User Manual

for S32K1_S32M24X MEM_43_INFLS Driver

Document Number: UM2MEM_43_INFLSASRR21-11 Rev0000R2.0.0 Rev. 1.0

Chapter 1

Revision History

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

Chapter 2

Introduction

This User Manual describes NXP Semiconductors' AUTOSAR Mem_43_INFLS driver for S32K1_S32M24X.

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 Driver Software Specification Document (version R21-11) [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:

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

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

All of the above microcontroller devices are collectively named as S32K1_S32M24X. Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power

2.2 Overview

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

AUTOSAR:

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

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

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

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
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	S32K1xx Series Reference Manual, Rev. 14, 09/2021
		S32M24x Reference Manual, Rev. 2 Draft A, 05/2023
3	Datasheet	S32K1xx Data Sheet, Rev. 14, 08/2021
		S32M2xx Data Sheet, Supports S32M24x and S32M27x, Rev. 3 Draft A, 05/2023
4	Errata	S32K116_0N96V Rev. 22/OCT/2021
		S32K118_0N97V Rev. 22/OCT/2021
		S32K142_0N33V Rev. 22/OCT/2021
		S32K144_0N57U Rev. 22/OCT/2021
		S32K144W_0P64A Rev. 22/OCT/2021
		S32K146_0N73V Rev. 22/OCT/2021
		S32K148_0N20V Rev. 22/OCT/2021
		S32M244_P64A+P73G, Rev. 0

Chapter 3

Driver

3.1 Requirements

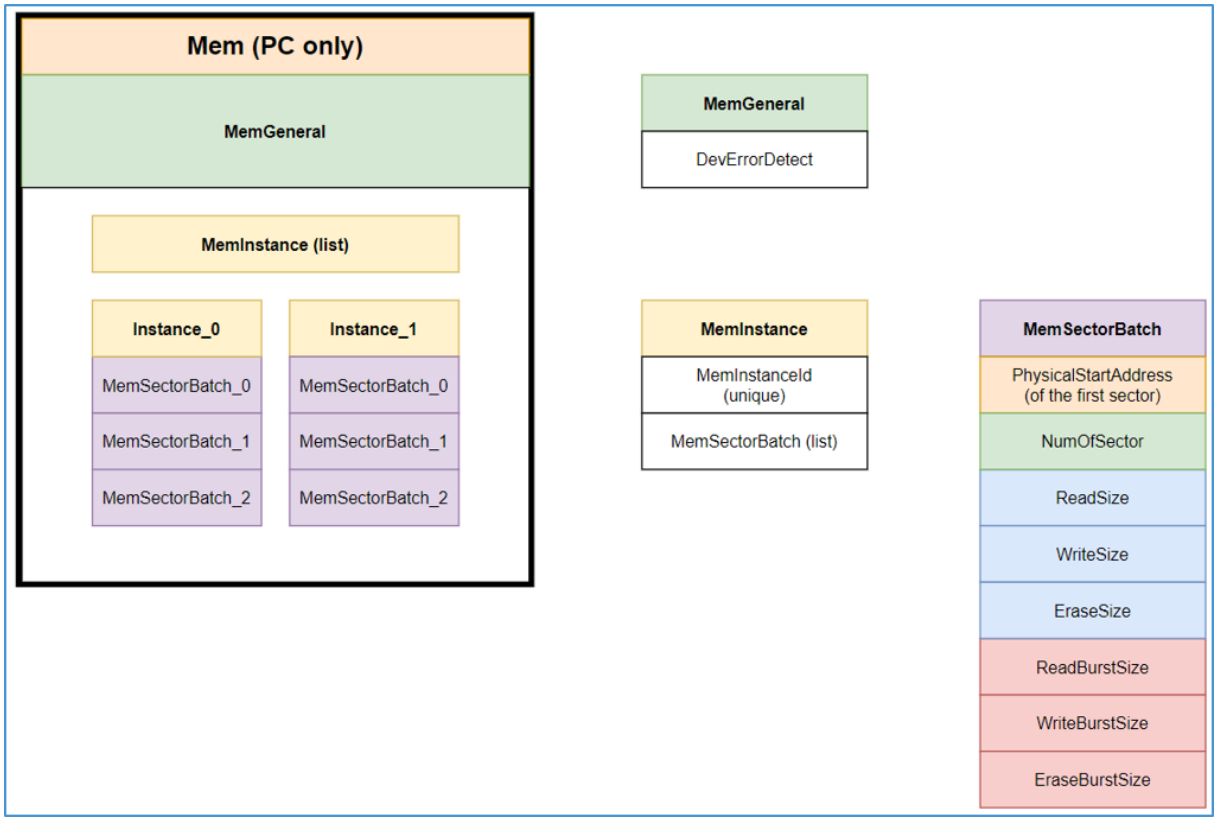
Requirements for this driver are detailed in the Autosar Driver Software Specification document (See Table Reference List).

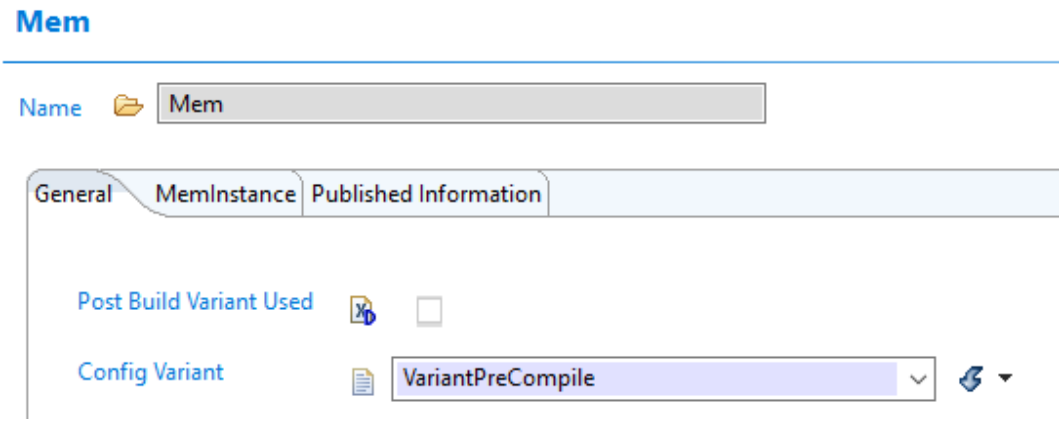
3.2 Driver Design Summary

3.2.1 the Mem_43_INFLS driver architecture

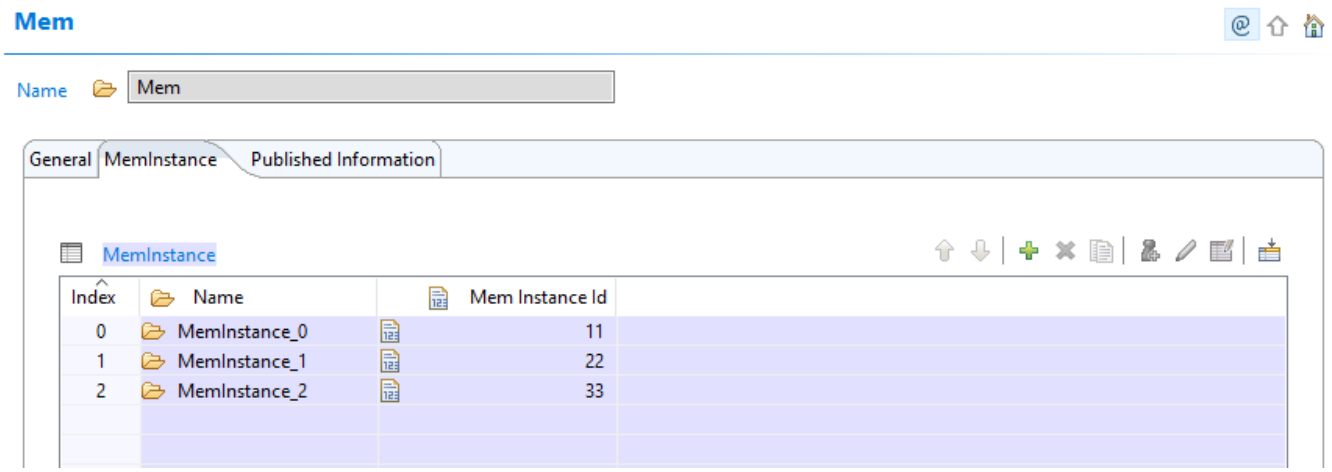
This section describes the detail for a high-level overview of Mem_43_INFLS driver .

- 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
  - Distinguished by a memory instance ID

