User Manual

for S32K1_S32M24X MEMACC Driver

Document Number: UM2MEMACC ASRR21-11 Rev0000R2.0.0 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	04.08.2023	NXP RTD Team	S32K1_S32M24X Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 2.0.0

Chapter 2

Introduction

This User Manual describes NXP Semiconductors' AUTOSAR MemAcc Driver for S32K1_S32M24X.

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:

- s32k116_qfn32
- s32k116_lqfp48
- s32k118_lqfp48
- s32k118_lqfp64
- s32k142_lqfp48
- s32k142_lqfp64
- s32k142_lqfp100
- s32k142w_lqfp48

- s32k142w_lqfp64
- s32k144_lqfp48
- s32k144_lqfp64 / MWCT1014S_lqfp64
- s32k144_lqfp100 / MWCT1014S_lqfp100
- s32k144_mapbga100
- s32k144w_lqfp48
- s32k144w_lqfp64
- s32k146_lqfp64
- s32k146_lqfp100 / MWCT1015S_lqfp100
- s32k146_mapbga100 / MWCT1015S_mapbga100
- s32k146_lqfp144
- s32k148_lqfp100
- s32k148_mapbga100 / MWCT1016S_mapbga100
- s32k148_lqfp144
- s32k148_lqfp176
- s32m241_lqfp64
- s32m242_lqfp64
- s32m243_lqfp64
- s32m244_lqfp64

All of the above microcontroller devices are collectively named as S32K1_S32M24X. 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	S32K1xx Series Reference Manual, Rev. 14, 09/2021
		S32M24x Reference Manual, Rev. 2 Draft A, 05/2023
3	Datasheet	S32K1xx Data Sheet, Rev. 14, 08/2021
		S32M2xx Data Sheet, Supports S32M24x and S32M27x, Rev. 3 Draft A, 05/2023
4	Errata	S32K116_0N96V Rev. 22/OCT/2021
		S32K118_0N97V Rev. 22/OCT/2021
		S32K142_0N33V Rev. 22/OCT/2021
		S32K144_0N57U Rev. 22/OCT/2021
		S32K144W_0P64A Rev. 22/OCT/2021
		S32K146_0N73V Rev. 22/OCT/2021
		S32K148_0N20V Rev. 22/OCT/2021
		S32M244_P64A+P73G, Rev. 0

Chapter 3

Driver

3.1 Requirements

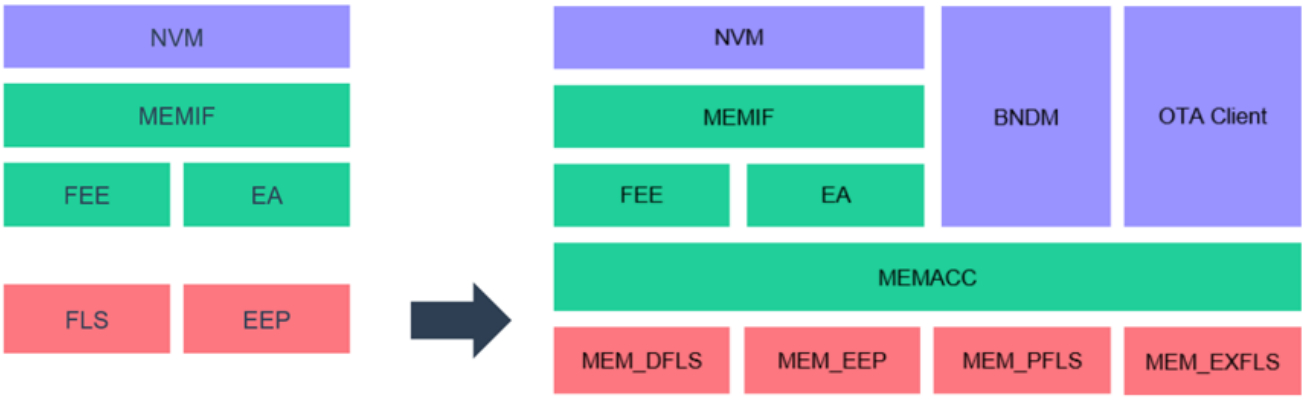
Requirements for this driver are detailed in the Autosar Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

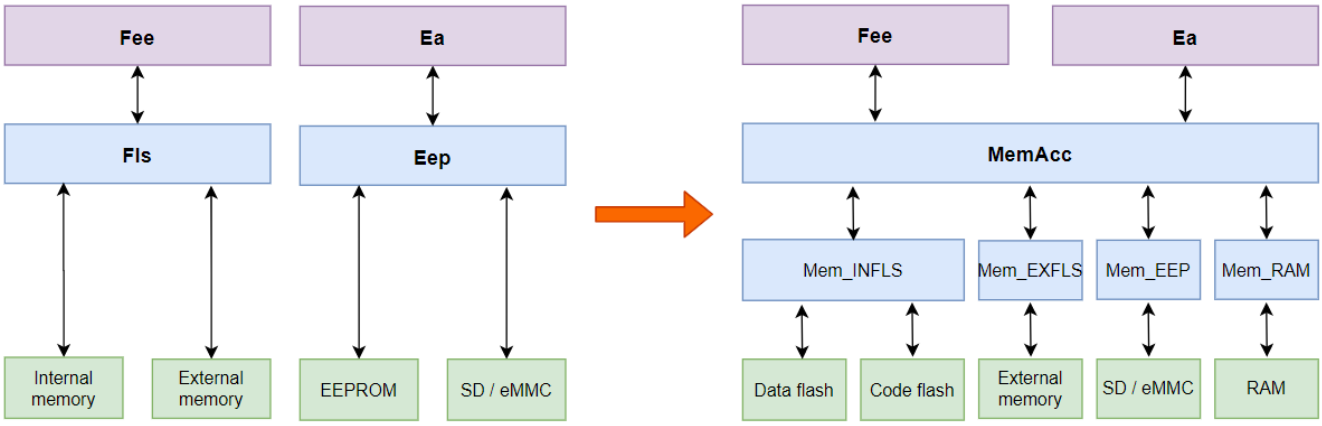
3.2.1 MemAcc stack architecture

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

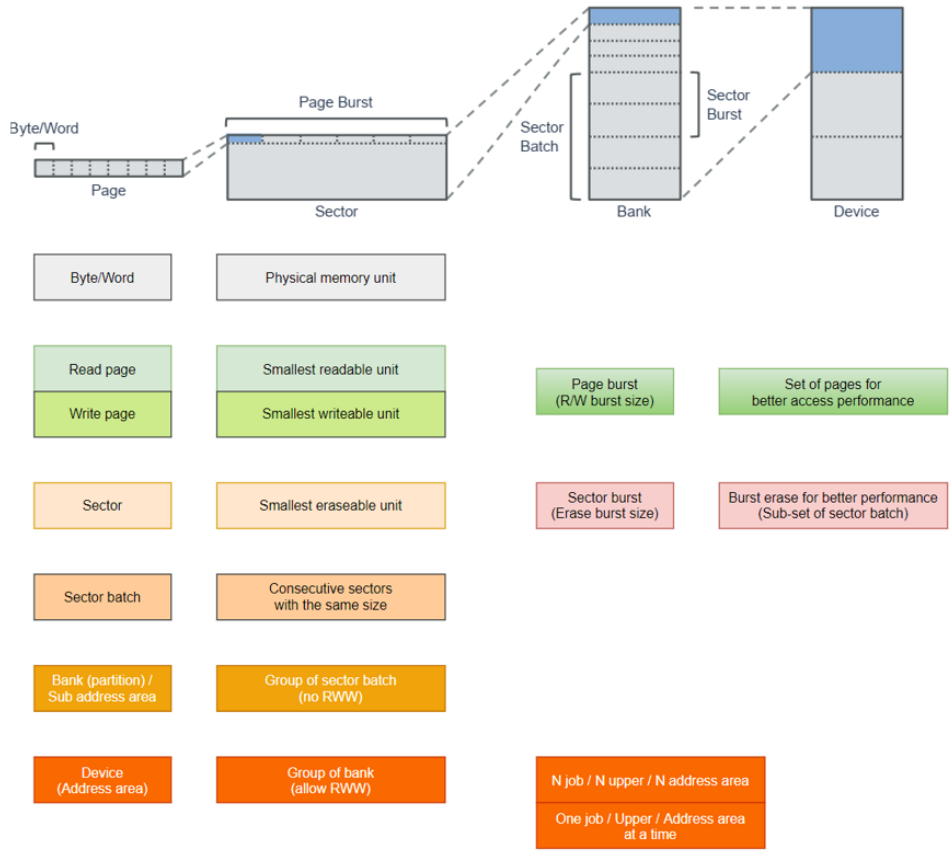


- The Mem driver's task is the low-level memory access based on physical segmentation of the memory device technology
  - Mem drivers provide basic functionalities to the Memory Access Module
  - The API of the Mem driver is memory device technology independent
- All high-level functionality like cross-segment 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**



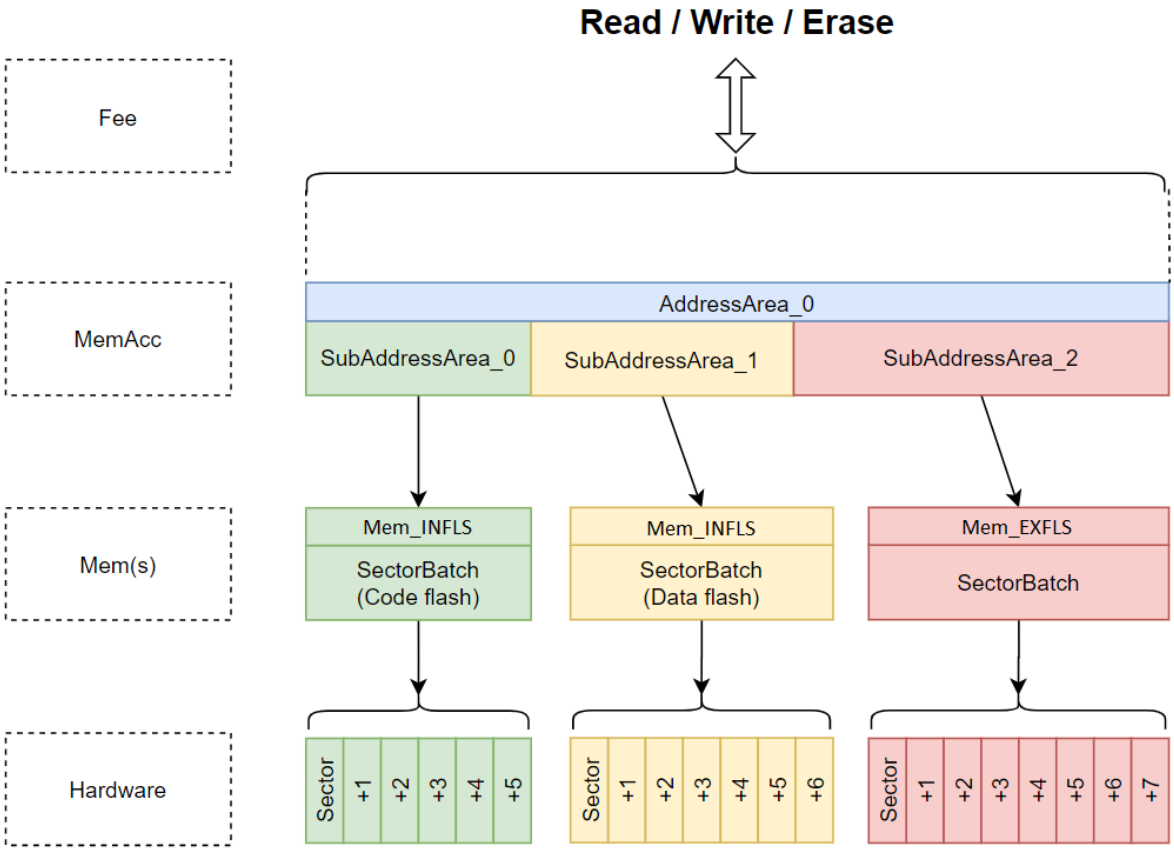
3.2.1.2 Glossaries

- 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)

