User Manual

for S32K14X FLS Driver

Document Number: UM2FLSASR4.2 Rev0002R1.0.2
Rev. 1.0

Contents

Section number | **Title** | **Page**

Section number	Title	Page

Section number	Title	Page

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	26/04/2019	NXP MCAL Team	Updated version for ASR 4.2.2S32K14XR1.0.2

Chapter 2 Introduction

This User Manual describes NXP Semiconductors AUTOSAR Flash (FLS) driver for S32K14X.

AUTOSAR FLS driver configuration parameters and deviations from the specification are described in FLS Driver chapter of this document. AUTOSAR FLS driver requirements and APIs are described in the AUTOSAR FLS driver software specification document.

2.1 Supported Derivatives

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

Table 2-1. S32K14X Derivatives

NXP Semiconductors	s32k148_lqfp144, s32k148_lqfp176, s32k148_mapbga100, s32k146_lqfp144, s32k146_lqfp100, s32k146_lqfp64, s32k146_mapbga100, s32k144_lqfp100, s32k144_lqfp64, s32k144_mapbga100, s32k142_lqfp100, s32k142_lqfp64, s32k118_lqfp48, s32k118_lqfp64, s32k142_lqfp48, s32k144_lqfp48, s32k148_lqfp100

All of the above microcontroller devices are collectively named as S32K14X .

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 type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is 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.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
DET	Default Error Tracer
ECC	Error Correcting Code
VLE	Variable Length Encoding
N/A	Not Applicable
MCU	Micro Controller Unit
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory

Table continues on the next page...

Table 2-2. Acronyms and Definitions (continued)

Term	Definition
FEE	Flash EEPROM Emulation
FLS	Flash
XML	Extensible Markup Language

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	Specification of FLS Driver	AUTOSAR Release 4.2.2
2	S32K14X Reference Manual	Reference Manual, Rev. 9, 9/2018
3	S32K142 Mask Set Errata for Mask 0N33V (0N33V)	30/11/2017
4	S32K144 Mask Set Errata for Mask 0N57U (0N57U)	30/11/2017
5	S32K146 Mask Set Errata for Mask 0N73V (0N73V)	30/11/2017
6	S32K148 Mask Set Errata for Mask 0N20V (0N20V)	25/10/2018
7	S32K118 Mask Set Errata for Mask 0N97V (0N97V)	07/01/2019

Chapter 3
Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 4.2 Rev0002FLS Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

- Linear Address.

The FLS driver provides services for reading, writing and erasing flash memory and it combines configured flash memory sectors into one linear address space.

The FLS module shall combine all available flash memory areas into one linear address space, it will always start at address 0 and continues without any gap.

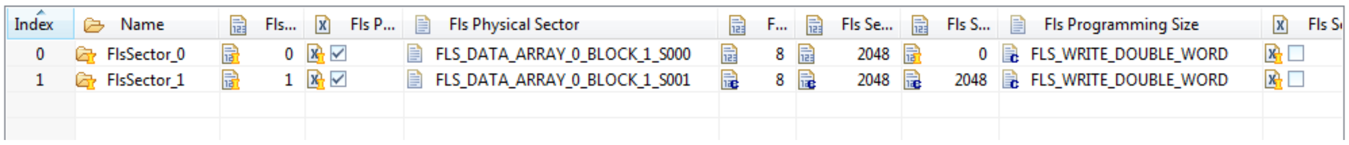
Example:

Suppose user want to configure following sectors:

Table 3-1. Sectors details Example

FlsPhysicalSector	Fls Physical Start Address	Fls Sector Size
FLS_DATA_ARRAY_0_BLOCK_1_S000	0x10000000	0x1000
FLS_DATA_ARRAY_0_BLOCK_1_S001	0x10001000	0x1000

The FlsSector List should be configured in the following way:



Index	Name	Fls...	Fls P...	Fls Physical Sector	F...	Fls Se...	Fls S...	Fls Programming Size	Fls S
0	FlsSector_0	0	✓	FLS_DATA_ARRAY_0_BLOCK_1_S000	8	2048	0	FLS_WRITE_DOUBLE_WORD	☐
1	FlsSector_1	1	✓	FLS_DATA_ARRAY_0_BLOCK_1_S001	8	2048	2048	FLS_WRITE_DOUBLE_WORD	☐

Figure 3-1. Fls Sector List

As you can see "Fls Sector Start Address" for FlsSector_0 will be 0 and "Fls Sector Start Address" for FlsSector_1 will be 0x1000(4096)

If user want to write FLS_DATA_ARRAY_0_BLOCK_1_S001, user need to write to the logical address 0x1000 - 0x1FFF.

If user want to erase it, user need to erase sector from logical address 0x1000 with size 0x1000.

Note: The user do not need to calculate the "Fls Sector Start Address" and "Fls Sector Size" they can be automatically computed.

- Page Programming Size.

The FLS driver for the platform S32K14X supports three different write.

- For internal flash sectors, the avaialable values are:
FLS_WRITE_DOUBLE_WORD(8bytes)

- For external flash sectors, the avaialable values are:
FLS_WRITE_128BYTES_PAGE(128bytes),
FLS_WRITE_256BYTES_PAGE(256bytes),
FLS_WRITE_512BYTES_PAGE(512bytes)

User has to select any one option from the above as desired for user's application. The Default value of this is a Double Word Write (8 bytes).

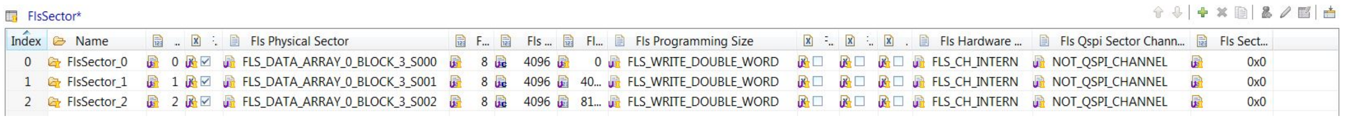
Example:

Suppose user want to configure following sectors:

Table 3-2. Page Programming Size Example

FlsPhysicalSector	Fls Page programming Size
FLS_DATA_ARRAY_0_BLOCK_3_S000	FLS_WRITE_DOUBLE_WORD
FLS_DATA_ARRAY_0_BLOCK_3_S001	FLS_WRITE_DOUBLE_WORD
FLS_DATA_ARRAY_0_BLOCK_3_S002	FLS_WRITE_DOUBLE_WORD

The FlsSector List should be configured in the following way:



FlsSector*

Index	Name	..	:	Fls Physical Sector	F...	Fls ...	Fl...	Fls Programming Size	-..	:..	.	Fls Hardware ...	Fls Qspi Sector Chann...	Fls Sect...
0	FlsSector_0	0	✓	FLS_DATA_ARRAY_0_BLOCK_3_S000	8	4096	0	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
1	FlsSector_1	1	✓	FLS_DATA_ARRAY_0_BLOCK_3_S001	8	4096	40...	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
2	FlsSector_2	2	✓	FLS_DATA_ARRAY_0_BLOCK_3_S002	8	4096	81...	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0

Figure 3-2. Fls Sector List

- Unlocking FLS Sectors.

The Flash memory physical sectors that are going to be modified by Fls driver (i.e. erase and write operations) have to be **unprotected** for a successful operation.

Sector **locking** is different than sector **protection**. Sector locking is a configuration state, used by the driver initialization to check if a sector is protected or not. Sector protection is a hardware state, which controls the write/erase permission. A sector unlocked in configuration, it is checked upon driver initialization(Fls_Init) for its protection state. If it is protected in hardware, the initialization will fail. A sector locked in configuration, it is checked for its protection state in the write/erase job processing. If it is protected in hardware, the write/erase will fail.

Sector unprotect is not handled by FLS driver and must be setup on an application level.

For external sectors, due to various memory implementations and options, the FLS driver does not handle the sector protection and it is the responsibility of the application to provide the configured sectors in a state in which the driver is able to operate on them. In order to aid this, at the end of Fls_Init() function, a user configurable callout(FlsQspiInitCallout) is provided, in which the application can interrogate and configure the protection settings accordingly.

It is recommended to configure only those FlsPhysicalSector(s) that are required by upper layer module FEE. As FLS driver can access only those configured FlsPhysicalSector(s) and can not modify rest of Flash address space.

Note: If the microcontroller is in user mode, be sure that the Flash memory controller registers are accessible.

For more information please refer to the 'Memory Protection Unit' and 'Register Protection' chapters in the device reference manual.

Note: Care should be taken if using the first program flash sector (FLS_CODE_ARRAY_0_BLOCK_0_S000), as that sector contains the flash configuration field. It is recommended to avoid using this sector, because the flash configuration field region has a default value different than the default erased value. Writing in improper value into the flash configuration field region(0x0_0400 - 0x0_040F) might lead to protected sectors, secured device or permanently secured device.

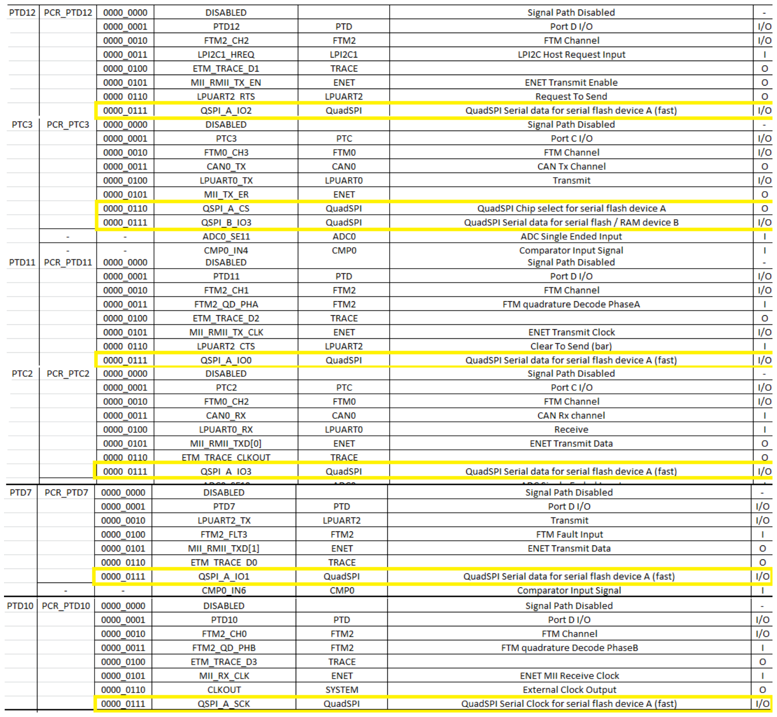
For more information you can also refer to the Fls driver IM document. An example of unprotecting the internal flash sectors is provided in this manual.

- Application's tasks.

It is responsibility of integrator/application to ensure that MCU-wide parameters like voltage supply etc. are according to and in limits specified in MCU documentation. Integrator/application is responsible to implement additional functionality that cancel any on-going erase/write Fls jobs if MCU conditions are not in such limits.

3.3 Hardware Resources

For external flash: Configure pins for QSPI pin mux, driver strength enable, pull up enabled.



PTD12	PCR_PTD12	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTD12	PTD	Port D I/O	I/O
		0000_0010	FTM2_CH2	FTM2	FTM Channel	I/O
		0000_0011	LPI2C1_HREQ	LPI2C1	LPI2C Host Request Input	I
		0000_0100	ETM_TRACE_D1	TRACE		O
		0000_0101	MII_RMII_TX_EN	ENET	ENET Transmit Enable	O
		0000_0110	LPUART2_RTS	LPUART2	Request To Send	O
		0000_0111	QSPI_A_IO2	QuadSPI	QuadSPI Serial data for serial flash device A (fast)	I/O
PTC3	PCR_PTC3	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTC3	PTC	Port C I/O	I/O
		0000_0010	FTM0_CH3	FTM0	FTM Channel	I/O
		0000_0011	CAN0_TX	CAN0	CAN Tx Channel	O
		0000_0100	LPUART0_TX	LPUART0	Transmit	I/O
		0000_0101	MII_TX_ER	ENET		O
		0000_0110	QSPI_A_CS	QuadSPI	QuadSPI Chip select for serial flash device A	O
		0000_0111	QSPI_B_IO3	QuadSPI	QuadSPI Serial data for serial flash / RAM device B	I/O
	-	-	ADC0_SE11	ADC0	ADC Single Ended Input	I
	-	-	CMP0_IN4	CMP0	Comparator Input Signal	I
PTD11	PCR_PTD11	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTD11	PTD	Port D I/O	I/O
		0000_0010	FTM2_CH1	FTM2	FTM Channel	I/O
		0000_0011	FTM2_QD_PHA	FTM2	FTM quadrature Decode PhaseA	I
		0000_0100	ETM_TRACE_D2	TRACE		O
		0000_0101	MII_RMII_TX_CLK	ENET	ENET Transmit Clock	I/O
		0000_0110	LPUART2_CTS	LPUART2	Clear To Send (bar)	I
		0000_0111	QSPI_A_IO0	QuadSPI	QuadSPI Serial data for serial flash device A (fast)	I/O
PTC2	PCR_PTC2	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTC2	PTC	Port C I/O	I/O
		0000_0010	FTM0_CH2	FTM0	FTM Channel	I/O
		0000_0011	CAN0_RX	CAN0	CAN Rx channel	I
		0000_0100	LPUART0_RX	LPUART0	Receive	I
		0000_0101	MII_RMII_TXD[0]	ENET	ENET Transmit Data	O
		0000_0110	ETM_TRACE_CLKOUT	TRACE		O
		0000_0111	QSPI_A_IO3	QuadSPI	QuadSPI Serial data for serial flash device A (fast)	I/O
PTD7	PCR_PTD7	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTD7	PTD	Port D I/O	I/O
		0000_0010	LPUART2_TX	LPUART2	Transmit	I/O
		0000_0100	FTM2_FLT3	FTM2	FTM Fault Input	I
		0000_0101	MII_RMII_TXD[1]	ENET	ENET Transmit Data	O
		0000_0110	ETM_TRACE_D0	TRACE		O
		0000_0111	QSPI_A_IO1	QuadSPI	QuadSPI Serial data for serial flash device A (fast)	I/O
	-	-	CMP0_IN6	CMP0	Comparator Input Signal	I
PTD10	PCR_PTD10	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTD10	PTD	Port D I/O	I/O
		0000_0010	FTM2_CH0	FTM2	FTM Channel	I/O
		0000_0011	FTM2_QD_PHB	FTM2	FTM quadrature Decode PhaseB	I
		0000_0100	ETM_TRACE_D3	TRACE		O
		0000_0101	MII_RX_CLK	ENET	ENET MII Receive Clock	I
		0000_0110	CLKOUT	SYSTEM	External Clock Output	O
		0000_0111	QSPI_A_SCK	QuadSPI	QuadSPI Serial Clock for serial flash device A (fast)	I/O

Figure 3-3. Hardware resource.

3.3.1 S32K14X Flash Banks/Arrays, Sectors details

For S32K142: has 256 KBytes of code flash(program flash) and 64 KBytes of data flash(FlexNVM).

Table 3-3. For S32K142 sectors details

Type	Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	S000	2
...	...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	S000	2
...	...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	S127	2

For S32K144: has 256 KBytes of code flash(program flash) and 64 KBytes of data flash(FlexNVM).

Table 3-4. For S32K144 sectors details

Type	Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	S000	2
...	...	...
FLS_DATA_ARRAY_0_BLOCK_1_S031	S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	S000	4
...	...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	S127	4

For S32K146: has 1 MBytes of code flash(program flash) and 64 KBytes of data flash(FlexNVM).

Table 3-5. For S32K146 sectors details

Type	Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_2_S000	S000	2
...	...	...
FLS_DATA_ARRAY_0_BLOCK_2_S031	S031	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	S000	4
...	...	...
FLS_CODE_ARRAY_0_BLOCK_1_S255	S255	4

For S32K148: has 1.5 MBytes of code flash(program flash) and 512 KBytes of data flash(FlexNVM).

Table 3-6. For S32K148 sectors details

Type	Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_3_S000	S000	4
...	...	...
FLS_DATA_ARRAY_0_BLOCK_3_S127	S127	4
FLS_CODE_ARRAY_0_BLOCK_0_S000	S000	4
...	...	...
FLS_CODE_ARRAY_0_BLOCK_2_S383	S383	4

For S32K118: has 256 KBytes of code flash(program flash) and 32 KBytes of data flash(FlexNVM).

Table 3-7. For S32K118 sectors details

Type	Sector name	Sector Size (KB)
FLS_DATA_ARRAY_0_BLOCK_1_S000	S000	2
...	...	...
FLS_DATA_ARRAY_0_BLOCK_1_S015	S015	2
FLS_CODE_ARRAY_0_BLOCK_0_S000	S000	2
...	...	...
FLS_CODE_ARRAY_0_BLOCK_0_S127	S127	2

