

MCAL User Manual for CD package

32-bit TriCore™ AURIX™ TC3xx microcontroller

About this document

Scope and purpose

This User Manual is intended to enable users to integrate the Microcontroller Abstraction Layer (MCAL) software for the TriCore™ AURIX™ family of 32-bit microcontrollers.

This document describes responsibilities of integrator in-charge of integrating MCAL software with the basic software (BSW) stack. This document also provides detailed information on safety, configuration and functions along with examples of usage of significant features.

Intended audience

This document is intended for anyone using the CD package of the TC3xx MCAL software.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, sub-menus
<i>Italics</i>	Denotes variable(s) and reference(s)
Courier New	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, code snippets, file/folder names, directories, command line inputs
Hyperlink	Provides quick and easy access to cross-referenced topics/sections
>	Indicates that a cascading sub-menu opens when you choose a menu item

Glossary of terms

Table 2 Glossary

Term	Description
ADC	Analog-to-digital converter
AS	AUTOSAR
ATOM	ARU connected TOM (functional block of a microcontroller peripheral)
AUTOSAR	Automotive Open System Architecture
Bank	A Flash module contains separate banks. In DFlash, there are one or more banks. Banks support concurrent operations with some limitations due to common logic.
BL	Bit-line
BSW	Basic software
Burst write	The maximum amount of data that can be written with one command. The programming throughput is higher than for programming single pages. For DFlash, it is 4 pages (32 bytes).
CAN	Controller Area Network
CAN FD	CAN flexible data rate
CC	Communication controller

About this document

Table 2 **Glossary (continued)**

Term	Description
CCM	Cluster control module (functional block of GTM)
Channel	A channel is a software exchange medium for data that are defined with the same criteria: configuration parameters, number of data elements with same size and data pointers (source, destination) or location.
CHI	Communication host interface
CMU	Clock management unit (functional block of GTM)
CPU	Central processing unit
CRC	Cyclic redundancy check
DEM	Diagnostics event manager (MCAL module)
DER	Destination error
DET	Development error tracer
DF_EEPROM	Data Flash dedicated for EEPROM emulation
DFDBER	Double bit error
DFlash	Data Flash
DFSBER	Single bit error
DFTBER	Triple bit error
DIO	Digital input output
DMA	Direct memory access
DMU	Data memory unit
Driver	A driver is a BSW module located in the MCAL layer and contains the functionality to control and access an internal or an external device.
EB	Externally defined buffer for QSPI
ECC	Error correction code
EEPROM	Electrically erasable and programmable ROM (read-only memory)
ERAY	FlexRay IP module (hardware)
ERU	External request unit
ESR	External service request (microcontroller pin)
ETH	Ethernet
EVADC	Enhanced versatile analog-to-digital converter
EVER	Bit indicating erase verify error
ex_r	Exclusive read access
ex_rw	Exclusive read and write access
ex_w	Exclusive write access
Fast-Mode	Triggering the watchdog hardware has to be done with a short timeout period. This mode can be used during normal operations of the ECU. For example, the watchdog hardware is configured for the Windows mode (triggering the watchdog should happen within certain minimum/maximum boundaries within the timeout period) and a timeout period of 5 ms.
FCE	Flexible CRC engine

About this document

Table 2 **Glossary (continued)**

Term	Description
FEE	Flash EEPROM emulation
FEE sector	The DF_EEPROM area is logically divided into two FEE sectors for implementing the double sector algorithm. If the QS region exists then the part of DF_EEPROM is used for QS blocks and the remaining is used for double sector algorithm.
FIFO	First in first out
FLS	Flash
GC	Garbage collection
GETH	Gigabit Ethernet MAC
GTM	Generic timer module (hardware)
HOH	Hardware object (transmit/receive) handle
HRH	Hardware receive handle
HSCT	High-speed communications tunnel
HSPDM	High-speed pulse density modulation module
HTH	Hardware transmit handle
I/O	Input/Output
IB	Driver-defined internal buffer / channel
IFX	Infineon Technologies
Internal job	FEE driver internal management activities which are not explicitly triggered by the upper layer
ISR	Interrupt service routine
Job	A job is composed of one or several channels with the same chip select (is not released during the processing of job). A job is considered atomic and therefore cannot be interrupted by another job. A job has an assigned priority.
(Logical) block	Smallest writable / erasable unit as seen by the modules user. Consists of one or more virtual pages.
LPdu	Datalink layer protocol data unit
LPM	Low power mode
MAC	Media access control
MC-ISAR	Microcontroller Infineon Software Architecture
MCU	Microcontroller unit
MII	Media independent interface
Module	The element is composed of various software units called modules. Typically each software driver is referred to as module. More explicitly, it is also referred to as software module.
MRST	Master receive slave transmit
MTL	MAC transaction layer
MTSR	Master transmit slave receive
(Normal) write mode	In this mode, the maximum amount of data that can be written with one command is 8 byte (1 Page).
NVM	AUTOSAR NVRAM manager

About this document

Table 2 **Glossary (continued)**

Term	Description
NVRAM	Non-volatile RAM
NVRAM block	Management unit as seen by the NVRAM manager
OCU	Output compare unit
Off-Mode	Watchdog hardware is disabled / shut-down
OS	Operating system
Page	A page is an aligned group of data double words plus an ECC extension. It is the smallest unit that can be programmed. DFlash: 1 data double word (8 bytes) plus 22-bit ECC extension.
Peripheral	Hardware module used by a driver. A driver can use one or more peripherals.
PHY	Physical layer device (Ethernet transceiver)
Physical address	Address information in device-specific format (depending on the underlying Flash driver and device) that is used to access a logical block.
Physical sector	It is a combination of 256 logical sectors (in DFlash), which comprises of 1 MB memory.
PLL	Phase lock loop
Pn_xxxxx	Port n register (xxxxx is register name)
POC	Protocol operation control
PORST	Power-on reset
PVER	Bit indicating program verify error
QS	Quasi-static
QSPI	Queued serial peripheral interface
r	Read access
RGMII	Reduced gigabit media independent interface
RMII	Reduced media independent interface
rw	Read and write access
SchM	Scheduler manager (AUTOSAR module)
SCU	System control unit
Sequence	A sequence is a number of consecutive jobs to transmit, but it can be rescheduled between jobs using a priority mechanism. A sequence transmission is interruptible (by another sequence transmission) or not depending on a static configuration.
SER	Source error
Slow-Mode	Triggering the watchdog hardware can be done with a long timeout period. This mode can be used during system start-up / initialization phase. For example, the watchdog hardware is configured for toggle mode (no constraints on the point in time at which the triggering is done) and a timeout period of 10 ms.
SLSO	Programmable slave select outputs
SMU	Safety management unit
SPB	System peripheral bus
SRC	Service request control register
SRI	System resource interconnect

About this document

Table 2 Glossary (continued)

Term	Description
SRN	Service request node
STC	Set trigger condition API
STM	System timer (hardware)
TBU	Time base unit (functional block of GTM)
TECQED	Triple bit error correction and quad-bit error detection
TIM	Timer input module (functional block of a microcontroller peripheral)
TOM	Timer output module (functional block of a microcontroller peripheral)
Unconfigured block	Data block which is stored in the DFlash (DF_EEPROM), but is not contained in the currently active configuration
User job	User requested read/write/invalidate job
VariantLT	This variant allows a mix of pre-compile time, link-time configuration parameters. The intention of this variant is to optimize the parameters configuration for an object code delivery.
VariantPB	This variant allows a mix of pre-compile time, post-build time and link time configuration parameters. The intention of this variant is to optimize the parameters configuration for a re-loadable binary
VarinatPC	This variant allows only pre-compile configuration parameters. The intention of this variant is to optimize the parameters configuration for a source code delivery.
Virtual address	Consisting of 16-bit block number and 16-bit offset inside the logical block
Virtual page	May consist of one or several physical pages to ease handling of logical blocks and address calculation
w	Write access
WDG	Watchdog
WDGx	Watchdog timer for CPUx (x = 0-5 for TC39x, x=0-3 for TC38x, x= 0-2 for TC37x, x=0,1 for TC36x, x=0-2 for TC35x, x=0 for TC33x and x=0,1 for TC33x(ED/ADAS))
WDT	Watchdog timer (hardware)
WL/BL	Word-line/Bit-line
Wordline (WL)	An aligned group of bytes. In DFlash, 512 bytes are in the single-ended mode and 256 bytes are in the complement-sensing mode.
WUT	WakeUp timer

Reference documents

This User Manual should be read in conjunction with the following documents:

- AURIX™ TC3xx User Manual, V1.0.0, 2018-08, Infineon Technologies Munich AG
- AURIX™ TC38x Appendix to User Manual, V1.0.0, 2018-08, Infineon Technologies Munich AG
- AURIX™ TC39x-B Appendix to User Manual, V1.0.0, 2018-08, Infineon Technologies Munich AG
- TC39x_BB_Errata_Sheet, Rel1.0, 2018-07-12, Infineon Technologies Munich AG
- TC39x_BA_Errata_Sheet, Rel1.2, 2018-07-02, Infineon Technologies Munich AG
- TC38x_AB_Errata_Sheet, Rel1.0, 2018-07-12, Infineon Technologies Munich AG
- TC38x_AA_Errata_Sheet, Rel1.2, 2018-07-02, Infineon Technologies Munich AG
- AURIX TC3xx Safety Manual, v0.9, 2018-05-18, Infineon Technologies Munich AG

Table of contents
Table of contents

	About this document	1
	Table of contents	6
1	DMA driver	17
1.1	User information	17
1.1.1	Description	17
1.1.2	Hardware-software mapping	17
1.1.2.1	DMA: primary hardware peripheral	18
1.1.2.2	SCU: dependent hardware peripheral	19
1.1.2.3	CPU - Interrupt Router: dependent hardware peripheral	20
1.1.3	File structure	20
1.1.3.1	C file structure	20
1.1.3.2	Code generator plugin files	22
1.1.4	Integration hints	24
1.1.4.1	Stub modules	24
1.1.4.2	Multicore and Resource Manager	26
1.1.4.3	MCU support	27
1.1.4.4	Port support	27
1.1.4.5	DMA support	27
1.1.4.6	Interrupt connections	27
1.1.4.7	Example usage	28
1.1.5	Key architectural considerations	30
1.1.5.1	Usage of general purpose software request (GPSR)	30
1.2	Assumptions of Use (AoUs)	31
1.3	Reference information	33
1.3.1	Configuration interfaces	33
1.3.1.1	Container: CommonPublishedInformation	33
1.3.1.1.1	ArMajorVersion	33
1.3.1.1.2	ArMinorVersion	34
1.3.1.1.3	ArPatchVersion	34
1.3.1.1.4	ModuleId	35
1.3.1.1.5	Release	35
1.3.1.1.6	SwMajorVersion	36
1.3.1.1.7	SwMinorVersion	36
1.3.1.1.8	SwPatchVersion	36
1.3.1.1.9	VendorId	37
1.3.1.2	Container: Dma	37
1.3.1.3	Container: DmaChannelConfig	37
1.3.1.3.1	DmaChannelAssignedPartition	37
1.3.1.3.2	DmaChannelId	38

Table of contents

1.3.1.3.3	DmaChannelNotification	38
1.3.1.3.4	DmaChannelNumTransactionSet	39
1.3.1.3.5	DmaTcsInterruptTransactionLoss	39
1.3.1.4	Container: DmaChannelTransactionSet	40
1.3.1.4.1	DmaNextTcsIndex	40
1.3.1.4.2	DmaTcsAppendTimeStamp	41
1.3.1.4.3	DmaTcsAutoStartEnable	41
1.3.1.4.4	DmaTcsCircularBufferDestinationEnable	42
1.3.1.4.5	DmaTcsCircularBufferDestinationLength	42
1.3.1.4.6	DmaTcsCircularBufferSourceEnable	43
1.3.1.4.7	DmaTcsCircularBufferSourceLength	44
1.3.1.4.8	DmaTcsDaisyChaining	45
1.3.1.4.9	DmaTcsDataTransferInterrupt	45
1.3.1.4.10	DmaTcsDestinationAddress	46
1.3.1.4.11	DmaTcsDestinationAddressModificationFactor	46
1.3.1.4.12	DmaTcsDestinationAddressMovement	47
1.3.1.4.13	DmaTcsDoubleBuffer	48
1.3.1.4.14	DmaTcsHardwareTrigger	48
1.3.1.4.15	DmaTcsIndex	49
1.3.1.4.16	DmaTcsInterruptDataTransfer	49
1.3.1.4.17	DmaTcsInterruptDataTransferThreshold	50
1.3.1.4.18	DmaTcsInterruptDestAddressWrap	50
1.3.1.4.19	DmaTcsInterruptSourceAddressWrap	51
1.3.1.4.20	DmaTcsMoveLength	51
1.3.1.4.21	DmaTcsReferenceAddressCrc	52
1.3.1.4.22	DmaTcsReferenceDataCrc	53
1.3.1.4.23	DmaTcsShadowRegisterConfiguration	53
1.3.1.4.24	DmaTcsSourceAddress	54
1.3.1.4.25	DmaTcsSourceAddressModificationFactor	55
1.3.1.4.26	DmaTcsSourceAddressMovement	56
1.3.1.4.27	DmaTcsTransactionLength	56
1.3.1.4.28	DmaTcsTransferLength	57
1.3.1.4.29	DmaTcsTriggerFrequency	57
1.3.1.5	Container: DmaGeneral	58
1.3.1.5.1	DmaBufferSwitchApiConfiguration	58
1.3.1.5.2	DmaChDeInitApiConfiguration	58
1.3.1.5.3	DmaDataPendingApiConfiguration	59
1.3.1.5.4	DmaDevErrorDetect	59
1.3.1.5.5	DmaGetVersionInfoApiConfiguration	60
1.3.1.5.6	DmaInitCheckApi	61
1.3.1.5.7	DmaInitDeInitApiMode	61
1.3.1.5.8	DmaMaxTransactionSetPerChannel	62

Table of contents

1.3.1.5.9	DmaMultiCoreErrorDetect	62
1.3.1.5.10	DmaRuntimeApiMode	63
1.3.1.5.11	DmaSafetyEnable	63
1.3.1.5.12	DmaSuspendApiConfiguration	64
1.3.1.5.13	DmaTriggerApiConfiguration	64
1.3.1.6	Container: DmaMoveEngineConfig	65
1.3.1.6.1	DmaMEDestinationErrorInterrupt	65
1.3.1.6.2	DmaMELinkedListErrorInterrupt	65
1.3.1.6.3	DmaMESourceErrorInterrupt	66
1.3.1.7	Container: DmaResourcePartition	66
1.3.1.7.1	DmaPermittedBusMaster	67
1.3.1.7.2	DmaResourcePartitionBusMode	67
1.3.1.7.3	DmaResourcePartitionErrorNotifRoutine	68
1.3.2	Functions - Type definitions	68
1.3.2.1	Dma_ChannelNotificationFnPtrType	68
1.3.2.2	Dma_ChEventType	68
1.3.2.3	Dma_ConfigType	69
1.3.2.4	Dma_ConfigUpdateType	69
1.3.2.5	Dma_CrcType	70
1.3.2.6	Dma_ErrorNotificationFnPtrType	71
1.3.2.7	Dma_MoveEngineEventType	71
1.3.2.8	Dma_MoveEngineListType	71
1.3.3	Functions - APIs	72
1.3.3.1	Dma_Init	72
1.3.3.2	Dma_IsInitDone	73
1.3.3.3	Dma_ChInit	74
1.3.3.4	Dma_ChUpdate	75
1.3.3.5	Dma_ChDelnit	76
1.3.3.6	Dma_ChTransferFreeze	77
1.3.3.7	Dma_ChTransferResume	78
1.3.3.8	Dma_ChEnableHardwareTrigger	79
1.3.3.9	Dma_ChDisableHardwareTrigger	80
1.3.3.10	Dma_ChStartTransfer	81
1.3.3.11	Dma_ChStopTransfer	82
1.3.3.12	Dma_ChGetRemainingData	83
1.3.3.13	Dma_ChSwitchBuffer	84
1.3.3.14	Dma_ChIsStatusAsserted	85
1.3.3.15	Dma_ChStatusClear	86
1.3.3.16	Dma_ChInterruptEnable	87
1.3.3.17	Dma_ChInterruptDisable	88
1.3.3.18	Dma_GetVersionInfo	89
1.3.3.19	Dma_GetChannelEvent	90

Table of contents

1.3.3.20	Dma_GetMoveEngineEvent	91
1.3.3.21	Dma_MEStatusClear	92
1.3.3.22	Dma_InitCheck	93
1.3.3.23	Dma_GetCrcValue	94
1.3.3.24	Dma_GetCurrentTimeStamp	95
1.3.3.25	Dma_IsChannelInitDone	95
1.3.4	Notifications and callbacks	96
1.3.5	Scheduled functions	96
1.3.6	Interrupt service routines	96
1.3.6.1	Dma_ChInterruptHandler	97
1.3.6.2	Dma_MEInterruptDispatcher	98
1.3.6.3	Dma_MEInterruptHandler	98
1.3.7	Error codes classification	99
1.3.7.1	Development errors	99
1.3.7.2	Production errors	103
1.3.7.3	Safety errors	103
1.3.7.4	Runtime errors	103
1.3.8	Deviations and limitations	103
1.3.8.1	Deviations	103
1.3.8.2	Limitations	104
1.3.9	Unsupported hardware features	104
2	FlsLoader driver	105
2.1	User information	105
2.1.1	Description	105
2.1.2	Hardware-software mapping	105
2.1.2.1	SCU: dependent hardware peripheral	106
2.1.2.2	DMU: primary hardware peripheral	107
2.1.3	File structure	107
2.1.3.1	C file structure	107
2.1.3.2	Code generator plugin files	109
2.1.4	Integration hints	110
2.1.4.1	Stub modules	110
2.1.4.2	Multicore and Resource Manager	113
2.1.4.3	MCU support	113
2.1.4.4	Port support	114
2.1.4.5	DMA support	114
2.1.4.6	Interrupt connections	114
2.1.4.7	Example usage	115
2.1.5	Key architectural considerations	120
2.1.5.1	User mode support by the driver	120
2.2	Assumptions of Use (AoUs)	121

Table of contents

2.3	Reference information	122
2.3.1	Configuration interfaces	122
2.3.1.1	Container: CommonPublishedInformation	122
2.3.1.1.1	ArMajorVersion	122
2.3.1.1.2	ArMinorVersion	123
2.3.1.1.3	ArPatchVersion	123
2.3.1.1.4	ModuleId	124
2.3.1.1.5	Release	124
2.3.1.1.6	SwMajorVersion	125
2.3.1.1.7	SwMinorVersion	125
2.3.1.1.8	SwPatchVersion	125
2.3.1.1.9	VendorId	126
2.3.1.2	Container: FlsLoader	126
2.3.1.3	Container: FlsLoaderDFlash0ProtConfig	126
2.3.1.3.1	FlsLoaderDF0Prot	127
2.3.1.3.2	FlsLoaderDF0UcbPW0_0	127
2.3.1.3.3	FlsLoaderDF0UcbPW0_1	128
2.3.1.3.4	FlsLoaderDF0UcbPW1_0	128
2.3.1.3.5	FlsLoaderDF0UcbPW1_1	128
2.3.1.3.6	FlsLoaderDF0UcbPW2_0	129
2.3.1.3.7	FlsLoaderDF0UcbPW2_1	129
2.3.1.3.8	FlsLoaderDF0UcbPW3_0	130
2.3.1.3.9	FlsLoaderDF0UcbPW3_1	130
2.3.1.4	Container: FlsLoaderDFLASHConfig	131
2.3.1.5	Container: FlsLoaderGeneral	131
2.3.1.5.1	FlsLoaderCallOutFunction	131
2.3.1.5.2	FlsLoaderCallOutTime	132
2.3.1.5.3	FlsLoaderDevErrorDetect	132
2.3.1.5.4	FlsLoaderEnableLockCheck	133
2.3.1.6	Container: FlsLoaderOptionalApi	133
2.3.1.6.1	FlsLoaderDeInitApi	133
2.3.1.6.2	FlsLoaderLockUnlockApi	134
2.3.1.6.3	FlsLoaderVersionInfoApi	134
2.3.1.7	Container: FlsLoaderPF0Sector	135
2.3.1.7.1	FlsLoaderPFSectorWriteProtection	135
2.3.1.8	Container: FlsLoaderPF1Sector	136
2.3.1.8.1	FlsLoaderPFSectorWriteProtection	136
2.3.1.9	Container: FlsLoaderPF2Sector	136
2.3.1.9.1	FlsLoaderPFSectorWriteProtection	137
2.3.1.10	Container: FlsLoaderPF3Sector	137
2.3.1.10.1	FlsLoaderPFSectorWriteProtection	137
2.3.1.11	Container: FlsLoaderPF4Sector	138

Table of contents

2.3.1.11.1	FlsLoaderPFSectorWriteProtection	138
2.3.1.12	Container: FlsLoaderPF5Sector	138
2.3.1.12.1	FlsLoaderPFSectorWriteProtection	139
2.3.1.13	Container: FlsLoaderPFlash0ProtConfig	139
2.3.1.13.1	FlsLoaderPF0UcbPW0_0	139
2.3.1.13.2	FlsLoaderPF0UcbPW0_1	140
2.3.1.13.3	FlsLoaderPF0UcbPW1_0	140
2.3.1.13.4	FlsLoaderPF0UcbPW1_1	141
2.3.1.13.5	FlsLoaderPF0UcbPW2_0	141
2.3.1.13.6	FlsLoaderPF0UcbPW2_1	141
2.3.1.13.7	FlsLoaderPF0UcbPW3_0	142
2.3.1.13.8	FlsLoaderPF0UcbPW3_1	142
2.3.1.13.9	FlsLoaderPFlash0WriteProt	143
2.3.1.14	Container: FlsLoaderPFlash1ProtConfig	143
2.3.1.14.1	FlsLoaderPFlash1WriteProt	144
2.3.1.15	Container: FlsLoaderPFlash2ProtConfig	144
2.3.1.15.1	FlsLoaderPFlash2WriteProt	144
2.3.1.16	Container: FlsLoaderPFlash3ProtConfig	145
2.3.1.16.1	FlsLoaderPFlash3WriteProt	145
2.3.1.17	Container: FlsLoaderPFlash4ProtConfig	145
2.3.1.17.1	FlsLoaderPFlash4WriteProt	146
2.3.1.18	Container: FlsLoaderPFlash5ProtConfig	146
2.3.1.18.1	FlsLoaderPFlash5WriteProt	146
2.3.1.19	Container: FlsLoaderPFLASHConfig	147
2.3.2	Functions - Type definitions	147
2.3.2.1	FlsLoader_AddressType	147
2.3.2.2	FlsLoader_CallOutFunc	147
2.3.2.3	FlsLoader_ConfigType	148
2.3.2.4	FlsLoader_LengthType	148
2.3.2.5	FlsLoader_ReturnType	148
2.3.2.6	FlsLoader_ValueType	149
2.3.3	Functions - APIs	149
2.3.3.1	FlsLoader_DeInit	149
2.3.3.2	FlsLoader_Erase	150
2.3.3.3	FlsLoader_GetVersionInfo	151
2.3.3.4	FlsLoader_Init	152
2.3.3.5	FlsLoader_Lock	152
2.3.3.6	FlsLoader_UnLock	154
2.3.3.7	FlsLoader_Write	155
2.3.4	Notifications and callbacks	156
2.3.5	Scheduled functions	156
2.3.6	Interrupt service routines	156

Table of contents

2.3.7	Error codes classification	156
2.3.7.1	Development errors	156
2.3.7.2	Production errors	157
2.3.7.3	Safety errors	157
2.3.7.4	Runtime errors	157
2.3.8	Deviations and limitations	157
2.3.8.1	Deviations	157
2.3.8.2	Limitations	157
2.3.9	Unsupported hardware features	157
3	SMU driver	158
3.1	User information	158
3.1.1	Description	158
3.1.2	Hardware-software mapping	158
3.1.2.1	PMS: primary hardware peripheral	160
3.1.2.2	SCU: dependent hardware peripheral	160
3.1.2.3	PORT: dependent hardware peripheral	160
3.1.2.4	SMU: primary hardware peripheral	161
3.1.2.5	SRC: dependent hardware peripheral	162
3.1.3	File structure	162
3.1.3.1	C file structure	162
3.1.3.2	Code generator plugin files	164
3.1.4	Integration hints	165
3.1.4.1	Stub modules	166
3.1.4.2	Multicore and Resource Manager	168
3.1.4.3	MCU support	169
3.1.4.4	Port support	169
3.1.4.5	DMA support	169
3.1.4.6	Interrupt connections	169
3.1.4.7	Example usage	170
3.1.5	Key architectural considerations	175
3.1.5.1	Clearing alarm status during initialization	175
3.1.5.2	Initialization and de-initialization	175
3.1.5.3	Smu_core state machine transitions	175
3.1.5.4	Recovery timer handling	176
3.2	Assumptions of Use (AoUs)	177
3.3	Reference information	178
3.3.1	Configuration interfaces	178
3.3.1.1	Container: CommonPublishedInformation	178
3.3.1.1.1	ArMajorVersion	178
3.3.1.1.2	ArMinorVersion	179
3.3.1.1.3	ArPatchVersion	179

