User Manual

for S32K3 MEM_INFLS Driver

Document Number: UM34MEM_INFLSASR21-11 Rev0000R3.0.0 Rev. 1.0

Chapter 1

Revision History

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

Chapter 2

Introduction

This User Manual describes NXP Semiconductors' AUTOSAR Mem_43_InFls Driver for S32K3XX. AUTOSAR Mem_43_InFls Driver configuration parameters description can be found in the configuration_parameters section. Deviations from the specification are described in the additional_requirements section. AUTOSAR Mem_43_InFls Driver requirements and APIs are described in the Mem_43_InFls Driver Software Specification Document (version 4.7.0) [1] and in the api_reference section.

2.1 Supported Derivatives

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

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100

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

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

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

2.2 Overview

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

AUTOSAR:

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

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

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

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
ASM	Assembler
BSMI	Basic Software Make file Interface
CAN	Controller Area Network
C/CPP	C and C++ Source Code
CS	Chip Select
CTU	Cross Trigger Unit
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DMA	Direct Memory Access
ECU	Electronic Control Unit
FIFO	First In First Out
LSB	Least Signifigant Bit
MCU	Micro Controller Unit
MIDE	Multi Integrated Development Environment
MSB	Most Significant Bit
N/A	Not Applicable
RAM	Random Access Memory
SIU	Systems Integration Unit
SWS	Software Specification
VLE	Variable Length Encoding
XML	Extensible Markup Language

2.5 Reference List

#	Title	Version
1	Specification of Mem Driver	AUTOSAR Release R21-11
2	Reference Manual	S32K3xx Reference Manual, Rev.6, Draft B, 01/2023
		S32K39 and S32K37 Reference Manual, Rev. 2 Draft A, 11/2022
		S32M27x Reference Manual, Rev.2, Draft A, — 02/2023
3	Datasheet	S32K3xx Data Sheet, Rev. 6, 11/2022
		S32K396 Data Sheet, Rev. 1.1 — 08/2022
		S32M2xx Data Sheet, Rev. 2 RC — 12/2022
4	Errata	S32K358_0P14E Mask Set Errata — Rev. 28, 9/2022
		S32K396_0P40E Mask Set Errata, Rev. DEC2022, 12/2022
		S32K311_0P98C Mask Set Errata, Rev. 6/March/2023, 3/2023
		S32K312: Mask Set Errata for Mask 0P09C, Rev. 25/April/2022
		S32K342: Mask Set Errata for Mask 0P97C, Rev. 10, 11/2022
		S32K3x4: Mask Set Errata for Mask 0P55A/1P55A, Rev. 14/Oct/2022

Chapter 3

Driver

3.1 Requirements

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

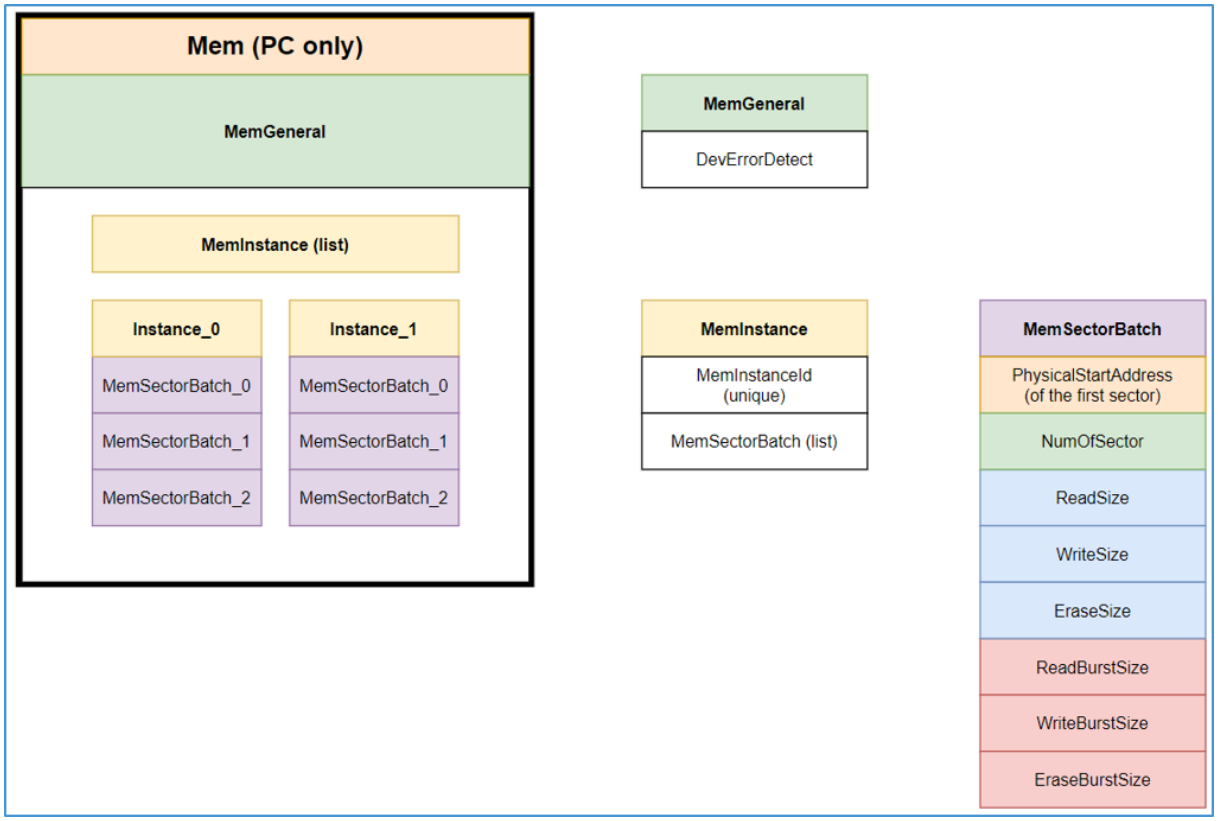
It has vendor-specific requirements and implementation.

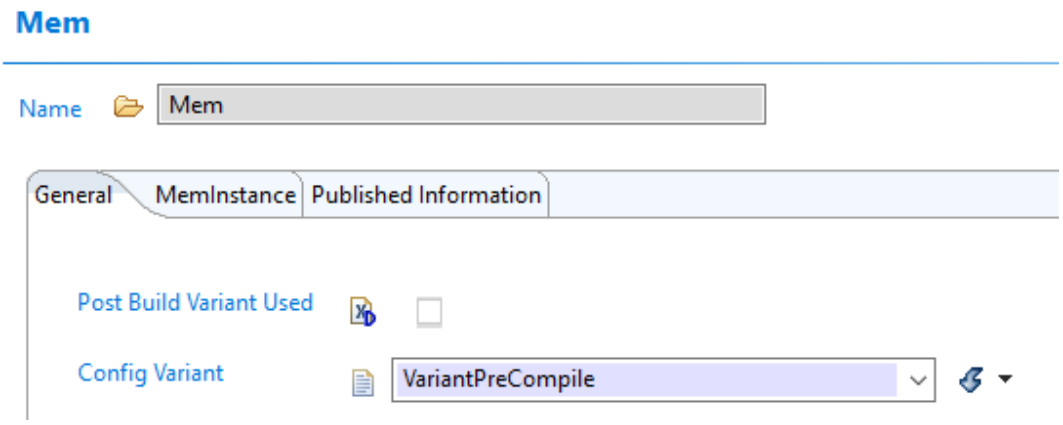
3.2 Driver Design Summary

3.2.1 Mem_43_InFls driver architecture

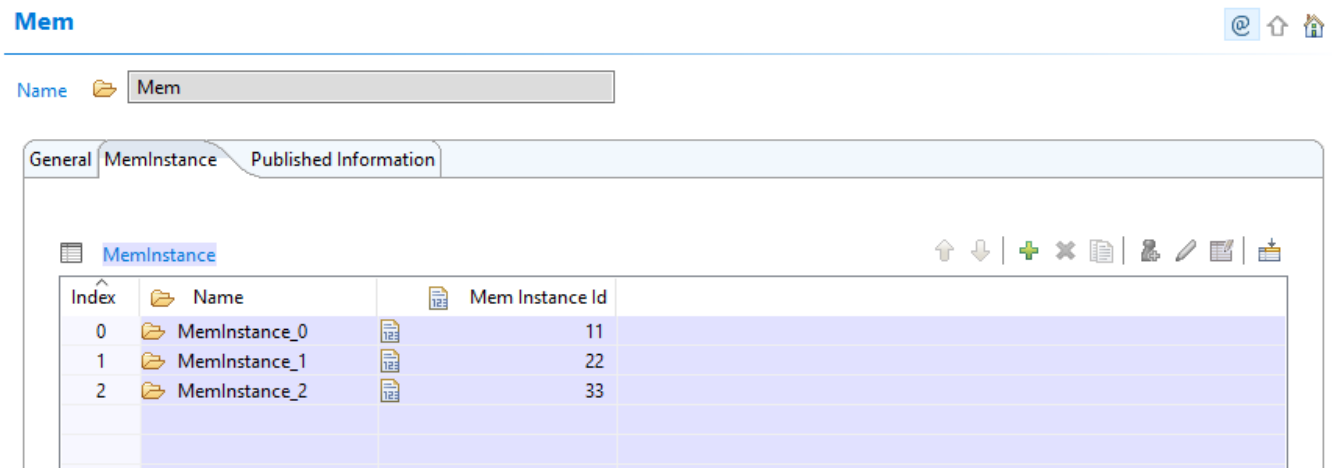
This section describes the detail for a high-level overview of Mem_43_InFls driver.

- The Mem drivers only support **VARIANT-PRE-COMPILE** (Pre-compile module)
- There are 2 main containers:
  - **MemGeneral**: pre-compile configuration parameters
  - **MemInstance**: includes the Mem driver instances specific configuration parameters





- **Multi-instance**: Mem_43_InFls driver supports multiple instances of the same memory type
  - Multiple devices of the same type can be handled by one Mem driver
  - Each device uses a (different) hardware configuration
  - Distinguished by a memory instance ID

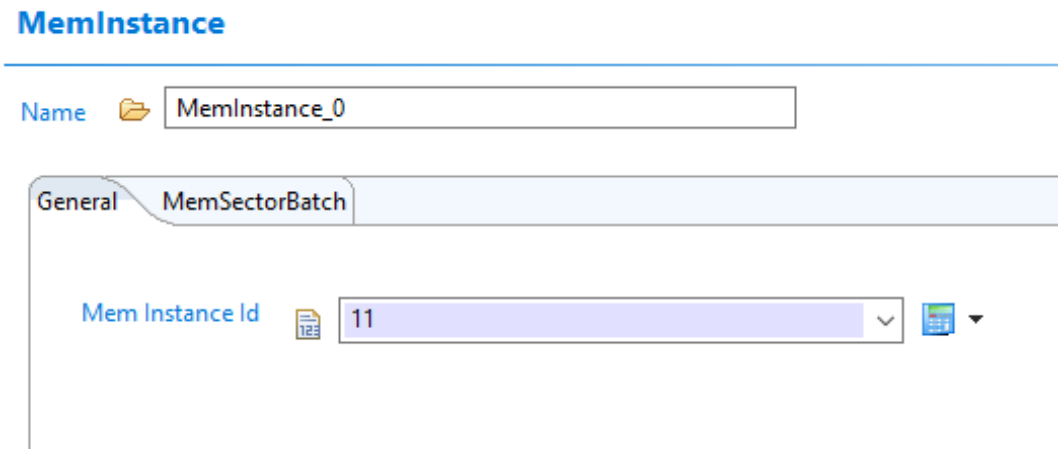


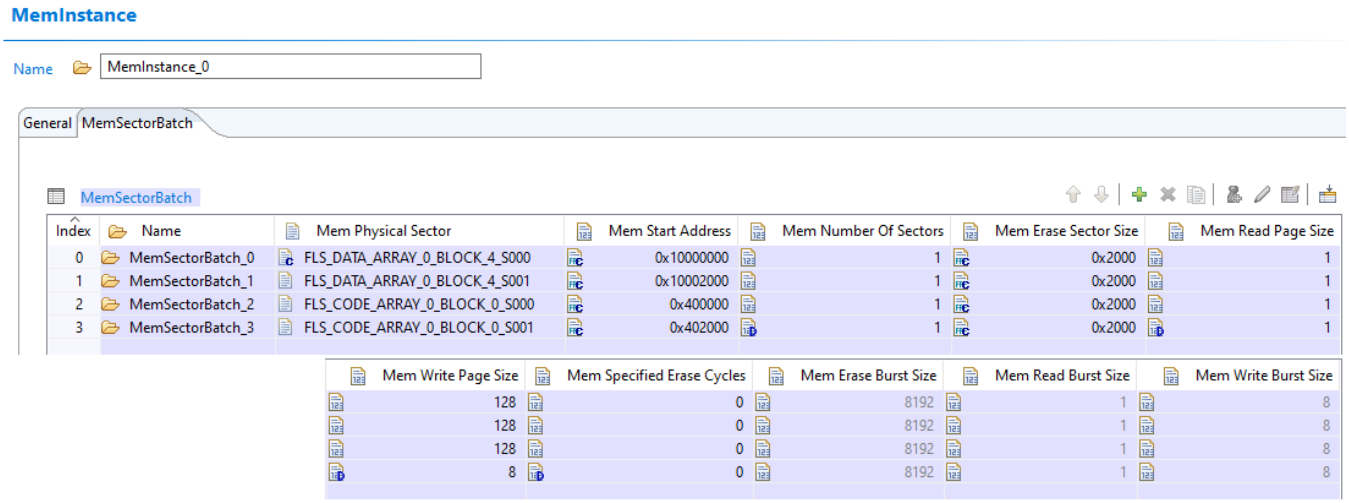
- For example: the OTA software update use case
  - Multiple memory devices of the same type are used to expand the memory resources(increase storage capacity)
  - Switching of memory address translation tables for memory swap use cases(while keep the same physical addresses)

3.2.1.1 MemInstance

This container includes the Mem driver instance specific configuration parameters:

- **MemInstanceID**: is passed as a parameter to select the corresponding driver instance
  - Mapping different instances to different physical memory areas
- **MemSectorBatch**: Configuration description of a programmable sector or sector batch.
  - It is recommended to group as many identical sectors as possible together
  - Aggregation of sectors with uniform size
  - Logical aggregation of contiguous sectors with the same size





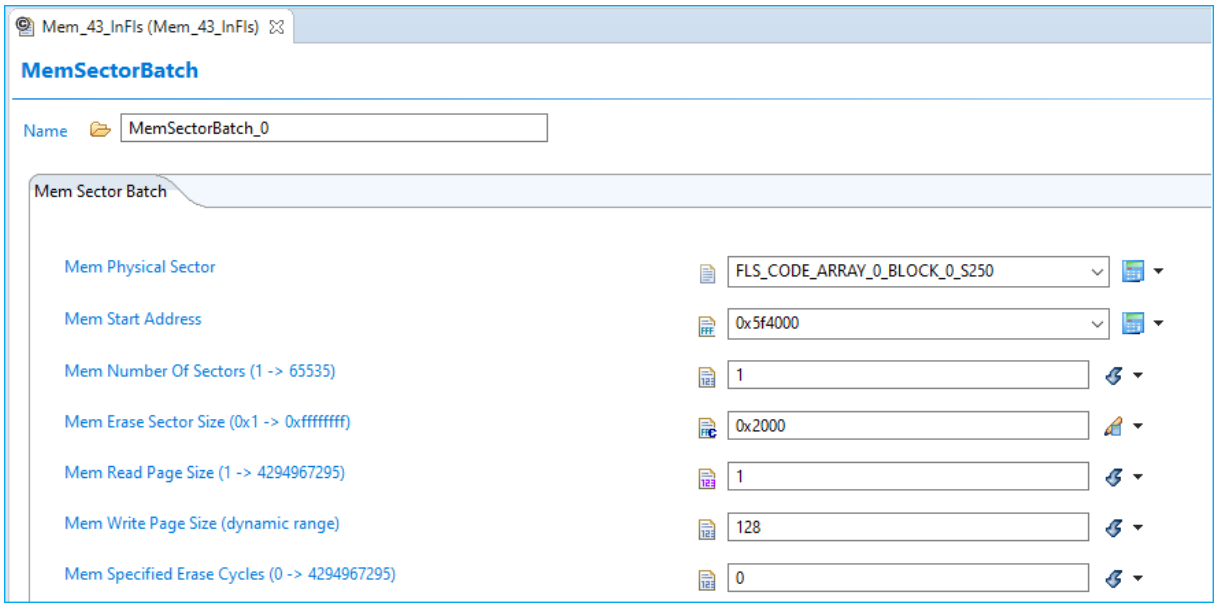
As you can see:

- Mem Start Address for MemSectorBatch_0 will be 0x10000000 and Mem Start Address for MemSectorBatch↩ _1 will be 0x10002000
- If users want to write MemSectorBatch_1, users need to write to the physical address 0x10002000 - 0x10003FFF
- If users want to erase MemSectorBatch_1, users need to erase sector from address 0x10002000 with size 0x2000

Note

The user do not need to calculate the Mem Start Address, it can be automatically computed.