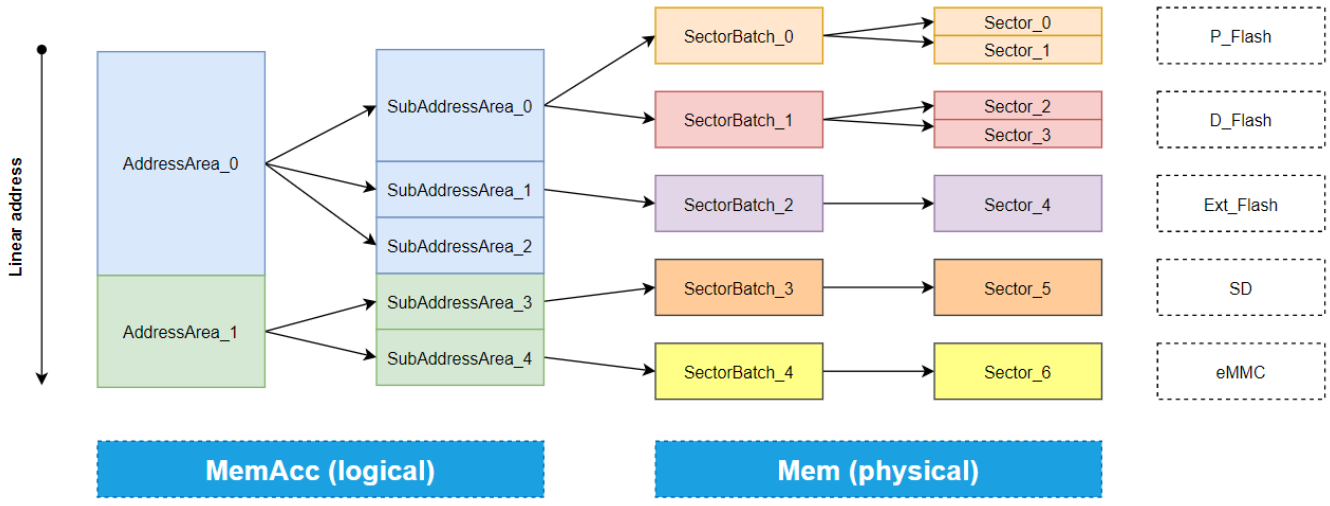


3.2.1.3 Examples of configuration

- 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

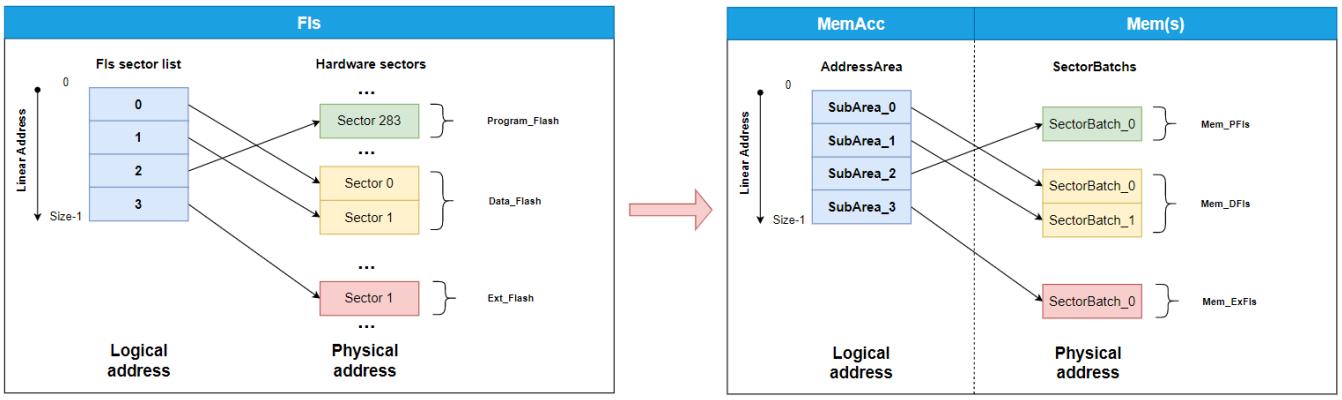


- Example 2: A configuration that uses 5 different types of memory
  - Two address areas for upper layers
  - The sector batches belong to different memory devices: Code flash, Data flash, External flash, SD card, eMMC card



3.2.1.4 Memory architectures comparison

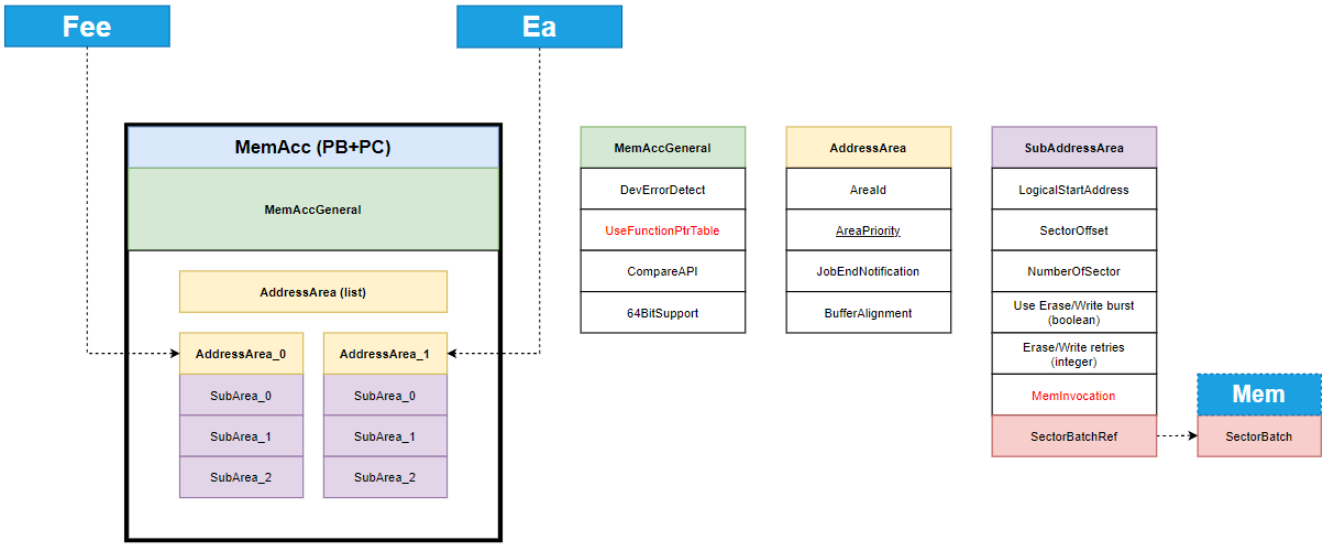
- In conclusion, we have to split the current Fls driver into 2 parts:
  - MemAcc (hardware independent)
  - Mem(s) (hardware dependent)
  - Mem_PFls and Mem_DFls will be combined into a single driver called Mem_INFLS