Table of contents

3.3.1.1.4	ModuleId	180
3.3.1.1.5	Release	180
3.3.1.1.6	SwMajorVersion	180
3.3.1.1.7	SwMinorVersion	181
3.3.1.1.8	SwPatchVersion	181
3.3.1.1.9	VendorId	182
3.3.1.2	Container: Smu	182
3.3.1.3	Container: SmuConfigSet	182
3.3.1.4	Container: SmuCoreAlarmBehavior	182
3.3.1.4.1	SmuCoreAlarmFSP	183
3.3.1.4.2	SmuCoreAlarmIntBeh	183
3.3.1.4.3	SmuCoreAlmBehaviourId	184
3.3.1.5	Container: SmuCoreAlarmGlobalConfig	185
3.3.1.5.1	SmuCoreCpu0ResetRequest	185
3.3.1.5.2	SmuCoreCpu1ResetRequest	185
3.3.1.5.3	SmuCoreCpu2ResetRequest	186
3.3.1.5.4	SmuCoreCpu3ResetRequest	186
3.3.1.5.5	SmuCoreCpu4ResetRequest	187
3.3.1.5.6	SmuCoreCpu5ResetRequest	187
3.3.1.5.7	SmuCoreCpuResetActivatePES	188
3.3.1.5.8	SmuCoreEnableFaultToRunState	189
3.3.1.5.9	SmuCoreIGCS0ActivatePES	189
3.3.1.5.10	SmuCoreIGCS1ActivatePES	190
3.3.1.5.11	SmuCoreIGCS2ActivatePES	190
3.3.1.5.12	SmuCoreInterruptSet0	191
3.3.1.5.13	SmuCoreInterruptSet1	192
3.3.1.5.14	SmuCoreInterruptSet2	192
3.3.1.5.15	SmuCoreNMIActivatePES	193
3.3.1.6	Container: SmuCoreAlarmGroup	194
3.3.1.6.1	SmuCoreAlmGrpId	194
3.3.1.7	Container: SmuCoreConfig	194
3.3.1.8	Container: SmuCoreFSPHandling	194
3.3.1.8.1	SmuCoreFSPPFaultStateDuration	195
3.3.1.8.2	SmuCoreFSPPrescaler1	195
3.3.1.8.3	SmuCoreFSPPrescaler2	196
3.3.1.8.4	SmuCoreFSPSignalingMode	197
3.3.1.8.5	SmuCorePESOnFSP	198
3.3.1.9	Container: SmuCoreRecoveryTimer	198
3.3.1.9.1	SmuCoreEnableRT0	198
3.3.1.9.2	SmuCoreEnableRT1	199
3.3.1.9.3	SmuCoreRTDuration	199
3.3.1.10	Container: SmuCoreRT0Alarm	200

Table of contents

3.3.1.10.1	SmuCoreRT0AlarmGroupId	200
3.3.1.10.2	SmuCoreRT0AlarmId	201
3.3.1.11	Container: SmuCoreRT1Alarm	202
3.3.1.11.1	SmuCoreRT1AlarmGroupId	202
3.3.1.11.2	SmuCoreRT1AlarmId	203
3.3.1.12	Container: SmuDemEventParameterRefsConf	204
3.3.1.12.1	SmuActivateFSPFailureNotification	204
3.3.1.12.2	SmuActivatePESFailureNotification	204
3.3.1.12.3	SmuActivateRunStateFailureNotification	205
3.3.1.12.4	SmuClearAlarmStatusFailureNotification	205
3.3.1.12.5	SmuCoreAliveFailureNotification	206
3.3.1.12.6	SmuRTStopFailureNotification	206
3.3.1.12.7	SmuReleaseFSPFailureNotification	207
3.3.1.12.8	SmuSetAlarmStatusFailureNotification	207
3.3.1.12.9	SmuSffFailureNotification	208
3.3.1.13	Container: SmuGeneral	208
3.3.1.13.1	SmuAGStatusTimeout	208
3.3.1.13.2	SmuCoreFSP0OutputEnable	209
3.3.1.13.3	SmuCoreFSP0PortEnable	209
3.3.1.13.4	SmuCoreFSP1OutputEnable	210
3.3.1.13.5	SmuCoreFSP1PortEnable	210
3.3.1.13.6	SmuCoreGlitchFilterSCU	211
3.3.1.13.7	SmuCoreGlitchFilterSTS	211
3.3.1.13.8	SmuDevErrorDetect	212
3.3.1.13.9	SmuInitCheckApi	212
3.3.1.13.10	SmuInitDeInitApiMode	213
3.3.1.13.11	SmuRuntimeApiMode	213
3.3.1.13.12	SmuSafetyEnable	214
3.3.1.13.13	SmuStdbbyEnable	214
3.3.1.13.14	SmuVersionInfoApi	215
3.3.1.14	Container: SmuStdbbyAlarmBehavior	215
3.3.1.14.1	SmuStdbbyAlarmFSP	216
3.3.1.14.2	SmuStdbbyAlmBehaviourId	216
3.3.1.15	Container: SmuStdbbyAlarmGlobalConfig	216
3.3.1.15.1	SmuStdbbyEnableFSP0	217
3.3.1.15.2	SmuStdbbyEnableFSP1	217
3.3.1.16	Container: SmuStdbbyAlarmGroup	218
3.3.1.16.1	SmuStdbbyAlmGrpId	218
3.3.1.17	Container: SmuStdbbyConfig	218
3.3.2	Functions - Type definitions	218
3.3.2.1	Smu_AlarmGroupId	218
3.3.2.2	Smu_AlarmIdType	219

Table of contents

3.3.2.3	Smu_ConfigType	221
3.3.2.4	Smu_CoreAlarmActionType	221
3.3.2.5	Smu_CoreCommandType	222
3.3.2.6	Smu_CoreStateType	223
3.3.2.7	Smu_EnableRunStateType	224
3.3.2.8	Smu_FSPActionType	225
3.3.2.9	Smu_SffTestResType	225
3.3.3	Functions - APIs	225
3.3.3.1	Smu_Init	225
3.3.3.2	Smu_DeInit	226
3.3.3.3	Smu_GetAlarmAction	227
3.3.3.4	Smu_SetAlarmAction	228
3.3.3.5	Smu_ClearAlarmStatus	230
3.3.3.6	Smu_GetAlarmStatus	231
3.3.3.7	Smu_SetAlarmStatus	232
3.3.3.8	Smu_GetAlarmDebugStatus	233
3.3.3.9	Smu_LockConfigRegs	234
3.3.3.10	Smu_ReleaseFSP	234
3.3.3.11	Smu_ActivateFSP	235
3.3.3.12	Smu_SetupErrorPin	236
3.3.3.13	Smu_ReleaseErrorPin	237
3.3.3.14	Smu_RTStop	238
3.3.3.15	Smu_GetRTMissedEvent	239
3.3.3.16	Smu_ActivatePES	240
3.3.3.17	Smu_RegisterMonitor	241
3.3.3.18	Smu_GetSmuState	242
3.3.3.19	Smu_ActivateRunState	243
3.3.3.20	Smu_GetVersionInfo	244
3.3.3.21	Smu_CoreAliveTest	245
3.3.3.22	Smu_InitCheck	246
3.3.3.23	Smu_GetAlarmExecutionStatus	247
3.3.3.24	Smu_ClearAlarmExecutionStatus	248
3.3.4	Notifications and callbacks	249
3.3.5	Scheduled functions	249
3.3.6	Interrupt service routines	249
3.3.7	Error codes classification	249
3.3.7.1	Development errors	249
3.3.7.2	Production errors	252
3.3.7.3	Safety errors	253
3.3.7.4	Runtime errors	253
3.3.8	Deviations and limitations	253
3.3.8.1	Deviations	253

Table of contents

3.3.8.2	Limitations	253
3.3.9	Unsupported hardware features	253
	Revision history	254
	Disclaimer	255

1 DMA driver

1.1 User information

1.1.1 Description

The DMA driver is responsible in providing the services and configuration options for performing the DMA operations using the AURIX™ DMA hardware module. Services for initialization, starting, stopping and updating the DMA channels are provided by the DMA driver. The driver is designed as a post-build variant and it would be possible to generate a HEX file specifically for a desired configuration.

1.1.2 Hardware-software mapping

This section describes the system view of the driver and peripherals administered by it.

DMA driver

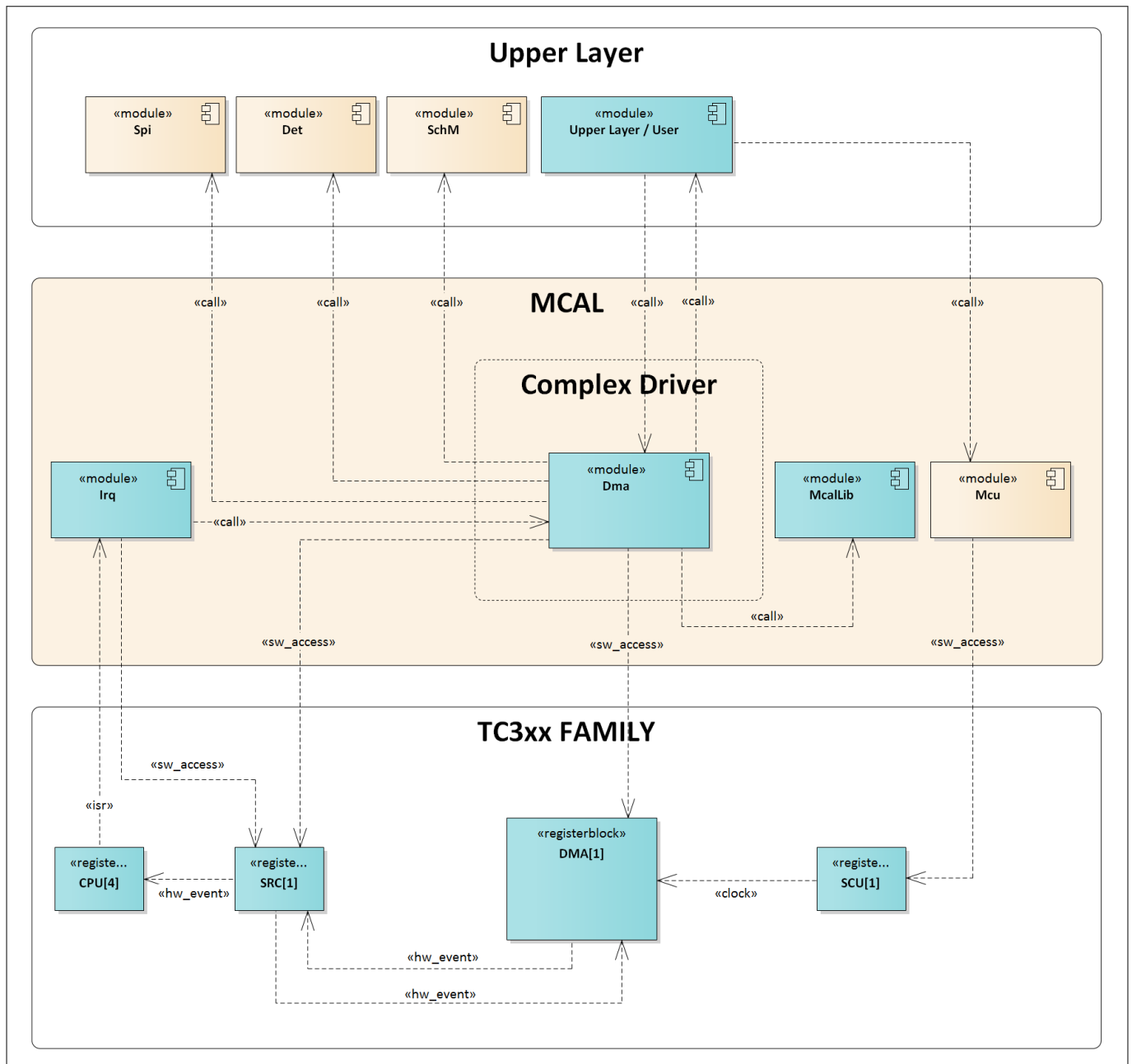


Figure 1 Mapping of hardware-software interfaces

1.1.2.1 DMA: primary hardware peripheral

Hardware functional features

The classic DMA functionality is fulfilled by two major functional blocks namely the DMA channel and the DMA move engine. A maximum of 128 channels and two move engines are available on the superset device.

The DMA channel holds the transfer characteristics such as the source address, destination address, transfer length, triggering criteria, etc. The move engine, which is a bus master, performs the physical movement of data from the source memory (SRAM, peripherals) to the destination memory (SRAM, peripherals), based on the transfer characteristics. The transfer characteristics can be configured using the configuration tools. In addition, the transaction control set can be updated through the driver APIs.

The key hardware functional features implemented in the DMA driver software are:

- Double buffering operations, source and destination side

DMA driver

- Circular buffer, source and destination
- Linked list, accumulated linked list and safe linked list
- Channel trigger by the software and hardware
- Daisy chaining
- CRC calculations for address and data
- Time stamping of the transactions
- Interrupts from channel and resource partitions

The hardware functional features which are not supported by the DMA driver software is/are:

- Pattern detection feature

Users of the hardware

The peripherals which require data transfer from one memory location to another can use the DMA driver.

Hardware diagnostic features

The hardware provides diagnostic alarms through SMU Alarm Groups for the following errors related to DMA:

- Alarm group 6:
 - Safety Mechanism: Safety flip-flop alarm-Safety uncorrectable error detected
- Alarm group 8:
 - Safety Mechanism: DMA SRI ECC Alarm-DMA SRI ECC error

Hardware events

The following hardware events are notified by flags and interrupts:

- Completion of a DMA transfer or transaction
- Wrap source and wrap destination buffer notifications
- Rap source
- Error events such as:
 - Transaction request lost or transfer request lost
 - Source error or destination error
 - DMARAM integrity error
 - Linked list load error interrupt
 - Safe linked list DMA address checksum error interrupt

In addition, the DMA being a service provider can take up the service requests from the other interrupt sources as well.

1.1.2.2 SCU: dependent hardware peripheral

Hardware functional features

The DMA driver depends upon the SCU for clock.

Users of the hardware

The SCU module supplies clock to all the peripherals. The configuration of the clock tree is performed by the MCU driver.

Hardware diagnostic features

None.

Hardware events

None.

DMA driver**1.1.2.3 CPU - Interrupt Router: dependent hardware peripheral****Hardware functional features**

The DMA driver depends upon the interrupt router (IR) for the following purposes. The IR is required to recognize an event flagged by a peripheral and route it to the DMA controller as a transfer request. Upon completion of a transaction, the DMA channel flags an event to the IR and the IR is required to translate the DMA channel event into an interrupt to the CPU.

For the move engine errors, the error events would get directed to one core. In this situation, the DMA would configure and use the general purpose service requests to route the interrupt to the respective core where the error has occurred.

Users of the hardware

The IR is administered by the IRQ driver. However, the SRC registers for the GPSR (General Purpose Service Request) are accessed by the DMA module to route the error interrupt requests to the appropriate core in a multicore environment.

Hardware diagnostic features

Not applicable.

Hardware events

Not applicable.

1.1.3 File structure**1.1.3.1 C file structure**

This section provides details on the C files of the DMA driver.

DMA driver

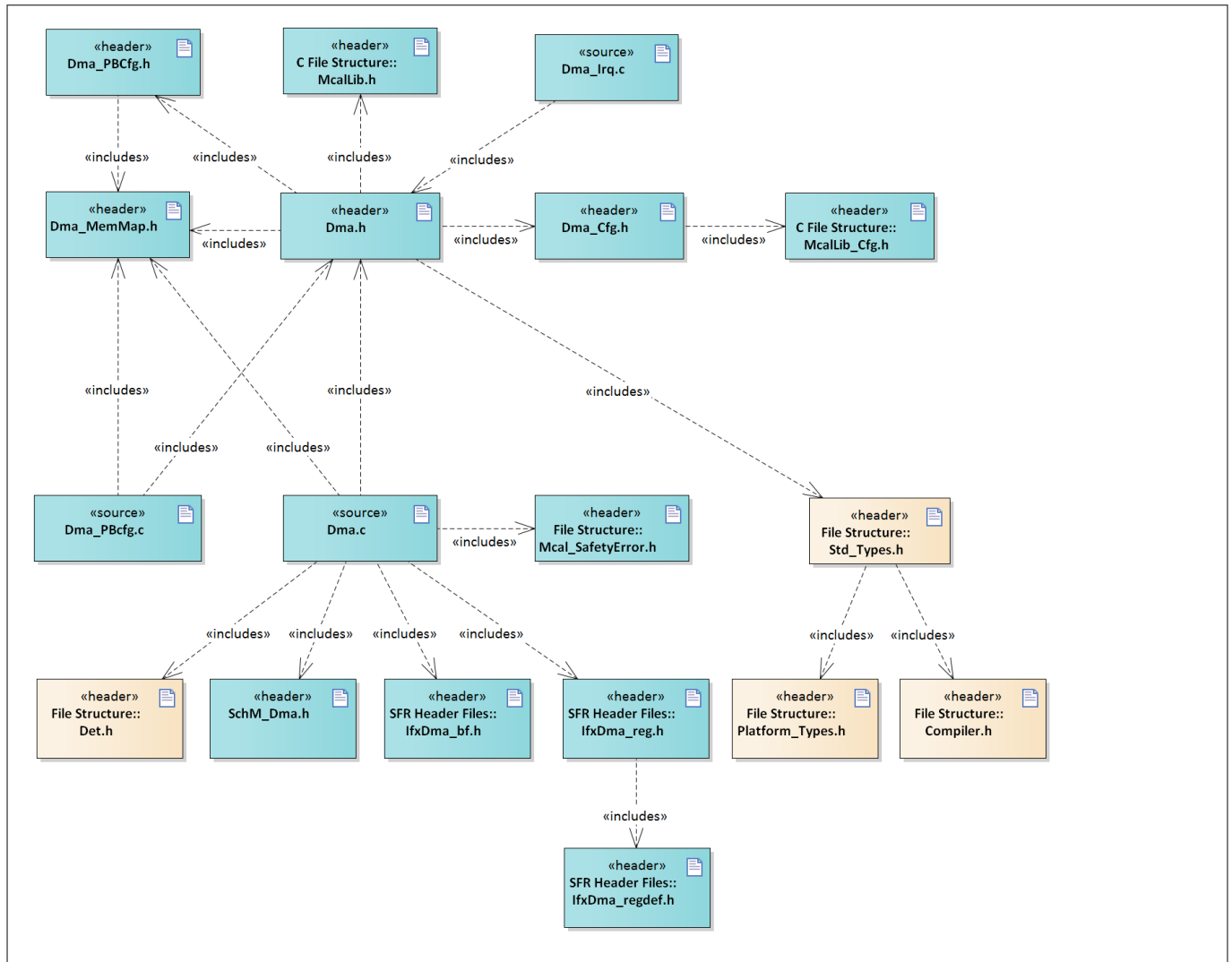


Figure 2 C file structure

Table 3 C file structure

File name	Description
Dma.c	File (static) containing implementation of APIs
Dma.h	Header file (static) defining prototypes of data structures and APIs
Dma_Cfg.h	Header file (generated) containing constants and pre-processor macros
Dma_PBcfg.c	File (generated) containing objects to data structures
Dma_Irq.c	Interrupt Handler file for DMA
Dma_MemMap.h	File containing the memory section definitions used by the DMA driver
SchM_Dma.h	Schedule manager header file for critical section management
McalLib.h	Header file (Static) defining prototypes of data structures and APIs of end-init and delay services and included by McalLib.c
IfxDma_reg.h	SFR header file for Dma
Det.h	Provides the exported interfaces of Development Error Tracer

DMA driver**Table 3** **C file structure (continued)**

File name	Description
Std_Types.h	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.
Compiler.h	Provides abstraction from compiler specific keywords
Platform_Types.h	Platform specific type declaration file as defined by AUTOSAR
IfxDma_regdef.h	SFR header file for Dma
IfxDma_bf.h	SFR header file for Dma
Dma_PBCfg.h	Post-Build header file for DMA module
McalLib_Cfg.h	Header file (Dynamic) providing information on number of cores, DSPR, PSPR (start and end addresses) and system and backup clock information
Mcal_SafetyError.h	Header file containing the prototype for the API for reporting safety-related errors

1.1.3.2 **Code generator plugin files**

This section provides details on the code generator plugin files of the DMA driver.

DMA driver

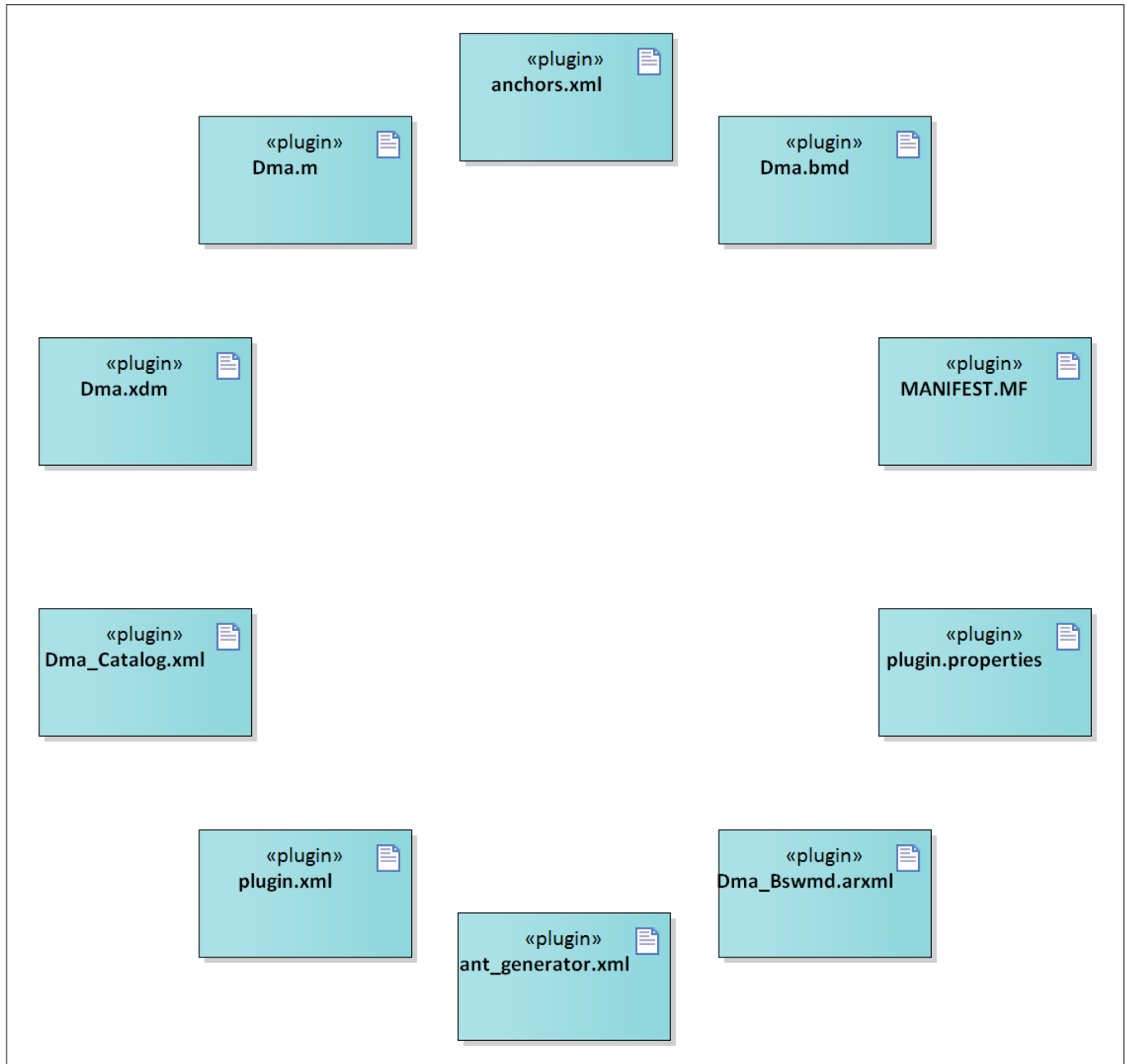


Figure 3 Code generator plugin files

Table 4 Code generator plugin files

File name	Description
anchors.xml	Tresos anchors support file for the DMA driver
Dma.xdm	Tresos format XML data model schema file
Dma.bmd	AUTOSAR format XML data model schema file (for each device)
Dma_Catalog.xml	AUTOSAR format catalog file
Dma_Bswmd.arxml	AUTOSAR format module description file
MANIFEST.MF	Tresos plugin support file containing the metadata for DMA driver
plugin.xml	Tresos plugin support file for the DMA driver

DMA driver**Table 4** **Code generator plugin files (continued)**

File name	Description
<code>plugin.properties</code>	Tresos plugin support file for the DMA driver
<code>Dma.m</code>	Code template macro file for DMA driver
<code>ant_generator.xml</code>	Tresos support file to generate and rename multiple post-build configurations when using variation point

1.1.4 **Integration hints**

This section lists the key points that an integrator or user of the DMA driver must consider.

In general, the APIs of DMA driver may be invoked from several CPU cores in parallel with some restrictions, which are also described in this section.

1.1.4.1 **Stub modules**

This section lists the modules required to integrate the DMA driver

EcuM

EcuM module is not provided as a standard MCAL module but rather provided as a stub module. The user of DMA driver may use APIs of EcuM to initialize the driver. The initialization of the DMA driver should be invoked from each CPU core which intends to use the services of the DMA driver. Initialization from logical master core must be completed prior to invoking of initialization from other slave cores. The slave cores can execute initialization in parallel. In case all the DMA resources are allocated to master core, invoking the initialization from master core alone is sufficient.

Memory mapping

The variables, constants and configuration data of DMA driver is encapsulated in specific memory section macros defined in the `Dma_MemMap.h` file. The naming of the memory sections is AUTOSAR compliant. The

DMA driver

user must place appropriate compiler pragmas to ensure the memory mapped macros are located to the correct memory space. Sample memory sections are as follows:

```

/**** GLOBAL RAM DATA -- NON-CACHED LMU ****/
#if defined DMA_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED
/*****User pragmas here for Non-cached LMU*****/
#undef DMA_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED
#undef MEMMAP_ERROR
#elif defined DMA_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED
#ifdef _TASKING_C_TRICORE_
/*****User pragmas here for Non-cached LMU*****/
#undef DMA_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED

/**** CORE[x] CONFIG DATA -- PF[x] ****/ /*[x]=0..5*/
#elif defined DMA_START_SEC_CONFIG_DATA_ASIL_B_CORE[x]_UNSPECIFIED
/*****User pragmas here for PF[x]*****/
#undef DMA_START_SEC_CONFIG_DATA_ASIL_B_CORE[x]_UNSPECIFIED
#undef MEMMAP_ERROR
#elif defined DMA_STOP_SEC_CONFIG_DATA_ASIL_B_CORE[x]_UNSPECIFIED
/*****User pragmas here for PF[x]*****/
#undef DMA_STOP_SEC_CONFIG_DATA_ASIL_B_CORE[x]_UNSPECIFIED

/**** CORE[x] CONFIG DATA WITH ALIGNMENT -- PF[x] ****/ /*[x]=0..5*/
#elif defined DMA_START_SEC_CONFIG_DATA_ASIL_B_CORE[x]_256
/*****User pragmas here for PF[x], with 256 bit alignment*****/
#undef DMA_START_SEC_CONFIG_DATA_ASIL_B_CORE[x]_256
#undef MEMMAP_ERROR
#elif defined DMA_STOP_SEC_CONFIG_DATA_ASIL_B_CORE[x]_256
/*****User pragmas here for PF[x], with 256 bit alignment*****/
#undef DMA_STOP_SEC_CONFIG_DATA_ASIL_B_CORE[x]_256

/**** CODE -- PF[x] ****/
#elif defined DMA_START_SEC_CODE_ASIL_B_GLOBAL
/*****User pragmas here for PF[x]*****/
#undef DMA_START_SEC_CODE_ASIL_B_GLOBAL
#undef MEMMAP_ERROR
#elif defined DMA_STOP_SEC_CODE_ASIL_B_GLOBAL
/*****User pragmas here for PF[x]*****/
#undef DMA_STOP_SEC_CODE_ASIL_B_GLOBAL
#endif

```

DET

The complete DET module is not provided as a standard MCAL module but rather provided as a stub module. The user of the DMA driver must process all the errors reported to the DET module through the `Det_ReportError()` API. The API `Det_ReportError()` is provided in the non-productive `Det.c` and `Det.h` files.