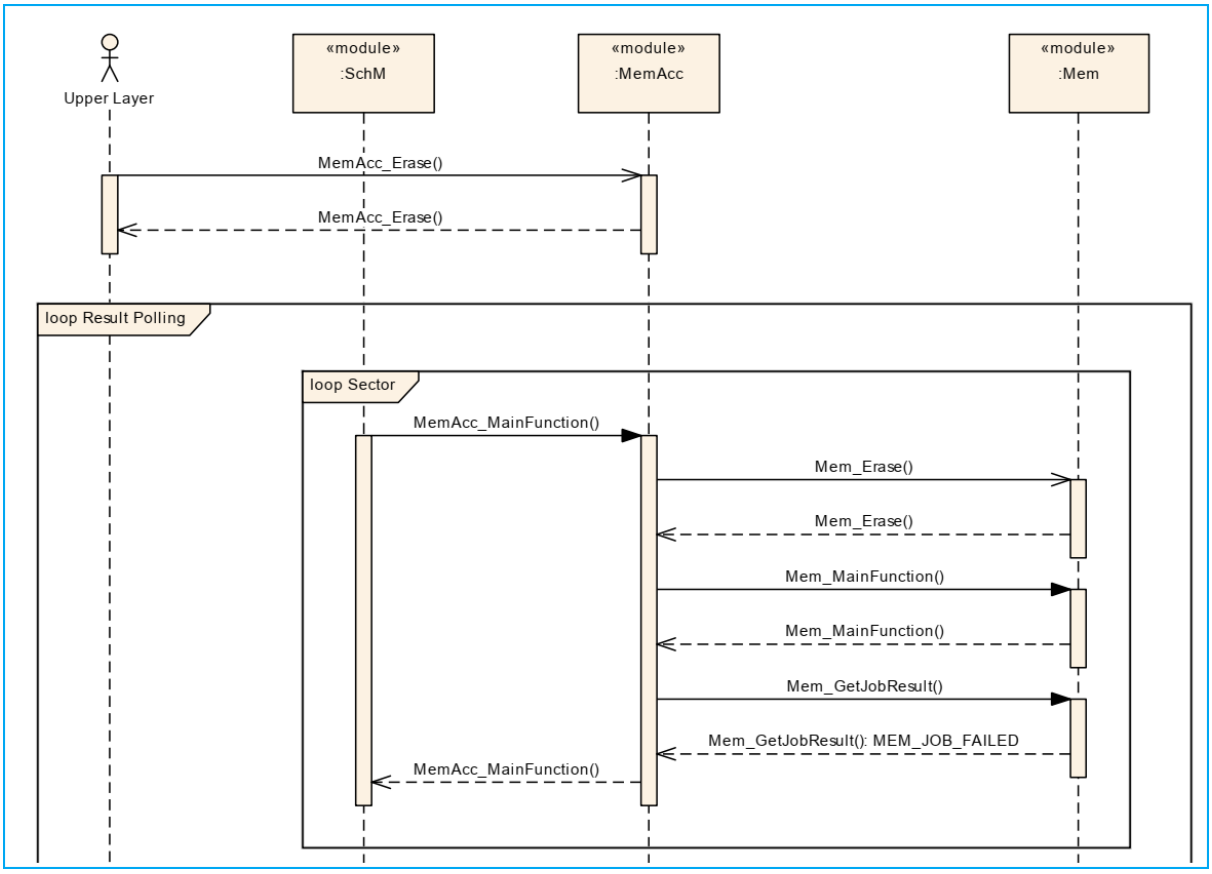
3.2.1.2 MemSectorBatch



- Mem Physical Sector: Physical flash device sector
- Mem Start Address: Physical start address of the sector (batch)
- Mem Number Of Sectors: Number of contiguous sectors with identical values for MemSectorSize and Mem↩ PageSize
- Mem Erase Sector Size: Size of this sector in bytes.
- Mem Read Page Size: Size of a read page of this sector in bytes.
- Mem Write Page Size: Size of a write page of this sector in bytes.
- Mem Specified Erase: Number of erase cycles specified for the memory device(usually given in the device data sheet)

3.2.2 Job management

- Mem drivers support basic services for reading, writing, and erasing of memory devices based on the physical segmentation
- Handle only one job at a time
- Keep track of the job proccesing and store result of the last job
- Do not need to measure times or do any timing supervision
- Mem drivers operates on flash pages and sectors
- Management of erasing multiple sectors or writing multiple pages is handled by MemAcc
  - Splitting of access request according to physical memory segmentation



Note

If Suspend/Resume are not supported by hardware, implementation based on physical segmentation.

3.3 Hardware Resources

3.3.1 Flash Banks/Arrays, Sectors details

- List of S32K3xx derivatives and their flash configuration:

Derivatives	PFlash [KB]	DFlash [KB]	UTest [KB]	SectorSize [KB]
S32K311	1024	64	8	8
S32K341		128	8	8
S32K312	2048	128	8	8
S32K322		128	8	8
S32K342		128	8	8
S32K314	4096	128	8	8
S32K324		128	8	8
S32K344		128	8	8
S32K396	6144	128	8	8
S32K358	8192	128	8	8
S32M276	1024	64	8	8

- For S32K3x2: has 2 MBytes of code flash (program flash), 128 KBytes of data flash and 8 Kbytes of Utest NVM
  - 2M P Flash (each sector is 8K so 2M/8K = 256 sectors)
  - 128K D Flash (each sector is 8K so 128K/8K = 16 sectors)
  - 8K U Flash (each sector is 8K so 8K/8K = 1 sectors)
  - There are 3 blocks (read partitions):
    - P Flash: Block code flash 0 (1M)
    - P Flash: Block code flash 1 (1M)
    - D Flash: Block data flash (128K)
    - U Flash: Block Utest (8K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_2_S000	8
...	...
FLS_DATA_ARRAY_0_BLOCK_2_S015	8
FLS_CODE_ARRAY_0_BLOCK_0_S000	8
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	8
FLS_CODE_ARRAY_0_BLOCK_1_S128	8
...	...
FLS_CODE_ARRAY_0_BLOCK_1_S255	8
FLS_UTEST_ARRAY_0_S000	8

- For S32K3x4: has 4 MBytes of code flash (program flash), 128 KBytes of data flash and 8 Kbytes of Utest NVM
  - 4M P Flash (each sector is 8K so 4M/8K = 512 sectors)
  - 128K D flash (each sector is 8K so 128K/8K = 16 sectors)
  - 8K U Flash (each sector is 8K so 8K/8K = 1 sectors)
  - There are 5 blocks (read partitions):
    * P Flash: Block code flash 0 (1M)
    * P Flash: Block code flash 1 (1M)
    * P Flash: Block code flash 2 (1M)
    * P Flash: Block code flash 3 (1M)
    * D Flash: Block data flash (128K)
    * U Flash: Block Utest (8K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_4_S000	8
...	...
FLS_DATA_ARRAY_0_BLOCK_4_S015	8
FLS_CODE_ARRAY_0_BLOCK_0_S000	8
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	8
FLS_CODE_ARRAY_0_BLOCK_1_S128	8
...	...
FLS_CODE_ARRAY_0_BLOCK_1_S255	8
FLS_CODE_ARRAY_0_BLOCK_2_S256	8
...	...
FLS_CODE_ARRAY_0_BLOCK_2_S383	8
FLS_CODE_ARRAY_0_BLOCK_3_S384	8
...	...
FLS_CODE_ARRAY_0_BLOCK_3_S511	8
FLS_UTEST_ARRAY_0_S000	8

- For S32K396: has 6 MBytes of code flash (program flash), 128 KBytes of data flash and 8 Kbytes of Utest NVM
  - 6M P Flash (each sector is 8K so 6M/8K = 768 sectors)
  - 128K D flash (each sector is 8K so 128K/8K = 16 sectors)
  - 8K U Flash (each sector is 8K so 8K/8K = 1 sectors)
  - There are 5 blocks (read partitions):
    * P Flash: Block code flash 0 (2M)
    * P Flash: Block code flash 1 (2M)
    * P Flash: Block code flash 2 (2M)
    * D Flash: Block data flash (128K)
    * U Flash: Block Utest (8K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_3_S000	8
...	...
FLS_DATA_ARRAY_0_BLOCK_3_S015	8
FLS_CODE_ARRAY_0_BLOCK_0_S000	8
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	8
FLS_CODE_ARRAY_0_BLOCK_1_S128	8
...	...
FLS_CODE_ARRAY_0_BLOCK_1_S255	8
FLS_CODE_ARRAY_0_BLOCK_2_S256	8
...	...
FLS_CODE_ARRAY_0_BLOCK_2_S383	8
FLS_UTEST_ARRAY_0_S000	8

- For S32K3x8: has 8 MBytes of code flash (program flash), 128 KBytes of data flash and 8 Kbytes of Utest NVM
  - 8M P Flash (each sector is 8K so 8M/8K = 1024 sectors)
  - 128K D flash (each sector is 8k so 128K/8K = 16 sectors)
  - 8K U Flash (each sector is 8K so 8K/8K = 1 sectors)
  - There are 5 blocks (read partitions):
    - P Flash: Block code flash 0 (2M)
    - P Flash: Block code flash 1 (2M)
    - P Flash: Block code flash 2 (2M)
    - P Flash: Block code flash 3 (2M)
    - D Flash: Block data flash (128K)
    - U Flash: Block Utest (8K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_4_S000	8
...	...
FLS_DATA_ARRAY_0_BLOCK_4_S015	8
FLS_CODE_ARRAY_0_BLOCK_0_S000	8
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S255	8
FLS_CODE_ARRAY_0_BLOCK_1_S256	8
...	...
FLS_CODE_ARRAY_0_BLOCK_1_S511	8
FLS_CODE_ARRAY_0_BLOCK_2_S512	8
...	...
FLS_CODE_ARRAY_0_BLOCK_2_S767	8
FLS_CODE_ARRAY_0_BLOCK_3_S768	8
...	...
FLS_CODE_ARRAY_0_BLOCK_3_S1023	8
FLS_UTEST_ARRAY_0_S000	8

- For S32M276: has 1 MBytes of code flash (program flash), 64 KBytes of data flash and 8 Kbytes of Utest NVM
  - 1M P Flash (each sector is 8K so 1M/8K = 128 sectors)
  - 64K D Flash (each sector is 8K so 64K/8K = 8 sectors)
  - 8K U Flash (each sector is 8K so 8K/8K = 1 sectors)
  - There are 3 blocks (read partitions):
    - P Flash: Block code flash 0 (512K)
    - P Flash: Block code flash 1 (512K)
    - D Flash: Block data flash (64K)
    - U Flash: Block Utest (8K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_2_S000	8
...	...
FLS_DATA_ARRAY_0_BLOCK_2_S007	8
FLS_CODE_ARRAY_0_BLOCK_0_S000	8
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S063	8
FLS_CODE_ARRAY_0_BLOCK_1_S064	8
...	...
FLS_CODE_ARRAY_0_BLOCK_1_S127	8
FLS_UTEST_ARRAY_0_S000	8

3.4 Deviations from Requirements

The driver deviates from the AUTOSAR MEM Driver software specification in some places. The table identifies the AUTOSAR requirements that are not fully implemented, implemented differently, not available, not testable or out of scope for the Mem_43_InFls Driver.

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 Mem_43_InFls driver.