3.2.2 User interface configuration

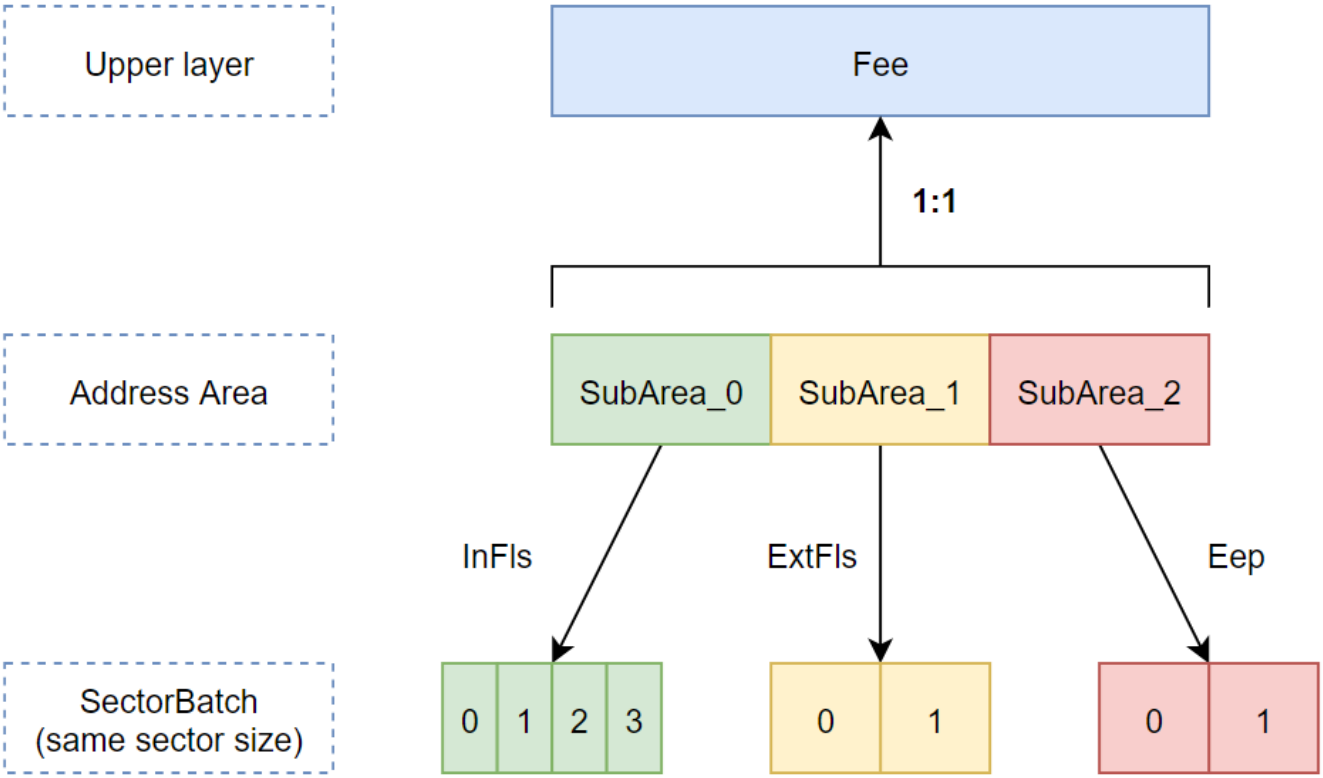
3.2.2.1 Main containers

- 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



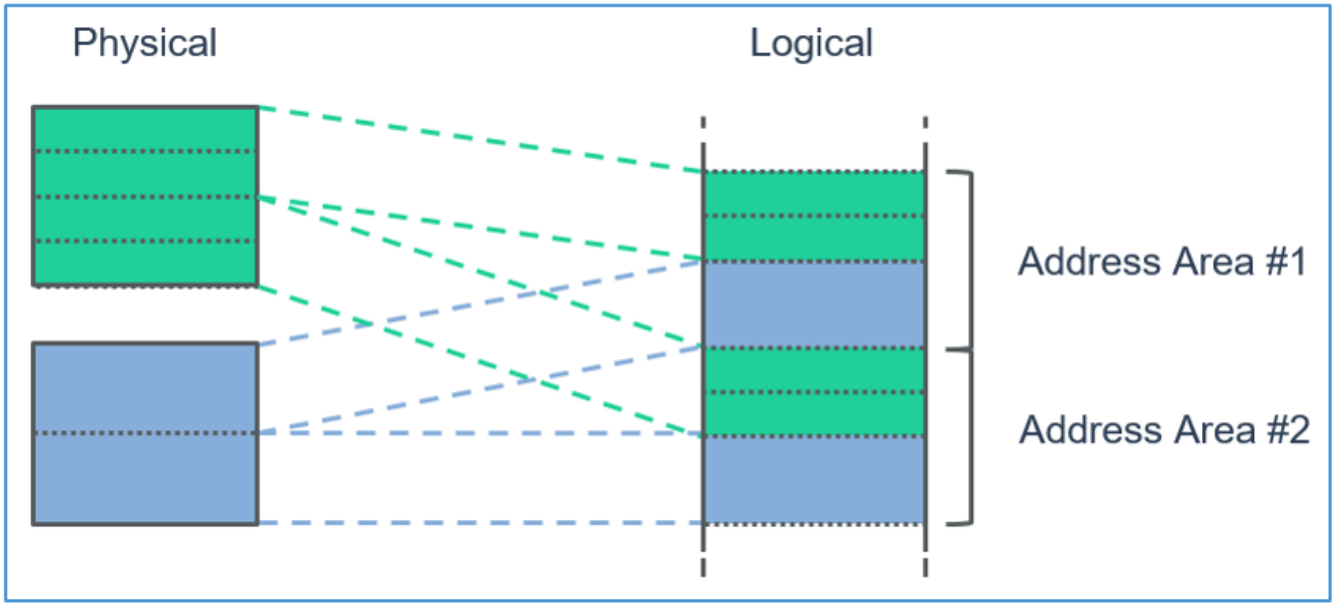
3.2.2.2 Memory address translation

- 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



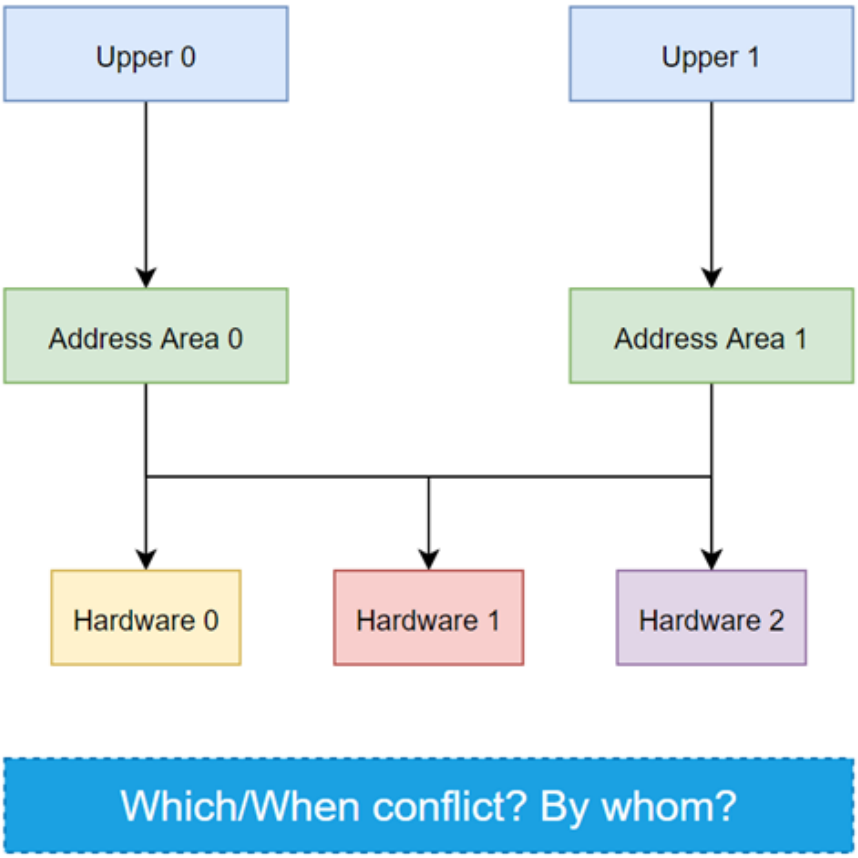
3.2.2.3 Variant support

- 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)