3.3.2 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 other 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 */
```

For external flash sectors, the FlsQspiInitCallout is provided for the application in order to check and modify the state of the external memory sectors, so that at the end of the driver initialization the external sectors to be in an appropriate state.

3.4 Deviation from Requirements

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

There are also some additional requirements (on top of requirements detailed in AUTOSAR FLS Driver software specification) which need to be satisfied for correct operation.

Table 3-8. Deviations Status Column Description

Term	Definition
N/A	Not Available
N/T	Not Testable
N/S	Out of Scope
N/I	Not Implemented
N/F	Not Fully Implemented
I/D	Implemented with Deviation

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

Table 3-9. Driver Deviations Table

SW Requirement ID	Status	Description	Notes
SWS_Fls_00107	N/F	The Fls module shall comply with the following file structure: Figure1 File include structure (see Figure image2.emf)	Not fully compliant
SWS_Fls_00004	N/F	(see Table Table_d2e28879.html)	FLS_E_TIMEOUT not implemented; FLS_E_UNEXPECTED_FLASH_ID is applicable only for external Fls driver

Table continues on the next page...

Table 3-9. Driver Deviations Table (continued)

SW Requirement ID	Status	Description	Notes
SWS_Fls_00015	I/D	If development error detection for the module Fls is enabled: the function Fls_Init shall check the (hardware specific) contents of the given configuration set <continue>	The check of the CRC computed over selected parameters of the configuration set is done independently of the development error detection setting. The setting itself just controls the DET reporting (FLS_E_PARAM_CONFIG).
SWS_Fls_00144	N/A	During the initialization of the external flash driver, the FLS module shall check the hardware ID of the external flash device against the corresponding published parameter. <continue>	Applicable only for external Fls driver
SWS_Fls_00272	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall provide a timeout monitoring for the currently running job, that is it shall supervise the deadline of the read / compare / erase or write job.	The timeout monitoring is provided independently of development error detection setting. Instead its own pre-compile switch is provided (see Form FlsTimeouts).
SWS_Fls_00359	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the configured maximum erase time <continue>	The same as for SWS_Fls_00272.
SWS_Fls_00360	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the expected maximum write time <continue>	The same as for SWS_Fls_00272.
SWS_Fls_00361	I/D	The development error code FLS_E_TIMEOUT shall be reported when the timeout supervision of a read, write, erase or compare job failed.	The same as for SWS_Fls_00272. See also SWS_Fls_00362's Note.
SWS_Fls_00362	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the expected maximum read / compare <continue>	Timeout check was implemented only for the operations whose termination is HW-dependent (erase/write).
SWS_Fls_00215	N/F	The FLS module's flash access routines shall only disable interrupts and wait for the completion of the erase / write command if necessary (that is if it has to be ensured that no other code is executed in the meantime).	Only RTE plug-in has the ability to enable/ disable interrupts. Additionaly there is possibility to alter default behaviour and have Erase/Write jobs asynchronous, i.e. Fls_MainFunction function doesn't wait (block) for completion of the erase sector/ page write operation(s).
SWS_Fls_00217	N/A	The FLS module shall add a device specific base address to the address type Fls_AddressType if necessary.	Unclear concept: device specific base address Not used
SWS_Fls_00208	N/F	The FLS module shall combine all available flash memory areas into one linear address space (denoted by the parameters FlsBaseAddress and FlsTotalSize).	Unclear Pourpose. FlsBaseAddress and FlsTotalSize not used. Impacted requitements are: SWS_Fls_00221, SWS_Fls_00020, SWS_Fls_00226, SWS_Fls_00026, SWS_Fls_00239, SWS_Fls_00097, SWS_Fls_00244, SWS_Fls_00150,

Table continues on the next page...

Table 3-9. Driver Deviations Table (continued)

SW Requirement ID	Status	Description	Notes
ECUC_Fls_00169	N/I	FlsBaseAddress	FlsBaseAddress not used. Unclear purpose
ECUC_Fls_00170	N/I	FlsTotalSize	FlsTotalSize not used. Unclear purpose
SWS_Fls_00145	I/D	If possible, e.g. with interrupt controlled implementations, the FLS module shall start the first round of the erase job directly within the function Fls_Erase to reduce overall runtime.	Implemented with deviation, interrupt supported only for external memory if supported on current platform.
SWS_Fls_00146	I/D	If possible, e.g. with interrupt controlled implementations, the FLS module shall start the first round of the write job directly within the function Fls_Write to reduce overall runtime.	Implemented with deviation, interrupt supported only for external memory if supported on current platform.
SWS_Fls_00247	N/F	If source code for caller and callee of the function Fls_GetVersionInfo is available, the FLS module should realize this function as a macro. The FLS module should define this macro in the module's header file.	The function will be implemented as function, not as a macro.
SWS_Fls_00040	N/F	The function Fls_MainFunction shall only process as much data in one call cycle as statically configured for the current job type (read, write, erase or compare) and the current FLS module's operating mode (normal, fast).	For Erase job not applicable as whole sector(s) is(are) erased.
SWS_Fls_00022	I/D	If development error detection for the module Fls is enabled: After a flash block has been erased, the function Fls_MainFunction shall compare <continue>	Functionality available if both FlsDevErrorDetect and FlsEraseBlankCheck configured to true. If only FlsEraseBlankCheck configured to true the DET error is not reported but Fls job ends with MEMIF_JOB_FAILED.
SWS_Fls_00055	I/D	If development error detection for the module Fls is enabled: Before writing a flash block, the function Fls_MainFunction shall compare <continue>	Functionality available if both FlsDevErrorDetect and FlsWriteBlankCheck configured to true. If only FlsWriteBlankCheck configured to true the DET error is not reported but Fls job ends with MEMIF_JOB_FAILED.
SWS_Fls_00056	I/D	If development error detection for the module Fls is enabled:: After writing a flash block, the function Fls_MainFunction shall compare <continue>	Functionality available if both FlsDevErrorDetect and FlsWriteVerifyCheck configured to true. If only FlsWriteVerifyCheck configured to true the DET error is not reported but Fls job ends with MEMIF_JOB_FAILED.
SWS_Fls_00232	I/D	The configuration parameter FlsUseInterrupts shall switch between interrupt and polling controlled job processing if this is supported by the flash memory hardware.	Implemented with deviation, interrupt supported only for external memory if supported on current platform.
SWS_Fls_00233	I/D	The FLS module's implementer shall locate the interrupt service routine in Fls_Irq.c.	Implemented with deviation, interrupt supported only for external memory if supported on current platform.

Table continues on the next page...

Table 3-9. Driver Deviations Table (continued)

SW Requirement ID	Status	Description	Notes
SWS_Fls_00234	I/D	If interrupt controlled job processing is supported and enabled with the configuration parameter FlsUseInterrupts, the interrupt service routine shall <continue>	Implemented with deviation, interrupt supported only for external memory if supported on current platform.
ECUC_Fls_00292	I/D	FlsUseInterrupts	Implemented with deviation, interrupt supported only for external memory if supported on current platform.
SWS_Fls_00196	N/I	The function Fls_MainFunction shall at the most issue one sector erase command (to the hardware) in each cycle.	Implementation now erases only one sector per cycle but the HW allows erasing of more physical sectors in parallel (but the final erase time is not reduced).
ECUC_Fls_00306	N/I	FlsCallCycle	FlsCallCycle not used, unclear purpose.
ECUC_Fls_00280	N/I	FlsNumberOfSectors	FlsNumberOfSectors not used, unclear purpose.
ECUC_Fls_00279	N/I	FlsProtection {FLS_PROTECTION} Erase/write protection settings. Only relevant if supported by hardware. (see Table TableConf_d2e40923.html)	FlsProtection: not used. Replaced by Vendor specific parameter (see PR-MCAL-3158)
SWS_Fls_00302	N/I	The module's status, mode and the job result shall be made available for debugging (reading)	Support for Debugging shall not be implemented according to PR-MCAL-3330.fls.

As a deviation from standard:

Fls_[VariantName]_PBcfg.c files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB) Fls_Cfg.c, Fls_Cfg.h file will contain the definition for all parameters that are not variant aware

3.5 Driver limitations

None

3.6 Driver usage and configuration tips

Async/sync mode

3.6.1 Introduction

For internal flash sectors, it's possible to modify the behavior of sector erase / page write using two configuration parameters (FlsSectorEraseAsynch, FlsPageWriteAsynch) in FlsSector TAB.

If FlsSectorEraseAsynch/FlsPageWriteAsynch are enabled sector erase / page write job in the Fls_MainFunction are executed asynchronously, it means that Fls_MainFunction will not wait (not blocking) for completion of high voltage operation.

If FlsSectorEraseAsynch/FlsPageWriteAsynch are disabled sector erase / page write job are executed synchronously, which means sector erase / page write job are blocking and any high voltage operation will be completed during one Fls_Mainfunction.

For external flash sectors, if they are available, it's possible to modify the behavior of the sector erase / page write and also of the read operation, using three configuration parameters (FlsSectorEraseAsynch, FlsPageWriteAsynch and FlsSectorIrqMode) in FlsSector TAB.

The ASYNC and IRQ modes are not compatible and only one can be chosen. The SYNC and ASYNC modes for external sectors are similar to the internal sectors. The IRQ mode launches the external QuadSPI transaction in the first iteration of the Fls_MainFunction. The job is continued and finalized on a separate path in interrupt context. The Fls_MainFunction periodically polls the status and monitors the timeout if enabled. Upon completion, the Fls_MainFunction schedules the next part of the current job or finalizes the job.

Selecting IRQ mode for a sector will force all operations to be performed in interrupt mode, thus will restrict read operations to be performed using the slower IP interface. It is recommended to select a mode(SYNC/ASYNC) which allows performing read operations using the faster AHB read interface. For write and erase jobs, the internal memory time required for the high voltage operation to complete is much larger than the software time it takes for the data to reach the memory over the QuadSPI lines, thus no significant improvement is obtained selecting IRQ mode over the SYNC/ASYNC modes. Also, taking into account the time duration for an external high voltage operation, there is usually little benefit in selecting DDR modes or Quad line modes for write operations.

3.6.2 Avoiding RWW problem

To avoid RWW (Read While Write) problems on the internal flash, the FLS driver provides the FlsAcLoadOnJobStart configuration parameter. If it is set to true the Fls 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 cancelled.

FlsAcLoadOnJobStart functionality can be used only in case of Sync Mode, in which case the flash access code is loaded to RAM and therefore the flash driver shouldn't have RWW problems; if FlsAcLoadOnJobStart 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.

In case of Async operations it is only possible to erase / write to flash array different from flash array the application is executing from. This mode is usable only if the platform supports different Read While Write partitions or if the entire code is executed from RAM, during the flash modify operation.

Note:

1. For internal flash, the flash driver use the sector erase / page write access code to clear the FSTAT:CCIF bit and wait for completion of high voltage operation (and therefore incompatible with Async operation).

2. The flash module is 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. FlsAcCallback should be in located in RAM if FlsACLoadOnJob is true to avoid RWW problem.

3.7 Runtime Errors

The driver supports runtime generation of the errors listed in the Table Runtime Errors. The runtime error reporting can be disabled globally (see Form FlsGeneral).

Table 3-10. Runtime Errors

Function	Error Code	Condition triggering the error
Fls_Flash_AbortSuspended()	FLS_E_ERASE_FAILED	Abort of the ongoing erase operation failed (in case of detected timeout event).
Fls_Flash_AbortSuspended()	FLS_E_WRITE_FAILED	Abort of the ongoing write operation failed (in case of detected timeout event).
Fls_Flash_Init()	FLS_E_ERASE_FAILED	Abort of the ongoing erase operation failed (in case of detected timeout event).
Fls_Flash_Init()	FLS_E_ERASE_FAILED	Resuming the suspended erase operation failed.
Fls_Flash_Init()	FLS_E_WRITE_FAILED	Abort of the ongoing write operation failed (in case of detected timeout event).

Table continues on the next page...

Table 3-10. Runtime Errors (continued)

Function	Error Code	Condition triggering the error
Fls_Flash_Init()	FLS_E_WRITE_FAILED	Resuming the suspended write operation failed.
Fls_Flash_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed.
Fls_Flash_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed (in case of interleaved blocks).
Fls_Flash_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed (in case of detected timeout event).
Fls_Flash_MainFunction()	FLS_E_WRITE_FAILED	Async Write operation failed.
Fls_Flash_MainFunction()	FLS_E_WRITE_FAILED	Async Write operation failed (in case of detected timeout event).
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed.
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed (in case of interleaved blocks).
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed. Sector is locked, must be unlocked before an HV operation can be set
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed (in case of interleaved blocks).
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed.
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed (in case of interleaved blocks).
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed (and previous two cases).
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed.
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed (in case of interleaved blocks).
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed. Sector is locked, must be unlocked before an HV operation can be set
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed (in case of interleaved blocks). Sector is locked, must be unlocked before an HV operation can be set
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Sync Write operation failed.
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Sync Write operation failed (in case of interleaved blocks).
Fls_Qspi_MainFunctionErase()	FLS_E_ERASE_FAILED	Error flags were set during programming
Fls_Qspi_ProcessCombinedIrq()	FLS_E_WRITE_FAILED	Incorrect IRQ state for a write job.
Fls_Qspi_MainFunctionErase()	FLS_E_ERASE_FAILED	Clear the errors (w1c).
Fls_Qspi_MainFunctionErase()	FLS_E_ERASE_FAILED	Check error from external flash chip status.
Fls_Qspi_MainFunctionErase()	FLS_E_ERASE_FAILED	Error, the write operation has timed out
Fls_Qspi_MainFunctionWrite()	FLS_E_WRITE_FAILED	Error flags were set during programming.
Fls_Qspi_MainFunctionWrite()	FLS_E_WRITE_FAILED	Error, Write operation failed.
Fls_Qspi_MainFunctionWrite()	FLS_E_WRITE_FAILED	Check if there is any error from external flash chip

Table continues on the next page...

Table 3-10. Runtime Errors (continued)

Function	Error Code	Condition triggering the error
Fls_Qspi_MainFunctionWrite()	FLS_E_WRITE_FAILED	Error, the memory locations were not correctly programmed or there was an error when attempting to read it.
Fls_Qspi_MainFunctionWrite()	FLS_E_WRITE_FAILED	Clear the errors (w1c)
Fls_Qspi_MainFunctionWrite()	FLS_E_WRITE_FAILED	Error, the Write command sequence failed.
Fls_Qspi_MainFunctionWrite()	FLS_E_WRITE_FAILED	Error, the write operation has timed out.
Fls_DoJobRead()	FLS_E_READ_FAILED	A non correctable ECC error is present at read location.
Fls_DoJobCompare()	FLS_E_COMPARE_FAILED	A non correctable ECC error is present at read location.
Fls_Qspi_Init()	FLS_E_UNEXPECTED_FLASH_ID	Configuration error.

3.8 Development Error Description

Table 3-11. Development Error Description

Error Code	Value	Condition triggering the error
FLS_E_PARAM_CONFIG	1	API service called with wrong parameter
FLS_E_PARAM_ADDRESS	2	u32TargetAddress is not in range and aligned to first byte of flash sector