Requirement	Status	Description	Notes
SWS_Mem_00062	N/I	If the memory hardware provides ECC information, the Mem driver shall check for correctable ECC errors and set the job result code to MEM_ECC_CORRECTED and proceed with the current job processing.	The update will be implemented on ARTD-27001
SWS_Mem_00063	N/I	If the memory hardware provides ECC information, the Mem driver shall check for uncorrectable ECC errors and set the job result code to MEM_ECC_UNCORRECTED and abort the current job processing.	The update will be implemented on ARTD-27001
SWS_Mem_00077	N/I	In case the last memory operation was completed but the ECC circuit corrected an ECC error, the job result shall be set to MEM_ECC_CORRECTED.	The update will be implemented on ARTD-27001
SWS_Mem_00078	N/I	In case the last memory operation didn't complete due to an uncorrectable ECC error, the job result shall be set to MEM_ECC_UNCORRECTED.	The update will be implemented on ARTD-27001
SWS_Mem_00082	N/I	In case a memory device provides a hardware-based suspend/resume mechanism, the Mem driver shall utilize the hardware capabilities to implement the Mem_Suspend and Mem_Resume services. If no hardware suspend/resume support is available, the Mem_Suspend and Mem_Resume services shall return MEM_E_SERVICE_NOT_AVAIL.	- Depend on mem device - The update will be implemented on ARTD-59202
SWS_Mem_00053	N/I	All hardware specific routines shall be accessed by the Mem_HwSpecificService() service.	The update will be implemented on ARTD-59205
SWS_Mem_00070	N/I	In case a service is not relevant for a specific memory device technology, the service shall always return MEM_E_SERVICE_NOT_AVAIL.	The update will be implemented on ARTD-59205
SWS_Mem_00080	N/I	The service Mem_Suspend shall suspend any ongoing flash operations using an according hardware mechanism.	- Depend on mem device - The update will be implemented on ARTD-59202
SWS_Mem_00083	N/I	In case a suspend operation is already in pending, Mem_Suspend shall reject the request by returning E_NOT_OK without further actions.	- Not mapping with return void in Requirement SWS_Mem_10024 - The update will be implemented on ARTD-59202

Requirement	Status	Description	Notes
SWS_Mem_00085	N/I	If development error detection is enabled by MemGeneral.MemDev↩ ErrorDetect, the service Mem_↩ Suspend shall check that the Mem driver has been initialized. If this check fails, Mem_Suspend shall raise the development error MEM_E_UNINIT.	The update will be implemented on ARTD-59202
SWS_Mem_00081	N/I	The service Mem_Resume shall resume a flash operations that was suspended by the service Mem_Suspend.	The update will be implemented on ARTD-59202
SWS_Mem_00084	N/I	In case no suspend operation is pending, Mem_Resume shall reject the request by returning E_NOT_OK without further actions.	The update will be implemented on ARTD-59202
SWS_Mem_00086	N/I	If development error detection is enabled by MemGeneral.MemDev↩ ErrorDetect, the service Mem_↩ Resume shall check that the Mem driver has been initialized. If this check fails, Mem_Resume shall raise the development error MEM_E_UNINIT.	The update will be implemented on ARTD-59202
SWS_Mem_00019	N/I	If development error detection is enabled by MemGeneral.MemDev↩ ErrorDetect, the service Mem_↩ PropagateError shall check that the Mem driver has been initialized. If this check fails, Mem_PropagateError shall raise the development error M↩ EM_E_UNINIT.	The update will be implemented on ARTD-59205
SWS_Mem_00020	N/I	If development error detection is enabled by MemGeneral.MemDev↩ ErrorDetect, the service Mem_↩ PropagateError shall check that the provided is consistent with the configuration. If this check fails, Mem_↩ PropagateError shall raise the development error MEM_E_PARAM_INS↩ TANCE_ID.	The update will be implemented on ARTD-59205
SWS_Mem_00061	N/I	If the Mem_PropagateError service is called, the Mem driver shall set the job result code to MEM_ECC_UNCOR↩ RECTED and cancel the current job processing.	The update will be implemented on ARTD-59205
SWS_Mem_00068	N/I	If development error detection is enabled by MemGeneral.MemDev↩ ErrorDetect, the service Mem_Hw↩ SpecificService shall check that the Mem driver has been initialized. If this check fails, Mem_HwSpecific↩ Service shall raise the development error MEM_E_UNINIT.	The update will be implemented on ARTD-59205

Requirement	Status	Description	Notes
SWS_Mem_00026	N/I	If development error detection is enabled by MemGeneral.MemDev↩ ErrorDetect, the service Mem_Hw↩ SpecificService shall check that the provided is consistent with the configuration. If this check fails, Mem_Hw↩ SpecificService shall raise the development error MEM_E_PARAM_INS↩ TANCE_ID.	The update will be implemented on ARTD-59205
SWS_Mem_00027	N/I	If development error detection is enabled by MemGeneral.MemDev↩ ErrorDetect, the service Mem_Hw↩ SpecificService shall raise the development error MEM_E_PARAM_PO↩ INTER if the or argument is a NULL pointer.	The update will be implemented on ARTD-59205

Requirement	Status	Description	Notes
SWS_Mem_10017	N/I	Service Name: Mem_HwSpecific↩ Service (draft) Syntax: Std_ReturnType Mem_↩ HwSpecificService (Mem_InstanceIdType instanceId, Mem_HwServiceIdType hwServiceId, Mem_DataType∗ dataPtr, Mem_LengthType∗ lengthPtr) Service ID [hex]: 0x0a Sync/Async: Asynchronous Reentrancy: Non Reentrant Parameters (in): instanceId: ID of the related memory driver instance. hwServiceId: Hardware specific service request identifier for dispatching the request. dataPtr: Request specific data pointer. lengthPtr: Size pointer of the data passed by dataPtr. Parameters (inout): None Parameters (out): None Return value: Std_ReturnType: E_OK: The requested job has been accepted by the module. E_NOT_OK: The requested job has not been accepted by the module. E_MEM_SERVICE_NOT_AV↩ AIL: The service function is not implemented. Description: Triggers a hardware specific memory driver job. dataPtr can be used to pass and return data to/from this service. This service is just a dispatcher to the hardware specific service implementation referenced by hwServiceId. The result of this service can be retrieved using the Mem_GetJobResult API. If the hardware specific operation was successful, the result of the job is MEM_JOB_OK. If the hardware specific operation failed, the result of the job is MEM_JOB_FAILED.: Tags: atp.Status=draft Available via: Mem.h	The update will be implemented on ARTD-59205

Requirement	Status	Description	Notes
SWS_Mem_10015	N/I	Service Name: Mem_PropagateError (draft) Syntax: void Mem_PropagateError (Mem_InstanceIdType instanceId) Service ID [hex]: 0x08 Sync/Async: Synchronous Reentrancy: Non Reentrant Parameters (in): instanceId: ID of the related memory driver instance. Parameters (inout): None Parameters (out): None Return value: None Description: This service can be used to report an access error in case the Mem driver cannot provide the access error information - typically for ECC faults. It is called by the system E↩ CC handler to propagate an ECC error to the memory upper layers..: Tags↩ : atp.Status=draft Available via: Mem.h	The update will be implemented on ARTD-59205
SWS_Mem_10025	N/I	Service Name: Mem_Resume (draft) Syntax: void Mem_Resume (Mem_InstanceIdType instanceId) Service ID [hex]: 0x0d Sync/Async: Synchronous Reentrancy: Non Reentrant Parameters (in): instanceId: ID of the related memory driver instance. Parameters (inout): None Parameters (out): None Return value: None Description: Resume suspended memory operation using hardware mechanism.: Tags: atp.Status=draft Available via: Mem.h	The update will be implemented on ARTD-59202
SWS_Mem_10026	N/I	Name: Mem_HwServiceIdType (draft) Kind: Type Derived from: uint32 Description: Hardware specific service request identifier type: Tags: atp.↩ Status=draft Available via: Mem.h	The update will be implemented on ARTD-59202

3.5 Driver Limitations

3.5.1 For Driver

- The driver does not support Mem_43_InFls_Suspend, Mem_43_InFls_Resume, Mem_43_InFls_↩ PropagateError and Mem_43_InFls_HwSpecificService functions. They will be implemented in next release.

- Mem Burst Mode will be implemented in next release.

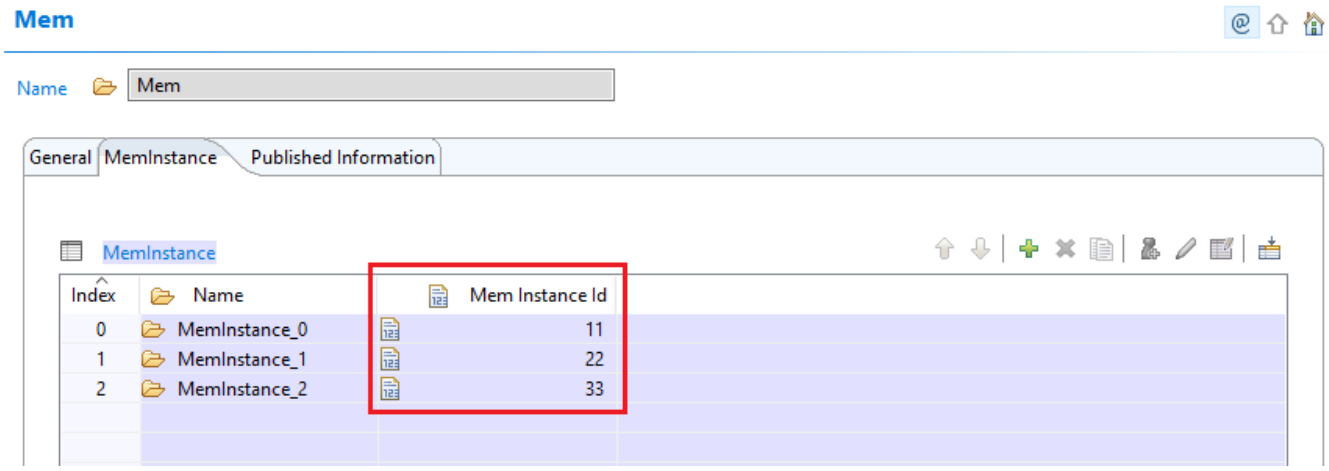
3.5.2 For Errata

- S32K3x4 and S32KK3x2
  - ERR051127 PFLASH: Flash read during array integrity may return incorrect read data
    - This will be implemented by ticket ARTD-59883
  - ERR050609 PFLASH: PFCR4[DERR_SUP] may not work as expected
    - Users need to be aware. FCCU need to be configured to workaround. Detail on Errata document.
  - ERR051114 PFLASH: PFCBLK0_LOCKMASTER_SS register provides incorrect status of the super sector program/erase lock bit domain ID owner.
    - There is no software workaround for this erratum. Master can find out if it owns the lock bit by toggling PFCBLK0_SSPELOCK register lock bit. If it owns lock bit then PFCBLK0_SSPELOCK register lock bit will be toggled
- S32K39x
  - ERR051358: Embedded Flash Memory: Unexpected false Address Encode Error may be generated
    - This error only appears when flash memory reads transition the 1 MB boundary of a 2 MB block in Utest mode, and it is not implemented yet in the driver code (C40_Ip.c).

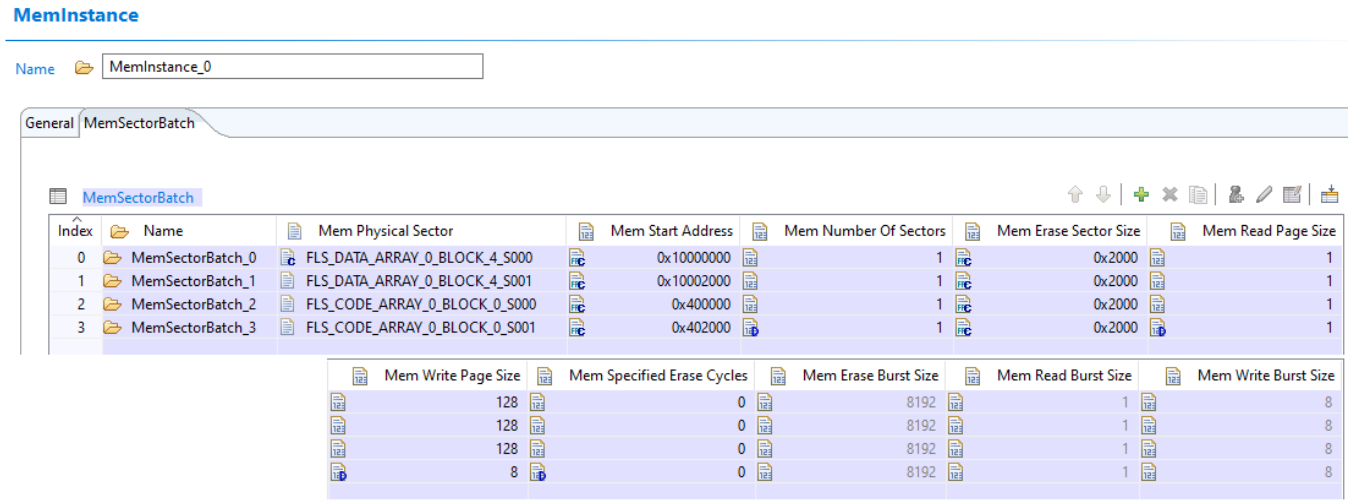
3.6 Driver usage and configuration tips

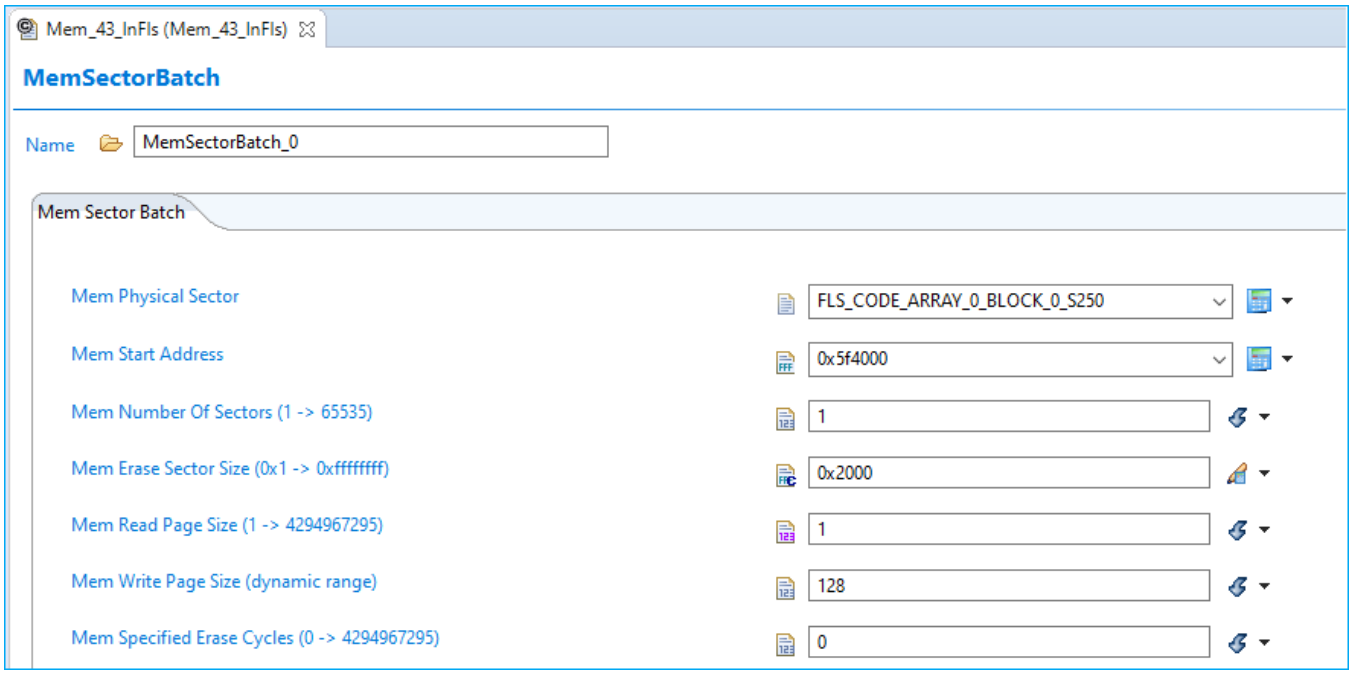
3.6.1 Tresos configuration

The MemInstance Element list can be configured to contain multiple independent instance, each instance must have different ID.



The MemSectorBatch Element list can be configured to contain multiple sector batch.





It's possible to modify the parameters for the sector erase, page write and also the read operation. The MemSector↩ Batch Element Configuration contain the static configuration of the element:

- Mem Physical Sector: Physical flash device sector (Data, Code and UTest flash block sector)
- Mem Start Address: Physical start address of the sector (batch).
- Mem Number Of Sectors (1 -> 65535): Number of contiguous sectors with identical values for MemSectorSize and MemPageSize.
- Mem Erase Sector Size (1 -> 4294967295): Size of this sector in bytes.

Note

A sector is the smallest erasable unit.
Max erase sector size = 0x2000 (8192)

- Mem Read Page Size (1 -> 4294967295): Size of a read page of this sector in bytes.

  Note

  A read page is the smallest readable unit.

- Mem Write Page Size: Size of a write page of this sector in bytes.

  Note

  A write page is the smallest writeable unit.