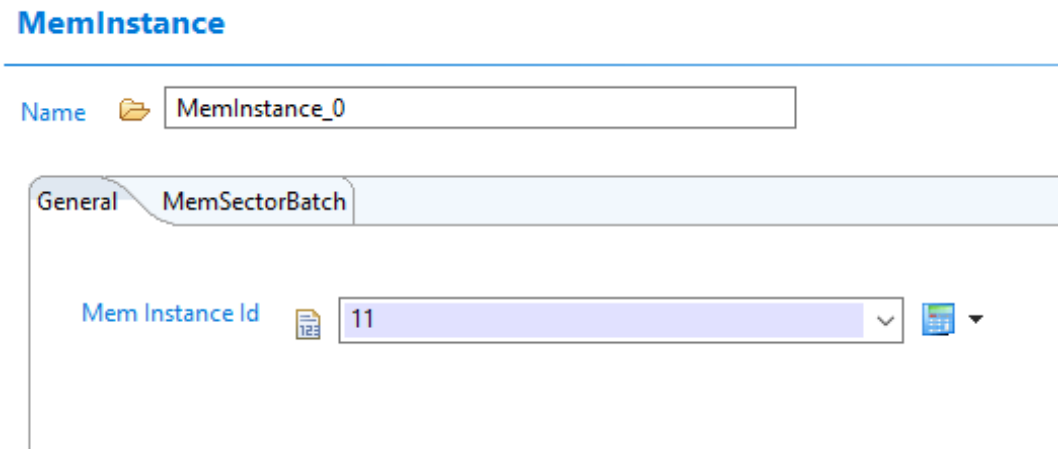


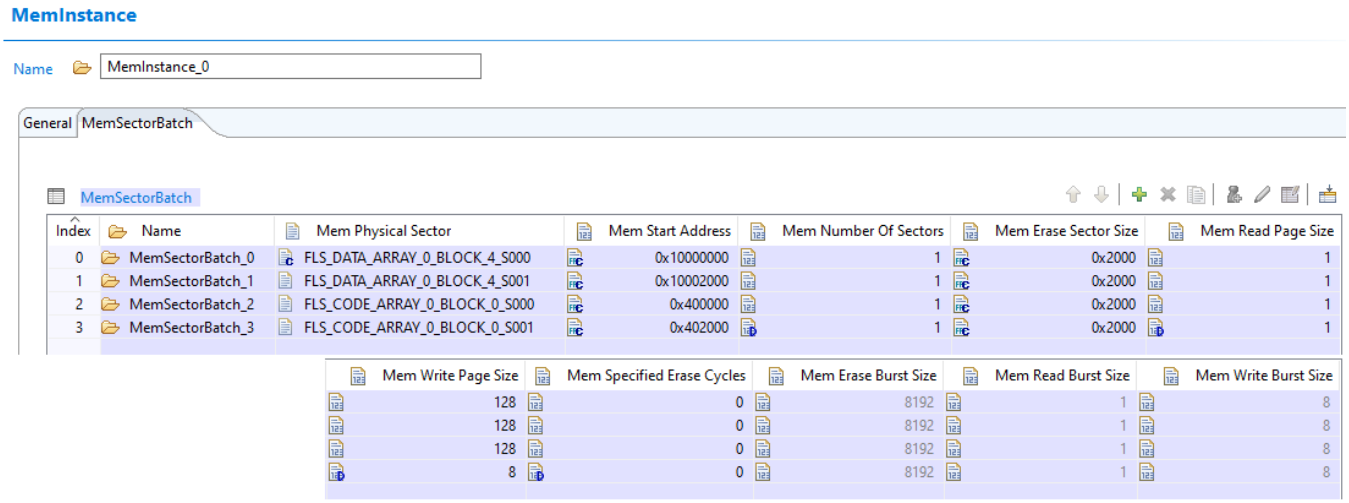
- For example: the OTA software update use case
  - Multiple memory devices of the same type are used to expand the memory resources(increase storage capacity)
  - Switching of memory address translation tables for memory swap use cases(while keep the same physical addresses)

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





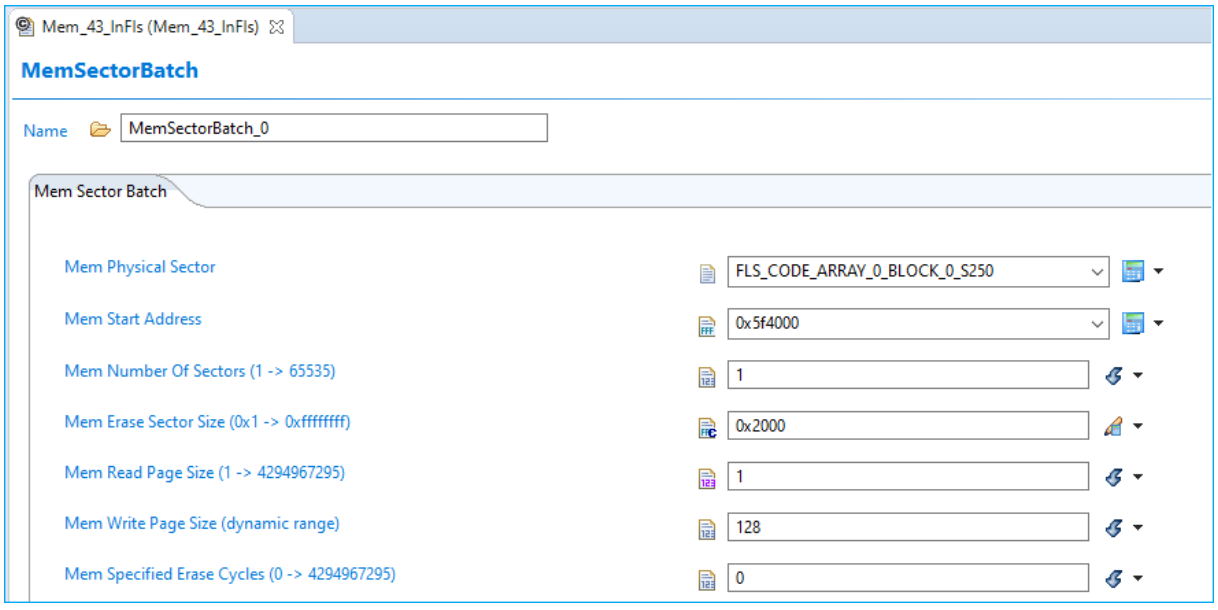
As you can see:

- 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.

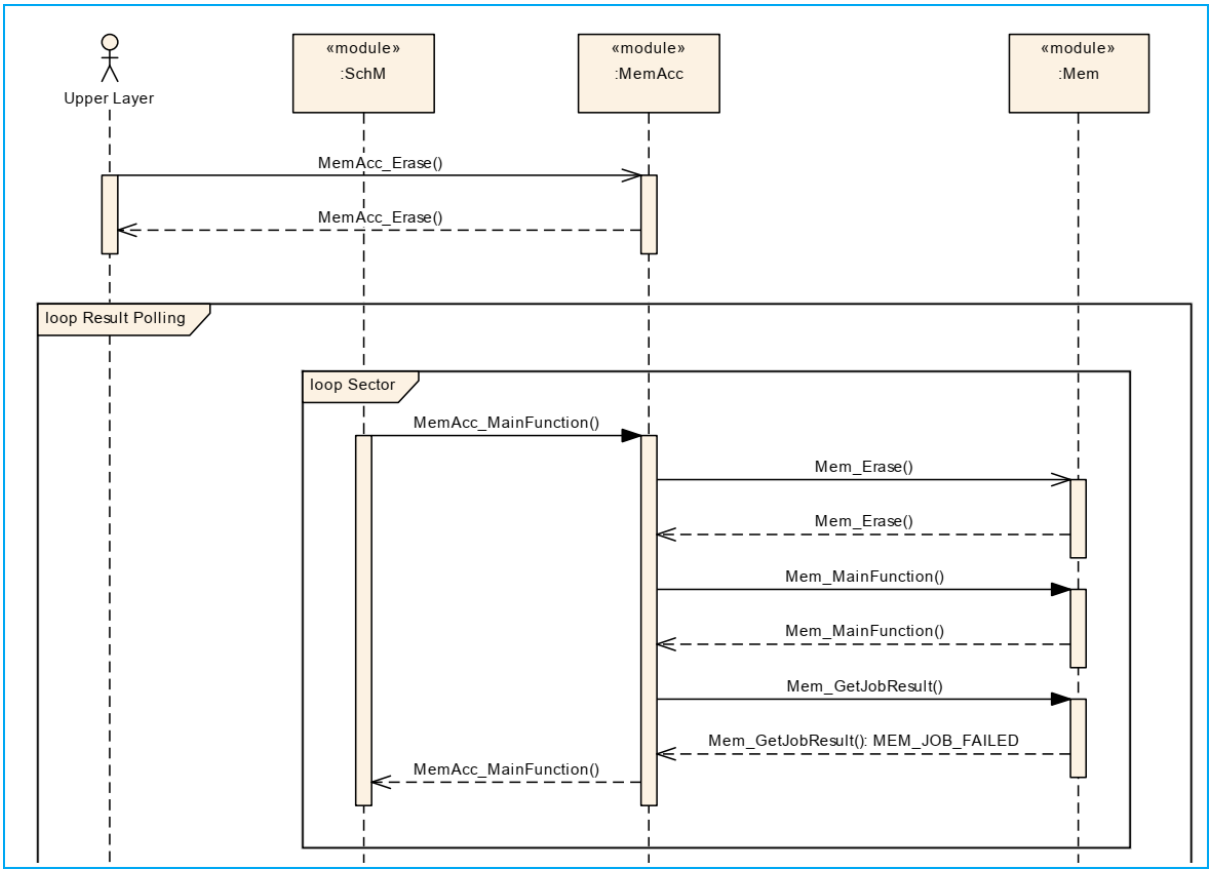
3.2.1.2 MemSectorBatch



- 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



3.2.3 Suspend and Resume Feature

- This feature in S32K1XX only support for Job Erase (because hardware limitation)
- When call Mem_43_INFLS_Suspend function:
  - Job current will suspend and return job result is pending
  - In case a suspend operation is already in pending, Mem_43_INFLS_Suspend reject the request by returning E_NOT_OK without further actions.
  - Other job request will be return error MEM_43_INFLS_E_JOB_PENDING
- When call Mem_43_INFLS_Resume function:
  - Shall resume a flash operations that was suspended by the Mem_43_INFLS_Suspend was called before
  - In case no suspend operation is pending, Mem_43_INFLS_Resume shall reject the request by returning E_NOT_OK without further actions.
  - Mem_43_INFLS_Resume return status E_OK new job can request.