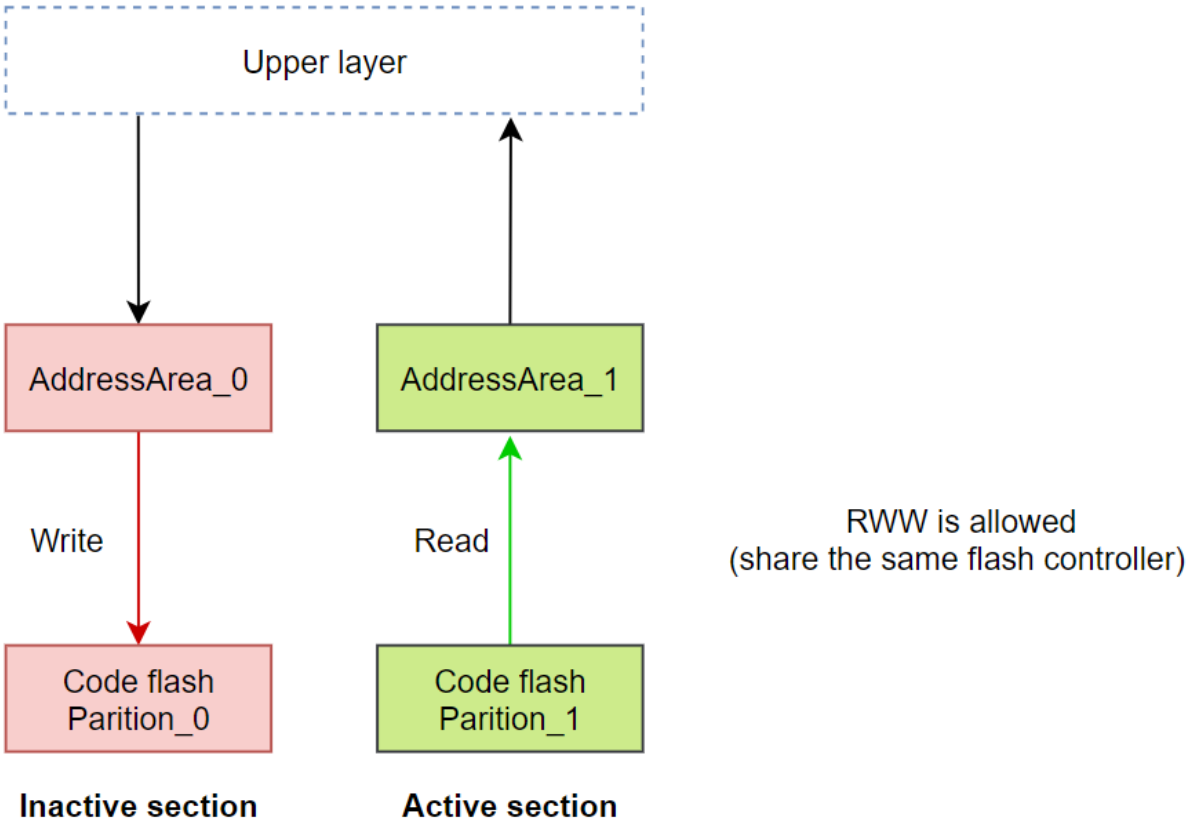
3.2.3 Job management

- 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



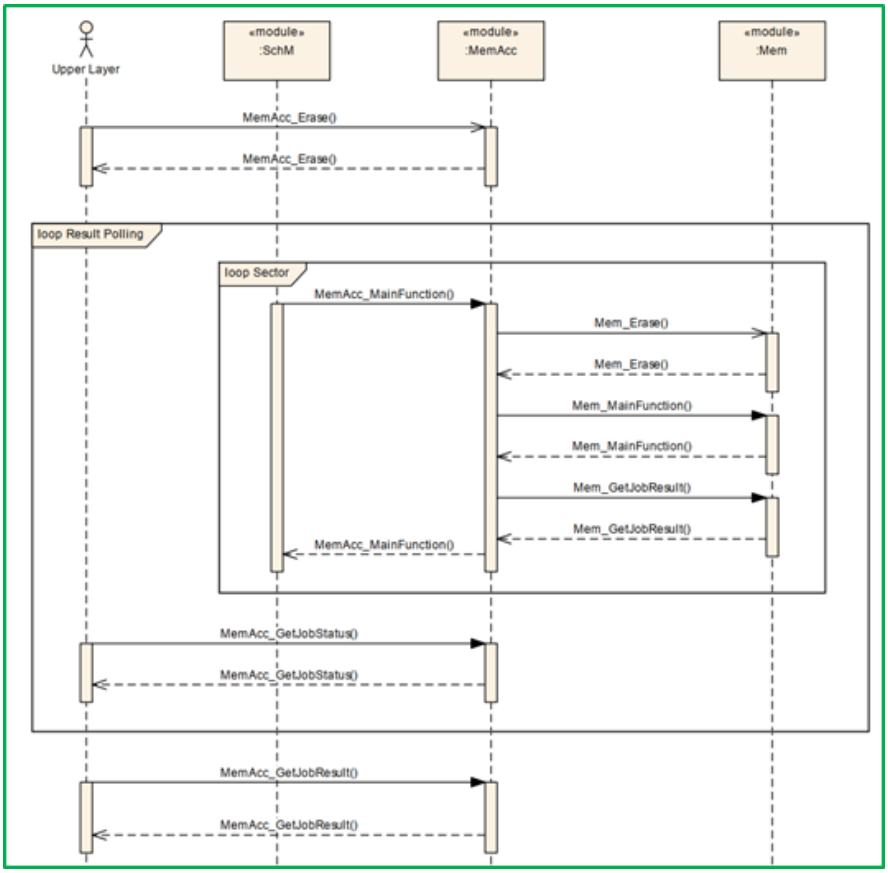
3.2.4 Read-while-write

- 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



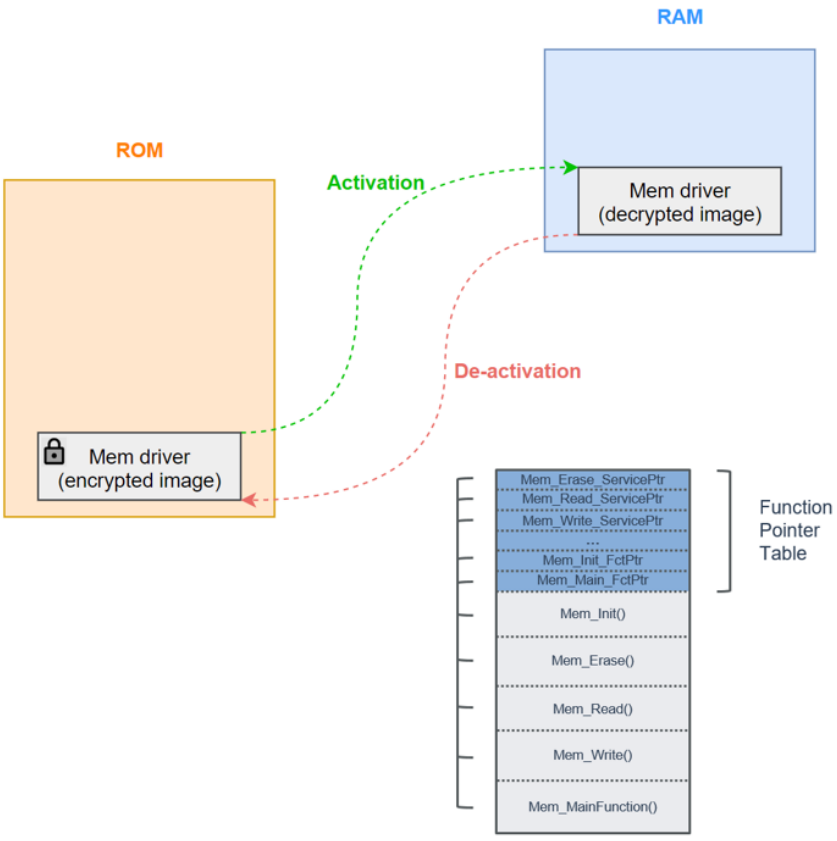
3.2.5 Job processing

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



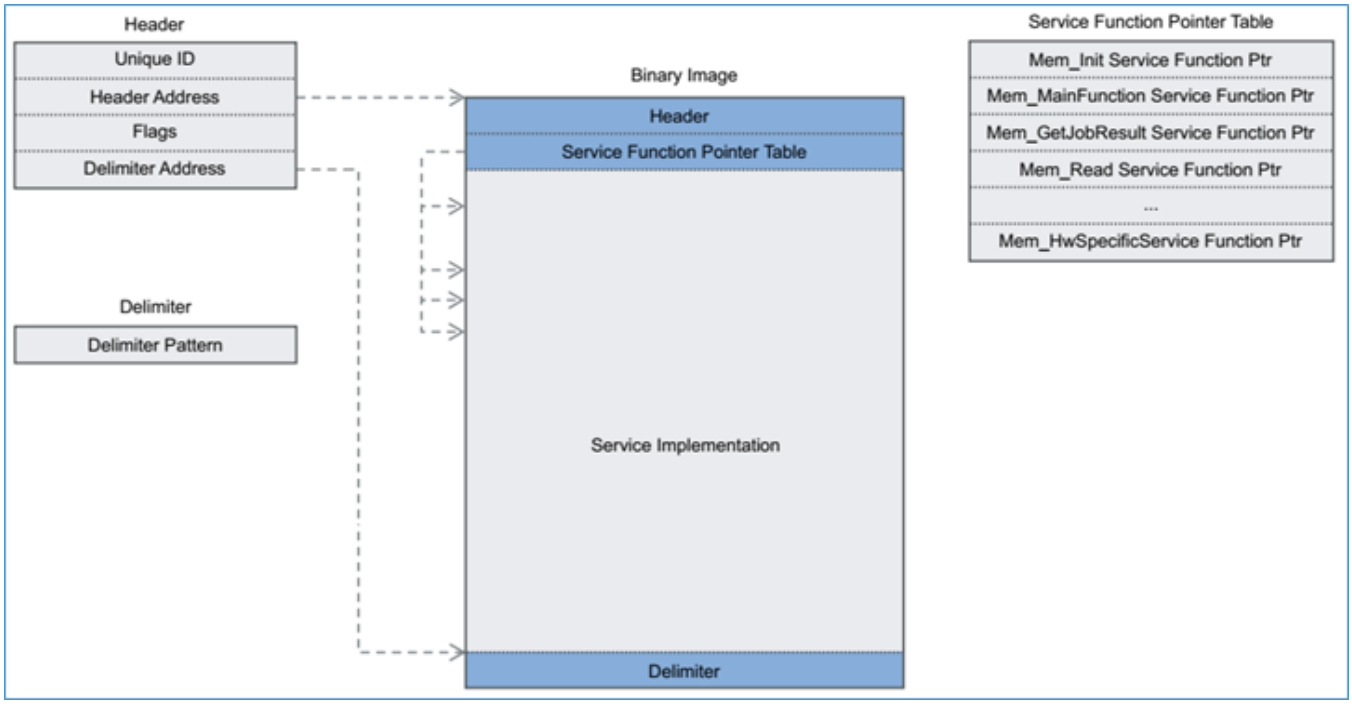
3.2.6 Dynamic Mem driver

- 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



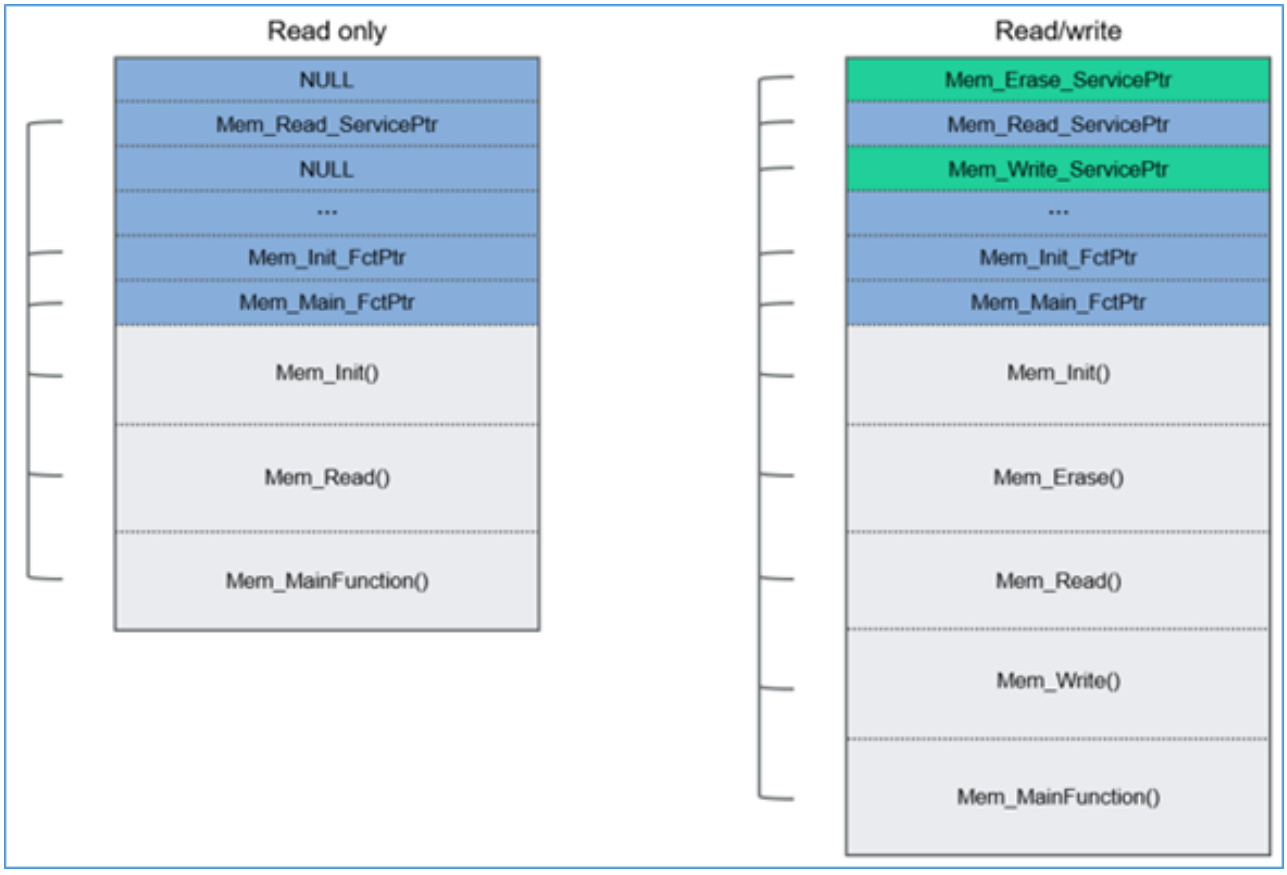
3.2.8 Optional Memory Services

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)

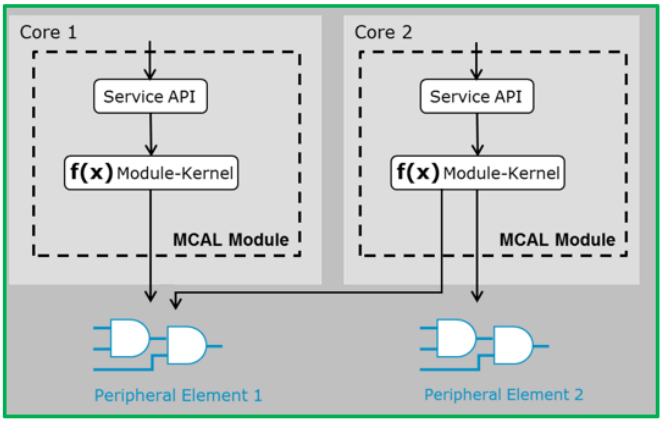
3.2.9.1 Motivation

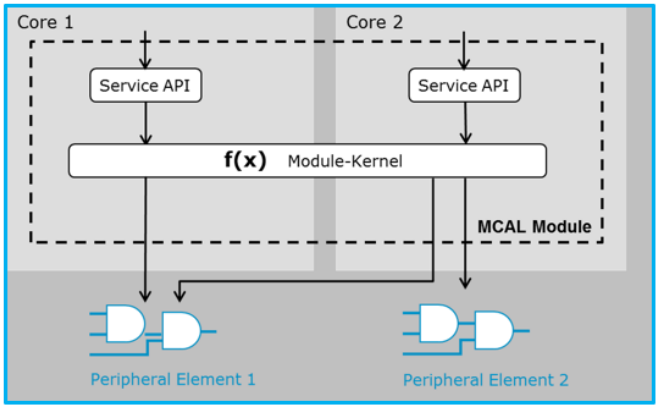
- To support multiple CPU cores share the memory resource
- Typical use case: HSM and host core share the same data flash controller
- A core can access all hardware resource elements (Cross Core Sharing)
- The shared resource initialization can be configurable to be performed by a designated core Only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

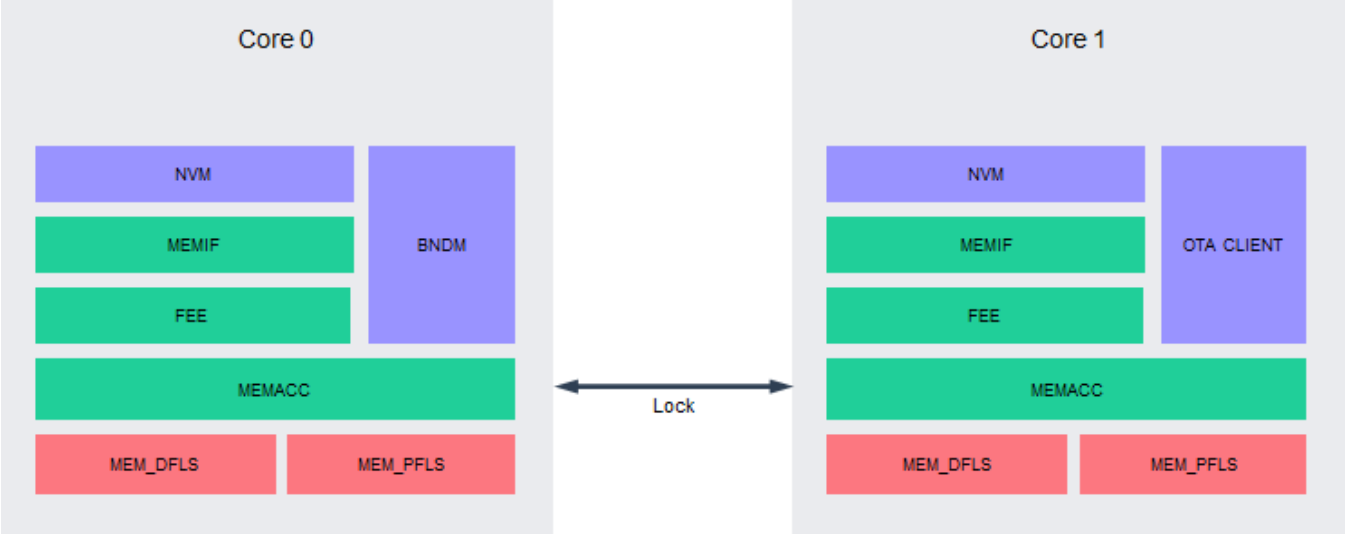
- Can use a shared global variable space for synchronization
- If the HSM core has its own memory space and does not share with other cores, this option cannot be used Another solution is Master-Satellite approach => Multi Core Type 4 -But RTD does not implement this type