- Mem Specified Erase: Number of erase cycles specified for the memory device (usually given in the device data sheet)

3.7 Runtime errors

The driver generates the following DET errors at runtime.

Table 3.9 Errors (reported by DET)

Function	Error Code	Condition triggering the error
Mem_43_InFls_Init()	MEM_43_INFLS_E_PARAM↩ _POINTER	Invalid input parameter.
Mem_43_InFls_GetVersionInfo()	MEM_43_INFLS_E_PARAM↩ _POINTER	Invalid input parameter.
Mem_43_InFls other functions	MEM_43_INFLS_E_UNINIT	The driver is in an uninitialized state.
	MEM_43_INFLS_E_PARAM↩ _POINTER	Invalid input parameter.
	MEM_43_INFLS_E_JOB_PE↩ NDING	A job request is still in progress.
	MEM_43_INFLS_E_PARAM↩ _INSTANCE_ID	Invalid driver instance ID.
	MEM_43_INFLS_E_PARAM↩ _ADDRESS	Invalid address.
	MEM_43_INFLS_E_PARAM↩ _LENGTH	Invalid length.

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 Mem
  - Container MemAutosarExt
    - Parameter MemUsesAlterInterface
    - Parameter MemEnableUserModeSupport
    - Parameter MemSynchronizeCache
    - Parameter MemDataErrorSuppression
    - Parameter MemBlock4PipeSelect
    - Parameter MemDomainID
  - Container MemGeneral
    - Parameter MemDevErrorDetect
    - Parameter MemSectorSetLockApi
    - Parameter MemEraseVerificationEnabled
    - Parameter MemWriteVerificationEnabled
    - Parameter MemECCCheck
    - Parameter MemECCHandlingProtectionHook
    - Parameter MemStartFlashAccessNotif
    - Parameter MemFinishedFlashAccessNotif
    - Parameter MemTimeoutSupervisionEnabled
    - Parameter MemTimeoutMethod
    - Parameter MemAsyncWriteTimeout
    - Parameter MemAsyncEraseTimeout
    - Parameter MemSyncWriteTimeout
    - Parameter MemSyncEraseTimeout
    - Parameter MemAbortTimeout
  - Container MemInstance
    - Parameter MemInstanceId
    - Container MemSectorBatch
      - Parameter MemPhysicalSector
      - Parameter MemStartAddress
      - Parameter MemNumberOfSectors

- Parameter MemEraseSectorSize
- Parameter MemReadPageSize
- Parameter MemWritePageSize
- Parameter MemSpecifiedEraseCycles
- Container MemBurstSettings
- Parameter MemEraseBurstSize
- Parameter MemReadBurstSize
- Parameter MemWriteBurstSize

– Container MemPublishedInformation

* Parameter MemErasedValue
* Parameter MemECCValue

– Container CommonPublishedInformation

* Parameter ArReleaseMajorVersion
* Parameter ArReleaseMinorVersion
* Parameter ArReleaseRevisionVersion
* Parameter ModuleId
* Parameter SwMajorVersion
* Parameter SwMinorVersion
* Parameter SwPatchVersion
* Parameter VendorApiInfix
* Parameter VendorId

4.1 Module Mem

Configuration of the Mem_43_InFls module.

Included containers:

- MemAutosarExt
- MemGeneral
- MemInstance
- MemPublishedInformation
- CommonPublishedInformation

Property	Value
type	ECUC-MODULE-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantSupport	false
supportedConfigVariants	VARIANT-PRE-COMPILE

4.2 Container MemAutosarExt

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

4.3 Parameter MemUsesAlterInterface

Vendor specific: When enabled: A second interface is made available for program and erase operations

The alternate interface includes an alternate MCR register, alternate MCRS register,

alternate PEADR register, and alternate sector and super sector PELOCK registers.

Program and Erase procedures on the alternate interface follow exactly the same flow as the main interface

Note:

+) Both the Alternate Interface and the Main Interface have the same priority which allows

both operations to initiate in parallel.

+) Alternate inteface may not be available for the application cores, it only allocated to the HSE core.

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
defaultValue	false

4.4 Parameter MemEnableUserModeSupport

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
defaultValue	false

4.5 Parameter MemSynchronizeCache

Vendor specific:

Synchronize the memory by invalidating the cache after each flash hardware operation.

The MEM driver needs to maintain the memory coherency by means of three methods:

1. Disable data cache, or
2. Configure the flash region upon which the driver operates, as non-cacheable, or
3. Enable the MemSynchronizeCache feature.

Depending on the application configuration, one option may be more beneficial than other.

Enabled: The MEM driver will call Mcl cache API functions in order to invalidate the cache

after each high voltage operation(write,erase) and before each read operation, in order

to ensure that the cache and the modified flash memory are in sync.

If enabled, the driver will attempt to invalidate only the modified lines from the cache.

If the size of the region to be invalidated is greater than half of the cache size, then

the entire cache is invalidated.

Note: If enabled, the MclEnableCache parameter has to be enabled and the MCL plugin included as a dependency.

Disabled: The upper layers have to ensure that the flash region upon which the driver operates is not cached.

This can be obtained by either disabling the data cache or by configuring the memory region as non-cacheable.

Note: This feature is applicable only if supported on the current platform.

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
defaultValue	false

4.6 Parameter MemDataErrorSuppression

Vendor specific: See the Embedded Flash Memory configuration information or system memory map for which flash memory blocks are affected by this field.

Disable - Reports ECC events on data flash memory accesses.

Enable - Single-bit and multi-bit ECC events on data flash memory accesses are suppressed.

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
defaultValue	false

4.7 Parameter MemBlock4PipeSelect

Vendor specific: Select the pipe to be used for accessing the internal flash block 4.

PFLASH has four independent command pipes to issue four parallel read commands to different flash memory blocks.

The access to block 4 can be through any of the command pipes:

MEM_COMMAND_PIPE_0 - Block 4 access is always through pipe 0.

MEM_COMMAND_PIPE_1 - Block 4 access is always through pipe 1.

MEM_COMMAND_PIPE_2 - Block 4 access is always through pipe 2.

MEM_COMMAND_PIPE_3 - Block 4 access is always through pipe 3.

MEM_ANY_COMMAND_PIPES - Block 4 access can be through any of the command pipes, based on which command pipe is available for block 4 access.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	MEM_COMMAND_PIPE_0
literals	['MEM_COMMAND_PIPE_0', 'MEM_COMMAND_PIPE_1', 'MEM_CO↩ MMAND_PIPE_2', 'MEM_COMMAND_PIPE_3', 'MEM_ANY_COMMA↩ ND_PIPES']

4.8 Parameter MemDomainID

Vendor specific: The domain ID assigned by the XRDC.

Note: Users have to fill using core.

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: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.9 Container MemGeneral

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

4.10 Parameter MemDevErrorDetect

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-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.11 Parameter MemSectorSetLockApi

Pre-processor switch to enable / disable the Sector Set Lock Api.

true: Sector Set Lock Api enabled.

false: Sector Set Lock Api disabled.

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
defaultValue	true

4.12 Parameter MemEraseVerificationEnabled

Pre-processor switch to enable / disable the erase blank check. After a flash block has been erased, the erase blank check compares the contents of the addressed memory area against the value of an erased flash cell to check that the block has been completely erased.

true: Memory region is checked to be erased.

false: Memory region is not checked to be erased.

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
defaultValue	false

4.13 Parameter MemWriteVerificationEnabled

Pre-processor switch to enable / disable the write verify check. After writing a flash block, the write verify check compares the contents of the reprogrammed memory area against the contents of the provided application buffer to check that the block has been completely reprogrammed.

true: Written data is compared directly after write.

false: Written date is not compared directly after write.

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
defaultValue	false

4.14 Parameter MemECCCheck

Vendor specific: Pre-processor switch to enable / disable the API to report data storage (ECC) errors to the flash driver.

This is the first ECC handling approach which modifies the program counter to skip the instruction causing the fault.

Please read the chapter Exception Handler in case of ECC error in IM for more information.

true : The ECC check by HardfaultHandler API is enabled.

false: The ECC check by HardfaultHandler API is disabled.

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
defaultValue	false

4.15 Parameter MemECCHandlingProtectionHook

Vendor specific: Pre-processor switch to enable / disable the API to report data storage (ECC) errors to the flash driver.

This is the second ECC handling approach which is compatible with Autosar Os.

Please read the chapter Exception Handler in case of ECC error in IM for more information.

true : The ECC check by AutosarOs API is enabled.

false: The ECC check by AutosarOs API is disabled.

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
defaultValue	false

4.16 Parameter MemStartFlashAccessNotif

Vendor specific: Start flash access. If configured, this notification will be called before any flash memory access.

It is called before flash memory read accesses(in read, compare, verify write, verify erase jobs) and

before flash memory program operations(in write and erase jobs).

The purpose of this notification together with MemFinishedFlashAccess, is to ensure that, if needed, no other

executed code(other tasks, cores, masters) will access the affected flash area simultaneously with the access

initiated by the driver. For more details, see Integration manual, chapter 5. Module requirements.

Note: Disable the error notification to have it set as NULL_PTR

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	true
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	Mem_StartFlashAccessNotif

4.17 Parameter MemFinishedFlashAccessNotif

Vendor specific: Finished flash access. If configured, this notification will be called after any flash memory access. It is called after flash memory read accesses(in read, compare, verify write, verify erase jobs).

The purpose of this notification together with MemStartFlashAccess, is to ensure that, if needed, no other executed code(other tasks, cores, masters) will access the affected flash area simultaneously with the access initiated by the driver. For more details, see Integration manual, chapter 5. Module requirements.

Note: Disable the error notification to have it set as NULL_PTR

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	true
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	Mem_FinishedFlashAccessNotif

4.18 Parameter MemTimeoutSupervisionEnabled

Compile switch to enable timeout supervision.

true: timeout supervision for read/erase/write/compare jobs enabled.

false: timeout supervision for read/erase/write/compare jobs disabled.

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
defaultValue	false

4.19 Parameter MemTimeoutMethod

Vendor specific: Counter type used in timeout detection for Mem service request.

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

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-PRE-COMPILE: PRE-COMPILE
defaultValue	OSIF_COUNTER_DUMMY
literals	['OSIF_COUNTER_DUMMY', 'OSIF_COUNTER_SYSTEM', 'OSIF_COU↩ NTER_CUSTOM']

4.20 Parameter MemAsyncWriteTimeout

Vendor specific: Mem Async Write Timeout is the timeout value for write operation in asynchronous mode.

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: PRE-COMPILE
defaultValue	2147483647
max	2147483647
min	0

4.21 Parameter MemAsyncEraseTimeout

Vendor specific: Mem Async Erase Timeout is the timeout value for erase operation in asynchronous mode.

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: PRE-COMPILE
defaultValue	2147483647
max	2147483647
min	0

4.22 Parameter MemSyncWriteTimeout

Vendor specific: Mem Sync Write Timeout is the timeout value for write operation in synchronous mode.

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: PRE-COMPILE
defaultValue	2147483647
max	2147483647
min	0

4.23 Parameter MemSyncEraseTimeout

Vendor specific: Mem Sync Erase Timeout is the timeout value for erase operation in synchronous mode.

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: PRE-COMPILE
defaultValue	2147483647
max	2147483647
min	0

4.24 Parameter MemAbortTimeout

Vendor specific: Mem Abort Timeout is the timeout value for aborting an ongoing operation.

The timeout is used also in Mem_Cancel API and Abort Erase suspend, if enabled and if the flash hardware channel does not support an immediate abort feature.

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: PRE-COMPILE
defaultValue	32767
max	2147483647
min	0

4.25 Container MemInstance

This container includes the Mem driver instance specific configuration parameters.

Included subcontainers:

- MemSectorBatch

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.26 Parameter MemInstanceId

The unique numeric identifier which is used to

reference a Mem driver instance in case multiple devices

of the same type shall be addressed by one Mem driver.

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

4.27 Container MemSectorBatch

Configuration description of a programmable sector or sector batch.

Sector batch means that homogenous and coherent sectors can be configured as one MemSector element.

It is recommended to group as many identical sectors as possible together.

Included subcontainers:

- MemBurstSettings

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.28 Parameter MemPhysicalSector

Vendor specific: Physical flash device sector.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	FLS_DATA_ARRAY_0_BLOCK_4_S000