SchM

DMA driver

SchM module is not provided as a standard MCAL module but rather provided as a stub module. The DMA driver uses the exclusive area defined in `SchM_Dma.h` to protect SFRs and variables from concurrent accesses from different threads. The user should implement the SchM functions defined by the DMA driver as suspend or resume of interrupts for the CPU on which the API is invoked. A sample implementation of the SchM functions is depicted as follows:

```

/**** Sample implementation of SchM_Dma.c ****/
#include "Os.h"
void SchM_Enter_Dma_ChannelConfigUpdate(void)
{
    /* Start of Critical Section */
    DISABLE(); /* Disable CPU core interrupt */
}
void SchM_Exit_Dma_ChannelConfigUpdate(void)
{
    /* End of Critical Section */
    ENABLE(); /* Enable CPU core interrupt */
}

```

Safety error

All safety-related errors are reported through `Mcal_ReportSafetyError()`. Only reporting of safety error is the responsibility of the driver, the handling of the reported error must be done by the user.

Note: All DET error codes are also reported as safety errors.

Note: The `Mcal_ReportSafetyError()` API is provided in the non-productive `Mcal_SafetyError.c` and `Mcal_SafetyError.h` files.

Notifications and callbacks

The driver does not implement any notifications. However, it does report the interrupts raised by the DMA channels and error interrupts from the DMA Resource Partitions through notification functions. These notification functions can be configured by the user for each DMA channel and DMA Resource Partitions respectively.

Operating system

Operating System or application must ensure that correct type of service and interrupt priority is configured in the Service Request (SR) register. Enabling and disabling of interrupts must also be managed by the OS or application.

1.1.4.2 Multicore and Resource Manager

The DMA driver supports execution of its APIs in parallel from all the CPU cores. The user has to allocate resources of DMA to CPU cores at pre-compile time using the Resource Manager module. The following are the key points to be considered with respect to multicore in the driver:

- The DMA channels can be allocated to the any of the CPU cores. For example, DMAchannel-0 and DMAchannel-55 can be allocated to core-1, DMAchannel-21 and DMAchannel-35 can be allocated to core-2 and DMAchannel-10 and DMAchannel-101 can be allocated to core-0.
- Any configured DMA channel, if unallocated in the Resource Manager, would be allocated to the master core.

DMA driver

- It must be ensured that the DMA channel number passed as a parameter while invoking a DMA API, belong to the same core on which the API is invoked.
- DETs and/or safety errors will be raised in case APIs are invoked with mismatch of core and DMA channel
- Channel interrupts raised by a DMA channel must be serviced by the CPU core to which the DMA channel has been allocated. However, the Resource Partition interrupts must be serviced by the master CPU core.
- The locating of constants, variables and configuration data to correct memory space should be done by the user. Memory sections are marked GLOBAL (common to all cores) and CORE[x] (specific to a CPU core). The following should be considered by the user to ensure better performance of the driver:

Code Section

The executable code of Dma driver is placed under single MemMap section. It can be relocated to any PFlash.

Data Section

The RAM variable memory sections marked as specific to core, should be re-located to the DSPR of the same core. Those marked as global should be relocated to the non-cached LMU region.

Configuration data and constants

The configuration data section sections marked as specific to core should be re-located to the PFLASH of the same core. Those marked as global should be relocated to the PFlash of the master core.

Note: Relocating code, data and constants to a distant memory space would impact execution timings.

Note: If the driver operates from a single (master) core, all the sections may be relocated to the PFlash or DSPR of the same CPU core.

1.1.4.3 MCU support

The MCU initialization should be performed before using the DMA APIs to ensure the clock supply to the DMA hardware.

1.1.4.4 Port support

Not applicable for the driver.

1.1.4.5 DMA support

Not applicable for the driver.

1.1.4.6 Interrupt connections

The DMA driver supports the following interrupts raised by the DMA hardware:

Channel interrupts

Each DMA channel has a dedicated interrupt. These interrupts are raised signifying a channel transfer completion, source wrap or a destination wrap. The channel interrupts may be enabled in the driver configuration. The integrator must ensure that the SR registers of these interrupts are configured appropriately.

Resource partition interrupts

Each Resource Partition has a dedicated interrupt line. These are the error interrupts possible during the DMA operations. These interrupts may be enabled in the driver configuration. In the multicore environment, the error interrupts are routed to the appropriate cores using a GPSR0 interrupt. The integrator must ensure that the SR registers of all these interrupts are configured appropriately.

DMA driver

1.1.4.7 Example usage

The following are some of the key use cases of the DMA driver.

Note: Refer to the comments in the code snippets for additional information.

Initialization of DMA driver

The driver can be initialized by invoking `Dma_Init()` API. During the invocation, pass the configuration structure pointer as input parameter.

Note: The driver gets initialized for the core (along with the channels allocated to the core) in which the API is invoked. The initialization has to be done for each core where the driver is going to be used.

```
/* Include the header files */
#include "Dma.h"
#include "Mcu.h"

/* MCU Initialization */
Mcu_Init(&Mcu_Config);
Mcu_InitClock(0U);
while(Mcu_GetPllStatus() == MCU_PLL_LOCKED);
Mcu_DistributePllClock();

/* Initialize the DMA driver*/
Dma_Init(&Dma_Config);
/* Driver is initialized for the current core */
```

De-initializing DMA channels

A DMA channel can be selectively de-initialized if needed. The `Dma_ChDeInit()` API can be used for this.

```
/*
 * Configuration values mandatory for below code snippet:-
 * DmaDeinitApiConfiguration = TRUE
 * -> This is needed to invoke the channel de-init API.
 */

/* De-initialize the DMA channel 12 */
Dma_ChDeInit(12);
```

Starting a DMA channel

After the initialization is complete, the desired DMA channel can be started using the `Dma_ChStartTransfer()` API. Provide the channel number as the parameter for this API.

```
/* Start the DMA channel 5 */
Dma_ChStartTransfer(5);
```

Enabling the hardware trigger for the DMA channel

DMA driver

To enable the DMA channels to receive the interrupts from different sources, the `Dma_ChEnableHardwareTrigger()` API can be used.

```
/*
 * Configuration values mandatory for below code snippet:-
 * DmaTriggerApiConfiguration = TRUE
 * -> This is needed to invoke the hardware trigger APIs.
 */

/* Enable the hardware triggers to the DMA channel 10 */
Dma_ChEnableHardwareTrigger(10);
```

Updating the channel settings during runtime

The driver provides the `Dma_ChUpdate()` API to update following configuration items at run time – Source Address, Destination Address, Shadow Address, Address CRC, Data CRC and DMA channel configuration registers.

```
/* Perform an update of the source and destination addresses
 * of channel 10 using the Dma_ChUpdate API. */

/* The source and destination locations */
uint32 ChSourceAddress;
uint32 ChDestinationAddress;

/* Here, 'DmaChannelConfigVar' is a structure of type Dma_ConfigUpdateType,
 * holding the channel configuration to be used for updating the channel */
Dma_ConfigUpdateType DmaChannelConfigVar;

/* Step 1: Set the source and destination address pointers.
 * Step 2: Set the corresponding bit fields indicating that an update is needed
 * for the source and destination addresses */
DmaChannelConfigVar.SourceAddress = &ChSourceAddress;
DmaChannelConfigVar.DestAddress = &ChDestinationAddress;
DmaChannelConfigVar.UpdateAddressCrc = 0;
DmaChannelConfigVar.UpdateConfig = 0;
DmaChannelConfigVar.UpdateControlAdicr = 0;
DmaChannelConfigVar.UpdateControlChcsr = 0;
DmaChannelConfigVar.UpdateDataCrc = 0;
DmaChannelConfigVar.UpdateDestAddress = 1;
DmaChannelConfigVar.UpdateShadowConfig = 0;
DmaChannelConfigVar.UpdateSourceAddress = 1;

/* Perform the update using the driver API */
Dma_ChUpdate(10, &DmaChannelConfigVar, NULL_PTR);
```

1.1.5 Key architectural considerations

1.1.5.1 Usage of general purpose software request (GPSR)

In the multicore environment, when a DMA error occurs, the error interrupt can get serviced only by any one of the cores. This can create problems since the cause of the error might be a transaction, which was triggered from another core. To circumvent this problem, it is decided to use the general purpose software requests (GPSR). The error interrupts shall be routed to the appropriate core using GPSRs. The error handler, which services the error interrupt, finds the core to which the error prone channel has been allocated, and triggers a GPSR to that core. The GPSR, in turn, calls the notification function, which is configured by the user.

DMA driver

1.2 Assumptions of Use (AoUs)

The AoUs for the driver are as follows:

- **DMA address CRC and data CRC verification**

The user of DMA driver shall compare the expected CRC with the CRC calculated by the DMA driver, to find any mismatches.

- **DMA timestamp feature**

If the timestamp feature of the DMA is used, the user shall compare the timestamp with expected timestamp and take appropriate measures. The user should also allocate a 4byte space (at the next 32bit aligned address, after the last byte copied at destination) for the DMA to store the timestamp.

- **Double buffering - software switch usage**

When using double buffering in software switch mode, a TRL event would be raised if the transaction completes before the DMA receives a software switch. In a typical use case, the user can set the RROAT bit as 0, so that each transfer would require a trigger.

[cover parentID DMA={946532FE-2B5D-438d-9A1B-01B98321ECA8}]

- **Monitoring the lost transactions of DMA**

The user of the DMA driver is recommended to check for any lost DMA transaction, using the interrupt handlers. If there are any requests lost, appropriate actions should be taken by the user.

- **Non-receipt of the interrupts to DMA**

If the DMA channel is configured to receive the interrupts from the hardware as triggers for the transfer/transaction, the user has to ensure the following: 1. The appropriate interrupt settings should be set to route the interrupts to DMA. 2. The user should also configure and monitor the channel transfer/transaction complete interrupts. These interrupts should be monitored with a timeout mechanism to ensure the correct reception of the hardware interrupts. The user can take appropriate error handling mechanism in the event of a timeout.

[cover parentID DMA={F8377F12-31F4-4e39-B3C8-ACC568522748}]

- **Protection of DMA data and registers**

The user has to ensure that adequate memory protections are made available, so that the global data and registers of DMA are not overwritten by any other software running in the system.

[cover parentID DMA={34CC22B0-38D9-436c-B0E3-2438A6DCAE7C}]

- **Usage of APIs for enabling/disabling hardware triggers**

The enable/disable hardware trigger APIs should not be called for a channel which is configured as 'no hardware trigger'

[cover parentID DMA={E7EB27D9-C3BB-4652-910B-4FE393BF8326}]

- **Usage of DMA interrupts**

It is recommended that the user enable the interrupts and configure the respective handlers also. This would enable the user to detect the events and errors occurring during the DMA transactions and to take the appropriate measures for handling them. Note 1: Polling should not be done for detecting the channel events or error events. This is not supported by the driver. Note 2: The user has to configure and invoke the channel interrupts for the used channels (for detecting channel events), the resource partition interrupts and the GPSR00 interrupt (for detecting the errors). The user has to ensure that the DMA interrupt handler function 'Dma_MEInterruptDispatcher()' is called from the GPSR00 interrupt handler.

[cover parentID DMA={D84CE84B-2D57-40bb-8DC0-C9EB17817712}]

- **Usage of Dma_ChUpdate API**

Dma_ChUpdate API should not be called by the user with invalid TCS contents (reserved bit fields selected, unused features selected, read-only or status bits set). The API also should not be used to update the TCS of an active/pending/halted(frozen) channel.

[cover parentID DMA={9960DEFF-E8A7-4a6d-A830-45F3BD872C96}]

DMA driver

- **Usage of Dma_IsInitDone API**

Dma InitDone API only indicates if the module initialization was invoked or not. In case channels were stopped, the user shall not rely or use this API to derive the initialization status of the channel

[cover parentID DMA={FC698CE5-041C-4033-98BB-9D89E5202455}]

- **User/Supervisor modes context usage**

The user has to ensure that the DMA APIs are invoked in the same context for which it is configured to be used. (For e.g. if the DMA is configured to be used in the supervisor mode, the user module should invoke the DMA APIs in the supervisor mode.)

- **Pattern detection not supported**

The pattern detection feature will not be supported by the DMA driver

[cover parentID DMA={76A849BA-6602-4f03-905B-DE829AA3954A}]

- **Config check**

The user has to ensure that the generated configuration structures are correct against the intended GUI configurations.

[cover parentID DMA={C9DFDA58-CB2E-4a50-9AF1-98102D9A3760}]

- **Configuration of GPSR interrupts**

The GPSR00 interrupt is used by the DMA to report the error events of DMA. The user has to ensure that the interrupt handler 'Dma_MEInterruptDispatcher' is invoked from the GPSR00 interrupt handler. This is needed for the error events to be notified.

- **Usage and configuration of interrupts**

When the DMA channels are allocated by the user to any particular core, the user has to ensure that the TOS (in the IRQ configuration) of the corresponding channel interrupts is also assigned to the same core.

- **Usage of cache and DMA**

If any module requires the DMA to transfer the data to and from the memory, the related memory should be accessed from the CPU without using cache. If cache is used, the consistency of the data cannot be ensured. If the linked list feature is being used, the transaction control sets (TCS) should also be located in the non-cached memory.

- **Usage of Init check feature**

The user has to ensure that the init check API of the driver is invoked from each core where the DMA initialization is performed. The init check API invocation has to happen after the initialization is performed.

[cover parentID DMA={5407022A-8D01-4646-A8E6-0D4716A6DDD0}]

- **Usage of TRL feature**

If the TRL event has to be reported for a particular channel, the user has to ensure that the ETRL bit is enabled for the channel, by setting the configuration parameter 'DmaTcsInterruptTransactionLoss' in the configuration tool.

[cover parentID DMA={6880790B-2476-4a53-AA66-9CB3565BE1AB}]

- **Software access protection feature usage**

The user has to configure and use the hardware functionality of bus access protection (DMA ACCEN registers) to ensure Freedom From Interference of software from different bus masters. The DMA driver provides means to configure these registers via Tresos, which can be used by the user.

DMA driver

1.3 Reference information

1.3.1 Configuration interfaces

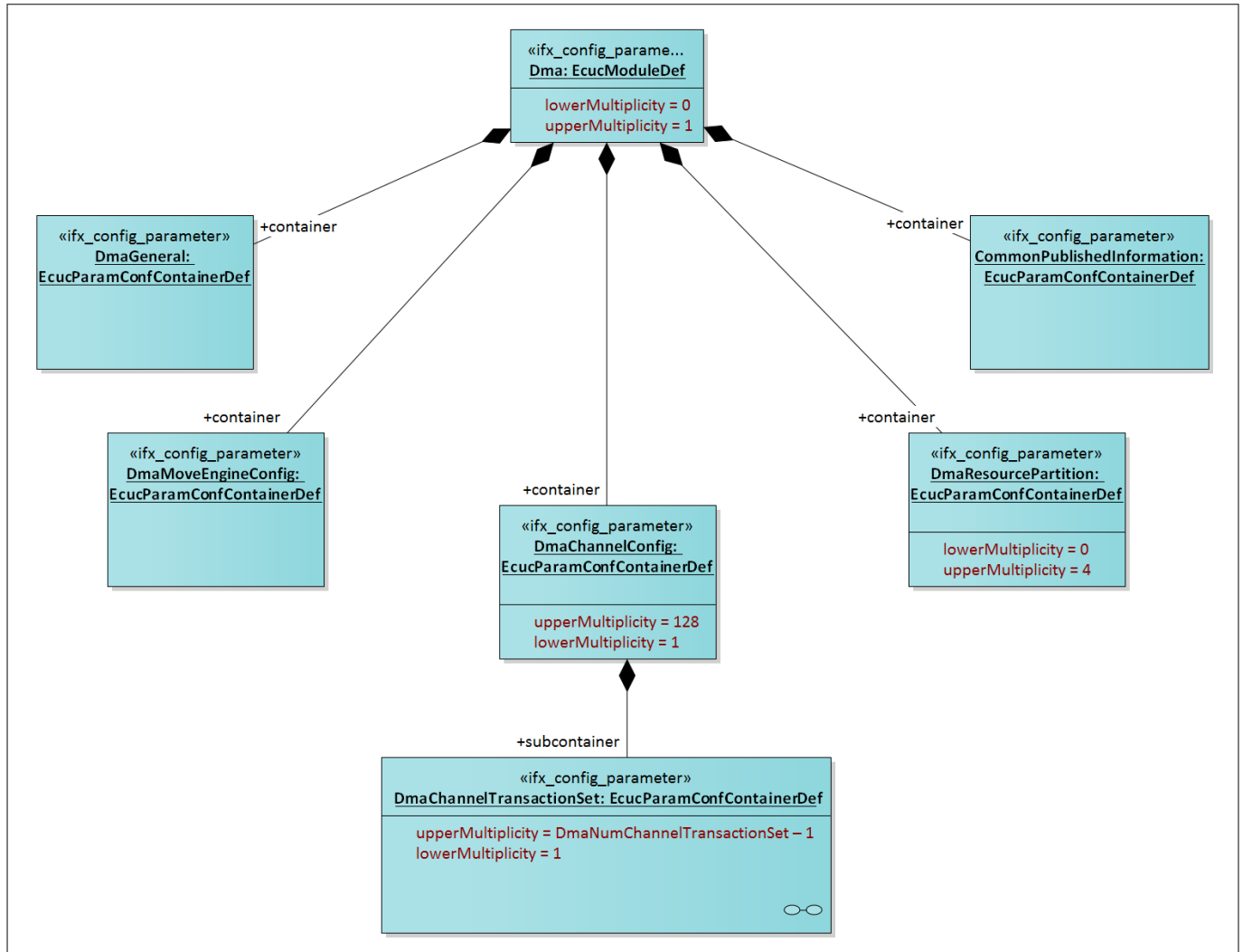


Figure 4 Container hierarchy along with their configuration parameters

1.3.1.1 Container: CommonPublishedInformation

This is the general configuration of DMA driver module common container, aggregated by all modules. It contains published information about vendor and versions.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.1.1 ArMajorVersion

Table 5 Specification for ArMajorVersion

Name	ArMajorVersion
Description	This parameter provides the major version of the AUTOSAR specification.

DMA driver

Table 5 Specification for ArMajorVersion (continued)

Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.2 ArMinorVersion

Table 6 Specification for ArMinorVersion

Name	ArMinorVersion		
Description	This parameter provides the minor version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.3 ArPatchVersion

Table 7 Specification for ArPatchVersion

Name	ArPatchVersion		
Description	This parameter provides the patch version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	Post-build variant multiplicity	-

DMA driver
Table 7 Specification for ArPatchVersion (continued)

Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.4 ModuleId
Table 8 Specification for ModuleId

Name	ModuleId		
Description	This parameter provides the module ID of DMA		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	255		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.5 Release
Table 9 Specification for Release

Name	Release		
Description	This parameter indicates the TC3xx device derivative used for the implementation.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	String		
Default value	As per hardware derivative		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

DMA driver

1.3.1.1.6 SwMajorVersion

Table 10 Specification for SwMajorVersion

Name	SwMajorVersion		
Description	This parameter provides the major version of the software.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver version		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.7 SwMinorVersion

Table 11 Specification for SwMinorVersion

Name	SwMinorVersion		
Description	This parameter provides the minor version of the software.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver version		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.8 SwPatchVersion

Table 12 Specification for SwPatchVersion

Name	SwPatchVersion		
Description	This parameter provides the patch version of the software.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		

DMA driver
Table 12 Specification for SwPatchVersion (continued)

Default value	As per Driver version		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.9 VendorId
Table 13 Specification for VendorId

Name	VendorId		
Description	This parameter provides the vendor ID		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2 Container: Dma

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: -

1.3.1.3 Container: DmaChannelConfig

This container holds the channel wise configuration data for the DMA driver.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.3.1 DmaChannelAssignedPartition
Table 14 Specification for DmaChannelAssignedPartition

Name	DmaChannelAssignedPartition
-------------	-----------------------------

DMA driver

Table 14 Specification for DmaChannelAssignedPartition (continued)

Description	This parameter determines the hardware partition which this channel is allocated to. Note: The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 3		
Default value	0		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.2 DmaChannelId

Table 15 Specification for DmaChannelId

Name	DmaChannelId		
Description	This parameter determines the physical channel number Note: The default value is an incremental value starting from zero, to make it unique automatically.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 127		
Default value	Incremental value starting with 0 and unique among all the channels		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.3 DmaChannelNotification

Table 16 Specification for DmaChannelNotification

Name	DmaChannelNotification		
Description	This parameter is used to register an application notification function which is invoked during interrupt handling. Note: Since the name of the function is user configurable, the default value is kept as NULL.		

DMA driver
Table 16 Specification for DmaChannelNotification (continued)

Multiplicity	1..1	Type	Dma_ChannelNotificationFnPtrType
Range			
Default value	NULL		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	DmaTcsInterruptSourceAddressWrap, DmaTcsInterruptDestAddressWrap, DmaTcsInterruptTransactionLoss		

1.3.1.3.4 DmaChannelNumTransactionSet
Table 17 Specification for DmaChannelNumTransactionSet

Name	DmaChannelNumTransactionSet		
Description	<p>This parameter indicates the number of transaction sets which can be configured to the particular channel. Linked list feature is offered in Transaction set configuration only if the value of this parameter is greater than one.</p> <p>The number of transaction sets is capped at DmaMaxTransactionSetPerChannel (to a max 0xFFFF).</p> <p>Note: Minimum TCS number is selected as the default value.</p>		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	1 - DmaMaxTransactionSetPerChannel (capped to a max 0xFFFF)		
Default value	1		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	DmaMaxTransactionSetPerChannel		

1.3.1.3.5 DmaTcsInterruptTransactionLoss
Table 18 Specification for DmaTcsInterruptTransactionLoss

Name	DmaTcsInterruptTransactionLoss
-------------	--------------------------------

DMA driver
Table 18 Specification for DmaTcsInterruptTransactionLoss (continued)

Description	This parameter enables/disables transaction loss interrupt. Note: The optional features are disabled by default to minimize the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4 Container: DmaChannelTransactionSet

Post-Build Variant Multiplicity: TRUE

Multiplicity Configuration Class: Pre-Compile

1.3.1.4.1 DmaNextTcsIndex
Table 19 Specification for DmaNextTcsIndex

Name	DmaNextTcsIndex		
Description	This parameter defines the next TCS node in case of linked list feature. Note: The next transaction set index is selected as the default value of the parameter.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - Number of DmaTcsIndex available in Dma channel (max 0xFFFF)		
Default value	DmaTcsIndex+1 (Incrementing index)		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

DMA driver

1.3.1.4.2 DmaTcsAppendTimeStamp

Table 20 Specification for DmaTcsAppendTimeStamp

Name	DmaTcsAppendTimeStamp		
Description	<p>This parameter determines if time stamp is to be appended after the last DMA move of a transaction.</p> <p>Note: The optional features are disabled by default to minimize the executable code size.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	<p>TRUE</p> <p>FALSE</p>		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.3 DmaTcsAutoStartEnable

Table 21 Specification for DmaTcsAutoStartEnable

Name	DmaTcsAutoStartEnable		
Description	<p>This parameter enables/disables the autostart feature in the case of a linked list</p> <p>Note 1: The optional features are kept disabled by default. To be enabled by user as per the need.</p> <p>Note 2: The last node of the linked list should not be configured for Autostart.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	<p>TRUE</p> <p>FALSE</p>		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	DmaTcsShadowRegisterConfiguration		

DMA driver
1.3.1.4.4 DmaTcsCircularBufferDestinationEnable
Table 22 Specification for DmaTcsCircularBufferDestinationEnable

Name	DmaTcsCircularBufferDestinationEnable		
Description	This parameter determines if a circular buffering scheme is to be implemented on the destination buffer. Note: The optional features are disabled by default to minimize the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.5 DmaTcsCircularBufferDestinationLength
Table 23 Specification for DmaTcsCircularBufferDestinationLength

Name	DmaTcsCircularBufferDestinationLength		
Description	This parameter determines how many bits of the destination address of a circular buffer can actually be modified. Remaining address bits stay unchanged. This parameter is grayed out if DmaTcsCircularBufferDestinationEnable is set to FALSE Note 1: The user has to ensure that the destination address is aligned accordingly (For e.g. if the circular buffer size is chosen to be 16, the address has to be aligned to the 16 byte boundary). Note 2: The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef

DMA driver
Table 23 Specification for DmaTcsCircularBufferDestinationLength (continued)

