User Manual

for S32K3 MEMACC Driver

Document Number: UM34MEMACC ASRR21-11 Rev0000R3.0.0 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	31.03.2023	NXP RTD Team	S32K3 Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 3.0.0

Chapter 2

Introduction

- Supported Derivatives
- Overview
- About This Manual
- Acronyms and Definitions
- Reference List

This User Manual describes NXP Semiconductors' AUTOSAR MemAcc Driver for S32K3XX.

AUTOSAR MemAcc Driver configuration parameters description can be found in the Tresos Configuration Plug-in section. Deviations from the specification are described in the Deviations from Requirements section.

AUTOSAR MemAcc driver requirements and APIs are described in the MemAcc Driver Software Specification Document (version R21-11) and in the Modules Documentation section.

2.1 Supported Derivatives

The software described in this document is intended to be used with the following microcontroller devices of NXP Semiconductors:

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257

- s32k322_mqfp100 / MWCT2D16S_mqfp100
- s32k322_mqfp172 / MWCT2D16S_mqfp172
- s32k324_mqfp172 / MWCT2D17S_mqfp172
- s32k324_mapbga257
- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

All of the above microcontroller devices are collectively named as S32K3.

Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power.

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR:

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

- **Boldface** style: Used for important terms, notes and warnings.
- *Italic* style: Used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
DET	Default Error Tracer
ECC	Error Correcting Code
VLE	Variable Length Encoding
N/A	Not Available
MCU	Microcontroller Unit
MEMACC	Memory Access
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEE	Flash EEPROM Emulation
FLS	Flash
RTD	Real Time Drivers
XML	Extensible Markup Language

2.5 Reference List

#	Title	Version
1	Specification of MemAcc Driver	AUTOSAR Release R21-11
2	Reference Manual	S32K3xx Reference Manual, Rev.6, Draft B, 01/2023
		S32K39 and S32K37 Reference Manual, Rev. 2 Draft A, 11/2022
3	Datasheet	S32K3xx Data Sheet, Rev. 6, 11/2022
		S32K396 Data Sheet, Rev. 1.1 — 08/2022
4	Errata	S32K358_0P14E Mask Set Errata — Rev. 28, 9/2022
		S32K396_0P40E Mask Set Errata, Rev. DEC2022, 12/2022
		S32K311_0P98C Chip Errata, 6/March/2023, 3/2023
		S32K312: Mask Set Errata for Mask 0P09C, Rev. 25/April/2022
		S32K342: Mask Set Errata for Mask 0P97C, Rev. 10, 11/2022
		S32K3x4: Mask Set Errata for Mask 0P55A/1P55A, Rev. 14/Oct/2022

Chapter 3

Driver

- Requirements
- Driver Design Summary
- Hardware Resources
- Deviations from Requirements
- Driver Limitations
- Driver usage and configuration tips
- Runtime errors
- Symbolic Names Disclaimer

3.1 Requirements

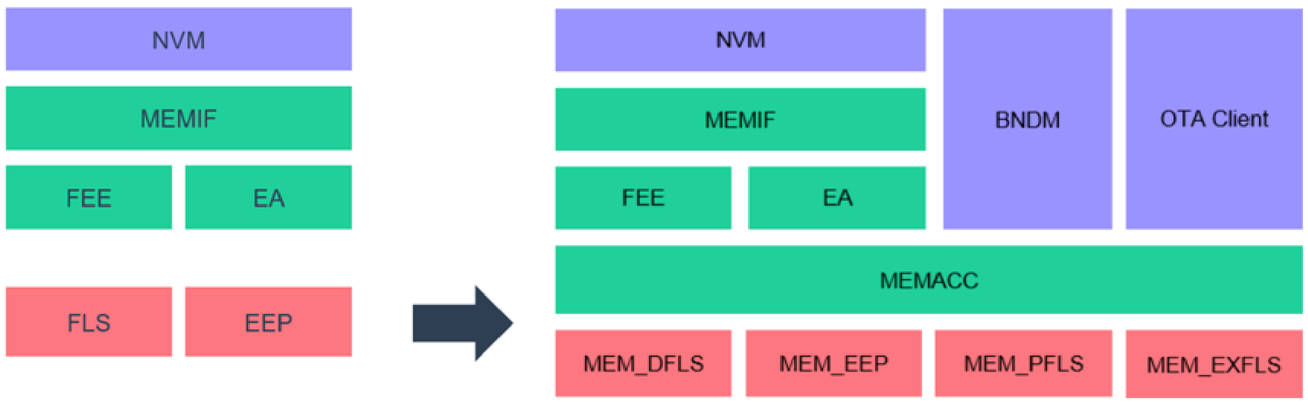
Requirements for this driver are detailed in the Autosar Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

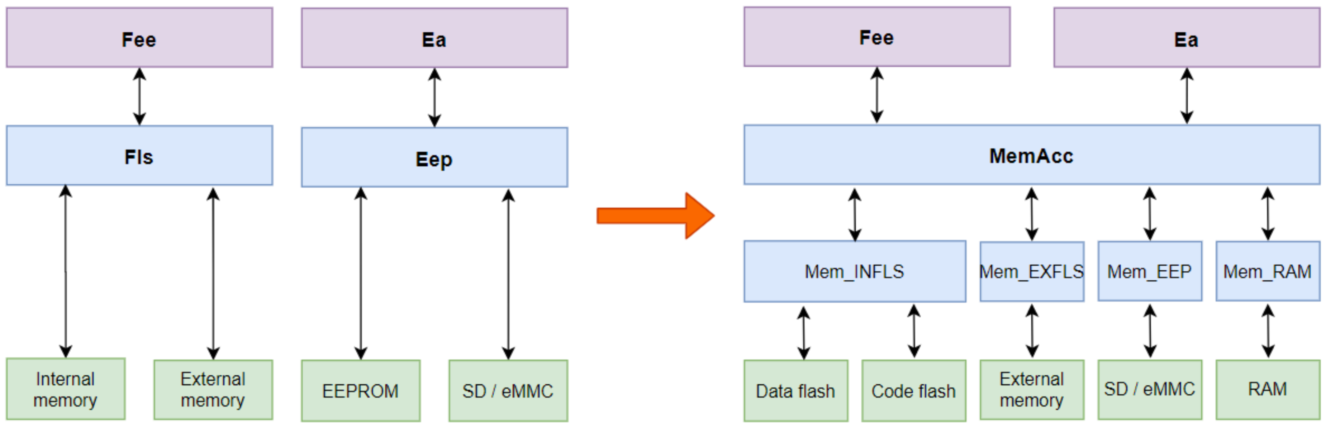
3.2.1 MemAcc stack architecture

3.2.1.1 Introduction

- Memory Access module and Memory Driver are located in the same layer as Fls and Eep, but split into:
  - Hardware independent part (**MemAcc**): minimize impact on existing memory stack
  - Hardware dependent part (**Mem**): as simple as possible, only provide basic functionalities to interact with hardware
  - APIs are the same as Fls / Eep
- Memory Access module provides access to different memory technology devices
  - Can be used with typical memory devices such as Flash, EEPROM, RAM or phase change memory (PCM)
    - Support to use specific hardware functionalities
  - Support both Fee and Ea (Fls and Eep become **obsolete** for the future)
  - Support OTA to access the code flash
  - Standardized the ECC handling

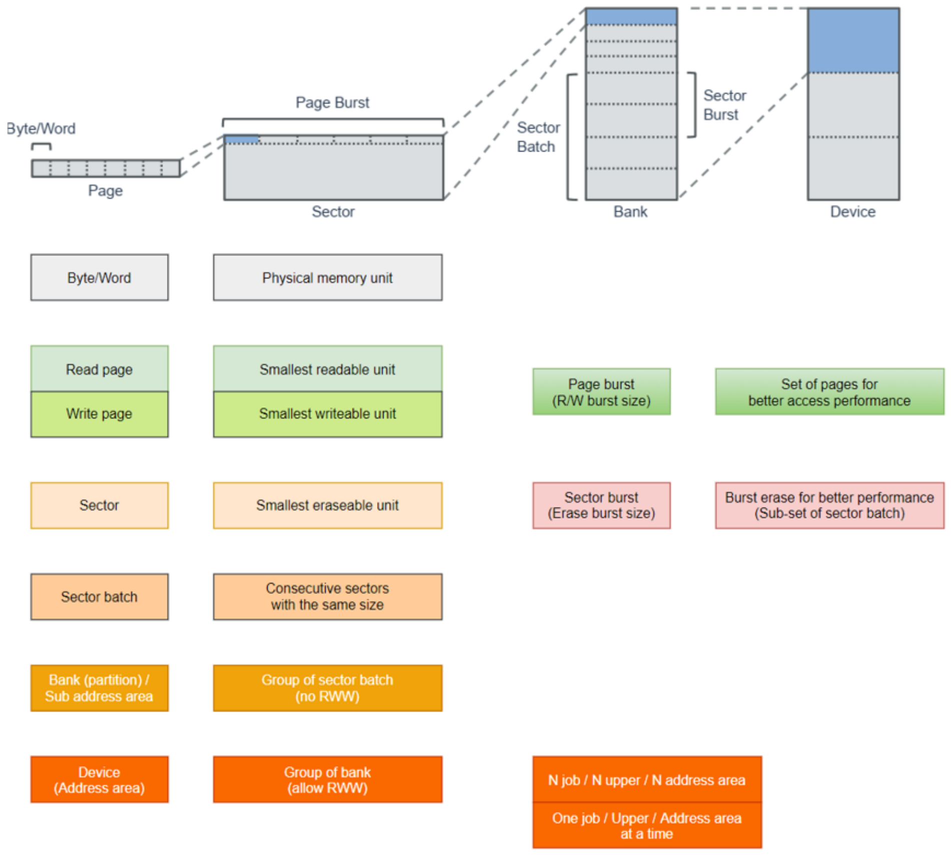


- The Mem driver's task is the low-level memory access based on physical segmentation of the memory device technology
  - Mem drivers provide basic functionalities to the Memory Access Module
  - The API of the Mem driver is memory device technology independent
- All high-level functionality like cross-segment operations are handled by the Memory Access
  - Keep the complexity and the footprint of the Mem drivers as small as possible
  - The MemAcc module provides an address-based memory access for different upper layers
  - It implements all high-level functionalities, for instance:
    - **Job management**
    - **Access coordination**
    - **Allocation of Memory Driver access requests**



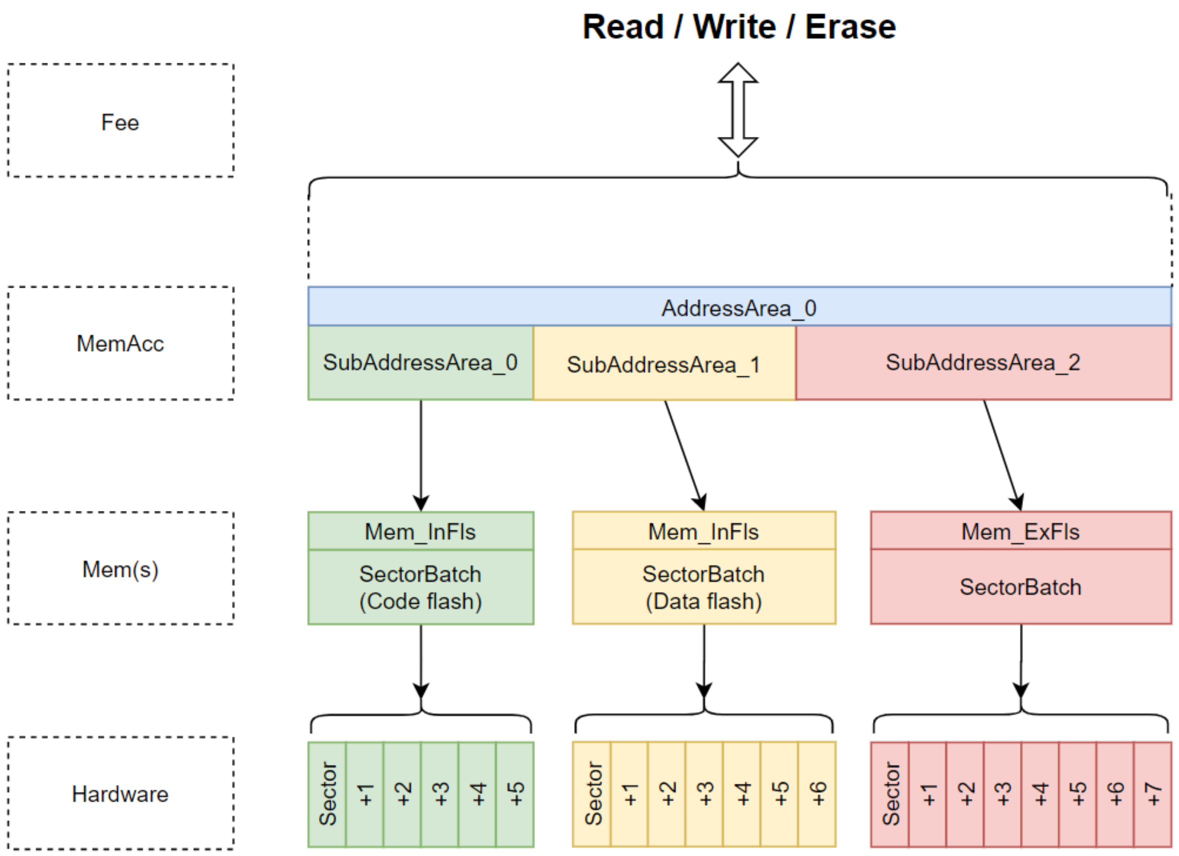
3.2.1.2 Glossaries

- Some new glossaries will be used between the MemAcc and Mem drivers:
  - Byte (the smallest unit)
  - Address area (the largest unit)
- The two important components:
  - Sector batch (in Mem layer)
  - Address area (in MemAcc layer)