Property	Value
literals	['FLS_DATA_ARRAY_0_BLOCK_4_S000', 'FLS_DATA_ARRAY_0_BLOCK_4_S001', 'FLS_DATA_ARRAY_0_BLOCK_4_S002', 'FLS_DATA_ARRAY_0_BLOCK_4_S003', 'FLS_DATA_ARRAY_0_BLOCK_4_S004', 'FLS_DATA_ARRAY_0_BLOCK_4_S005', 'FLS_DATA_ARRAY_0_BLOCK_4_S006', 'FLS_DATA_ARRAY_0_BLOCK_4_S007', 'FLS_DATA_ARRAY_0_BLOCK_4_S008', 'FLS_DATA_ARRAY_0_BLOCK_4_S009', 'FLS_DATA_ARRAY_0_BLOCK_4_S010', 'FLS_DATA_ARRAY_0_BLOCK_4_S011', 'FLS_DATA_ARRAY_0_BLOCK_4_S012', 'FLS_DATA_ARRAY_0_BLOCK_4_S013', 'FLS_DATA_ARRAY_0_BLOCK_4_S014', 'FLS_DATA_ARRAY_0_BLOCK_4_S015', 'FLS_CODE_ARRAY_0_BLOCK_0_S000', 'FLS_CODE_ARRAY_0_BLOCK_0_S001', 'FLS_CODE_ARRAY_0_BLOCK_0_S002', 'FLS_CODE_ARRAY_0_BLOCK_0_S003', 'FLS_CODE_ARRAY_0_BLOCK_0_S004', 'FLS_CODE_ARRAY_0_BLOCK_0_S005', 'FLS_CODE_ARRAY_0_BLOCK_0_S006', 'FLS_CODE_ARRAY_0_BLOCK_0_S007', 'FLS_CODE_ARRAY_0_BLOCK_0_S008', 'FLS_CODE_ARRAY_0_BLOCK_0_S009', 'FLS_CODE_ARRAY_0_BLOCK_0_S010', 'FLS_CODE_ARRAY_0_BLOCK_0_S011', 'FLS_CODE_ARRAY_0_BLOCK_0_S012', 'FLS_CODE_ARRAY_0_BLOCK_0_S013', 'FLS_CODE_ARRAY_0_BLOCK_0_S014', 'FLS_CODE_ARRAY_0_BLOCK_0_S015', 'FLS_CODE_ARRAY_0_BLOCK_0_S016', 'FLS_CODE_ARRAY_0_BLOCK_0_S017', 'FLS_CODE_ARRAY_0_BLOCK_0_S018', 'FLS_CODE_ARRAY_0_BLOCK_0_S019', 'FLS_CODE_ARRAY_0_BLOCK_0_S020', 'FLS_CODE_ARRAY_0_BLOCK_0_S021', 'FLS_CODE_ARRAY_0_BLOCK_0_S022', 'FLS_CODE_ARRAY_0_BLOCK_0_S023', 'FLS_CODE_ARRAY_0_BLOCK_0_S024', 'FLS_CODE_ARRAY_0_BLOCK_0_S025', 'FLS_CODE_ARRAY_0_BLOCK_0_S026', 'FLS_CODE_ARRAY_0_BLOCK_0_S027', 'FLS_CODE_ARRAY_0_BLOCK_0_S028', 'FLS_CODE_ARRAY_0_BLOCK_0_S029', 'FLS_CODE_ARRAY_0_BLOCK_0_S030', 'FLS_CODE_ARRAY_0_BLOCK_0_S031', 'FLS_CODE_ARRAY_0_BLOCK_0_S032', 'FLS_CODE_ARRAY_0_BLOCK_0_S033', 'FLS_CODE_ARRAY_0_BLOCK_0_S034', 'FLS_CODE_ARRAY_0_BLOCK_0_S035', 'FLS_CODE_ARRAY_0_BLOCK_0_S036', 'FLS_CODE_ARRAY_0_BLOCK_0_S037', 'FLS_CODE_ARRAY_0_BLOCK_0_S038', 'FLS_CODE_ARRAY_0_BLOCK_0_S039', 'FLS_CODE_ARRAY_0_BLOCK_0_S040', 'FLS_CODE_ARRAY_0_BLOCK_0_S041', 'FLS_CODE_ARRAY_0_BLOCK_0_S042', 'FLS_CODE_ARRAY_0_BLOCK_0_S043', 'FLS_CODE_ARRAY_0_BLOCK_0_S044', 'FLS_CODE_ARRAY_0_BLOCK_0_S045', 'FLS_CODE_ARRAY_0_BLOCK_0_S046', 'FLS_CODE_ARRAY_0_BLOCK_0_S047', 'FLS_CODE_ARRAY_0_BLOCK_0_S048', 'FLS_CODE_ARRAY_0_BLOCK_0_S049', 'FLS_CODE_ARRAY_0_BLOCK_0_S050', 'FLS_CODE_ARRAY_0_BLOCK_0_S051', 'FLS_CODE_ARRAY_0_BLOCK_0_S052', 'FLS_CODE_ARRAY_0_BLOCK_0_S053', 'FLS_CODE_ARRAY_0_BLOCK_0_S054', 'FLS_CODE_ARRAY_0_BLOCK_0_S055', 'FLS_CODE_ARRAY_0_BLOCK_0_S056', 'FLS_CODE_ARRAY_0_BLOCK_0_S057', 'FLS_CODE_ARRAY_0_BLOCK_0_S058', 'FLS_CODE_ARRAY_0_BLOCK_0_S059', 'FLS_CODE_ARRAY_0_BLOCK_0_S060', 'FLS_CODE_ARRAY_0_BLOCK_0_S061', 'FLS_CODE_ARRAY_0_BLOCK_0_S062', 'FLS_CODE_ARRAY_0_BLOCK_0_S063', 'FLS_CODE_ARRAY_0_BLOCK_0_S064', 'FLS_CODE_ARRAY_0_BLOCK_0_S065', 'FLS_CODE_ARRAY_0_BLOCK_0_S066', 'FLS_CODE_ARRAY_0_BLOCK_0_S067', 'FLS_CODE_ARRAY_0_BLOCK_0_S068', 'FLS_CODE_ARRAY_0_BLOCK_0_S069', 'FLS_CODE_ARRAY_0_BLOCK_0_S070', 'FLS_CODE_ARRAY_0_BLOCK_0_S071', 'FLS_CODE_ARRAY_0_BLOCK_0_S072', 'FLS_CODE_ARRAY_0_BLOCK_0_S073', 'FLS_CODE_ARRAY_0_BLOCK_0_S074', 'FLS_CODE_ARRAY_0_BLOCK_0_S075', 'FLS_CODE_ARRAY_0_BLOCK_0_S076', 'FLS_CODE_ARRAY_0_BLOCK_0_S077', 'FLS_CODE_ARRAY_0_BLOCK_0_S078', 'FLS_CODE_ARRAY_0_BLOCK_0_S079',

Property	Value

4.29 Parameter MemStartAddress

Physical start address of the sector (batch).

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-PRE-COMPILE: PRE-COMPILE
defaultValue	4194304
max	9223372036854775807
min	0

4.30 Parameter MemNumberOfSectors

Number of contiguous sectors with identical values for MemSectorSize and MemPageSize.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	65535
min	1

4.31 Parameter MemEraseSectorSize

Size of this sector in bytes.

A sector is the smallest erasable unit.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	8192
max	4294967295
min	1

4.32 Parameter MemReadPageSize

Size of a read page of this sector in bytes.

A read page is the smallest readable unit.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	4294967295
min	1

4.33 Parameter MemWritePageSize

Size of a write page of this sector in bytes.

A write page is the smallest writeable unit.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	8
max	128
min	1

4.34 Parameter MemSpecifiedEraseCycles

Number of erase cycles specified for the memory device

(usually given in the device data sheet).

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-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.35 Container MemBurstSettings

Container for burst setting configuration parameters of the Mem driver.

A sector burst can be used for improved performance.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.36 Parameter MemEraseBurstSize

Size of sector erase burst in bytes. A sector burst can be used for

improved performance and is typically (a subset of) a sector batch.

To make use of the sector erase burst feature, the physical start

address of the sector batch must be aligned to the sector erase burst size.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	4096
max	4294967295
min	1

4.37 Parameter MemReadBurstSize

This value specifies the maximum number of bytes the MemAcc

module requests within one Mem read request.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	4294967295
min	1

4.38 Parameter MemWriteBurstSize

Size of page write/program burst in bytes. A sector burst can be used

for improved performance and is typically (a subset of) a sector batch.

To make use of the write burst feature, the physical start address must

be aligned to the write burst size.

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-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	4294967295
min	1

4.39 Container MemPublishedInformation

Additional published parameters not covered by CommonPublishedInformation container.

Note that these parameters do not have any configuration class setting, since they are published information.

Included subcontainers:

- None

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

4.40 Parameter MemErasedValue

The contents of an erased memory cell.

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-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	4294967295
max	4294967295
min	0

4.41 Parameter MemECCValue

Vendor specific: The contents of an ECC flash memory line.

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
defaultValue	4294967295
max	4294967295
min	0

4.42 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

4.43 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

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	4
max	4
min	4

4.44 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
defaultValue	7
max	7
min	7

4.45 Parameter ArReleaseRevisionVersion

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

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

Property	Value
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.46 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
defaultValue	91
max	91
min	91

4.47 Parameter SwMajorVersion

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

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

Property	Value
defaultValue	3
max	3
min	3

4.48 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
defaultValue	0
max	0
min	0

4.49 Parameter SwPatchVersion

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

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

4.50 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
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	

4.51 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
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 C40 IP Driver

6.1.1 Detailed Description implements C40_Ip.h_Artifact

implements C40_Ip_Ac.h_Artifact

implements C40_Ip_TrustedFunctions.h_Artifact

implements C40_Ip_Types.h_Artifact

Data Structures

- struct C40_Ip_MisrType
  MISR structure. Implements : C40_Ip_MisrType_Class. More...
- struct Fls_ExceptionDetailsType
  Detailed information on the exception. More...
- struct C40_ConfigType
  C40 Configuration Structure. More...

Macros

- #define C40_SECTOR_SIZE
  Each sector has a size of 8k.
- #define C40_CODE_BASE_ADDRESS
  Code sectors base address.
- #define C40_DATA_BASE_ADDRESS
  Data sectors base address.
- #define C40_DATA_END_ADDRESS
  Data sectors end address.
- #define C40_UTEST_BASE_ADDRESS
  UTest sector base address.

- #define C40_WRITE_DOUBLE_WORD
  Program alignment.
- #define C40_DATA_SIZE_BYTES_U32
  Main interface program data registers (DATA0 - DATA31)
- #define C40_USER_TEST_PASSWORD
  For UTE bit, the password 0xF9F9_9999 must be written to the UT0 register, and this must be a 32bit write.
- #define FLASH_USER_TEST_WAIT
  Time out for wait done.
- #define FLS_UNHANDLED
  Return value for Fls_DsiHandler and Fls_MciHandler.
- #define FLS_HANDLED_RETRY
  Return value for Fls_DsiHandler and Fls_MciHandler.
- #define FLS_HANDLED_SKIP
  Return value for Fls_DsiHandler and Fls_MciHandler.
- #define FLS_HANDLED_STOP
  Return value for Fls_DsiHandler and Fls_MciHandler.
- #define FLS_SIZE_1BYTE
  the number of bytes uses to compare (1 byte).
- #define FLS_SIZE_2BYTE
  the number of bytes uses to compare (2 bytes).
- #define FLS_SIZE_4BYTE
  the number of bytes uses to compare (4 bytes).

Types Reference

- typedef uint8 Fls_CompHandlerReturnType
  return value of ecc checking function
- typedef const uint8 * Fls_InstructionAddressType
  the instruction that generated the ECC
- typedef uint32 Fls_DataAddressType
  data address that caused the ECC error
- typedef void(* C40_StartFlashAccessNotifPtrType) (void)
  Fls Start Flash Access Notification Pointer Type.
- typedef void(* C40_FinishedFlashAccessNotifPtrType) (void)
  Fls Finished Flash Access Notification Pointer Type.

Enum Reference

- enum C40_Ip_InstanceType
  C40 hardware instance.
- enum C40_Ip_StatusType
  Enumeration of checking status errors or not.
- enum C40_Ip_FlashBlocksNumberType
  Enumeration of Blocks of memory flash .
- enum C40_Ip_FlashBreakPointsType

Enumeration breakpoints .

- enum C40_Ip_ArrayIntegritySequenceType

  Enumeration of Array Integrity Sequence(proprietary sequence or sequential) .
- enum C40_Ip_MarginOptionType

  Declarations of margin levels.
- enum C40_Ip_UtestStateType

  Declarations for flash suspend operation and resume operation and user test check state.

Function Reference

- C40_Ip_StatusType C40_Ip_Init (const C40_ConfigType *InitConfig)

  Initializes the C40 module.
- C40_Ip_StatusType C40_Ip_Abort (void)

  Abort a program or erase operation.
- C40_Ip_StatusType C40_Ip_Read (uint32 LogicalAddress, uint32 Length, uint8 *DestAddressPtr)

  This function fills data to DestAddressPtr.
- C40_Ip_StatusType C40_Ip_Compare (uint32 LogicalAddress, uint32 Length, const uint8 *SourceAddress↩ Ptr)

  Checks that there is the desired data at the specified address.
- C40_Ip_StatusType C40_Ip_GetLock (C40_Ip_VirtualSectorsType VirtualSector)

  Returns the locking status of the selected sector.
- C40_Ip_StatusType C40_Ip_ClearLock (C40_Ip_VirtualSectorsType VirtualSector, uint8 DomainIdValue)

  Unlocks the selected sector for the requesting core if possible.
- C40_Ip_VirtualSectorsType C40_Ip_GetSectorNumberFromAddress (uint32 TargetAddress)

  Get sector number from specified address.
- C40_Ip_FlashBlocksNumberType C40_Ip_GetBlockNumberFromAddress (uint32 TargetAddress)

  Get block number from target address.
- C40_Ip_StatusType C40_Ip_CheckUserTestStatus (const C40_Ip_MisrType *MisrExpectedValues, C40_Ip_UtestStateType *TestResult)

  Check the operation in user test mode.
- C40_Ip_StatusType C40_Ip_ArrayIntegrityCheck (uint32 SelectBlock, C40_Ip_ArrayIntegritySequenceType AddressSequence, C40_Ip_FlashBreakPointsType BreakPoints, const C40_Ip_MisrType *MisrSeedValues, uint8 DomainIdValue)

  Check the array integrity of the flash memory.
- C40_Ip_StatusType C40_Ip_ArrayIntegrityCheckSuspend (void)

  Suspend an on-going array integrity check.
- C40_Ip_StatusType C40_Ip_ArrayIntegrityCheckResume (void)

  Resume the previous suspend operation.
- C40_Ip_StatusType C40_Ip_UserMarginReadCheck (uint32 SelectBlock, C40_Ip_FlashBreakPointsType BreakPoints, C40_Ip_MarginOptionType MarginLevel, const C40_Ip_MisrType *MisrSeedValues, uint8 DomainIdValue)

  Check the user margin read of the flash memory.
- uint32 C40_Ip_GetFailedAddress (void)

  Get the failing address in memory.
- void C40_Ip_SetAsyncMode (const boolean Async)

  Set synch/Async at IP layer base on the Async of HLD.
- void C40_Ip_DataErrorSuppression (void)

Setup the ECC error handling on data flash block.

- uint32 C40_Ip_GetLockProtect (C40_Ip_VirtualSectorsType VirtualSector)

  Read lock bit status of flash sectors.
- void C40_Ip_SetLockProtect (C40_Ip_VirtualSectorsType VirtualSector)

  Locks the selected sector for the requesting core.
- void C40_Ip_ClearLockProtect (C40_Ip_VirtualSectorsType VirtualSector)

  Unlocks the selected sector for the requesting core.
- void C40_Ip_CheckLockDomainID_CheckRegister (C40_Ip_VirtualSectorsType VirtualSector, uint32 *CheckRegister, uint32 *TempLockMasterRegister)

  Read and check the lock domain ID for flash sectors.

6.1.2 Data Structure Documentation

6.1.2.1 struct C40_Ip_MisrType

MISR structure. Implements : C40_Ip_MisrType_Class.

Definition at line 282 of file C40_Ip_Types.h.

Data Fields

Type	Name	Description
uint32	arrMISRValue[10U]	The value of MISR, size of arrMISRValue is (FLASH_UM_COUNT +1)

6.1.2.2 struct Fls_ExceptionDetailsType

Detailed information on the exception.

The following information will be checked by the driver:

- if there is a pending read, compare.
- data_pt matches address currently accessed by pending flash read or flash compare job.
- if the exception syndrome register indicates DSI or MCI reason.

Definition at line 294 of file C40_Ip_Types.h.

Data Fields

Type	Name	Description
Fls_InstructionAddressType	instruction_pt	pointer to the instruction that generated the ECC
Fls_DataAddressType	data_pt	data address that caused the ECC error
uint32	syndrome_u32	details on the type of exception

6.1.2.3 struct C40_ConfigType

C40 Configuration Structure.

Implements : C40_ConfigType_Class

Definition at line 318 of file C40_Ip_Types.h.

Data Fields

Type	Name	Description
C40_StartFlashAccessNotifPtrType	startFlashAccessNotifPtr	Pointer to start flash access callout
C40_FinishedFlashAccessNotifPtrType	finishedFlashAccessNotifPtr	Pointer to finish flash access callout

6.1.3 Macro Definition Documentation

6.1.3.1 C40_SECTOR_SIZE

```
#define C40_SECTOR_SIZE
```

Each sector has a size of 8k.

Definition at line 77 of file C40_Ip_Types.h.

6.1.3.2 C40_CODE_BASE_ADDRESS

```
#define C40_CODE_BASE_ADDRESS
```

Code sectors base address.

Definition at line 81 of file C40_Ip_Types.h.

6.1.3.3 C40_DATA_BASE_ADDRESS

```
#define C40_DATA_BASE_ADDRESS
```

Data sectors base address.

Definition at line 85 of file C40_Ip_Types.h.

6.1.3.4 C40_DATA_END_ADDRESS

```
#define C40_DATA_END_ADDRESS
```

Data sectors end address.

Definition at line 89 of file C40_Ip_Types.h.