Range	DMA_CIRCULAR_BUFFER_LENGTH_1024BYTE: Circular buffer length of 1024 byte DMA_CIRCULAR_BUFFER_LENGTH_128BYTE: Circular buffer length of 128 byte DMA_CIRCULAR_BUFFER_LENGTH_16384BYTE: Circular buffer length of 16384 byte DMA_CIRCULAR_BUFFER_LENGTH_16BYTE: Circular buffer length of 16 byte DMA_CIRCULAR_BUFFER_LENGTH_1BYTE: Circular buffer length of 1 byte DMA_CIRCULAR_BUFFER_LENGTH_2048BYTE: Circular buffer length of 2048 byte DMA_CIRCULAR_BUFFER_LENGTH_256BYTE: Circular buffer length of 256 byte DMA_CIRCULAR_BUFFER_LENGTH_2BYTE: Circular buffer length of 2 byte DMA_CIRCULAR_BUFFER_LENGTH_32768BYTE: Circular buffer length of 32768 byte DMA_CIRCULAR_BUFFER_LENGTH_32BYTE: Circular buffer length of 32 byte DMA_CIRCULAR_BUFFER_LENGTH_4096BYTE: Circular buffer length of 4096 byte DMA_CIRCULAR_BUFFER_LENGTH_4BYTE: Circular buffer length of 4 byte DMA_CIRCULAR_BUFFER_LENGTH_512BYTE: Circular buffer length of 512 byte DMA_CIRCULAR_BUFFER_LENGTH_64BYTE: Circular buffer length of 64 byte DMA_CIRCULAR_BUFFER_LENGTH_8192BYTE: Circular buffer length of 8192 byte DMA_CIRCULAR_BUFFER_LENGTH_8BYTE: Circular buffer length of 8 byte		
Default value	CIRCULAR_BUFFER_LENGTH_1BYTE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.6 DmaTcsCircularBufferSourceEnable
Table 24 Specification for DmaTcsCircularBufferSourceEnable

Name	DmaTcsCircularBufferSourceEnable		
Description	This parameter determines if a circular buffering scheme is to be implemented on the source buffer. Note: The optional features are disabled by default to minimize the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-

DMA driver
Table 24 Specification for DmaTcsCircularBufferSourceEnable (continued)

Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.7 DmaTcsCircularBufferSourceLength
Table 25 Specification for DmaTcsCircularBufferSourceLength

Name	DmaTcsCircularBufferSourceLength		
Description	<p>This parameter determines how many bits of the source address of a circular buffer can actually be modified. Remaining address bits stay unchanged.</p> <p>This parameter is greyed out if DmaTcsCircularBufferSourceEnable is set to FALSE.</p> <p>Note 1: The user has to ensure that the source address is aligned accordingly (For e.g. if the circular buffer size is chosen to be 16, the address has to be aligned to the 16 byte boundary).</p> <p>Note 2: The default value of this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_CIRCULAR_BUFFER_LENGTH_1024BYTE: Circular buffer length of 1024 byte DMA_CIRCULAR_BUFFER_LENGTH_128BYTE: Circular buffer length of 128 byte DMA_CIRCULAR_BUFFER_LENGTH_16384BYTE: Circular buffer length of 16384 byte DMA_CIRCULAR_BUFFER_LENGTH_16BYTE: Circular buffer length of 16 byte DMA_CIRCULAR_BUFFER_LENGTH_1BYTE: Circular buffer length of 1 byte DMA_CIRCULAR_BUFFER_LENGTH_2048BYTE: Circular buffer length of 1024 byte DMA_CIRCULAR_BUFFER_LENGTH_2048BYTE: Circular buffer length of 2048 byte DMA_CIRCULAR_BUFFER_LENGTH_256BYTE: Circular buffer length of 256 byte DMA_CIRCULAR_BUFFER_LENGTH_2BYTE: Circular buffer length of 2 byte DMA_CIRCULAR_BUFFER_LENGTH_32768BYTE: Circular buffer length of 32768 byte DMA_CIRCULAR_BUFFER_LENGTH_32BYTE: Circular buffer length of 32 byte DMA_CIRCULAR_BUFFER_LENGTH_4096BYTE: Circular buffer length of 4096 byte DMA_CIRCULAR_BUFFER_LENGTH_4BYTE: Circular buffer length of 4 byte DMA_CIRCULAR_BUFFER_LENGTH_512BYTE: Circular buffer length of 512 byte DMA_CIRCULAR_BUFFER_LENGTH_64BYTE: Circular buffer length of 64 byte DMA_CIRCULAR_BUFFER_LENGTH_8192BYTE: Circular buffer length of 8192 byte DMA_CIRCULAR_BUFFER_LENGTH_8BYTE: Circular buffer length of 8 byte		
Default value	CIRCULAR_BUFFER_LENGTH_1BYTE		
Post-build variant value	TRUE	Post-build variant multiplicity	-

DMA driver
Table 25 Specification for DmaTcsCircularBufferSourceLength (continued)

Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.8 DmaTcsDaisyChaining
Table 26 Specification for DmaTcsDaisyChaining

Name	DmaTcsDaisyChaining		
Description	<p>This parameter determines whether the adjacent lower channel is daisy chained with this channel. If so, after a "transaction" successfully completes on this channel, an internal hardware trigger is forwarded to the adjacent channel.</p> <p>Note 1: The optional features are kept disabled by default.</p> <p>Note 2: When using the daisy chain feature, user should enable the transfer interrupt only for the last channel in the daisy chain sequence.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.9 DmaTcsDataTransferInterrupt
Table 27 Specification for DmaTcsDataTransferInterrupt

Name	DmaTcsDataTransferInterrupt		
Description	<p>This parameter enables/disables the data transfer/transaction interrupt.</p> <p>Note 1: The interrupts are kept disabled by default. To be enabled by user as per the need.</p> <p>Note 2: When using the daisy chain feature, user should enable the interrupt only for the last channel in the daisy chain sequence.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef

DMA driver
Table 27 Specification for DmaTcsDataTransferInterrupt (continued)

Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.10 DmaTcsDestinationAddress
Table 28 Specification for DmaTcsDestinationAddress

Name	DmaTcsDestinationAddress		
Description	<p>This parameter defines the start address of the destination memory. No checks are performed by the UI or code generator on the validity of the address entered.</p> <p>Value is input in hexadecimal format.</p> <p>No guarantees can be given about the behavior of the driver when an incorrect address is setup as the destination address.</p> <p>If double buffering is enabled, this parameter indicates the first of the two buffers.</p> <p>Note: Since the address value is user configurable, the default value is kept as zero.</p>		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x00000000 - 0xFFFFFFFF		
Default value	0x00000000		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.11 DmaTcsDestinationAddressModificationFactor
Table 29 Specification for DmaTcsDestinationAddressModificationFactor

Name	DmaTcsDestinationAddressModificationFactor
-------------	--

DMA driver
Table 29 Specification for DmaTcsDestinationAddressModificationFactor (continued)

Description	<p>This parameter defines how the destination address changes after every move operation. If DmaTcsCircularBufferDestinationLength is 0 and DmaTcsCircularBufferDestinationEnable is TRUE, the UI widget of this parameter is grayed out.</p> <p>The description field of Tresos is used to explain the significance of each of the values is provided.</p> <p>Note: The default value of this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	<p>DMA_FACTOR_128: Address offset is 128 x DmaChBitsPerMove</p> <p>DMA_FACTOR_16: Address offset is 16 x DmaChBitsPerMove</p> <p>DMA_FACTOR_1: Address offset is 1 x DmaChBitsPerMove</p> <p>DMA_FACTOR_2: Address offset is 2 x DmaChBitsPerMove</p> <p>DMA_FACTOR_4: Address offset is 4 x DmaChBitsPerMove</p> <p>DMA_FACTOR_64: Address offset is 64 x DmaChBitsPerMove</p> <p>DMA_FACTOR_8: Address offset is 8 x DmaChBitsPerMove</p>		
Default value	FACTOR_1		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.12 DmaTcsDestinationAddressMovement
Table 30 Specification for DmaTcsDestinationAddressMovement

Name	DmaTcsDestinationAddressMovement		
Description	<p>This parameter determines if the destination address after every move operation is to be increased or decreased.</p> <p>Note: The default value is chosen as 'increasing' for the natural progression. User can choose other values as needed.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	<p>DMA DECREASING: Address decrements</p> <p>DMA INCREASING: Address increments</p>		
Default value	INCREASING		
Post-build variant value	TRUE	Post-build variant multiplicity	-

DMA driver
Table 30 Specification for DmaTcsDestinationAddressMovement (continued)

Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.13 DmaTcsDoubleBuffer
Table 31 Specification for DmaTcsDoubleBuffer

Name	DmaTcsDoubleBuffer		
Description	<p>This parameter defines the start address of the double buffer.</p> <p>If double buffering is enabled, this parameter gets enabled and indicates the second buffer. With double buffering disabled, this parameter is grayed out.</p> <p>No checks are performed by the UI or code generator on the validity of the address entered. Value is input in hexadecimal.</p> <p>No guarantees can be given about the behavior of the driver when an incorrect address is setup as the double buffer.</p> <p>Note: Since the address value is user configurable, the default value is kept as zero.</p>		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x00000000 - 0xFFFFFFFF		
Default value	0x00000000		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.14 DmaTcsHardwareTrigger
Table 32 Specification for DmaTcsHardwareTrigger

Name	DmaTcsHardwareTrigger		
Description	<p>This parameter determines whether hardware initiated trigger is to be supported or not and if affirmative its mode</p> <p>Note: Since the hardware trigger is user configurable, the default value is kept as disabled.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef

DMA driver
Table 32 Specification for DmaTcsHardwareTrigger (continued)

Range	DMA_HARDWARE_TRIGGER_CONTINUOUS_MODE: After a DMA transaction, bit TSR.HTRE remains set, ready to accept the next trigger. DMA_HARDWARE_TRIGGER_SINGLE_MODE: After a DMA transaction, the DMA channel is disabled for further hardware requests DMA_NO_HARDWARE_TRIGGER: No hardware trigger would be used		
Default value	NO_HARDWARE_TRIGGER		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.15 DmaTcsIndex
Table 33 Specification for DmaTcsIndex

Name	DmaTcsIndex		
Description	This parameter defines the index of current TCS node in the list of TCS nodes. Note: The current transaction set index is selected as the default value of the parameter.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - DmaChannelNumTransactionSet -1 (max 0xFFFE)		
Default value	DmaChannelTransactionSet current Index		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.16 DmaTcsInterruptDataTransfer
Table 34 Specification for DmaTcsInterruptDataTransfer

Name	DmaTcsInterruptDataTransfer		
Description	Interrupts can be generated either for every transfer or even transaction. Additionally, an interrupt can be generated after a certain number of transfers equaling a threshold have been completed Note: The default value of this parameter is set to the reset value of the corresponding SFR.		

DMA driver
Table 34 Specification for DmaTcsInterruptDataTransfer (continued)

Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_INTERRUPT_AFTER_THRESHOLD: Interrupt is triggered when TCOUNT equals IRDV DMA_INTERRUPT_PER_TRANSACTION: Interrupt is triggered after the transaction is complete. DMA_INTERRUPT_PER_TRANSFER: Interrupt is triggered after the transfer is complete.		
Default value	INTERRUPT_PER_TRANSACTION		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.17 DmaTcsInterruptDataTransferThreshold
Table 35 Specification for DmaTcsInterruptDataTransferThreshold

Name	DmaTcsInterruptDataTransferThreshold		
Description	If the parameter DmaTcsInterruptDataTransfer is set to a value of INTERRUPT_AFTER_THRESHOLD, this parameter determines the threshold of transfer following which an interrupt is generated. Note: The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 15		
Default value	0		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.18 DmaTcsInterruptDestAddressWrap
Table 36 Specification for DmaTcsInterruptDestAddressWrap

Name	DmaTcsInterruptDestAddressWrap
-------------	--------------------------------

DMA driver
Table 36 Specification for DmaTcsInterruptDestAddressWrap (continued)

Description	This parameter enables/disables destination address wrap around interrupt. Note: The interrupts are kept disabled by default. To be enabled by user as per the need.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.19 DmaTcsInterruptSourceAddressWrap
Table 37 Specification for DmaTcsInterruptSourceAddressWrap

Name	DmaTcsInterruptSourceAddressWrap		
Description	This parameter enables/disables source address wrap around interrupt Note: The interrupts are kept disabled by default. To be enabled by user as per the need.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.20 DmaTcsMoveLength
Table 38 Specification for DmaTcsMoveLength

Name	DmaTcsMoveLength
-------------	------------------

DMA driver
Table 38 Specification for DmaTcsMoveLength (continued)

Description	<p>This parameter defines the amount of data transferred per move operation. The description field of Tresos tool explains the significance of the range of input values.</p> <p>Note: The channel data width has to be chosen depending on whether the SRI or SPB bus is used. User has to ensure that the appropriate channel width is chosen. Refer the Target Specification for the valid configurations.</p> <p>Note: The default value of this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	<p>DMA_WIDTH_128BITS: Number of bits per move is 128 bits</p> <p>DMA_WIDTH_16BITS: Number of bits per move is 16 bits</p> <p>DMA_WIDTH_256BITS: Number of bits per move is 256 bits</p> <p>DMA_WIDTH_32BITS: Number of bits per move is 32 bits</p> <p>DMA_WIDTH_64BITS: Number of bits per move is 64 bits</p> <p>DMA_WIDTH_8BITS: Number of bits per move is 8 bits</p>		
Default value	WIDTH_8BITS		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.21 DmaTcsReferenceAddressCrc
Table 39 Specification for DmaTcsReferenceAddressCrc

Name	DmaTcsReferenceAddressCrc		
Description	<p>This parameter defines the reference address CRC address calculated by the user. No checks are performed by the UI or code generator on the validity of the CRC entered. Value is input in hexadecimal format.</p> <p>Note: Since the CRC value is user configurable, the default value is kept as zero.</p>		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x00000000 - 0xFFFFFFFF		
Default value	0x00000000		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-

DMA driver
Table 39 Specification for DmaTcsReferenceAddressCrc (continued)

Origin	IFX	Scope	LOCAL
Dependency	DmaTcsShadowRegisterConfiguration		

1.3.1.4.22 DmaTcsReferenceDataCrc
Table 40 Specification for DmaTcsReferenceDataCrc

Name	DmaTcsReferenceDataCrc		
Description	<p>This parameter defines the reference data CRC address calculated by the user. No checks are performed by the UI or code generator on the validity of the CRC entered. Value is input in hexadecimal format.</p> <p>Note: Since the CRC value is user configurable, the default value is kept as zero.</p>		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x00000000 - 0xFFFFFFFF		
Default value	0x00000000		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	DmaTcsShadowRegisterConfiguration		

1.3.1.4.23 DmaTcsShadowRegisterConfiguration
Table 41 Specification for DmaTcsShadowRegisterConfiguration

Name	DmaTcsShadowRegisterConfiguration		
Description	<p>This parameter determines how the shadow register is to be used. Details are in the Internal Target Specification.</p> <p>If the parameter DmaChannelNumTransactionSet is set to a value greater than one, user is allowed to choose only the linked list options.</p> <p>Further, it is only for the first node of the linked list that the user is allowed to choose the specific linked list type. All other non-leaf nodes of this linked list inherit the property of the root transaction set. The leaf node of the linked list has SHADOWING_DISABLED configured</p> <p>Note: The default value of this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef

DMA driver
Table 41 Specification for DmaTcsShadowRegisterConfiguration (continued)

Range	DMA_ACCUMULATED_LINKED_LIST: Accumulated Linked List (ACLL) DMA_CONDITIONAL_LINKED_LIST: Conditional Linked List (CONLL) DMA_DEST_ADDRESS_BUFFERING_RO: Shadow Operation Read Only Mode Destination Address DMA_DEST_ADDRESS_BUFFERING_RW: Shadow Operation Direct Write Mode Destination Address DMA_DEST_DOUBLE_BUFFERING_HW_SW_SWITCH: DMA Double Destination Buffering with Software Switch and Automatic Hardware Switch DMA_DEST_DOUBLE_BUFFERING_SW_SWITCH: DMA Double Destination Buffering with Software Switch Only DMA_LINKED_LIST: DMA Linked List (DMALL) DMA_SAFE_LINKED_LIST: Safe Linked List (SAFLL) DMA_SHADOWING_DISABLED: Shadow operation disabled DMA_SOURCE_ADDRESS_BUFFERING_RO: Shadow Operation Read Only Mode Source Address DMA_SOURCE_ADDRESS_BUFFERING_RW: Shadow Operation Direct Write Mode Source Address DMA_SOURCE_DOUBLE_BUFFERING_HW_SW_SWITCH: DMA Double Source Buffering with Software Switch and Automatic Hardware Switch DMA_SOURCE_DOUBLE_BUFFERING_SW_SWITCH: DMA Double Source Buffering with Software Switch Only		
Default value	SHADOWING_DISABLED		
Post-build variant value		Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.24 DmaTcsSourceAddress
Table 42 Specification for DmaTcsSourceAddress

Name	DmaTcsSourceAddress
Description	<p>This parameter defines the start address of the source memory. No checks are performed by the UI or code generator on the validity of the address entered.</p> <p>Value is input in hexadecimal format.</p> <p>No guarantees can be given about the behavior of the driver when an incorrect address is setup as the source address.</p> <p>If double buffering is enabled, this parameter indicates the first of the two buffers.</p> <p>Note: Since the address value is user configurable, the default value is kept as zero.</p>

DMA driver
Table 42 Specification for DmaTcsSourceAddress (continued)

Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x00000000 - 0xFFFFFFFF		
Default value	0x00000000		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.25 DmaTcsSourceAddressModificationFactor
Table 43 Specification for DmaTcsSourceAddressModificationFactor

Name	DmaTcsSourceAddressModificationFactor		
Description	<p>This parameter defines how the source address changes after every move operation. If DmaTcsCircularBufferSourceLength is 0 and DmaTcsCircularBufferSourceEnable is TRUE, the UI widget of this parameter is grayed out.</p> <p>The description field of Tresos is used to explain the significance of each of the values is provided.</p> <p>Note: The default value of this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_FACTOR_128: Address offset is 128 x DmaChBitsPerMove DMA_FACTOR_16: Address offset is 16 x DmaChBitsPerMove DMA_FACTOR_1: Address offset is 1 x DmaChBitsPerMove DMA_FACTOR_2: Address offset is 2 x DmaChBitsPerMove DMA_FACTOR_32: Address offset is 32 x DmaChBitsPerMove DMA_FACTOR_4: Address offset is 4 x DmaChBitsPerMove DMA_FACTOR_64: Address offset is 64 x DmaChBitsPerMove DMA_FACTOR_8: Address offset is 8 x DmaChBitsPerMove		
Default value	FACTOR_1		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

DMA driver
1.3.1.4.26 DmaTcsSourceAddressMovement
Table 44 Specification for DmaTcsSourceAddressMovement

Name	DmaTcsSourceAddressMovement		
Description	This parameter determines if the source address after every move operation is to be increased or decreased. Note: The default value is chosen as 'increasing' for the natural progression. User can choose other values as needed.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_DECREASING: Address decrements DMA_INCREASING: Address increments		
Default value	INCREASING		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.27 DmaTcsTransactionLength
Table 45 Specification for DmaTcsTransactionLength

Name	DmaTcsTransactionLength		
Description	This parameter defines how many transfers are performed per DMA transaction. Value 0 and 1 will have same impact (i.e. one transfer) as DMA hardware executes at least one DMA transfer on a channel start. Note: Since the length value is user configurable, the default value is kept as zero.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 16383		
Default value	0		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

DMA driver
1.3.1.4.28 DmaTcsTransferLength
Table 46 Specification for DmaTcsTransferLength

Name	DmaTcsTransferLength		
Description	This parameter defines how many moves are performed per DMA transfer. The description field of Tresos tool explains the significance of the range of input values. Note: The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_MOVES_16: One transfer has 16 move DMA_MOVES_1: One transfer has 1 move DMA_MOVES_2: One transfer has 2 move DMA_MOVES_3: One transfer has 3 move DMA_MOVES_4: One transfer has 4 move DMA_MOVES_5: One transfer has 5 move DMA_MOVES_8: One transfer has 8 move DMA_MOVES_9: One transfer has 9 move		
Default value	MOVES_1		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.4.29 DmaTcsTriggerFrequency
Table 47 Specification for DmaTcsTriggerFrequency

Name	DmaTcsTriggerFrequency		
Description	This parameter determines how much of data will the move engine move upon a trigger request (either hardware or software trigger). The move engine, upon receipt of a trigger request, could effectuate a transfer or even a transaction based on the choice made. Note: The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_TRANSACTION_PER_TRIGGER: One DMA request starts a complete DMA transaction DMA_TRANSFER_PER_TRIGGER: A DMA request is required for each DMA transfer		
Default value	TRANSFER_PER_TRIGGER		
Post-build variant value	TRUE	Post-build variant multiplicity	-

DMA driver
Table 47 Specification for DmaTcsTriggerFrequency (continued)

Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5 Container: DmaGeneral

This container contains the general configuration parameters needed for the DMA driver module.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.5.1 DmaBufferSwitchApiConfiguration
Table 48 Specification for DmaBufferSwitchApiConfiguration

Name	DmaBufferSwitchApiConfiguration		
Description	This parameter results in generation of a preprocessor macro that conditionally includes code related to buffer switch interface Note: The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.2 DmaChDeInitApiConfiguration
Table 49 Specification for DmaChDeInitApiConfiguration

Name	DmaChDeInitApiConfiguration		
Description	Adds / removes the service Dma_ChDeInit() from the code. true: Dma_ChDeInit() can be used. false: Dma_ChDeInit() cannot be used. Note: The optional APIs are disabled by default to minimize the executable code size.		

DMA driver
Table 49 Specification for DmaChDeInitApiConfiguration (continued)

Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.3 DmaDataPendingApiConfiguration
Table 50 Specification for DmaDataPendingApiConfiguration

Name	DmaDataPendingApiConfiguration		
Description	Adds / removes the service Dma_ChGetRemainingData() from the code. true: Dma_ChGetRemainingData() can be used. false: Dma_ChGetRemainingData() cannot be used. Note: The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.4 DmaDevErrorDetect
Table 51 Specification for DmaDevErrorDetect

Name	DmaDevErrorDetect
-------------	-------------------

DMA driver
Table 51 Specification for DmaDevErrorDetect (continued)

Description	Switches the Default Error Tracer (DET) detection and notification ON or OFF. true: enabled (ON). false: disabled (OFF).		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.5 DmaGetVersionInfoApiConfiguration
Table 52 Specification for DmaGetVersionInfoApiConfiguration

Name	DmaGetVersionInfoApiConfiguration		
Description	Adds / removes the API Dma_GetVersionInfo() from the code. true: Dma_GetVersionInfo() can be used. false: Dma_GetVersionInfo() cannot be used. Note: The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

DMA driver

1.3.1.5.6 DmaInitCheckApi

Table 53 Specification for DmaInitCheckApi

Name	DmaInitCheckApi		
Description	<p>Adds / removes the service Dma_InitCheck() from the code.</p> <p>true: Dma_InitCheck() can be used.</p> <p>false: Dma_InitCheck() cannot be used.</p> <p>Note: The optional APIs are disabled by default to minimize the executable code size.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.7 DmaInitDeInitApiMode

Table 54 Specification for DmaInitDeInitApiMode

Name	DmaInitDeInitApiMode		
Description	<p>This configuration parameter gives the mode in which the Dma Init and De-Init APIs will be used.</p> <p>Note: Since DMA driver accesses the SFRs, it is more efficient to operate the DMA driver in supervisor mode. Hence, the default mode of operation is supervisor.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	<p>DMA_MCAL_SUPERVISORMODE: Operating mode used is Supervisor</p> <p>DMA_MCAL_USER1MODE: Operating mode used is USER1</p>		
Default value	DMA_MCAL_SUPERVISORMODE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL

DMA driver
Table 54 Specification for DmaInitDeInitApiMode (continued)

Dependency	DmaRuntimeApiMode
-------------------	-------------------

1.3.1.5.8 DmaMaxTransactionSetPerChannel
Table 55 Specification for DmaMaxTransactionSetPerChannel

Name	DmaMaxTransactionSetPerChannel		
Description	This parameter controls how many transaction sets are allowed to be defined per channel. No code is generated based on this parameter Note: Minimum TCS number is selected as the default value.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	1 - 65535		
Default value	1		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.9 DmaMultiCoreErrorDetect
Table 56 Specification for DmaMultiCoreErrorDetect

Name	DmaMultiCoreErrorDetect		
Description	The parameter enables or disables the multi core related default error tracer (DET) detection and reporting. It is applicable only when DET/Safety is enabled.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	DmaDevErrorDetect		

DMA driver
1.3.1.5.10 DmaRuntimeApiMode
Table 57 Specification for DmaRuntimeApiMode

Name	DmaRuntimeApiMode		
Description	This configuration parameter gives the mode in which the Runtime API will be used. Note: Since DMA driver accesses the SFRs, it is more efficient to operate the DMA driver in supervisor mode. Hence, the default mode of operation is supervisor.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_MCAL_SUPERVISORMODE: Operating mode used is Supervisor DMA_MCAL_USER1MODE: Operating mode used is USER1		
Default value	DMA_MCAL_SUPERVISORMODE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.11 DmaSafetyEnable
Table 58 Specification for DmaSafetyEnable

Name	DmaSafetyEnable		
Description	Switch to enable/disable the safety check and reporting. true: Enable safety check and reporting false: Disable safety check and reporting Note: The detection of safety related errors is enabled by default to ensure that safety issues are addressed during the product lifecycle.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

DMA driver
1.3.1.5.12 DmaSuspendApiConfiguration
Table 59 Specification for DmaSuspendApiConfiguration

Name	DmaSuspendApiConfiguration		
Description	<p>Adds / removes the services Dma_ChTransferFreeze() and Dma_ChTransferResume() from the code.</p> <p>true: Dma_ChTransferFreeze() and Dma_ChTransferResume() can be used.</p> <p>false: Dma_ChTransferFreeze() and Dma_ChTransferResume() cannot be used.</p> <p>Note: The optional APIs are disabled by default to minimize the executable code size.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.5.13 DmaTriggerApiConfiguration
Table 60 Specification for DmaTriggerApiConfiguration

Name	DmaTriggerApiConfiguration		
Description	<p>Adds / removes the services Dma_ChEnableHardwareTrigger() and Dma_ChDisableHardwareTrigger() from the code.</p> <p>true: Dma_ChEnableHardwareTrigger() and Dma_ChDisableHardwareTrigger() can be used.</p> <p>false: Dma_ChEnableHardwareTrigger() and Dma_ChDisableHardwareTrigger() cannot be used.</p> <p>Note: The optional APIs are disabled by default to minimize the executable code size.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-

DMA driver
Table 60 Specification for DmaTriggerApiConfiguration (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.6 Container: DmaMoveEngineConfig

DmaMoveEngineConfig contains the configuration parameters for the notifications from the move engine.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.6.1 DmaMEDestinationErrorInterrupt
Table 61 Specification for DmaMEDestinationErrorInterrupt