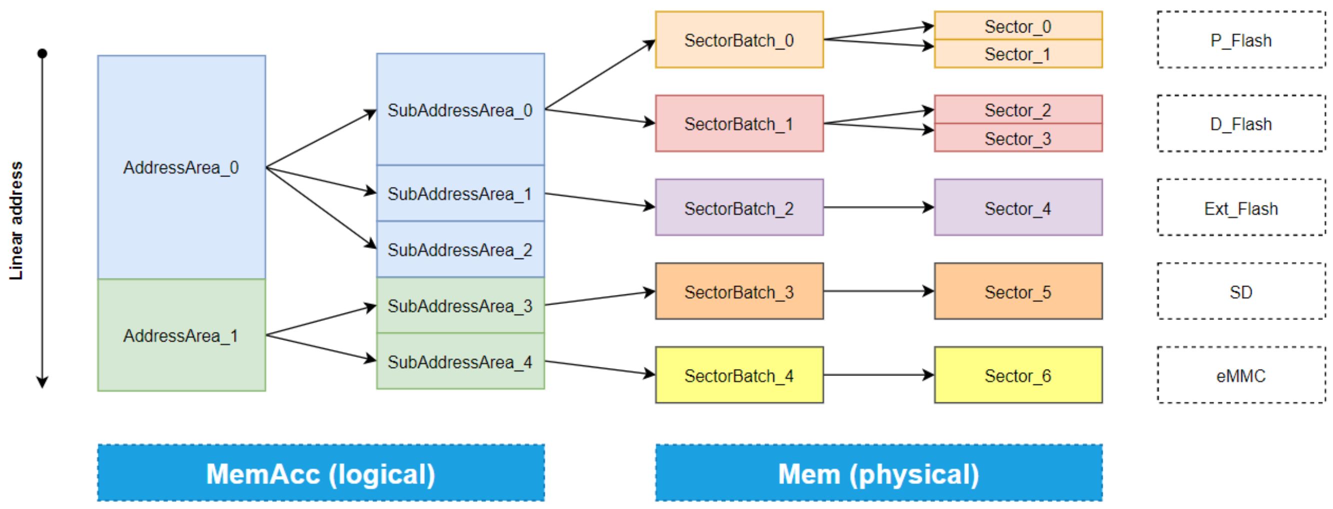

Byte/Word
Page
Page Burst
Sector
Sector Batch
Sector Burst
Bank
Device
Byte/Word
Physical memory unit
Read page
Smallest readable unit
Write page
Smallest writeable unit
Page burst (R/W burst size)
Set of pages for better access performance
Sector
Smallest eraseable unit
Sector burst (Erase burst size)
Burst erase for better performance (Sub-set of sector batch)
Sector batch
Consecutive sectors with the same size
Bank (partition) / Sub address area
Group of sector batch (no RWW)
Device (Address area)
Group of bank (allow RWW)
N job / N upper / N address area
One job / Upper / Address area at a time

3.2.1.3 Examples of configuration

- Example 1: a configuration for an AddressArea
  - Consist of 3 SubAddressArea span 3 different memory technologies:
    - ∗ Code flash (Mem_InFls)
    - ∗ Data flash (Mem_InFls)
    - ∗ External flash (Mem_ExFls)
- The SubAddressArea_0 references to a SectorBatch of the Mem_InFls driver
  - This SectorBatch consists of 6 continuous physical sectors with the same size
- For every access request to the AddressArea_0 from upper layer:
  - The MemAcc translates the logical address to the physical address corresponding to the Mem driver
  - And forwards the request to that driver

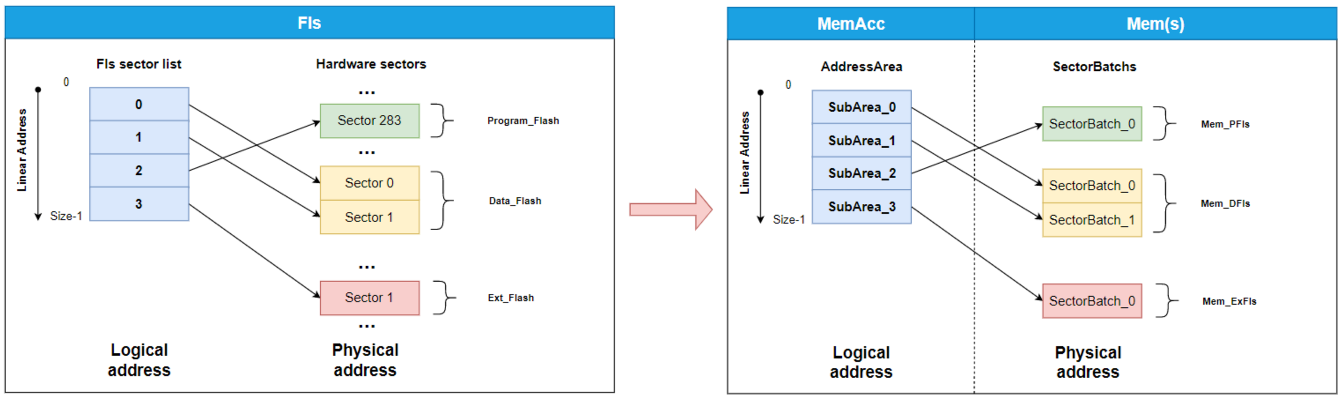


- Example 2: A configuration that uses 5 different types of memory
  - Two address areas for upper layers
  - The sector batches belong to different memory devices: Code flash, Data flash, External flash, SD card, eMMC card



3.2.1.4 Memory architectures comparison

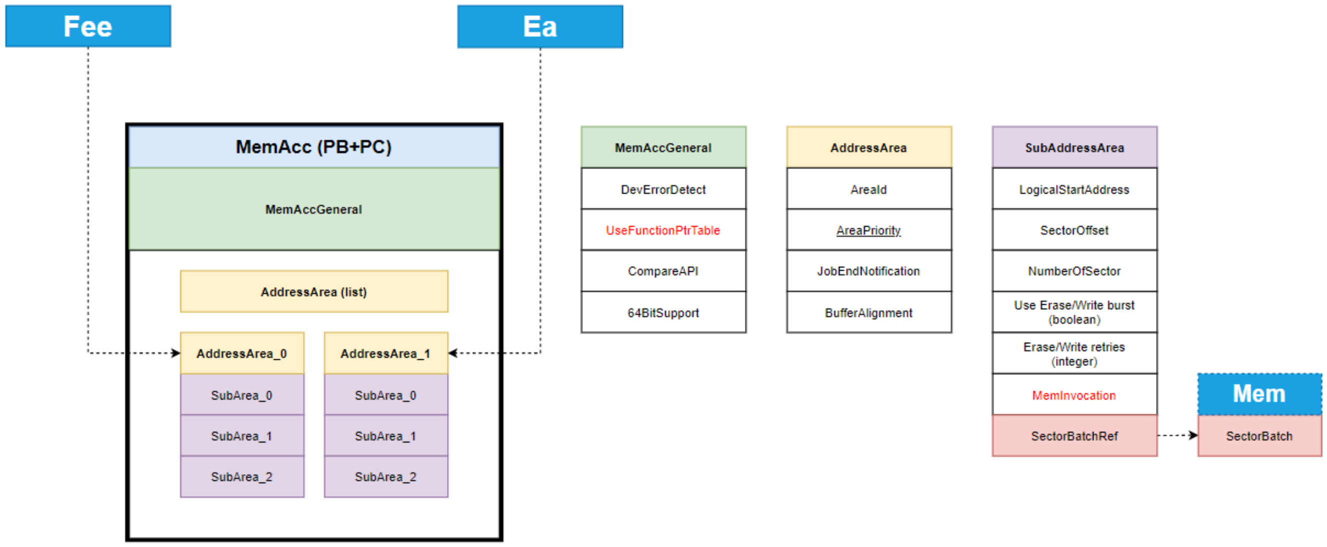
- In conclusion, we have to split the current Fls driver into 2 parts:
  - MemAcc (hardware independent)
  - Mem(s) (hardware dependent)
  - Mem_PFls and Mem_DFls will be combined into a single driver called Mem_InFls



3.2.2 User interface configuration

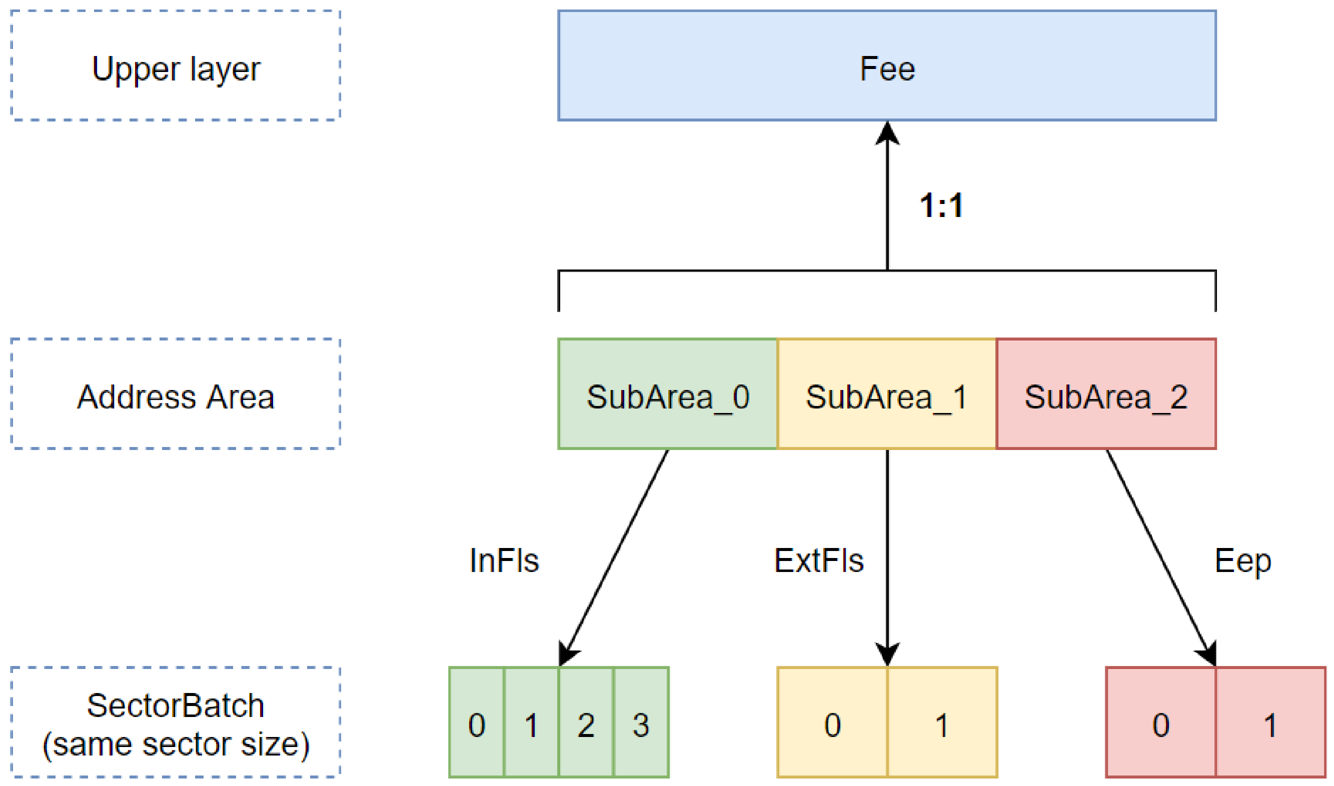
3.2.2.1 Main containers

- MemAccGeneral
  - General precompile configuration parameters
- AddressArea (list)
  - Contains SubAddressAreas and each SubAddressArea is part of a physically continuous memory area (sector batch)
  - Each AddressArea is assigned to an upper layer