6.1.3.5 C40_UTEST_BASE_ADDRESS

```
#define C40_UTEST_BASE_ADDRESS
```

UTest sector base address.

Definition at line 93 of file C40_Ip_Types.h.

6.1.3.6 C40_WRITE_DOUBLE_WORD

```
#define C40_WRITE_DOUBLE_WORD
```

Program alignment.

Definition at line 97 of file C40_Ip_Types.h.

6.1.3.7 C40_DATA_SIZE_BYTES_U32

```
#define C40_DATA_SIZE_BYTES_U32
```

Main interface program data registers (DATA0 - DATA31)

Definition at line 101 of file C40_Ip_Types.h.

6.1.3.8 C40_USER_TEST_PASSWORD

```
#define C40_USER_TEST_PASSWORD
```

For UTE bit, the password 0xF9F9_9999 must be written to the UT0 register, and this must be a 32bit write.

Definition at line 107 of file C40_Ip_Types.h.

6.1.3.9 FLASH_USER_TEST_WAIT

```
#define FLASH_USER_TEST_WAIT
```

Time out for wait done.

Definition at line 111 of file C40_Ip_Types.h.

6.1.3.10 FLS_UNHANDLED

```
#define FLS_UNHANDLED
```

Return value for Fls_DsiHandler and Fls_MciHandler.

Module doesn't feel responsible (e.g. address does not belong to its current job, there is no current pending read/compare job, the syndrome is different).

Definition at line 118 of file C40_Ip_Types.h.

6.1.3.11 FLS_HANDLED_RETRY

```
#define FLS_HANDLED_RETRY
```

Return value for Fls_DsiHandler and Fls_MciHandler.

Module feels responsible, but wants to repeat the causing instruction. Maybe: it still uses information in MCM or ECSM module, but they are outdated (e.g. due to an erroneous DMA transfer in the meantime)

Definition at line 125 of file C40_Ip_Types.h.

6.1.3.12 FLS_HANDLED_SKIP

```
#define FLS_HANDLED_SKIP
```

Return value for Fls_DsiHandler and Fls_MciHandler.

Module feels responsible, the current job is marked as failed, processing may continue, skipping the causing instruction.

Definition at line 131 of file C40_Ip_Types.h.

6.1.3.13 FLS_HANDLED_STOP

```
#define FLS_HANDLED_STOP
```

Return value for Fls_DsiHandler and Fls_MciHandler.

Module feels responsible, but the only reaction is to stop the system (e.g.: try to shut-down in a quite safe way)

Definition at line 137 of file C40_Ip_Types.h.

6.1.3.14 FLS_SIZE_1BYTE

```
#define FLS_SIZE_1BYTE
```

the number of bytes uses to compare (1 byte).

Definition at line 143 of file C40_Ip_Types.h.

6.1.3.15 FLS_SIZE_2BYTE

```
#define FLS_SIZE_2BYTE
```

the number of bytes uses to compare (2 bytes).

Definition at line 148 of file C40_Ip_Types.h.

6.1.3.16 FLS_SIZE_4BYTE

```
#define FLS_SIZE_4BYTE
```

the number of bytes uses to compare (4 bytes).

Definition at line 153 of file C40_Ip_Types.h.

6.1.4 Types Reference

6.1.4.1 Fls_CompHandlerReturnType

```
typedef uint8 Fls_CompHandlerReturnType
```

return value of ecc checking function

Definition at line 266 of file C40_Ip_Types.h.

6.1.4.2 Fls_InstructionAddressType

```
typedef const uint8* Fls_InstructionAddressType
```

the instruction that generated the ECC

Definition at line 271 of file C40_Ip_Types.h.

6.1.4.3 Fls_DataAddressType

```
typedef uint32 Fls_DataAddressType
```

data address that caused the ECC error

Definition at line 276 of file C40_Ip_Types.h.

6.1.4.4 C40_StartFlashAccessNotifPtrType

```
typedef void(* C40_StartFlashAccessNotifPtrType) (void)
```

Fls Start Flash Access Notification Pointer Type.

Pointer type of Fls_StartFlashAccessNotif function

Definition at line 305 of file C40_Ip_Types.h.

6.1.4.5 C40_FinishedFlashAccessNotifPtrType

```
typedef void(* C40_FinishedFlashAccessNotifPtrType) (void)
```

Fls Finished Flash Access Notification Pointer Type.

Pointer type of Fls_FinishedFlashAccessNotif function

Definition at line 311 of file C40_Ip_Types.h.

6.1.5 Enum Reference

6.1.5.1 C40_Ip_InstanceType

```
enum C40_Ip_InstanceType
```

C40 hardware instance.

Enumerator

C40_IP_INSTANCE↩_0	C40 hardware instance 0

Definition at line 163 of file C40_Ip_Types.h.

6.1.5.2 C40_Ip_StatusType

enum C40_Ip_StatusType

Enumeration of checking status errors or not.

Enumerator

STATUS_C40_IP_SUCCESS	Successful job
STATUS_C40_IP_BUSY	IP is performing an operation
STATUS_C40_IP_ERROR	Error - general code
STATUS_C40_IP_ERROR_TIMEOUT	Error - exceeded timeout
STATUS_C40_IP_ERROR_INPUT_PARAM	Error - wrong input parameter
STATUS_C40_IP_ERROR_BLANK_CHECK	Error - selected memory area is not erased
STATUS_C40_IP_ERROR_PROGRAM_VERIFY	Error - selected memory area doesn't contain desired value
STATUS_C40_IP_ERROR_USER_TEST_BRE↩AK_SBC	Error - single bit correction
STATUS_C40_IP_ERROR_USER_TEST_BRE↩AK_DBD	Error - double bit detection
STATUS_C40_IP_SECTOR_UNPROTECTED	Checked sector is unlocked
STATUS_C40_IP_SECTOR_PROTECTED	Checked sector is locked

Definition at line 171 of file C40_Ip_Types.h.

6.1.5.3 C40_Ip_FlashBlocksNumberType

enum C40_Ip_FlashBlocksNumberType

Enumeration of Blocks of memory flash .

Enumerator

FLS_CODE_BLOCK_0	code block number 0

Enumerator

FLS_CODE_BLOCK_1	code block number 1
FLS_CODE_BLOCK_2	code block number 2
FLS_CODE_BLOCK_3	code block number 3
FLS_DATA_BLOCK	data block
FLS_BLOCK_UTEST	block Utest
FLS_BLOCK_INVALID	invalid block

Definition at line 189 of file C40_Ip_Types.h.

6.1.5.4 C40_Ip_FlashBreakPointsType

`enum C40_Ip_FlashBreakPointsType`

Enumeration breakpoints .

Enumerator

FLS_BREAKPOINTS_ON_DBD	Break on Double bit detection
FLS_BREAKPOINTS_ON_DBD_SBC	Break on both Double bit detection and Single bit correction
FLS_NO_BREAKPOINTS	No break at all

Definition at line 203 of file C40_Ip_Types.h.

6.1.5.5 C40_Ip_ArrayIntegritySequenceType

`enum C40_Ip_ArrayIntegritySequenceType`

Enumeration of Array Integrity Sequence(proprietary sequence or sequential) .

Enumerator

FLS_PROPRIETARY_SEQENCE	Array integrity sequence is proprietary sequence
FLS_SEQUENTIAL	Array integrity sequence is sequential

Definition at line 213 of file C40_Ip_Types.h.

6.1.5.6 C40_Ip_MarginOptionType

enum C40_Ip_MarginOptionType

Declarations of margin levels.

This is used to selects the margin level that is being checked. Implements : C40_Ip_MarginOptionType_Class

Enumerator

C40_MARGIN_LEVEL_PROGRAM	a programmed level
C40_MARGIN_LEVEL_ERASE	a erased level

Definition at line 225 of file C40_Ip_Types.h.

6.1.5.7 C40_Ip_UtestStateType

enum C40_Ip_UtestStateType

Declarations for flash suspend operation and resume operation and user test check state.

This is used to indicators for suspending state, resuming state and operation is broken by single bit correction or double bit detection. Implements : C40_Ip_UtestStateType

Enumerator

C40_IP_OK	Successful operation
C40_IP_SUS_NOTHING	No program/erase operation
C40_IP_PGM_WRITE	A program sequence in interlock write stage.
C40_IP_ERS_WRITE	An erase sequence in interlock write stage.
C40_IP_ERS_SUS_PGM_WRITE	An erase-suspend program sequence in interlock write stage.
C40_IP_PGM_SUS	The program operation is in suspend state
C40_IP_ERS_SUS	The erase operation is in suspend state
C40_IP_ERS_SUS_PGM_SUS	The erase-suspended program operation is in suspend state
C40_IP_USER_TEST_SUS	The UTest check operation is in suspend state C40_IP
C40_IP_RES_NOTHING	No suspended program/erase operation
C40_IP_RES_PGM	The program operation is resumed
C40_IP_RES_ERS	The erase operation is resumed
C40_IP_RES_ERS_PGM	The erase-suspended program operation is resumed
C40_IP_RES_USER_TEST	The UTest check operation is resumed C40_IP
C40_IP_USER_TEST_BREAK_SBC	The UTest check operation is broken by Single bit correction
C40_IP_USER_TEST_BREAK_DBD	The UTest check operation is broken by Double bit detection

Definition at line 238 of file C40_Ip_Types.h.

6.1.6 Function Reference

6.1.6.1 C40_Ip_Init()

```
C40_Ip_StatusType C40_Ip_Init (
            const C40_ConfigType * InitConfig )
```

Initializes the C40 module.

This function will initialize c40 module and clear all error flags.

Parameters

in	*InitConfig*	Pointer to the driver configuration structure.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	Initialization is success
STATUS_C40_IP_ERROR_TIMEOUT	Errors Timeout because wait for the Done bit long time

6.1.6.2 C40_Ip_Abort()

```
C40_Ip_StatusType C40_Ip_Abort (
            void  )
```

Abort a program or erase operation.

This function will abort a program or erase operation in user mode and clear all PGM, APGM, ERS, AERS, EHV, AEHV bits in MCR,AMCRS registers

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	The operation is successful.
STATUS_C40_IP_ERROR_TIMEOUT	the operation error because wait for the Done bit long time

6.1.6.3 C40_Ip_Read()

```
C40_Ip_StatusType C40_Ip_Read (
            uint32 LogicalAddress,
            uint32 Length,
            uint8 * DestAddressPtr )
```

This function fills data to DestAddressPtr.

This function fills data to DestAddressPtr with data from the specified address

Parameters

in	*LogicalAddress*	The start address of the area to be read.
in	*Length*	Read size
in	*DestAddressPtr*	Pointer to the destination of the read.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	Read performed successfully.
STATUS_C40_IP_ERROR_INPUT_PARAM	Input parameters are invalid.
STATUS_C40_IP_ERROR	There was an error while reading.

Precondition

The module has to be initialized and not busy.

6.1.6.4 C40_Ip_Compare()

```
C40_Ip_StatusType C40_Ip_Compare (
            uint32 LogicalAddress,
```

```
uint32 Length,
const uint8 * SourceAddressPtr )
```

Checks that there is the desired data at the specified address.

Checks that there is the desired data at the specified address. If the compare is intended to be a blank check, the SourceAddressPtr should be NULL.

Parameters

in	*LogicalAddress*	The start address of the area to be checked.
in	*Length*	Check size
in	*SourceAddressPtr*	Pointer to the data expected to be read.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	Read performed successfully.
STATUS_C40_IP_ERROR_INPUT_PARAM	Input parameters are invalid.
STATUS_C40_IP_ERROR	There was an error while reading.
STATUS_C40_IP_ERROR_PROGRAM_VERIFY	The expected data was not found completely at the specified address

Precondition

The module has to be initialized and not busy.

6.1.6.5 C40_Ip_GetLock()

```
C40_Ip_StatusType C40_Ip_GetLock (
            C40_Ip_VirtualSectorsType VirtualSector )
```

Returns the locking status of the selected sector.

Returns the locking status of the selected sector. This function shall cover all the address spaces available.

Parameters

in	*VirtualSector*	Sector to be checked for locking.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SECTOR_UNPROTECTED	Sector was not locked
STATUS_C40_IP_SECTOR_PROTECTED	Sector was locked
STATUS_C40_IP_ERROR	The requested sector is invalid

Precondition

The module has to be initialized and not busy.

6.1.6.6 C40_Ip_ClearLock()

```
C40_Ip_StatusType C40_Ip_ClearLock (
            C40_Ip_VirtualSectorsType VirtualSector,
            uint8 DomainIdValue )
```

Unlocks the selected sector for the requesting core if possible.

Unlocks the selected sector for the requesting core if possible. This function shall cover all the address spaces available.

Parameters

in	*VirtualSector*	Sector to be unlocked.
in	*DomainIdValue*	ID for the core that requests sector lock

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	Sector was unlocked successfully
STATUS_C40_IP_ERROR	The requested sector was unlocked by another core or the sector input is out of range

Precondition

The module has to be initialized and not busy.

6.1.6.7 C40_Ip_GetSectorNumberFromAddress()

```
C40_Ip_VirtualSectorsType C40_Ip_GetSectorNumberFromAddress (
            uint32 TargetAddress )
```

Get sector number from specified address.

Get sector number from specified address.

Parameters

in	*TargetAddress*	target address

Returns

C40_Ip_VirtualSectorsType

Return values

Address	of sector

Precondition

The module has to be initialized and not busy.

6.1.6.8 C40_Ip_GetBlockNumberFromAddress()

```
C40_Ip_FlashBlocksNumberType C40_Ip_GetBlockNumberFromAddress (
            uint32 TargetAddress )
```

Get block number from target address.

Get block number from target address

Parameters

in	*TargetAddress*	target address

Returns

C40_Ip_GetBlockNumberFromAddress

Return values

The	block number which contains the target address.

6.1.6.9 C40_Ip_CheckUserTestStatus()

```
C40_Ip_StatusType C40_Ip_CheckUserTestStatus (
            const C40_Ip_MisrType * MisrExpectedValues,
            C40_Ip_UtestStateType * TestResult )
```

Check the operation in user test mode.

This function will check the status array integrity check in user test mode.

Parameters

in	*MisrExpectedValues*	The MISR values calculated by the user to do comparison with MISR values generated by hardware.
out	*TestResult*	The value return the state of flash.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	The operation is successful
STATUS_C40_IP_ERROR	Operation failure status
STATUS_C40_IP_BUSY	In progress status
STATUS_C40_IP_ERROR_INPUT_PARAM	input parameters is invalid

Precondition

The module has to be initialized

6.1.6.10 C40_Ip_ArrayIntegrityCheck()

```
C40_Ip_StatusType C40_Ip_ArrayIntegrityCheck (
            uint32 SelectBlock,
            C40_Ip_ArrayIntegritySequenceType AddressSequence,
```

```
C40_Ip_FlashBreakPointsType BreakPoints,
const C40_Ip_MisrType * MisrSeedValues,
uint8 DomainIdValue )
```

Check the array integrity of the flash memory.

This function will check the array integrity of the flash via main interface. The user specified address sequence is used for array integrity reads and the operation is done on the specified blocks. The MISR values calculated by the hardware is compared to the values passed by the user, if they are not the same, then an error code is returned. User should call C40_Ip_CheckUserTestStatus to check the on-going status of this function. And once finish, it will do comparison between MISR values provided by user which is currently stored in 'pMisrExpectedValues' and MISR values generated by hardware and return an appropriate code according to this compared result.