FLS_E_PARAM_LENGTH	3	u32TargetAddress is not in range and aligned to last byte of flash sector
FLS_E_PARAM_DATA	4	NULL_PTR == SourceAddressPtr
FLS_E_UNINIT	5	API service called without module initialization
FLS_E_BUSY	6	API service called while driver still busy
FLS_E_VERIFY_ERASE_FAILED	7	Erase verification (blank check) failed
FLS_E_VERIFY_WRITE_FAILED	8	Write verification (compare) failed
FLS_E_PARAM_POINTER	10	NULL_PTR passed
FLS_E_PARTITION_ERR	11	FlsCheckFlexNvmRatio is enabled and D-Flash is not configured exclusively flash usage.
FLS_E_SECTOR_PROTECTED	12	Configured internal sector is protected against write/erase operations.

3.9 Software specification

The following sections contains driver software specifications.

3.9.1 Define Reference

Constants supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.3 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.3.1 Function Fls_Cancel

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

Details:

Abort a running job synchronously so that directly after returning from this function a new job can be started.

Pre: The module must be initialized.

Post: Fls_Cancelchanges module status andFls_eJobResultinternal variable.

Prototype: `void Fls_Cancel(void);`

Note: If external memory(QuadSPI) is supported on current platform, the Fls_Cancel function will abort the running job synchronously. The subsequent job will check the external memory and poll the idle status. If the external memory is still busy and if configured, the FlsQspiResetCallout is provided and called periodically in order to provide to the application a means of aborting the hardware job also, if supported by the external memory(ex: toggle the reset pin, send a reset command, etc).

3.9.3.2 Function Fls_Compare

Compares a flash memory area with an application data buffer.

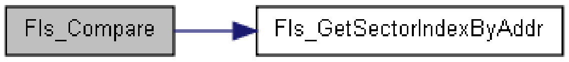


Figure 3-4. Function Fls_Compare References.

Details:

Starts a compare job asynchronously. The actual job is performed byFls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_Readchanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_pJobDataSrcPtr,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Compare(Fls_AddressType u32SourceAddress, const uint8 *pTargetAddressPtr, Fls_LengthType u32Length);`

Table 3-12. Fls_Compare Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32SourceAddress	**input**	Source address in flash memory.
const uint8 *	pTargetAddressPtr	**input**	Pointer to source data buffer.
`Fls_LengthType`	u32Length	**input**	Number of bytes to compare.

Table 3-13. Fls_Compare Return Values

Name	Description
E_OK	Compare command has been accepted.
E_NOT_OK	Compare command has not been accepted.

3.9.3.3 Function Fls_Erase

Erase one or more complete flash sectors.

Details:

Starts an erase job asynchronously. The actual job is performed by theFls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_Erasechanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobSectorEnd,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Erase(Fls_AddressType u32TargetAddress, Fls_LengthType u32Length);`

Table 3-14. Fls_Erase Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32TargetAddress	**input**	Target address in flash memory.
`Fls_LengthType`	u32Length	**input**	Number of bytes to erase.

Table 3-15. Fls_Erase Return Values

Name	Description
E_OK	Erase command has been accepted.
E_NOT_OK	Erase command has not been accepted.

3.9.3.4 Function Fls_GetJobResult

Returns the result of the last job.

Details:

Returns synchronously the result of the last job.

Return: MemIf_JobResultType.

Prototype: `MemIf_JobResultType Fls_GetJobResult(void);`

Table 3-16. Fls_GetJobResult Return Values

Name	Description
MEMIF_JOB_OK	Successfully completed job.
MEMIF_JOB_FAILED	Not successfully completed job.
MEMIF_JOB_PENDING	Still pending job (not yet completed).
MEMIF_JOB_CANCELED	Job has been canceled.
MEMIF_BLOCK_INCONSISTENT	Inconsistent block requested, it may contains corrupted data.
MEMIF_BLOCK_INVALID	Invalid block requested.

3.9.3.5 Function Fls_GetStatus

Returns the FLS module status.

Details:

Returns the FLS module status synchronously.

Return: MemIf_StatusType.

Prototype: `MemIf_StatusType Fls_GetStatus(void);`

Table 3-17. Fls_GetStatus Return Values

Name	Description
MEMIF_UNINIT	Module has not been initialized (yet).
MEMIF_IDLE	Module is currently idle.
MEMIF_BUSY	Module is currently busy.

3.9.3.6 Function Fls_GetVersionInfo

Returns version information about FLS module.

Details:

Version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

Prototype: `void Fls_GetVersionInfo(Std_VersionInfoType *pVersionInfoPtr);`

Table 3-18. Fls_GetVersionInfo Arguments

Type	Name	Direction	Description
Std_VersionInfoType *	pVersionInfoPtr	**input, output**	Pointer to where to store the version information of this module.

3.9.3.7 Function Fls_Init

The function initializes Fls module.

Details:

The function sets the internal module variables according to given configuration set.

Pre: pConfigPtr must not be NULL_PTR and the module status must not be MEMIF_BUSY.

Prototype: `void Fls_Init(const Fls_ConfigType *pConfigPtr);`

Table 3-19. Fls_Init Arguments

Type	Name	Direction	Description
const`Fls_ConfigType`*	pConfigPtr	**input**	Pointer to flash driver configuration set.

Note: If external memory(QuadSPI) is supported on current platform, the Fls_Init function will configure the QuadSPI IP and also provide an user configurable callout(FlsQspiInitCallout) at the end of the initialization sequence. The callout is intended to be used for memory specific extra configurations(sector protection, non volatile settings, data learning, etc), so that at the return of it the memory is left in the state required by the application and driver.

3.9.3.8 Function Fls_MainFunction

Performs actual flash read, write, erase, compare and blank check jobs.

Details:

Bytes number processed per cycle depends by job type (erase, write, read, compare, blank check) current FLS module's operating mode (normal, fast) and write, erase Mode of Execution (sync, async).

Pre: The module has to be initialized.

Note

This function have to be called cyclically by the Basic Software Module; it will do nothing if there aren't pending job.

Prototype: `void Fls_MainFunction(void);`

3.9.3.9 Function Fls_Read

Reads from flash memory.

Details:

Starts a read job asynchronously. The actual job is performed byFls_MainFunction.

Return: MemIf_JobResultType.

Pre: The module has to be initialized and not busy.

Post: Fls_Readchanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_pJobDataDestPtr,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Read(Fls_AddressType u32SourceAddress, uint8 *pTargetAddressPtr, Fls_LengthType u32Length);`

Table 3-20. Fls_Read Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32SourceAddress	**input**	Source address in flash memory.
`Fls_LengthType`	u32Length	**input**	Number of bytes to read.
uint8 *	pTargetAddressPtr	**output**	Pointer to target data buffer.

Table 3-21. Fls_Read Return Values

Name	Description
MEMIF_JOB_OK	Successfully completed job.
MEMIF_JOB_FAILED	Not successfully completed job.
MEMIF_JOB_PENDING	Still pending job (not yet completed).
MEMIF_JOB_CANCELED	Job has been canceled.
MEMIF_BLOCK_INCONSISTENT	Inconsistent block requested, it may contains corrupted data.
MEMIF_BLOCK_INVALID	Invalid block requested.

3.9.3.10 Function Fls_SetMode

Sets the FLS module's operation mode to the given Mode.

Details:

Every given mode determinates maximum bytes for read/write operations. Every mode has a set of pre-configured values.

Pre: The module has to be initialized and not busy.

Post: Fls_SetModechanges internal variablesFls_u32MaxReadandFls_u32MaxWrite.

Prototype: `void Fls_SetMode(MemIf_ModeType Mode);`

Table 3-22. Fls_SetMode Arguments

Type	Name	Direction	Description
MemIf_ModeType	Mode	**input**	MEMIF_MODE_FAST or MEMIF_MODE_SLOW.

3.9.3.11 Function Fls_Write

Write one or more complete flash pages to the flash device.

Details:

Starts a write job asynchronously. The actual job is performed by Fls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_Writechanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_pJobDataSrcPtr,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Write(Fls_AddressType u32TargetAddress, const uint8 *pSourceAddressPtr, Fls_LengthType u32Length);`

Table 3-23. Fls_Write Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32TargetAddress	**input**	Target address in flash memory.
const uint8 *	pSourceAddressPtr	**input**	Pointer to source data buffer.
`Fls_LengthType`	u32Length	**input**	Number of bytes to write.

Table 3-24. Fls_Write Return Values

Name	Description
E_OK	Write command has been accepted.
E_NOT_OK	Write command has not been accepted.

3.9.3.12 Function Fls_BlankCheck

Verifies if a given memory area is erased and not yet programmed. The API is intended to be used on platforms on which reading an erased location is not supported or there is no fixed erased value.

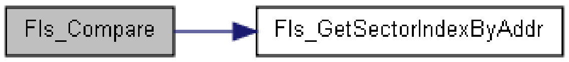


Figure 3-5. Function Fls_BlankCheck References.

Details:

Starts a compare job asynchronously. The actual job is performed byFls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_BlankCheckchanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_BlankCheck(Fls_AddressType u32TargetAddress, Fls_LengthType u32Length);`

Table 3-25. Fls_BlankCheck Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32TargetAddress	**input**	Address in flash memory from which the blank check should be started.
`Fls_LengthType`	u32Length	**input**	Number of bytes to compare.

Table 3-26. Fls_BlankCheck Return Values

Name	Description
E_OK	Blank check command has been accepted.
E_NOT_OK	Blank check command has not been accepted.

3.9.3.13 Function Fls_QspiCheckControllerIdle

External API usable in driver callout functions.

Details:

Checks the busy status of the internal QuadSPI controller.

Return: Std_ReturnType.

Prototype: `Std_ReturnType Fls_QspiCheckControllerIdle(Fls_QspiUnitNameType eHwUnitName);`

Table 3-27. Fls_QspiCheckControllerIdle Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.

Table 3-28. Fls_QspiCheckControllerIdle Return Values

Name	Description
E_OK	Controller is idle.
E_NOT_OK	Controller is busy.

3.9.3.14 Function Fls_QspiDisableModule

External API usable in driver callout functions.

Details:

Disables the internal QuadSPI controller in order to allow parameter modifications(ex: calibration delay values).

Return: void.

Prototype: `void Fls_QspiDisableModule(Fls_QspiUnitNameType eHwUnitName);`

Table 3-29. Fls_QspiDisableModule Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.

3.9.3.15 Function Fls_QspiEnableModule

External API usable in driver callout functions.

Details:

Enables the internal QuadSPI controller after parameter modifications(ex: calibration delay values).

Return: void.

Prototype: `void Fls_QspiEnableModule(Fls_QspiUnitNameType eHwUnitName);`

Table 3-30. Fls_QspiEnableModule Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.

3.9.3.16 Function Fls_QspiGetHwUnitBaseAddr

External API usable in driver callout functions.

Details:

Returns the base address for the QSPI hardware IP(register access) corresponding to the specified channel.

Return: Fls_AddressType.

Prototype: `Fls_AddressType Fls_QspiGetHwUnitBaseAddr(Fls_QspiUnitNameType eHwUnitName);`

Table 3-31. Fls_QspiGetHwUnitBaseAddr Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.

3.9.3.17 Function Fls_QspiLaunchLUTNumber

External API usable in driver callout functions.

Details:

Launches the external QuadSPI memory transaction.

Return: Std_ReturnType.

Prototype: `Std_ReturnType Fls_QspiLaunchLUTNumber(Fls_QspiUnitNameType eHwUnitName, uint8 u8LUTNumber, uint32 u32DataSize, uint8 u8ParEn);`

Table 3-32. Fls_QspiLaunchLUTNumber Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.
`uint8`	u8LUTNumber	**input**	Number of the LUT sequence used in the command. Available LUT numbers are [0-63] but only DIV4 LUT numbers are valid for launching a LUT command sequence (LUT0, LUT4...LUT60).
`uint32`	u32DataSize	**input**	Data size(in bytes) of the external transaction(used for read/write commands). When this value is not zero, it overrides the transaction length set in the LUT command operands.
`uint8`	u8ParEn	**input**	Parity enabled. if enabled, the external transaction is sent to both external parallel channels.

Table 3-33. Fls_QspiLaunchLUTNumber Return Values

Name	Description
E_OK	Controller is idle.
E_NOT_OK	Controller is busy.

3.9.3.18 Function Fls_QspiLoadTXBuffer

External API usable in driver callout functions.

Details:

Loads the QuadSPI TX buffer with the data to be transfered.

Return: void.

Pre: The module has to be initialized and not busy.

Post: Fls_QspiLoadTXBufferchanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobSectorEnd,Fls_eJob,Fls_eJobResult).

Prototype: `void Fls_QspiLoadTXBuffer(Fls_QspiUnitNameType eHwUnitName, *uint8 p8DataPtr, uint8 u8BytesToWrite);`

Table 3-34. Fls_QspiLoadTXBuffer Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.
`*uint8`	p8DataPtr	**input**	Pointer to application source buffer.
`uint8`	u8BytesToWrite	**input**	Number of bytes written in current transaction.

3.9.3.19 Function Fls_QspiReadRXBuffer

External API usable in driver callout functions.

Details:

Reads data from QuadSPI RX buffer into the application buffer

Return: void.

Pre: The module has to be initialized and not busy.

Post: Fls_QspiReadRXBufferchanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobSectorEnd,Fls_eJob,Fls_eJobResult).

Prototype: `void Fls_QspiReadRXBuffer(Fls_QspiUnitNameType eHwUnitName, *uint8 p8DataPtr, uint8 u8BytesToWrite);`

Table 3-35. Fls_QspiReadRXBuffer Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.
`*uint8`	p8DataPtr	**input**	Pointer to application destination buffer.
`uint8`	u8BytesToWrite	**input**	Number of bytes written in current transaction.

Table 3-36. Fls_QspiReadRXBuffer Return Values

Name	Description
E_OK	Read data successfully.

Table continues on the next page...

Table 3-36. Fls_QspiReadRXBuffer Return Values (continued)

Name	Description
E_NOT_OK	Error occured during data read.

3.9.3.20 Function Fls_QspiResetFlags

External API usable in driver callout functions.

Details:

Resets all QuadSPI error flags.

Return: void.

Prototype: `void Fls_QspiResetFlags(Fls_QspiUnitNameType eHwUnitName);`

Table 3-37. Fls_QspiResetFlags Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.

3.9.3.21 Function Fls_QspiSetCalibDelayValues

External API usable in driver callout functions.

Details:

Sets the calibration timing parameters inside QuadSPI IP. The module has to be disabled prior to modifying these parameters.

Return: void.

Prototype: `void Fls_QspiSetCalibDelayValues(Fls_QspiUnitNameType eHwUnitName, uint8 u8CoarseDelay, uint8 u8FineDelay);`

Table 3-38. Fls_QspiSetCalibDelayValues Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.

Table continues on the next page...

Table 3-38. Fls_QspiSetCalibDelayValues Arguments (continued)

Type	Name	Direction	Description
`uint8`	u8CoarseDelay	**input**	Calibration coarse delay. Consult platform specific details to check if this parameter is applicable or not.
`uint8`	u8FineDelay	**input**	Calibration fine delay. Consult platform specific details to check if this parameter is applicable or not.

3.9.3.22 Function Fls_QspiSetLUT

External API usable in driver callout functions.

Details:

Launches the external QuadSPI memory transaction.

Return: Std_ReturnType.

Prototype: `Std_ReturnType Fls_QspiSetLUT(Fls_QspiUnitNameType eHwUnitName, uint8 u8LUTNumber, uint32 LUTValue);`

Table 3-39. Fls_QspiSetLUT Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.
`uint8`	u8LUTNumber	**input**	Number of the LUT sequence used in the command. Available LUT numbers are [0-63] but only DIV4 LUT numbers are valid for launching a LUT command sequence (LUT0, LUT4...LUT60).
`uint32`	LUTValue	**input**	LUT value, two concatenated instruction operand pairs.

Table 3-40. Fls_QspiSetLUT Return Values

Name	Description
E_OK	LUT setting successful.
E_NOT_OK	Error during LUT setting.

3.9.3.23 Function Fls_QspiSetRXWmrk

External API usable in driver callout functions.

Details:

Disables the internal QuadSPI controller in order to allow parameter modifications(ex: calibration delay values).

Return: void.

Prototype: `void Fls_QspiSetRXWmrk(Fls_QspiUnitNameType eHwUnitName, uint8 u8BytesToRead);`

Table 3-41. Fls_QspiSetRXWmrk Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.
`uint8`	u8BytesToRead	**input**	Number of bytes which are expected to be received.

3.9.3.24 Function Fls_QspiSetSfarAddr

External API usable in driver callout functions.

Details:

Sets the serial flash start address used for the next transaction.

Return: Std_ReturnType.

Prototype: `Std_ReturnType Fls_QspiSetSfarAddr(Fls_QspiUnitNameType eHwUnitName, Fls_AddressType u32Addr);`

Table 3-42. Fls_QspiSetSfarAddr Arguments

Type	Name	Direction	Description
`Fls_QspiUnitNameType`	eHwUnitName	**input**	Name of the QuadSPI IP controller.