Name	DmaMEDestinationErrorInterrupt		
Description	This parameter enables/disables Move engine destination error interrupt. Note: The interrupts are kept disabled by default. To be enabled by user as per the need.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.6.2 DmaMELinkedListErrorInterrupt
Table 62 Specification for DmaMELinkedListErrorInterrupt

Name	DmaMELinkedListErrorInterrupt		
Description	This parameter enables/disables linked list error interrupts Note: The interrupts are kept disabled by default. To be enabled by user as per the need.		
Multiplicity	1..1	Type	EcucBooleanParamDef

DMA driver
Table 62 Specification for DmaMELinkedListErrorInterrupt (continued)

Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.6.3 DmaMESourceErrorInterrupt
Table 63 Specification for DmaMESourceErrorInterrupt

Name	DmaMESourceErrorInterrupt		
Description	This parameter enables/disables Move engine source error interrupt. Note: The interrupts are kept disabled by default. To be enabled by user as per the need.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.7 Container: DmaResourcePartition

DmaResourcePartition contains the configuration related to the resource partitions of DMA.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

DMA driver

1.3.1.7.1 DmaPermittedBusMaster

Table 64 Specification for DmaPermittedBusMaster

Name	DmaPermittedBusMaster		
Description	This parameter is a 32 bit hexadecimal mask and represents the bus masters that are allowed to access this partition. Note: The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x00000000 - 0xFFFFFFFF		
Default value	0xFFFFFFFF		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.7.2 DmaResourcePartitionBusMode

Table 65 Specification for DmaResourcePartitionBusMode

Name	DmaResourcePartitionBusMode		
Description	This parameter defines the bus access mode of the resource partition Note: The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	DMA_RP_SUPERVISOR_MODE: Resource partition would be in the supervisor mode. DMA_RP_USER_MODE: Resource partition would be in the user mode.		
Default value	DMA_RP_SUPERVISOR_MODE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

DMA driver
1.3.1.7.3 DmaResourcePartitionErrorNotifRoutine
Table 66 Specification for DmaResourcePartitionErrorNotifRoutine

Name	DmaResourcePartitionErrorNotifRoutine		
Description	This parameter defines the callback routine which is to be invoked by the resource partition error interrupt handler when an error interrupt occurs. Note: Since the name of the function is user configurable, the default value is kept as NULL.		
Multiplicity	1..1	Type	Dma_ErrorNotificationFnPtrType
Range			
Default value	NULL		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.2 Functions - Type definitions
1.3.2.1 Dma_ChannelNotificationFnPtrType
Table 67 Specification for Dma_ChannelNotificationFnPtrType

Syntax	Dma_ChannelNotificationFnPtrType
Type	Pointer to a function of type void Function_Name (const uint8 Channel, const uint32 Event)
File	Dma . h
Description	This function pointer type would hold the address of the function which has to be invoked when there is a channel event.
Source	IFX

1.3.2.2 Dma_ChEventType
Table 68 Specification for Dma_ChEventType

Syntax	Dma_ChEventType
Type	Enumeration
File	Dma . h

DMA driver
Table 68 Specification for Dma_ChEventType (continued)

Range	0 - DMA_CHEVENT_NONE	No channel events
	1 - DMA_CHEVENT_TRANSFER_COMPLETE	Desired transfers completed
	2 - DMA_CHEVENT_BUFFER_WRAP_SOURCE	Source circular buffer wrap detected
	4 - DMA_CHEVENT_BUFFER_WRAP_DEST	Destination circular buffer wrap detected
	8 - DMA_CHEVENT_UNKNOWN_EVENT	Unknown event due to hardware limitations
Description	Dma_ChEventType is an enumeration for representing the various channel events that may be the cause of a channel interrupt.	
Source	IFX	

1.3.2.3 Dma_ConfigType
Table 69 Specification for Dma_ConfigType

Syntax	Dma_ConfigType	
Type	Structure	
File	Dma . h	
Range	--	
Description	Defines the type for data structure containing the set of configuration parameters required for initializing the DMA driver and DMA hardware unit(s).	
Source	IFX	

1.3.2.4 Dma_ConfigUpdateType
Table 70 Specification for Dma_ConfigUpdateType

Syntax	Dma_ConfigUpdateType	
Type	Structure	
File	Dma . h	

DMA driver
Table 70 Specification for Dma_ConfigUpdateType (continued)

Range	uint32 AddressCrc	Contents of SDCRC register
	uint32 Config	Contents of CHCFG register
	uint32 Control	Contents of ADICR register
	uint32 DestAddress	Start of the destination buffer address
	unsigned_int UpdateAddressCrc:1	Indicates whether the SDCRC should be updated or not
	unsigned_int UpdateConfig:1	Indicates whether the CHCFG should be updated or not
	unsigned_int UpdateControlAdicr:1	Indicates whether the ADICR should be updated or not
	unsigned_int UpdateDestAddress:1	Indicates whether the Destination address should be updated or not
	unsigned_int UpdateShadowConfig:1	Indicates whether the SHADR should be updated or not
	unsigned_int UpdateSourceAddress:1	Indicates whether the Source address should be updated or not
	uint32 ShadowConfig	Contents of SHADR register
	uint32 SourceAddress	Start of the source buffer address
	unsigned_int UpdateControlChcsr:1	Indicates whether the chcsr register should be updated or not
Description	Dma_ConfigUpdateType contains the elements which are needed in updating the channel settings of an already configured channel.	
Source	IFX	

1.3.2.5 Dma_CrcType
Table 71 Specification for Dma_CrcType

Syntax	Dma_CrcType	
Type	Enumeration	
File	Dma . h	
Range	0 - DMA_NO_CRC_TYPE	Invalid option
	1 - DMA_DATA_CRC_TYPE	Refers to Data CRC
	2 - DMA_ADDRESS_CRC_TYPE	Refers to Address CRC
Description	This enum holds the list of the CRC types available	
Source	IFX	

DMA driver

1.3.2.6 Dma_ErrorNotificationFnPtrType

Table 72 Specification for Dma_ErrorNotificationFnPtrType

Syntax	Dma_ErrorNotificationFnPtrType
Type	Pointer to a function of type void Function_Name (const uint8 Channel, const uint32 Event)
File	Dma.h
Description	This function pointer holds the function which needs to be invoked if there an error event from the move engine. There would be 1 interrupt per resource partition.
Source	IFX

1.3.2.7 Dma_MoveEngineEventType

Table 73 Specification for Dma_MoveEngineEventType

Syntax	Dma_MoveEngineEventType	
Type	Enumeration	
File	Dma.h	
Range	0 - DMA_MOVE_ENGINE_EVENT_NONE	No move engine events
	1 - DMA_MOVE_ENGINE_SOURCE_ERROR	Bus error while moving data from source
	2 - DMA_MOVE_ENGINE_DESTINATION_ERROR	Bus error while moving data to destination
	4 - DMA_MOVE_ENGINE_SPB_ERROR	Bus error-SPB
	8 - DMA_MOVE_ENGINE_SRI_ERROR	Bus error-SRI
	16 - DMA_MOVE_ENGINE_RAM_ERROR	DMARAM access error
	32 - DMA_MOVE_ENGINE_SAFE_LINKEDLIST_ERROR	CRC comparison error
	64 - DMA_MOVE_ENGINE_DMA_LINKEDLIST_ERROR	DMARAM access error (during over write of TCS in linked list scenario)
	128 - DMA_CHANNEL_TRL_ERROR	Transaction request lost error
Description	Dma_MoveEngineEventType is an enumeration for representing the various channel events that may be the cause of a channel interrupt	
Source	IFX	

1.3.2.8 Dma_MoveEngineListType

Table 74 Specification for Dma_MoveEngineListType

Syntax	Dma_MoveEngineListType
Type	Enumeration

DMA driver

Table 74 Specification for Dma_MoveEngineListType (continued)

File	Dma . h	
Range	0 - DMA_ME_NONE	Invalid option
	1 - DMA_ME_0	Refers to Move Engine 0
	2 - DMA_ME_1	Refers to Move Engine 1
	4 - DMA_ME_ALL	Refers to both Move Engines
Description	This enum holds the list of the Move Engines available	
Source	IFX	

1.3.3 Functions - APIs

1.3.3.1 Dma_Init

Table 75 Specification for Dma_Init API

Syntax	void Dma_Init (const Dma_ConfigType * const ConfigPtr)	
Service ID	0x01	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to root data structure of all post build configurations
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface is meant to be called by the ECU initialization routine. The move engines, the resource partitions and the channels are initialized.	
Source	IFX	

DMA driver
Table 75 **Specification for Dma_Init API (continued)**

Error handling	DET: DMA_E_NULL_POINTER: NULL pointer DMA_E_ALREADY_INITIALIZED: DMA driver initialization requested despite the driver already initialized DMA_E_MASTER_UNINIT: A slave core initialization is attempted, before initializing the master core. DMA_E_CORE_NOT_CONFIGURED: Channel not configured for the particular core is invoked Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.2 Dma_IsInitDone
Table 76 **Specification for Dma_IsInitDone API**

Syntax	Std_ReturnType Dma_IsInitDone (void)	
Service ID	0x02	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK if the driver has been initialized, E_NOT_OK otherwise
Description	This interface is meant to provide the status whether the initialization process for the DMA driver has been executed or not.	
Source	IFX	
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	

DMA driver

Table 76 Specification for Dma_IsInitDone API (continued)

Configuration dependencies	-
User hints	Note: Dma_InitDone API only indicates if the module initialization was invoked or not. In case channels were stopped, the user shall not rely or use this API to derive the initialization status of the channel.

1.3.3.3 Dma_ChInit

Table 77 Specification for Dma_ChInit API

Syntax	void Dma_ChInit (const uint8 Channel)	
Service ID	0x03	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface is meant to be called to initialize the specified DMA channel with parameters defined in DMAChannelTransactionSet container. This interface is called by the owner (e.g. ADC driver) of the specified channel. A typical use case would be that an ongoing transfer was aborted by invocation of Dma_ChStopTransfer entailing re-initialization of the channel. Any previously triggered interrupt statues would be cleared when the channel init is done.	
Source	IFX	
Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CH_ALREADY_INITIALIZED: DMA channel initialization requested despite the channel already initialized DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	DmaDevErrorDetect	

DMA driver
Table 77 **Specification for Dma_ChInit API (continued)**

User hints	Hint: If a DMA channel is reset (for e.g. due to invocation of 'Dma_ChStopTransfer' or 'Dma_ChDelnit' APIs), the user has to invoke 'Dma_ChInit' to restore the initialization status of the channel.
-------------------	---

1.3.3.4 **Dma_ChUpdate**
Table 78 **Specification for Dma_ChUpdate API**

Syntax	void Dma_ChUpdate (const uint8 Channel, const Dma_ConfigUpdateType * const Config, const uint32 * const NodeAddress)	
Service ID	0x04	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel Config NodeAddress	DMA channel number Pointer to configuration data Start of a memory block which must be formatted with information in "Config" object
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	In certain cases, applications wishing to use the DMA are informed of transfer attributes only at run time. In such cases, it is not possible to use the post-build configuration data. This interface may be used to program transfer characteristics at run time.	
Source	IFX	

DMA driver
Table 78 Specification for Dma_ChUpdate API (continued)

Error handling	DET: DMA_E_NULL_POINTER: NULL pointer DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_INVALID_SHADOW_CONFIG_REQ: Dma_ChUpdate API requested for Shadow register update while ADICR is configured in non-shadow configuration DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	Note: Dma_ChUpdate API should not be called by the user with invalid TCS contents (reserved bit fields selected, unused features selected, read-only or status bits set).

1.3.3.5 Dma_ChDeInit
Table 79 Specification for Dma_ChDeInit API

Syntax	void Dma_ChDeInit (const uint8 Channel)	
Service ID	0x05	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface is meant to be called to de-initialize a previously configured channel.	
Source	IFX	

DMA driver

Table 79 Specification for Dma_ChDeInit API (continued)

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaChDeInitApiConfiguration, DmaDevErrorDetect
User hints	-

1.3.3.6 Dma_ChTransferFreeze

Table 80 Specification for Dma_ChTransferFreeze API

Syntax	void Dma_ChTransferFreeze (const uint8 Channel)	
Service ID	0x06	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface may be used to temporarily freeze an ongoing transfer. This is typically useful in situations where a block of memory which is an endpoint in a DMA transaction is required to undergo memory testing by a diagnostics task scheduled periodically.	
Source	IFX	

DMA driver
Table 80 Specification for Dma_ChTransferFreeze API (continued)

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_NO_TRANSFERS_PENDING: Freeze requested when no transfers are pending DMA_E_TIMEOUT: Timeout detected while waiting for a certain value of critical register bit fields DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaSuspendApiConfiguration,DmaDevErrorDetect
User hints	-

1.3.3.7 Dma_ChTransferResume
Table 81 Specification for Dma_ChTransferResume API

Syntax	void Dma_ChTransferResume (const uint8 Channel)	
Service ID	0x07	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface may be used to resume a previously frozen channel.	
Source	IFX	

DMA driver
Table 81 Specification for Dma_ChTransferResume API (continued)

Error handling	<p>DET:</p> <p>DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init</p> <p>DMA_E_NOT_IN_FREEZE_STATE: Resume requested without a prior freeze</p> <p>DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel</p> <p>DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter</p> <p>DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>
Configuration dependencies	DmaSuspendApiConfiguration, DmaDevErrorDetect
User hints	The resume interface would put the channel from the freeze (halt) state to the idle state. To continue any DMA operations, which was in progress, the user has to call the start transfer interface.

1.3.3.8 Dma_ChEnableHardwareTrigger
Table 82 Specification for Dma_ChEnableHardwareTrigger API

Syntax	<code>void Dma_ChEnableHardwareTrigger (const uint8 Channel)</code>	
Service ID	0x08	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	<p>This interface is meant to be used to enable (or re-enable) hardware trigger recognition capability. This is particularly important if the channel has been configured for a DMA trigger to be asserted upon detection of a hardware trigger from a peripheral. In "Hardware Trigger-Single" mode of operation, the hardware trigger detection capability is lost after a transaction and must be reinstalled. This API is precisely meant to do just that.</p>	
Source	IFX	

DMA driver
Table 82 Specification for Dma_ChEnableHardwareTrigger API (continued)

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_NOT_IN_FREEZE_STATE: Resume requested without a prior freeze DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_FREEZE_STATE: Data transfer start/channel update requested by application during freeze state DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaTriggerApiConfiguration,DmaDevErrorDetect
User hints	Note: The Dma_ChEnableHardwareTrigger API should not be called for a channel which is configured as 'no hardware trigger'

1.3.3.9 Dma_ChDisableHardwareTrigger
Table 83 Specification for Dma_ChDisableHardwareTrigger API

Syntax	void Dma_ChDisableHardwareTrigger (const uint8 Channel)	
Service ID	0x09	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface disables hardware trigger detection capability.	
Source	IFX	

DMA driver
Table 83 Specification for Dma_ChDisableHardwareTrigger API (continued)

Error handling	<p>DET:</p> <p>DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init</p> <p>DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter</p> <p>DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel</p> <p>DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked</p> <p>DMA_E_FREEZE_STATE: Data transfer start/channel update requested by application during freeze state</p> <p>DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>
Configuration dependencies	DmaTriggerApiConfiguration, DmaDevErrorDetect
User hints	Note: The Dma_ChDisableHardwareTrigger API should not be called for a channel which is configured as 'no hardware trigger'

1.3.3.10 Dma_ChStartTransfer
Table 84 Specification for Dma_ChStartTransfer API

Syntax	void Dma_ChStartTransfer (const uint8 Channel)	
Service ID	0x0A	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface may be used by users of DMA channels (e.g. SPI driver) to manually start a DMA transfer.	
Source	IFX	

DMA driver
Table 84 Specification for Dma_ChStartTransfer API (continued)

Error handling	<p>DET:</p> <p>DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init</p> <p>DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel</p> <p>DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter</p> <p>DMA_E_FREEZE_STATE: Data transfer start/channel update requested by application during freeze state</p> <p>DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress</p> <p>DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.11 Dma_ChStopTransfer
Table 85 Specification for Dma_ChStopTransfer API

Syntax	void Dma_ChStopTransfer (const uint8 Channel)	
Service ID	0x0B	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface may be used by users of DMA channels (e.g. ADC driver) to abort an ongoing transfer.	
Source	IFX	

DMA driver
Table 85 **Specification for Dma_ChStopTransfer API (continued)**

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_TIMEOUT: Timeout detected while waiting for a certain value of critical register bit fields DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	The channel stoppage would result in the hardware resetting the registers of the channel being used. To use the channel again, the user should call the 'Dma_ChInit' API, to get the registers re-initialized.

1.3.3.12 Dma_ChGetRemainingData
Table 86 **Specification for Dma_ChGetRemainingData API**

Syntax	uint32 Dma_ChGetRemainingData (const uint8 Channel)	
Service ID	0x0C	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	Pending number of Dma transfers
Description	This interface may be used by users of DMA channels to find out how many DMA transfers are pending to copy data from source to destination address.	
Source	IFX	

DMA driver

Table 86 Specification for Dma_ChGetRemainingData API (continued)

Error handling	<p>DET:</p> <p>DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init</p> <p>DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel</p> <p>DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter</p> <p>DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked</p> <p>DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>
Configuration dependencies	DmaDataPendingApiConfiguration,DmaDevErrorDetect
User hints	-

1.3.3.13 Dma_ChSwitchBuffer

Table 87 Specification for Dma_ChSwitchBuffer API

Syntax	void Dma_ChSwitchBuffer (const uint8 Channel)	
Service ID	0x0D	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	<p>This interface may be used in a double buffering scheme, to redirect the data from the source to destination in either of the following ways:</p> <ol style="list-style-type: none"> 1. From two source buffers to a destination buffer or 2. From a source buffer to two destination buffers. 	
Source	IFX	

DMA driver
Table 87 **Specification for Dma_ChSwitchBuffer API (continued)**

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_DATA_CHANNEL_INVALID_SWITCH_REQ: Switch Buffer API invoked despite the feature not configured DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaBufferSwitchApiConfiguration,DmaDevErrorDetect
User hints	-

1.3.3.14 Dma_ChIsStatusAsserted
Table 88 **Specification for Dma_ChIsStatusAsserted API**

Syntax	Std_ReturnType Dma_ChIsStatusAsserted (const uint8 Channel, const Dma_ChEventType Event)	
Service ID	0x0E	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel Event	DMA channel number Event whose status is to be evaluated
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK - if the requested Event is found asserted by hardware, E_NOT_OK - otherwise
Description	This interface may be used by users of DMA channels to find out whether the specified event (Transaction request loss, Wrap around and Transfer Complete) has occurred or not. This is particularly useful in polling mode of operation.	
Source	IFX	

DMA driver
Table 88 Specification for Dma_ChIsStatusAsserted API (continued)

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_EVENT: Incorrect channel event DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.15 Dma_ChStatusClear
Table 89 Specification for Dma_ChStatusClear API

Syntax	void Dma_ChStatusClear (const uint8 Channel, const Dma_ChEventType Event)	
Service ID	0x0F	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel Event	DMA channel number Event whose status is to be cleared
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface may be used by users of DMA channels (e.g. SPI driver) to reset the specified status event.	
Source	IFX	

DMA driver
Table 89 Specification for Dma_ChStatusClear API (continued)

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_EVENT: Incorrect channel event DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.16 Dma_ChInterruptEnable
Table 90 Specification for Dma_ChInterruptEnable API

Syntax	void Dma_ChInterruptEnable (const uint8 Channel, const Dma_ChEventType Event)	
Service ID	0x10	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel Event	DMA channel number Event whose status is to be cleared
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface may be used by users of DMA channels (e.g. SPI driver) to enable interrupt generation upon occurrence of the specified event.	
Source	IFX	

DMA driver
Table 90 Specification for Dma_ChInterruptEnable API (continued)

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_EVENT: Incorrect channel event DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress DMA_E_CH_NO_NOTIFICATION_CONFIGURED: No notification function is configured for channel interrupt DMA_E_FREEZE_STATE: Data transfer start/channel update requested by application during freeze state Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.17 Dma_ChInterruptDisable
Table 91 Specification for Dma_ChInterruptDisable API

Syntax	<pre>void Dma_ChInterruptDisable (const uint8 Channel, const Dma_ChEventType Event)</pre>	
Service ID	0x11	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel Event	DMA channel number Event whose status is to be cleared
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interface may be used by users of DMA channels (e.g. SPI driver) to disable interrupt generation upon occurrence of the specified event.	

DMA driver
Table 91 Specification for Dma_ChInterruptDisable API (continued)

Source	IFX
Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_EVENT: Incorrect channel event DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress DMA_E_FREEZE_STATE: Data transfer start/channel update requested by application during freeze state Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.18 Dma_GetVersionInfo
Table 92 Specification for Dma_GetVersionInfo API

Syntax	void Dma_GetVersionInfo (Std_VersionInfoType * const VersionInfoPtr)	
Service ID	0x12	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	VersionInfoPtr	Pointer where the version information of this Driver is stored
Parameters (in - out)	-	-
Return	void	-
Description	This function provides the version information of the DMA driver	
Source	IFX	

DMA driver

Table 92 **Specification for Dma_GetVersionInfo API (continued)**

Error handling	DET: DMA_E_NULL_POINTER: NULL pointer Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaGetVersionInfoApiConfiguration,DmaDevErrorDetect,ModuleId,VendorId,SwPatchVersion,SwMinorVersion,SwMajorVersion
User hints	-

1.3.3.19 Dma_GetChannelEvent

Table 93 **Specification for Dma_GetChannelEvent API**

Syntax	uint32 Dma_GetChannelEvent (const uint8 Channel)	
Service ID	0x13	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	Channel Id
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	Status word containing consolidated channel events
Description	This function queries the channel event register and returns events that are found to have occurred. The type of the events returned by the API is of the type 'Dma_ChEventType'. Users of this API are required to use this enumerator to determine the type of the event.	
Source	IFX	

DMA driver

Table 93 **Specification for Dma_GetChannelEvent API (continued)**

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.20 Dma_GetMoveEngineEvent

Table 94 **Specification for Dma_GetMoveEngineEvent API**

Syntax	uint32 Dma_GetMoveEngineEvent (const uint8 Channel)	
Service ID	0x14	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	Channel Id
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	Status word containing consolidated move engine events
Description	This function receives a channel number and determines if asserted move engine events if any are due to the specified channel. If the specified channel and the channel identified by the move engine are the same, then the driver consolidates and returns asserted move engine errors. Otherwise, a value of 0 is returned to the caller.	
Source	IFX	

DMA driver
Table 94 Specification for Dma_GetMoveEngineEvent API (continued)

Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_NOT_INITIALIZED: A channel API invoked on a channel before initialization of that channel DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	DmaDevErrorDetect
User hints	-

1.3.3.21 Dma_MEStatusClear
Table 95 Specification for Dma_MEStatusClear API

Syntax	Std_ReturnType Dma_MEStatusClear (const Dma_MoveEngineListType MoveEngineId)	
Service ID	0x15	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	MoveEngineId	The Move Engine number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK if the ME status clear could be done, E_NOT_OK otherwise
Description	This interface may be used by users of DMA channels clear move engine events. The error flags of both the move engines would be cleared by this interface.	
Source	IFX	

DMA driver

Table 95 **Specification for Dma_MEStatusClear API (continued)**

Error handling	DET: DMA_E_MOVE_ENGINE_INVALID_ID: Incorrect channel passed as parameter Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	-
User hints	-

1.3.3.22 Dma_InitCheck

Table 96 **Specification for Dma_InitCheck API**

Syntax	Std_ReturnType Dma_InitCheck (const Dma_ConfigType ConfigPtr)	
Service ID	0x16	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to root data structure of all post build configurations
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	Indication as to whether the init check was successful or not.
Description	This interface is meant to be called by the upper layer using the DMA, to check if the initialization has been successfully performed or not. For the same reason, this interface is expected to be called after the initialization is done. After the checks, the interface would indicate the success or failure in the return value.	
Source	IFX	
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	DmaInitCheckApi, DmaDevErrorDetect	
User hints	-	

DMA driver

1.3.3.23 Dma_GetCrcValue

Table 97 Specification for **Dma_GetCrcValue** API

Syntax	uint32 Dma_GetCrcValue (const uint8 ChannelId, const Dma_CrcType CrcType)	
Service ID	0x17	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	ChannelId	DMA channel ID
	CrcType	Address CRC or Data CRC selection
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	CRC value calculated by the hardware
Description	This API allows the user to read the CRC value calculated by the DMA hardware.	
Source	IFX	
Error handling	<p>DET:</p> <p>DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init</p> <p>DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter</p> <p>DMA_E_CH_ALREADY_INITIALIZED: DMA channel initialization requested despite the channel already initialized</p> <p>DMA_E_INVALID_CRC_TYPE_REQ: Dma_GetCrcValue API requested with an invalid CRC type, which is not either Address CRC or Data CRC</p> <p>DMA_E_CRC_NOT_SUPPORTED: Dma_GetCrcValue API requested for a channel which does not support the CRC calculation</p> <p>DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked</p> <p>DMA_E_NULL_POINTER: NULL pointer</p> <p>DMA_E_DATA_TRANSFER_IN_PROGRESS: Start transfer requested when a data transfer is already in progress</p> <p>DMA_E_FREEZE_STATE: Data transfer start/channel update requested by application during freeze state</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>	
Configuration dependencies	-	

DMA driver

Table 97 **Specification for Dma_GetCrcValue API (continued)**

User hints	The 'Dma_GetCrcValue' API provides the CRC value computed by the DMA hardware. Computing the reference CRC and comparing the calculated CRC and reference CRC should be done by the user.
-------------------	---

1.3.3.24 Dma_GetCurrentTimeStamp

Table 98 **Specification for Dma_GetCurrentTimeStamp API**

Syntax	uint32 Dma_GetCurrentTimeStamp (void)	
Service ID	0x18	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	The current timestamp value
Description	This API allows the user to read the current time stamp in the DMA hardware. This time stamp can be used to compare against the time stamp appended by the DMA when the data is moved.	
Source	IFX	
Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

1.3.3.25 Dma_IsChannelInitDone

Table 99 **Specification for Dma_IsChannelInitDone API**

Syntax	Std_ReturnType Dma_IsChannelInitDone (const uint8 ChannelId)
Service ID	0x19

DMA driver
Table 99 Specification for Dma_IsChannelInitDone API (continued)

Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	ChannelId	The DMA channel ID
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	The initialization status of the channel E_OK - The initialization is done E_NOT_OK - The initialization is not done
Description	This API allows the user to check if the channel being used is in initialized state or not.	
Source	IFX	
Error handling	DET: DMA_E_DRIVER_NOT_INITIALIZED: An API called before invocation of Dma_Init DMA_E_CHANNEL_INVALID_ID: Incorrect channel passed as parameter DMA_E_CORE_CHANNEL_MISMATCH: Channel not configured for the particular core is invoked Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	-	

1.3.4 Notifications and callbacks

There are no notifications and callbacks for DMA.