3.3 Hardware Resources

3.3.1 Flash Banks/Arrays, Sectors details

- The sizes of flash memory types on the derivatives are:

Derivatives	Program Flash	Data Flash (FlexNVM)
S32K116	128 KB (sector size 2k)	32 KB (sector size 2k)
S32K118	256 KB (sector size 2k)	32 KB (sector size 2k)
S32K142	256 KB (sector size 2k)	64 KB (sector size 2k)
S32K142W	256 KB (sector size 4k)	64 KB (sector size 2k)
S32K144	512 KB (sector size 4k)	64 KB (sector size 2k)
S32K144W	512 KB (sector size 4k)	64 KB (sector size 2k)
S32K146	1 MB (sector size 4k)	64 KB (sector size 2k)
S32K148	1.5 MB (sector size 4k)	512 KB (sector size 4k)
S32M242	256 KB (sector size 2k)	64 KB (sector size 2k)
S32M244	512 KB (sector size 4k)	64 KB (sector size 2k)

- For S32K116: has 128 KBytes of code flash (program flash) and 32 KBytes of data flash (FlexNVM)
  - 128K P Flash (each sector is 2K so 128K/2K = 64 sectors)
  - 32K D Flash (each sector is 2K so 32K/2K = 16 sectors)
  - There are 2 blocks (read partitions):
    - P Flash: Block 0 (128K)
    - D Flash: Block 1 (32K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S015	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	2
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S063	2

- For S32K118: has 256 KBytes of code flash (program flash) and 32 KBytes of data flash (FlexNVM)
  - 256K P Flash (each sector is 2K so 256K/2K = 128 sectors)
  - 32K D Flash (each sector is 2K so 32K/2K = 16 sectors)
  - There are 2 blocks (read partitions):
    - P Flash: Block 0 (256K)
    - D Flash: Block 1 (32K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S015	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	2
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	2

- For S32K142: has 256 KBytes of code flash (program flash) and 64 KBytes of data flash (FlexNVM)
  - 256K P Flash (each sector is 2K so 256K/2K = 128 sectors)
  - 64K D Flash (each sector is 2K so 64K/2K = 32 sectors)
  - There are 2 blocks (read partitions):
    - P Flash: Block 0 (256K)
    - D Flash: Block 1 (64K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	2
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	2

- For S32K142W: has 256 KBytes of code flash (program flash) and 64 KBytes of data flash (FlexNVM)
  - 256K P Flash (each sector is 4K so 256K/4K = 64 sectors)
  - 64K D Flash (each sector is 2K so 64K/2K = 32 sectors)

  - There are 2 blocks (read partitions):
    * P Flash: Block 0 (256K)
    * D Flash: Block 1 (64K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	4
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S063	4

- For S32K144: has 512 KBytes of code flash (program flash) and 64 KBytes of data flash (FlexNVM)
  - 512K P Flash (each sector is 4K so 512/4K = 128 sectors)
  - 64K D Flash (each sector is 2K so 64K/2K = 32 sectors)
  - There are 2 blocks (read partitions):
    * P Flash: Block 0 (512K)
    * D Flash: Block 1 (64K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	4
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	4

- For S32K144W: has 512 KBytes of code flash (program flash) and 64 KBytes of data flash (FlexNVM)
  - 512K P Flash (each sector is 4K so 512/4K = 128 sectors)
  - 64K D Flash (each sector is 2K so 64K/2K = 32 sectors)
  - There are 2 blocks (read partitions):
    * P Flash: Block 0 (512K)
    * D Flash: Block 1 (64K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	4
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	4

- For S32K146: has 1 MBytes of code flash (program flash) and 64 KBytes of data flash (FlexNVM)
  - 1M P Flash (each sector is 4K so 1M/4K = 256 sectors)
  - 64K D Flash (each sector is 2K so 64K/2K = 128 sectors)
  - There are 3 blocks (read partitions):
    - P Flash: Block 0 (512K), Block 1 (512K)
    - D Flash: Block 2 (64K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_2_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_2_S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	4
...	...
FLS_CODE_ARRAY_0_BLOCK_1_S255	4

- For S32K148: has 1.5 MBytes of code flash (program flash) and 512 KBytes of data flash (FlexNVM)
  - 1.5M P Flash (each sector is 4K so 1.5M/4K = 384 sectors)
  - 512K D Flash (each sector is 4K so 512K/4K = 128 sectors)
  - There are 4 blocks (read partitions):
    - P Flash: Block 0 (512K), Block 1 (512K), Block 2 (512K)
    - D Flash: Block 3 (512K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_3_S000	4
...	...
FLS_DATA_ARRAY_0_BLOCK_3_S127	4
FLS_CODE_ARRAY_0_BLOCK_0_S000	4
...	...
FLS_CODE_ARRAY_0_BLOCK_2_S383	4

- For S32M242: has 256 KBytes of code flash (program flash) and 64 KBytes of data flash (FlexNVM)
  - 256K P Flash (each sector is 2K so 256/2K = 128 sectors)
  - 64K D Flash (each sector is 2K so 64K/2K = 32 sectors)
  - There are 2 blocks (read partitions):
    - P Flash: Block 0 (256K)
    - D Flash: Block 1 (64K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	2

Sector name	Sector Size (KB)
FLS_CODE_ARRAY_0_BLOCK_0_S000	2
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	2

- For S32M244: has 512 KBytes of code flash (program flash) and 64 KBytes of data flash (FlexNVM)
  - 512K P Flash (each sector is 4K so 512/4K = 128 sectors)
  - 64K D Flash (each sector is 2K so 64K/2K = 32 sectors)
  - There are 2 blocks (read partitions):
    - P Flash: Block 0 (512K)
    - D Flash: Block 1 (64K)

Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	2
...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	4
...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	4

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/S	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.	Hardware not support

Requirement	Status	Description	Notes
SWS_Mem_00077	N/S	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.	Hardware not support

3.5 Driver Limitations

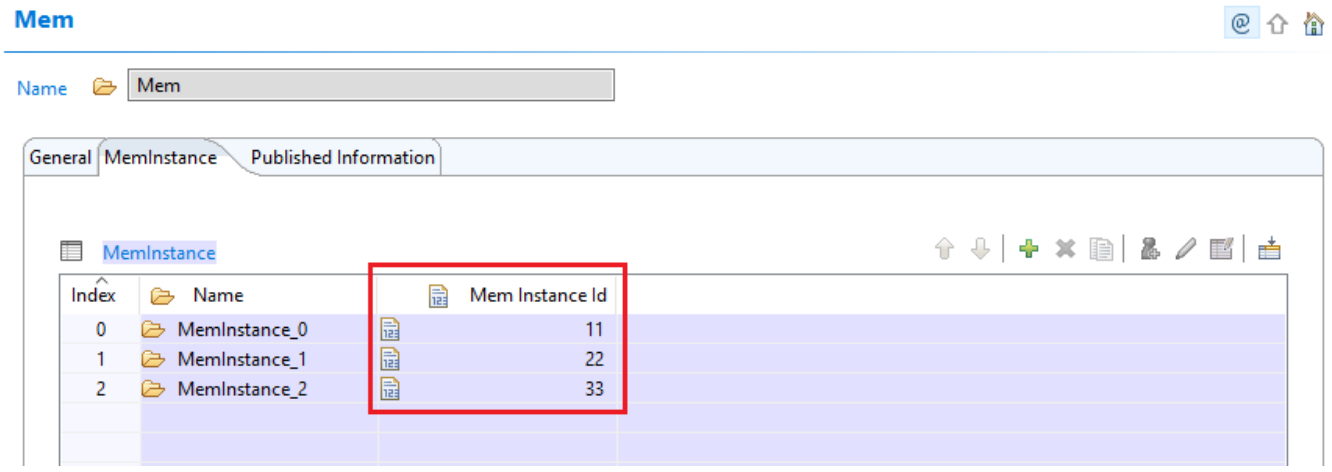
Note:

- On S32K148 derivative, PCCRMR[R2] should be programmed as 00b (NonCacheable) as S32K148 FlexNVMs region is not cacheable
- For more information, please refer to the LMEM chapter in the Reference manual and "Data Cache Restrictions" chapter in the Integration Manual