`Fls_AddressType`	u32Addr	**input**	The serial flash address used for the external transaction. The address represents the external sector address offsetted on the QSPI channel chip specific memory address.

3.9.4 Structs Reference

Data structures supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.5 Types Reference

Types supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.10 Symbolic Names DISCLAIMER

All containers having the symbolic name tag set as true in the Autosar schema will generate defines like

#define <Container_Short_Name> <Container_ID>

For this reason it is forbidden to duplicate the name of such containers across the MCAL configuration, 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 FLS Driver. The most of the parameters are described below.

4.1 Configuration elements of Fls

Included forms :

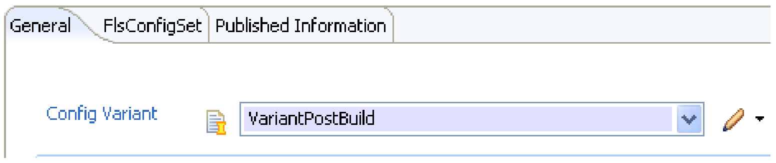
- IMPLEMENTATION_CONFIG_VARIANT
- NonAutosar
- FlsGeneral
- FlsPublishedInformation
- CommonPublishedInformation
- FlsConfigSet

4.2 Form IMPLEMENTATION_CONFIG_VARIANT

VariantPostBuild: Mix of precompile and postbuild time configuration parameters.

If Config Variant = VariantPostBuild, the files Fls_Cfg.h and Fls_PBcfg.c should be used.

If Config Variant = VariantPreCompile, the files Fls_Cfg.h and Fls_Cfg.c should be used.



Config Variant: VariantPostBuild

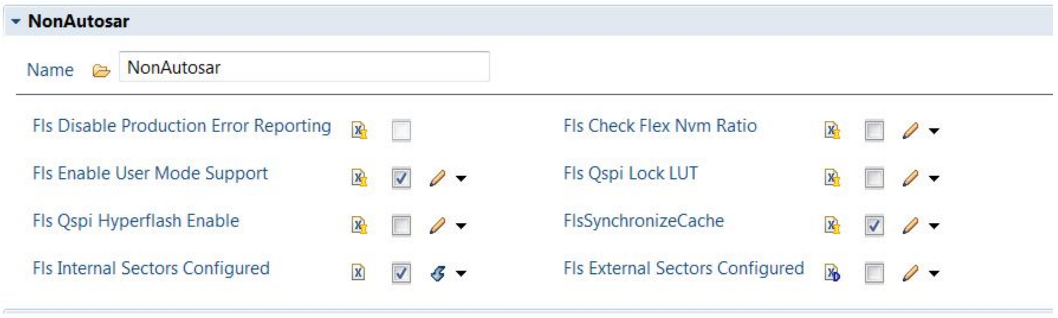
Figure 4-1. Tresos Plugin snapshot for IMPLEMENTATION_CONFIG_VARIANT form.

Table 4-1. Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description

Property	Value
Label	Config Variant
Default	VariantPostBuild
Range	VariantPostBuild VariantPreCompile

4.3 Form NonAutosar

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


NonAutosar

Name: NonAutosar

Fls Disable Production Error Reporting	☐	Fls Check Flex Nvm Ratio	☐
Fls Enable User Mode Support	✓	Fls Qspi Lock LUT	☐
Fls Qspi Hyperflash Enable	☐	FlsSynchronizeCache	✓
Fls Internal Sectors Configured	✓	Fls External Sectors Configured	☐

Figure 4-2. Tresos Plugin snapshot for NonAutosar form.

4.3.1 FlsEnableUserModeSupport (NonAutosar)

When this parameter is enabled, the FLS module will adapt to run from User Mode, with the following measure : configuring REG_PROT for Fls 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.

true: FLS module will adapt to run from User Mode

false: FLS module will adapt to run from User Mode

Table 4-2. Attribute FlsEnableUserModeSupport (NonAutosar) detailed description

Property	Value
Label	Fls Enable User Mode Support
Type	BOOLEAN

Table continues on the next page...

Table 4-2. Attribute FlsEnableUserModeSupport (NonAutosar) detailed description (continued)

Property	Value
Origin	NXP
Symbolic Name	false
Default	false

4.3.2 FlsCheckFlexNvmRatio (NonAutosar)

Vendor specific: Switches the FlexNvm partion ratio check ON at initialization.

If check is ON, the driver initialization will fail if FlexNvm is partitioned otherwise than: Data flash only, no EEPROM emulation.

true: FLS module will check the partitioning

false: FLS module will not check the partitioning

Table 4-3. Attribute FlsCheckFlexNvmRatio (NonAutosar) detailed description

Property	Value
Label	Fls Check Flex Nvm Ratio
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.3.3 FlsInternalSectorsConfigured (NonAutosar)

Vendor specific: Boolean parameter which must be enabled if internal flash sectors are configured.

true: At least one internal flash sector is configured in any variant.

false: No internal flash sector is configured in any variant, only external flash sectors are present.

Table 4-4. Attribute FlsInternalSectorsConfigured (NonAutosar) detailed description

Property	Value
Label	Fls Internal Sectors Configured

Table continues on the next page...

Table 4-4. Attribute FlsInternalSectorsConfigured (NonAutosar) detailed description (continued)

Property	Value
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.3.4 FlsExternalSectorsConfigured (NonAutosar)

Vendor specific: Boolean parameter which must be enabled if external flash sectors are configured.

true: At least one external flash sector is configured in any variant.

false: No external flash sector is configured in any variant, only internal flash sectors are present.

Table 4-5. Attribute FlsExternalSectorsConfigured (NonAutosar) detailed description

Property	Value
Label	Fls External Sectors Configured
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.3.5 FlsSynchronizeCache(NonAutosar)

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

The FLS 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 FlsSynchronizeCache feature.

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

Enabled: The FLS 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 MclLmemEnableCacheApi 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.

true: Fls Synchronize Cache enabled.

false: Fls Synchronize Cache disabled.

Table 4-6. Attribute FlsSynchronizeCache (NonAutosar) detailed description

Property	Value
Label	Fls Synchronize Cache
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.3.6 FlsQspiLockLUT (NonAutosar)

Enable the Lock/Unlock of the LUT for the external QuadSPI memory

true: The LUT is unlocked at the beginning of the Init phase and locked at the end of it

false: The LUT has to be unlocked if the Init phase is supposed to populate it.

Table 4-7. Attribute FlsQspiLockLUT (NonAutosar) detailed description

Property	Value
Label	Fls Qspi Lock LUT
Type	BOOLEAN
Origin	NXP

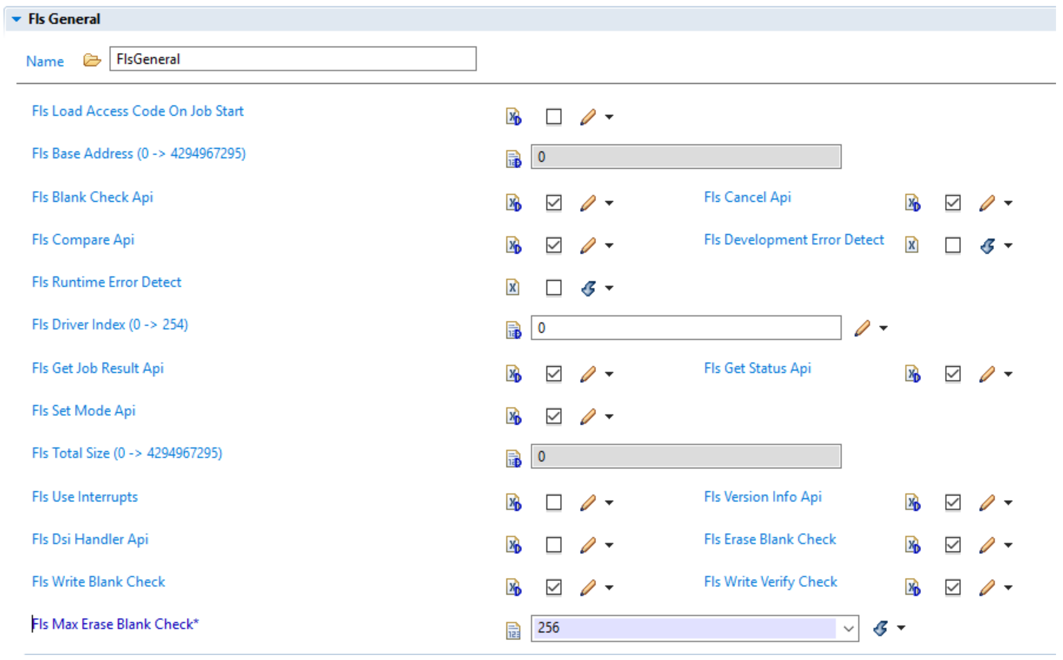
Table continues on the next page...

Table 4-7. Attribute FlsQspiLockLUT (NonAutosar) detailed description (continued)

Property	Value
Symbolic Name	false
Default	false

4.4 Form FlsGeneral

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



Fls General

Name: FlsGeneral

Fls Load Access Code On Job Start	☐		
Fls Base Address (0 -> 4294967295)	0		
Fls Blank Check Api	☑	Fls Cancel Api	☑
Fls Compare Api	☑	Fls Development Error Detect	☐
Fls Runtime Error Detect	☐		
Fls Driver Index (0 -> 254)	0		
Fls Get Job Result Api	☑	Fls Get Status Api	☑
Fls Set Mode Api	☑		
Fls Total Size (0 -> 4294967295)	0		
Fls Use Interrupts	☐	Fls Version Info Api	☑
Fls Dsi Handler Api	☐	Fls Erase Blank Check	☑
Fls Write Blank Check	☑	Fls Write Verify Check	☑
Fls Max Erase Blank Check*	256		

Figure 4-3. Tresos Plugin snapshot for FlsGeneral form.

4.4.1 FlsAcLoadOnJobStart (FlsGeneral)

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.

Table 4-8. Attribute FlsAcLoadOnJobStart (FlsGeneral) detailed description

Property	Value
Label	Fls Load Access Code On Job Start
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

4.4.2 FlsBaseAddress (FlsGeneral)

The flash memory start address.

This parameter defines the lower boundary for read / write / erase and compare jobs.

Note

Not needed / supported by the driver.

Table 4-9. Attribute FlsBaseAddress (FlsGeneral) detailed description

Property	Value
Label	Fls Base Address
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.4.3 FlsBlankCheckApi (FlsGeneral)

Compile switch to enable and disable the Fls_BlankCheck function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-10. Attribute FlsBlankCheckApi (FlsGeneral) detailed description

Property	Value
Label	Fls Blank Check Api

Table continues on the next page...

Table 4-10. Attribute FlsBlankCheckApi (FlsGeneral) detailed description (continued)

Property	Value
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.4 FlsCancelApi (FlsGeneral)

Compile switch to enable and disable the Fls_Cancel function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-11. Attribute FlsCancelApi (FlsGeneral) detailed description

Property	Value
Label	Fls Cancel Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.5 FlsCompareApi (FlsGeneral)

Compile switch to enable and disable the Fls_Compare function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-12. Attribute FlsCompareApi (FlsGeneral) detailed description

Property	Value
Label	Fls Compare Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.6 FlsDevErrorDetect (FlsGeneral)

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

true: Development error detection enabled.

false: Development error detection disabled.

Table 4-13. Attribute FlsDevErrorDetect (FlsGeneral) detailed description

Property	Value
Label	Fls Development Error Detect
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.7 FlsRuntimeErrorDetect (FlsGeneral)

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

true: Runtime error detection enabled.

false: Runtime error detection disabled.

Table 4-14. Attribute FlsRuntimeErrorDetect (FlsGeneral) detailed description

Property	Value
Label	Fls Runtime Error Detect
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.8 FlsDriverIndex (FlsGeneral)

Index of the driver, used by FEE.

Table 4-15. Attribute FlsDriverIndex (FlsGeneral) detailed description

Property	Value
Label	Fls Driver Index
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	true
Default	0
Invalid	Range <=254 >=0

4.4.9 FlsGetJobResultApi (FlsGeneral)

Compile switch to enable and disable the Fls_GetJobResult function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-16. Attribute FlsGetJobResultApi (FlsGeneral) detailed description

Property	Value
Label	Fls Get Job Result Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.10 FlsGetStatusApi (FlsGeneral)

Compile switch to enable and disable the Fls_GetStatus function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-17. Attribute FlsGetStatusApi (FlsGeneral) detailed description

Property	Value
Label	Fls Get Status Api
Type	BOOLEAN

Table continues on the next page...

Table 4-17. Attribute FlsGetStatusApi (FlsGeneral) detailed description (continued)

Property	Value
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.11 FlsSetModeApi (FlsGeneral)

Compile switch to enable and disable the Fls_SetMode function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-18. Attribute FlsSetModeApi (FlsGeneral) detailed description

Property	Value
Label	Fls Set Mode Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.12 FlsTotalSize (FlsGeneral)

The total amount of flash memory in bytes. This parameter in conjunction with FLS_BASE_ADDRESS defines the upper boundary for read / write / erase and compare jobs.

Note

Not needed / supported by the driver.

Table 4-19. Attribute FlsTotalSize (FlsGeneral) detailed description

Property	Value
Label	Fls Total Size
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0

Table continues on the next page...

Table 4-19. Attribute FlsTotalSize (FlsGeneral) detailed description (continued)

Property	Value
Invalid	Range <=4294967295 >=0

4.4.13 FlsUseInterrupts (FlsGeneral)

Job processing triggered by hardware interrupt.

true: Job processing triggered by interrupt (hardware controlled)

false: Job processing not triggered by interrupt (software controlled)

Note

Parameter is supported if there is hardware support on current platform.

Table 4-20. Attribute FlsUseInterrupts (FlsGeneral) detailed description

Property	Value
Label	Fls Use Interrupts
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

4.4.14 FlsVersionInfoApi (FlsGeneral)

Pre-processor switch to enable / disable the API to read out the modules version information.

true: Version info API enabled.

false: Version info API disabled.

Table 4-21. Attribute FlsVersionInfoApi (FlsGeneral) detailed description

Property	Value
Label	Fls Version Info Api
Type	BOOLEAN

Table continues on the next page...

Table 4-21. Attribute FlsVersionInfoApi (FlsGeneral) detailed description (continued)

Property	Value
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.15 FlsDsiHandlerApi (FlsGeneral)

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

true: Data storage handler API enabled.

false: Data storage handler API disabled.

Note

1. The FlsDsiHandlerApi handler is applicable to ECC errors triggered on internal flash only. If external flash is present, the FlsQspiCheckECCCallout is provided to the application for ECC management.

2. The FLS driver is responsible for the FlsDsiHandler implementation and for marking the current job as failed if the ECC error was correctly acknoleadged as being generated by a FLS driver access, based on the input provided by the application error handler.

3. The customer application should use Fls_DsiHandler if the customer exception handling strategy is to allow the memory stack to handle data flash ECC errors.

4. The customer is responsible for the exception handler implementation based on its project specific strategy and can choose the appropriate recovery strategy based on the platform setting conditions.

Table 4-22. Attribute FlsDsiHandlerApi (FlsGeneral) detailed description

Property	Value
Label	Fls Dsi Handler Api
Type	BOOLEAN

Table continues on the next page...

Table 4-22. Attribute FlsDsiHandlerApi (FlsGeneral) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	false

4.4.16 FlsEraseBlankCheck (FlsGeneral)

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: Erase blank check enabled.

false: Erase blank check disabled.

Note

Vendor specific parameter

Table 4-23. Attribute FlsEraseBlankCheck (FlsGeneral) detailed description

Property	Value
Label	Fls Erase Blank Check
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.17 FlsWriteBlankCheck (FlsGeneral)

Pre-processor switch to enable / disable the write blank check. Before writing a flash block, the write 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: Write blank check enabled.

false: Write blank check disabled.

Note

Vendor specific parameter

Table 4-24. Attribute FlsWriteBlankCheck (FlsGeneral) detailed description

Property	Value
Label	Fls Write Blank Check
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.18 FlsWriteVerifyCheck (FlsGeneral)

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: Write verify check enabled.

false: Write verify check disabled.

Note

Vendor specific parameter

Table 4-25. Attribute FlsWriteVerifyCheck (FlsGeneral) detailed description

Property	Value
Label	Fls Write Verify Check
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.19 FlsMaxEraseBlankCheck (FlsGeneral)

The maximum number of bytes to blank check in one cycle of the flash driver's job processing function. Affects only the flash blocks that have enabled asynchronous execution of the erase job (FlsSectorEraseAsynch=true).

Note

Vendor specific parameter

Table 4-26. Attribute FlsMaxEraseBlankCheck (FlsGeneral) detailed description

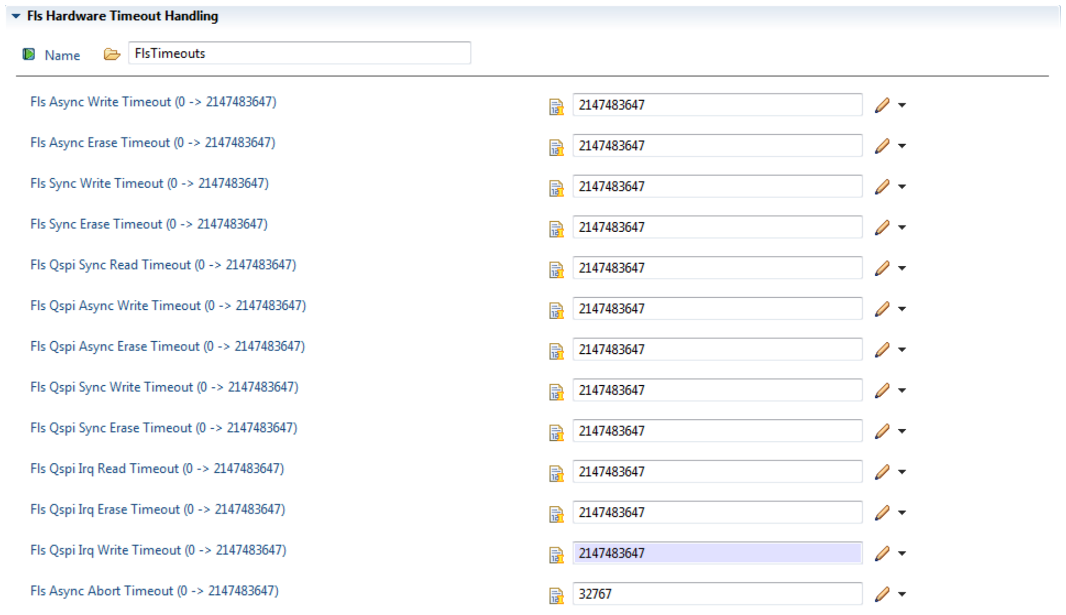
Property	Value
Label	Fls Max Erase Blank Check
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	256
Invalid	Range <=65536 >=8

4.5 Form FlsTimeouts

Container for timeout parameters of the flash driver. The implemented timeout check functionality provides monitoring and deadline supervision of the currently running HW job.

NOTE

Disabling/deleting the container suppresses compilation of the code providing the timeout check functionality.



Name	FlsTimeouts
Fls Async Write Timeout (0 -> 2147483647)	2147483647
Fls Async Erase Timeout (0 -> 2147483647)	2147483647
Fls Sync Write Timeout (0 -> 2147483647)	2147483647
Fls Sync Erase Timeout (0 -> 2147483647)	2147483647
Fls Qspi Sync Read Timeout (0 -> 2147483647)	2147483647
Fls Qspi Async Write Timeout (0 -> 2147483647)	2147483647
Fls Qspi Async Erase Timeout (0 -> 2147483647)	2147483647
Fls Qspi Sync Write Timeout (0 -> 2147483647)	2147483647
Fls Qspi Sync Erase Timeout (0 -> 2147483647)	2147483647
Fls Qspi Irq Read Timeout (0 -> 2147483647)	2147483647
Fls Qspi Irq Erase Timeout (0 -> 2147483647)	2147483647
Fls Qspi Irq Write Timeout (0 -> 2147483647)	2147483647
Fls Async Abort Timeout (0 -> 2147483647)	32767

Figure 4-4. Tresos Plugin snapshot for FlsTimeouts form.

4.5.1 FlsAsyncWriteTimeout (FlsTimeouts)

The timeout value applied for the HW write operations handled in asynchronous mode. Each time the Fls_MainFunction() starts the flash program sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each of the next Fls_MainFunction() calls. Once it reaches 0, the ongoing program operation is aborted.

Table 4-27. Attribute FlsAsyncWriteTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Async Write Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.2 FlsAsyncEraseTimeout (FlsTimeouts)

The timeout value applied for the HW erase operations handled in asynchronous mode. Each time the Fls_MainFunction() starts the flash erase sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each of the next Fls_MainFunction() calls. Once it reaches 0, the ongoing erase operation is aborted.

Table 4-28. Attribute FlsAsyncEraseTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Async Erase Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.3 FlsSyncWriteTimeout (FlsTimeouts)

The timeout value applied for the HW write operations handled in synchronous mode. Inside Fls_MainFunction() a SW loop is implemented waiting for the ongoing program operation to finish. Each time the Fls_MainFunction() starts the flash program sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each execution of the SW loop. Once it reaches 0, the ongoing program operation is aborted.