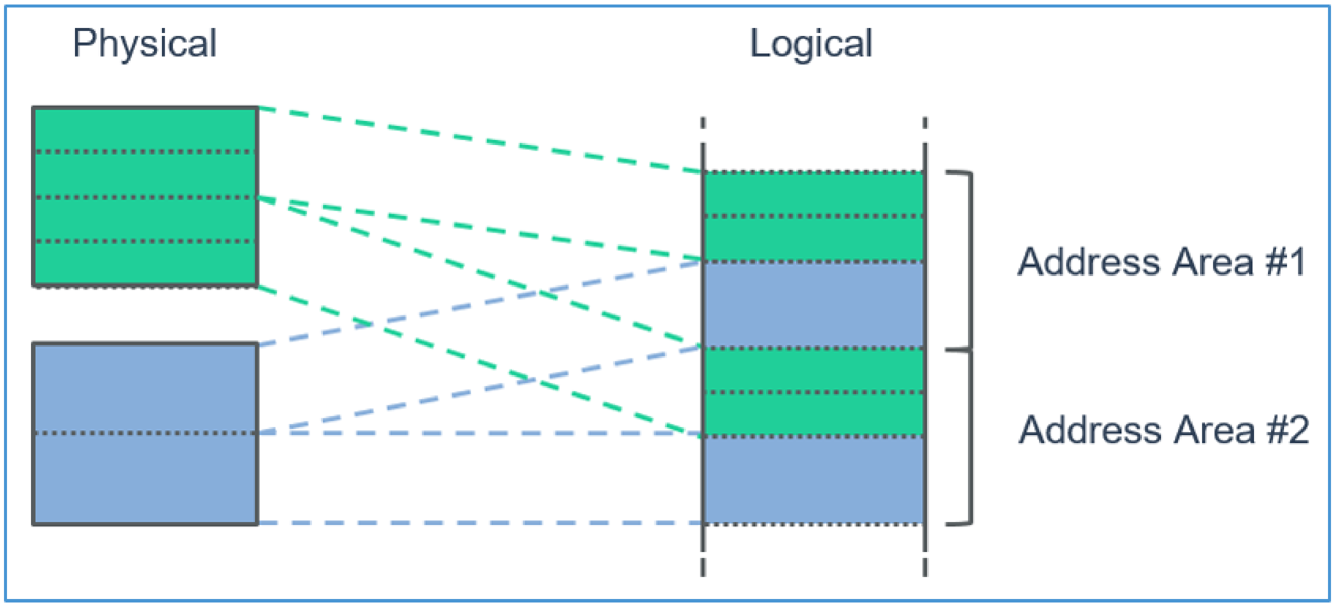
3.2.2.2 Memory address translation

- Overview
  - Merging of discontinuous physical addresses to a continuous logical address
  - Provide to upper layer a linear address (called AddressArea)
  - An address area can span multiple memory devices (contain multiple sub-address areas)
- Constraints:
  - An address area can only be assigned to one upper layer
  - Start address and length shall be aligned to the physical sectors
  - Within a sub-address area, only one sector size is allowed



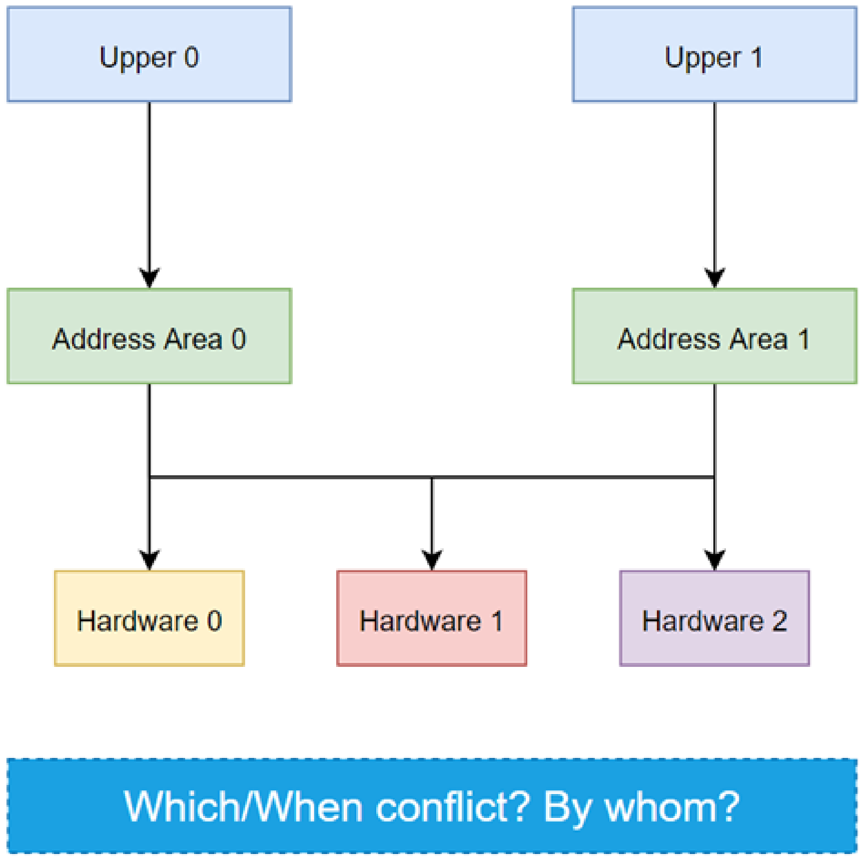
3.2.2.3 Variant support

- MemAcc supports variant mapping of two physical address areas to one logical address area at initialization time
- Use Case: OTA software update use case with active/inactive memory areas
  - The memory access from the OTA software always works with the same address area
  - A variant mapping of two physical memory areas is necessary to one logical address area is needed
  - The variant selection shall be done at startup time (reinitialize with a different configuration variant)



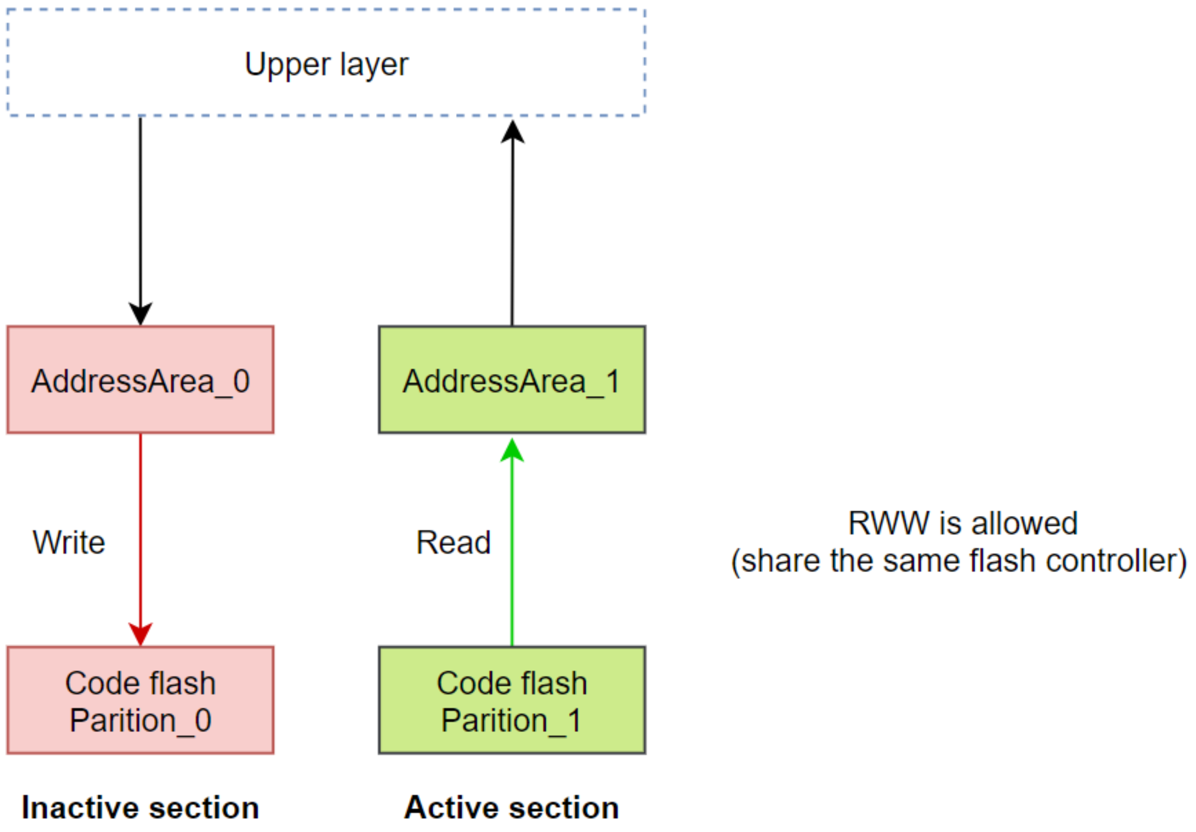
3.2.3 Job management

- Allow only one job per address area
- Support parallel job requests on different memory address areas from different upper layers
- Split access request according to physical segmentation
- Synchronization of conflicting hardware resource (based on configuration)
  - Dependencies will be configurable in the configuration tool
  - In case of resource conflict, still accept job and process once the resource is free
- Allow priority configuration for different address areas
- Use Suspend/Resume if hardware supported
- If not, implemented based on physical segmentation