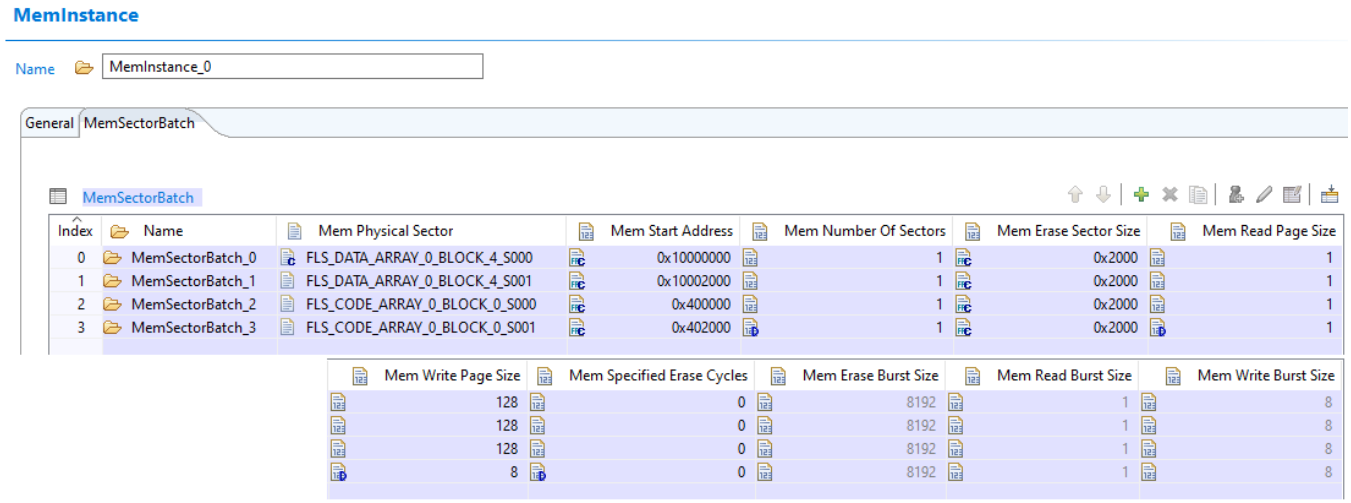
3.6 Driver usage and configuration tips

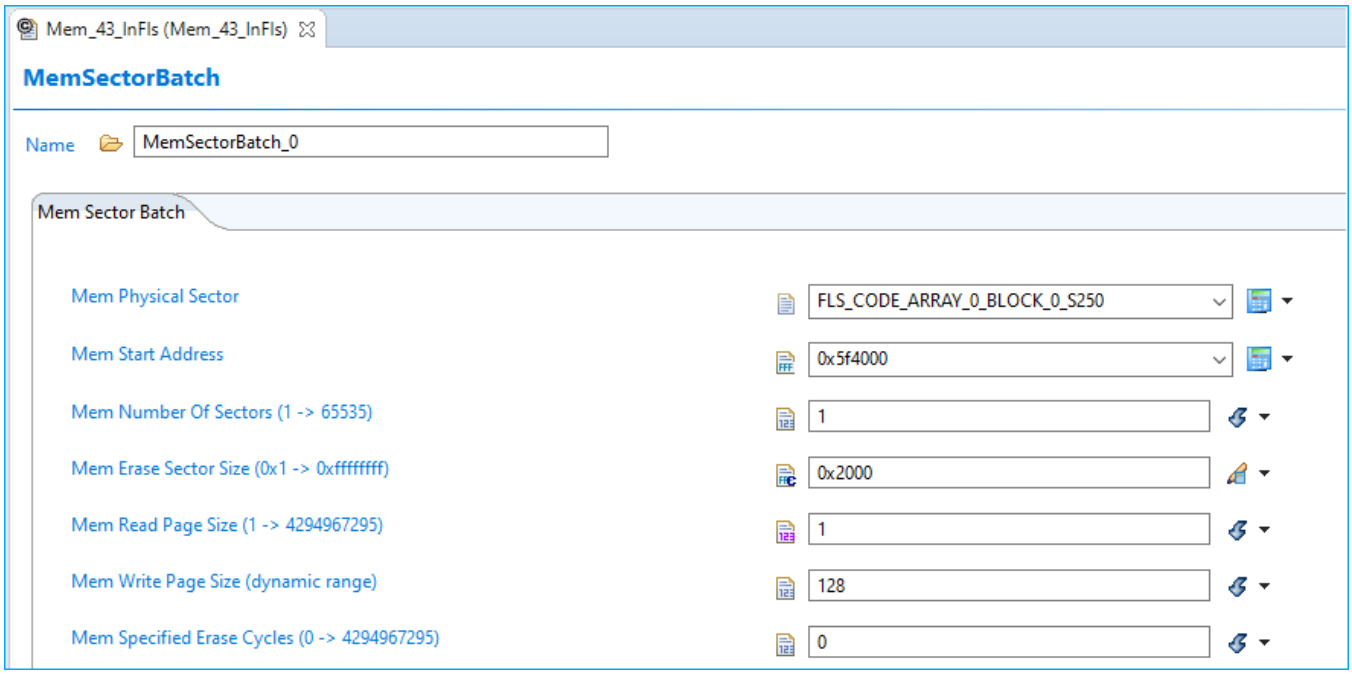
3.6.1 Tresos configuration

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)

3.6.2 The Compare feature in the Mem_43_INFLS_HwSpecificService() API.

- There are 4 input parameters to the HwSpecificService() function.
  - Mem_HwSpecificService(instanceId, hwServiceId, dataPtr,lengthPtr);
  - Mem_43_INFLS_InstanceIdType **instanceId**: ID of the related memory driver.
  - Mem_43_INFLS_HwServiceIdType **hwServiceId**: ID of the hardware service.
  - Mem_43_INFLS_DataType* **dataPtr**: Pointer to the configuration structure for the corresponding hwServiceID and cast to Mem_43_INFLS_DataType.
  - Mem_43_INFLS_LengthType* **lengthPtr**: The size of the configuration structure for hwServiceID.
- **hwServiceId** : Currently, the driver supports features hwSpecificID defined as follows.

  - Define ___MEM_43_INFLS_HWSERVICEID_COMPARE___ 0x11U.

- **dataPtr** : Depending on **hwServiceId**, **dataPtr** will be pointers to different structs, containing configuration information for that hwServiceID. Users need to declare and set the value for this struct variable before calling the hwspecifc function.

- The **dataPtr** parameter for **MEM_43_INFLS_HWSERVICEID_COMPARE** is a pointer to a configuration data structure defined as follows:

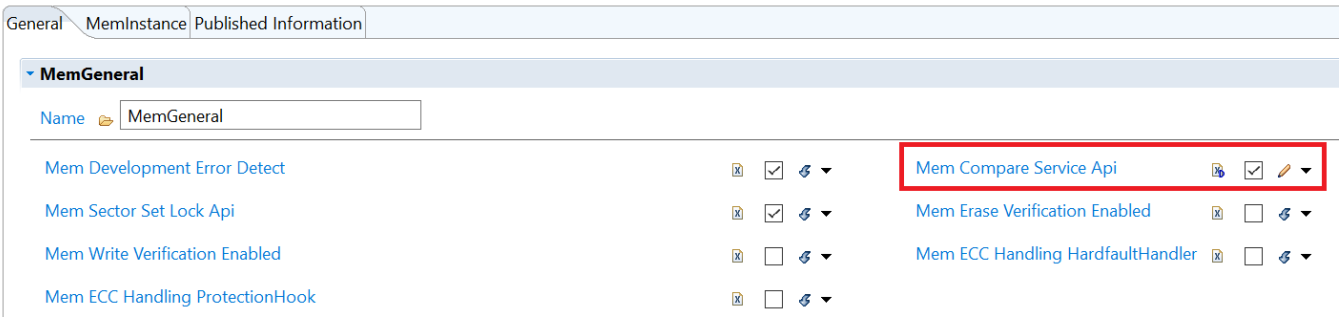
```
typedef struct
{
Mem_43_INFLS_AddressType    CompareAddr;     /* Address Source needs compare */
Mem_43_INFLS_DataType     * DataSourcePtr;   /* Data source pointer */
Mem_43_INFLS_LengthType     Length;          /* Length of data needs compare */
} Mem_43_INFLS_CompareConfigType;
```

- The user needs to declare and set the value for the variable of Mem_43_INFLS_CompareConfigType that the **dataPtr** points to before calling the Mem_43_INFLS_HwSpecificService() function.

- For example:
  - The following C code contains an example of how to call the Mem_43_INFLS_HwSpecificService() function with **MEM_43_INFLS_HWSERVICEID_COMPARE**:

```
/* Declare a configuration struct variable for hwServiceID */
Mem_43_INFLS_CompareConfigType sCompareCfg;
Mem_43_INFLS_LengthType        u32CompareCfgLength;
/* Initialize the configuration value for hwServiceID.*/
sCompareCfg.CompareAddr   = 0x1000;
sCompareCfg.DataSourcePtr = TxBuffer;
sCompareCfg.Length        = 512;
/* Save the length of the variable of struct configuration hwServiceID */
u32CompareCfgLength = sizeof(sCompareCfg);
/* Set job for Compare feature */
Status = Mem_43_INFLS_HwSpecificService(Instance0,
MEM_43_INFLS_HWSERVICEID_COMPARE,
(Mem_43_INFLS_DataType *)&sCompareCfg,
&u32CompareCfgLength);
if (E_OK == Status)
{
/* Perform the Compare job */
while (MEM_43_INFLS_JOB_PENDING == Mem_43_INFLS_GetJobResult(Instance0))
{
Mem_43_INFLS_MainFunction();
}
}
/* Get the Compare job result */
JobResult = Mem_43_INFLS_GetJobResult(Instance0);
```

- To use the hwServiceId Compare, the user needs to enable the "Mem Compare Service" button on the interface.



- When calling Mem_43_INFLS_HwSpecificService() with service ID is **MEM_43_INFLS_HWSERVICEID_COMPARE**, this API just sets the job. It will return **MEM_43_INFLS_E_NOT_OK** if the job set is successful and **MEM_43_INFLS_E_NOT_OK** if the job cannot be set. The actual Compare job will be done in the Mem_43_INFLS_MainFunction() function.
- The comparison result is returned when calling Mem_43_INFLS_GetJobResult().

- If both data blocks are the same: return: **MEM_43_INFLS_JOB_OK**.
- If both data blocks are not identical, return: **MEM_43_INFLS_JOB_INCONSISTENT**.
- If the execution of reading data from memory internal fails, return: **MEM_43_INFLS_JOB_FAI↩ LED**.

3.6.3 Avoiding RWW problem

To avoid RWW (Read While Write) problems on the internal flash, the Mem_43_INFLS driver provides the MemAcLoadOnJobStart configuration parameter. If it is set to true the Mem_43_INFLS driver will load the flash access code routine to RAM whenever an erase or write job is started and unload (overwrite) it after that job has been finished or canceled.

The flash access code is loaded to RAM and therefore the flash driver shouldn't have RWW problems; if MemAc↩ LoadOnJobStart is set to false the sector erased / page written must belong to flash array / partition different from flash array / partition the application is executing from.

If a platform does not support multiple read-while-write partitions, either the flash access code must be loaded to RAM or the entire code execution has to be moved to RAM while the Mem_43_INFLS driver is performing the modify operation.

Note:

1. the Mem_43_INFLS driver uses the sector erase / page write access code to clear the MCRS:DONE bit and wait for completion of high voltage operation.
2. The flash module might be further divided into partitions/blocks that determine locations for valid read-while-write (RWW) operations(Ex: Program flash block 0 and Data flash/FlexNvm). While the embedded flash memory is performing a 'write' (program or erase) to a given partition, it can simultaneously perform a read from any other partition.
3. MemAcCallback should be in located in RAM if MemAcLoadOnJobStart is true to avoid RWW problem.
4. When performing flash modifying operations which might interfere with the executing code, the application should take into account also the possible interrupts which could execute from flash during a flash modify operation. The flash access notifications can be used, in order to notify the start and finish of the flash access.
5. MemCleanCacheAfterLoadAc allow to clean cache after loading AccessCode to RAM to ensure the synchronization between cache and RAM memory. This action might be needed in case the AccessCode function is copied to a cacheable area.

3.6.4 Flash memory physical sectors unlock example

For unprotecting internal flash sectors, the flash field configuration locations corresponding to sector protection have to be erased (Addresses 0x0_0408 - 0x0_040B and 0x0_040F), reprogrammed if needed and the chip reset.

Care has to be taken when programming the flash configuration field, so that FSEC location (0x0_040C) is reprogrammed to value 0xFE after erase, and all configuration locations are erased or programmed as needed.

Code example for resetting configuration field using direct register access.

```
/* Erase flash sector containing configuration field */
FTFC->FCCOB0 = 0x09;    /* Erase sector */
FTFC->FCCOB1 = 0x00;    /* Address 0x0_0000*/
FTFC->FCCOB2 = 0x00;    /* Address */
FTFC->FCCOB3 = 0x00;    /* Address */

/* Program flash configuration field */
FTFC->FCCOB0 = 0x07;    /* Program phrase */
FTFC->FCCOB1 = 0x08;    /* Address 0x0_0408*/
FTFC->FCCOB2 = 0x04;    /* Address */
FTFC->FCCOB3 = 0x00;    /* Address */
FTFC->FCCOB4 = 0xFF;    /* Data for flash location 0x0_040B, FPROT3 */
FTFC->FCCOB5 = 0xFF;    /* Data for flash location 0x0_040A, FPROT2 */
FTFC->FCCOB6 = 0xFF;    /* Data for flash location 0x0_0409, FPROT1 */
FTFC->FCCOB7 = 0xFF;    /* Data for flash location 0x0_0408, FPROT0 */
FTFC->FCCOB8 = 0xFF;    /* Data for flash location 0x0_040F, FDPROT */
FTFC->FCCOB9 = 0xFF;    /* Data for flash location 0x0_040E, FEPROT */
FTFC->FCCOBA = 0xFF;    /* Data for flash location 0x0_040D, FOPT */
FTFC->FCCOBB = 0xFE;    /* Data for flash location 0x0_040C, FSEC */

/* Reset */
```

3.7 Runtime errors

The driver generates the following DET errors at runtime.