1.3.5 Scheduled functions

There are no scheduled functions for DMA.

1.3.6 Interrupt service routines

DMA driver

1.3.6.1 Dma_ChInterruptHandler

Table 100 Specification for Dma_ChInterruptHandler API

Syntax	<pre>void Dma_ChInterruptHandler (const uint8 Channel)</pre>	
Service ID	Not Applicable	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Conditionally Reentrant - This API is re-entrant for different channels	
Parameters (in)	Channel	DMA channel number
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	<p>This is the interrupt service routine of a channel interrupt invoked by the interrupt frame (installed in the interrupt vector table). It identifies the cause of the interrupt, clears the status and invokes registered notification routines.</p> <p>Note: If there are no status flags set in the interrupt registers, the notification function would be given the reason as 'unknown reason'. The user can trigger the appropriate error handling.</p>	
Source	IFX	
Error handling	<p>DET: None</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>	
Configuration dependencies	-	
User hints	-	

DMA driver

1.3.6.2 Dma_MEInterruptDispatcher

Table 101 Specification for Dma_MEInterruptDispatcher API

Syntax	<pre>void Dma_MEInterruptDispatcher (void)</pre>	
Service ID	Not Applicable	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Not Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This is the interrupt service routine meant to be called by the interrupt frame (installed in the interrupt vector table). The ISR identifies the move engine responsible for the error interrupt and calls the interrupt handler.	
Source	IFX	
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	-	

1.3.6.3 Dma_MEInterruptHandler

Table 102 Specification for Dma_MEInterruptHandler API

Syntax	<pre>void Dma_MEInterruptHandler (const uint8 lErrorSource, const uint32 lErrorData)</pre>	
---------------	--	--

DMA driver
Table 102 Specification for Dma_MEInterruptHandler API (continued)

Service ID	Not Applicable	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Not Reentrant	
Parameters (in)	lErrorSource lErrorData	The ME number or channel number of error The reason for the error
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This is the interrupt handler routine of the move engine error interrupt. It identifies the cause of the interrupt, clears the status and invokes registered notification routines.	
Source	IFX	
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	-	

1.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

1.3.7.1 Development errors

The following table lists all the development errors reported by the driver.

Note: The following error IDs are also reported as safety errors.

DMA driver
Table 103 Description of development errors reported

Description	Source	Error code and value	Applicable APIs
An API called before invocation of Dma_Init	IFX	DMA_E_DRIVER_NOT_INITIALIZED=1	Dma_GetCurrentTimeStamp, Dma_GetCrcValue, Dma_IsChannelInitDone, Dma_GetMoveEngineEvent, Dma_GetChannelEvent, Dma_ChInterruptDisable, Dma_ChInterruptEnable, Dma_ChStatusClear, Dma_ChIsStatusAsserted, Dma_ChSwitchBuffer, Dma_ChGetRemainingData, Dma_ChStopTransfer, Dma_ChStartTransfer, Dma_ChDisableHardwareTrigger, Dma_ChEnableHardwareTrigger, Dma_ChTransferResume, Dma_ChTransferFreeze, Dma_ChDelnit, Dma_ChUpdate, Dma_ChInit
A channel API invoked on a channel before initialization of that channel	IFX	DMA_E_CHANNEL_NOT_INITIALIZED=2	Dma_GetMoveEngineEvent, Dma_GetChannelEvent, Dma_ChInterruptDisable, Dma_ChInterruptEnable, Dma_ChStatusClear, Dma_ChIsStatusAsserted, Dma_ChSwitchBuffer, Dma_ChGetRemainingData, Dma_ChStopTransfer, Dma_ChStartTransfer, Dma_ChDisableHardwareTrigger, Dma_ChEnableHardwareTrigger, Dma_ChTransferResume, Dma_ChTransferFreeze, Dma_ChDelnit

DMA driver
Table 103 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
Incorrect channel passed as parameter	IFX	DMA_E_CHANNEL_INVALID_ID=3	Dma_GetCrcValue, Dma_IsChannelInitDone, Dma_GetMoveEngineEvent, Dma_GetChannelEvent, Dma_ChInterruptDisable, Dma_ChInterruptEnable, Dma_ChStatusClear, Dma_ChIsStatusAsserted, Dma_ChSwitchBuffer, Dma_ChGetRemainingData, Dma_ChStopTransfer, Dma_ChStartTransfer, Dma_ChDisableHardwareTrigger, Dma_ChEnableHardwareTrigger, Dma_ChTransferResume, Dma_ChTransferFreeze, Dma_ChDeInit, Dma_ChUpdate, Dma_ChInit
No notification function is configured for channel interrupt	IFX	DMA_E_CH_NO_NOTIFICATION_CONFIGURED=18	Dma_ChInterruptEnable
Incorrect channel passed as parameter	IFX	DMA_E_MOVE_ENGINE_INVALID_ID=17	Dma_MEStatusClear
Incorrect channel event	IFX	DMA_E_CHANNEL_INVALID_EVENT=4	Dma_ChInterruptDisable, Dma_ChInterruptEnable, Dma_ChStatusClear, Dma_ChIsStatusAsserted
NULL pointer	IFX	DMA_E_NULL_POINTER=5	Dma_GetCrcValue, Dma_GetVersionInfo, Dma_ChUpdate, Dma_Init
Data transfer start/channel update requested by application during freeze state	IFX	DMA_E_FREEZE_STATE=6	Dma_ChInterruptDisable, Dma_ChInterruptEnable, Dma_ChDisableHardwareTrigger, Dma_ChEnableHardwareTrigger, Dma_GetCrcValue, Dma_ChStartTransfer
Freeze requested when no transfers are pending	IFX	DMA_E_NO_TRANSFERS_PENDING=7	Dma_ChTransferFreeze

DMA driver
Table 103 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
Resume requested without a prior freeze	IFX	DMA_E_NOT_IN_FREEZE_STATE=8	Dma_ChEnableHardwareTrigger, Dma_ChTransferResume
Start transfer requested when a data transfer is already in progress	IFX	DMA_E_DATA_TRANSFER_IN_PROGRESS=9	Dma_GetMoveEngineEvent, Dma_GetChannelEvent, Dma_ChStatusClear, Dma_ChIsStatusAsserted, Dma_ChSwitchBuffer, Dma_ChDisableHardwareTrigger, Dma_ChEnableHardwareTrigger, Dma_GetCrcValue, Dma_ChGetRemainingData, Dma_ChDeInit, Dma_ChUpdate, Dma_ChInterruptEnable, Dma_ChInterruptDisable, Dma_ChStartTransfer
DMA driver initialization requested despite the driver already initialized	IFX	DMA_E_ALREADY_INITIALIZED=10	Dma_Init
DMA channel initialization requested despite the channel already initialized	IFX	DMA_E_CH_ALREADY_INITIALIZED=11	Dma_GetCrcValue, Dma_ChInit
Timeout detected while waiting for a certain value of critical register bit fields	IFX	DMA_E_TIMEOUT=12	Dma_ChStopTransfer, Dma_ChTransferFreeze
Switch Buffer API invoked despite the feature not configured	IFX	DMA_E_DATA_CHANNEL_INVALID_SWITCH_REQ=13	Dma_ChSwitchBuffer
Dma_GetCrcValue API requested for a channel which does not support the CRC calculation	IFX	DMA_E_CRC_NOT_SUPPORTED=16	Dma_GetCrcValue
Dma_GetCrcValue API requested with an invalid CRC type, which is not either Address CRC or Data CRC	IFX	DMA_E_INVALID_CRC_TYPE_REQ=15	Dma_GetCrcValue
Dma_ChUpdate API requested for Shadow register update while ADICR is configured in non-shadow configuration	IFX	DMA_E_INVALID_SHADOW_CONFIG_REQ=14	Dma_ChUpdate

DMA driver
Table 103 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
Channel not configured for the particular core is invoked	IFX	DMA_E_CORE_NOT_CONFIGURED=100	Dma_Init
Channel not configured for the particular core is invoked	IFX	DMA_E_CORE_CHANNEL_MISMATCH=101	Dma_GetCrcValue, Dma_IsChannelInitDone, Dma_ChDeInit, Dma_ChInterruptDisable, Dma_GetChannelEvent, Dma_GetMoveEngineEvent, Dma_ChStartTransfer, Dma_ChStopTransfer, Dma_ChGetRemainingData, Dma_ChSwitchBuffer, Dma_ChIsStatusAsserted, Dma_ChInterruptEnable, Dma_ChStatusClear, Dma_ChUpdate, Dma_ChTransferFreeze, Dma_ChTransferResume, Dma_ChEnableHardwareTrigger, Dma_ChDisableHardwareTrigger, Dma_ChInit
A slave core initialization is attempted, before initializing the master core.	IFX	DMA_E_MASTER_UNINIT=102	Dma_Init

1.3.7.2 Production errors

The module does not report any production errors.

1.3.7.3 Safety errors

The module does not report any safety errors.

1.3.7.4 Runtime errors

The module does not report any runtime errors.

1.3.8 Deviations and limitations

The section describes the deviations and limitations from software specification.

1.3.8.1 Deviations

The section describes the deviations from software specification.

DMA driver

Table 104 Known deviations

Reference	Deviation
DMA driver writes into the interrupt control registers for the GPSR interrupts.	DMA driver requires accessing the SRC registers of the Interrupt Router in a multicore scenario. The access is needed to route the resource partition interrupts to the appropriate CPU core when an error event occurs.
De-initialization of DMA not supported	The DMA driver does not support the driver de-initialization feature. Note: De-initialization of individual DMA channels is supported by the driver.

1.3.8.2 Limitations

The section describes the limitations from software specification.

Table 105 Known limitations

Reference	Limitation
The last TCS node of a linked list, cannot be configured as 'autostart'.	This limitation is applicable only if the linked list feature is being used in the autostart mode. If the linked list feature with autostart is used, the last node must not be configured as autostart. For more information, refer to Chapter 21.3.4.9.3 in the Target Specification.
Global addresses should be used when specifying the addresses to be used by DMA.	When specifying the addresses for the DMA transactions, the addresses to be used should be in the global address range and not in the local address range, as the DMA hardware can handle only the global addresses. User should ensure this when configuring the DMA channels.

1.3.9 Unsupported hardware features

This section lists all the AURIX™ DMA hardware features that are not supported by the driver. These features are out of scope of the DMA driver specifications.

Hardware features not supported:

- Pattern detection feature and the corresponding interrupts
- Conditional linked list
- Activation of error interrupts using Error Interrupt Set Register

[cover parentID DMA={76A849BA-6602-4f03-905B-DE829AA3954A}]

2 FlsLoader driver

2.1 User information

2.1.1 Description

Two types of non-volatile memory (NVM) are instantiated in the 2nd Generation AURIX™ (TC3xx) microcontroller.

- Program Flash (PFlash) stores the program code and constant data
- Data Flash (DFlash) stores the application specific data

The FLSLOADER driver provides the following services:

- Initialization and de-initialization of Flash
- Writing the program and data to Flash
- Erasing the contents of Flash
- Locking and unlocking of Flash

The above-mentioned services of the driver are operable on DFlash bank 0 and all PFlash banks of the microcontroller. All reference to DFlash and PFlash, in this Chapter, are meant for bank 0 of the DFlash and all banks of the PFlash, respectively. The driver is delivered as a pre-compile variant. Therefore, the driver supports configuration parameters with only pre-compile configuration class.

2.1.2 Hardware-software mapping

This section describes the system view of the driver and peripherals administered by it.

FlsLoader driver

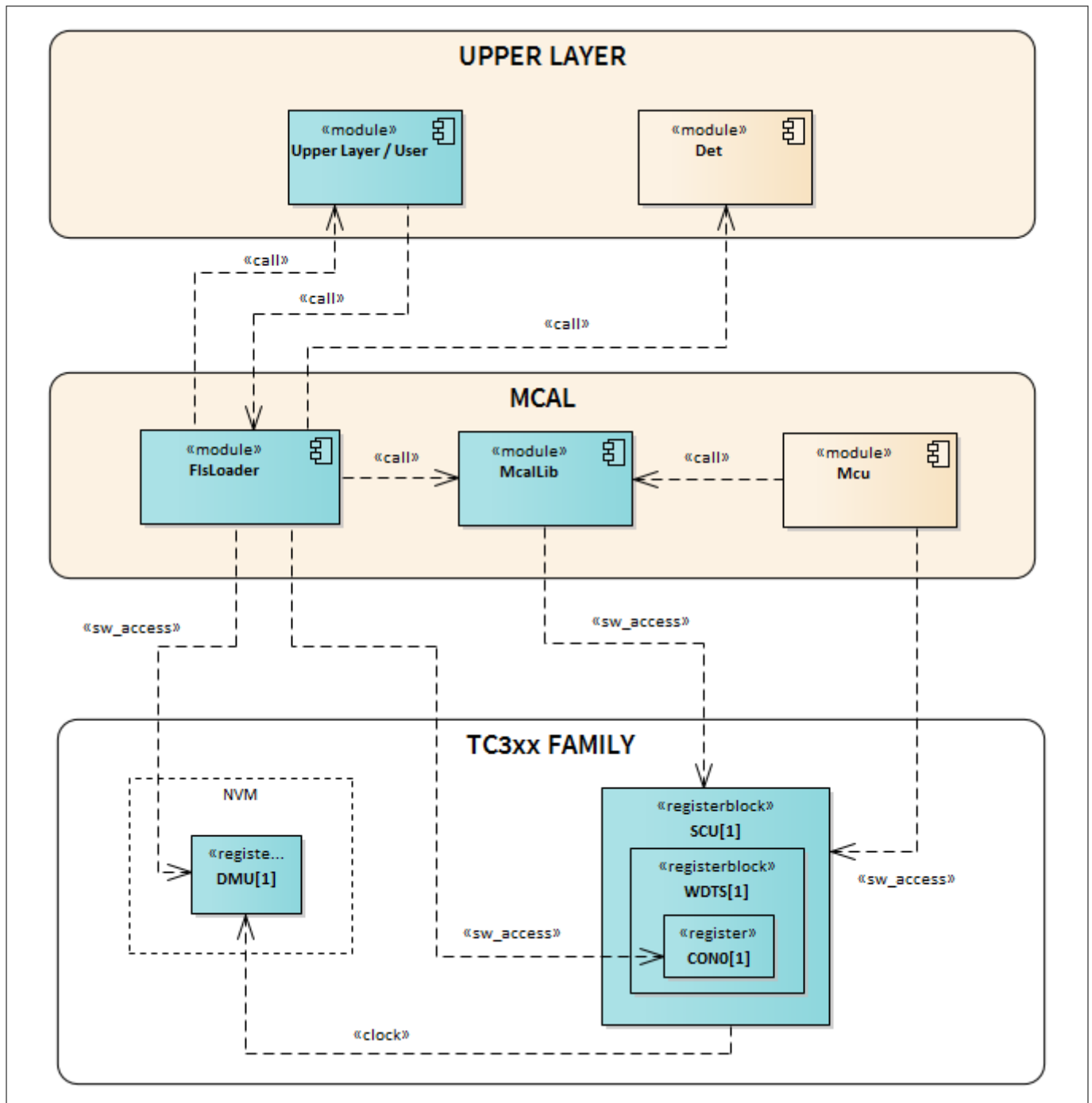


Figure 5 Mapping of hardware-software interfaces

2.1.2.1 SCU: dependent hardware peripheral

Hardware functional features

- Clock frequency for DMU depends on the clock configuration in SCU
- Safety watchdog timer (WDT) for safety ENDINIT protection

Users of the hardware

- All drivers are affected by the clock system. To avoid resource conflict, only MCU driver modifies the SCU
- The FLSLOADER driver accesses the safety WDT to disable the safety ENDINIT protection temporarily for PFlash write and erase operations.

FlsLoader driver**Hardware diagnostic features**

Not applicable.

Hardware events

Not applicable.

2.1.2.2 DMU: primary hardware peripheral**Hardware functional features**

- Write and erase to PFlash:
 - 32 bytes page programming and 256 bytes burst programming
 - Erase by multi-sector erase commands
- Write and erase to DFlash 0 and UCB:
 - 8 bytes page programming and 32 bytes burst programming
 - Erase by multi-sector erase commands
- Single-ended mode support for DFlash 0
- Password-based protection of PFlash and DFlash 0 banks through UCBs

Users of the hardware

The FLSLOADER driver accesses the DMU to erase and program the PFlash, DFlash and UCBs.

Hardware diagnostic features

- Erase verify error (EVER): this flag is set by the erase commands when there is a erase verification error
- Program verify error (PVER): this flag is set by the program commands when there is a program verification error
- Protection error (PROER): this flag is set by hardware when write or erase command executed on protected memory section
- Operation error (OPER): this flag is set by hardware when Flash standard interface (FSI) encounters any error
- Sequence error (SQER): this flag is set by hardware when improper DMU command sequences are executed

Hardware events

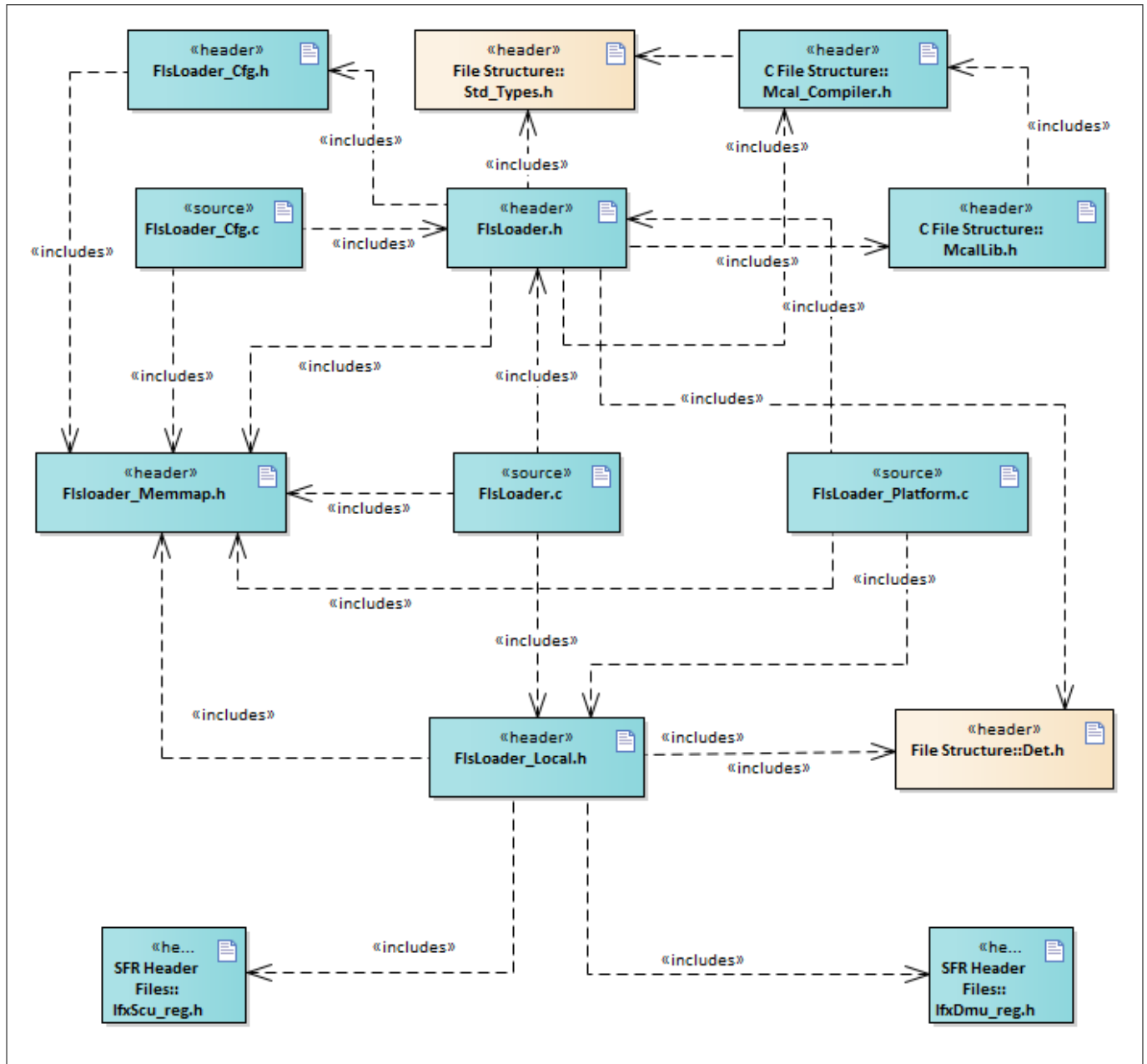
Not applicable.

Features of the hardware not used

- Compliment sensing mode for DFlash 0
- Suspend, resume operations
- ECC error reporting to safety management unit (SMU)
- Interrupt service requests

2.1.3 File structure**2.1.3.1 C file structure**

This section provides details on the C files of the FLSLOADER driver.

FlsLoader driver

Figure 6 C file structure
Table 106 C file structure

File name	Description
FlsLoader.h	FLSLOADER driver header definition file
FlsLoader.c	Implementation of FLSLOADER driver functionality
FlsLoader_Cfg.h	FLSLOADER driver configuration generated out of ECUC file
FlsLoader_Local.h	FLSLOADER driver local header definition file
FlsLoader_Platform.c	FLSLOADER driver platform specific source file
Flsloader_Memmap.h	Mapping of code and data (variables, constant variables) used by FLSLOADER driver to specific memory sections

FlsLoader driver

Table 106 C file structure (continued)

File name	Description
Std_Types.h	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.
Det.h	Provides the exported interfaces of Development Error Tracer
McalLib.h	Header file (Static) defining prototypes of data structures and APIs of end-init and delay services and included by McalLib.c
Mcal_Compiler.h	The file exports Mcal compiler specific functions and macros
IfxDmu_reg.h	SFR header file for Dmu
IfxScu_reg.h	SFR header file for Scu
FlsLoader_Cfg.c	Pre-compile configuration data file for the FLSLOADER driver functionality

2.1.3.2 Code generator plugin files

This section provides details on the code generator plugin files of the FLSLOADER driver.


Figure 7 Code generator plugin files
Table 107 Code generator plugin files

File name	Description
MANIFEST.MF	Tresos plugin support file containing the meta-data for FLSLOADER driver
plugin.xml	Tresos plugin support file for the FLSLOADER driver
anchors.xml	AUTOSAR format module description file
plugin.properties	Tresos plugin support file for the FLSLOADER driver

FlsLoader driver

Table 107 **Code generator plugin files (continued)**

File name	Description
FlsLoader_Bswmd.arxml	AUTOSAR format module description file
FlsLoader.xdm	Tresos format XML data model schema file
FlsLoader.bmd	Code template macro file for FLSLOADER driver
FlsLoader_Catalog.xml	AUTOSAR format catalog file

2.1.4 Integration hints

This section describes the key points that the integrator or user of the FLSLOADER driver must consider. In general the APIs of FLSLOADER are synchronous in nature and is designed to work on single core only.

Flash reprogramming

- Complete driver code or write and erase APIs can be placed in RAM or in another Flash bank for which Flash operations are not being executed.
For example, if the erase or write operations need to be executed on PFlash bank 0 sectors, place the code in either PFlash bank 1 to 5 or Ram.
- The driver does not provide any APIs or functions for moving the write and erase APIs to Ram. User of this API has to take care in the application.

Safety watchdog configuration

- Application shall configure the Safety watchdog timer in static and time independent password mode.

2.1.4.1 Stub modules

This section describes the modules required to integrate the FLSLOADER driver.

EcuM

The FLSLOADER driver is designed for boot loader application which does not contain EcuM.

The application software shall ensure that the initialization and de-initialization of the driver are invoked from the CPU core which intends to use its services.

Memory mapping

The variables, constants and code of the FLSLOADER driver are encapsulated in specific memory section macros defined in the Flsloader_MemMap.h. The naming of the memory sections are AUTOSAR compliant.

FlsLoader driver

The user must place appropriate compiler pragmas to ensure the memory mapped macros are located to the correct memory space. Following example code shows sample memory mapping:

```
#define MEMMAP_ERROR

/** GLOBAL DATA **/
#if defined FLSLOADER_START_SEC_VAR_CLEARED_QM_LOCAL_8
/*****User pragmas here for DSPR of CPU core*****/
#undef FLSLOADER_START_SEC_VAR_CLEARED_QM_LOCAL_8
#undef MEMMAP_ERROR
#elif defined FLSLOADER_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
/*****User pragmas here for DSPR of CPU core*****/
#undef FLSLOADER_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
#undef MEMMAP_ERROR
#elif defined FLSLOADER_START_SEC_VAR_CLEARED_QM_LOCAL_32
/*****User pragmas here for DSPR of CPU core*****/
#undef FLSLOADER_START_SEC_VAR_CLEARED_QM_LOCAL_32
#undef MEMMAP_ERROR
#elif defined FLSLOADER_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
/*****User pragmas here for DSPR of CPU core*****/
#undef FLSLOADER_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
#undef MEMMAP_ERROR

/** CONSTANT DATA **/
#elif defined FLSLOADER_START_SEC_CONST_QM_LOCAL_8
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_START_SEC_CONST_QM_LOCAL_8
#undef MEMMAP_ERROR
#elif defined FLSLOADER_STOP_SEC_CONST_QM_LOCAL_8
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_STOP_SEC_CONST_QM_LOCAL_8
#undef MEMMAP_ERROR
#elif defined FLSLOADER_START_SEC_CONST_QM_LOCAL_32
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_START_SEC_CONST_QM_LOCAL_32
#undef MEMMAP_ERROR
#elif defined FLSLOADER_STOP_SEC_CONST_QM_LOCAL_32
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_STOP_SEC_CONST_QM_LOCAL_32
#undef MEMMAP_ERROR

/** CODE DATA **/
#elif defined FLSLOADER_START_SEC_CODE_QM_LOCAL
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_START_SEC_CODE_QM_LOCAL
#undef MEMMAP_ERROR
#elif defined FLSLOADER_STOP_SEC_CODE_QM_LOCAL
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_STOP_SEC_CODE_QM_LOCAL
#undef MEMMAP_ERROR
#elif defined FLSLOADER_START_SEC_WRITEERASE_CODE_QM_LOCAL
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_START_SEC_WRITEERASE_CODE_QM_LOCAL
```

FlsLoader driver

```
#undef MEMMAP_ERROR
#elif defined FLSLOADER_STOP_SEC_WRITEERASE_CODE_QM_LOCAL
/*****User pragmas here for PFlash*****/
#undef FLSLOADER_STOP_SEC_WRITEERASE_CODE_QM_LOCAL
#undef MEMMAP_ERROR
#endif

#if defined MEMMAP_ERROR
#error "Flsloader_MemMap.h, wrong pragma command"

#endif
```

DET

The complete DET module is not provided as a standard MCAL module but rather provided as a stub module. The user of the FLSLOADER driver must process all the errors reported to the DET module through the `Det_ReportError()` API. The `Det_ReportError()` API is provided in the non-productive `Det.c` and `Det.h` files.