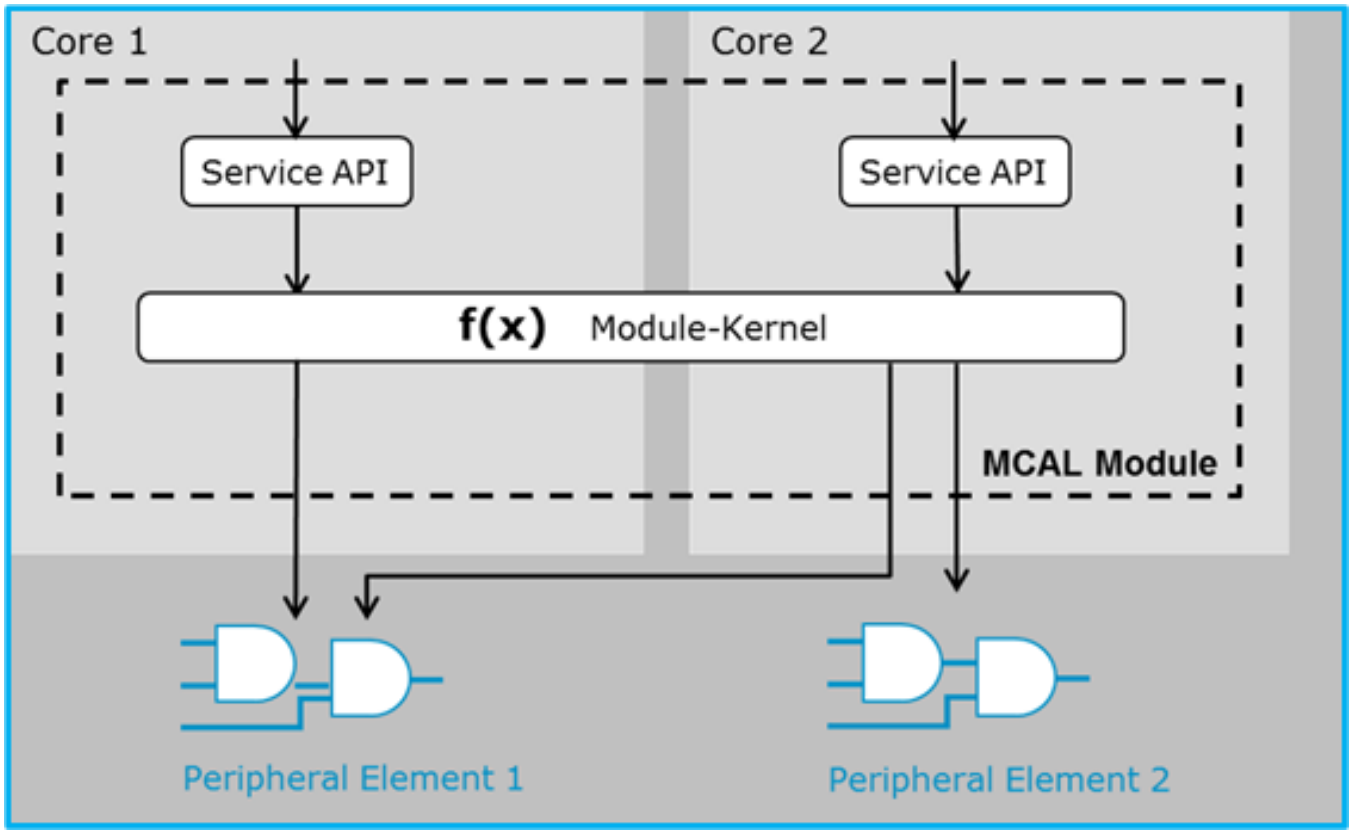
3.2.9.2 Type 1: One Local kernel

- 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



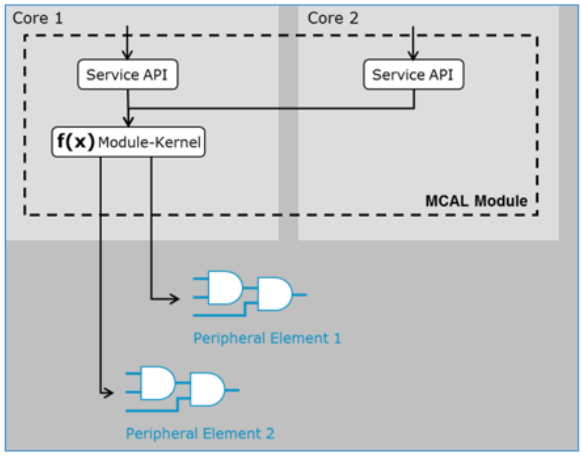
3.2.9.3 Type 3: Global kernel

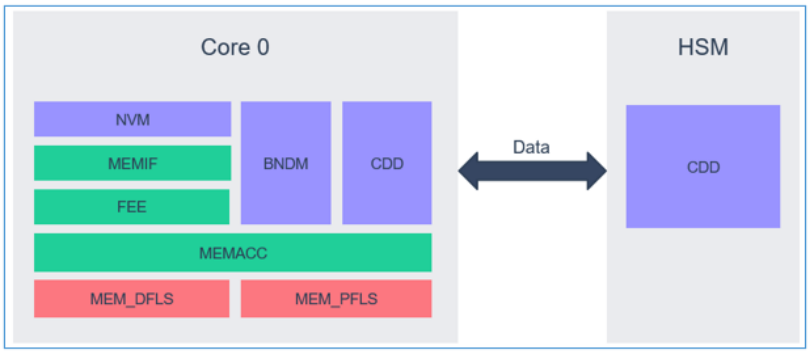
- Need a single global variable space, all global variables are shared by the Service APIs on different cores
  - Placed in non-cacheable section
  - Each core operates on its own set of data which is placed in an array, indexed by the Core/Partition ID



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
None	None	None	None

3.5 Driver Limitations

- The following features are not suported:
  - Multicore.

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

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	4
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-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
destination	/AUTOSAR/EcucDefs/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
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
	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

Property	Value
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
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

Property	Value
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	2
max	2
min	2

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
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

Property	Value
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
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_SectorBatchInfoType

  MemAcc_SectorBatchInfoType. *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...*
- struct MemAcc_MemDriverUniqueIDType

  MemAcc Mem Driver UniqueID Type. *More...*
- struct MemAcc_MemDriverFlagsType

  MemAcc Mem Driver Flags Type. *More...*
- struct MemAcc_MemDriverHeaderType

  MemAcc Mem Driver Header 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). API service called without module initialization
- #define MEMACC_E_PARAM_POINTER

  Development error codes (passed to DET). API service called with NULL pointer argument
- #define MEMACC_E_PARAM_ADDRESS_AREA_ID

  Development error codes (passed to DET). API service called with wrong address area ID
- #define MEMACC_E_PARAM_ADDRESS_LENGTH

  Development error codes (passed to DET). API service called with address and length not belonging to the passed address area ID
- #define MEMACC_E_PARAM_HW_ID

  Development error codes (passed to DET). API service called with a hardware ID not belonging to the passed address area ID
- #define MEMACC_E_BUSY

  Development error codes (passed to DET). API service called for an address area ID with a pending job request
- #define MEMACC_E_MEM_INIT_FAILED

  Development error codes (passed to DET). Dynamic MEM driver activation failed due to inconsistent MEM driver binary
- #define MEMACC_E_OK

  Development error codes (passed to DET). API service called without errors
- #define MEMACC_DEINIT_ID

  Service ID of function MemAcc_DeInit
- #define MEMACC_INIT_ID

  Service ID of function MemAcc_Init
- #define MEMACC_GETVERSIONINFO_ID

  Service ID of function MemAcc_GetVersionInfo
- #define MEMACC_GETJOBRESULT_ID

  Service ID of function MemAcc_GetJobResult
- #define MEMACC_GETJOBSTATUS_ID

  Service ID of function MemAcc_GetJobStatus
- #define MEMACC_GETMEMORYINFO_ID

Service ID of function MemAcc_GetMemoryInfo

- #define MEMACC_GETPROCESSEDLENGTH_ID
  Service ID of function MemAcc_GetProcessedLength
- #define MEMACC_GETJOBINFO_ID
  Service ID of function MemAcc_GetJobInfo
- #define MEMACC_ACTIVATEMEM_ID
  Service ID of function MemAcc_ActivateMem
- #define MEMACC_DEACTIVATEMEM_ID
  Service ID of function MemAcc_DeactivateMem
- #define MEMACC_CANCEL_ID
  Service ID of function MemAcc_Cancel
- #define MEMACC_READ_ID
  Service ID of function MemAcc_Read
- #define MEMACC_WRITE_ID
  Service ID of function MemAcc_Write
- #define MEMACC_ERASE_ID
  Service ID of function MemAcc_Erase
- #define MEMACC_COMPARE_ID
  Service ID of function MemAcc_Compare
- #define MEMACC_BLANKCHECK_ID
  Service ID of function MemAcc_BlankCheck
- #define MEMACC_HWSPECIFICSERVICE_ID
  Service ID of function MemAcc_HwSpecificService
- #define MEMACC_REQUESTLOCK_ID
  Service ID of function MemAcc_RequestLock
- #define MEMACC_RELEASELOCK_ID
  Service ID of function MemAcc_ReleaseLock
- #define MEMACC_MAINFUNCTION_ID
  Service ID of function MemAcc_MainFunction
- #define MEMACC_ADDRESSAREAJOBENDNOTIFICATION_ID
  Service ID of function AddressAreaJobEndNotification
- #define MEMACC_APPLICATIONLOCKNOTIFICATION_ID
  Service ID of function MemAcc_ApplicationLockNotification

Types Reference

- typedef uint16 MemAcc_AddressAreaIdType

  MemAcc Address Area Id Type.
- typedef MEMACC_ADDRESSTYPE MemAcc_AddressType

  MemAcc Address Type.
- typedef MEMACC_LENGTHTYPE 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 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.
- typedef Std_ReturnType(* MemAcc_MemHwSpecificServiceFuncType) (MemAcc_MemInstanceIdType InstanceId, MemAcc_MemHwServiceIdType 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.
- typedef void(* MemAcc_MemMainFuncType) (void)

  Function pointer for the Mem_MainFunction service for the invocation of the Mem driver API via function pointer interface.
- typedef void(* MemAcc_AddressAreaJobEndNotification) (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_JobResultType JobResult)

  MemAcc application job end notification callback.The function name is configurable.
- typedef void(* MemAcc_ApplicationLockNotification) (void)

  Address area lock application callback. The function name is configurable.

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 MemAcc_JobStateType_enumeration.

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)

Service to return the version information of the MemAcc module.

- MemAcc_JobResultType MemAcc_GetJobResult (MemAcc_AddressAreaIdType AddressAreaId)

  Get the most recent job result of the referenced address area.
- MemAcc_JobStatusType MemAcc_GetJobStatus (MemAcc_AddressAreaIdType AddressAreaId)

  Get the most recent job status of the referenced address area.
- 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 job 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 an address area.
- Std_ReturnType MemAcc_Write (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType TargetAddress, const MemAcc_DataType *SourceDataPtr, MemAcc_LengthType Length)

  Writes to an address area.
- Std_ReturnType MemAcc_Erase (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType TargetAddress, MemAcc_LengthType Length)

  Erase one or more complete subaddress area in an address area.
- 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_LengthType Length, MemAcc_ApplicationLockNotification LockNotificationFctPtr)

  Request lock of an address area for exclusive access.
- Std_ReturnType MemAcc_ReleaseLock (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_AddressType Address, MemAcc_LengthType 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 485 of file MemAcc_Types.h.

Data Fields

- MemAcc_AddressType LogicalStartAddress
  Logical start address of sub address area
- MemAcc_AddressType PhysicalStartAddress
  Physical start address of sub address area
- MemAcc_LengthType MaxOffset
  Size of sub address area in bytes -1
- uint32 EraseSectorSize
  Size of a sector in bytes
- uint32 EraseSectorBurstSize
  Size of a sector burst in bytes. Equals SectorSize in case burst is disabled
- uint32 ReadPageSize
  Read size of a page in bytes
- uint32 WritePageSize
  Write size of a page in bytes
- uint32 ReadPageBurstSize
  Size of a read page burst in bytes. Equals ReadPageSize in case burst is disabled
- uint32 WritePageBurstSize
  Size of a page burst in bytes. Equals WritePageSize in case burst is disabled
- MemAcc_HwIdType HwId
  Referenced memory driver hardware identifier

6.1.2.1.1 Field Documentation

6.1.2.1.1.1 LogicalStartAddress `MemAcc_AddressType LogicalStartAddress`

Logical start address of sub address area

Definition at line 487 of file MemAcc_Types.h.

6.1.2.1.1.2 PhysicalStartAddress `MemAcc_AddressType PhysicalStartAddress`

Physical start address of sub address area

Definition at line 488 of file MemAcc_Types.h.

6.1.2.1.1.3 MaxOffset `MemAcc_LengthType MaxOffset`

Size of sub address area in bytes -1

Definition at line 489 of file MemAcc_Types.h.

6.1.2.1.1.4 EraseSectorSize `uint32 EraseSectorSize`

Size of a sector in bytes

Definition at line 490 of file MemAcc_Types.h.

6.1.2.1.1.5 EraseSectorBurstSize `uint32 EraseSectorBurstSize`

Size of a sector burst in bytes. Equals SectorSize in case burst is disabled

Definition at line 491 of file MemAcc_Types.h.

6.1.2.1.1.6 ReadPageSize `uint32 ReadPageSize`

Read size of a page in bytes

Definition at line 492 of file MemAcc_Types.h.

6.1.2.1.1.7 WritePageSize `uint32 WritePageSize`

Write size of a page in bytes

Definition at line 493 of file MemAcc_Types.h.

6.1.2.1.1.8 ReadPageBurstSize `uint32 ReadPageBurstSize`

Size of a read page burst in bytes. Equals ReadPageSize in case burst is disabled

Definition at line 494 of file MemAcc_Types.h.

6.1.2.1.1.9 WritePageBurstSize `uint32 WritePageBurstSize`

Size of a page burst in bytes. Equals WritePageSize in case burst is disabled

Definition at line 495 of file MemAcc_Types.h.

6.1.2.1.1.10 HwId `MemAcc_HwIdType HwId`

Referenced memory driver hardware identifier

Definition at line 496 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 505 of file MemAcc_Types.h.

Data Fields

- MemAcc_AddressType LogicalAddress
  Address of currently active address area request
- MemAcc_LengthType Length
  Length of the currently active address area request
- MemAcc_HwIdType HwId
  Referenced memory driver hardware identifier
- uint32 MemInstanceId
  Instance ID of the current memory request
- uint32 MemAddress
  Physical address of the current memory driver request
- uint32 MemLength
  Length of memory driver request
- MemAcc_JobType CurrentJob
  Currently active MemAcc job
- MemAcc_JobResultType MemResult
  Current or last Mem driver result

6.1.2.2.1 Field Documentation

6.1.2.2.1.1 LogicalAddress `MemAcc_AddressType LogicalAddress`

Address of currently active address area request

Definition at line 507 of file MemAcc_Types.h.

6.1.2.2.1.2 Length `MemAcc_LengthType Length`

Length of the currently active address area request

Definition at line 508 of file MemAcc_Types.h.

6.1.2.2.1.3 HwId MemAcc_HwIdType HwId

Referenced memory driver hardware identifier

Definition at line 509 of file MemAcc_Types.h.

6.1.2.2.1.4 MemInstanceId uint32 MemInstanceId

Instance ID of the current memory request

Definition at line 510 of file MemAcc_Types.h.

6.1.2.2.1.5 MemAddress uint32 MemAddress

Physical address of the current memory driver request

Definition at line 511 of file MemAcc_Types.h.

6.1.2.2.1.6 MemLength uint32 MemLength

Length of memory driver request

Definition at line 512 of file MemAcc_Types.h.