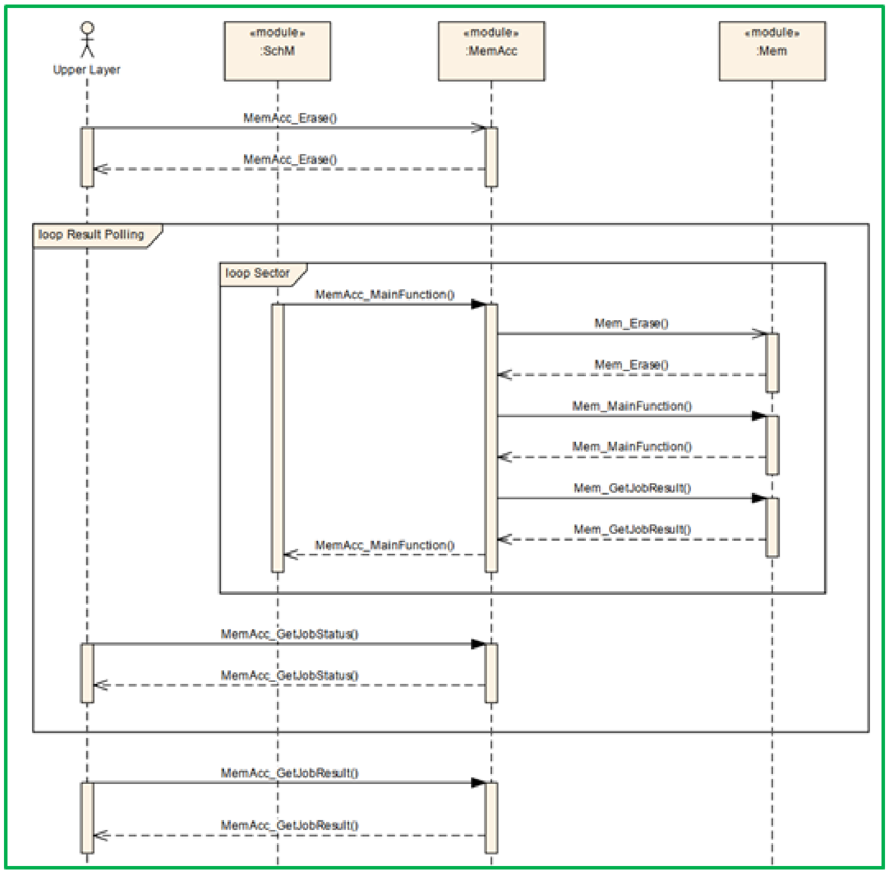
3.2.4 Read-while-write

- MemAcc manages the memory accesses from upper layers to support the capability of RWW
  - Keeps track of all on-going jobs to detect the RWW
- For example: OTA software update use case with active/inactive memory areas
  - Write the new software to an inactive memory section
  - While the active section is accessed for reading during normal execution in parallel



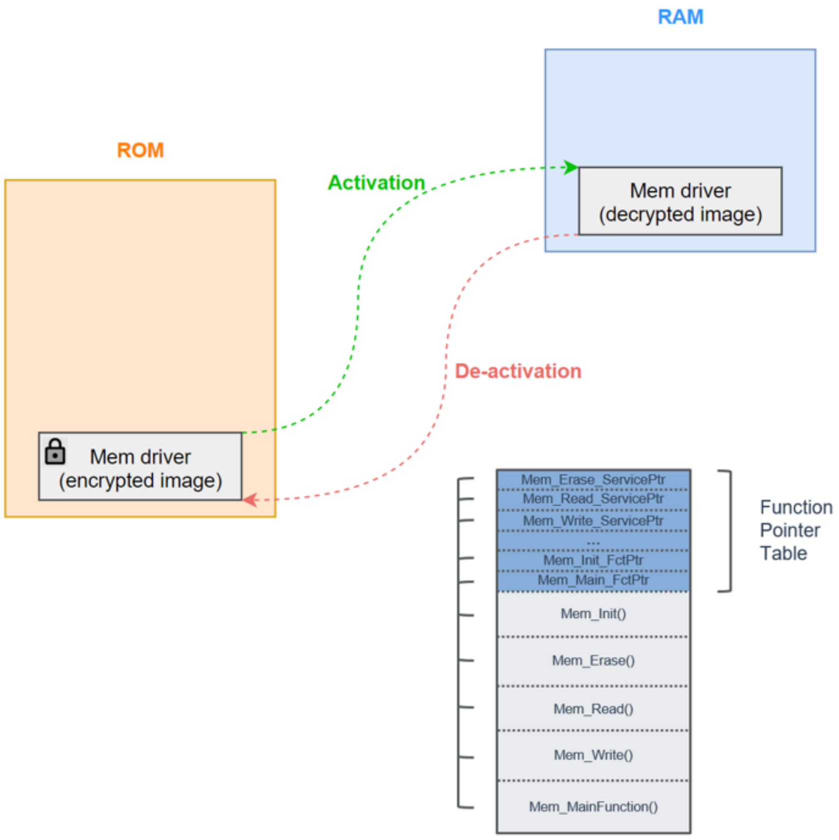
3.2.5 Job processing

There are two configurable options per AddressArea (upper layer): Polling status & Job end notification



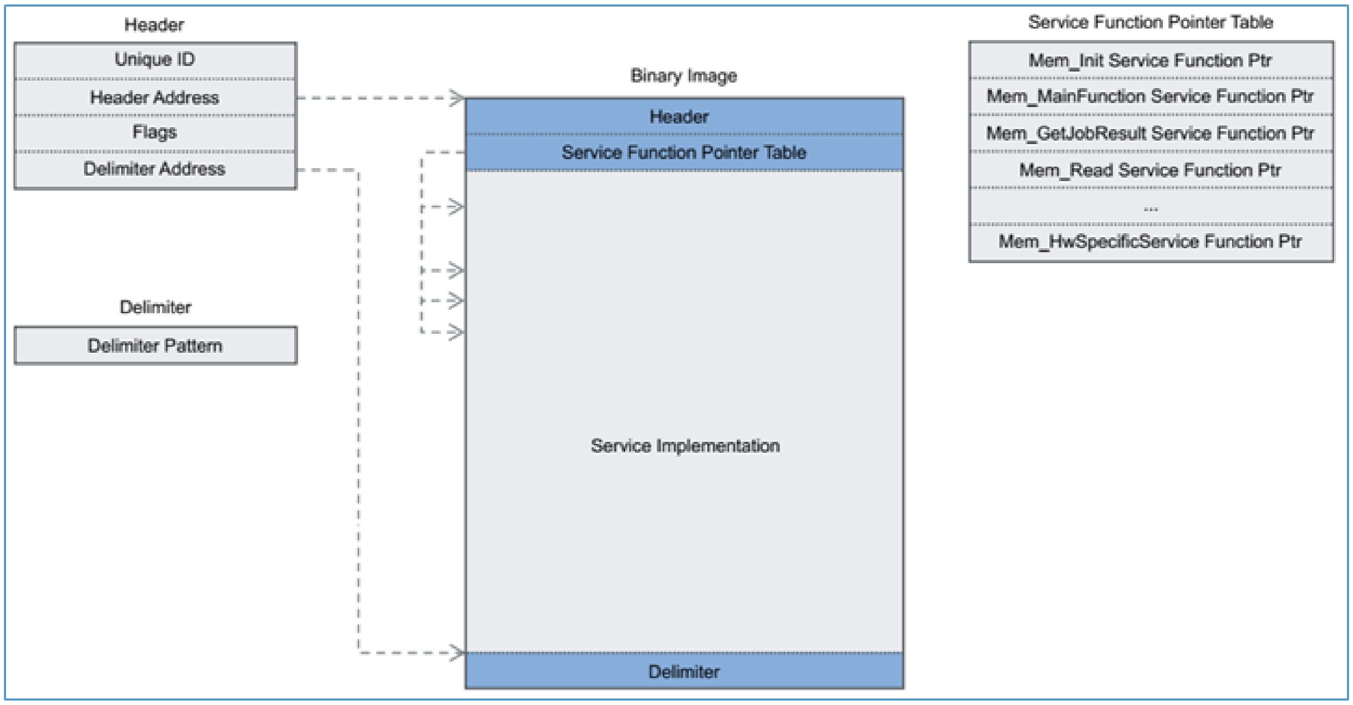
3.2.6 Dynamic Mem driver

- For safety purpose, the memory drivers are not available all the time
  - To prevent accidental calls and overwriting of memory areas
- Memory drivers might be stored in encrypted form
  - Only be downloaded dynamically and decrypted in RAM as needed
  - Similar to the feature **Load Access Code to RAM**, but the scale applies to the whole Mem driver
- Mem driver is compiled as a separate binary which contains a function pointer table to expose service functions
  - MemAcc parses the table header to find the physical address of Mem driver service functions
  - MemAcc calls service functions indirectly using function pointer
- For projects that don't need dynamic memory drivers in RAM, memory drivers can also be directly compiled and linked with the application code
- In case Mem driver binary is part of the application, the binary image should be encrypted
  - e.g. by a XOR operation and only be decrypted in RAM if the Mem driver is activated



3.2.7 Mem driver binary image format

- Header: contains management information
- Service function pointer table
- Service function implementation
- Delimiter: marks the end of the binary image


Header
Unique ID
Header Address
Flags
Delimiter Address
Delimiter
Delimiter Pattern
Binary Image
Header
Service Function Pointer Table
Service Implementation
Delimiter
Service Function Pointer Table
Mem_Init Service Function Ptr
Mem_MainFunction Service Function Ptr
Mem_GetJobResult Service Function Ptr
Mem_Read Service Function Ptr
...
Mem_HwSpecificService Function Ptr

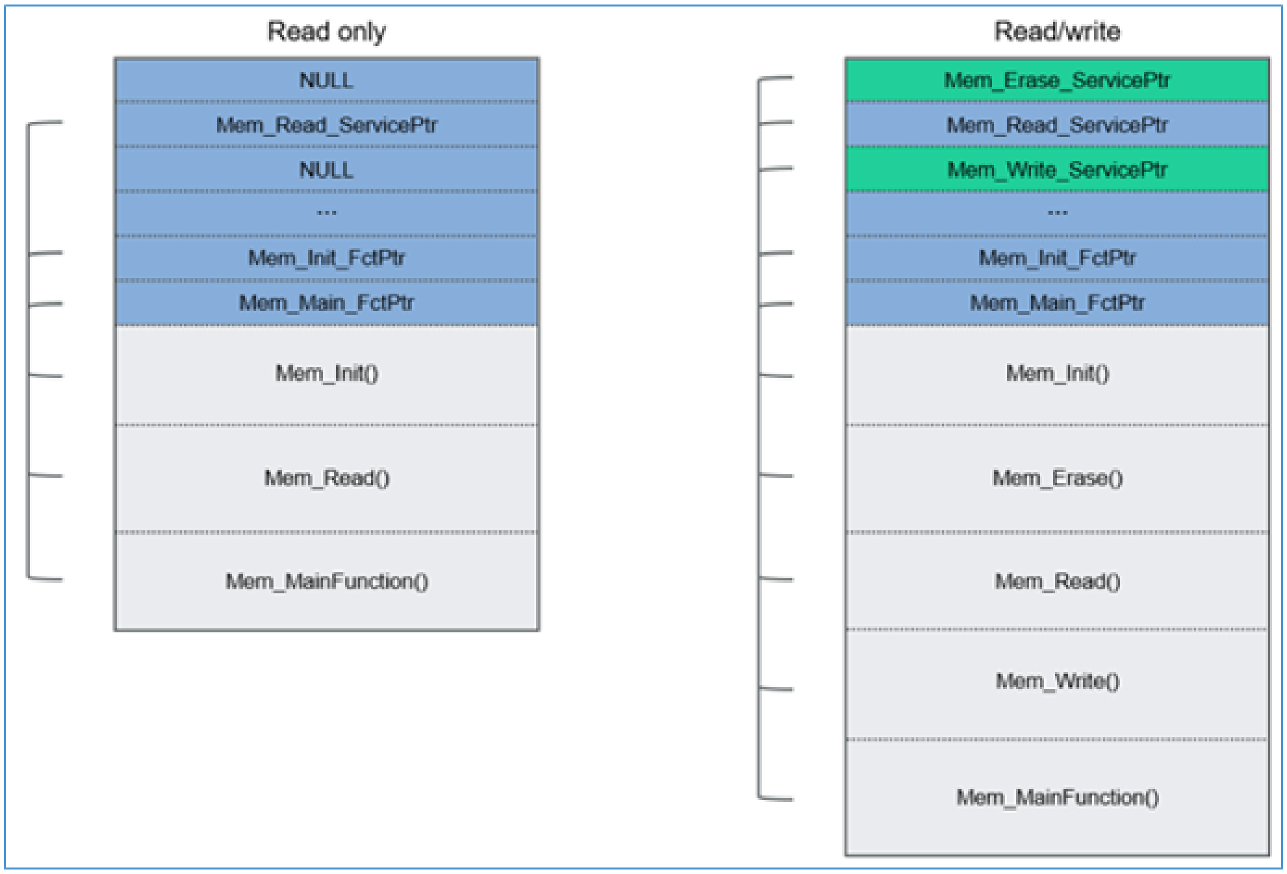
3.2.8 Optional Memory Services

Motivations:

- Some memory devices don't need an explicit erase service function
- In some cases, the erase and write service function shall not be available all the time
- Due to safety reasons, write access shall be limited to special modes-