One SW loop execution takes 10 or 12 machine instructions depending on whether FlsACCallback is a NULL_PTR or not, respectively, plus additional instructions to execute FlsACCallback (if not a NULL_PTR).

Table 4-29. Attribute FlsSyncWriteTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Sync Write Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.4 FlsSyncEraseTimeout (FlsTimeouts)

The timeout value applied for the HW erase operations handled in synchronous mode. Inside Fls_MainFunction() a SW loop is implemented waiting for the ongoing erase operation to finish. Each time the Fls_MainFunction() starts the flash erase sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each execution of the SW loop. Once it reaches 0, the ongoing erase operation is aborted.

One SW loop execution takes 10 or 12 machine instructions depending on whether FlsACCallback is a NULL_PTR or not, respectively, plus additional instructions to execute FlsACCallback (if not a NULL_PTR).

Table 4-30. Attribute FlsSyncEraseTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Sync Erase Timeout

Table continues on the next page...

Table 4-30. Attribute FlsSyncEraseTimeout (FlsTimeouts) detailed description (continued)

Property	Value
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.5 FlsQspiSyncReadTimeout (FlsTimeouts)

Fls Qspi Sync Read Timeout is the timeout value for read operation.

Table 4-31. Attribute FlsQspiSyncReadTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Sync Read Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.6 FlsQspiAsyncWriteTimeout (FlsTimeouts)

Fls Qspi Async Write Timeout is the timeout value for QSPI write operation in asynchronous mode.

Table 4-32. Attribute FlsQspiAsyncWriteTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Async Write Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.7 FlsQspiAsyncEraseTimeout (FlsTimeouts)

Fls Qspi Async Erase Timeout is the timeout value for QSPI erase operation in asynchronous mode.

Table 4-33. Attribute FlsQspiAsyncEraseTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Async Erase Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.8 FlsQspiSyncWriteTimeout (FlsTimeouts)

Fls Qspi Sync Write Timeout is the timeout value for QSPI write operation in synchronous mode

Table 4-34. Attribute FlsQspiSyncWriteTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Sync Write Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.9 FlsQspiSyncEraseTimeout (FlsTimeouts)

Fls Qspi Sync Erase Timeout is the timeout value for QSPI erase operation in synchronous mode.

Table 4-35. Attribute FlsQspiSyncEraseTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Sync Erase Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.10 FlsQspiIrqReadTimeout (FlsTimeouts)

Fls Qspi Irq Read Timeout is the timeout value for QSPI read operation in interrupt mode. The value represents the number of Fls_MainFunctions it takes to complete the QSPI interrupt job.

Table 4-36. Attribute FlsQspiIrqReadTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Irq Read Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.11 FlsQspiIrqEraseTimeout (FlsTimeouts)

Fls Qspi Irq Erase Timeout is the timeout value for QSPI read operation in interrupt mode. The value represents the number of Fls_MainFunctions it takes to complete the QSPI interrupt job.

Table 4-37. Attribute FlsQspiIrqEraseTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Irq Erase Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.12 FlsQspiIrqWriteTimeout (FlsTimeouts)

Fls Qspi Irq Write Timeout is the timeout value for QSPI read operation in interrupt mode. The value represents the number of Fls_MainFunctions it takes to complete the QSPI interrupt job.

Table 4-38. Attribute FlsQspiIrqWriteTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Qspi Irq Write Timeout
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.13 FlsAbortTimeout (FlsTimeouts)

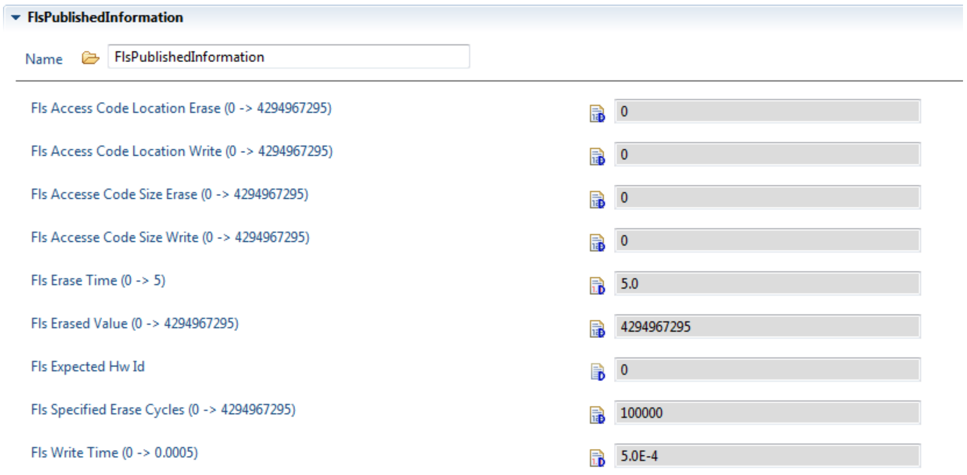
The timeout value applied for the abort of the HW write or erase operation. Inside Fls_MainFunction() a SW loop is implemented waiting for the ongoing abort operation to finish. When the Fls_MainFunction() starts the abort, its internal timeout counter is initialized with the configured value. The counter is decremented per each execution of the SW loop. Once it reaches 0, the SW loop is escaped.

Table 4-39. Attribute FlsAbortTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Async Abort Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	32767
INVALID	Range <=2147483647 >=0

4.6 Form FlsPublishedInformation

Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.



FlsPublishedInformation

Name	FlsPublishedInformation
Fls Access Code Location Erase (0 -> 4294967295)	0
Fls Access Code Location Write (0 -> 4294967295)	0
Fls Accesse Code Size Erase (0 -> 4294967295)	0
Fls Accesse Code Size Write (0 -> 4294967295)	0
Fls Erase Time (0 -> 5)	5.0
Fls Erased Value (0 -> 4294967295)	4294967295
Fls Expected Hw Id	0
Fls Specified Erase Cycles (0 -> 4294967295)	100000
Fls Write Time (0 -> 0.0005)	5.0E-4

Figure 4-5. Tresos Plugin snapshot for FlsPublishedInformation form.

4.6.1 FlsAcLocationErase (FlsPublishedInformation)

Position in RAM, to which the erase flash access code has to be loaded. Only relevant if the erase flash access code is not position independent. If this information is not provided (or the value is zero) it is assumed that the erase flash access code is position independent and therefore the RAM position can be freely configured.

Note: This parameter is applicable only for internal flash.

Table 4-40. Attribute FlsAcLocationErase (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Location Erase
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.6.2 FlsAcLocationWrite (FlsPublishedInformation)

Position in RAM, to which the write flash access code has to be loaded. Only relevant if the write flash access code is not position independent. If this information is not provided (or the value is zero) it is assumed that the write flash access code is position independent and therefore the RAM position can be freely configured.

Note: This parameter is applicable only for internal flash.

Table 4-41. Attribute FlsAcLocationWrite (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Location Write
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.6.3 FlsAcSizeErase (FlsPublishedInformation)

Number of bytes in RAM needed for the erase flash access code. If this information is not provided (or the value is zero) it is assumed that the access code is delivered not as a precompiled but C source code and its size is not known before linking. In such a case a size of the RAM memory reserved for the access code needs to determined dynamically by the linker (see the Integration Manual for more information).

Note: This parameter is applicable only for internal flash.

Table 4-42. Attribute FlsAcSizeErase (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Size Erase
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.6.4 FlsAcSizeWrite (FlsPublishedInformation)

Number of bytes in RAM needed for the write flash access code. If this information is not provided (or the value is zero) it is assumed that the access code is delivered not as a precompiled but C source code and its size is not known before linking. In such a case a size of the RAM memory reserved for the access code needs to determined dynamically by the linker (see the Integration Manual for more information).

Note: This parameter is applicable only for internal flash.

Table 4-43. Attribute FlsAcSizeWrite (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Size Write
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false

Table continues on the next page...

Table 4-43. Attribute FlsAcSizeWrite (FlsPublishedInformation) detailed description (continued)

Property	Value
Default	0
Invalid	Range <=4294967295 >=0

4.6.5 FlsEraseTime (FlsPublishedInformation)

Maximum time to erase one complete flash sector [sec].

Note

This value can be found on DS as the maximum erase time occurs after the specified number of program/erase cycles .

Table 4-44. Attribute FlsEraseTime (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Erase Time
Type	FLOAT_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	5.0
Invalid	Range <=5.0 >=0

4.6.6 FlsErasedValue (FlsPublishedInformation)

The contents of an erased flash memory cell.

Table 4-45. Attribute FlsErasedValue (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Erased Value
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	4294967295
Invalid	Range

Table 4-45. Attribute FlsErasedValue (FlsPublishedInformation) detailed description

Property	Value
	<=4294967295 >=0

4.6.7 FlsExpectedHwId (FlsPublishedInformation)

Unique identifier of the hardware device that is expected by this driver (the device for which this driver has been implemented). Only relevant for external flash drivers.

Table 4-46. Attribute FlsExpectedHwId (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Expected Hw Id
Type	STRING_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0

4.6.8 FlsSpecifiedEraseCycles (FlsPublishedInformation)

Number of erase cycles specified for the flash device (usually given in the device data sheet). If the number of specified erase cycles depends on the operating environment (temperature, voltage, ...) during reprogramming of the flash device, the minimum number for which a data retention of at least 15 years over the temperature range from -40C .. +125C can be guaranteed shall be given.

Note

If there are different numbers of specified erase cycles for different flash sectors of the device this parameter has to be extended to a parameter list (similar to the sector list above).

Table 4-47. Attribute FlsSpecifiedEraseCycles (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Specified Erase Cycles
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false

Table continues on the next page...

Table 4-47. Attribute FlsSpecifiedEraseCycles (FlsPublishedInformation) detailed description (continued)

Property	Value
Default	100000
Invalid	Range <=4294967295 >=0

4.6.9 FlsWriteTime (FlsPublishedInformation)

Maximum time to program one complete flash page [sec].

Table 4-48. Attribute FlsWriteTime (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Write Time
Type	FLOAT_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0.0005
Invalid	Range <=0.0005 >=0

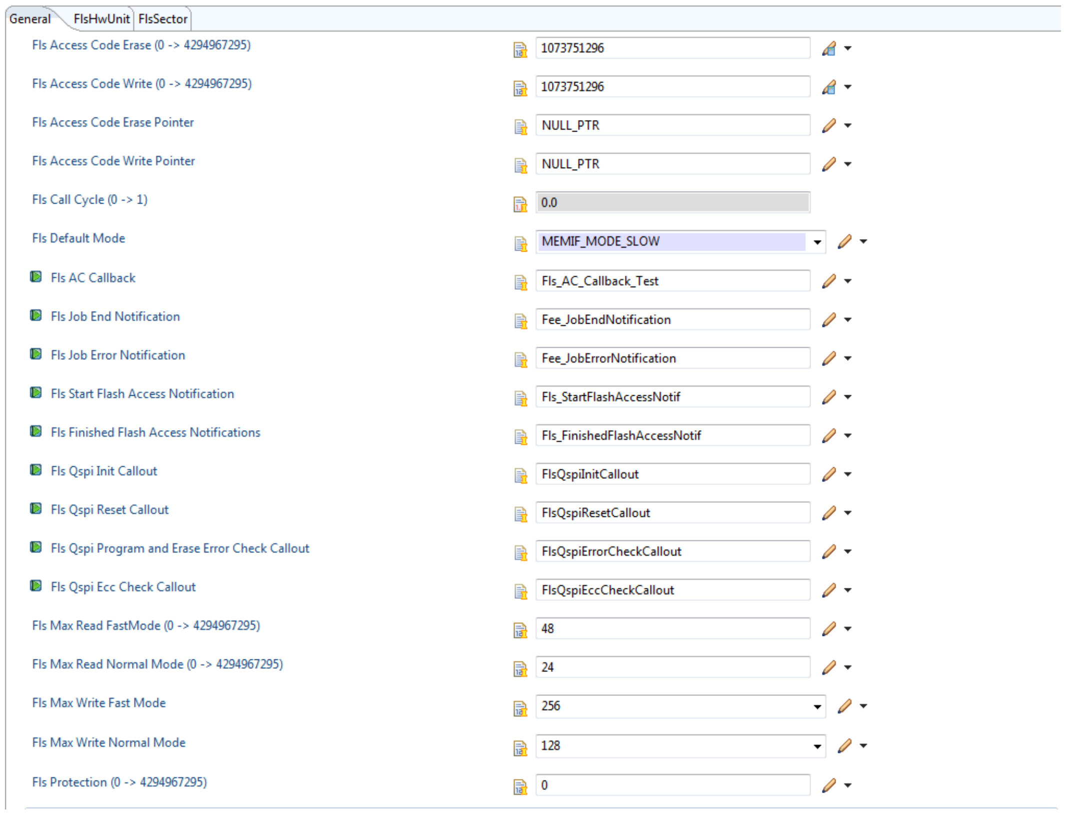
4.7 Form FlsConfigSet

Container for runtime configuration parameters of the flash driver.

Implementation Type: Fls_ConfigType.

Included forms :

- Form FlsSectorList


General | FlsHwUnit | FlsSector

Fls Access Code Erase (0 -> 4294967295)	1073751296
Fls Access Code Write (0 -> 4294967295)	1073751296
Fls Access Code Erase Pointer	NULL_PTR
Fls Access Code Write Pointer	NULL_PTR
Fls Call Cycle (0 -> 1)	0.0
Fls Default Mode	MEMIF_MODE_SLOW
Fls AC Callback	Fls_AC_Callback_Test
Fls Job End Notification	Fee_JobEndNotification
Fls Job Error Notification	Fee_JobErrorNotification
Fls Start Flash Access Notification	Fls_StartFlashAccessNotif
Fls Finished Flash Access Notifications	Fls_FinishedFlashAccessNotif
Fls Qspi Init Callout	FlsQspiInitCallout
Fls Qspi Reset Callout	FlsQspiResetCallout
Fls Qspi Program and Erase Error Check Callout	FlsQspiErrorCheckCallout
Fls Qspi Ecc Check Callout	FlsQspiEccCheckCallout
Fls Max Read FastMode (0 -> 4294967295)	48
Fls Max Read Normal Mode (0 -> 4294967295)	24
Fls Max Write Fast Mode	256
Fls Max Write Normal Mode	128
Fls Protection (0 -> 4294967295)	0

Figure 4-6. Tresos Plugin snapshot for FlsConfigSet form.

4.7.1 FlsAcErase (FlsConfigSet)

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 Fls Access Code Erase be sure Fls Access Code Erase Pointer is NULL or NULL_PTR.

Table 4-49. Attribute FlsAcErase (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Erase
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.7.2 FlsAcWrite (FlsConfigSet)

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 Fls Access Code Write be sure Fls Access Code Write Pointer is NULL or NULL_PTR.

Table 4-50. Attribute FlsAcWrite (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Write
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.7.3 FlsAcErasePointer (FlsConfigSet)

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

Table 4-51. Attribute FlsAcErasePointer (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Erase Pointer
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.7.4 FlsAcWritePointer (FlsConfigSet)

Pointer in RAM to which the write flash access code shall be loaded. Used as function pointer to access the write flash access code.

Table 4-52. Attribute FlsAcWritePointer (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Write Pointer

Table continues on the next page...

Table 4-52. Attribute FlsAcWritePointer (FlsConfigSet) detailed description (continued)

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.7.5 FlsCallCycle (FlsConfigSet)

Cycle time of calls of the flash driver main function

Note

Not supported by the driver.

Table 4-53. Attribute FlsCallCycle (FlsConfigSet) detailed description

Property	Value
Label	Fls Call Cycle
Type	FLOAT_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0.0
Invalid	Range <=1.0 >=0.0

4.7.6 FlsDefaultMode (FlsConfigSet)

This parameter is the default FLS device mode after initialization.

Table 4-54. Attribute FlsDefaultMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Default Mode
Type	ENUMERATION
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	MEMIF_MODE_SLOW
Range	MEMIF_MODE_FAST MEMIF_MODE_SLOW

4.7.7 FlsACCallback (FlsConfigSet)

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

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-55. Attribute FlsACCallback (FlsConfigSet) detailed description

Property	Value
Label	Fls AC Callback
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	Fls_AC_Callback
Enable	false

4.7.8 FlsJobEndNotification (FlsConfigSet)

Mapped to the job end notification routine provided by some upper layer module, typically the Fee module.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-56. Attribute FlsJobEndNotification (FlsConfigSet) detailed description

Property	Value
Label	Fls Job End Notification
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	Fee_JobEndNotification
Enable	false

4.7.9 FlsJobErrorNotification (FlsConfigSet)

Mapped to the job error notification routine provided by some upper layer module, typically the Fee module.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-57. Attribute FlsJobErrorNotification (FlsConfigSet) detailed description

Property	Value
Label	Fls Job Error Notification
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	Fee_JobErrorNotification
Enable	false

4.7.10 FlsStartFlashAccessNotif (FlsConfigSet)

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 FlsFinishedFlashAccess, 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 also Integration manual, chapter 5. Module requirements.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-58. Attribute FlsStartFlashAccessNotif (FlsConfigSet) detailed description

Property	Value
Label	Fls Start Flash Access Notification
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	Fls_StartFlashAccessNotif
Enable	false

4.7.11 FlsFinishedFlashAccessNotif (FlsConfigSet)

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) and after flash memory program operations(in write and erase jobs). The purpose of this notification together with FlsStartFlashAccess, 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 also Integration manual, chapter 5. Module requirements.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-59. Attribute FlsFinishedFlashAccessNotif (FlsConfigSet) detailed description

Property	Value
Label	Fls Finished Flash Access Notification
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	Fls_FinishedFlashAccessNotif
Enable	false

4.7.12 FlsQspiInitCallout (FlsConfigSet)

Callout function called by the driver at the end of the QSPI Init phase. The intended purpose of this callout is to provide to the application the possibility of performing additional configuration to the QSPI hardware IP or to the external memories connected(for ex: sending the lock/unlock sequences for the external flash sectors, altering QSPI IP timing, etc.) Note: Disable the callout in order to have it set as NULL_PTR. Note: The callout can be configured only if the FlsExternalDriver is enabled.

Table 4-60. Attribute FlsQspiInitCallout (FlsConfigSet) detailed description

Property	Value
Label	Fls Qspi Init Callout
Type	FUNCTION-NAME
Origin	NXP

Table continues on the next page...

Table 4-60. Attribute FlsQspiInitCallout (FlsConfigSet) detailed description (continued)

Property	Value
Symbolic Name	false
Default	FlsQspiInitCallout
Enable	false

4.7.13 FlsQspiResetCallout (FlsConfigSet)