6.1.2.2.1.7 CurrentJob MemAcc_JobType CurrentJob

Currently active MemAcc job

Definition at line 513 of file MemAcc_Types.h.

6.1.2.2.1.8 MemResult MemAcc_JobResultType MemResult

Current or last Mem driver result

Definition at line 514 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 523 of file MemAcc_Types.h.

Data Fields

- uint64 UniqueId
  Unique ID
- uint64 Flags
  Header flags
- MemAcc_AddressType Header
  Address of Mem driver header structure
- MemAcc_AddressType Delimiter
  Address of Mem driver delimiter field
- MemAcc_MemInitFuncType InitFunc
  Mem_Init function pointer
- MemAcc_MemDeInitFuncType DeInitFunc
  Mem_Init function pointer
- MemAcc_MemMainFuncType MainFunc
  Mem_Main function pointer
- MemAcc_MemGetJobResultFuncType GetJobResultFunc
  Mem_GetJobResult function pointer
- MemAcc_MemReadFuncType ReadFunc
  Mem_Read function pointer
- MemAcc_MemWriteFuncType WriteFunc
  Mem_Write function pointer
- MemAcc_MemEraseFuncType EraseFunc
  Mem_Erase function pointer
- MemAcc_MemBlankCheckFuncType BlankCheckFunc
  Mem_BlankCheck function pointer
- MemAcc_MemPropagateErrorFuncType PropagateErrorFunc
  Mem_PropagateError function pointer
- MemAcc_MemSuspendFuncType SuspendFunc

Mem_Suspend function pointer

- MemAcc_MemResumeFuncType ResumeFunc
  Mem_Resume function pointer
- MemAcc_MemHwSpecificServiceFuncType HwSpecificServiceFunc
  Hardware specific service function pointer

6.1.2.3.1 Field Documentation

6.1.2.3.1.1 UniqueId `uint64 UniqueId`

Unique ID

Definition at line 525 of file MemAcc_Types.h.

6.1.2.3.1.2 Flags `uint64 Flags`

Header flags

Definition at line 526 of file MemAcc_Types.h.

6.1.2.3.1.3 Header `MemAcc_AddressType Header`

Address of Mem driver header structure

Definition at line 527 of file MemAcc_Types.h.

6.1.2.3.1.4 Delimiter `MemAcc_AddressType Delimiter`

Address of Mem driver delimiter field

Definition at line 528 of file MemAcc_Types.h.

6.1.2.3.1.5 InitFunc MemAcc_MemInitFuncType InitFunc

Mem_Init function pointer

Definition at line 529 of file MemAcc_Types.h.

6.1.2.3.1.6 DeInitFunc MemAcc_MemDeInitFuncType DeInitFunc

Mem_Init function pointer

Definition at line 530 of file MemAcc_Types.h.

6.1.2.3.1.7 MainFunc MemAcc_MemMainFuncType MainFunc

Mem_Main function pointer

Definition at line 531 of file MemAcc_Types.h.

6.1.2.3.1.8 GetJobResultFunc MemAcc_MemGetJobResultFuncType GetJobResultFunc

Mem_GetJobResult function pointer

Definition at line 532 of file MemAcc_Types.h.

6.1.2.3.1.9 ReadFunc MemAcc_MemReadFuncType ReadFunc

Mem_Read function pointer

Definition at line 533 of file MemAcc_Types.h.

6.1.2.3.1.10 WriteFunc MemAcc_MemWriteFuncType WriteFunc

Mem_Write function pointer

Definition at line 534 of file MemAcc_Types.h.

6.1.2.3.1.11 EraseFunc MemAcc_MemEraseFuncType EraseFunc

Mem_Erase function pointer

Definition at line 535 of file MemAcc_Types.h.

6.1.2.3.1.12 BlankCheckFunc MemAcc_MemBlankCheckFuncType BlankCheckFunc

Mem_BlankCheck function pointer

Definition at line 536 of file MemAcc_Types.h.

6.1.2.3.1.13 PropagateErrorFunc MemAcc_MemPropagateErrorFuncType PropagateErrorFunc

Mem_PropagateError function pointer

Definition at line 537 of file MemAcc_Types.h.

6.1.2.3.1.14 SuspendFunc MemAcc_MemSuspendFuncType SuspendFunc

Mem_Suspend function pointer

Definition at line 538 of file MemAcc_Types.h.

6.1.2.3.1.15 ResumeFunc MemAcc_MemResumeFuncType ResumeFunc

Mem_Resume function pointer

Definition at line 539 of file MemAcc_Types.h.

6.1.2.3.1.16 HwSpecificServiceFunc MemAcc_MemHwSpecificServiceFuncType HwSpecificServiceFunc

Hardware specific service function pointer

Definition at line 540 of file MemAcc_Types.h.

6.1.2.4 struct MemAcc_SectorBatchInfoType

MemAcc_SectorBatchInfoType.

Data structure for a sector batch infomation.

Definition at line 547 of file MemAcc_Types.h.

Data Fields

Type	Name	Description
const uint32	SectorSize	Size of a sector in bytes in this sector batch (smallest erasable unit)
const uint32	ReadPageSize	Size of a read page of this sector in bytes (smallest readable unit)
const uint32	WritePageSize	Size of a write page of this sector in bytes (smallest writeable unit)
const uint32	EraseBurstSize	Size of sector erase burst in bytes (for improved performance)
const uint32	ReadBurstSize	Size of page read burst in bytes (for improved performance)
const uint32	WriteBurstSize	Size of page write/program burst in bytes (for improved performance)

6.1.2.5 struct MemAcc_SubAddressAreaType

MemAcc_SubAddressAreaType.

Data structure for a sub address area

Definition at line 562 of file MemAcc_Types.h.

Data Fields

- MemAcc_AddressType LogicalStartAddress
  Logical start address of sub address area
- MemAcc_AddressType PhysicalStartAddress
  Physical start address of sub address area
- MemAcc_LengthType Length
  Size of sub address area in bytes
- uint8 NumOfEraseRetries
  The number of retries of a failed erase job
- uint8 NumOfWriteRetries
  The number of retries of a failed write job
- uint8 BurstSettings
  Burst settings of Erase/read/write operations
- MemAcc_MemInvocationType MemInvocation
  How the Mem driver is invocated
- const MemAcc_MemApiType * MemApi
  Mem driver service functions and consistency information
- MemAcc_MemInstanceIdType MemInstanceId
  Instance ID of the current memory request
- const MemAcc_SectorBatchInfoType MemSectorBatchInfo
  Sector batch information
- uint8 MemHwResource
  The hardware resource identifier value
- uint8 Hw_Id
  Referenced memory driver hardware identifier

6.1.2.5.1 Field Documentation

6.1.2.5.1.1 LogicalStartAddress `MemAcc_AddressType LogicalStartAddress`

Logical start address of sub address area

Definition at line 564 of file MemAcc_Types.h.

6.1.2.5.1.2 PhysicalStartAddress `MemAcc_AddressType PhysicalStartAddress`

Physical start address of sub address area

Definition at line 565 of file MemAcc_Types.h.

6.1.2.5.1.3 Length `MemAcc_LengthType Length`

Size of sub address area in bytes

Definition at line 566 of file MemAcc_Types.h.

6.1.2.5.1.4 NumOfEraseRetries `uint8 NumOfEraseRetries`

The number of retries of a failed erase job

Definition at line 567 of file MemAcc_Types.h.

6.1.2.5.1.5 NumOfWriteRetries `uint8 NumOfWriteRetries`

The number of retries of a failed write job

Definition at line 568 of file MemAcc_Types.h.

6.1.2.5.1.6 BurstSettings `uint8 BurstSettings`

Burst settings of Erase/read/write operations

Definition at line 569 of file MemAcc_Types.h.

6.1.2.5.1.7 MemInvocation `MemAcc_MemInvocationType MemInvocation`

How the Mem driver is invocated

Definition at line 571 of file MemAcc_Types.h.

6.1.2.5.1.8 MemApi `const MemAcc_MemApiType* MemApi`

Mem driver service functions and consistency information

Definition at line 572 of file MemAcc_Types.h.

6.1.2.5.1.9 MemInstanceId `MemAcc_MemInstanceIdType MemInstanceId`

Instance ID of the current memory request

Definition at line 573 of file MemAcc_Types.h.

6.1.2.5.1.10 MemSectorBatchInfo `const MemAcc_SectorBatchInfoType MemSectorBatchInfo`

Sector batch information

Definition at line 574 of file MemAcc_Types.h.

6.1.2.5.1.11 MemHwResource `uint8 MemHwResource`

The hardware resource identifier value

Definition at line 575 of file MemAcc_Types.h.

6.1.2.5.1.12 Hw_Id `uint8 Hw_Id`

Referenced memory driver hardware identifier

Definition at line 576 of file MemAcc_Types.h.

6.1.2.6 struct MemAcc_AddressAreaType

MemAcc_AddressAreaType.

This structure contains configured information for each memory address areas.

Definition at line 585 of file MemAcc_Types.h.

Data Fields

- MemAcc_AddressAreaIdType AreaId
  Unique numeric identifier of address area
- MemAcc_LengthType Length
  Size of the whole address area in bytes
- uint16 Priority
  The priority of an AddressArea compared to other AddressAreas
- uint8 BufferAlignment
  Buffer alignment value
- MemAcc_AddressAreaJobEndNotification JobEndNotif
  The notification function which is called after MemAcc job completion
- uint16 SubAreaCount
  Number of sub address areas in this memory address area
- const MemAcc_SubAddressAreaType * SubAreas
  Point to first element in array of sub address areas configurations

6.1.2.6.1 Field Documentation

6.1.2.6.1.1 AreaId `MemAcc_AddressAreaIdType AreaId`

Unique numeric identifier of address area

Definition at line 587 of file MemAcc_Types.h.

6.1.2.6.1.2 Length `MemAcc_LengthType Length`

Size of the whole address area in bytes

Definition at line 588 of file MemAcc_Types.h.

6.1.2.6.1.3 Priority `uint16 Priority`

The priority of an AddressArea compared to other AddressAreas

Definition at line 589 of file MemAcc_Types.h.

6.1.2.6.1.4 BufferAlignment `uint8 BufferAlignment`

Buffer alignment value

Definition at line 590 of file MemAcc_Types.h.

6.1.2.6.1.5 JobEndNotif `MemAcc_AddressAreaJobEndNotification JobEndNotif`

The notification function which is called after MemAcc job completion

Definition at line 591 of file MemAcc_Types.h.

6.1.2.6.1.6 SubAreaCount `uint16 SubAreaCount`

Number of sub address areas in this memory address area

Definition at line 592 of file MemAcc_Types.h.

6.1.2.6.1.7 SubAreas `const MemAcc_SubAddressAreaType* SubAreas`

Point to first element in array of sub address areas configurations

Definition at line 593 of file MemAcc_Types.h.

6.1.2.7 struct MemAcc_JobRuntimeInfoType

MemAcc job runtime information type.

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

Definition at line 602 of file MemAcc_Types.h.

Data Fields

- MemAcc_AddressAreaIdType AreaId
  Requested address area id
- MemAcc_JobType JobType
  Type of currently executed job (Erase, Write, or Read,...)
- MemAcc_AddressType LogicAddress
  Last logical address to be processed
- MemAcc_AddressType PhysicAddress
  Last physical address to be processed
- MemAcc_LengthType LengthOrigin
  Requested bytes of data to be processed
- MemAcc_DataType * DataPtr
  Pointer to user data buffer (used in Read, Write, Compare)
- MemAcc_LengthType * LengthPtr
  Size pointer of the data passed by dataPtr (used in HwSpecificService only)
- MemAcc_MemHwServiceIdType MemHwServiceId
  Hardware specific service request identifier for dispatching the request
- uint16 AreaIndex
  The index of address area being processed
- const MemAcc_SubAddressAreaType * SubArea
  The start sub address area of the requested job
- MemAcc_LengthType LengthRemain
  Remaining bytes of data to be processed
- MemAcc_LengthType LengthChunk
  The amount of data in bytes to be processed each chunk
- MemAcc_JobStateType JobState
  The internal transition state
- MemAcc_JobResultType JobResult
  The result of the most recent job
- MemAcc_JobStatusType JobStatus
  The asynchronous job status
- uint8 JobRetries
  The number of retries of a failed erase or write job

- MemAcc_LockStatusType LockStatus
  The lock status of the address area
- boolean JobLocked
  The requested job is locked or not
- MemAcc_AddressType LockAddress
  The start address of locked area
- MemAcc_LengthType LockLength
  The length of locked area
- MemAcc_ApplicationLockNotification LockNotif
  The callback function is called when the lock is complete

6.1.2.7.1 Field Documentation

6.1.2.7.1.1 AreaId MemAcc_AddressAreaIdType AreaId

Requested address area id

Definition at line 605 of file MemAcc_Types.h.