Unavailable services are marked by a NULL pointer in the function pointer table

- MemAcc omits calling unavailable services by null pointer checks and just returns E_MEM_SERVICE_↩ NOT_AVAIL



3.2.9 Multicore (proposal)

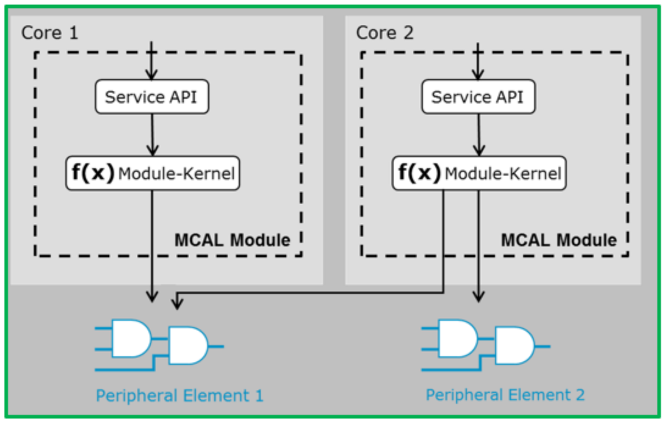
3.2.9.1 Motivation

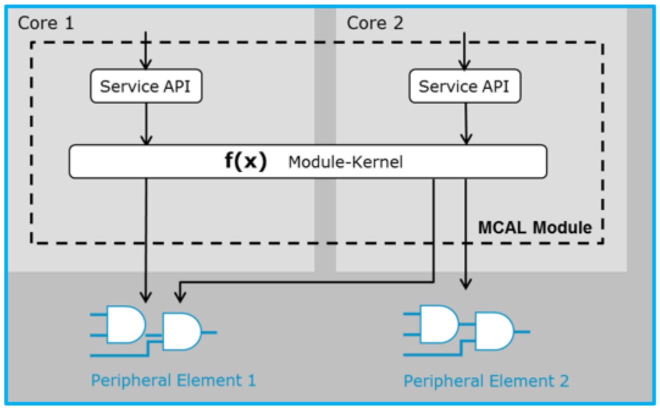
- To support multiple CPU cores share the memory resource
- Typical use case: HSM and host core share the same data flash controller
- A core can access all hardware resource elements (Cross Core Sharing)
- The shared resource initialization can be configurable to be performed by a designated core Only MemAcc is multi core capable
- Provides measures for synchronizing the memory accesses

There are two options depending on the scope of Kernel Execution (memory stack driver)

- One Local kernel (executable on one core only) => Multi Core Type 1
- Can use Semaphore for synchronization
- Global (shared) kernel (executable on any core) => Multi Core Type 3

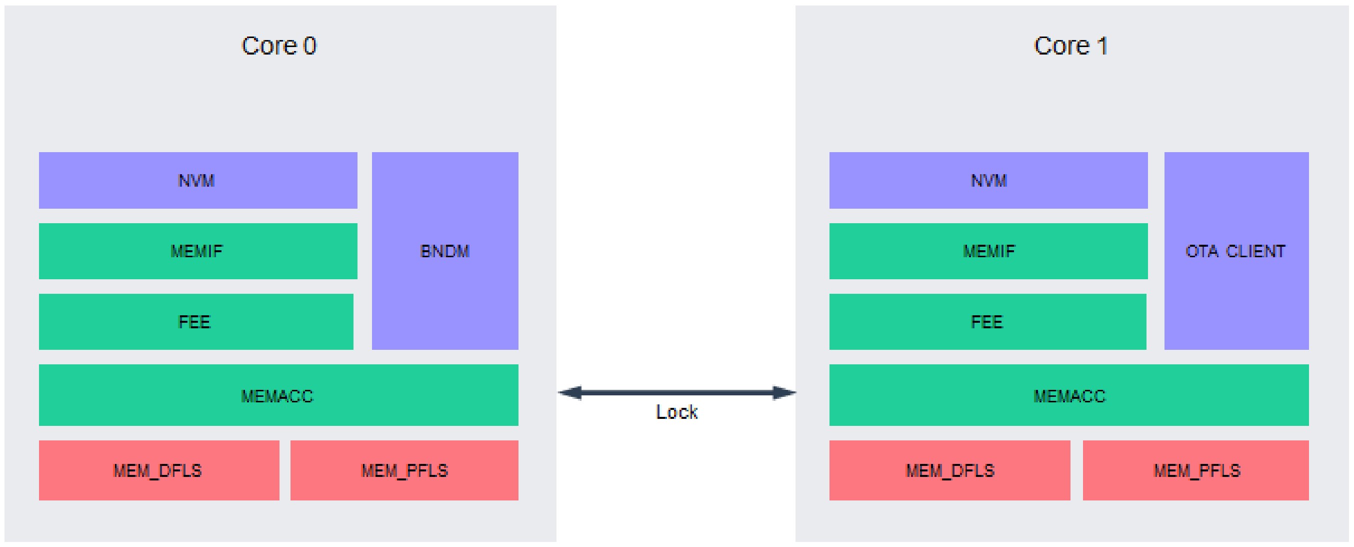
- Can use a shared global variable space for synchronization
- If the HSM core has its own memory space and does not share with other cores, this option cannot be used Another solution is Master-Satellite approach => Multi Core Type 4 -But RTD does not implement this type





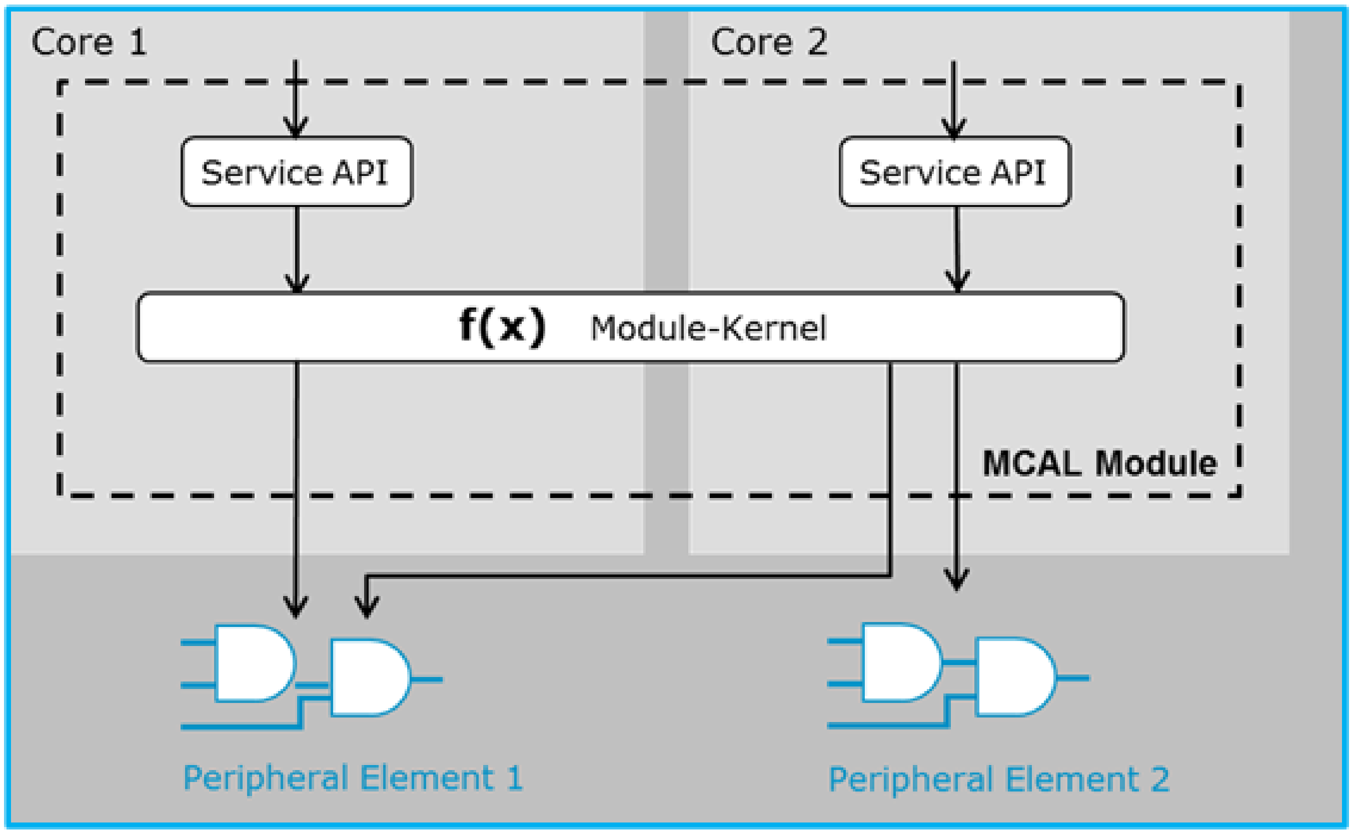
3.2.9.2 Type 1: One Local kernel

- No data exchange between the cores
- Each core accesses the memory by its own memory stack
- MemAcc only synchronizes memory driver access by a locking mechanism (e.g: Semaphore)
  - For better performance: only apply spin lock for the shared hardwares



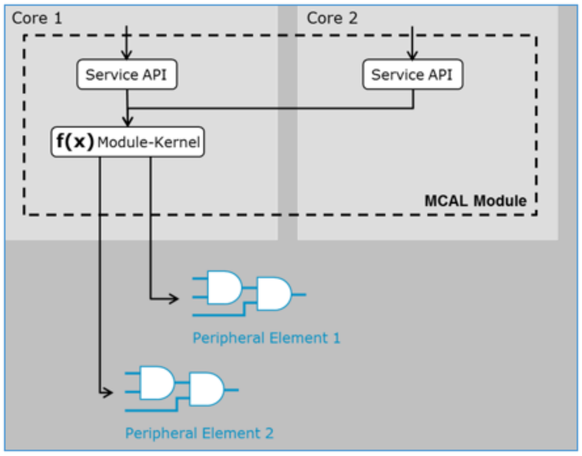
3.2.9.3 Type 3: Global kernel

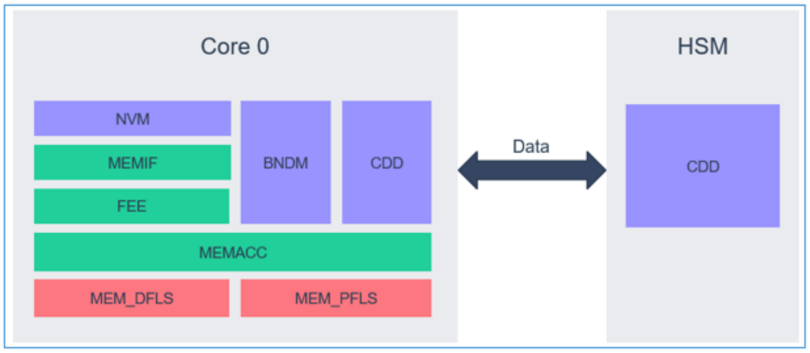
- Need a single global variable space, all global variables are shared by the Service APIs on different cores
  - Placed in non-cacheable section
  - Each core operates on its own set of data which is placed in an array, indexed by the Core/Partition ID



3.2.9.4 Type 4: Master - Satellite

- Type IV: Memory access proxy option with single memory driver on one CPU core
  - Please note that: RTD does not implement type 4
- Only one core performs the physical memory access
- MemAcc does have to implement locking mechanism
- Memory driver job requests are forwarded to the master
  - Read / write data is exchanged between both cores
  - Data needs to be encrypted for security use cases
- This approach is typically applied for shared memory access between host and HSM core





3.3 Hardware Resources

The MemAcc driver is hardware independent.

3.4 Deviations from Requirements

The driver deviates from the AUTOSAR MEMACC Driver software specification in some places.

Term	Definition
N/A	Not available
N/T	Not testable
N/S	Out of scope
N/I	Not implemented
N/F	Not fully implemented

Below table identifies the AUTOSAR requirements that are not fully implemented, implemented differently, not available, not testable or out of scope for the MemAcc driver.