Table 3.14 Errors (reported by DET)

Error Code	Value	Function	Condition triggering the error
MEM_43_INFLS_E_UNINIT	1	Mem_43_INFLS other functions	The driver is in an uninitialized state.
MEM_43_INFLS_E_PARAM_POINTER	2	Mem_43_INFLS_Init() Mem_43_INFLS_GetVersionInfo()	Invalid input parameter.
MEM_43_INFLS_E_PARAM_ADDRESS	3	Mem_43_INFLS other functions	Invalid address.
MEM_43_INFLS_E_PARAM_LENGTH	4		Invalid length.
MEM_43_INFLS_E_PARAM_INSTANCE_ID	5		Invalid driver instance ID.

Error Code	Value	Function	Condition triggering the error
MEM_43_INFLS_E_JOB_↩ PENDING	6		A job request is still in progress.

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 MemGeneral
    - Parameter MemDevErrorDetect
    - Parameter MemHwCompareService
    - Parameter MemSectorSetLockApi
    - Parameter MemEraseVerificationEnabled
    - Parameter MemWriteVerificationEnabled
    - Parameter MemAcLoadOnJobStart
    - Parameter MemCleanCacheAfterLoadAc
    - Parameter MemAcErase
    - Parameter MemAcWrite
    - Parameter MemAcErasePointer
    - Parameter MemAcWritePointer
    - Parameter MemAcCallback
    - Parameter MemStartFlashAccessNotif
    - Parameter MemFinishedFlashAccessNotif
    - Parameter MemTimeoutSupervisionEnabled
    - Parameter MemTimeoutMethod
    - Parameter MemEraseTimeout
    - Parameter MemWriteTimeout
    - Parameter MemAbortTimeout
    - Parameter MemEraseSuspendTimeout
    - Parameter MemEraseResumeTimeout
  - 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 MemAutosarExt

* Parameter MemUsesAlterInterface
* Parameter MemEnableUserModeSupport
* Parameter MemSynchronizeCache

– 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:

- MemGeneral
- MemInstance
- MemAutosarExt
- MemPublishedInformation
- CommonPublishedInformation

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

4.2 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

4.3 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

4.4 Parameter MemHwCompareService

Compile switch to enable and disable the Mem_Compare function.

- true: API supported / function provided.

- false: API not supported / function not provided

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

4.5 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

4.6 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

4.7 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

4.8 Parameter MemAcLoadOnJobStart

The flash driver shall load the flash access code to RAM whenever an erase or write job is started and unload (overwrite) it after that job has been finished or canceled.

true: Flash access code loaded on job start / unloaded on job end or error.

false: Flash access code not loaded to / unloaded from RAM at all.

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

4.9 Parameter MemCleanCacheAfterLoadAc

Vendor specific: Pre-processor switch to allow to clean cache after loading AccessCode to RAM to ensure the synchronization between cache and RAM memory.

This action might be needed in case the AccessCode function is copied to a cacheable area.

true: cleans cache after loading AccessCode function to RAM to write cache data to the actual RAM memory

false: does not clean cache after loading AccessCode function to RAM

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

4.10 Parameter MemAcErase

Address offset in RAM to which the erase flash access code shall be loaded.

Used as function pointer to access the erase flash access code.

Note: To use Mem Access Code Erase be sure Mem Access Code Erase Pointer is NULL or NULL_PTR.

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	536740096
max	4294967295
min	0

4.11 Parameter MemAcWrite

Address offset in RAM to which the write flash access code shall be loaded.

Used as function pointer to access the write flash access code.

Note: To use Mem Access Code Write be sure Mem Access Code Write Pointer is NULL or NULL_PTR.

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	536740096
max	4294967295
min	0

4.12 Parameter MemAcErasePointer

Vendor specific: Pointer in RAM to which the erase flash access code shall be loaded.

Eg: May be use pointer is _ERASE_FUNC_ADDRESS_

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

4.13 Parameter MemAcWritePointer

Vendor specific: Pointer in RAM to which the write flash access code shall be loaded.

Used as function pointer to access the write flash access code.

Eg: May be use pointer is _WRITE_FUNC_ADDRESS_

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

4.14 Parameter MemAcCallback

Vendor specific: Mapped to the Access Code Callback provided by some upper layer module, typically the Wdg module.

Note: Disable the Access Code Callback to have it set as NULL_PTR.

Property	Value
type	ECUC-FUNCTION-NAME-DEF

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

4.15 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	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	Mem_StartFlashAccessNotif

4.16 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	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	Mem_FinishedFlashAccessNotif

4.17 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

4.18 Parameter MemTimeoutMethod

Vendor specific: Counter type used in timeout detection for Mem Internal 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']

4.19 Parameter MemEraseTimeout

Vendor specific: The timeout value for Erase Operation.

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

4.20 Parameter MemWriteTimeout

Vendor specific: The timeout value for Write Operation.

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 MemAbortTimeout

Vendor specific: The timeout value for aborting an ongoing operation.

The timeout is used also in 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

4.22 Parameter MemEraseSuspendTimeout

Vendor specific: Mem Erase Suspend Timeout is the timeout value to suspend an Erase Flash Sector operation.

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

4.23 Parameter MemEraseResumeTimeout

Vendor specific: Mem Erase Resume Timeout is the timeout value to resume an Erase Flash Sector operation.

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

4.24 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	65535
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.25 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

4.26 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

4.27 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_3_S000

Property	Value
literals	['FLS_DATA_ARRAY_0_BLOCK_3_S000', 'FLS_DATA_ARRAY_0_BLOCK_3_S001', 'FLS_DATA_ARRAY_0_BLOCK_3_S002', 'FLS_DATA_ARRAY_0_BLOCK_3_S003', 'FLS_DATA_ARRAY_0_BLOCK_3_S004', 'FLS_DATA_ARRAY_0_BLOCK_3_S005', 'FLS_DATA_ARRAY_0_BLOCK_3_S006', 'FLS_DATA_ARRAY_0_BLOCK_3_S007', 'FLS_DATA_ARRAY_0_BLOCK_3_S008', 'FLS_DATA_ARRAY_0_BLOCK_3_S009', 'FLS_DATA_ARRAY_0_BLOCK_3_S010', 'FLS_DATA_ARRAY_0_BLOCK_3_S011', 'FLS_DATA_ARRAY_0_BLOCK_3_S012', 'FLS_DATA_ARRAY_0_BLOCK_3_S013', 'FLS_DATA_ARRAY_0_BLOCK_3_S014', 'FLS_DATA_ARRAY_0_BLOCK_3_S015', 'FLS_DATA_ARRAY_0_BLOCK_3_S016', 'FLS_DATA_ARRAY_0_BLOCK_3_S017', 'FLS_DATA_ARRAY_0_BLOCK_3_S018', 'FLS_DATA_ARRAY_0_BLOCK_3_S019', 'FLS_DATA_ARRAY_0_BLOCK_3_S020', 'FLS_DATA_ARRAY_0_BLOCK_3_S021', 'FLS_DATA_ARRAY_0_BLOCK_3_S022', 'FLS_DATA_ARRAY_0_BLOCK_3_S023', 'FLS_DATA_ARRAY_0_BLOCK_3_S024', 'FLS_DATA_ARRAY_0_BLOCK_3_S025', 'FLS_DATA_ARRAY_0_BLOCK_3_S026', 'FLS_DATA_ARRAY_0_BLOCK_3_S027', 'FLS_DATA_ARRAY_0_BLOCK_3_S028', 'FLS_DATA_ARRAY_0_BLOCK_3_S029', 'FLS_DATA_ARRAY_0_BLOCK_3_S030', 'FLS_DATA_ARRAY_0_BLOCK_3_S031', 'FLS_DATA_ARRAY_0_BLOCK_3_S032', 'FLS_DATA_ARRAY_0_BLOCK_3_S033', 'FLS_DATA_ARRAY_0_BLOCK_3_S034', 'FLS_DATA_ARRAY_0_BLOCK_3_S035', 'FLS_DATA_ARRAY_0_BLOCK_3_S036', 'FLS_DATA_ARRAY_0_BLOCK_3_S037', 'FLS_DATA_ARRAY_0_BLOCK_3_S038', 'FLS_DATA_ARRAY_0_BLOCK_3_S039', 'FLS_DATA_ARRAY_0_BLOCK_3_S040', 'FLS_DATA_ARRAY_0_BLOCK_3_S041', 'FLS_DATA_ARRAY_0_BLOCK_3_S042', 'FLS_DATA_ARRAY_0_BLOCK_3_S043', 'FLS_DATA_ARRAY_0_BLOCK_3_S044', 'FLS_DATA_ARRAY_0_BLOCK_3_S045', 'FLS_DATA_ARRAY_0_BLOCK_3_S046', 'FLS_DATA_ARRAY_0_BLOCK_3_S047', 'FLS_DATA_ARRAY_0_BLOCK_3_S048', 'FLS_DATA_ARRAY_0_BLOCK_3_S049', 'FLS_DATA_ARRAY_0_BLOCK_3_S050', 'FLS_DATA_ARRAY_0_BLOCK_3_S051', 'FLS_DATA_ARRAY_0_BLOCK_3_S052', 'FLS_DATA_ARRAY_0_BLOCK_3_S053', 'FLS_DATA_ARRAY_0_BLOCK_3_S054', 'FLS_DATA_ARRAY_0_BLOCK_3_S055', 'FLS_DATA_ARRAY_0_BLOCK_3_S056', 'FLS_DATA_ARRAY_0_BLOCK_3_S057', 'FLS_DATA_ARRAY_0_BLOCK_3_S058', 'FLS_DATA_ARRAY_0_BLOCK_3_S059', 'FLS_DATA_ARRAY_0_BLOCK_3_S060', 'FLS_DATA_ARRAY_0_BLOCK_3_S061', 'FLS_DATA_ARRAY_0_BLOCK_3_S062', 'FLS_DATA_ARRAY_0_BLOCK_3_S063', 'FLS_DATA_ARRAY_0_BLOCK_3_S064', 'FLS_DATA_ARRAY_0_BLOCK_3_S065', 'FLS_DATA_ARRAY_0_BLOCK_3_S066', 'FLS_DATA_ARRAY_0_BLOCK_3_S067', 'FLS_DATA_ARRAY_0_BLOCK_3_S068', 'FLS_DATA_ARRAY_0_BLOCK_3_S069', 'FLS_DATA_ARRAY_0_BLOCK_3_S070', 'FLS_DATA_ARRAY_0_BLOCK_3_S071', 'FLS_DATA_ARRAY_0_BLOCK_3_S072', 'FLS_DATA_ARRAY_0_BLOCK_3_S073', 'FLS_DATA_ARRAY_0_BLOCK_3_S074', 'FLS_DATA_ARRAY_0_BLOCK_3_S075', 'FLS_DATA_ARRAY_0_BLOCK_3_S076', 'FLS_DATA_ARRAY_0_BLOCK_3_S077', 'FLS_DATA_ARRAY_0_BLOCK_3_S078', 'FLS_DATA_ARRAY_0_BLOCK_3_S079', 'FLS_DATA_ARRAY_0_BLOCK_3_S080', 'FLS_DATA_ARRAY_0_BLOCK_3_S081', 'FLS_DATA_ARRAY_0_BLOCK_3_S082', 'FLS_DATA_ARRAY_0_BLOCK_3_S083', 'FLS_DATA_ARRAY_0_BLOCK_3_S084', 'FLS_DATA_ARRAY_0_BLOCK_3_S085', 'FLS_DATA_ARRAY_0_BLOCK_3_S086', 'FLS_DATA_ARRAY_0_BLOCK_3_S087', 'FLS_DATA_ARRAY_0_BLOCK_3_S088', 'FLS_DATA_ARRAY_0_BLOCK_3_S089', 'FLS_DATA_ARRAY_0_BLOCK_3_S090', 'FLS_DATA_ARRAY_0_BLOCK_3_S091', 'FLS_DATA_ARRAY_0_BLOCK_3_S092', 'FLS_DATA_ARRAY_0_BLOCK_3_S093', 'FLS_DATA_ARRAY_0_BLOCK_3_S094', 'FLS_DATA_ARRAY_0_BLOCK_3_S095', 'FLS_DATA_