Parameters

in	*SelectBlock*	Select the block base address for checking.
in	*AddressSequence*	Determine the address sequence to be used during array integrity checks.
in	*BreakPoints*	Specify an option to allow stopping the operation on errors.
in	*MisrSeedValues*	Value to be written in the MISR registers prior to the check
in	*DomainIdValue*	ID for the core that requests program sequence.

Returns

C40_Ip_StatusTypes

Return values

STATUS_C40_IP_SUCCESS	The operation is successful.
STATUS_C40_IP_BUSY	New operation cannot be performed while previous high voltage operation in progress.
STATUS_C40_IP_ERROR_INPUT_PARAM	Input parameters are invalid.
STATUS_C40_IP_ERROR	It's impossible to enable an operation
STATUS_C40_IP_ERROR_TIMEOUT	Errors Timeout because wait for the Done bit long time

Precondition

The module has to be initialized

6.1.6.11 C40_Ip_ArrayIntegrityCheckSuspend()

```
C40_Ip_StatusType C40_Ip_ArrayIntegrityCheckSuspend (
            void  )
```

Suspend an on-going array integrity check.

This function will check if there is an on-going array integrity check of the flash and suspend it via main interface.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_Ip_SUCCESS	array integrity check suspending was successful.
STATUS_C40_IP_ERROR	there is no suspended array integrity check or not successfully resumed.

Precondition

The module has to be initialized

6.1.6.12 C40_Ip_ArrayIntegrityCheckResume()

```
C40_Ip_StatusType C40_Ip_ArrayIntegrityCheckResume (
            void  )
```

Resume the previous suspend operation.

This function will check if there is an on-going array integrity check of the flash being suspended and resume it via main interface.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	array integrity check resuming was successful.
STATUS_C40_IP_ERROR	there is no suspended array integrity check or not successfully resumed.

Precondition

The module has to be initialized

6.1.6.13 C40_Ip_UserMarginReadCheck()

```
C40_Ip_StatusType C40_Ip_UserMarginReadCheck (
            uint32 SelectBlock,
```

```
C40_Ip_FlashBreakPointsType BreakPoints,
C40_Ip_MarginOptionType MarginLevel,
const C40_Ip_MisrType * MisrSeedValues,
uint8 DomainIdValue )
```

Check the user margin read of the flash memory.

This function will check the user margin reads of the flash via main interface. The user specified margin level is used for reads and the operation is done on the specified blocks. The MISR values calculated by the hardware are compared to the values passed by the user, if they are not the same, then an error code is returned. User should call C40_Ip_CheckUserTestStatus to check the on-going status of this function. And once finish, it will do comparison between MISR values provided by user which are currently stored in 'pMisrExpectedValues,' and MISR values generated by hardware and return an appropriate code according to this compared result.

Parameters

in	*SelectBlock*	Select the block base address for checking.
in	*BreakPoints*	An option to allow stopping the operation on errors.
in	*MarginLevel*	The margin level to be used during margin read checks.
in	*MisrSeedValues*	Value to be written in the MISR registers prior to the check
in	*DomainIdValue*	ID for the core that requests program sequence.

Returns

C40_Ip_StatusType

Return values

STATUS_C40_IP_SUCCESS	The operation is successful.
STATUS_C40_IP_BUSY	New operation cannot be performed while previous high voltage operation in progress.
STATUS_C40_IP_ERROR_INPUT_PARAM	Input parameters are invalid.
STATUS_C40_IP_ERROR	It's impossible to enable an operation
STATUS_C40_IP_ERROR_TIMEOUT	Errors Timeout because wait for the Done bit long time

Precondition

The module has to be initialized

6.1.6.14 C40_Ip_GetFailedAddress()

```
uint32 C40_Ip_GetFailedAddress (
            void  )
```

Get the failing address in memory.

This function will get the failing address in the event of ECC event error, Single Bit Correction, as well as providing the address of a failure that may have occurred in a program/erase operation.

Returns

uint32

Return values

Return	the address is failed in the event or single bit correction.

Precondition

The module has to be initialized

6.1.6.15 C40_Ip_SetAsyncMode()

```
void C40_Ip_SetAsyncMode (
             const boolean Async )
```

Set synch/Async at IP layer base on the Async of HLD.

This function will change C40_Ip_Async value at IP layer. Its param base on the Async of HLD. Thanks for this param, writing and erasing will operate at synch or Async mode.

Precondition

The module has to be initialized

6.1.6.16 C40_Ip_DataErrorSuppression()

```
void C40_Ip_DataErrorSuppression (
             void  )
```

Setup the ECC error handling on data flash block.

Parameters

none	

Returns

none

6.1.6.17 C40_Ip_GetLockProtect()

```
uint32 C40_Ip_GetLockProtect (
            C40_Ip_VirtualSectorsType VirtualSector )
```

Read lock bit status of flash sectors.

Parameters

in	*VirtualSector*	Sector to be checked

Returns

uint32 Lock bit status value

6.1.6.18 C40_Ip_SetLockProtect()

```
void C40_Ip_SetLockProtect (
            C40_Ip_VirtualSectorsType VirtualSector )
```

Locks the selected sector for the requesting core.

Parameters

in	*VirtualSector*	Sector to be locked

Returns

none

6.1.6.19 C40_Ip_ClearLockProtect()

```
void C40_Ip_ClearLockProtect (
            C40_Ip_VirtualSectorsType VirtualSector )
```

Unlocks the selected sector for the requesting core.

Parameters

in	*VirtualSector*	Sector to be unlocked

Return values

none	

6.1.6.20 C40_Ip_CheckLockDomainID_CheckRegister()

```
void C40_Ip_CheckLockDomainID_CheckRegister (
            C40_Ip_VirtualSectorsType VirtualSector,
            uint32 * CheckRegister,
            uint32 * TempLockMasterRegister )
```

Read and check the lock domain ID for flash sectors.

Parameters

in	*VirtualSector*	Sector to be checked
in,out	*CheckRegister*	Lock status value of the sector
in,out	*TempLockMasterRegister*	The address of the register that contain domain ID of the master currently acquiring the lock bit.

Returns

none

6.2 MEM_43_INFLS Driver

6.2.1 Detailed Description implements Mem_43_InFls_IPW.h_Artifact

implements Mem_43_InFls_Types.h_Artifact

Data Structures

- struct Mem_43_InFls_InternalUnitType
  Mem internal unit type. *More...*
- struct Mem_43_InFls_MemDeviceType
  Mem device config type. *More...*
- struct Mem_43_InFls_SectorBatchType
  Sector Batch Type. *More...*
- struct Mem_43_InFls_MemInstanceType
  Mem Instance Type. *More...*
- struct Mem_43_InFls_ConfigType
  Mem Configuration Type. *More...*
- struct Mem_43_InFls_JobRuntimeInfoType
  Mem job runtime information Type. *More...*

Macros

- #define MEM_43_INFLS_MODULE_ID
  AUTOSAR module identification.
- #define MEM_43_INFLS_INSTANCE_ID
  AUTOSAR module instance identification.
- #define MEM_43_INFLS_E_UNINIT
  API service called without module initialization.
- #define MEM_43_INFLS_E_PARAM_POINTER
  API service called with NULL pointer.
- #define MEM_43_INFLS_E_PARAM_ADDRESS
  API service called with an invalid address.
- #define MEM_43_INFLS_E_PARAM_LENGTH
  API service called with an invalid length.
- #define MEM_43_INFLS_E_PARAM_INSTANCE_ID
  API service called with an invalid driver instance ID.
- #define MEM_43_INFLS_E_JOB_PENDING
  API service called while a job request is still in progress.
- #define MEM_43_INFLS_E_OK
  API service called without errors.
- #define MEM_43_INFLS_DEINIT_ID
  Service ID of function Mem_43_InFls_DeInit.
- #define MEM_43_INFLS_INIT_ID
  Service ID of function Mem_43_InFls_Init.

- #define MEM_43_INFLS_GETVERSIONINFO_ID

  Service ID of function Mem_43_InFls_GetVersionInfo.
- #define MEM_43_INFLS_GETJOBRESULT_ID

  Service ID of function Mem_43_InFls_GetJobResult.
- #define MEM_43_INFLS_PROPAGATEERROR_ID

  Service ID of function Mem_43_InFls_PropagateError.
- #define MEM_43_INFLS_SUSPEND_ID

  Service ID of function Mem_43_InFls_Suspend.
- #define MEM_43_INFLS_RESUME_ID

  Service ID of function Mem_43_InFls_Resume.
- #define MEM_43_INFLS_READ_ID

  Service ID of function Mem_43_InFls_Read.
- #define MEM_43_INFLS_WRITE_ID

  Service ID of function Mem_43_InFls_Write.
- #define MEM_43_INFLS_ERASE_ID

  Service ID of function Mem_43_InFls_Erase.
- #define MEM_43_INFLS_BLANKCHECK_ID

  Service ID of function Mem_43_InFls_BlankCheck.
- #define MEM_43_INFLS_HWSPECIFICSERVICE_ID

  Service ID of function Mem_43_InFls_HwSpecificService.
- #define MEM_43_INFLS_MAINFUNCTION_ID

  Service ID of function Mem_43_InFls_MainFunction.
- #define MEM_43_INFLS_JOB_FLAG_NONE

  Initial value.
- #define MEM_43_INFLS_JOB_FLAG_STARTED

  Indicates that new job has been accepted.

Types Reference

- typedef uint32 Mem_43_InFls_AddressType

  Mem Address Type.
- typedef uint32 Mem_43_InFls_InstanceIdType

  Mem Instance Id Type.
- typedef uint32 Mem_43_InFls_LengthType

  Mem Length Type.
- typedef uint8 Mem_43_InFls_DataType

  Mem Data Type.
- typedef uint16 Mem_43_InFls_CrcType

  Mem CRC Type.
- typedef uint32 Mem_43_InFls_HwServiceIdType

  Mem Hardware Service Id Type.
- typedef C40_ConfigType Mem_43_InFls_InternalConfigType

  Mem Internal Flash Type.

Enum Reference

- enum Mem_43_InFls_JobResultType

  Mem job result type Mem_43_InFlsJobResultType_enumeration.
- enum Mem_43_InFls_JobType

  Type of job currently executed by Mem_43_InFls_MainFunction.

Function Reference

- void Mem_43_InFls_Init (const Mem_43_InFls_ConfigType *ConfigPtr)

  The function initializes Mem_43_InFls module.
- void Mem_43_InFls_DeInit (void)

  The function de-initializes the Mem_43_InFls module.
- void Mem_43_InFls_GetVersionInfo (Std_VersionInfoType *VersionInfoPtr)

  Return the version information of the Mem module.
- Mem_43_InFls_JobResultType Mem_43_InFls_GetJobResult (Mem_43_InFls_InstanceIdType InstanceId)

  Returns the result of the most recent job.
- void Mem_43_InFls_Suspend (Mem_43_InFls_InstanceIdType InstanceId)

  Suspends active memory operation using hardware mechanism.
- void Mem_43_InFls_Resume (Mem_43_InFls_InstanceIdType InstanceId)

  Resumes suspended memory operation using hardware mechanism.
- void Mem_43_InFls_PropagateError (Mem_43_InFls_InstanceIdType InstanceId)

  Propagates an ECC error to the memory upper layers.
- Std_ReturnType Mem_43_InFls_Read (Mem_43_InFls_InstanceIdType InstanceId, Mem_43_InFls_AddressType SourceAddress, Mem_43_InFls_DataType *DestinationDataPtr, Mem_43_InFls_LengthType Length)

  Reads from flash memory.
- Std_ReturnType Mem_43_InFls_Write (Mem_43_InFls_InstanceIdType InstanceId, Mem_43_InFls_AddressType TargetAddress, const Mem_43_InFls_DataType *SourceDataPtr, Mem_43_InFls_LengthType Length)

  Writes to flash memory.
- Std_ReturnType Mem_43_InFls_Erase (Mem_43_InFls_InstanceIdType InstanceId, Mem_43_InFls_AddressType TargetAddress, Mem_43_InFls_LengthType Length)

  Erase one or more complete flash sectors.
- Std_ReturnType Mem_43_InFls_BlankCheck (Mem_43_InFls_InstanceIdType InstanceId, Mem_43_InFls_AddressTy TargetAddress, Mem_43_InFls_LengthType Length)

  Verify whether a given memory area has been erased but not (yet) programmed.
- Std_ReturnType Mem_43_InFls_HwSpecificService (Mem_43_InFls_InstanceIdType InstanceId, Mem_43_InFls_HwS HwServiceId, Mem_43_InFls_DataType *DataPtr, Mem_43_InFls_LengthType *LengthPtr)

  Trigger a hardware specific service.
- void Mem_43_InFls_MainFunction (void)

  Service to handle the requested jobs and the internal management operations.
- Std_ReturnType Mem_43_InFls_IPW_Init (void)

  Initialize the hardware resources.
- Mem_43_InFls_JobResultType Mem_43_InFls_IPW_Read (uint32 InstanceIndex, Mem_43_InFls_JobRuntimeInfoTy *JobInfo)

  IP wrapper read function.

- Mem_43_InFls_JobResultType Mem_43_InFls_IPW_BlankCheck (uint32 InstanceIndex, Mem_43_InFls_JobRuntime
  ∗JobInfo)

  IP wrapper blank check function.
- Mem_43_InFls_JobResultType Mem_43_InFls_IPW_Write (uint32 InstanceIndex, Mem_43_InFls_JobRuntimeInfoTy
  ∗JobInfo)

  IP wrapper write function.
- Mem_43_InFls_JobResultType Mem_43_InFls_IPW_Erase (uint32 InstanceIndex, Mem_43_InFls_JobRuntimeInfoTy
  ∗JobInfo)

  IP wrapper erase function.
- Mem_43_InFls_JobResultType Mem_43_InFls_IPW_GetJobResult (uint32 InstanceIndex, Mem_43_InFls_JobType
  JobType)

  Returns synchronously the result of the last job.
- Mem_43_InFls_JobResultType Mem_43_InFls_IPW_AbortSuspended (uint32 InstanceIndex, Mem_43_InFls_JobTyp
  JobType)

  Abort a suspended hardware job to prepare for a new job.

6.2.2 Data Structure Documentation

6.2.2.1 struct Mem_43_InFls_InternalUnitType

Mem internal unit type.

Mem internal unit config data structure Mem_43_InFls_InternalUnitType_struct

Definition at line 356 of file Mem_43_InFls_Types.h.

6.2.2.2 struct Mem_43_InFls_MemDeviceType

Mem device config type.

Mem device config data structure Mem_43_InFls_MemDeviceType_struct

Definition at line 366 of file Mem_43_InFls_Types.h.

6.2.2.3 struct Mem_43_InFls_SectorBatchType

Sector Batch Type.

Sector Batch data structure for group of identical sectors Note: burst sizes equal to normal sizes in case burst disabled Mem_43_InFls_SectorBatchType_struct

Definition at line 378 of file Mem_43_InFls_Types.h.

6.2.2.4 struct Mem_43_InFls_MemInstanceType

Mem Instance Type.

Mem Instance data structure Mem_43_InFls_MemInstanceType_struct

Definition at line 395 of file Mem_43_InFls_Types.h.