Requirement	Status	Description	Notes
SWS_MemAcc_10022	N/I	MemAcc_GetJobInfo API.	Will be implemented on next release.
SWS_MemAcc_00040	N/I	If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJob↩ Info shall check that the MemAcc module has been initialized. If this check fails, MemAcc_GetJobInfo shall raise the development error MEMACC_E_UNINIT.	Will be implemented on next release.
SWS_MemAcc_00041	N/I	If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJob↩ Info 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.	Will be implemented on next release.
SWS_MemAcc_00042	N/I	If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_GetJob↩ Info shall raise the development error MEMACC_E_PARAM_↩ POINTER if the memoryInfoPtr argument is a NULL pointer.	Will be implemented on next release.
SWS_MemAcc_10028	N/I	MemAcc_HwSpecificService API.	Will be implemented on next release.
SWS_MemAcc_00066	N/I	If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_HwSpecific↩ Service shall check that the MemAcc module has been initialized. If this check fails, MemAcc_HwSpecific↩ Service shall raise the development error MEMACC_E_UNINIT.	Will be implemented on next release.
SWS_MemAcc_00067	N/I	If development error detection is enabled by MemAccDevError↩ Detect, the service MemAcc_Hw↩ SpecificService shall check that the provided addressAreaId is consistent with the configuration. If this check fails, MemAcc_HwSpecific↩ Service shall raise the development error MEMACC_E_PARAM_↩ ADDRESS_AREA_ID.	Will be implemented on next release.

Requirement	Status	Description	Notes
SWS_MemAcc_00068	N/I	If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_HwSpecific↩ Service 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.	Will be implemented on next release.
SWS_MemAcc_00070	N/I	If development error detection is enabled by MemAccDevErrorDetect, the service MemAcc_HwSpecific↩ Service shall raise the development error MEMACC_E_BUSY if a previous MemAcc job for the same addressAreaId is still being processed.	Will be implemented on next release.

3.5 Driver Limitations

- The following features/APIs are not suported:
  - MemAcc_HwSpecificService
  - MemAcc_GetJobInfo
  - Multicore.
- Timing supervision is not supported

3.6 Driver usage and configuration tips

None.

3.7 Runtime errors

- There are no runtime errors.

3.7.1 Transient Faults

- The MemAcc driver does not use transient faults.

3.7.2 Development Errors

- The development error types defined for MemAcc are listed in the table:

Type of error	Related error	Value (hex)
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

3.8 Symbolic Names Disclaimer

All containers having symbolicNameValue set to TRUE in the AUTOSAR schema will generate defines like:

#define <Mip>Conf_<Container_ShortName>_<Container_ID>

For this reason it is forbidden to duplicate the names of such containers across the RTD configurations or to use names that may trigger other compile issues (e.g. match existing `#ifdefs` arguments).

Chapter 4

Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the driver. All the parameters are described below.

- Module MemAcc
  - Container MemAccGeneral
    - Parameter MemAccDevErrorDetect
    - Parameter MemAccUseMemFuncPtrTable
    - Parameter MemAcc64BitSupport
    - Parameter MemAccCompareApi
    - Parameter MemAccCompareBufferSize
    - Parameter MemAccMainFunctionPeriod
    - Parameter MemAccTimeoutMethod
  - Container MemAccAddressAreaConfiguration
    - Parameter MemAccAddressAreaId
    - Parameter MemAccAddressAreaPriority
    - Parameter MemAccBufferAlignmentValue
    - Parameter MemAccJobEndNotificationName
    - Container MemAccSubAddressAreaConfiguration
      - Parameter MemAccLogicalStartAddress
      - Parameter MemAccSectorOffset
      - Parameter MemAccNumberOfSectors
      - Parameter MemAccMemInvocation
      - Parameter MemAccMemNamePrefix
      - Parameter MemAccNumberOfEraseRetries
      - Parameter MemAccNumberOfWriteRetries
      - Parameter MemAccUseEraseBurst
      - Parameter MemAccUseReadBurst
      - Parameter MemAccUseWriteBurst
      - Reference MemAccSectorBatchRef
  - Container AutosarExt
    - Parameter MemAccEnableUserModeSupport
  - Container CommonPublishedInformation

- ∗ Parameter ArReleaseMajorVersion
- ∗ Parameter ArReleaseMinorVersion
- ∗ Parameter ArReleaseRevisionVersion
- ∗ Parameter ModuleId
- ∗ Parameter SwMajorVersion
- ∗ Parameter SwMinorVersion
- ∗ Parameter SwPatchVersion
- ∗ Parameter VendorApiInfix
- ∗ Parameter VendorId

4.1 Module MemAcc

Configuration of the MemAcc module.

Included containers:

- MemAccGeneral
- MemAccAddressAreaConfiguration
- AutosarExt
- CommonPublishedInformation

Property	Value
type	ECUC-MODULE-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantSupport	true
supportedConfigVariants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

4.2 Container MemAccGeneral

Container for general parameters of the flash driver. These parameters are always pre-compile.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.3 Parameter MemAccDevErrorDetect

Pre-processor switch to enable and disable development error detection.

true : Development error detection enabled.

false: Development error detection disabled.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.4 Parameter MemAccUseMemFuncPtrTable

This parameter defines if the Mem driver functions are called using the Mem function pointer table API.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.5 Parameter MemAcc64BitSupport

If this option is selected, the address type shall be implemented in 64Bit.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.6 Parameter MemAccCompareApi

This parameter enables/disables the function MemAcc_Compare().

This function allows to compare data stored in a buffer with data stored in memory.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.7 Parameter MemAccCompareBufferSize

This value specifies the maximum number of bytes the MemAcc

module requests within one Mem compare request.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP

Property	Value
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	128
max	1024
min	1

4.8 Parameter MemAccMainFunctionPeriod

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.

Property	Value
type	ECUC-FLOAT-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0.2
max	1.0
min	1.0E-4

4.9 Parameter MemAccTimeoutMethod

Vendor specific: Counter type used in timeout detection for driver operations.

Based on selected counter type the timeout value will be interpreted as follows:

OSIF_COUNTER_DUMMY - Counts the number of iterations of the waiting loop. The actual timeout depends on many factors: operation type, compiler optimizations, interrupts or other tasks in the system, etc.

OSIF_COUNTER_SYSTEM - Microseconds.

OSIF_COUNTER_CUSTOM - Defined by user implementation of timing services

Note: Qspi always uses timeout

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	OSIF_COUNTER_DUMMY
literals	['OSIF_COUNTER_DUMMY', 'OSIF_COUNTER_SYSTEM', 'OSIF_↩ COUNTER_CUSTOM']

4.10 Container MemAccAddressAreaConfiguration

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 SubAddressArea is part of a physically continuous memory area (sector batch).

Included subcontainers:

- MemAccSubAddressAreaConfiguration

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	65535
postBuildVariantMultiplicity	true
multiplicityConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE

4.11 Parameter MemAccAddressAreaId

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.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	65535
min	0

4.12 Parameter MemAccAddressAreaPriority

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.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	65535
min	0

4.13 Parameter MemAccBufferAlignmentValue

Buffer alignment value inherited by MemAcc upper layer modules.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.14 Parameter MemAccJobEndNotificationName

Job end notification function which is called after MemAcc job completion.

If this parameter is left empty, no job end notification is triggered and the upper layer module needs to poll the job results.

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	NULL_PTR

4.15 Container MemAccSubAddressAreaConfiguration

This container includes the configuration parameters for a physically continuous area of memory.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	65535
postBuildVariantMultiplicity	true
multiplicityConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE

4.16 Parameter MemAccLogicalStartAddress

This value specifies the logical start address of the SubAddressArea.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	9223372036854775807
min	0

4.17 Parameter MemAccSectorOffset

This value specifies the sector offset of the SubAddressArea in case the SubAddressArea should not start with the first sector of the referenced MemSectorBatch.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false

Property	Value
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.18 Parameter MemAccNumberOfSectors

This value specifies the number of physical sectors of the SubAddressArea.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	4294967295
min	1

4.19 Parameter MemAccMemInvocation

Defines how the Mem driver services are accessed and how the Mem driver is scheduled and activated/initialized.

DIRECT_STATIC : Mem driver is linked with application. Mem service functions are directly called by MemAcc. Mem_Init is called by EcuM and Mem_MainFunction is triggered by SchM.

INDIRECT_DYNAMIC: Mem driver is linked as a separate binary and is dynamically activated. MemAcc will use Mem driver header table to invoke Mem service functions. Call of Mem_Init and Mem_MainFunction is handled by MemAcc.

INDIRECT_STATIC : Mem driver is linked with application. MemAcc will use Mem driver header table to invoke Mem service functions. Call of Mem_Init and Mem_MainFunction is handled by MemAcc.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	DIRECT_STATIC
literals	['DIRECT_STATIC', 'INDIRECT_DYNAMIC', 'INDIRECT_STATIC']

4.20 Parameter MemAccMemNamePrefix

Depending on the MemAccUseMemFuncPtrTable configuration, this prefix is either used to reference the Mem driver header structure or the according Mem API function.

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
defaultValue	Mem_43_ExFls

4.21 Parameter MemAccNumberOfEraseRetries

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.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.22 Parameter MemAccNumberOfWriteRetries

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.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.23 Parameter MemAccUseEraseBurst

This parameter enables erase bursting for the related sub address area.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.24 Parameter MemAccUseReadBurst

This parameter enables read bursting for the related sub address area.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.25 Parameter MemAccUseWriteBurst

This parameter enables write bursting for the related sub address area.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.26 Reference MemAccSectorBatchRef

Reference to MemSectorBatch mapped to the SubAddressArea.

Property	Value
type	ECUC-CHOICE-REFERENCE-DEF
origin	AUTOSAR_ECUC
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destinations	['/Mem_43_InFls_TS_T40D34M30I0R0/Mem/MemInstance/MemSector↩ Batch', '/Mem_43_ExFls_TS_T40D34M30I0R0/Mem/MemInstance/Mem↩ SectorBatch', '/Mem_43_Eep_TS_T40D34M30I0R0/Mem/MemInstance/↩ MemSectorBatch']

4.27 Container AutosarExt

Vendor specific: This container contains the global Non-Autosar configuration parameters of the driver.

This container is a MultipleConfigurationContainer, i.e. this container and

its sub-containers exist once per configuration set.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.28 Parameter MemAccEnableUserModeSupport

Vendor specific: When this parameter is enabled, the module will adapt to run from User Mode, with the following measures:

configuring REG_PROT for IPs so that the registers under protection can be accessed from user mode by setting UAA bit in REG_PROT_GCR to 1

for more information and availability on this platform, please see chapter User Mode Support in IM

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
defaultValue	false

4.29 Container CommonPublishedInformation

Common container, aggregated by all modules. It contains published information about vendor and versions.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF

Property	Value
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.30 Parameter ArReleaseMajorVersion

Vendor specific: Major version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
defaultValue	4
max	4
min	4

4.31 Parameter ArReleaseMinorVersion

Vendor specific: Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION

Property	Value
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
defaultValue	7
max	7
min	7

4.32 Parameter ArReleaseRevisionVersion

Vendor specific: Patch version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.33 Parameter ModuleId

Vendor specific: Module ID of this module.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION

Property	Value
defaultValue	41
max	41
min	41

4.34 Parameter SwMajorVersion

Vendor specific: Major version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
defaultValue	3
max	3
min	3

4.35 Parameter SwMinorVersion

Vendor specific: Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION

Property	Value
defaultValue	0
max	0
min	0

4.36 Parameter SwPatchVersion

Vendor specific: Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.37 Parameter VendorApiInfix

Vendor specific: In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the name of APIs is extended by the VendorId and a vendor specific name.

This parameter is used to specify the vendor specific name. In total, the implementation specific name is generated as follows:

<ModuleName>_<VendorId>_<VendorApiInfix>.

E.g. assuming that the VendorId of the implementor is 123 and the implementer chose a VendorApiInfix of "v11r456" a api name Can_Write defined in the SWS will translate to Can_123_v11r456Write.