Callout function called by the driver at the beginning of a new job. The intended purpose of this callout is to provide to the application the possibility of reseting the external memory to an idle and error free state. If the callout is disabled, at the beginnig of a new job the Fls_MainFunction will check the external memory status and if not, poll and wait for it to become idle. If the callout is enabled and the memory is not idle, the Fls_MainFunction will also call the configured function to allow the application to send extra commands to the external memory(software reset, abort any suspended operation, error flags clearing, etc.) Note: Disable the callout in order to have it set as NULL_PTR. Note: The callout can be configured only if the FlsExternalDriver is enabled.

Table 4-61. Attribute FlsQspiResetCallout (FlsConfigSet) detailed description

Property	Value
Label	Fls Qspi Reset Callout
Type	FUNCTION-NAME
Origin	NXP
Symbolic Name	false
Default	FlsQspiResetCallout
Enable	false

4.7.14 FlsQspiErrorCheckCallout (FlsConfigSet)

Callout function called by the driver at the end of each program and erase job The intended purpose of this callout is to provide to the application the possibility of interrogating the error status of the memory after each program and erase job. The application should check any error or status bits available and reset the memory after interrogation in case an error condition was detected. If the callout is enabled, at the end of each job, the callout is called and the return value is checked to determine if there was any error during the memory operation. Return values: E_OK(0) E_NOT_OK(1). If E_OK(0) is received, the job is considered successful. If E_NOT_OK(1) is received, the job is considered unsuccessful and marked as failed. If the callout is disabled, the job is assumed as successful from the memory status point of view. Note: Disable the callout in order to have it set as NULL_PTR. Note: The callout can be configured only if the FlsExternalDriver is enabled.

Table 4-62. Attribute FlsQspiErrorCheckCallout (FlsConfigSet) detailed description

Property	Value
Label	Fls Qspi Program and Erase Error Check Callout
Type	FUNCTION-NAME
Origin	NXP
Symbolic Name	false
Default	FlsQspiErrorCheckCallout
Enable	false

4.7.15 FlsQspiEccCheckCallout (FlsConfigSet)

Callout function called by the driver at the end of each read operation The intended purpose of this callout is to provide to the application the possibility of interrogating the ECC status of the memory after each read operation. The callout provide the hardware channel, start address and the size of the read operation. The application should check there is any ECC error in the current read data If the callout is enabled, at the end of each read operation, the callout is called and the return value is checked to determine if there was any error during the memory operation. Return values: E_OK(0) E_NOT_OK(1). If E_OK(0) is received, the job is considered successful. If E_NOT_OK(1) is received, the job is considered unsuccessful and marked as failed. If the callout is disabled, the job is assumed as successful from the memory status point of view. Note: Disable the callout in order to have it set as NULL_PTR. Note: The callout can be configured only if the FlsExternalDriver is enabled.

Table 4-63. Attribute FlsQspiEccCheckCallout (FlsConfigSet) detailed description

Property	Value
Label	Fls Qspi Ecc Check Callout
Type	FUNCTION-NAME
Origin	NXP
Symbolic Name	false
Default	FlsQspiEccCheckCallout
Enable	false

4.7.16 FlsMaxReadFastMode (FlsConfigSet)

The maximum number of bytes to read or compare in one cycle of the flash driver's job processing function in fast mode.

Note: If external sectors are configured and if FlsHwUnitWordAddressable is set, the FlsMaxReadFastMode must be an even value(two bytes aligned).

Table 4-64. Attribute FlsMaxReadFastMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Read FastMode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	1048576
Invalid	Range <=4294967295 >=0

4.7.17 FlsMaxReadNormalMode (FlsConfigSet)

The maximum number of bytes to read or compare in one cycle of the flash driver's job processing function in normal mode.

Note: If external sectors are configured and if FlsHwUnitWordAddressable is set, the FlsMaxReadNormalMode must be an even value(two bytes aligned).

Table 4-65. Attribute FlsMaxReadNormalMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Read Normal Mode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	1024
Invalid	Range <=4294967295 >=0

4.7.18 FlsMaxWriteFastMode (FlsConfigSet)

The maximum number of bytes to write in one cycle of the flash driver's job processing function in fast mode.

Note

If external QuadSPI sectors are configured, the parameter must be configured to a value an integer multiple of the FlsPageSize.

Table 4-66. Attribute FlsMaxWriteFastMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Write Fast Mode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	256
Invalid	Range <=4294967295 >=0

4.7.19 FlsMaxWriteNormalMode (FlsConfigSet)

The maximum number of bytes to write in one cycle of the flash driver's job processing function in normal mode.

Note

If external QuadSPI sectors are configured, the parameter must be configured to a value an integer multiple of the FlsPageSize.

Table 4-67. Attribute FlsMaxWriteNormalMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Write Normal Mode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	8
Invalid	Range <=4294967295 >=0

4.7.20 FlsProtection (FlsConfigSet)

Erase/write protection settings. Only relevant if supported by hardware.

Note

Not supported by the driver.

Table 4-68. Attribute FlsProtection (FlsConfigSet) detailed description

Property	Value
Label	Fls Protection
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

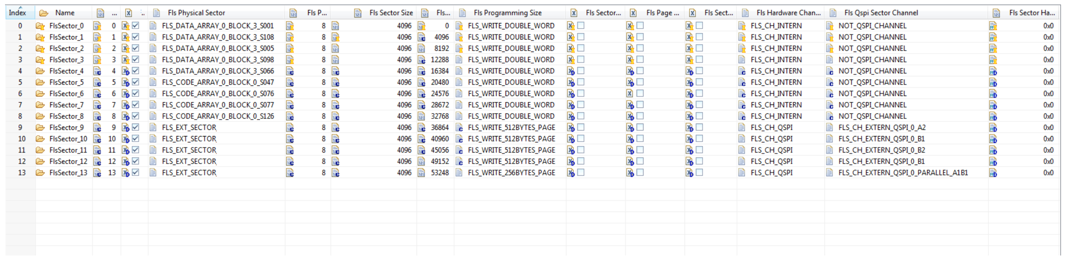
4.7.21 Form FlsSectorList

List of flashable sectors and pages.

Is included by form : Form FlsConfigSet

Included forms :

- Form FlsSector



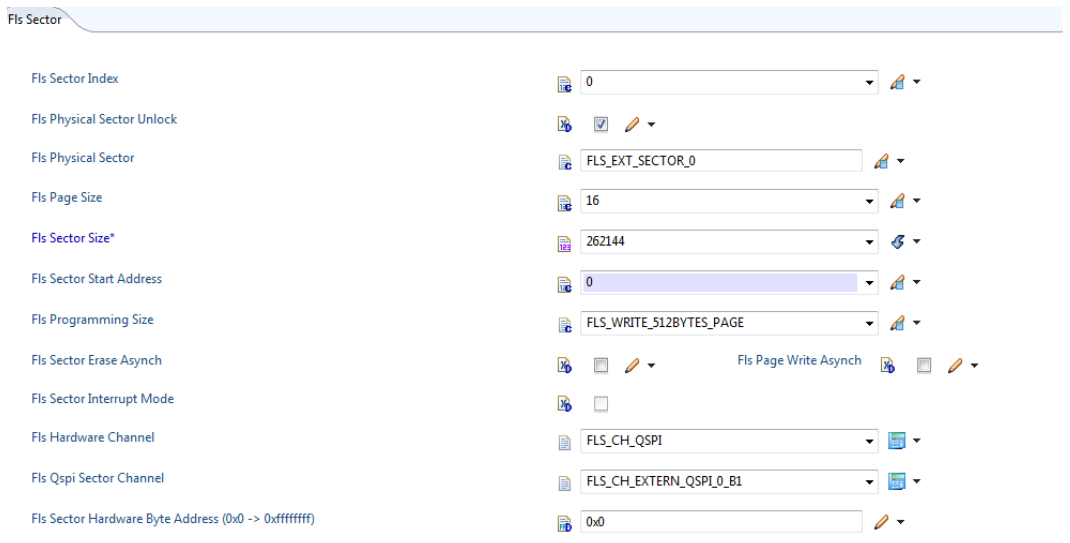
Index	Name	...	..	Fls Physical Sector	Fls P...	Fls Sector Size	Fls...	Fls Programming Size	Fls Sector...	Fls Page ...	Fls Sect...	Fls Hardware Chan...	Fls Qspi Sector Channel	Fls Sector Ha...
0	FlsSector_0	0	✓	FLS_DATA_ARRAY_0_BLOCK_3_S001	8	4096	0	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
1	FlsSector_1	1	✓	FLS_DATA_ARRAY_0_BLOCK_3_S108	8	4096	4096	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
2	FlsSector_2	2	✓	FLS_DATA_ARRAY_0_BLOCK_3_S005	8	4096	8192	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
3	FlsSector_3	3	✓	FLS_DATA_ARRAY_0_BLOCK_3_S098	8	4096	12288	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
4	FlsSector_4	4	✓	FLS_DATA_ARRAY_0_BLOCK_3_S066	8	4096	16384	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
5	FlsSector_5	5	✓	FLS_CODE_ARRAY_0_BLOCK_0_S047	8	4096	20480	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
6	FlsSector_6	6	✓	FLS_CODE_ARRAY_0_BLOCK_0_S076	8	4096	24576	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
7	FlsSector_7	7	✓	FLS_CODE_ARRAY_0_BLOCK_0_S077	8	4096	28672	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
8	FlsSector_8	8	✓	FLS_CODE_ARRAY_0_BLOCK_0_S126	8	4096	32768	FLS_WRITE_DOUBLE_WORD	☐	☐	☐	FLS_CH_INTERN	NOT_QSPI_CHANNEL	0x0
9	FlsSector_9	9	✓	FLS_EXT_SECTOR	8	4096	36864	FLS_WRITE_512BYTES_PAGE	☐	☐	☐	FLS_CH_QSPI	FLS_CH_EXTERN_QSPI_0_A2	0x0
10	FlsSector_10	10	✓	FLS_EXT_SECTOR	8	4096	40960	FLS_WRITE_512BYTES_PAGE	☐	☐	☐	FLS_CH_QSPI	FLS_CH_EXTERN_QSPI_0_B1	0x0
11	FlsSector_11	11	✓	FLS_EXT_SECTOR	8	4096	45056	FLS_WRITE_512BYTES_PAGE	☐	☐	☐	FLS_CH_QSPI	FLS_CH_EXTERN_QSPI_0_B2	0x0
12	FlsSector_12	12	✓	FLS_EXT_SECTOR	8	4096	49152	FLS_WRITE_512BYTES_PAGE	☐	☐	☐	FLS_CH_QSPI	FLS_CH_EXTERN_QSPI_0_B1	0x0
13	FlsSector_13	13	✓	FLS_EXT_SECTOR	8	4096	53248	FLS_WRITE_256BYTES_PAGE	☐	☐	☐	FLS_CH_QSPI	FLS_CH_EXTERN_QSPI_0_PARALLEL_A1B1	0x0

Figure 4-7. Tresos Plugin snapshot for FlsSectorList form.

4.7.21.1 Form FlsSector

Configuration description of a flashable sector

Is included by form : Form FlsSectorList



Fls Sector

Fls Sector Index	0
Fls Physical Sector Unlock	✓
Fls Physical Sector	FLS_EXT_SECTOR_0
Fls Page Size	16
Fls Sector Size*	262144
Fls Sector Start Address	0
Fls Programming Size	FLS_WRITE_512BYTES_PAGE
Fls Sector Erase Asynch	☐
Fls Page Write Asynch	☐
Fls Sector Interrupt Mode	☐
Fls Hardware Channel	FLS_CH_QSPI
Fls Qspi Sector Channel	FLS_CH_EXTERN_QSPI_0_B1
Fls Sector Hardware Byte Address (0x0 -> 0xffffffff)	0x0

Figure 4-8. Tresos Plugin snapshot for FlsSector form.

4.7.21.1.1 FlsSectorIndex (FlsSector)

Fls Sector Index is an invariant index, used to order flash sectors and loop over them in the correct, configured order. Its value should be equal with the position of the configured sector inside the configured sector list (the same value as the shown index). Rationale: The generated .epc configuration might reorder the flash sectors(alphabetically), thus the index parameter changes, becoming out of sync with the real intended order (for example: Fls Sector Start Addresses).

Note

When generating a new configuration, the value of FlsSectorIndex is intended to be kept default, the same as the sector index.

Note

Vendor specific parameter

Table 4-69. Attribute FlsSectorIndex (FlsSector) detailed description

Property	Value
Label	Fls Sector Index
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	index

4.7.21.1.2 FlsPhysicalSectorUnlock (FlsSector)

Fls_Init ensures unlock modify operation for this Flash Physical Sector, it is not possible to lock until a new reset.

Note

Vendor specific parameter. Not applicable for current platform.

Table 4-70. Attribute FlsPhysicalSectorUnlock (FlsSector) detailed description

Property	Value
Label	Fls Physical Sector Unlock
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.7.21.1.3 FlsPhysicalSector (FlsSector)

Physical flash device sector.

Note

Vendor specific parameter

Table 4-71. Attribute FlsPhysicalSector (FlsSector) detailed description

Property	Value
Label	Fls Physical Sector
Type	ENUMERATION
Origin	Custom
Symbolic Name	false

4.7.21.1.4 FlsPageSize (FlsSector)

Size of one page of this sector. Implementation Type: Fls_LengthType.

- For Internal Code Flash page size is 8 bytes
- For Internal Data Flash page size is 8 bytes
- For External Flash page size is configurable and minimum size and alignment is 16 bytes

Table 4-72. Attribute FlsPageSize (FlsSector) detailed description

Property	Value
Label	Fls Page Size
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=8 >=4

4.7.21.1.5 FlsSectorSize (FlsSector)

Size of this sector.

Implementation Type: Fls_LengthType.

Table 4-73. Attribute FlsSectorSize (FlsSector) detailed description

Property	Value
Label	Fls Sector Size
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.7.21.1.6 FlsSectorStartaddress (FlsSector)

Start address of this sector.

Implementation Type: Fls_AddressType.

Table 4-74. Attribute FlsSectorStartaddress (FlsSector) detailed description

Property	Value
Label	Fls Sector Start Address
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.7.21.1.7 FlsProgrammingSize (FlsSector)

This is the size to program Flash memory during one High voltage operation.

Note

For external sectors, this parameter is used for determining the proper amount of data to be written in order to maintain consistency in the external memory, considering the internal buffer and alignment restrictions used for avoiding wraparound. The size of programming size should be set equal to the memory internal buffer

Table 4-75. Attribute FlsProgrammingSize (FlsSector) detailed description

Property	Value
Label	Fls Programming Size
Type	ENUMERATION
Origin	Custom
Symbolic Name	false

4.7.21.1.8 FlsSectorEraseAsynch (FlsSector)

Enable asynchronous execution of the erase job in the Fls_MainFunction function which doesn't wait (block) for completion of the sector erase operation. The flash driver doesn't use the erase access code to the erase flash sector in asynchronous mode so it can be used only on flash sectors which belong to flash array different from flash array the application is executing from.

Note

Vendor specific parameter

Table 4-76. Attribute FlsSectorEraseAsynch (FlsSector) detailed description

Property	Value
Label	Fls Sector Erase Asynch
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.7.21.1.9 FlsPageWriteAsynch (FlsSector)

Enable asynchronous execution of the write job in the Fls_MainFunction function which doesn't wait (block) for completion of the page write operation(s). The flash driver doesn't use the write access code to the write flash page(s) in asynchronous mode so it can be used only on flash sectors which belong to flash array different from flash array the application is executing from.

Note

Vendor specific parameter

Table 4-77. Attribute FlsPageWriteAsynch (FlsSector) detailed description

Property	Value
Label	Fls Page Write Asynch
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.7.21.1.10 FlsSectorIrqMode (FlsSector)

Enable the flash jobs associated with this sector to be executed in interrupt context. This option applies for all type of jobs: read, write, erase. This option applies only for the sectors on which there is hardware support. For example, if a sector is assigned on the internal flash hardware channel, there might not be any interrupt capability avaialable. Note: Flash driver support to execute Sector Erase/Page Write job in either Asynch mode or Interrupt mode

Note

Vendor specific parameter

Table 4-78. Attribute FlsSectorIrqMode (FlsSector) detailed description

Property	Value
Label	Fls Sector Interrupt Mode
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.21.1.11 FlsHwCh (FlsSector)

Assigned Chip Select

Note

Vendor specific parameter

Table 4-79. Attribute FlsHwCh (FlsSector) detailed description

Property	Value
Label	Fls Hardware Channel
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.21.1.12 FlsQspiSectorCh (FlsSector)

Assigned Chip Select

Note

Vendor specific parameter

Table 4-80. Attribute FlsQspiSectorCh (FlsSector) detailed description

Property	Value
Label	Fls Qspi Sector Channel
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.21.1.13 FlsSectorHwAddress (FlsSector)

Hardware address of this sector, as needed by the external flash device (usually starting from 0). Applicable only to external sectors. This value is used to access the hardware sector on the attached device and will be sent as parameter of flash comands, so it should be completed to meet the requirements of the external flash memory type and configured operating mode. Internally, this address is added to the MCU base addresses of each channel, configured in SF{A/B}{1/2}AD registers, in order to select the corresponding external device hw channel. Example: FlsSectorHwAddress = 0x100 Sector hardware

channel = FLS_CH_EXTERN_QSPI_0_A2 FlsSerialFlashA1TopAddr = 0x24000000
The address used by the driver internally will be 0x24000100, thus selecting external flash device A2 and accessing internal location 0x100 of the memory.

Note

Vendor specific parameter

Table 4-81. Attribute FlsSectorHwAddress (FlsSector) detailed description

Property	Value
Label	Fls Qspi Sector Channel
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.22 Form FlsExternalDriver

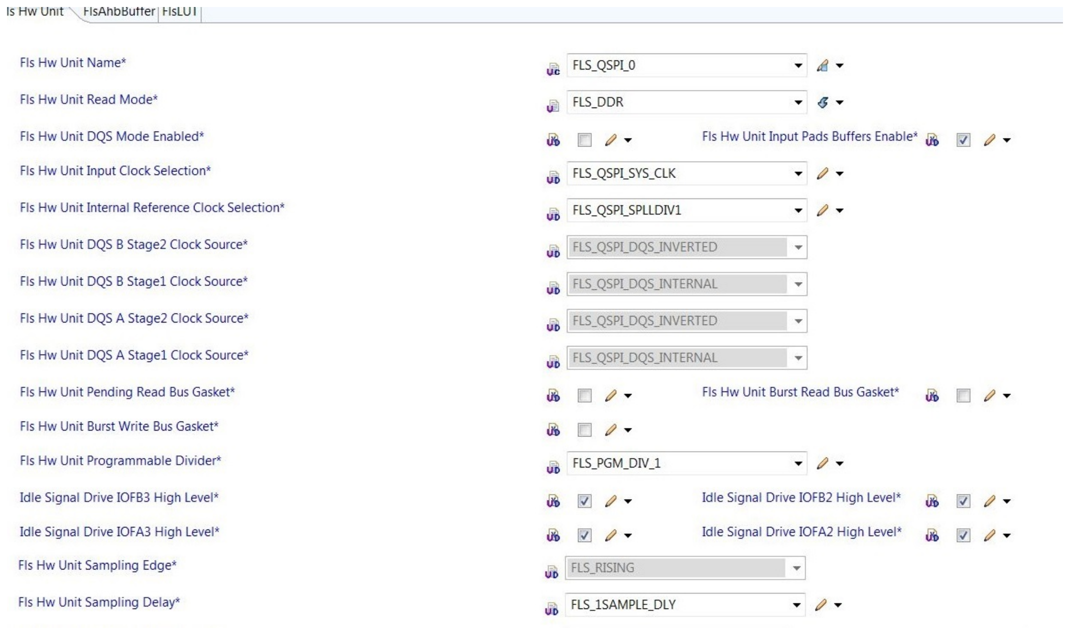
This container is present for external Flash drivers only. Internal Flash drivers do not use the parameter listed in this container, hence its multiplicity is 0 for internal drivers.

Name* FlsExternalDriver

Figure 4-9. Tresos Plugin snapshot for FlsExternalDriver_QSPI form.

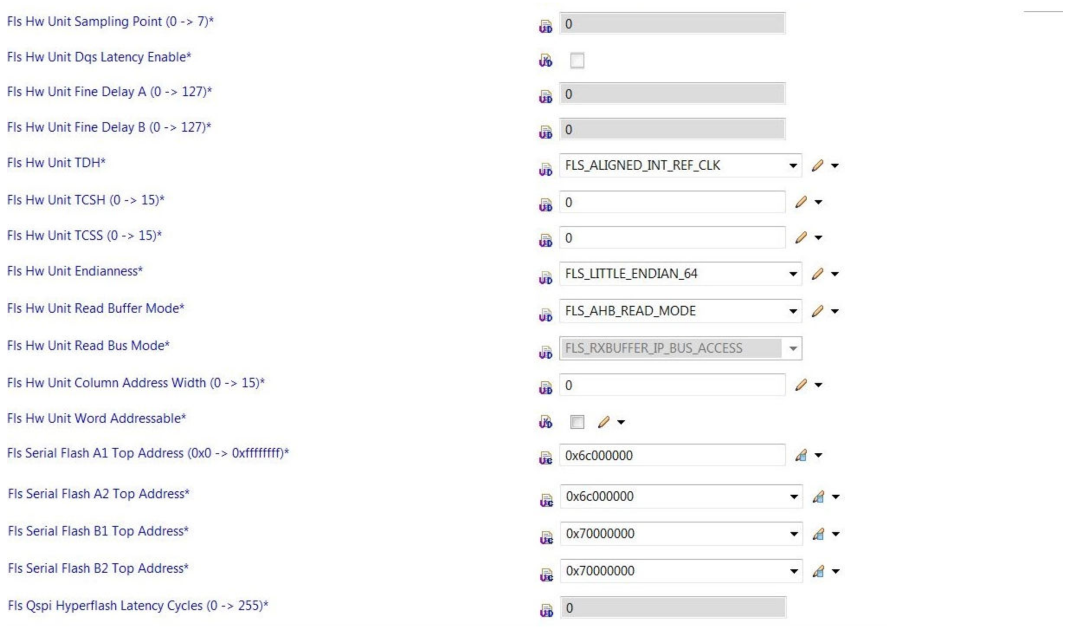
4.7.22.1 Form FlsHwUnit

Container for the configuration of the avaialable external flash memory hardware units



Fls Hw Unit Name*	FLS_QSPI_0
Fls Hw Unit Read Mode*	FLS_DDR
Fls Hw Unit DQS Mode Enabled*	☐
Fls Hw Unit Input Pads Buffers Enable*	✓
Fls Hw Unit Input Clock Selection*	FLS_QSPI_SYS_CLK
Fls Hw Unit Internal Reference Clock Selection*	FLS_QSPI_SPLLDIV1
Fls Hw Unit DQS B Stage2 Clock Source*	FLS_QSPI_DQS_INVERTED
Fls Hw Unit DQS B Stage1 Clock Source*	FLS_QSPI_DQS_INTERNAL
Fls Hw Unit DQS A Stage2 Clock Source*	FLS_QSPI_DQS_INVERTED
Fls Hw Unit DQS A Stage1 Clock Source*	FLS_QSPI_DQS_INTERNAL
Fls Hw Unit Pending Read Bus Gasket*	☐
Fls Hw Unit Burst Read Bus Gasket*	☐
Fls Hw Unit Burst Write Bus Gasket*	☐
Fls Hw Unit Programmable Divider*	FLS_PGM_DIV_1
Idle Signal Drive IOFB3 High Level*	✓
Idle Signal Drive IOFB2 High Level*	✓
Idle Signal Drive IOFA3 High Level*	✓
Idle Signal Drive IOFA2 High Level*	✓
Fls Hw Unit Sampling Edge*	FLS_RISING
Fls Hw Unit Sampling Delay*	FLS_1SAMPLE_DLY

Figure 4-10. Tresos Plugin snapshot for FlsHwUnit form.



Fls Hw Unit Sampling Point (0 -> 7)*	0
Fls Hw Unit Dqs Latency Enable*	☐
Fls Hw Unit Fine Delay A (0 -> 127)*	0
Fls Hw Unit Fine Delay B (0 -> 127)*	0
Fls Hw Unit TDH*	FLS_ALIGNED_INT_REF_CLK
Fls Hw Unit TCSH (0 -> 15)*	0
Fls Hw Unit TCSS (0 -> 15)*	0
Fls Hw Unit Endianness*	FLS_LITTLE_ENDIAN_64