DEM

The FLSLOADER driver does not define production errors.

SchM

The FLSLOADER driver does not define SchM sections.

Safety error

The FLSLOADER driver does not report safety errors.

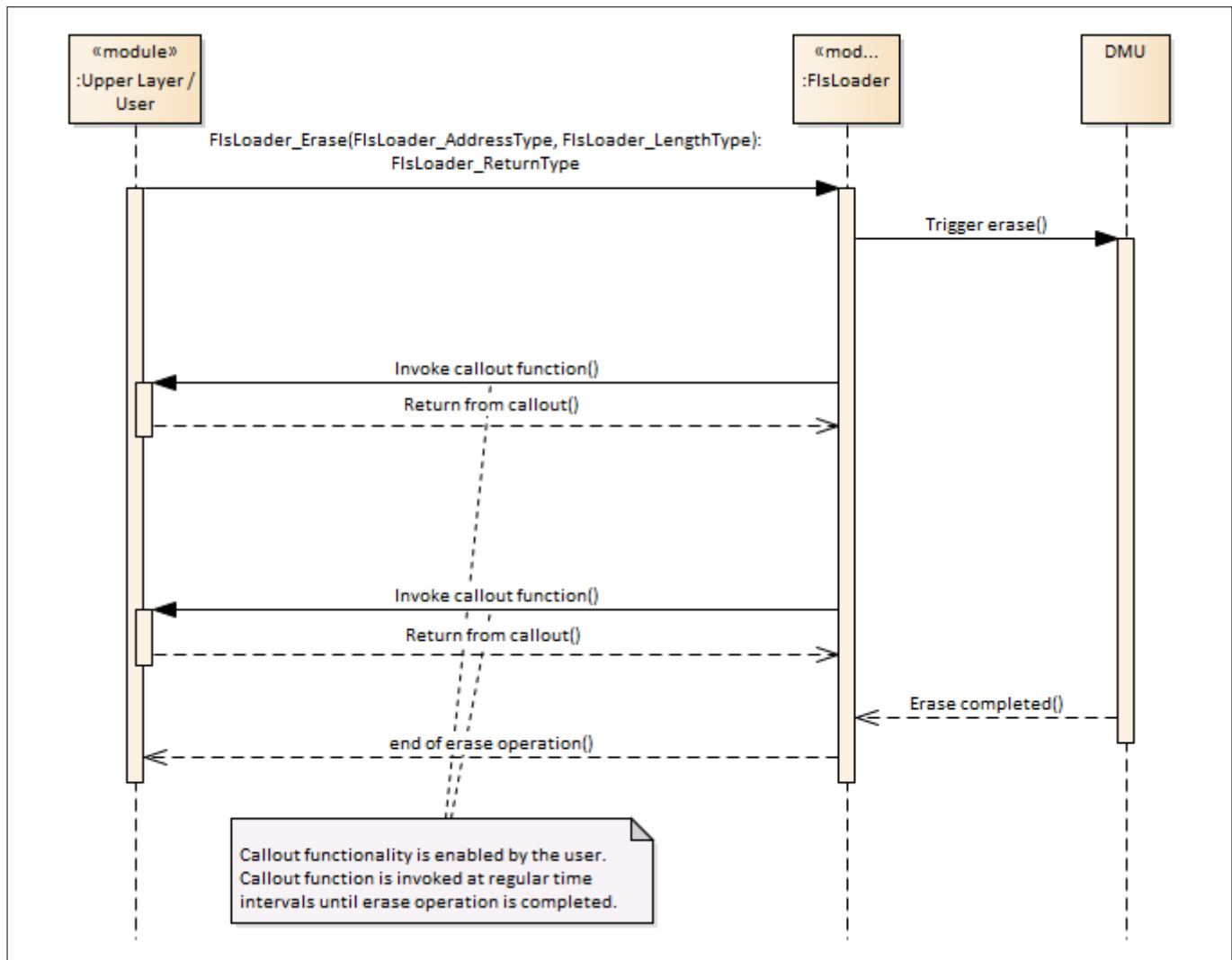
Callout

The FLSLOADER driver provides optional callout function to application while looping for hardware busy status during write and erase operations.

To enable the callout function, user shall define callout function name using configuration `FlsLoaderCallOutFunction` and a regular time interval using configuration `FlsLoaderCallOutTime`.

If enabled, the callout function is invoked at regular time intervals during FLSLOADER write and erase operations. Application shall define the callout function in application software.

Following sequence diagram shows callout functionality during erase operation. Similarly callout shall be invoked by the driver during write and lock operations.

FlsLoader driver

Figure 8 Erase operation with callout enabled
Operating system

The FLSLOADER driver does not depend on OS for its operation.

Notifications and callbacks

The FLSLOADER driver does not implement notifications and callbacks.

2.1.4.2 Multicore and Resource Manager

The FLSLOADER driver supports execution of APIs from single core. User need not explicitly select the NVM resources required by the driver. All required resources are enabled by default through the Resource Manager module.

The locating of constants, variables and code to correct memory space shall be done by the user. As the driver operates from single core, all the sections may be located to the PFlash and DSPR of the same CPU core.

2.1.4.3 MCU support

The FLSLOADER driver depends on the MCU initialization for proper configuration of clock sources to the Flash unit. Therefore, the `Mcu_Init()` API shall be invoked before using the driver services.

FlsLoader driver**2.1.4.4 Port support**

Not applicable for the FLSLOADER driver.

2.1.4.5 DMA support

The FLSLOADER driver does not use DMA for its operations.

2.1.4.6 Interrupt connections

The FLSLOADER driver does not configure interrupts for its operations. All write and erase functions are implemented using the polling method.

FlsLoader driver**2.1.4.7 Example usage****Configuration**

Refer to *Section 2.3*, in the CD User Manual, for all EB tresos configuration interfaces of the FLSLOADER module.

Initialization of the driver

Initialization of the FLSLOADER driver is done by calling `FlsLoader_Init` API with the NULL pointer. The following code listing shows call to `FlsLoader_Init` API which initializes the FLSLOADER driver.

```
print_f(" FlsLoader_Init Execution start.. ");

retval = FlsLoader_Init(NULL_PTR);

/* FlsLoader Initialized successfully */
if (retval == FLSLOADER_E_OK)
{
    print_f(" FlsLoader_Init returned FLSLOADER_E_OK. expected output.");
}
```

FLSLOADER erase and write

The main APIs of FLSLOADER driver are `FlsLoader_Erase` and `FlsLoader_Write`.

Note: If callout function is enabled by the user, the driver invokes the callout function to application during operations of these APIs.

Erase and write an user configuration block (UCB): User can update an UCB using erase and write APIs.

For single UCB erase, start address should be a valid UCB sector address and length should be 1 sector.

For single UCB write, start address should be a valid UCB sector address, length should be size of a UCB sector (512 bytes) and data should contain valid confirmation code on specified offset (refer to TS v2.5.1).

Note: User can update multiple UCBs by giving length as multiple of UCB sector.

FlsLoader driver

The following code listing shows usage of erase and write APIs for DFlash.

```
print_f(" Erase operation start.. ");
print_f(" Erase 2 sectors starting from 0xA0600000U");

/* Erase Start address */
FlsLoader_Address = 0xA0600000U;

/*Erase Length*/
EraseLength = 2U;

retval = FlsLoader_Erase((uint32)FlsLoader_Address, EraseLength);

/* Erase executed successfully */
if(retval == FLSLOADER_E_OK)
{
    print_f(" Erase Execution Passed ");
    print_f(" Programming operation start.. ");
    print_f(" Writing 1024byte data to 0xA0600000U ");

    /*Write Start Address*/
    FlsLoader_Address = 0xA0600000U;

    /*Write Length*/
    WriteLength = 1024U;

    /* Buffer[1024] is source data buffer*/
    retval = FlsLoader_Write((uint32)FlsLoader_Address, WriteLength, &Buffer[0]);

    /* If write is successful */
    if(retval == FLSLOADER_E_OK)
    {
        print_f(" Write Execution Passed ");
    }
}
```

FLSLOADER lock and unlock

DFlash can be configured with Read and Write protection for complete bank.

PFlash can be configured with Write protection, One Time protection and Write Once protection for each PFlash sectors.

Note: If callout function is enabled by the user, the driver invokes the callout function to application during operation of FlsLoader_Lock() API.

If the requirement is to enable the following:

- Write protection for DFlash bank 0
- Write protection for sector 0 of PFlash bank 1,
- OTP protection for sector 0 of PFlash bank 2 and
- WOP protection for sector 0 of PFlash bank3

Then the following configuration should be made in the EB tresos tool:

FlsLoader driver

FlsLoaderDevErrorDetect-> true

FlsLoaderDelnitApi-> true

FlsLoaderLockUnlockApi-> true

FlsLoaderEnableLockCheck->true

FlsLoaderDF0Prot = WRITE_PROTECTION

FlsLoaderPF0WriteProt-> NO_PROTECTION

FlsLoaderPF1WriteProt-> WRITE_PROTECTION

FlsLoaderPF1Sector<0> / FlsLoaderPFSectorWriteProtection = WRITE_PROTECTION

FlsLoaderPF2WriteProt-> OTP_PROTECTION

FlsLoaderPF2Sector<0> / FlsLoaderPFSectorWriteProtection = OTP_PROTECTION

FlsLoaderPF3WriteProt-> WOP_PROTECTION

FlsLoaderPF3Sector<0> / FlsLoaderPFSectorWriteProtection = WOP_PROTECTION

FlsLoaderPF4WriteProt-> NO_PROTECTION

FlsLoaderPF5WriteProt-> NO_PROTECTION

Configure 4 double words FlsLoaderPFUcbPW<x>_<y> with the correct password.

Configure 4 double words FlsLoaderDF0UcbPW<x>_<y> with the correct password.

And then initialize the FLSLOADER. Now Lock and Unlock sequence can be called as shown in the code listings:

FlsLoader driver

Following code listing shows the call to Lock API.

```
print_f(" FlsLoader_Lock Execution start.. ");

retval = FlsLoader_Lock();

/*Lock executed successfully*/
if(retval == FLSLOADER_E_OK)
{
    print_f(" FlsLoader_Lock Execution Passed ");
    print_f(" Trigger micro-controller reset for HW to install protections");
}

/* Following part should be executed after micro-controller reset */

print_f(" Erasing the Sector-0 of PFlash Bank-2 where write Protection is
Enabled ");
print_f(" API should return FLSLOADER_E_LOCKED ");

/*Address of Sector 0 of PFlash bank 2*/
FlsLoader_Address = 0xA0600000U;

/*Number of sectors to be erased is 1*/
Length = 1U;

retval = FlsLoader_Erase(FlsLoader_Address, Length);

/*If Erase operation returned with FLSLOADER_E_LOCKED */
if (retval == FLSLOADER_E_LOCKED)
{
    print_f(" FlsLoader_Erase returned FLSLOADER_E_LOCKED. Expected output.");
}
```

FlsLoader driver

In the following sample the code is executed for Unlock API.

```
print_f(" FlsLoader_Unlock Execution start.. ");
print_f(" Unlocking the Write Protections installed for PFlash sectors.");

/* PFLASH_ORIG ucb address*/
FlsLoader_Address = 0xAF402000U;

/*PfPassword[8] is 8-word password for PFlash */
retval = FlsLoader_Unlock(FlsLoader_Address, &PfPassword[0]);

/* PFlash Write protection Disabled */
if(retval == FLSLOADER_E_OK)
{
    print_f(" FlsLoader_Unlock Execution Passed");
    print_f(" Erasing the Sector-0 of PFlash Bank-2 where write Protection is Disabled.");
    print_f(" API should return FLSLOADER_E_OK ");

    /*Address of Sector 0 of PFlash bank 2*/
    FlsLoader_Address = 0xA0600000U;

    /*Number of sectors to be erased is 1*/
    Length = 1U;

    retval = FlsLoader_Erase(FlsLoader_Address, Length);

    /* Erase executed successfully */
    if (retval == FLSLOADER_E_OK)
    {
        print_f(" FlsLoader_Erase returned FLSLOADER_E_OK. expected output.");
    }
}
```

FLSLOADER de-initialization

De-Initialization of FLSLOADER driver is done by calling the FlsLoader_DeInit () API.

```
print_f(" FlsLoader_DeInit Execution start.. ");

retval = FlsLoader_DeInit();

/* FlsLoader de-initialized successfully */
if (retval == FLSLOADER_E_OK)
{
    print_f(" FlsLoader_DeInit returned FLSLOADER_E_OK. expected output.");
}
```

2.1.5 Key architectural considerations

2.1.5.1 User mode support by the driver

The FLSLOADER driver does not support user mode configuration for any of its APIs as the driver is meant to be used in the boot loader application for Flash programming. Boot loader is not a part of the MCAL runtime component and does not require controlling dynamic runtime operations. Therefore, all APIs of the driver shall be executed in the Supervisor mode.

2.2 Assumptions of Use (AoUs)

There are no AoUs for the driver.

FlsLoader driver

2.3 Reference information

2.3.1 Configuration interfaces

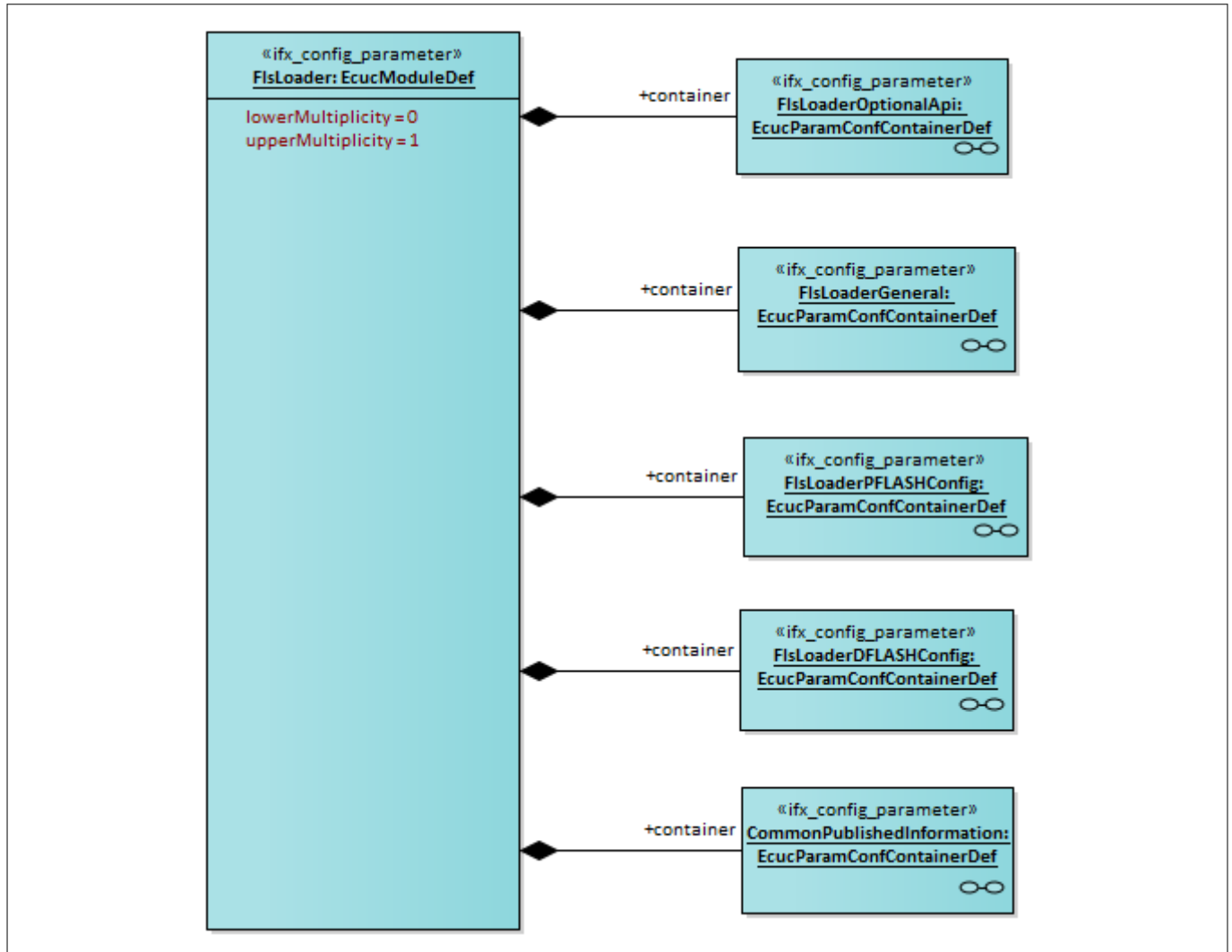


Figure 9 Container hierarchy along with their configuration parameters

2.3.1.1 Container: CommonPublishedInformation

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.1.1 ArMajorVersion

Table 108 Specification for ArMajorVersion

Name	ArMajorVersion		
Description	This parameter provides the major version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef

FlsLoader driver

Table 108 **Specification for ArMajorVersion (continued)**

Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.1.2 **ArMinorVersion**

Table 109 **Specification for ArMinorVersion**

Name	ArMinorVersion		
Description	This parameter provides the minor version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.1.3 **ArPatchVersion**

Table 110 **Specification for ArPatchVersion**

Name	ArPatchVersion		
Description	This parameter provides the patch version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	Post-build variant multiplicity	-

FlsLoader driver
Table 110 Specification for ArPatchVersion (continued)

Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.1.4 ModuleId
Table 111 Specification for ModuleId

Name	ModuleId		
Description	This parameter provides the Module ID.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	255		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.1.5 Release
Table 112 Specification for Release

Name	Release		
Description	The configuration parameter defines the TC3xx derivative used for the implementation.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	String		
Default value	As per selected TC3xx derivative		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

FlsLoader driver

2.3.1.1.6 SwMajorVersion

Table 113 Specification for SwMajorVersion

Name	SwMajorVersion		
Description	This parameter provides the major version of the software.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.1.7 SwMinorVersion

Table 114 Specification for SwMinorVersion

Name	SwMinorVersion		
Description	This parameter provides the minor version of the software.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.1.8 SwPatchVersion

Table 115 Specification for SwPatchVersion

Name	SwPatchVersion		
Description	This parameter provides the patch version of the software.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		

FlsLoader driver
Table 115 Specification for SwPatchVersion (continued)

Default value	As per Driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.1.9 VendorId
Table 116 Specification for VendorId

Name	VendorId		
Description	This parameter provides the Vendor ID.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.2 Container: FlsLoader

Configuration of the FLSLOADER driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.3 Container: FlsLoaderDFlash0ProtConfig

Container for configuration of DFlash bank 0

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

FlsLoader driver
2.3.1.3.1 FlsLoaderDF0Prot
Table 117 Specification for FlsLoaderDF0Prot

Name	FlsLoaderDF0Prot		
Description	Configures DFlash bank 0 protection. Any active protection for the bank shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection READ_PROTECTION: Read protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.3.2 FlsLoaderDF0UcbPW0_0
Table 118 Specification for FlsLoaderDF0UcbPW0_0

Name	FlsLoaderDF0UcbPW0_0		
Description	PW0: First least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

FlsLoader driver

2.3.1.3.3 FlsLoaderDF0UcbPW0_1

Table 119 Specification for FlsLoaderDF0UcbPW0_1

Name	FlsLoaderDF0UcbPW0_1		
Description	PW1: Second least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

2.3.1.3.4 FlsLoaderDF0UcbPW1_0

Table 120 Specification for FlsLoaderDF0UcbPW1_0

Name	FlsLoaderDF0UcbPW1_0		
Description	PW2: Third least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

2.3.1.3.5 FlsLoaderDF0UcbPW1_1

Table 121 Specification for FlsLoaderDF0UcbPW1_1

Name	FlsLoaderDF0UcbPW1_1		
-------------	----------------------	--	--

FlsLoader driver
Table 121 Specification for FlsLoaderDF0UcbPW1_1 (continued)

Description	PW3: Fourth least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

2.3.1.3.6 FlsLoaderDF0UcbPW2_0
Table 122 Specification for FlsLoaderDF0UcbPW2_0

Name	FlsLoaderDF0UcbPW2_0		
Description	PW4: Fifth least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

2.3.1.3.7 FlsLoaderDF0UcbPW2_1
Table 123 Specification for FlsLoaderDF0UcbPW2_1

Name	FlsLoaderDF0UcbPW2_1		
Description	PW5: Sixth least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		

FlsLoader driver
Table 123 Specification for FlsLoaderDF0UcbPW2_1 (continued)

Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

2.3.1.3.8 FlsLoaderDF0UcbPW3_0
Table 124 Specification for FlsLoaderDF0UcbPW3_0

Name	FlsLoaderDF0UcbPW3_0		
Description	PW6: Seventh least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

2.3.1.3.9 FlsLoaderDF0UcbPW3_1
Table 125 Specification for FlsLoaderDF0UcbPW3_1

Name	FlsLoaderDF0UcbPW3_1		
Description	PW7: Eighth least significant 32-bit word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-

FlsLoader driver
Table 125 Specification for FlsLoaderDF0UcbPW3_1 (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderDF0Prot		

2.3.1.4 Container: FlsLoaderDFLASHConfig

This container contains the configuration parameters and sub-containers for configuration of DFlash.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.5 Container: FlsLoaderGeneral

This container contains the general configuration parameters of the FLSLOADER driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.5.1 FlsLoaderCallOutFunction
Table 126 Specification for FlsLoaderCallOutFunction

Name	FlsLoaderCallOutFunction		
Description	<p>FLSLOADER provides a call out function to the user while performing Flash write and erase operations using its APIs. It is invoked at every user defined time rate (FlsLoaderCallOutTime).</p> <p>Call out function should be defined by the user using this parameter. User can configure either the name or the valid address of the call out function in Flash.</p> <p>Call out function is enabled only if valid value as mentioned above is configured for the parameter. By default this parameter contains NULL_PTR and call out function is disabled to reduce the executable code size.</p> <p>When call out is enabled user should configure call out time using parameter FlsLoaderCallOutTime.</p>		
Multiplicity	1..1	Type	EcucFunctionNameDef
Range	String		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL

FlsLoader driver
Table 126 Specification for FlsLoaderCallOutFunction (continued)

Dependency	-
-------------------	---

2.3.1.5.2 FlsLoaderCallOutTime
Table 127 Specification for FlsLoaderCallOutTime

Name	FlsLoaderCallOutTime		
Description	<p>Specifies the maximum time in nanoseconds to wait before invoking call out function to application while looping for status during write and erase operations.</p> <p>The default value of this parameter is set to 5000000 as an example value within the range.</p> <p>This parameter is valid only if FlsLoaderCallOutFuntion is configured with a value other than NULL_PTR.</p>		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	10000 - 4294967295		
Default value	5000000		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderCallOutFunction		

2.3.1.5.3 FlsLoaderDevErrorDetect
Table 128 Specification for FlsLoaderDevErrorDetect

Name	FlsLoaderDevErrorDetect		
Description	<p>Switch for enabling the development error tracer (DET).</p> <p>True: DET enabled</p> <p>False: DET disabled</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

FlsLoader driver
Table 128 Specification for FlsLoaderDevErrorDetect (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.5.4 FlsLoaderEnableLockCheck
Table 129 Specification for FlsLoaderEnableLockCheck

Name	FlsLoaderEnableLockCheck		
Description	Switch for enabling the Lock-check functionality True: Enable the Lock-check routine in write /erase APIs. False: Disable the Lock-check routine in write/erase APIs. This optional feature is disabled by default to reduce the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.6 Container: FlsLoaderOptionalApi

This container contains the configuration parameters for optional APIs of the FLSLOADER driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.6.1 FlsLoaderDeInitApi
Table 130 Specification for FlsLoaderDeInitApi

Name	FlsLoaderDeInitApi		
Description	Switch for enabling the FlsLoader_DeInit API. True: FlsLoader_DeInit API enabled False: FlsLoader_DeInit API disabled This optional API is disabled by default to reduce the executable code size.		