This parameter is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

Property	Value
type	ECUC-STRING-PARAM-DEF

Property	Value
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
defaultValue	

4.38 Parameter VendorId

Vendor specific: Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
defaultValue	43
max	43
min	43

Chapter 5

Module Index

5.1 Software Specification

Here is a list of all modules:

Chapter 6

Module Documentation

6.1 Driver

6.1.1 Detailed Description

Data Structures

- struct MemAcc_MemoryInfoType

  MemAcc memory infomation type. More...
- struct MemAcc_JobInfoType

  MemAcc job infomation type. More...
- struct MemAcc_MemApiType

  MemAcc_MemApiType. More...
- struct MemAcc_SubAddressAreaType

  MemAcc_SubAddressAreaType. More...
- struct MemAcc_AddressAreaType

  MemAcc_AddressAreaType. More...
- struct MemAcc_JobRuntimeInfoType

  MemAcc job runtime information type. More...
- struct MemAcc_ConfigType

  MemAcc Configuration type. More...

Macros

- #define MEMACC_MODULE_ID

  AUTOSAR module identification.
- #define MEMACC_INSTANCE_ID

  AUTOSAR module instance identification.
- #define MEMACC_E_UNINIT

  Development error codes (passed to DET)
- #define MEMACC_DEINIT_ID

  All service IDs (passed to DET)

Types Reference

- typedef uint16 MemAcc_AddressAreaIdType

  MemAcc Address Area Id Type.
- typedef uint32 MemAcc_AddressType

  MemAcc Address Type.
- typedef uint32 MemAcc_LengthType

  MemAcc Length Type.
- typedef uint8 MemAcc_DataType

  MemAcc Data Type.
- typedef void MemAcc_MemConfigType

  Memory driver configuration structure type.
- typedef uint8 MemAcc_MemDataType

  MemAcc Mem Data Type.
- typedef uint32 MemAcc_MemInstanceIdType

  Memory driver instance ID type.
- typedef MemAcc_LengthType MemAcc_MemLengthType

  Physical memory device length type.
- typedef MemAcc_AddressType MemAcc_MemAddressType

  Physical memory device address type.
- typedef uint32 MemAcc_MemHwServiceIdType

  Index type for Mem driver hardware specific service table.
- typedef uint32 MemAcc_HwIdType

  MemAcc Hardware Id Type.
- typedef void(∗ MemAcc_MemInitFuncType) (MemAcc_MemConfigType ∗ConfigPtr)

  Function pointer for the Mem_Init service for the invocation of the Mem driver API via function pointer interface.
- typedef void(∗ MemAcc_MemDeInitFuncType) (void)

  Function pointer for the Mem_DeInit service for the invocation of the Mem driver API via function pointer interface.
- typedef MemAcc_MemJobResultType(∗ MemAcc_MemGetJobResultFuncType) (MemAcc_MemInstanceIdType InstanceId)

  Function pointer for the Mem_JobResultType service for the invocation of the Mem driver API via function pointer interface.
- typedef void(∗ MemAcc_MemSuspendFuncType) (MemAcc_MemInstanceIdType InstanceId)

  Function pointer for the Mem_Suspend service for the invocation of the Mem driver API via function pointer interface.
- typedef void(∗ MemAcc_MemResumeFuncType) (MemAcc_MemInstanceIdType InstanceId)

  Function pointer for the Mem_Resume service for the invocation of the Mem driver API via function pointer interface.
- typedef void(∗ MemAcc_MemPropagateErrorFuncType) (MemAcc_MemInstanceIdType InstanceId)

  Function pointer for the Mem_PropagateError service for the invocation of the Mem driver API via function pointer interface.
- typedef Std_ReturnType(∗ MemAcc_MemReadFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemAddressType SourceAddress, MemAcc_MemDataType ∗DestinationDataPtr, MemAcc_MemLengthType Length)

  Function pointer for the Mem_Read service for the invocation of the Mem driver API via function pointer interface.
- typedef Std_ReturnType(∗ MemAcc_MemWriteFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemAddressType TargetAddress, const MemAcc_MemDataType ∗SourceDataPtr, MemAcc_MemLengthType Length)

Function pointer for the Mem_Write service for the invocation of the Mem driver API via function pointer interface.

- typedef Std_ReturnType(∗ MemAcc_MemEraseFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemAddressType TargetAddress, MemAcc_MemLengthType Length)

  Function pointer for the Mem_Erase service for the invocation of the Mem driver API via function pointer interface.
- typedef Std_ReturnType(∗ MemAcc_MemBlankCheckFuncType) (MemAcc_MemInstanceIdType Instance↩ Id, MemAcc_MemAddressType TargetAddress, MemAcc_MemLengthType Length)

  Function pointer for the Mem_BlankCheck service for the invocation of the Mem driver API via function pointer interface.
- typedef Std_ReturnType(∗ MemAcc_MemHwSpecificServiceFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemHwServiceIdType HwServiceId, MemAcc_MemDataType ∗DataPtr, MemAcc_MemLengthTyp ∗LengthPtr)

  Function pointer for the Mem_HwSpecificService service for the invocation of the Mem driver API via function pointer interface.
- typedef void(∗ MemAcc_MemMainFuncType) (void)

  Function pointer for the Mem_MainFunction service for the invocation of the Mem driver API via function pointer interface.

Enum Reference

- enum MemAcc_MemJobResultType

  MemAcc mem job result type.
- enum MemAcc_MemInvocationType

  MemAcc_MemInvocationType.
- enum MemAcc_JobResultType

  Asynchronous job result type.
- enum MemAcc_LockStatusType

  Lock address area status type.
- enum MemAcc_JobStatusType

  Asynchronous job status type.
- enum MemAcc_JobType

  Type for asynchronous jobs.
- enum MemAcc_JobStateType

  Internal asynchronous job state transition.

Function Reference

- void MemAcc_Init (const MemAcc_ConfigType ∗ConfigPtr)

  The function initializes MemAcc module.
- void MemAcc_DeInit (void)

  The function de-initializes the MemAcc module.
- void MemAcc_GetVersionInfo (Std_VersionInfoType ∗VersionInfoPtr)

  Return the version information of the Mem module.
- MemAcc_JobResultType MemAcc_GetJobResult (MemAcc_AddressAreaIdType AddressAreaId)

  Get the most recent job result.
- MemAcc_JobStatusType MemAcc_GetJobStatus (MemAcc_AddressAreaIdType AddressAreaId)

  Get the most recent job status.

- Std_ReturnType MemAcc_GetMemoryInfo (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType Address, MemAcc_MemoryInfoType ∗MemoryInfoPtr)

  Get the memory information of the referenced address area.
- MemAcc_LengthType MemAcc_GetProcessedLength (MemAcc_AddressAreaIdType AddressAreaId)

  Get the processed length of data of the referenced address area.
- void MemAcc_GetJobInfo (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_JobInfoType ∗JobInfo↩ Ptr)

  Get the memory information of the referenced address area.
- Std_ReturnType MemAcc_ActivateMem (MemAcc_AddressType HeaderAddress, MemAcc_HwIdType HwId)

  Dynamic activation and initialization of a Mem driver referenced by HwId and HeaderAddress.
- Std_ReturnType MemAcc_DeactivateMem (MemAcc_HwIdType HwId, MemAcc_AddressType Header↩ Address)

  Dynamic deactivation of a Mem driver referenced by HwId and HeaderAddress.
- void MemAcc_Cancel (MemAcc_AddressAreaIdType AddressAreaId)

  Cancel an ongoing job.
- Std_ReturnType MemAcc_Read (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType SourceAddress, MemAcc_DataType ∗DestinationDataPtr, MemAcc_LengthType Length)

  Reads from flash memory.
- Std_ReturnType MemAcc_Write (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType TargetAddress, const MemAcc_DataType ∗SourceDataPtr, MemAcc_LengthType Length)

  Writes to flash memory.
- Std_ReturnType MemAcc_Erase (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType TargetAddress, MemAcc_LengthType Length)

  Erase one or more complete flash sectors.
- Std_ReturnType MemAcc_Compare (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType SourceAddress, const MemAcc_DataType ∗DataPtr, MemAcc_LengthType Length)

  Compares a flash memory area with an application data buffer.
- Std_ReturnType MemAcc_BlankCheck (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType TargetAddress, MemAcc_LengthType Length)

  Verify whether a given memory area has been erased but not (yet) programmed.
- Std_ReturnType MemAcc_HwSpecificService (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_HwIdType HwId, MemAcc_MemHwServiceIdType HwServiceId, MemAcc_DataType ∗DataPtr, MemAcc_LengthType ∗LengthPtr)

  Trigger a hardware specific service.
- Std_ReturnType MemAcc_RequestLock (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType Address, MemAcc_AddressType Length, MemAcc_ApplicationLockNotification LockNotificationFctPtr)

  Request lock of an address area for exclusive access.
- Std_ReturnType MemAcc_ReleaseLock (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType Address, MemAcc_AddressType Length)

  Release access lock of provided address area.

6.1.2 Data Structure Documentation

6.1.2.1 struct MemAcc_MemoryInfoType

MemAcc memory infomation type.

This structure contains information of Mem device characteristics. It can be accessed via the MemAcc_GetMemoryInfo() service.

Definition at line 445 of file MemAcc_Types.h.

6.1.2.2 struct MemAcc_JobInfoType

MemAcc job infomation type.

This structure contains information the current processing state of the MemAcc module.

Definition at line 465 of file MemAcc_Types.h.

6.1.2.3 struct MemAcc_MemApiType

MemAcc_MemApiType.

This structure contains elements for accessing the Mem driver service functions and consistency information.

Definition at line 483 of file MemAcc_Types.h.

6.1.2.4 struct MemAcc_SubAddressAreaType

MemAcc_SubAddressAreaType.

Data structure for a sub address area

Definition at line 517 of file MemAcc_Types.h.

6.1.2.5 struct MemAcc_AddressAreaType

MemAcc_AddressAreaType.

This structure contains configured information for each memory address areas.

Definition at line 540 of file MemAcc_Types.h.

6.1.2.6 struct MemAcc_JobRuntimeInfoType

MemAcc job runtime information type.

This structure contains runtime information the current processing job of each address area.

Definition at line 557 of file MemAcc_Types.h.

6.1.2.7 struct MemAcc_ConfigType

MemAcc Configuration type.

Postbuild configuration structure type

Definition at line 590 of file MemAcc_Types.h.

6.1.3 Macro Definition Documentation

6.1.3.1 MEMACC_MODULE_ID

```
#define MEMACC_MODULE_ID
```

AUTOSAR module identification.

Definition at line 76 of file MemAcc.h.

6.1.3.2 MEMACC_INSTANCE_ID

```
#define MEMACC_INSTANCE_ID
```

AUTOSAR module instance identification.

Definition at line 81 of file MemAcc.h.

6.1.3.3 MEMACC_E_UNINIT

```
#define MEMACC_E_UNINIT
```

Development error codes (passed to DET)

Definition at line 87 of file MemAcc.h.

6.1.3.4 MEMACC_DEINIT_ID

```
#define MEMACC_DEINIT_ID
```

All service IDs (passed to DET)

Definition at line 101 of file MemAcc.h.

6.1.4 Types Reference

6.1.4.1 MemAcc_AddressAreaIdType

```
typedef uint16 MemAcc_AddressAreaIdType
```

MemAcc Address Area Id Type.

Unique address area ID type

Definition at line 103 of file MemAcc_Types.h.

6.1.4.2 MemAcc_AddressType

```
typedef uint32 MemAcc_AddressType
```

MemAcc Address Type.

Logical memory address type (depends on based on MemAcc64BitSupport configuration 32 or 64 bit)

Definition at line 111 of file MemAcc_Types.h.

6.1.4.3 MemAcc_LengthType

```
typedef uint32 MemAcc_LengthType
```

MemAcc Length Type.

Job length type (depends on based on MemAcc64BitSupport configuration 32 or 64 bit)

Definition at line 119 of file MemAcc_Types.h.

6.1.4.4 MemAcc_DataType

```
typedef uint8 MemAcc_DataType
```

MemAcc Data Type.

General data type

Definition at line 127 of file MemAcc_Types.h.

6.1.4.5 MemAcc_MemConfigType

```
typedef void MemAcc_MemConfigType
```

Memory driver configuration structure type.