Fls Hw Unit Read Buffer Mode*	FLS_AHB_READ_MODE
Fls Hw Unit Read Bus Mode*	FLS_RXBUFFER_IP_BUS_ACCESS
Fls Hw Unit Column Address Width (0 -> 15)*	0
Fls Hw Unit Word Addressable*	☐
Fls Serial Flash A1 Top Address (0x0 -> 0xffffffff)*	0x6c000000
Fls Serial Flash A2 Top Address*	0x6c000000
Fls Serial Flash B1 Top Address*	0x70000000
Fls Serial Flash B2 Top Address*	0x70000000
Fls Qspi Hyperflash Latency Cycles (0 -> 255)*	0

Figure 4-11. Tresos Plugin snapshot for FlsHwUnit form(continues).

4.7.22.1.1 FlsHwUnitName (FlsHwUnit)

The name of the configured harwdare unit name. The configured parameters will apply to this hardware unit name only. The name of the hardware unit name represents the physical hardware unit available on chip.

Table 4-82. Attribute FlsHwUnitName (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Name
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_QSPI_0

4.7.22.1.2 FlsHwUnitReadMode (FlsHwUnit)

The hardware unit read mode: FLS_SDR (single data rate) which samples incoming data on a single edge. FLS_DDR (double data rate) which samples incoming data on both edges.

Table 4-83. Attribute FlsHwUnitReadMode (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Read Mode
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_SDR

4.7.22.1.3 FlsHwUnitDqsMode (FlsHwUnit)

Enables DQS sampling mode. If enabled, two DQS sampling modes are available(Internal loopback DQS and Pad loopback DQS). If disabled, only Internal sampling mode is available.

Table 4-84. Attribute FlsHwUnitDqsMode (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit DQS Mode Enabled
Type	BOOLEAN
Origin	NXP
Symbolic Name	false

4.7.22.1.4 FlsHwUnitInputPadsBufferEn (FlsHwUnit)

Enables input buffer of QSPI pads for SDR modes SCLKCFG[7].

Table 4-85. Attribute FlsHwUnitInputPadsBufferEn (FlsHwUnit) detailed description

Property	Value
Label	Fls Page Size
Type	BOOLEAN
Origin	NXP
Default	true

4.7.22.1.5 FlsHwUnitInputClockSel (FlsHwUnit)

Select source for AHB read interface clock, module clock and bus interface clock. SCLKCFG[6] bit. Option 1: BUS_CLK. Option 2: SYS_CLK.

Table 4-86. Attribute FlsHwUnitInputClockSel (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Input Clock Selection
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_QSPI_SYS_CLK

4.7.22.1.6 FlsHwUnitInternalRefClockSel (FlsHwUnit)

Source clock for external QSPI clock(SFCK) sent to the memory. SCLKCFG[4] bit. Option 1: SPLLDIV1. Option 2: FIRCDIV1

Table 4-87. Attribute FlsHwUnitInternalRefClockSel (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Internal Reference Clock Selection
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_QSPI_SPLLDIV1

4.7.22.1.7 FlsHwUnitDqsBStage2ClkSource (FlsHwUnit)

Select source clock for DQS B Stage 2 clock. SCLKCFG[3] bit. Option 1: INVERTED. Option 2: NON-INVERTED

Table 4-88. Attribute FlsHwUnitDqsBStage2ClkSource (FlsHwUnit) detailed description

Property	Value
Label	Fls Programming Size
Type	ENUMERATION
Origin	NXP
Symbolic Name	false

4.7.22.1.8 FlsHwUnitDqsBStage1ClkSource (FlsHwUnit)

Select source clock for DQS B Stage 1 clock. SCLKCFG[2] bit. Option 1: DQS_INTERNAL. DQS internal loopback clock. Option 2: DQS_LOOPBACK. DQS pad loopback clock, derived from SCKB pad. .

Table 4-89. Attribute FlsHwUnitDqsBStage1ClkSource (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit DQS B Stage1 Clock Source
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_QSPI_DQS_INTERNAL

4.7.22.1.9 FlsHwUnitDqsAStage2ClkSource (FlsHwUnit)

Select source clock for DQS A Stage 2 clock. SCLKCFG[1] bit. Option 1: INVERTED. Option 2: NON-INVERTED

Table 4-90. Attribute FlsHwUnitDqsAStage2ClkSource (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit DQS A Stage2 Clock Source
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_QSPI_DQS_INVERTED

4.7.22.1.10 FlsHwUnitDqsAStage1ClkSource (FlsHwUnit)

Select source clock for DQS A Stage 1 clock. SCLKCFG[0] bit. Option 1: DQS_INTERNAL. DQS internal loopback clock. Option 2: DQS_LOOPBACK. DQS pad loopback clock, derived from SCKB pad.

Table 4-91. Attribute FlsHwUnitDqsAStage1ClkSource (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit DQS A Stage1 Clock Source
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_QSPI_DQS_INTERNAL

4.7.22.1.11 FlsHwUnitPendingReadBusGasket (FlsHwUnit)

Assigned Chip Select

Table 4-92. Attribute FlsHwUnitPendingReadBusGasket (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Pending Read Bus Gasket
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.22.1.12 FlsHwUnitBurstReadBusGasket (FlsHwUnit)

Assigned Chip Select

Table 4-93. Attribute FlsHwUnitBurstReadBusGasket (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Burst Read Bus Gasket
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.22.1.13 FlsHwUnitBurstWriteBusGasket (FlsHwUnit)

Burst writes enable controls the bus gasket's handling of burst read transactions. SOCCFG[18]. 0: Burst writes are converted into a series of single transactions on the slave side of the gasket. 1: Burst writes are optimized for best system performance. Note this setting treats writes as imprecise such that an error response on any beat of the burst is reported on the last beat.

Table 4-94. Attribute FlsHwUnitBurstWriteBusGasket (FlsHwUnit) detailed description

Property	Value
Label	Fls Qspi Sector Channel
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.22.1.14 FlsHwUnitProgrammableDivider (FlsHwUnit)

Programmable divider configuration selection. Based on this devider value, the external memory clock and internal sampling clock is derived.

Table 4-95. Attribute FlsHwUnitProgrammableDivider (FlsHwUnit) detailed description

Property	Value
Label	Fls Qspi Sector Channel
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_PGM_DIV_1

4.7.22.1.15 IdleSignalDriveIOFB3HighLvl (FlsHwUnit)

Idle Signal Drive IOFB[3] Flash B. This bit determines the logic level the IOFB[3] output of the QuadSPI module is driven to in the inactive state.

Table 4-96. Attribute IdleSignalDriveIOFB3HighLvl (FlsHwUnit) detailed description

Property	Value
Label	Idle Signal Drive IOFB3 High Level

Table continues on the next page...

Table 4-96. Attribute IdleSignalDriveIOFB3HighLvl (FlsHwUnit) detailed description (continued)

Property	Value
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	true

4.7.22.1.16 IdleSignalDriveIOFB2HighLvl (FlsHwUnit)

Idle Signal Drive IOFB[2] Flash B. This bit determines the logic level the IOFB[2] output of the QuadSPI module is driven to in the inactive state.

Table 4-97. Attribute IdleSignalDriveIOFB2HighLvl (FlsHwUnit) detailed description

Property	Value
Label	Idle Signal Drive IOFB2 High Level
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	true

4.7.22.1.17 IdleSignalDriveIOFA3HighLvl (FlsHwUnit)

Idle Signal Drive IOFA[3] Flash A. This bit determines the logic level the IOFA[3] output of the QuadSPI module is driven to in the inactive state.

Table 4-98. Attribute IdleSignalDriveIOFA3HighLvl (FlsHwUnit) detailed description

Property	Value
Label	Idle Signal Drive IOFA3 High Level
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	true

4.7.22.1.18 IdleSignalDriveIOFA2HighLvl (FlsHwUnit)

Idle Signal Drive IOFA[3] Flash A. This bit determines the logic level the IOFA[3] output of the QuadSPI module is driven to in the inactive state.

Table 4-99. Attribute IdleSignalDriveIOFA2HighLvl (FlsHwUnit) detailed description

Property	Value
Label	Idle Signal Drive IOFA2 High Level
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	true

4.7.22.1.19 FlsHwUnitSamplingEdge (FlsHwUnit)

The hardware unit sampling edge. Selects the edge of the sampling clock valid for full speed commands. This option is also used in DQS mode and ignored when using non-DQS DDR instructions. FLS_RISING: Select sampling at non-inverted internal DQS (use rising edge). FLS_FALLING: Select sampling at inverted internal DQS (use falling edge). The sampling edge selection is valid only for FLS_SDR mode, not FLS_DDR.

Table 4-100. Attribute FlsHwUnitSamplingEdge (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Sampling Edge
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_RISING

4.7.22.1.20 FlsHwUnitSamplingDly (FlsHwUnit)

The hardware unit sampling edge. Selects the edge of the sampling clock valid for full speed commands. This option is also used in DQS mode and ignored when using non-DQS DDR instructions. FLS_RISING: Select sampling at non-inverted internal DQS (use rising edge). FLS_FALLING: Select sampling at inverted internal DQS (use falling edge). The sampling edge selection is valid only for FLS_SDR mode, not FLS_DDR.

Table 4-101. Attribute FlsHwUnitSamplingDly (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Sampling Delay
Type	ENUMERATION
Origin	NXP

Table continues on the next page...

Table 4-101. Attribute FlsHwUnitSamplingDly (FlsHwUnit) detailed description (continued)

Property	Value
Symbolic Name	false
Default	FLS_RISING

4.7.22.1.21 FlsHwUnitSamplingPoint (FlsHwUnit)

DDR Sampling point. Selects the sampling point for incoming data when serial flash is executing a DDR instruction. The sampling edge selection is valid only for FLS_DDR mode, not FLS_SDR.

Table 4-102. Attribute FlsHwUnitSamplingPoint (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Sampling Point
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.1.22 FlsHwUnitDqsLatencyEnable (FlsHwUnit)

DQS Latency Enable. Feature used to support external devices which add latency cycles in the DQS signal. When no signal is provided by the external devices, data is not sampled, thus the memory stretches the timing. DQS Latency is applicable only for DQS sampling mode: FLS_EXTERNAL_DQS.

Table 4-103. Attribute FlsHwUnitDqsLatencyEnable (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Dqs Latency Enable
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	true

4.7.22.1.23 FlsHwUnitFineDelayA (FlsHwUnit)

The hardware unit fine delay: Coarse delay value, applicable for FLS_INTERNAL_DQS and FLS_EXTERNAL_DQS sampling modes. Fine delay applicable to Flash A. 0 - 63 delay units.

Table 4-104. Attribute FlsHwUnitFineDelayA (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Fine Delay A
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.1.24 FlsHwUnitFineDelayB (FlsHwUnit)

The hardware unit fine delay: Coarse delay value, applicable for FLS_INTERNAL_DQS and FLS_EXTERNAL_DQS sampling modes. Fine delay applicable to Flash B. 0 - 63 delay units

Table 4-105. Attribute FlsHwUnitFineDelayB (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Fine Delay B
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.1.25 FlsHwUnitTdh (FlsHwUnit)

The hardware unit fine delay: Coarse delay value, applicable for FLS_INTERNAL_DQS and FLS_EXTERNAL_DQS sampling modes. Fine delay applicable to Flash B. 0 - 63 delay units

Table 4-106. Attribute FlsHwUnitTdh (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit TDH
Type	ENUMERATION

Table continues on the next page...

Table 4-106. Attribute FlsHwUnitTdh (FlsHwUnit) detailed description (continued)

Property	Value
Origin	NXP
Symbolic Name	false
Default	FLS_ALIGNED_INT_REF_CLK

4.7.22.1.26 FlsHwUnitTcsh (FlsHwUnit)

TCSH: Serial flash CS hold time in terms of serial flash clock cycles. A bigger value will release the CS signal later after the transaction ends. The actual delay between chip select and clock is defined as: TCSH = 1 SCK clk if N= 0/1 else, N SCK clk if N>1, where N is the setting of TCSH

Table 4-107. Attribute FlsHwUnitTcsh (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit TCSH
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.1.27 FlsHwUnitTcss (FlsHwUnit)

TCSS: Serial flash CS setup time in terms of serial flash clock cycles. A bigger value will pull the CS signal earlier before the transaction starts. The actual delay between chip select and clock is defined as: TCSS = 0.5 SCK clk if N= 0/1 else, N+0.5 SCK clk if N>1, where N is the setting of TCSS.

Table 4-108. Attribute FlsHwUnitTcss (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit TCSS
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.1.28 FlsHwUnitEndianness (FlsHwUnit)

Defines the endianness of the QuadSPI module, which can swap the orderd of the read/ write data. By default the data is always returned in 64 bit LE format on the AHB bus and 32 bit LE format on the IPS interface when read via the RX buffer and written in 32 bit LE format when written via the TX buffer. FLS_BIG_ENDIAN_64 = 64 bit BE. FLS_LITTLE_ENDIAN_32 = 32 bit LE. FLS_BIG_ENDIAN_32 = 32 bit BE. FLS_LITTLE_ENDIAN_64 = 64 bit LE

Table 4-109. Attribute FlsHwUnitEndianness (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Endianness
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_LITTLE_ENDIAN_64

4.7.22.1.29 FlsHwUnitReadBufferMode (FlsHwUnit)

Selects the read buffer mode: IP buffer or AHB buffer. If AHB mode is used, read transactions are triggered by memory mapped access and data read from external flash device is stored inside the AHB buffers. If IP mode is used, read transactions are triggered by IP commands and data read from external flash device is stored inside the RX buffer.

Table 4-110. Attribute FlsHwUnitReadBufferMode (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Read Buffer Mode
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_AHB_READ_MODE

4.7.22.1.30 FlsHwUnitRxBufferAccessMode (FlsHwUnit)

Selects the bus(AHB or IP) used to access the RX Buffer content. This setting is only applicable for RX Buffer, thus is applicable only for IP bus mode, otherwise the AHB buffers are used instead of the RX Buffer.

Table 4-111. Attribute FlsHwUnitRxBufferAccessMode (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Read Bus Mode
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_RXBUFFER_IP_BUS_ACCESS

4.7.22.1.31 FlsHwUnitColumnAddressWidth (FlsHwUnit)

Column Address Space. Defines the width of the column address. Example: If the coulmn address is for example [2:0] of QSPI_SFAR/AHB address, then CAS must be 3. If there is no column address separation in any serial flash this value must be programmed to 0.

Table 4-112. Attribute FlsHwUnitColumnAddressWidth (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Column Address Width
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.1.32 FlsHwUnitWordAddressable (FlsHwUnit)

Defines whether the serial flash is a byte addressable flash or a word addressable flash. According to this bit configuration the address is re-mapped to the flash interface. DISABLED: Byte addressable serial flash mode. ENABLED: Word (2 byte) addressable serial flash mode. If the incoming address is 0x2004, the controller re-maps this address to access the flash location 0x1002

Note: When FlsHwUnitWordAddressable is set, the API functions(especially Read, Compare, BlankCheck) require both length and address parameters to be word aligned.

Table 4-113. Attribute FlsHwUnitWordAddressable (FlsHwUnit) detailed description

Property	Value
Label	Fls Hw Unit Word Addressable
Type	BOOLEAN

Table continues on the next page...

Table 4-113. Attribute FlsHwUnitWordAddressable (FlsHwUnit) detailed description (continued)

Property	Value
Origin	NXP
Symbolic Name	false
Default	false

4.7.22.1.33 FlsSerialFlashA1TopAddr (FlsHwUnit)

The address mapping for the serial flash A1 external memory device. All accesses targeting the memory range [TOP_ADDR_MEMA1(SFA1AD) - QSPI_AMBA_BASE] will be directed to flash memory A1. The memory range should be configured to the size of the external memory. If an external device is not attached, or not used on this channel, the value of the register can be set the same as the previous one, thus allocating no space.

Table 4-114. Attribute FlsSerialFlashA1TopAddr (FlsHwUnit) detailed description

Property	Value
Label	Fls Serial Flash A1 Top Address
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0x6c000000

4.7.22.1.34 FlsSerialFlashA2TopAddr (FlsHwUnit)

The address mapping for the serial flash A2 external memory device. All accesses targeting the memory range [TOP_ADDR_MEMA2(SFA2AD) - TOP_ADDR_MEMA1(SFA1AD)] will be directed to flash memory A2. The memory range should be configured to the size of the external memory. If an external device is not attached, or not used on this channel, the value of the register can be set the same as the previous one, thus allocating no space.

Table 4-115. Attribute FlsSerialFlashA2TopAddr (FlsHwUnit) detailed description

Property	Value
Label	Fls Serial Flash A2 Top Address
Type	INTEGER
Origin	NXP
Symbolic Name	false

Table continues on the next page...

Table 4-115. Attribute FlsSerialFlashA2TopAddr (FlsHwUnit) detailed description (continued)

Property	Value
Default	0x6c000000

4.7.22.1.35 FlsSerialFlashB1TopAddr (FlsHwUnit)

The address mapping for the serial flash B1 external memory device. All accesses targeting the memory range [TOP_ADDR_MEMB1(SFB1AD) - TOP_ADDR_MEMA2(SFA2AD)] will be directed to flash memory B1. The memory range should be configured to the size of the external memory. If an external device is not attached, or not used on this channel, the value of the register can be set the same as the previous one, thus allocating no space.