6.1.2.7.1.2 JobType MemAcc_JobType JobType

Type of currently executed job (Erase, Write, or Read,...)

Definition at line 606 of file MemAcc_Types.h.

6.1.2.7.1.3 LogicAddress MemAcc_AddressType LogicAddress

Last logical address to be processed

Definition at line 607 of file MemAcc_Types.h.

6.1.2.7.1.4 PhysicAddress MemAcc_AddressType PhysicAddress

Last physical address to be processed

Definition at line 608 of file MemAcc_Types.h.

6.1.2.7.1.5 LengthOrigin MemAcc_LengthType LengthOrigin

Requested bytes of data to be processed

Definition at line 609 of file MemAcc_Types.h.

6.1.2.7.1.6 DataPtr MemAcc_DataType* DataPtr

Pointer to user data buffer (used in Read, Write, Compare)

Definition at line 610 of file MemAcc_Types.h.

6.1.2.7.1.7 LengthPtr MemAcc_LengthType* LengthPtr

Size pointer of the data passed by dataPtr (used in HwSpecificService only)

Definition at line 611 of file MemAcc_Types.h.

6.1.2.7.1.8 MemHwServiceId MemAcc_MemHwServiceIdType MemHwServiceId

Hardware specific service request identifier for dispatching the request

Definition at line 612 of file MemAcc_Types.h.

6.1.2.7.1.9 AreaIndex `uint16 AreaIndex`

The index of address area being processed

Definition at line 614 of file MemAcc_Types.h.

6.1.2.7.1.10 SubArea `const MemAcc_SubAddressAreaType* SubArea`

The start sub address area of the requested job

Definition at line 615 of file MemAcc_Types.h.

6.1.2.7.1.11 LengthRemain `MemAcc_LengthType LengthRemain`

Remaining bytes of data to be processed

Definition at line 617 of file MemAcc_Types.h.

6.1.2.7.1.12 LengthChunk `MemAcc_LengthType LengthChunk`

The amount of data in bytes to be processed each chunk

Definition at line 618 of file MemAcc_Types.h.

6.1.2.7.1.13 JobState `MemAcc_JobStateType JobState`

The internal transition state

Definition at line 619 of file MemAcc_Types.h.

6.1.2.7.1.14 JobResult MemAcc_JobResultType JobResult

The result of the most recent job

Definition at line 620 of file MemAcc_Types.h.

6.1.2.7.1.15 JobStatus MemAcc_JobStatusType JobStatus

The asynchronous job status

Definition at line 621 of file MemAcc_Types.h.

6.1.2.7.1.16 JobRetries uint8 JobRetries

The number of retries of a failed erase or write job

Definition at line 622 of file MemAcc_Types.h.

6.1.2.7.1.17 LockStatus MemAcc_LockStatusType LockStatus

The lock status of the address area

Definition at line 623 of file MemAcc_Types.h.

6.1.2.7.1.18 JobLocked boolean JobLocked

The requested job is locked or not

Definition at line 624 of file MemAcc_Types.h.

6.1.2.7.1.19 LockAddress MemAcc_AddressType LockAddress

The start address of locked area

Definition at line 625 of file MemAcc_Types.h.

6.1.2.7.1.20 LockLength MemAcc_LengthType LockLength

The length of locked area

Definition at line 626 of file MemAcc_Types.h.

6.1.2.7.1.21 LockNotif MemAcc_ApplicationLockNotification LockNotif

The callback function is called when the lock is complete

Definition at line 627 of file MemAcc_Types.h.

6.1.2.8 struct MemAcc_ConfigType

MemAcc Configuration type.

Postbuild configuration structure type

Definition at line 636 of file MemAcc_Types.h.

Data Fields

- const uint32 AddressAreaCount
  Number of address areas are configured in this configuration set
- const MemAcc_AddressAreaType * AddressAreas
  Point to first element in array of address areas configurations
- MemAcc_JobRuntimeInfoType * JobRuntimeInfo
  Point to first element in array of job runtime infomation
- const uint32 MemApiCount
  Number of Mem drivers are used by MemAcc in this configuration set.
- MemAcc_MemApiType * MemApis
  Mem driver service functions and consistency information
- const MemAcc_MemInvocationType * MemApisInvocation
  How the Mem drivers is invocated

6.1.2.8.1 Field Documentation

6.1.2.8.1.1 AddressAreaCount `const uint32 AddressAreaCount`

Number of address areas are configured in this configuration set

Definition at line 638 of file MemAcc_Types.h.

6.1.2.8.1.2 AddressAreas `const MemAcc_AddressAreaType* AddressAreas`

Point to first element in array of address areas configurations

Definition at line 639 of file MemAcc_Types.h.

6.1.2.8.1.3 JobRuntimeInfo `MemAcc_JobRuntimeInfoType* JobRuntimeInfo`

Point to first element in array of job runtime infomation

Definition at line 640 of file MemAcc_Types.h.

6.1.2.8.1.4 MemApiCount `const uint32 MemApiCount`

Number of Mem drivers are used by MemAcc in this configuration set.

Definition at line 641 of file MemAcc_Types.h.

6.1.2.8.1.5 MemApis `MemAcc_MemApiType* MemApis`

Mem driver service functions and consistency information

Definition at line 642 of file MemAcc_Types.h.

6.1.2.8.1.6 MemApisInvocation `const MemAcc_MemInvocationType* MemApisInvocation`

How the Mem drivers is invocated

Definition at line 643 of file MemAcc_Types.h.

6.1.2.9 struct MemAcc_MemDriverUniqueIDType

MemAcc Mem Driver UniqueID Type.

Mem Driver binary UniqueID structure type

Definition at line 650 of file MemAcc_Types.h.

6.1.2.10 struct MemAcc_MemDriverFlagsType

MemAcc Mem Driver Flags Type.

Mem Driver binary Flags structure type

Definition at line 661 of file MemAcc_Types.h.

6.1.2.11 struct MemAcc_MemDriverHeaderType

MemAcc Mem Driver Header Type.

Mem Driver binary Header structure type

Definition at line 671 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 92 of file MemAcc.h.

6.1.3.2 MEMACC_INSTANCE_ID

```
#define MEMACC_INSTANCE_ID
```

AUTOSAR module instance identification.

Definition at line 97 of file MemAcc.h.

6.1.3.3 MEMACC_E_UNINIT

```
#define MEMACC_E_UNINIT
```

Development error codes (passed to DET). API service called without module initialization

Development error codes (passed to DET)

Definition at line 103 of file MemAcc.h.

6.1.3.4 MEMACC_E_PARAM_POINTER

```
#define MEMACC_E_PARAM_POINTER
```

Development error codes (passed to DET). API service called with NULL pointer argument

Definition at line 104 of file MemAcc.h.

6.1.3.5 MEMACC_E_PARAM_ADDRESS_AREA_ID

```
#define MEMACC_E_PARAM_ADDRESS_AREA_ID
```

Development error codes (passed to DET). API service called with wrong address area ID

Definition at line 105 of file MemAcc.h.

6.1.3.6 MEMACC_E_PARAM_ADDRESS_LENGTH

```
#define MEMACC_E_PARAM_ADDRESS_LENGTH
```

Development error codes (passed to DET). API service called with address and length not belonging to the passed address area ID

Definition at line 106 of file MemAcc.h.

6.1.3.7 MEMACC_E_PARAM_HW_ID

```
#define MEMACC_E_PARAM_HW_ID
```

Development error codes (passed to DET). API service called with a hardware ID not belonging to the passed address area ID

Definition at line 107 of file MemAcc.h.

6.1.3.8 MEMACC_E_BUSY

```
#define MEMACC_E_BUSY
```

Development error codes (passed to DET). API service called for an address area ID with a pending job request

Definition at line 108 of file MemAcc.h.

6.1.3.9 MEMACC_E_MEM_INIT_FAILED

```
#define MEMACC_E_MEM_INIT_FAILED
```

Development error codes (passed to DET). Dynamic MEM driver activation failed due to inconsistent MEM driver binary

Definition at line 109 of file MemAcc.h.

6.1.3.10 MEMACC_E_OK

```
#define MEMACC_E_OK
```

Development error codes (passed to DET). API service called without errors

Definition at line 110 of file MemAcc.h.

6.1.3.11 MEMACC_DEINIT_ID

```
#define MEMACC_DEINIT_ID
```

Service ID of function MemAcc_DeInit

All service IDs (passed to DET)

Definition at line 117 of file MemAcc.h.

6.1.3.12 MEMACC_INIT_ID

```
#define MEMACC_INIT_ID
```

Service ID of function MemAcc_Init

Definition at line 118 of file MemAcc.h.

6.1.3.13 MEMACC_GETVERSIONINFO_ID

```
#define MEMACC_GETVERSIONINFO_ID
```

Service ID of function MemAcc_GetVersionInfo

Definition at line 119 of file MemAcc.h.

6.1.3.14 MEMACC_GETJOBRESULT_ID

```
#define MEMACC_GETJOBRESULT_ID
```

Service ID of function MemAcc_GetJobResult

Definition at line 121 of file MemAcc.h.

6.1.3.15 MEMACC_GETJOBSTATUS_ID

```
#define MEMACC_GETJOBSTATUS_ID
```

Service ID of function MemAcc_GetJobStatus

Definition at line 122 of file MemAcc.h.

6.1.3.16 MEMACC_GETMEMORYINFO_ID

```
#define MEMACC_GETMEMORYINFO_ID
```

Service ID of function MemAcc_GetMemoryInfo

Definition at line 123 of file MemAcc.h.

6.1.3.17 MEMACC_GETPROCESSEDLENGTH_ID

```
#define MEMACC_GETPROCESSEDLENGTH_ID
```

Service ID of function MemAcc_GetProcessedLength

Definition at line 124 of file MemAcc.h.

6.1.3.18 MEMACC_GETJOBINFO_ID

```
#define MEMACC_GETJOBINFO_ID
```

Service ID of function MemAcc_GetJobInfo

Definition at line 125 of file MemAcc.h.

6.1.3.19 MEMACC_ACTIVATEMEM_ID

```
#define MEMACC_ACTIVATEMEM_ID
```

Service ID of function MemAcc_ActivateMem

Definition at line 127 of file MemAcc.h.

6.1.3.20 MEMACC_DEACTIVATEMEM_ID

```
#define MEMACC_DEACTIVATEMEM_ID
```

Service ID of function MemAcc_DeactivateMem

Definition at line 128 of file MemAcc.h.

6.1.3.21 MEMACC_CANCEL_ID

```
#define MEMACC_CANCEL_ID
```

Service ID of function MemAcc_Cancel

Definition at line 132 of file MemAcc.h.

6.1.3.22 MEMACC_READ_ID

```
#define MEMACC_READ_ID
```

Service ID of function MemAcc_Read

Definition at line 133 of file MemAcc.h.

6.1.3.23 MEMACC_WRITE_ID

```
#define MEMACC_WRITE_ID
```

Service ID of function MemAcc_Write

Definition at line 134 of file MemAcc.h.

6.1.3.24 MEMACC_ERASE_ID

```
#define MEMACC_ERASE_ID
```

Service ID of function MemAcc_Erase

Definition at line 135 of file MemAcc.h.

6.1.3.25 MEMACC_COMPARE_ID

```
#define MEMACC_COMPARE_ID
```

Service ID of function MemAcc_Compare

Definition at line 136 of file MemAcc.h.

6.1.3.26 MEMACC_BLANKCHECK_ID

```
#define MEMACC_BLANKCHECK_ID
```

Service ID of function MemAcc_BlankCheck

Definition at line 137 of file MemAcc.h.

6.1.3.27 MEMACC_HWSPECIFICSERVICE_ID

```
#define MEMACC_HWSPECIFICSERVICE_ID
```

Service ID of function MemAcc_HwSpecificService

Definition at line 138 of file MemAcc.h.

6.1.3.28 MEMACC_REQUESTLOCK_ID

```
#define MEMACC_REQUESTLOCK_ID
```

Service ID of function MemAcc_RequestLock

Definition at line 139 of file MemAcc.h.

6.1.3.29 MEMACC_RELEASELOCK_ID

```
#define MEMACC_RELEASELOCK_ID
```

Service ID of function MemAcc_ReleaseLock

Definition at line 140 of file MemAcc.h.

6.1.3.30 MEMACC_MAINFUNCTION_ID

```
#define MEMACC_MAINFUNCTION_ID
```

Service ID of function MemAcc_MainFunction

Definition at line 144 of file MemAcc.h.