Memory driver configuration structure type

Definition at line 139 of file MemAcc_Types.h.

6.1.4.6 MemAcc_MemDataType

```
typedef uint8 MemAcc_MemDataType
```

MemAcc Mem Data Type.

General data type

Definition at line 147 of file MemAcc_Types.h.

6.1.4.7 MemAcc_MemInstanceIdType

```
typedef uint32 MemAcc_MemInstanceIdType
```

Memory driver instance ID type.

Memory driver instance ID type

Definition at line 155 of file MemAcc_Types.h.

6.1.4.8 MemAcc_MemLengthType

```
typedef MemAcc_LengthType MemAcc_MemLengthType
```

Physical memory device length type.

Physical memory device length type

Definition at line 163 of file MemAcc_Types.h.

6.1.4.9 MemAcc_MemAddressType

```
typedef MemAcc_AddressType MemAcc_MemAddressType
```

Physical memory device address type.

Derived from MemAcc_AddressType

Definition at line 171 of file MemAcc_Types.h.

6.1.4.10 MemAcc_MemHwServiceIdType

```
typedef uint32 MemAcc_MemHwServiceIdType
```

Index type for Mem driver hardware specific service table.

Index type for Mem driver hardware specific service table

Definition at line 179 of file MemAcc_Types.h.

6.1.4.11 MemAcc_HwIdType

```
typedef uint32 MemAcc_HwIdType
```

MemAcc Hardware Id Type.

The name of each enum parameter is constructed from the Mem module name and the Mem instance name. Type for the unique numeric identifiers of all Mem hardware instances used for hardware specific requests.

Definition at line 308 of file MemAcc_Types.h.

6.1.4.12 MemAcc_MemInitFuncType

```
typedef void(* MemAcc_MemInitFuncType) (MemAcc_MemConfigType *ConfigPtr)
```

Function pointer for the Mem_Init service for the invocation of the Mem driver API via function pointer interface.

Definition at line 318 of file MemAcc_Types.h.

6.1.4.13 MemAcc_MemDeInitFuncType

```
typedef void(* MemAcc_MemDeInitFuncType) (void)
```

Function pointer for the Mem_DeInit service for the invocation of the Mem driver API via function pointer interface.

Definition at line 326 of file MemAcc_Types.h.

6.1.4.14 MemAcc_MemGetJobResultFuncType

```
typedef MemAcc_MemJobResultType(* MemAcc_MemGetJobResultFuncType) (MemAcc_MemInstanceIdType InstanceId)
```

Function pointer for the Mem_JobResultType service for the invocation of the Mem driver API via function pointer interface.

Definition at line 332 of file MemAcc_Types.h.

6.1.4.15 MemAcc_MemSuspendFuncType

```
typedef void(* MemAcc_MemSuspendFuncType) (MemAcc_MemInstanceIdType InstanceId)
```

Function pointer for the Mem_Suspend service for the invocation of the Mem driver API via function pointer interface.

Definition at line 340 of file MemAcc_Types.h.

6.1.4.16 MemAcc_MemResumeFuncType

```
typedef void(* MemAcc_MemResumeFuncType) (MemAcc_MemInstanceIdType InstanceId)
```

Function pointer for the Mem_Resume service for the invocation of the Mem driver API via function pointer interface.

Definition at line 348 of file MemAcc_Types.h.

6.1.4.17 MemAcc_MemPropagateErrorFuncType

```
typedef void(* MemAcc_MemPropagateErrorFuncType) (MemAcc_MemInstanceIdType InstanceId)
```

Function pointer for the Mem_PropagateError service for the invocation of the Mem driver API via function pointer interface.

Definition at line 356 of file MemAcc_Types.h.

6.1.4.18 MemAcc_MemReadFuncType

```
typedef Std_ReturnType(* MemAcc_MemReadFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemAddressType
SourceAddress, MemAcc_MemDataType *DestinationDataPtr, MemAcc_MemLengthType Length)
```

Function pointer for the Mem_Read service for the invocation of the Mem driver API via function pointer interface.

Definition at line 364 of file MemAcc_Types.h.

6.1.4.19 MemAcc_MemWriteFuncType

```
typedef Std_ReturnType(* MemAcc_MemWriteFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemAddressType
TargetAddress, const MemAcc_MemDataType *SourceDataPtr, MemAcc_MemLengthType Length)
```

Function pointer for the Mem_Write service for the invocation of the Mem driver API via function pointer interface.

Definition at line 375 of file MemAcc_Types.h.

6.1.4.20 MemAcc_MemEraseFuncType

```
typedef Std_ReturnType(* MemAcc_MemEraseFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemAddressType
TargetAddress, MemAcc_MemLengthType Length)
```

Function pointer for the Mem_Erase service for the invocation of the Mem driver API via function pointer interface.

Definition at line 386 of file MemAcc_Types.h.

6.1.4.21 MemAcc_MemBlankCheckFuncType

```
typedef Std_ReturnType(* MemAcc_MemBlankCheckFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemAddressType
TargetAddress, MemAcc_MemLengthType Length)
```

Function pointer for the Mem_BlankCheck service for the invocation of the Mem driver API via function pointer interface.

Definition at line 396 of file MemAcc_Types.h.

6.1.4.22 MemAcc_MemHwSpecificServiceFuncType

```
typedef Std_ReturnType(* MemAcc_MemHwSpecificServiceFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemHwServi
HwServiceId, MemAcc_MemDataType *DataPtr, MemAcc_MemLengthType *LengthPtr)
```

Function pointer for the Mem_HwSpecificService service for the invocation of the Mem driver API via function pointer interface.

Definition at line 406 of file MemAcc_Types.h.

6.1.4.23 MemAcc_MemMainFuncType

```
typedef void(* MemAcc_MemMainFuncType) (void)
```

Function pointer for the Mem_MainFunction service for the invocation of the Mem driver API via function pointer interface.

Definition at line 417 of file MemAcc_Types.h.

6.1.5 Enum Reference

6.1.5.1 MemAcc_MemJobResultType

```
enum MemAcc_MemJobResultType
```

MemAcc mem job result type.

Definition at line 186 of file MemAcc_Types.h.

6.1.5.2 MemAcc_MemInvocationType

```
enum MemAcc_MemInvocationType
```

MemAcc_MemInvocationType.

Defines how the Mem driver services are accessed and how the Mem driver is scheduled and activated/initialized.

Enumerator

MEMACC_MEM_DIRECT_STATIC	Mem driver is linked with application. Mem service functions are directly called by MemAcc. Mem_Init is called by EcuM and Mem_MainFunction is triggered by SchM.
MEMACC_MEM_INDIRECT_DYNAMIC	Mem driver is linked as a separate binary and is dynamically activated. MemAcc will use Mem driver header table to invoke Mem service functions. Call of Mem_Init and Mem_MainFunction is handled by MemAcc.
MEMACC_MEM_INDIRECT_STATIC	Mem driver is linked with application. MemAcc will use Mem

Definition at line 203 of file MemAcc_Types.h.

6.1.5.3 MemAcc_JobResultType

`enum MemAcc_JobResultType`

Asynchronous job result type.

Definition at line 238 of file MemAcc_Types.h.

6.1.5.4 MemAcc_LockStatusType

`enum MemAcc_LockStatusType`

Lock address area status type.

Definition at line 251 of file MemAcc_Types.h.

6.1.5.5 MemAcc_JobStatusType

`enum MemAcc_JobStatusType`

Asynchronous job status type.

Definition at line 262 of file MemAcc_Types.h.

6.1.5.6 MemAcc_JobType

`enum MemAcc_JobType`

Type for asynchronous jobs.

Definition at line 272 of file MemAcc_Types.h.

6.1.5.7 MemAcc_JobStateType

```
enum MemAcc_JobStateType
```

Internal asynchronous job state transition.

Definition at line 289 of file MemAcc_Types.h.

6.1.6 Function Reference

6.1.6.1 MemAcc_Init()

```
void MemAcc_Init (
             const MemAcc_ConfigType * ConfigPtr )
```

The function initializes MemAcc module.

6.1.6.2 MemAcc_DeInit()

```
void MemAcc_DeInit (
             void  )
```

The function de-initializes the MemAcc module.

6.1.6.3 MemAcc_GetVersionInfo()

```
void MemAcc_GetVersionInfo (
             Std_VersionInfoType * VersionInfoPtr )
```

Return the version information of the Mem module.

6.1.6.4 MemAcc_GetJobResult()

```
MemAcc_JobResultType MemAcc_GetJobResult (
             MemAcc_AddressAreaIdType AddressAreaId )
```

Get the most recent job result.

6.1.6.5 MemAcc_GetJobStatus()

```
MemAcc_JobStatusType MemAcc_GetJobStatus (
            MemAcc_AddressAreaIdType AddressAreaId )
```

Get the most recent job status.

6.1.6.6 MemAcc_GetMemoryInfo()

```
Std_ReturnType MemAcc_GetMemoryInfo (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType Address,
            MemAcc_MemoryInfoType * MemoryInfoPtr )
```

Get the memory information of the referenced address area.

6.1.6.7 MemAcc_GetProcessedLength()

```
MemAcc_LengthType MemAcc_GetProcessedLength (
            MemAcc_AddressAreaIdType AddressAreaId )
```

Get the processed length of data of the referenced address area.

6.1.6.8 MemAcc_GetJobInfo()

```
void MemAcc_GetJobInfo (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_JobInfoType * JobInfoPtr )
```

Get the memory information of the referenced address area.

6.1.6.9 MemAcc_ActivateMem()

```
Std_ReturnType MemAcc_ActivateMem (
            MemAcc_AddressType HeaderAddress,
            MemAcc_HwIdType HwId )
```

Dynamic activation and initialization of a Mem driver referenced by HwId and HeaderAddress.

6.1.6.10 MemAcc_DeactivateMem()

```
Std_ReturnType MemAcc_DeactivateMem (
            MemAcc_HwIdType HwId,
            MemAcc_AddressType HeaderAddress )
```

Dynamic deactivation of a Mem driver referenced by HwId and HeaderAddress.

6.1.6.11 MemAcc_Cancel()

```
void MemAcc_Cancel (
            MemAcc_AddressAreaIdType AddressAreaId )
```

Cancel an ongoing job.

6.1.6.12 MemAcc_Read()

```
Std_ReturnType MemAcc_Read (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType SourceAddress,
            MemAcc_DataType * DestinationDataPtr,
            MemAcc_LengthType Length )
```

Reads from flash memory.

6.1.6.13 MemAcc_Write()

```
Std_ReturnType MemAcc_Write (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType TargetAddress,
            const MemAcc_DataType * SourceDataPtr,
            MemAcc_LengthType Length )
```

Writes to flash memory.

6.1.6.14 MemAcc_Erase()

```
Std_ReturnType MemAcc_Erase (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType TargetAddress,
            MemAcc_LengthType Length )
```

Erase one or more complete flash sectors.

6.1.6.15 MemAcc_Compare()

```
Std_ReturnType MemAcc_Compare (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType SourceAddress,
            const MemAcc_DataType * DataPtr,
            MemAcc_LengthType Length )
```

Compares a flash memory area with an application data buffer.

6.1.6.16 MemAcc_BlankCheck()

```
Std_ReturnType MemAcc_BlankCheck (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType TargetAddress,
            MemAcc_LengthType Length )
```

Verify whether a given memory area has been erased but not (yet) programmed.

6.1.6.17 MemAcc_HwSpecificService()

```
Std_ReturnType MemAcc_HwSpecificService (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_HwIdType HwId,
            MemAcc_MemHwServiceIdType HwServiceId,
            MemAcc_DataType * DataPtr,
            MemAcc_LengthType * LengthPtr )
```

Trigger a hardware specific service.

6.1.6.18 MemAcc_RequestLock()

```
Std_ReturnType MemAcc_RequestLock (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType Address,
            MemAcc_AddressType Length,
            MemAcc_ApplicationLockNotification LockNotificationFctPtr )
```

Request lock of an address area for exclusive access.

6.1.6.19 MemAcc_ReleaseLock()

```
Std_ReturnType MemAcc_ReleaseLock (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType Address,
            MemAcc_AddressType Length )
```

Release access lock of provided address area.