Property	Value

4.28 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	0
max	9223372036854775807
min	0

4.29 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

4.30 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	4096
max	4294967295
min	1

4.31 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

4.32 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	4294967295
min	1

4.33 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

4.34 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

4.35 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

4.36 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

4.37 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

4.38 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

4.39 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

4.40 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

4.41 Parameter MemSynchronizeCache

Vendor specific:

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

The Mem Internal 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 Internal 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

4.42 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

4.43 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

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	4294967295
max	4294967295
min	0

4.44 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	1427461397
max	4294967295
min	0

4.45 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

4.46 Parameter ArReleaseMajorVersion

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

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

4.47 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

4.48 Parameter ArReleaseRevisionVersion

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

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

4.49 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

4.50 Parameter SwMajorVersion

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

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

Property	Value
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	2
max	2
min	2

4.51 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

4.52 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

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.53 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	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	INFLS

4.54 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 FTFC IP Driver

6.1.1 Detailed Description implement Ftfc_Mem_InFls_Ip_Types.h_Artifact

Data Structures

- struct Ftfc_Mem_InFls_Ip_ConfigType
  Ftfc Configuration Structure. *More...*

Macros

- #define FLASH_CMD_PROGRAM_PHRASE
  FCCOB commands IDs.
- #define FTFC_MEM_INFLS_IP_WRITE_DOUBLE_WORD
  Program alignment.
- #define FTFC_MEM_INFLS_IP_SIZE_1BYTE
  the number of bytes uses to compare (1 byte).
- #define FTFC_MEM_INFLS_IP_SIZE_2BYTE
  the number of bytes uses to compare (2 bytes).
- #define FTFC_MEM_INFLS_IP_SIZE_4BYTE
  the number of bytes uses to compare (4 bytes).

Types Reference

- typedef void(* Ftfc_Mem_InFls_Ip_StartFlashAccessNotifPtrType) (void)
  Start Flash Access Notification Pointer Type.
- typedef void(* Ftfc_Mem_InFls_Ip_FinishedFlashAccessNotifPtrType) (void)
  Finished Flash Access Notification Pointer Type.

Enum Reference

- enum Ftfc_Mem_InFls_Ip_FlashBlocksNumberType
  Enumeration of Blocks of memory flash .
- enum Ftfc_Mem_InFls_Ip_StatusType
  Enumeration of checking status errors or not.

Function Reference

- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Init (const Ftfc_Mem_InFls_Ip_ConfigType *Ftfc_Mem_InFls_Ip_pInitConfig)
  Initializes the FTCF module.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Abort (void)
  Abort a program or erase operation.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Read (Ftfc_Mem_InFls_Ip_AddressType u32SrcAddress, uint8 *pDestAddressPtr, Ftfc_Mem_InFls_Ip_LengthType u32Length)
  This function fills data to pDestAddressPtr.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Compare (Ftfc_Mem_InFls_Ip_AddressType u32SrcAddress, const uint8 *pCompareAddressPtr, Ftfc_Mem_InFls_Ip_LengthType u32Length)
  Checks that there is the desired data at the specified address.
- Ftfc_Mem_InFls_Ip_FlashBlocksNumberType Ftfc_Mem_InFls_Ip_GetBlockNumberFromAddress (uint32 u32TargetAddress)
  Get block number from target address.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_SectorErase (Ftfc_Mem_InFls_Ip_AddressType u32SectorStartAddress)
  Accepts and erases a selected program flash or data flash sector if possible.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_SectorEraseStatus (void)
  Checks the status of the hardware erase started by the Ftfc_Mem_InFls_Ip_SectorErase function.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_EraseSuspend (void)
  Suspend a current operation of Flash erase sector command.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_EraseResume (void)
  Resume a current operation of Flash erase sector command.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Write (uint32 u32DestAddress, const uint8 *pSourceAddressPtr, uint32 u32Length)
  Writes data into the memory array using the main interface. Initiates the hardware write and then exits.
- Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_WriteStatus (void)
  Checks the status of the hardware program started by the FTFC_Ip_Write function.
- void Ftfc_Mem_InFls_Ip_SetLoadAcStatus (const boolean Status)
  Set Status for Ftfc_Mem_InFls_Ip_LoadAc_Status.
- void Ftfc_Mem_InFls_Ip_InvalidPrefetchBuff_Ram (void)
  Invalidate prefetch buffer before reading to make sure that the driver always reads the new data from flash.

6.1.2 Data Structure Documentation

6.1.2.1 struct Ftfc_Mem_InFls_Ip_ConfigType

Ftfc Configuration Structure.

Implements : Ftfc_Mem_InFls_Ip_ConfigType_Class

Definition at line 143 of file Ftfc_Mem_InFls_Ip_Types.h.

Data Fields

Type	Name	Description
Ftfc_Mem_InFls_Ip_StartFlashAccessNotifPtrType	StartFlashAccessNotifPtr	Pointer to start flash access callout
Ftfc_Mem_InFls_Ip_FinishedFlashAccessNotifPtrType	FinishedFlashAccessNotifPtr	Pointer to finish flash access callout

6.1.3 Macro Definition Documentation

6.1.3.1 FLASH_CMD_PROGRAM_PHRASE

```
#define FLASH_CMD_PROGRAM_PHRASE
```

FCCOB commands IDs.

Definition at line 78 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.3.2 FTFC_MEM_INFLS_IP_WRITE_DOUBLE_WORD

```
#define FTFC_MEM_INFLS_IP_WRITE_DOUBLE_WORD
```

Program alignment.

Definition at line 84 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.3.3 FTFC_MEM_INFLS_IP_SIZE_1BYTE

```
#define FTFC_MEM_INFLS_IP_SIZE_1BYTE
```

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

Definition at line 90 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.3.4 FTFC_MEM_INFLS_IP_SIZE_2BYTE

```
#define FTFC_MEM_INFLS_IP_SIZE_2BYTE
```

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

Definition at line 96 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.3.5 FTFC_MEM_INFLS_IP_SIZE_4BYTE

```
#define FTFC_MEM_INFLS_IP_SIZE_4BYTE
```

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

Definition at line 102 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.4 Types Reference

6.1.4.1 Ftfc_Mem_InFls_Ip_StartFlashAccessNotifPtrType

```
typedef void(* Ftfc_Mem_InFls_Ip_StartFlashAccessNotifPtrType) (void)
```

Start Flash Access Notification Pointer Type.

Pointer type of Ftfc_Mem_InFls_Ip_StartFlashAccessNotifPtrType function

Definition at line 129 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.4.2 Ftfc_Mem_InFls_Ip_FinishedFlashAccessNotifPtrType

```
typedef void(* Ftfc_Mem_InFls_Ip_FinishedFlashAccessNotifPtrType) (void)
```

Finished Flash Access Notification Pointer Type.

Pointer type of Ftfc_Mem_InFls_Ip_FinishedFlashAccessNotifPtrType function

Definition at line 136 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.5 Enum Reference

6.1.5.1 Ftfc_Mem_InFls_Ip_FlashBlocksNumberType

```
enum Ftfc_Mem_InFls_Ip_FlashBlocksNumberType
```

Enumeration of Blocks of memory flash .

Enumerator

FLS_CODE_BLOCK_0	code block number 0
FLS_CODE_BLOCK_1	code block number 1
FLS_CODE_BLOCK_2	code block number 2
FLS_DATA_BLOCK	data block

Definition at line 115 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.5.2 Ftfc_Mem_InFls_Ip_StatusType

```
enum Ftfc_Mem_InFls_Ip_StatusType
```

Enumeration of checking status errors or not.