Table 4-116. Attribute FlsSerialFlashB1TopAddr (FlsHwUnit) detailed description

Property	Value
Label	Fls Serial Flash B1 Top Address
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0x70000000

4.7.22.1.36 FlsSerialFlashB2TopAddr (FlsHwUnit)

The address mapping for the serial flash B2 external memory device. All accesses targeting the memory range [TOP_ADDR_MEMB2(SFB2AD) - TOP_ADDR_MEMB1(SFB1AD)] will be directed to flash memory B2. The memory range should be configured to the size of the external memory. If an external device is not attached, or not used on this channel, the value of the register can be set the same as the previous one, thus allocating no space.

Table 4-117. Attribute FlsSerialFlashB2TopAddr (FlsHwUnit) detailed description

Property	Value
Label	Fls Serial Flash B2 Top Address
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0x70000000

4.7.22.1.37 FlsExtUnitRegWidth (FlsHwUnit)

The external memory unit register width, expressed in bytes (For example the width of the Status register). 1 - External unit register width is 1 byte (8bits). 2 - External unit register width is 2 byte (16bits). 3 - External unit register width is 3 byte (24bits). 4 - External unit register width is 4 byte (32bits).

Table 4-118. Attribute FlsExtUnitRegWidth (FlsHwUnit) detailed description

Property	Value
Label	Fls External Unit Register Width
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	1

4.7.22.1.38 FlsBusyBitValue (FlsHwUnit)

The busy bit flag value, from the external device status register, corresponding to the BUSY state. 0 - Busy flag in status register is 0 when flash devices are busy. 1 - Busy flag in status register is 1 when flash devices are busy.

Table 4-119. Attribute FlsBusyBitValue (FlsHwUnit) detailed description

Property	Value
Label	Fls Busy Bit Value
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	1

4.7.22.1.39 FlsBusyBitPositionOffset (FlsHwUnit)

The busy bit flag position offset, inside the status register of the external device status register. Valid range: 0 - 31, for a 32 bits status register. If the width of the status register si smaller, the range is restricted accordingly. Ex: 3 - Busy bit flag is bit number 3 in the status register.

Table 4-120. Attribute FlsBusyBitPositionOffset (FlsHwUnit) detailed description

Property	Value
Label	Fls Busy Bit Position Offset
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.1.40 FlsQspiDeviceId (FlsHwUnit)

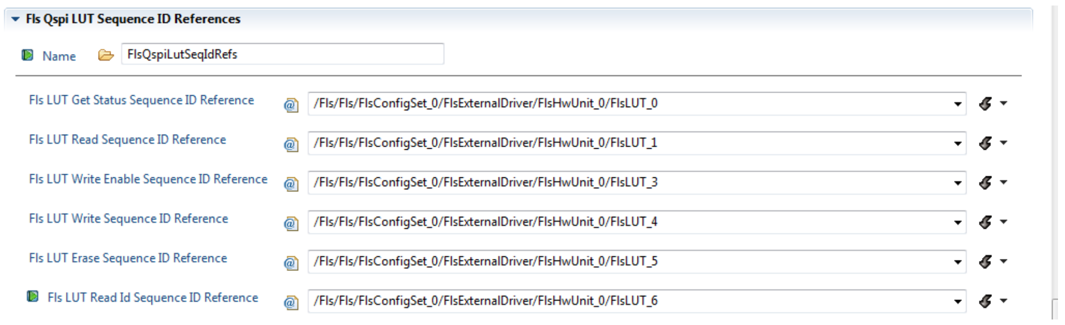
External memory ID. If the associated "FLS_E_UNEXPECTED_FLASH_ID" error is enabled, at Init, the configured value is checked against the value read from memory. The memory ID is read from the memory using the configured READ_ID LUT sequence. Example for a Macronix device: Configured value of FlsQspiDeviceId = 0x16C2, meaning Manufacturer ID: 0xC2, Device ID: 0x16. The configured READ_ID LUT sequence schedules a read id command(ex:REMS 0x90) with read length 2 bytes. Note: The read ID command is sent only to the device attached to QSPI_x_A1 device. Note: This parameter can be configured only when both FLS_E_UNEXPECTED_FLASH_ID and FlsLUTReadIdSequenceIdRef references are enabled.

Table 4-121. Attribute FlsQspiDeviceId (FlsHwUnit) detailed description

Property	Value
Label	Fls Qspi Device Id
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.2 Form FlsQspiLutSeqIdRefs

Container for the references to the QSPI LUT Sequences ID. The ID is used as a pointer to the correct position of the LUT sequence inside the LUT.



Name	FlsQspiLutSeqIdRefs
Fls LUT Get Status Sequence ID Reference	/Fls/Fls/FlsConfigSet_0/FlsExternalDriver/FlsHwUnit_0/FlsLUT_0
Fls LUT Read Sequence ID Reference	/Fls/Fls/FlsConfigSet_0/FlsExternalDriver/FlsHwUnit_0/FlsLUT_1
Fls LUT Write Enable Sequence ID Reference	/Fls/Fls/FlsConfigSet_0/FlsExternalDriver/FlsHwUnit_0/FlsLUT_3
Fls LUT Write Sequence ID Reference	/Fls/Fls/FlsConfigSet_0/FlsExternalDriver/FlsHwUnit_0/FlsLUT_4
Fls LUT Erase Sequence ID Reference	/Fls/Fls/FlsConfigSet_0/FlsExternalDriver/FlsHwUnit_0/FlsLUT_5
Fls LUT Read Id Sequence ID Reference	/Fls/Fls/FlsConfigSet_0/FlsExternalDriver/FlsHwUnit_0/FlsLUT_6

Figure 4-12. Tresos Plugin snapshot for FlsQspiLutSeqIdRefs form.

4.7.22.2.1 FlsLUTGetStatusSequenceIdRef (FlsQspiLutSeqIdRefs)

Reference to the LUT Sequence ID which will be used for getting the status(idle/busy) of the external memory. Command is used for polling the job status of the external memory, usually by reading the external memory status register.

Table 4-122. Attribute FlsLUTGetStatusSequenceIdRef (FlsHwUnit) detailed description

Property	Value
Label	Fls LUT Get Status Sequence ID Reference
Type	REFERENCE
Origin	NXP

4.7.22.2.2 FlsLUTReadSequenceIdRef (FlsQspiLutSeqIdRefs)

Reference to the LUT Sequence ID which will be used for read and compare operations, in both AHB and IP modes.

Table 4-123. Attribute FlsLUTReadSequenceIdRef (FlsHwUnit) detailed description

Property	Value
Label	Fls LUT Read Sequence ID Reference
Type	REFERENCE
Origin	NXP

4.7.22.2.3 FlsLUTWriteEnableSequenceIdRef (FlsQspiLutSeqIdRefs)

Reference to the LUT Sequence ID which will be used prior to write/erase operations, in order to set the write enable latch.

Table 4-124. Attribute FlsLUTWriteEnableSequenceIdRef (FlsHwUnit) detailed description

Property	Value
Label	Fls LUT Write Enable Sequence ID Reference
Type	REFERENCE
Origin	NXP

4.7.22.2.4 FlsLUTWriteSequenceIdRef (FlsQspiLutSeqIdRefs)

Reference to the LUT Sequence ID which will be used for write operations.

Table 4-125. Attribute FlsLUTWriteSequenceIdRef (FlsHwUnit) detailed description

Property	Value
Label	Fls LUT Write Sequence ID Reference
Type	REFERENCE
Origin	NXP

4.7.22.2.5 FlsLUTEraseSequenceIdRef (FlsQspiLutSeqIdRefs)

Reference to the LUT Sequence ID which will be used for erase operations.

Table 4-126. Attribute FlsLUTEraseSequenceIdRef (FlsHwUnit) detailed description

Property	Value
Label	Fls LUT Erase Sequence ID Reference
Type	REFERENCE
Origin	NXP

4.7.22.2.6 FlsLUTReadIdSequenceIdRef (FlsQspiLutSeqIdRefs)

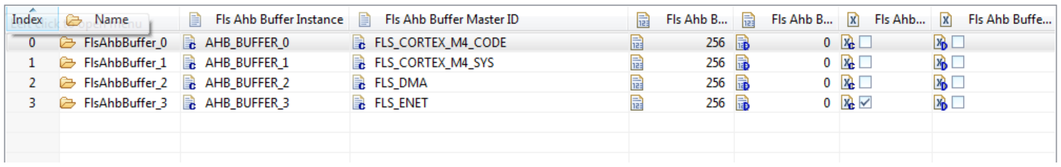
Reference to the LUT Sequence ID which will be used for erase operations. Note: When this sequence reference is enabled, the user should enable and configure also the FLS_E_UNEXPECTED_FLASH_ID error and the FlsQspiDeviceID.

Table 4-127. Attribute FlsLUTReadIdSequenceIdRef (FlsHwUnit) detailed description

Property	Value
Label	Fls LUT Read Id Sequence ID Reference
Type	REFERENCE
Origin	NXP

4.7.22.3 Form FlsAhbBuffer

Container for the configuration of the AHB read buffers. Holds the configuration for each AHB buffer configured for AHB read mode.

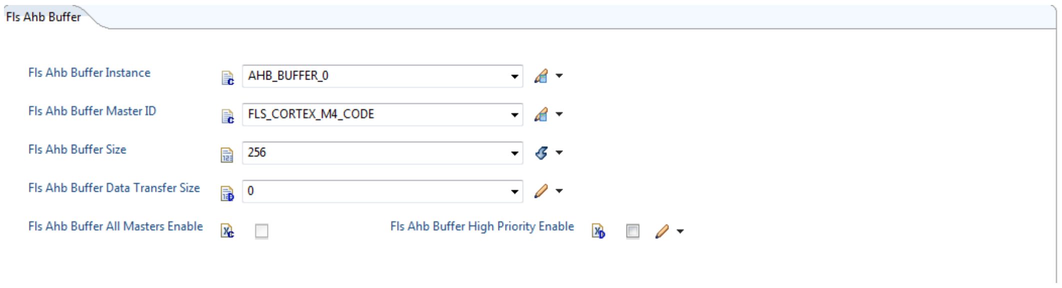


Index	Name	Fls Ahb Buffer Instance	Fls Ahb Buffer Master ID	Fls Ahb B...	Fls Ahb B...	Fls Ahb...	Fls Ahb Buffe...
0	FlsAhbBuffer_0	AHB_BUFFER_0	FLS_CORTEX_M4_CODE	256	0	☐	☐
1	FlsAhbBuffer_1	AHB_BUFFER_1	FLS_CORTEX_M4_SYS	256	0	☐	☐
2	FlsAhbBuffer_2	AHB_BUFFER_2	FLS_DMA	256	0	☐	☐
3	FlsAhbBuffer_3	AHB_BUFFER_3	FLS_ENET	256	0	☑	☐

Figure 4-13. Tresos Plugin snapshot for FlsAhbBuffer form.

4.7.22.3.1 Form FlsAhbBuffe

Configuration description of a flashable FlsAhbBuffe



Fls Ahb Buffer

Fls Ahb Buffer Instance	AHB_BUFFER_0
Fls Ahb Buffer Master ID	FLS_CORTEX_M4_CODE
Fls Ahb Buffer Size	256
Fls Ahb Buffer Data Transfer Size	0
Fls Ahb Buffer All Masters Enable	☐
Fls Ahb Buffer High Priority Enable	☐

Figure 4-14. Tresos Plugin snapshot for FlsSector form.

4.7.22.3.1.1 FlsAhbBufferInstance (FlsAhbBuffe)

Selects the AHB buffer instance for which this configuration applies. If an instance is not present, the corresponding AHB buffer will be configured with size 0. The size of the AHB_BUFFER_3 instance will be configured to at least the selected size, or more, up until the maximum value is reached. For more details about the maximum avaialable size see chip specific details.

Table 4-128. Attribute FlsAhbBufferInstance (FlsAhbBuffe) detailed description

Property	Value
Label	Fls Ahb Buffer Instance
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	AHB_BUFFER_0

4.7.22.3.1.2 FlsAhbBufferMasterId (FlsAhbBuffe)

The ID of the AHB master associated with this buffer. Any AHB access with this master port number is routed to this buffer. It must be ensured that the master IDs associated with all buffers must be different.

Table 4-129. Attribute FlsAhbBufferMasterId (FlsAhbBuffe) detailed description

Property	Value
Label	Fls Ahb Buffer Master ID
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_CORTEX_M4_CODE

4.7.22.3.1.3 FlsAhbBufferSize (FlsAhbBuffe)

The size allocated to this AHB Buffer instance. The minimum size is 8 bytes, the maximum size is the entire AHB Buffers section size(1024 bytes). There is no benefit in configuring a buffer size greater than FlsAhbBufferDataTransferSize, as the locations greater than this will never be used..

Table 4-130. Attribute FlsAhbBufferSize (FlsAhbBuffe) detailed description

Property	Value
Label	Fls Physical Sector
Type	INTEGER
Origin	NXP
Symbolic Name	0
Default	0

4.7.22.3.1.4 FlsAhbBufferDataTransferSize (FlsAhbBuffe)

Defines the data transfer size in bytes of an AHB triggered access to serial flash (ADATSZ bitfield). When value is 0, default, the data size mentioned the sequence pointed to by the SEQID field overrides this value. Software should ensure that this transfer size is not greater than the size of this buffer. Note: When this value is 0,the data size mentioned in the LUT sequence pointed to by the SEQID field overrides this value.

Table 4-131. Attribute FlsAhbBufferDataTransferSize (FlsAhbBuffe) detailed description

Property	Value
Label	Fls Page Size
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.3.1.5 FlsAhbBufferAllMasters (FlsAhbBuffe)

When set, buffer3 acts as an all-master buffer. Any AHB access with a master port number not matching with the master ID of buffer0 or buffer1 or buffer2 is routed to buffer3. When set, the Master ID parameter for this buffer is ignored.

Table 4-132. Attribute FlsAhbBufferAllMasters (FlsAhbBuffe) detailed description

Property	Value
Label	Fls Ahb Buffer All Masters Enable
Type	BOOLEAN
Origin	NXP
Symbolic Name	false

4.7.22.3.1.6 FlsAhbBufferHighPriority (FlsAhbBuffe)

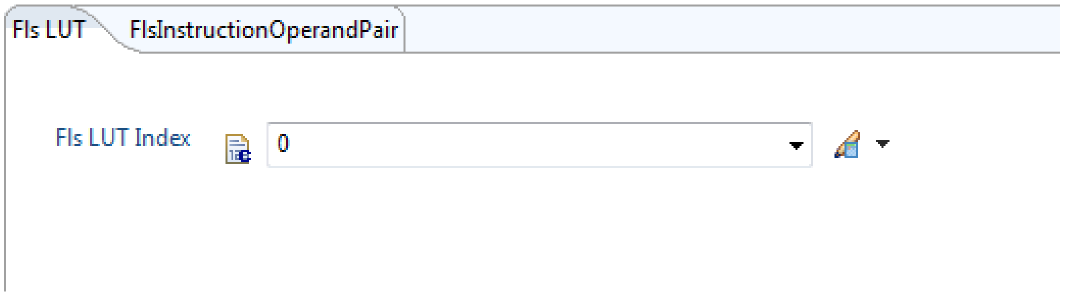
When set, the master associated with buffer0 is assigned a priority higher than the rest of the masters. An access by a high priority master will suspend any ongoing prefetch by another AHB master and will be serviced on high priority.

Table 4-133. Attribute FlsAhbBufferHighPriority (FlsAhbBuffe) detailed description

Property	Value
Label	Fls Ahb Buffer High Priority Enable
Type	BOOLEAN
Origin	NXP
Symbolic Name	false
Default	false

4.7.22.4 Form FlsLUT

Configuration description of a flashable FlsLUT



Fls LUT | FlsInstructionOperandPair

Fls LUT Index: 0

Figure 4-15. Tresos Plugin snapshot for FlsLUT form.

4.7.22.4.1 FlsLUTIndex (FlsAhbBuffe)

Container for the configuration of the Look Up Table holding all the Instruction/ Operands sequences. A sequence consists of a series of up to 8 instruction/operands pairs, which can ocupy up to 4 LUTs, which are executed whenever a command is triggered to the external flash memory.

Note

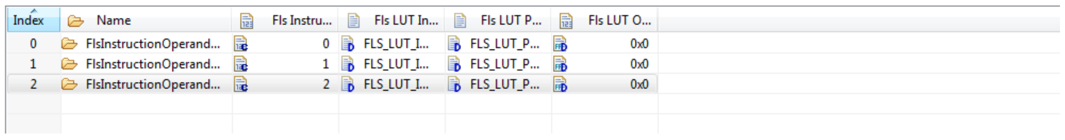
Vendor specific parameter

Table 4-134. Attribute FlsLUTIndex (FlsAhbBuffe) detailed description

Property	Value
Label	Fls LUT Index
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.5 Form FlsInstructionOperandPairList

List of flashable FlsInstructionOperandPairList.

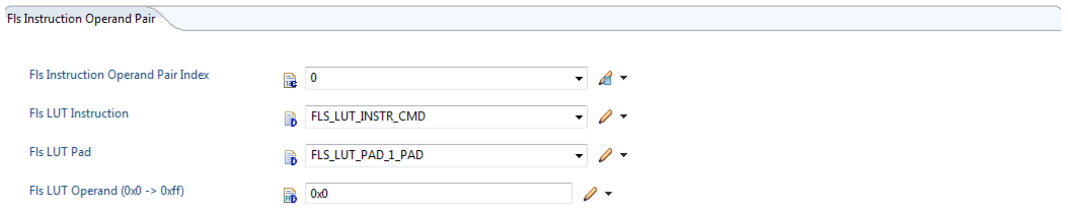


Index	Name	Fls Instru...	Fls LUT In...	Fls LUT P...	Fls LUT O...
0	FlsInstructionOperand...	0	FLS_LUT_I...	FLS_LUT_P...	0x0
1	FlsInstructionOperand...	1	FLS_LUT_I...	FLS_LUT_P...	0x0
2	FlsInstructionOperand...	2	FLS_LUT_I...	FLS_LUT_P...	0x0

Figure 4-16. Tresos Plugin snapshot for FlsInstructionOperandPairList form.

4.7.22.5.1 Form FlsInstructionOperandPair

Configuration description of a flashable FlsInstructionOperandPair


Fls Instruction Operand Pair

Fls Instruction Operand Pair Index: 0

Fls LUT Instruction: FLS_LUT_INSTR_CMD

Fls LUT Pad: FLS_LUT_PAD_1_PAD

Fls LUT Operand (0x0 -> 0xff): 0x0

Figure 4-17. Tresos Plugin snapshot for FlsInstructionOperandPair form.

4.7.22.5.1.1 FlsInstrOperPairIndex (FlsInstructionOperandPair)

Fls Instruction Operand Pair Index is an invariant index, used to order the Instr.Oper. entries and loop over them in the correct, configured order. Its value should be equal with the position of the configured pair inside the configured pair list (the same value as the shown index). Rationale: The generated .epc configuration might reorder the instr.oper. pairs (alphabetically), thus the index parameter changes, becoming out of sync with the real intended order (the values are not generated in the intended order, they are sorted).

Table 4-135. Attribute FlsInstrOperPairIndex (FlsInstructionOperandPair) detailed description

Property	Value
Label	Fls Instruction Operand Pair Index
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.7.22.5.1.2 FlsLUTInstruction (FlsInstructionOperandPair)

Table 4-136. Attribute FlsLUTInstruction (FlsInstructionOperandPair) detailed description

Property	Value
Label	Fls Instruction Operand Pair Index
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_LUT_INSTR_CMD

4.7.22.5.1.3 FlsLUTPad (FlsInstructionOperandPair)

Table 4-137. Attribute FlsLUTPad (FlsInstructionOperandPair) detailed description

Property	Value
Label	Fls Instruction Operand Pair Index
Type	ENUMERATION
Origin	NXP
Symbolic Name	false
Default	FLS_LUT_PAD_1_PAD

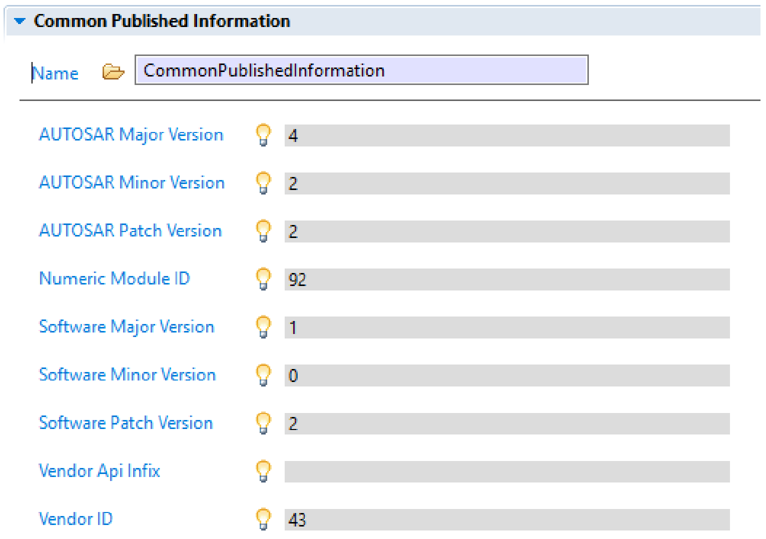
4.7.22.5.1.4 FlsLUTOperand (FlsInstructionOperandPair)

Table 4-138. Attribute FlsLUTOperand (FlsInstructionOperandPair) detailed description

Property	Value
Label	Fls LUT Operand
Type	INTEGER
Origin	NXP
Symbolic Name	false
Default	0

4.8 Form CommonPublishedInformation

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



Common Published Information

Name	CommonPublishedInformation
AUTOSAR Major Version	4
AUTOSAR Minor Version	2
AUTOSAR Patch Version	2
Numeric Module ID	92
Software Major Version	1
Software Minor Version	0
Software Patch Version	2
Vendor Api Infix	
Vendor ID	43

Figure 4-18. Tresos Plugin snapshot for CommonPublishedInformation form.

4.8.1 ArReleaseMajorVersion (CommonPublishedInformation)

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

Table 4-139. Attribute ArReleaseMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	4
Invalid	Range >=4 <=4

4.8.2 ArReleaseMinorVersion (CommonPublishedInformation)

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

Table 4-140. Attribute ArReleaseMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.8.3 ArReleaseRevisionVersion (CommonPublishedInformation)

Revision version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-141. Attribute ArReleaseRevisionVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Release Revision Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.8.4 ModuleId (CommonPublishedInformation)

Module ID of this module from Module List.

Table 4-142. Attribute ModuleId (CommonPublishedInformation) detailed description

Property	Value
Label	Module Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false

Table continues on the next page...

Table 4-142. Attribute ModuleId (CommonPublishedInformation) detailed description (continued)

Property	Value
Default	92
Invalid	Range >=92 <=92

4.8.5 SwMajorVersion (CommonPublishedInformation)

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

Table 4-143. Attribute SwMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

4.8.6 SwMinorVersion (CommonPublishedInformation)

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

Table 4-144. Attribute SwMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range

Table 4-144. Attribute SwMinorVersion (CommonPublishedInformation) detailed description

Property	Value
	>=0 <=0

4.8.7 SwPatchVersion (CommonPublishedInformation)

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

Table 4-145. Attribute SwPatchVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Patch Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.8.8 VendorApiInfix (CommonPublishedInformation)

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><Api name from SWS>. 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.

Table 4-146. Attribute VendorApiInfix (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Api Infix

Table continues on the next page...

Table 4-146. Attribute VendorApiInfix (CommonPublishedInformation) detailed description (continued)

Property	Value
Type	STRING_LABEL
Origin	Custom
Symbolic Name	false
Default	
Enable	false

4.8.9 VendorId (CommonPublishedInformation)

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

Table 4-147. Attribute VendorId (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	43
Invalid	Range >=43 <=43

Document Number UM2FLSASR4.2 Rev0002R1.0.2
Revision 1.0