6.2.2.5 struct Mem_43_InFls_ConfigType

Mem Configuration Type.

Mem module initialization data structure

Definition at line 408 of file Mem_43_InFls_Types.h.

6.2.2.6 struct Mem_43_InFls_JobRuntimeInfoType

Mem job runtime information Type.

This structure contains runtime information the current processing job of each Mem instance. Mem_43_InFls_↩ JobRuntimeInfoType_struct

Definition at line 424 of file Mem_43_InFls_Types.h.

6.2.3 Macro Definition Documentation

6.2.3.1 MEM_43_INFLS_MODULE_ID

```
#define MEM_43_INFLS_MODULE_ID
```

AUTOSAR module identification.

Definition at line 118 of file Mem_43_InFls_Types.h.

6.2.3.2 MEM_43_INFLS_INSTANCE_ID

```
#define MEM_43_INFLS_INSTANCE_ID
```

AUTOSAR module instance identification.

Definition at line 123 of file Mem_43_InFls_Types.h.

6.2.3.3 MEM_43_INFLS_E_UNINIT

```
#define MEM_43_INFLS_E_UNINIT
```

API service called without module initialization.

Development error codes (passed to DET)

Definition at line 133 of file Mem_43_InFls_Types.h.

6.2.3.4 MEM_43_INFLS_E_PARAM_POINTER

```
#define MEM_43_INFLS_E_PARAM_POINTER
```

API service called with NULL pointer.

Definition at line 139 of file Mem_43_InFls_Types.h.

6.2.3.5 MEM_43_INFLS_E_PARAM_ADDRESS

```
#define MEM_43_INFLS_E_PARAM_ADDRESS
```

API service called with an invalid address.

Definition at line 145 of file Mem_43_InFls_Types.h.

6.2.3.6 MEM_43_INFLS_E_PARAM_LENGTH

```
#define MEM_43_INFLS_E_PARAM_LENGTH
```

API service called with an invalid length.

Definition at line 151 of file Mem_43_InFls_Types.h.

6.2.3.7 MEM_43_INFLS_E_PARAM_INSTANCE_ID

```
#define MEM_43_INFLS_E_PARAM_INSTANCE_ID
```

API service called with an invalid driver instance ID.

Definition at line 157 of file Mem_43_InFls_Types.h.

6.2.3.8 MEM_43_INFLS_E_JOB_PENDING

```
#define MEM_43_INFLS_E_JOB_PENDING
```

API service called while a job request is still in progress.

Definition at line 163 of file Mem_43_InFls_Types.h.

6.2.3.9 MEM_43_INFLS_E_OK

```
#define MEM_43_INFLS_E_OK
```

API service called without errors.

Definition at line 169 of file Mem_43_InFls_Types.h.

6.2.3.10 MEM_43_INFLS_DEINIT_ID

```
#define MEM_43_INFLS_DEINIT_ID
```

Service ID of function Mem_43_InFls_DeInit.

END Development error codes All service IDs (passed to DET)

Definition at line 182 of file Mem_43_InFls_Types.h.

6.2.3.11 MEM_43_INFLS_INIT_ID

```
#define MEM_43_INFLS_INIT_ID
```

Service ID of function Mem_43_InFls_Init.

Definition at line 187 of file Mem_43_InFls_Types.h.

6.2.3.12 MEM_43_INFLS_GETVERSIONINFO_ID

```
#define MEM_43_INFLS_GETVERSIONINFO_ID
```

Service ID of function Mem_43_InFls_GetVersionInfo.

Definition at line 192 of file Mem_43_InFls_Types.h.

6.2.3.13 MEM_43_INFLS_GETJOBRESULT_ID

```
#define MEM_43_INFLS_GETJOBRESULT_ID
```

Service ID of function Mem_43_InFls_GetJobResult.

Definition at line 197 of file Mem_43_InFls_Types.h.

6.2.3.14 MEM_43_INFLS_PROPAGATEERROR_ID

```
#define MEM_43_INFLS_PROPAGATEERROR_ID
```

Service ID of function Mem_43_InFls_PropagateError.

Definition at line 202 of file Mem_43_InFls_Types.h.

6.2.3.15 MEM_43_INFLS_SUSPEND_ID

```
#define MEM_43_INFLS_SUSPEND_ID
```

Service ID of function Mem_43_InFls_Suspend.

Definition at line 207 of file Mem_43_InFls_Types.h.

6.2.3.16 MEM_43_INFLS_RESUME_ID

```
#define MEM_43_INFLS_RESUME_ID
```

Service ID of function Mem_43_InFls_Resume.

Definition at line 212 of file Mem_43_InFls_Types.h.

6.2.3.17 MEM_43_INFLS_READ_ID

```
#define MEM_43_INFLS_READ_ID
```

Service ID of function Mem_43_InFls_Read.

Definition at line 219 of file Mem_43_InFls_Types.h.

6.2.3.18 MEM_43_INFLS_WRITE_ID

```
#define MEM_43_INFLS_WRITE_ID
```

Service ID of function Mem_43_InFls_Write.

Definition at line 224 of file Mem_43_InFls_Types.h.

6.2.3.19 MEM_43_INFLS_ERASE_ID

```
#define MEM_43_INFLS_ERASE_ID
```

Service ID of function Mem_43_InFls_Erase.

Definition at line 229 of file Mem_43_InFls_Types.h.

6.2.3.20 MEM_43_INFLS_BLANKCHECK_ID

```
#define MEM_43_INFLS_BLANKCHECK_ID
```

Service ID of function Mem_43_InFls_BlankCheck.

Definition at line 234 of file Mem_43_InFls_Types.h.

6.2.3.21 MEM_43_INFLS_HWSPECIFICSERVICE_ID

```
#define MEM_43_INFLS_HWSPECIFICSERVICE_ID
```

Service ID of function Mem_43_InFls_HwSpecificService.

Definition at line 239 of file Mem_43_InFls_Types.h.

6.2.3.22 MEM_43_INFLS_MAINFUNCTION_ID

```
#define MEM_43_INFLS_MAINFUNCTION_ID
```

Service ID of function Mem_43_InFls_MainFunction.

Definition at line 246 of file Mem_43_InFls_Types.h.

6.2.3.23 MEM_43_INFLS_JOB_FLAG_NONE

```
#define MEM_43_INFLS_JOB_FLAG_NONE
```

Initial value.

END All service IDs

Definition at line 256 of file Mem_43_InFls_Types.h.

6.2.3.24 MEM_43_INFLS_JOB_FLAG_STARTED

```
#define MEM_43_INFLS_JOB_FLAG_STARTED
```

Indicates that new job has been accepted.

Definition at line 261 of file Mem_43_InFls_Types.h.

6.2.4 Types Reference

6.2.4.1 Mem_43_InFls_AddressType

```
typedef uint32 Mem_43_InFls_AddressType
```

Mem Address Type.

Physical memory device address type

Definition at line 272 of file Mem_43_InFls_Types.h.

6.2.4.2 Mem_43_InFls_InstanceIdType

```
typedef uint32 Mem_43_InFls_InstanceIdType
```

Mem Instance Id Type.

Mem Instance Id Type

Definition at line 279 of file Mem_43_InFls_Types.h.

6.2.4.3 Mem_43_InFls_LengthType

```
typedef uint32 Mem_43_InFls_LengthType
```

Mem Length Type.

Physical memory device length type

Definition at line 286 of file Mem_43_InFls_Types.h.

6.2.4.4 Mem_43_InFls_DataType

```
typedef uint8 Mem_43_InFls_DataType
```

Mem Data Type.

Read data user buffer type

Definition at line 293 of file Mem_43_InFls_Types.h.

6.2.4.5 Mem_43_InFls_CrcType

```
typedef uint16 Mem_43_InFls_CrcType
```

Mem CRC Type.

CRC computed over config set Mem_43_InFls_CrcType_typedef

Definition at line 300 of file Mem_43_InFls_Types.h.

6.2.4.6 Mem_43_InFls_HwServiceIdType

```
typedef uint32 Mem_43_InFls_HwServiceIdType
```

Mem Hardware Service Id Type.

Hardware specific service request identifier type

Definition at line 307 of file Mem_43_InFls_Types.h.

6.2.4.7 Mem_43_InFls_InternalConfigType

```
typedef C40_ConfigType Mem_43_InFls_InternalConfigType
```

Mem Internal Flash Type.

Configuration structure of internal flash.

Definition at line 313 of file Mem_43_InFls_Types.h.

6.2.5 Enum Reference

6.2.5.1 Mem_43_InFls_JobResultType

```
enum Mem_43_InFls_JobResultType
```

Mem job result type Mem_43_InFlsJobResultType_enumeration.

Definition at line 323 of file Mem_43_InFls_Types.h.

6.2.5.2 Mem_43_InFls_JobType

```
enum Mem_43_InFls_JobType
```

Type of job currently executed by Mem_43_InFls_MainFunction.

Definition at line 336 of file Mem_43_InFls_Types.h.

6.2.6 Function Reference

6.2.6.1 Mem_43_InFls_Init()

```
void Mem_43_InFls_Init (
            const Mem_43_InFls_ConfigType * ConfigPtr )
```

The function initializes Mem_43_InFls module.

6.2.6.2 Mem_43_InFls_DeInit()

```
void Mem_43_InFls_DeInit (
            void  )
```

The function de-initializes the Mem_43_InFls module.

6.2.6.3 Mem_43_InFls_GetVersionInfo()

```
void Mem_43_InFls_GetVersionInfo (
            Std_VersionInfoType * VersionInfoPtr )
```

Return the version information of the Mem module.

6.2.6.4 Mem_43_InFls_GetJobResult()

```
Mem_43_InFls_JobResultType Mem_43_InFls_GetJobResult (
            Mem_43_InFls_InstanceIdType InstanceId )
```

Returns the result of the most recent job.

6.2.6.5 Mem_43_InFls_Suspend()

```
void Mem_43_InFls_Suspend (
            Mem_43_InFls_InstanceIdType InstanceId )
```

Suspends active memory operation using hardware mechanism.

6.2.6.6 Mem_43_InFls_Resume()

```
void Mem_43_InFls_Resume (
            Mem_43_InFls_InstanceIdType InstanceId )
```

Resumes suspended memory operation using hardware mechanism.

6.2.6.7 Mem_43_InFls_PropagateError()

```
void Mem_43_InFls_PropagateError (
            Mem_43_InFls_InstanceIdType InstanceId )
```

Propagates an ECC error to the memory upper layers.

6.2.6.8 Mem_43_InFls_Read()

```
Std_ReturnType Mem_43_InFls_Read (
            Mem_43_InFls_InstanceIdType InstanceId,
            Mem_43_InFls_AddressType SourceAddress,
            Mem_43_InFls_DataType * DestinationDataPtr,
            Mem_43_InFls_LengthType Length )
```

Reads from flash memory.

6.2.6.9 Mem_43_InFls_Write()

```
Std_ReturnType Mem_43_InFls_Write (
            Mem_43_InFls_InstanceIdType InstanceId,
            Mem_43_InFls_AddressType TargetAddress,
            const Mem_43_InFls_DataType * SourceDataPtr,
            Mem_43_InFls_LengthType Length )
```

Writes to flash memory.

6.2.6.10 Mem_43_InFls_Erase()

```
Std_ReturnType Mem_43_InFls_Erase (
            Mem_43_InFls_InstanceIdType InstanceId,
            Mem_43_InFls_AddressType TargetAddress,
            Mem_43_InFls_LengthType Length )
```

Erase one or more complete flash sectors.

6.2.6.11 Mem_43_InFls_BlankCheck()

```
Std_ReturnType Mem_43_InFls_BlankCheck (
            Mem_43_InFls_InstanceIdType InstanceId,
            Mem_43_InFls_AddressType TargetAddress,
            Mem_43_InFls_LengthType Length )
```

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

6.2.6.12 Mem_43_InFls_HwSpecificService()

```
Std_ReturnType Mem_43_InFls_HwSpecificService (
            Mem_43_InFls_InstanceIdType InstanceId,
            Mem_43_InFls_HwServiceIdType HwServiceId,
            Mem_43_InFls_DataType * DataPtr,
            Mem_43_InFls_LengthType * LengthPtr )
```

Trigger a hardware specific service.

6.2.6.13 Mem_43_InFls_MainFunction()

```
void Mem_43_InFls_MainFunction (
            void  )
```

Service to handle the requested jobs and the internal management operations.

6.2.6.14 Mem_43_InFls_IPW_Init()

```
Std_ReturnType Mem_43_InFls_IPW_Init (
            void  )
```

Initialize the hardware resources.

Returns

Std_ReturnType

6.2.6.15 Mem_43_InFls_IPW_Read()

```
Mem_43_InFls_JobResultType Mem_43_InFls_IPW_Read (
            uint32 InstanceIndex,
            Mem_43_InFls_JobRuntimeInfoType * JobInfo )
```

IP wrapper read function.

Parameters

in	*InstanceIndex*	ID of the related memory driver instance.
in	*JobInfo*	Job runtime information

Returns

Mem_43_InFls_JobResultType

6.2.6.16 Mem_43_InFls_IPW_BlankCheck()

```
Mem_43_InFls_JobResultType Mem_43_InFls_IPW_BlankCheck (
            uint32 InstanceIndex,
            Mem_43_InFls_JobRuntimeInfoType * JobInfo )
```

IP wrapper blank check function.

Parameters

in	*InstanceIndex*	ID of the related memory driver instance.
in	*JobInfo*	Job runtime information

Returns

Mem_43_InFls_JobResultType

6.2.6.17 Mem_43_InFls_IPW_Write()

```
Mem_43_InFls_JobResultType Mem_43_InFls_IPW_Write (
            uint32 InstanceIndex,
            Mem_43_InFls_JobRuntimeInfoType * JobInfo )
```

IP wrapper write function.

Parameters

in	*InstanceIndex*	ID of the related memory driver instance.
in	*JobInfo*	Job runtime information

Returns

Mem_43_InFls_JobResultType

6.2.6.18 Mem_43_InFls_IPW_Erase()

```
Mem_43_InFls_JobResultType Mem_43_InFls_IPW_Erase (
            uint32 InstanceIndex,
            Mem_43_InFls_JobRuntimeInfoType * JobInfo )
```

IP wrapper erase function.

Parameters

in	*InstanceIndex*	ID of the related memory driver instance.
in	*JobInfo*	Job runtime information

Returns

Mem_43_InFls_JobResultType

6.2.6.19 Mem_43_InFls_IPW_GetJobResult()

```
Mem_43_InFls_JobResultType Mem_43_InFls_IPW_GetJobResult (
            uint32 InstanceIndex,
            Mem_43_InFls_JobType JobType )
```

Returns synchronously the result of the last job.

Parameters

in	*InstanceIndex*	ID of the related memory driver instance.
in	*JobType*	Job Erase or Write.

Returns

Mem_43_InFls_JobResultType

6.2.6.20 Mem_43_InFls_IPW_AbortSuspended()

```
Mem_43_InFls_JobResultType Mem_43_InFls_IPW_AbortSuspended (
            uint32 InstanceIndex,
            Mem_43_InFls_JobType JobType )
```

Abort a suspended hardware job to prepare for a new job.

Parameters

in	*InstanceIndex*	ID of the related memory driver instance.
in	*JobType*	Job Erase, Write, or Read.

Returns

Mem_43_InFls_JobResultType

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.