Enumerator

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Successful job
STATUS_FTFC_MEM_INFLS_IP_BUSY	IP is performing an operation
STATUS_FTFC_MEM_INFLS_IP_ERROR	Error - general code
STATUS_FTFC_MEM_INFLS_IP_ERROR_TIMEOUT	Error - exceeded timeout
STATUS_FTFC_MEM_INFLS_IP_ERROR_INPUT_PARAM	Error - wrong input parameter
STATUS_FTFC_MEM_INFLS_IP_ERROR_BLANK_CHECK	Error - selected memory area is not erased
STATUS_FTFC_MEM_INFLS_IP_ERROR_PROGRAM_VERIFY	Error - selected memory area doesn't contain desired value
STATUS_FTFC_MEM_INFLS_IP_ERROR_USER_TEST_BREAK_SBC	Error - single bit correction
STATUS_FTFC_MEM_INFLS_IP_ERROR_USER_TEST_BREAK_DBD	Error - double bit detection
STATUS_FTFC_MEM_INFLS_IP_SECTOR_UNPROTECTED	Checked sector is unlocked
STATUS_FTFC_MEM_INFLS_IP_SECTOR_PROTECTED	Checked sector is locked
STATUS_FTFC_MEM_INFLS_IP_ECC_UNCORRECTED	Ecc uncorrected error
STATUS_FTFC_MEM_INFLS_IP_ECC_CORRECTED	Ecc corrected error

Definition at line 152 of file Ftfc_Mem_InFls_Ip_Types.h.

6.1.6 Function Reference

6.1.6.1 Ftfc_Mem_InFls_Ip_Init()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Init (
            const Ftfc_Mem_InFls_Ip_ConfigType * Ftfc_Mem_InFls_Ip_pInitConfig )
```

Initializes the FTCF module.

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

Parameters

in	*Ftfc_Mem_InFls_Ip_pInitConfig*	Pointer to the driver configuration structure.

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Initialization is success
STATUS_FTFC_MEM_INFLS_IP_ERROR_TIMEOUT	Errors Timeout because wait for the Done bit long time

6.1.6.2 Ftfc_Mem_InFls_Ip_Abort()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_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

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	: The operation is successful.
STATUS_FTFC_MEM_INFLS_IP_ERROR_TIMEOUT	the operation error because wait for the Done bit long time

6.1.6.3 Ftfc_Mem_InFls_Ip_Read()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Read (
            Ftfc_Mem_InFls_Ip_AddressType u32SrcAddress,
            uint8 * pDestAddressPtr,
            Ftfc_Mem_InFls_Ip_LengthType u32Length )
```

This function fills data to pDestAddressPtr.

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

Parameters

in	*u32SrcAddress*	The start address of the area to be read.
in	*pDestAddressPtr*	Pointer to the destination of the read.
in	*u32Length*	Read size

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Read performed successfully.
STATUS_FTFC_MEM_INFLS_IP_ERROR_INPUT_PARAM	Input parameters are invalid.
STATUS_FTFC_MEM_INFLS_IP_ERROR	There was an error while reading.

Precondition

The module has to be initialized and not busy.

6.1.6.4 Ftfc_Mem_InFls_Ip_Compare()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Compare (
            Ftfc_Mem_InFls_Ip_AddressType u32SrcAddress,
            const uint8 * pCompareAddressPtr,
            Ftfc_Mem_InFls_Ip_LengthType u32Length )
```

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 intented to be a blank check, the pSourceAddressPtr should be NULL.

Parameters

in	*u32SrcAddress*	The start address of the area to be checked.
in	*pCompareAddressPtr*	Pointer to the data expected to be read.
in	*u32Length*	Check size

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Read performed successfully.
STATUS_FTFC_MEM_INFLS_IP_ERROR_IN↩ PUT_PARAM	Input parameters are invalid.
STATUS_FTFC_MEM_INFLS_IP_ERROR	There was an error while reading.
STATUS_FTFC_MEM_INFLS_IP_ERROR_PR↩ OGRAM_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 Ftfc_Mem_InFls_Ip_GetBlockNumberFromAddress()

```
Ftfc_Mem_InFls_Ip_FlashBlocksNumberType Ftfc_Mem_InFls_Ip_GetBlockNumberFromAddress (
            uint32 u32TargetAddress )
```

Get block number from target address.

Get block number from target address

Parameters

in	*u32TargetAddress*	target address

Returns

Ftfc_Mem_InFls_Ip_GetBlockNumberFromAddress

Return values

The	block number which contains the target address.

6.1.6.6 Ftfc_Mem_InFls_Ip_SectorErase()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_SectorErase (
            Ftfc_Mem_InFls_Ip_AddressType u32SectorStartAddress )
```

Accepts and erases a selected program flash or data flash sector if possible.

Accepts an erase job over one of the sectors if possible. Starts the high voltage erase and then exits. The status of the hardware erase must be verified by calling asynchronously the Ftfc_Mem_InFls_Ip_SectorEraseStatus function. The Ftfc_Mem_InFls_Ip_SectorErase function shall cover all the available sectors.

Parameters

in	*u32SectorStartAddress*	The start address of the sector to be erased.

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Hardware erase started successfully
STATUS_FTFC_MEM_INFLS_IP_ERROR_IN↩ PUT_PARAM	The selected sector is out of bound
STATUS_FTFC_MEM_INFLS_IP_ERROR	There is another job configured or in progress or The sector is locked by another core or couldn't be unlocked.
STATUS_FTFC_MEM_INFLS_IP_ERROR_TI↩ MEOUT	The erase operation exceeded the timeout - Status value available only if the timeout feature is enabled

Precondition

The module has to be initialized.

6.1.6.7 Ftfc_Mem_InFls_Ip_SectorEraseStatus()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_SectorEraseStatus (
            void  )
```

Checks the status of the hardware erase started by the Ftfc_Mem_InFls_Ip_SectorErase function.

Checks the status of the hardware erase started by the Ftfc_Mem_InFls_Ip_SectorErase function.

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Erase performed successfully
STATUS_Ftfc_Mem_InFls_Ip_BUSY	Hardware erase is still in progress
STATUS_FTFC_MEM_INFLS_IP_ERROR	There was an error during the hardware erase.
STATUS_FTFC_MEM_INFLS_IP_ERROR_TIMEOUT	The erase operation exceeded the timeout - Status value available only if the timeout feature is enabled.
STATUS_FTFC_MEM_INFLS_IP_ERROR_BLANK_CHECK	The sector was not erased correctly - Status value available only if the blank check feature is enabled

Precondition

The module has to be initialized.

6.1.6.8 Ftfc_Mem_InFls_Ip_EraseSuspend()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_EraseSuspend (
            void  )
```

Suspend a current operation of Flash erase sector command.

Suspend a current operation of Flash erase sector command.

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Operation of Flash erase sector suspended successfully
STATUS_FTFC_MEM_INFLS_IP_ERROR	Operation of Flash erase sector suspended fail

Precondition

The module has to be initialized.

6.1.6.9 Ftfc_Mem_InFls_Ip_EraseResume()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_EraseResume (
            void  )
```

Resume a current operation of Flash erase sector command.

Resume a current operation of Flash erase sector command.

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Operation of Flash erase sector resumed successfully
STATUS_FTFC_MEM_INFLS_IP_ERROR	Operation of Flash erase sector resumed fail

Precondition

The module has to be initialized.

6.1.6.10 Ftfc_Mem_InFls_Ip_Write()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Write (
            uint32 u32DestAddress,
            const uint8 * pSourceAddressPtr,
            uint32 u32Length )
```

Writes data into the memory array using the main interface. Initiates the hardware write and then exits.

Writes data into the memory array using the main interface. Initiates the hardware write and then exits. the status of the hardware erase must be verified by calling asynchronously the Ftfc_Mem_InFls_Ip_WriteStatus function.

Parameters

in	*u32DestAddress*	The start address of the write, must be aligned with 8 bytes.
in	*pSourceAddressPtr*	Source program buffer address.
in	*u32Length*	Size in bytes of the flash region to be programed, must be aligned with 8 bytes and the maximum value is 128 bytes.

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Program performed successfully
STATUS_FTFC_MEM_INFLS_IP_ERROR_INPUT_PARAM	The input parameters are invaid.
STATUS_FTFC_MEM_INFLS_IP_ERROR	There is another job configured or in progress or The sector is locked by another core or couldn't be unlocked.
STATUS_FTFC_MEM_INFLS_IP_ERROR_TIMEOUT	The erase operation exceeded the timeout - Status value available only if the timeout feature is enabled

Precondition

The module has to be initialized.

6.1.6.11 Ftfc_Mem_InFls_Ip_WriteStatus()

```
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_WriteStatus (
            void  )
```

Checks the status of the hardware program started by the FTFC_Ip_Write function.

Checks the status of the hardware program started by the FTFC_Ip_Write function.

Returns

Ftfc_Mem_InFls_Ip_StatusType

Return values

STATUS_FTFC_MEM_INFLS_IP_SUCCESS	Program performed successfully
STATUS_Ftfc_Mem_InFls_Ip_BUSY	Hardware program is still in progress
STATUS_FTFC_MEM_INFLS_IP_ERROR	There was an error during the hardware program.
STATUS_FTFC_MEM_INFLS_IP_ERROR_TIMEOUT	The program operation exceeded the timeout - Status value available only if the timeout feature is enabled.
STATUS_FTFC_MEM_INFLS_IP_ERROR_PROGRAM_VERIFY	The data was not written corectly into the memory - Status available only of program verify feature is enabled

Precondition

The module has to be initialized.

6.1.6.12 Ftfc_Mem_InFls_Ip_SetLoadAcStatus()

```
void Ftfc_Mem_InFls_Ip_SetLoadAcStatus (
            const boolean Status )
```

Set Status for Ftfc_Mem_InFls_Ip_LoadAc_Status.

Parameters

in	*Status*	

6.1.6.13 Ftfc_Mem_InFls_Ip_InvalidPrefetchBuff_Ram()

```
void Ftfc_Mem_InFls_Ip_InvalidPrefetchBuff_Ram (
             void  )
```

Invalidate prefetch buffer before reading to make sure that the driver always reads the new data from flash.

