccDevErrorDetect
[ECUC_MemAcc_00006]	Definition of EcucBooleanParamDef MemAccCompareApi
[ECUC_MemAcc_00010]	Definition of EcucParamConfContainerDef MemAccAddressArea Configuration
[ECUC_MemAcc_00011]	Definition of EcucIntegerParamDef MemAccAddressAreaId
[ECUC_MemAcc_00012]	Definition of EcucIntegerParamDef MemAccAddressAreaPriority
[ECUC_MemAcc_00013]	Definition of EcucParamConfContainerDef MemAccSubAddressArea Configuration
[ECUC_MemAcc_00014]	Definition of EcucIntegerParamDef MemAccNumberOfSectors
[ECUC_MemAcc_00015]	Definition of EcucIntegerParamDef MemAccLogicalStartAddress
[ECUC_MemAcc_00016]	Definition of EcucIntegerParamDef MemAccSectorOffset
[ECUC_MemAcc_00018]	Definition of EcucBooleanParamDef MemAccUseEraseBurst
[ECUC_MemAcc_00019]	Definition of EcucBooleanParamDef MemAccUseWriteBurst
[ECUC_MemAcc_00020]	Definition of EcucIntegerParamDef MemAccNumberOfWriteRetries

▽

△

Number	Heading
[ECUC_MemAcc_00021]	Definition of EcucIntegerParamDef MemAccNumberOfEraseRetries
[ECUC_MemAcc_00023]	Definition of EcucReferenceDef MemAccSectorBatchRef
[ECUC_MemAcc_00024]	Definition of EcucBooleanParamDef MemAcc64BitSupport
[ECUC_MemAcc_00025]	Definition of EcucIntegerParamDef MemAccBufferAlignmentValue
[ECUC_MemAcc_00027]	Definition of EcucFunctionNameDef MemAccJobEndNotificationName
[ECUC_MemAcc_00028]	Definition of EcucStringParamDef MemAccPublicCddHeaderFile
[ECUC_MemAcc_00029]	Definition of EcucParamConfContainerDef MemAccSynchronizationGroup Configuration
[ECUC_MemAcc_00030]	Definition of EcucIntegerParamDef MemAccMSynchronizationGroupId
[ECUC_MemAcc_00031]	Definition of EcucReferenceDef MemAccMSynchronizationSubAddressArea Ref
[SWS_MemAcc_10037]	Definition of imported datatypes of module MemAcc

Table A.1: Changed Specification Items in R26-11

A.1.3 Deleted Specification Items in R26-11

Number	Heading
[SWS_MemAcc_00027]	
[SWS_MemAcc_10016]	Definition of API function MemAcc_GetVersionInfo

Table A.2: Deleted Specification Items in R26-11

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

A.2.1 Added Specification Items in R25-11

none

A.2.2 Changed Specification Items in R25-11

Number	Heading
[ECUC_MemAcc_00027]	Definition of EcucFunctionNameDef MemAccJobEndNotificationName
[SWS_MemAcc_00085]	

Table A.3: Changed Specification Items in R25-11

A.2.3 Deleted Specification Items in R25-11

none

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

A.3.1 Added Specification Items in R24-11

[ECUC_MemAcc_00029] [ECUC_MemAcc_00030] [ECUC_MemAcc_00031] [SWS_MemAcc_91021] [SWS_MemAcc_91022]

A.3.2 Changed Specification Items in R24-11

[ECUC_MemAcc_00001] [ECUC_MemAcc_00004] [ECUC_MemAcc_00020] [ECUC_MemAcc_00021] [SWS_MemAcc_00007] [SWS_MemAcc_00025] [SWS_MemAcc_00064] [SWS_MemAcc_00084] [SWS_MemAcc_00085] [SWS_MemAcc_00111] [SWS_MemAcc_00121] [SWS_MemAcc_00122] [SWS_MemAcc_10002] [SWS_MemAcc_10010] [SWS_MemAcc_10012] [SWS_MemAcc_10014] [SWS_MemAcc_10017] [SWS_MemAcc_10023] [SWS_MemAcc_10024] [SWS_MemAcc_10025] [SWS_MemAcc_10026] [SWS_MemAcc_10027] [SWS_MemAcc_10028] [SWS_MemAcc_10029] [SWS_MemAcc_10030] [SWS_MemAcc_10031] [SWS_MemAcc_10034] [SWS_MemAcc_10036] [SWS_MemAcc_10039] [SWS_MemAcc_91003] [SWS_MemAcc_91004] [SWS_MemAcc_91005] [SWS_MemAcc_91007] [SWS_MemAcc_91008] [SWS_MemAcc_91009] [SWS_MemAcc_91010]

A.3.3 Deleted Specification Items in R24-11

[ECUC_MemAcc_00007] [ECUC_MemAcc_00017] [ECUC_MemAcc_00026] [SWS_MemAcc_10032]

A.4 Traceable item history of this document according to AUTOSAR Release R23-11

A.4.1 Added Specification Items in R23-11

[SWS_MemAcc_00125] [SWS_MemAcc_00126] [SWS_MemAcc_00127]

A.4.2 Changed Specification Items in R23-11

[SWS_MemAcc_00003] [SWS_MemAcc_00004] [SWS_MemAcc_00021] [SWS_MemAcc_00037] [SWS_MemAcc_00042] [SWS_MemAcc_00046] [SWS_MemAcc_00060] [SWS_MemAcc_00064] [SWS_MemAcc_00101] [SWS_MemAcc_00105] [SWS_MemAcc_00106] [SWS_MemAcc_00107] [SWS_MemAcc_00108] [SWS_MemAcc_00109] [SWS_MemAcc_00112] [SWS_MemAcc_00114] [SWS_MemAcc_00120] [SWS_MemAcc_10012] [SWS_MemAcc_10013] [SWS_MemAcc_10022] [SWS_MemAcc_10023] [SWS_MemAcc_10024] [SWS_MemAcc_10026] [SWS_MemAcc_10027] [SWS_MemAcc_10028] [SWS_MemAcc_10037] [SWS_MemAcc_10039] [SWS_MemAcc_91016]

A.4.3 Deleted Specification Items in R23-11

none

B Not applicable requirements

No content.