6.1.3.31 MEMACC_ADDRESSAREAJOBENDNOTIFICATION_ID

```
#define MEMACC_ADDRESSAREAJOBENDNOTIFICATION_ID
```

Service ID of function AddressAreaJobEndNotification

Definition at line 148 of file MemAcc.h.

6.1.3.32 MEMACC_APPLICATIONLOCKNOTIFICATION_ID

```
#define MEMACC_APPLICATIONLOCKNOTIFICATION_ID
```

Service ID of function MemAcc_ApplicationLockNotification

Definition at line 149 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 129 of file MemAcc_Types.h.

6.1.4.2 MemAcc_AddressType

```
typedef MEMACC_ADDRESSTYPE MemAcc_AddressType
```

MemAcc Address Type.

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

Definition at line 137 of file MemAcc_Types.h.

6.1.4.3 MemAcc_LengthType

```
typedef MEMACC_LENGTHTYPE MemAcc_LengthType
```

MemAcc Length Type.

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

Definition at line 145 of file MemAcc_Types.h.

6.1.4.4 MemAcc_DataType

```
typedef uint8 MemAcc_DataType
```

MemAcc Data Type.

General data type

Definition at line 153 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 165 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 173 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 181 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 189 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 197 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 205 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 331 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 342 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 351 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 358 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 367 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 376 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 385 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 394 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 406 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 418 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 429 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 440 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 451 of file MemAcc_Types.h.

6.1.4.24 MemAcc_AddressAreaJobEndNotification

```
typedef void(* MemAcc_AddressAreaJobEndNotification) (MemAcc_AddressAreaIdType AddressAreaId, MemAcc_JobResultType
JobResult)
```

MemAcc application job end notification callback.The function name is configurable.

Definition at line 462 of file MemAcc_Types.h.

6.1.4.25 MemAcc_ApplicationLockNotification

```
typedef void(* MemAcc_ApplicationLockNotification) (void)
```

Address area lock application callback. The function name is configurable.

Definition at line 472 of file MemAcc_Types.h.

6.1.5 Enum Reference

6.1.5.1 MemAcc_MemJobResultType

```
enum MemAcc_MemJobResultType
```

MemAcc mem job result type.

Enumerator

MEM_JOB_OK	The last job has been finished successfully
MEM_JOB_PENDING	A job is currently being processed
MEM_JOB_FAILED	Job failed for some unspecific reason
MEM_INCONSISTENT	The checked page is not blank
MEM_ECC_UNCORRECTED	Uncorrectable ECC errors occurred during memory access
MEM_ECC_CORRECTED	Correctable ECC errors occurred during memory access

Definition at line 212 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 driver header table to invoke Mem service functions. Call of Mem_Init and Mem_MainFunction is handled by MemAcc.

Definition at line 228 of file MemAcc_Types.h.

6.1.5.3 MemAcc_JobResultType

`enum MemAcc_JobResultType`

Asynchronous job result type.

Enumerator

MEMACC_MEM_OK	The last MemAcc job was finished successfully
MEMACC_MEM_FAILED	The last MemAcc job resulted in an unspecific failure and the job was not completed
MEMACC_MEM_INCONSISTENT	The last MemAcc job didn't meet the expected result, e.g. a blank check operation
MEMACC_MEM_CANCELED	The last MemAcc job was canceled
MEMACC_MEM_ECC_UNCORRECTED	The last memory operation returned an uncorrectable ECC error
MEMACC_MEM_ECC_CORRECTED	The last memory operation returned a correctable ECC error

Definition at line 261 of file MemAcc_Types.h.

6.1.5.4 MemAcc_LockStatusType

`enum MemAcc_LockStatusType`

Lock address area status type.

Enumerator

MEMACC_UNLOCK	The address area is unlock
MEMACC_LOCKING	The address area is locking
MEMACC_LOCKED	The address area is locked

Definition at line 274 of file MemAcc_Types.h.

6.1.5.5 MemAcc_JobStatusType

`enum MemAcc_JobStatusType`

Asynchronous job status type.

Enumerator

MEMACC_JOB_IDLE	Job processing was completed or no job currently pending
MEMACC_JOB_PENDING	A job is currently being processed

Definition at line 285 of file MemAcc_Types.h.

6.1.5.6 MemAcc_JobType

enum MemAcc_JobType

Type for asynchronous jobs.

Enumerator

MEMACC_NO_JOB	No job currently pending
MEMACC_WRITE_JOB	Write job pending
MEMACC_READ_JOB	Read job pending
MEMACC_COMPARE_JOB	Compare job pending
MEMACC_ERASE_JOB	Erase job pending
MEMACC_MEMHWSPECIFIC_JOB	Hardware specific job pending
MEMACC_BLANKCHECK_JOB	Blank check job pending
MEMACC_REQUESTLOCK_JOB	Request lock job pending

Definition at line 295 of file MemAcc_Types.h.

6.1.5.7 MemAcc_JobStateType

enum MemAcc_JobStateType

Internal asynchronous job state transition MemAcc_JobStateType_enumeration.

Enumerator

MEMACC_JOB_STATE_STARTING	The job is pending to be processed
MEMACC_JOB_STATE_PROCESSING	The job is being processed
MEMACC_JOB_STATE_RETRYING	The job is attempting a retry after a failure
MEMACC_JOB_STATE_SUSPENDING	The job is being suspended
MEMACC_JOB_STATE_RESUMING	The job is being resumed
MEMACC_JOB_STATE_CANCELING	The job is being canceled
MEMACC_JOB_STATE_STOP	The job is stop

Definition at line 312 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.

Initialization function - initializes all variables and sets the module state to initialized.

Parameters

in	*ConfigPtr*	Pointer to selected configuration structure.

Returns

None

Precondition

ConfigPtr must not be NULL_PTR and the module status must not be BUSY.

6.1.6.2 MemAcc_DeInit()

```
void MemAcc_DeInit (
            void  )
```

The function de-initializes the MemAcc module.

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.

Returns

None

Precondition

The MemAcc module must be re-initialized before it will accept any new job requests after this service is processed.

6.1.6.3 MemAcc_GetVersionInfo()

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

Service to return the version information of the MemAcc module.

Version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers

Parameters

out	*VersionInfoPtr*	Pointer to where to store the version information of this module.

Returns

None

6.1.6.4 MemAcc_GetJobResult()

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

Get the most recent job result of the referenced address area.

Returns the consolidated job result of the address area referenced by AddressAreaId. If a MemAcc job is still pending, the API returns the result of the last MemAcc job.

Parameters

in	*AddressAreaId*	Numeric identifier of address area

Returns

MemAcc_JobResultType Most recent job result.

6.1.6.5 MemAcc_GetJobStatus()

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

Get the most recent job status of the referenced address area.

Returns the status of the MemAcc job referenced by addressAreaId.

Parameters

in	*AddressAreaId*	Numeric identifier of address area

Returns

MemAcc_JobStatusType 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.

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.

Parameters

in	*AddressAreaId*	Numeric identifier of address area
in	*Address*	Address in logical address space from which corresponding memory device information shall be retrieved.
out	*MemoryInfoPtr*	Destination memory pointer to store the memory device information.

Returns

Std_ReturnType

- E_OK : The requested addressAreaId and address are valid.
- E_NOT_OK : The requested addressAreaId and address are invalid.

6.1.6.7 MemAcc_GetProcessedLength()

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

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

Returns the accumulated number of bytes that have already been processed in the current job.

Parameters

in	*AddressAreaId*	Numeric identifier of address area

Returns

MemAcc_LengthType Processed length of current job (in bytes).

6.1.6.8 MemAcc_GetJobInfo()

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

Get the job information of the referenced address area.

Returns detailed information of the current memory job like memory device ID, job type, job processing state or job result, address area as well as address and length.

Parameters

in	*AddressAreaId*	Numeric identifier of address area
out	*JobInfoPtr*	Structure pointer to return the detailed processing information of the current job.

Returns

None

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.

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

Parameters

in	*HeaderAddress*	Physical start address of Mem driver header structure.
in	*HwId*	Unique numeric memory driver identifier.

Returns

Std_ReturnType

- E_OK Mem driver activation successful.
- E_NOT_OK Mem driver activation failed.

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.

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

Parameters

in	*HwId*	Unique numeric memory driver identifier.
in	*HeaderAddress*	Physical start address of Mem driver header structure.

Returns

Std_ReturnType

- E_OK Mem driver deactivation successful.
- E_NOT_OK Mem driver deactivation failed.

6.1.6.11 MemAcc_Cancel()

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

Cancel an ongoing job.

Triggers a cancel operation of the pending job for the address area referenced by the AddressAreaId. Cancelling affects only jobs in pending state. For any other states, the request will be ignored. Abort a running job synchronously so that directly after returning from this function a new job can be started.

Parameters

in	*AddressAreaId*	Numeric identifier of address area.

Returns

None

Precondition

The module has to be initialized.

6.1.6.12 MemAcc_Read()

```
Std_ReturnType MemAcc_Read (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType SourceAddress,
```

```
            MemAcc_DataType * DestinationDataPtr,
            MemAcc_LengthType Length )
```

Reads from an address area.

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_MEM_OK. If the read operation failed, the result of the job is either MEMACC_MEM_FAILED in case of a general error or MEMACC_MEM_ECC_CORRECTED/MEMACC_MEM_ECC_UNCORRECTED in case of a correctable/uncorrectable ECC error.

Parameters

in	*AddressAreaId*	Numeric identifier of address area.
in	*SourceAddress*	Read address in logical address space.
in	*Length*	Read length in bytes (aligned to read page size).
out	*DestinationDataPtr*	Destination memory pointer to store the read data

Returns

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.

Precondition

The module has to be initialized.

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 an address area.

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_MEM_OK. If there was an issue writing the data, the result is MEMACC_MEM_FAILED.

Parameters

in	*AddressAreaId*	Numeric identifier of address area.
in	*TargetAddress*	Write address in logical address space.
out	*SourceDataPtr*	Source data pointer (aligned to MemAccBufferAlignmentValue).
in	*Length*	Write length in bytes (aligned to page size).

Returns

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.

Precondition

The module has to be initialized.

6.1.6.14 MemAcc_Erase()

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

Erase one or more complete subaddress area in an address area.

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.

Parameters

in	*AddressAreaId*	Numeric identifier of address area.
in	*TargetAddress*	Erase address in logical address space (aligned to sector size).
in	*Length*	Erase length in bytes (aligned to sector size).

Returns

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.

Precondition

The module has to be initialized.

6.1.6.15 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.

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_MEM_OK, otherwise the result is MEMACC_MEM_INCONSISTENT.

Parameters

in	*AddressAreaId*	Numeric identifier of address area.
in	*TargetAddress*	Blank check address in logical address space.
in	*Length*	Blank check length in bytes.

Returns

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.

Precondition

The module has to be initialized.

6.1.6.16 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.

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_MEM_OK. If the hardware specific operation failed, the result of the job is MEMACC_MEM_FAILED.

Parameters

in	*AddressAreaId*	Numeric identifier of address area.
in	*HwId*	Unique numeric memory driver identifier.
in	*HwServiceId*	Array index pointing to the hardware specific service function pointer.
in,out	*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.
in,out	*LengthPtr*	Size pointer of the data passed by dataPtr. Can be NULL_PTR if dataPtr is also NULL_PTR.
out	*None*	

Returns

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.

Precondition

The module has to be initialized.

6.1.6.17 MemAcc_RequestLock()

```
Std_ReturnType MemAcc_RequestLock (
            MemAcc_AddressAreaIdType AddressAreaId,
            MemAcc_AddressType Address,
```

```
            MemAcc_LengthType Length,
            MemAcc_ApplicationLockNotification LockNotificationFctPtr )
```

Request lock of an address area for exclusive access.

Request lock of an address area for exclusive access. Once the lock is granted, the referenced lock notification function is called by MemAcc.

Parameters

in	*AddressAreaId*	Numeric identifier of address area.
in	*Address*	Logical start address of the address area to identify the Mem driver to be locked.
in	*Length*	Length of the address area to identify the Mem driver to be locked.
in	*LockNotificationFctPtr*	Pointer to address area lock notification callback function.

Returns

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. The address area is locked/locking or validation fail.

Precondition

The module has to be initialized.

6.1.6.18 MemAcc_ReleaseLock()

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

Release access lock of provided address area.

Release access lock of provided address area.

Parameters

in	*Address↩ AreaId*	Numeric identifier of address area.
in	*Address*	Logical start address to identify lock area.
in	*Length*	Length to identify lock area.

Returns

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.

Precondition

The module has to be initialized.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support