Parameters

in	*void*	

Return values

none	

6.2 MEM_43_INFLS Driver

6.2.1 Detailed Description

implement Mem_43_INFLS_IPW.h_Artifact

implement 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_CompareConfigType
  Mem_43_INFLS Compare 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_GET_VERSION_INFO_ID
  Service ID of function Mem_43_INFLS_GetVersionInfo.
- #define MEM_43_INFLS_GET_JOB_RESULT_ID
  Service ID of function Mem_43_INFLS_GetJobResult.
- #define MEM_43_INFLS_PROPAGATE_ERROR_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_BLANK_CHECK_ID
  Service ID of function Mem_43_INFLS_BlankCheck.
- #define MEM_43_INFLS_HW_SPECIFIC_SERVICE_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_HWSERVICEID_COMPARE
  Define ServiceId for Compare feature.
- #define MEM_43_INFLS_HWSERVICEID_INVALID
  Define ServiceId Invalid.
- #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_InstanceIdType
  Mem Instance Id Type.
- typedef uint32 Mem_43_INFLS_AddressType
  Mem Address Type.
- typedef uint32 Mem_43_INFLS_LengthType
  Mem Length Type.
- typedef uint8 Mem_43_INFLS_DataType
  Mem Data Type.
- typedef uint32 Mem_43_INFLS_HwServiceIdType
  Mem CRC Type.
- typedef void(∗ Mem_43_INFLS_ACCallbackPtrType) (void)

Mem Access Code Callback.

- typedef uint8 Mem_43_INFLS_BlockType

  Mem Hardware Service Id Type.
- typedef void(* Mem_43_INFLS_AcErasePtrType) (void(*CallBack) (void))

  Mem Access Code Pointer Erase Type.
- typedef void(* Mem_43_INFLS_AcWritePtrType) (void(*CallBack) (void))

  Mem Access Code Write Pointer Type.
- typedef Ftfc_Mem_InFls_Ip_ConfigType Mem_43_INFLS_InternalConfigType

  Mem Internal Flash Type.

Enum Reference

- enum Mem_43_INFLS_JobResultType

  Mem job result type.
- 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.
- Std_ReturnType Mem_43_INFLS_Suspend (Mem_43_INFLS_InstanceIdType InstanceId)

  Suspends active memory operation using hardware mechanism.
- Std_ReturnType 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_AddressType 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_HwServiceIdType 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_JobRuntimeIn *JobInfo)

  IP wrapper read function.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_Compare (uint32 InstanceIndex, Mem_43_INFLS_JobRuntim *JobInfo)

  IP wrapper Compare function.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_BlankCheck (uint32 InstanceIndex, Mem_43_INFLS_JobRun *JobInfo)

  IP wrapper blank check function.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_Write (uint32 InstanceIndex, Mem_43_INFLS_JobRuntimeIn *JobInfo)

  IP wrapper write function.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_Erase (uint32 InstanceIndex, Mem_43_INFLS_JobRuntimeIn *JobInfo)

  IP wrapper erase function.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_EraseSuspend (uint32 InstanceIndex)

  IP wrapper Erase Suspend function.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_EraseResume (uint32 InstanceIndex)

  IP wrapper Erase Resume function.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_GetJobResult (uint32 InstanceIndex, Mem_43_INFLS_JobTy JobType)

  Returns synchronously the result of the last job.
- Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_Cancel (uint32 InstanceIndex)

  Cancel an ongoing flash read, write, erase or compare job.
- void Mem_43_INFLS_IPW_ReportEccValueToLayerUnder (void)

  Report Ecc value result.

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 395 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 405 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 417 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 434 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 447 of file Mem_43_INFLS_Types.h.

6.2.2.6 struct Mem_43_INFLS_CompareConfigType

Mem_43_INFLS Compare Configuration Type.

Mem_43_INFLS Compare data structure for dataPtr of HwSpecificService

```
Mem_43_INFLS_CompareConfigType
```

Definition at line 465 of file Mem_43_INFLS_Types.h.

6.2.2.7 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 481 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 108 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 113 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 123 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 129 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 135 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 141 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 147 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 153 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 159 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 172 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 177 of file Mem_43_INFLS_Types.h.

6.2.3.12 MEM_43_INFLS_GET_VERSION_INFO_ID

```
#define MEM_43_INFLS_GET_VERSION_INFO_ID
```

Service ID of function Mem_43_INFLS_GetVersionInfo.

Definition at line 182 of file Mem_43_INFLS_Types.h.

6.2.3.13 MEM_43_INFLS_GET_JOB_RESULT_ID

```
#define MEM_43_INFLS_GET_JOB_RESULT_ID
```

Service ID of function Mem_43_INFLS_GetJobResult.

Definition at line 187 of file Mem_43_INFLS_Types.h.

6.2.3.14 MEM_43_INFLS_PROPAGATE_ERROR_ID

```
#define MEM_43_INFLS_PROPAGATE_ERROR_ID
```

Service ID of function Mem_43_INFLS_PropagateError.

Definition at line 192 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 197 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 202 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 209 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 214 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 219 of file Mem_43_INFLS_Types.h.

6.2.3.20 MEM_43_INFLS_BLANK_CHECK_ID

```
#define MEM_43_INFLS_BLANK_CHECK_ID
```

Service ID of function Mem_43_INFLS_BlankCheck.

Definition at line 224 of file Mem_43_INFLS_Types.h.

6.2.3.21 MEM_43_INFLS_HW_SPECIFIC_SERVICE_ID

```
#define MEM_43_INFLS_HW_SPECIFIC_SERVICE_ID
```

Service ID of function Mem_43_INFLS_HwSpecificService.

Definition at line 229 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 236 of file Mem_43_INFLS_Types.h.

6.2.3.23 MEM_43_INFLS_HWSERVICEID_COMPARE

```
#define MEM_43_INFLS_HWSERVICEID_COMPARE
```

Define ServiceId for Compare feature.

END All service IDs Hardware specific service request identifier

Definition at line 249 of file Mem_43_INFLS_Types.h.

6.2.3.24 MEM_43_INFLS_HWSERVICEID_INVALID

```
#define MEM_43_INFLS_HWSERVICEID_INVALID
```

Define ServiceId Invalid.

Definition at line 254 of file Mem_43_INFLS_Types.h.

6.2.3.25 MEM_43_INFLS_JOB_FLAG_NONE

```
#define MEM_43_INFLS_JOB_FLAG_NONE
```

Initial value.

END All service IDs

Definition at line 265 of file Mem_43_INFLS_Types.h.

6.2.3.26 MEM_43_INFLS_JOB_FLAG_STARTED

```
#define MEM_43_INFLS_JOB_FLAG_STARTED
```

Indicates that new job has been accepted.

Definition at line 270 of file Mem_43_INFLS_Types.h.

6.2.4 Types Reference

6.2.4.1 Mem_43_INFLS_InstanceIdType

```
typedef uint32 Mem_43_INFLS_InstanceIdType
```

Mem Instance Id Type.

Mem Instance Id Type

Definition at line 282 of file Mem_43_INFLS_Types.h.

6.2.4.2 Mem_43_INFLS_AddressType

```
typedef uint32 Mem_43_INFLS_AddressType
```

Mem Address Type.

Physical memory device address type

Definition at line 289 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 296 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 303 of file Mem_43_INFLS_Types.h.

6.2.4.5 Mem_43_INFLS_HwServiceIdType

```
typedef uint32 Mem_43_INFLS_HwServiceIdType
```

Mem CRC Type.

CRC computed over config set (will be implement in next feature) Mem_43_INFLS_CrcType_typedef typedef uint16 Mem_43_INFLS_CrcType;

Mem Hardware Service Id Type

Hardware specific service request identifier type

Definition at line 318 of file Mem_43_INFLS_Types.h.

6.2.4.6 Mem_43_INFLS_ACCallbackPtrType

```
typedef void(* Mem_43_INFLS_ACCallbackPtrType) (void)
```

Mem Access Code Callback.

Pointer type of Access Code Callback function.

Definition at line 324 of file Mem_43_INFLS_Types.h.

6.2.4.7 Mem_43_INFLS_BlockType

```
typedef uint8 Mem_43_INFLS_BlockType
```

Mem Hardware Service Id Type.

Hardware specific service request identifier type

Definition at line 332 of file Mem_43_INFLS_Types.h.

6.2.4.8 Mem_43_INFLS_AcErasePtrType

```
typedef void(* Mem_43_INFLS_AcErasePtrType) (void(*CallBack)(void))
```

Mem Access Code Pointer Erase Type.

Define pointer type of erase access code function.

Definition at line 338 of file Mem_43_INFLS_Types.h.

6.2.4.9 Mem_43_INFLS_AcWritePtrType

```
typedef void(* Mem_43_INFLS_AcWritePtrType) (void(*CallBack)(void))
```

Mem Access Code Write Pointer Type.

Define pointer type of write access code function.

Definition at line 344 of file Mem_43_INFLS_Types.h.

6.2.4.10 Mem_43_INFLS_InternalConfigType

```
typedef Ftfc_Mem_InFls_Ip_ConfigType Mem_43_INFLS_InternalConfigType
```

Mem Internal Flash Type.

Configuration structure of internal flash.

Definition at line 388 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.

Definition at line 356 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 369 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()

```
Std_ReturnType Mem_43_INFLS_Suspend (
             Mem_43_INFLS_InstanceIdType InstanceId )
```

Suspends active memory operation using hardware mechanism.

6.2.6.6 Mem_43_INFLS_Resume()

```
Std_ReturnType 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_Compare()

```
Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_Compare (
            uint32 InstanceIndex,
            Mem_43_INFLS_JobRuntimeInfoType * JobInfo )
```

IP wrapper Compare 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_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

6.2.6.18 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

6.2.6.19 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

6.2.6.20 Mem_43_INFLS_IPW_EraseSuspend()

```
Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_EraseSuspend (
            uint32 InstanceIndex )
```

IP wrapper Erase Suspend function.

Returns

Mem_43_INFLS_JobResultType

6.2.6.21 Mem_43_INFLS_IPW_EraseResume()

```
Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_EraseResume (
            uint32 InstanceIndex )
```

IP wrapper Erase Resume function.

Returns

Mem_43_INFLS_JobResultType

6.2.6.22 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

6.2.6.23 Mem_43_INFLS_IPW_Cancel()

```
Mem_43_INFLS_JobResultType Mem_43_INFLS_IPW_Cancel (
            uint32 InstanceIndex )
```

Cancel an ongoing flash read, write, erase or compare job.

Parameters

in	*InstanceIndex*	ID of the related memory driver instance.

Returns

Mem_43_INFLS_JobResultType

6.2.6.24 Mem_43_INFLS_IPW_ReportEccValueToLayerUnder()

```
void Mem_43_INFLS_IPW_ReportEccValueToLayerUnder (
            void  )
```

Report Ecc value result.

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.