FlsLoader driver

Table 130 **Specification for FlsLoaderDeInitApi (continued)**

Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.6.2 FlsLoaderLockUnlockApi

Table 131 **Specification for FlsLoaderLockUnlockApi**

Name	FlsLoaderLockUnlockApi		
Description	Switch for enabling the FlsLoader_Lock and FlsLoader_Unlock APIs. True: FlsLoader_Lock and FlsLoader_Unlock APIs enabled False: FlsLoader_Lock and FlsLoader_Unlock APIs disabled These optional APIs are disabled by default to reduce the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.6.3 FlsLoaderVersionInfoApi

Table 132 **Specification for FlsLoaderVersionInfoApi**

Name	FlsLoaderVersionInfoApi
-------------	-------------------------

FlsLoader driver
Table 132 Specification for FlsLoaderVersionInfoApi (continued)

Description	Switch for enabling the FlsLoaderVersionInfo API. True: FlsLoaderVersionInfo API enabled False: FlsLoaderVersionInfo API disabled This optional API is disabled by default to reduce the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.7 Container: FlsLoaderPF0Sector

Container for configuration of PFlash bank 0 sectors

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.7.1 FlsLoaderPFSectorWriteProtection
Table 133 Specification for FlsLoaderPFSectorWriteProtection

Name	FlsLoaderPFSectorWriteProtection		
Description	Configuration of PFlash bank 0 sector protection. Any active protection for the sector shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-

FlsLoader driver
Table 133 Specification for FlsLoaderPFSectorWriteProtection (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.8 Container: FlsLoaderPF1Sector

Container for configuration of PFlash bank 1 sectors.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.8.1 FlsLoaderPFSectorWriteProtection
Table 134 Specification for FlsLoaderPFSectorWriteProtection

Name	FlsLoaderPFSectorWriteProtection		
Description	Configuration of PFlash bank 1 sector protection. Any active protection for the sector shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash1WriteProt		

2.3.1.9 Container: FlsLoaderPF2Sector

Container for configuration of PFlash bank 2 sectors.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

FlsLoader driver
2.3.1.9.1 FlsLoaderPFSectorWriteProtection
Table 135 Specification for FlsLoaderPFSectorWriteProtection

Name	FlsLoaderPFSectorWriteProtection		
Description	Configuration of PFlash bank 2 sector protection. Any active protection for the sector shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protection		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash2WriteProt		

2.3.1.10 Container: FlsLoaderPF3Sector

Container for configuration of PFlash bank 3 sectors.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.10.1 FlsLoaderPFSectorWriteProtection
Table 136 Specification for FlsLoaderPFSectorWriteProtection

Name	FlsLoaderPFSectorWriteProtection		
Description	Configuration of PFlash bank 3 sector protection. Any active protection for the sector shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		

FlsLoader driver
Table 136 Specification for FlsLoaderPFSectorWriteProtection (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash3WriteProt		

2.3.1.11 Container: FlsLoaderPF4Sector

Container for configuration of PFlash bank 4 sectors.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.11.1 FlsLoaderPFSectorWriteProtection
Table 137 Specification for FlsLoaderPFSectorWriteProtection

Name	FlsLoaderPFSectorWriteProtection		
Description	Configuration of PFlash bank 4 sector protection. Any active protection for the sector shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash4WriteProt		

2.3.1.12 Container: FlsLoaderPF5Sector

Container for configuration of PFlash bank 5 sectors.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

FlsLoader driver
2.3.1.12.1 FlsLoaderPFSectorWriteProtection
Table 138 Specification for FlsLoaderPFSectorWriteProtection

Name	FlsLoaderPFSectorWriteProtection		
Description	Configuration of PFlash bank 5 sector protection. Any active protection for the sector shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash5WriteProt		

2.3.1.13 Container: FlsLoaderPFlash0ProtConfig

Container for configuration of PFlash bank 0

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.13.1 FlsLoaderPF0UcbPW0_0
Table 139 Specification for FlsLoaderPF0UcbPW0_0

Name	FlsLoaderPF0UcbPW0_0		
Description	PW0: First least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

FlsLoader driver
Table 139 Specification for FlsLoaderPF0UcbPW0_0 (continued)

Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.13.2 FlsLoaderPF0UcbPW0_1
Table 140 Specification for FlsLoaderPF0UcbPW0_1

Name	FlsLoaderPF0UcbPW0_1		
Description	PW1: Second least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.13.3 FlsLoaderPF0UcbPW1_0
Table 141 Specification for FlsLoaderPF0UcbPW1_0

Name	FlsLoaderPF0UcbPW1_0		
Description	PW2: Third least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

FlsLoader driver
2.3.1.13.4 FlsLoaderPF0UcbPW1_1
Table 142 Specification for FlsLoaderPF0UcbPW1_1

Name	FlsLoaderPF0UcbPW1_1		
Description	PW3: Fourth least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.13.5 FlsLoaderPF0UcbPW2_0
Table 143 Specification for FlsLoaderPF0UcbPW2_0

Name	FlsLoaderPF0UcbPW2_0		
Description	PW4: Fifth least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.13.6 FlsLoaderPF0UcbPW2_1
Table 144 Specification for FlsLoaderPF0UcbPW2_1

Name	FlsLoaderPF0UcbPW2_1		
-------------	----------------------	--	--

FlsLoader driver
Table 144 Specification for FlsLoaderPF0UcbPW2_1 (continued)

Description	PW5: Sixth least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.13.7 FlsLoaderPF0UcbPW3_0
Table 145 Specification for FlsLoaderPF0UcbPW3_0

Name	FlsLoaderPF0UcbPW3_0		
Description	PW6: Seventh least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.13.8 FlsLoaderPF0UcbPW3_1
Table 146 Specification for FlsLoaderPF0UcbPW3_1

Name	FlsLoaderPF0UcbPW3_1		
Description	PW7: Eighth least significant word of 256-bit password. Default value is 0 as the initial password is set to 0.		
Multiplicity	1..1	Type	EcucIntegerParamDef

FlsLoader driver
Table 146 Specification for FlsLoaderPF0UcbPW3_1 (continued)

Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	FlsLoaderPFlash0WriteProt		

2.3.1.13.9 FlsLoaderPFlash0WriteProt
Table 147 Specification for FlsLoaderPFlash0WriteProt

Name	FlsLoaderPFlash0WriteProt		
Description	Configuration of PFlash bank 0 protection. Any active protection for the bank shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.14 Container: FlsLoaderPFlash1ProtConfig

Container for configuration of PFlash bank 1

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

FlsLoader driver
2.3.1.14.1 FlsLoaderPFlash1WriteProt
Table 148 Specification for FlsLoaderPFlash1WriteProt

Name	FlsLoaderPFlash1WriteProt		
Description	Configuration of PFlash bank 1 protection. Any active protection for the bank shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.15 Container: FlsLoaderPFlash2ProtConfig

Container for configuration of PFlash bank 2

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.15.1 FlsLoaderPFlash2WriteProt
Table 149 Specification for FlsLoaderPFlash2WriteProt

Name	FlsLoaderPFlash2WriteProt		
Description	Configuration of PFlash bank 2 protection. Any active protection for the bank shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		

FlsLoader driver
Table 149 Specification for FlsLoaderPFlash2WriteProt (continued)

Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.16 Container: FlsLoaderPFlash3ProtConfig

Container for configuration of PFlash bank 3

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.16.1 FlsLoaderPFlash3WriteProt
Table 150 Specification for FlsLoaderPFlash3WriteProt

Name	FlsLoaderPFlash3WriteProt		
Description	Configuration of PFlash bank 3 protection. Any active protection for the bank shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.17 Container: FlsLoaderPFlash4ProtConfig

Container for configuration of PFlash bank 4

Post-Build Variant Multiplicity: -

FlsLoader driver

Multiplicity Configuration Class: -

2.3.1.17.1 FlsLoaderPFlash4WriteProt
Table 151 Specification for FlsLoaderPFlash4WriteProt

Name	FlsLoaderPFlash4WriteProt		
Description	Configuration of PFlash bank 4 protection. Any active protection for the bank shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.18 Container: FlsLoaderPFlash5ProtConfig

Container for configuration of PFlash bank 5

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.18.1 FlsLoaderPFlash5WriteProt
Table 152 Specification for FlsLoaderPFlash5WriteProt

Name	FlsLoaderPFlash5WriteProt		
Description	Configuration of PFlash bank 5 protection. Any active protection for the bank shall be selected by the user as per application need. Therefore default value is provided as no protection.		
Multiplicity	1..1	Type	EcucEnumerationParamDef

FlsLoader driver
Table 152 Specification for FlsLoaderPFlash5WriteProt (continued)

Range	NO_PROTECTION: No protection OTP_PROTECTION: OTP protected WOP_PROTECTION: WOP protected WRITE_PROTECTION: Write protected		
Default value	NO_PROTECTION		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

2.3.1.19 Container: FlsLoaderPFLASHConfig

This container contains the configuration parameters and sub-containers for configuration of PFlash.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.2 Functions - Type definitions
2.3.2.1 FlsLoader_AddressType
Table 153 Specification for FlsLoader_AddressType

Syntax	FlsLoader_AddressType	
Type	uint32	
File	FlsLoader.h	
Range	0 to 4294967295	Target address in Flash
Description	This type specifies the logical destination address of Flash in PFlash or DFlash	
Source	IFX	

2.3.2.2 FlsLoader_CallOutFunc
Table 154 Specification for FlsLoader_CallOutFunc

Syntax	FlsLoader_CallOutFunc
Type	Pointer to a function of type void Function_Name (void)
File	FlsLoader.h
Description	Call out function to application which is called at every user defined time rate

FlsLoader driver

Table 154 Specification for FlsLoader_CallOutFunc (continued)

Source	IFX
---------------	-----

2.3.2.3 FlsLoader_ConfigType

Table 155 Specification for FlsLoader_ConfigType

Syntax	FlsLoader_ConfigType	
Type	void	
File	FlsLoader.h	
Range	None	
Description	This defines the void configuration type as the module supports single configuration variant	
Source	IFX	

2.3.2.4 FlsLoader_LengthType

Table 156 Specification for FlsLoader_LengthType

Syntax	FlsLoader_LengthType	
Type	uint32	
File	FlsLoader.h	
Range	0 to 4294967295	Length information for write and erase operations
Description	This type specifies length information for write and erase operations as following: Write: Number of bytes to be written Erase: Number of logical sectors to be erased	
Source	IFX	

2.3.2.5 FlsLoader_ReturnType

Table 157 Specification for FlsLoader_ReturnType

Syntax	FlsLoader_ReturnType	
Type	uint32	
File	FlsLoader.h	
Range	0 - FLSLOADER_E_OK	Successful execution
	1 - FLSLOADER_E_NOT_OK	Development error occurred
	2 - FLSLOADER_E_LOCKED	Sectors are read/write protected
	3 - FLSLOADER_E_ROMVERSION	All sectors are protected under OTP
	5 - FLSLOADER_E_BUSY	Device is busy

FlsLoader driver

Table 157 Specification for FlsLoader_ReturnType (continued)

Description	This specifies the various Return types that can be specified by the APIs. This type is used for the errors detected by the APIs
Source	IFX

2.3.2.6 FlsLoader_ValueType

Table 158 Specification for FlsLoader_ValueType

Syntax	FlsLoader_ValueType	
Type	uint32	
File	FlsLoader.h	
Range	0 to 4294967295	Password (8 words)
Description	The type specifies values for Flash (PFlash or DFlash) protection password (8 words).	
Source	IFX	

2.3.3 Functions - APIs

This section provides list of all APIs available to upper layer.

2.3.3.1 FlsLoader_DeInit

Table 159 Specification for FlsLoader_DeInit API

Syntax	FlsLoader_ReturnType FlsLoader_DeInit (void)	
Service ID	0x30	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	FlsLoader_ReturnType	FLSLOADER_E_OK: Successful execution. FLSLOADER_E_BUSY: Flash is busy with erase/write operation. FLSLOADER_E_NOT_OK: Development error occurred.
Description	This function de-initializes the Flash module. This Function sets the registers to their default state and executes the reset to read command.	
Source	IFX	

FlsLoader driver
Table 159 Specification for FlsLoader_DeInit API (continued)

Error handling	DET: FLSLOADER_E_BUSY: API service called while driver still busy FLSLOADER_E_UNINIT: APIs are invoked without initialization of the driver Runtime Errors: None DEM: None Safety Errors: None
Configuration dependencies	FlsLoaderDeInitApi
User hints	-

2.3.3.2 FlsLoader_Erase

Table 160 Specification for FlsLoader_Erase API

Syntax	<pre>FlsLoader_ReturnType FlsLoader_Erase (const FlsLoader_AddressType TargetAddress, const FlsLoader_LengthType Length)</pre>	
Service ID	0x32	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	TargetAddress Length	Target address in Flash memory. It should be aligned to the following sector sizes of the selected Flash for erase. PFlash: 16 Kbyte DFlash: 4 Kbyte Number of Flash (PFlash or DFlash) sectors to be erased. Note: Number of sectors should lie within single Flash bank. Erase operation across the Flash banks is not supported.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	FlsLoader_ReturnType	FLSLOADER_E_OK: Successful completion. FLSLOADER_E_BUSY: Flash is busy with erase/write operation. FLSLOADER_E_NOT_OK: DET error, Sequence error, Erase verify error, Program verify error (for PFlash), Protection error or Operation error occurred. FLSLOADER_E_LOCKED: Sector is protected (If FlsLoaderEnableLockCheck is enabled).
Description	This function erases the logical sectors of the internal Flash. The completion of this operation is denoted by clearing of busy status flag or error.	

FlsLoader driver
Table 160 Specification for FlsLoader_Erase API (continued)

Source	IFX
Error handling	DET: FLSLOADER_E_PARAM_ADDRESS: API service called with wrong address FLSLOADER_E_PARAM_LENGTH: API service called with wrong length FLSLOADER_E_UNINIT: APIs are invoked without initialization of the driver FLSLOADER_E_BUSY: API service called while driver still busy Runtime Errors: None DEM: None Safety Errors: None
Configuration dependencies	FlsLoaderEnableLockCheck
User hints	-

2.3.3.3 FlsLoader_GetVersionInfo
Table 161 Specification for FlsLoader_GetVersionInfo API

Syntax	void FlsLoader_GetVersionInfo (Std_VersionInfoType * const VersionInfoPtr)	
Service ID	0x35	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	VersionInfoPtr	Pointer to where the version information has to be stored
Parameters (in - out)	-	-
Return	void	-
Description	This functions provides the version information of the FLSLOADER driver	
Source	IFX	
Error handling	DET: FLSLOADER_E_PARAM_POINTER: API service called with NULL pointer Runtime Errors: None DEM: None Safety Errors: None	
Configuration dependencies	FlsLoaderVersionInfoApi	

FlsLoader driver

Table 161 **Specification for FlsLoader_GetVersionInfo API (continued)**

User hints	None
-------------------	------

2.3.3.4 FlsLoader_Init

Table 162 **Specification for FlsLoader_Init API**

Syntax	FlsLoader_ReturnType FlsLoader_Init (const FlsLoader_ConfigType * const Address)	
Service ID	0x2F	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	Address	NULL pointer because the driver supports single configuration variant
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	FlsLoader_ReturnType	FLSLOADER_E_OK: Successful execution FLSLOADER_E_ROMVERSION: All sectors are protected with OTP or WOP protection FLSLOADER_E_NOT_OK: Development error occurred
Description	This function initializes the Flash module and checks if all the Flash sectors are configured as ROM (OTP or WOP protected)	
Source	IFX	
Error handling	DET: FLSLOADER_E_PARAM_IGNORED: API service called with not a NULL pointer Runtime Errors: None DEM: None Safety Errors: None	
Configuration dependencies	-	
User hints	None	

2.3.3.5 FlsLoader_Lock

Table 163 **Specification for FlsLoader_Lock API**

Syntax	FlsLoader_ReturnType FlsLoader_Lock (void)
Service ID	0x33

FlsLoader driver
Table 163 Specification for FlsLoader_Lock API (continued)

Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	FlsLoader_ReturnType	FLSLOADER_E_OK: Successful completion. FLSLOADER_E_BUSY: Flash is busy with erase/write operation. FLSLOADER_E_NOT_OK: DET error, Sequence error, Protection error, Operation error, Program verify error or Erase verify error occurred. FLSLOADER_E_LOCKED: Flash is already in locked state.
Description	<p>This function locks (protect) the internal PFlash and DFlash 0 of micro-controller with pre-configured protections.</p> <p>Following protection configurations are supported by the driver:</p> <ul style="list-style-type: none"> -DFlash: Read protection, write protection. -DFlash protections are configurable at bank level. -PFlash: Write protection, write once protection (WOP), one time programmable (OTP) protection. -PFlash protections are configurable at sector level. However a bank and its corresponding sectors to be protected shall have same protection configured. 	
Source	IFX	
Error handling	DET: FLSLOADER_E_UNINIT: APIs are invoked without initialization of the driver FLSLOADER_E_BUSY: API service called while driver still busy Runtime Errors: None DEM: None Safety Errors: None	
Configuration dependencies	FlsLoaderLockUnlockApi	
User hints	None	

FlsLoader driver

2.3.3.6 FlsLoader_UnLock

Table 164 Specification for FlsLoader_UnLock API

Syntax	<code>FlsLoader_ReturnType FlsLoader_UnLock (const FlsLoader_AddressType TargetAddress, const FlsLoader_ValueType * const Password)</code>	
Service ID	0x34	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	TargetAddress Password	UCB address of corresponding Flash to be unlocked. 0xAF402000 - PFlash UCB 0xAF402200 - DFlash UCB Pointer to the 4 double word (256 bit) UCB password of corresponding Flash to be unlocked.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	FlsLoader_ReturnType	FLSLOADER_E_OK: Successful completion. FLSLOADER_E_BUSY: Flash is busy with erase/write operation. FLSLOADER_E_NOT_OK: DET error, Operation error, Sequence error or Protection error occurred.
Description	This function is used to unlock the internal PFlash and DFlash 0 of the micro-controller from active protection. It temporarily (until next controller reset) disables the current active read or write protection. A wrong password results in protection error. - DFlash 0 can be unlocked from read and write protections. - PFlash can be unlocked from write protection. WOP and OTP cannot be unlocked.	
Source	IFX	
Error handling	DET: FLSLOADER_E_UNINIT: APIs are invoked without initialization of the driver FLSLOADER_E_PARAM_ADDRESS: API service called with wrong address FLSLOADER_E_BUSY: API service called while driver still busy Runtime Errors: None DEM: None Safety Errors: None	
Configuration dependencies	FlsLoaderLockUnlockApi	
User hints	None	

FlsLoader driver
2.3.3.7 FlsLoader_Write
Table 165 Specification for FlsLoader_Write API

Syntax	<code>FlsLoader_ReturnType FlsLoader_Write (const FlsLoader_AddressType TargetAddress, const FlsLoader_LengthType Length, const uint8 * const SourceAddressPtr)</code>	
Service ID	0x31	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	TargetAddress Length SourceAddressPtr	Target address in Flash memory. It should be aligned to the following page sizes of the selected Flash for write. PFlash: 32 Bytes DFlash: 8 Bytes Number of bytes to be written. It should be multiple of the following page sizes of the selected Flash for write. PFlash: 32 Bytes DFlash: 8 Bytes Pointer to source data buffer
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	FlsLoader_ReturnType	FLSLOADER_E_OK: Successful execution. FLSLOADER_E_BUSY: Flash is busy with erase/write operation. FLSLOADER_E_NOT_OK: DET error, Sequence error, Program verify error, Protection error or Operation error occurred. FLSLOADER_E_LOCKED: Sector is protected (If FlsLoaderEnableLockCheck is enabled).
Description	This function is used to program a page of internal Flash. Sectors of PFlash and DFlash can be programmed.	
Source	IFX	
Error handling	DET: FLSLOADER_E_PARAM_ADDRESS: API service called with wrong address FLSLOADER_E_PARAM_LENGTH: API service called with wrong length FLSLOADER_E_UNINIT: APIs are invoked without initialization of the driver FLSLOADER_E_BUSY: API service called while driver still busy Runtime Errors: None DEM: None Safety Errors: None	
Configuration dependencies	FlsLoaderEnableLockCheck	

FlsLoader driver

Table 165 **Specification for FlsLoader_Write API (continued)**

User hints	None
-------------------	------

2.3.4 Notifications and callbacks

The driver by does not implement any notifications. However, if the callout feature is enabled by user, it provides a call out to application while looping for status during write and erase operations.

2.3.5 Scheduled functions

This driver does not support any scheduled functions.

2.3.6 Interrupt service routines

This driver does not configure any interrupts for its operation.

2.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

2.3.7.1 Development errors

The following table lists all the development errors reported by the driver.

Table 166 **Description of development errors reported**

Description	Source	Error code and value	Applicable APIs
API service called while driver still busy	IFX	FLSLOADER_E_BUSY=5	FlsLoader_Erase, FlsLoader_Write, FlsLoader_UnLock, FlsLoader_Lock, FlsLoader_DeInit
API service called with wrong address	IFX	FLSLOADER_E_PARAM_ADDRESS=3	FlsLoader_Erase, FlsLoader_Write, FlsLoader_UnLock
API service called with not a NULL pointer	IFX	FLSLOADER_E_PARAM_IGNORED=0	FlsLoader_Init
API service called with wrong length	IFX	FLSLOADER_E_PARAM_LENGTH=2	FlsLoader_Erase, FlsLoader_Write
API service called with NULL pointer	IFX	FLSLOADER_E_PARAM_POINTER=6	FlsLoader_GetVersionInfo
APIs are invoked without initialization of the driver	IFX	FLSLOADER_E_UNINIT=4	FlsLoader_DeInit, FlsLoader_Erase, FlsLoader_Write, FlsLoader_UnLock, FlsLoader_Lock

FlsLoader driver**2.3.7.2 Production errors**

The module does not report any production errors.

2.3.7.3 Safety errors

The module does not report any safety errors.

2.3.7.4 Runtime errors

The module does not report any runtime errors.

2.3.8 Deviations and limitations

The section describes the deviations and limitations from software specification.

2.3.8.1 Deviations

Currently there are no deviations for the FLSLOADER driver.

2.3.8.2 Limitations

The section describes the limitations from software specification.

Table 167 Known limitations

Reference	Limitation
PFlash erase	The maximum PFlash sectors that can be erased by erase API is 192. If length is more than 192 sectors, the application should call erase API multiple times as per the erase length.
Write to WOP protected sector	The driver does not support write to the PFlash sectors which are protected with WOP (Write Once Protection).
Callout function usage	Application can use callout to service external watchdog or other similar functionality, however it should not call FLSLOADER APIs inside callout.

2.3.9 Unsupported hardware features

All intended features of FLSLOADER driver are supported.

3 SMU driver

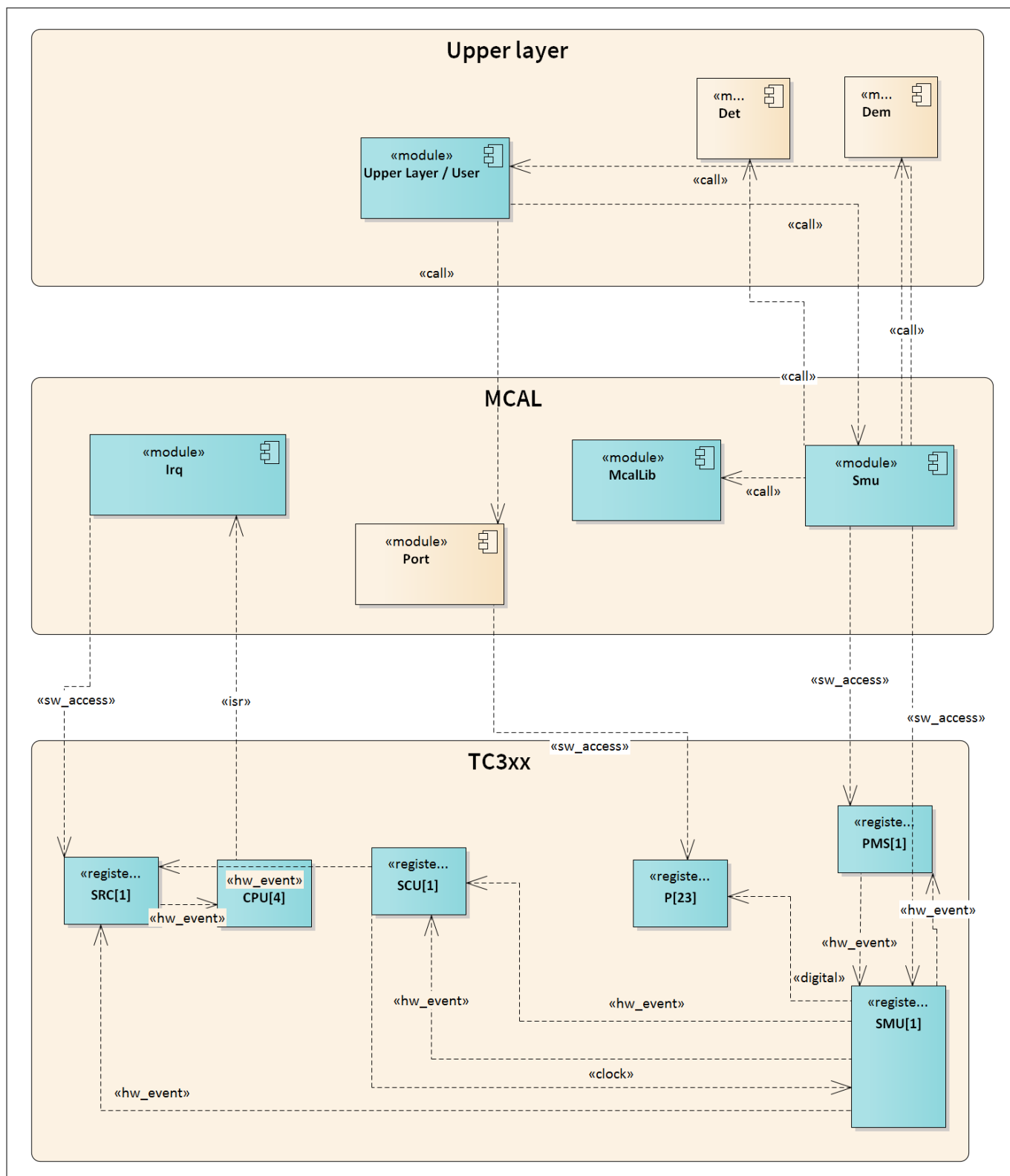
3.1 User information

3.1.1 Description

The SMU driver is an abstraction of the SMU peripheral in the AURIX™ microcontroller family. The SMU peripheral centralizes all the alarm signals related to different hardware safety mechanisms. Each alarm can trigger internal actions and/or notify the presence of faults to the external world through a fault signaling protocol. The SMU driver is active before any of the peripherals are active. Therefore, SMU initialization and de-initialization must be called on only the master core. However, as it encompasses the hardware safety mechanisms distributed across all the cores, SMU runtime services will be accessible from all cores.

3.1.2 Hardware-software mapping

This section describes the system view of the driver and peripherals administered by it.

SMU driver

Figure 10 Mapping of hardware-software interfaces

SMU driver**3.1.2.1 PMS: primary hardware peripheral**

The SMU_stdby operates in fBCK frequency. SMU_stdby has an interface to the PMS. The SMU_stdby is configurable that it can be put to an idle state. All the alarms in fBCK domain would be forwarded to both SMU_stdby and SMU_core.

Hardware functional features**Features supported:**

The alarm status with respect to group 20 and 21 can be queried and cleared. The Smu_core is monitored through a runtime service for any alarms.

Features not supported:

Not applicable.

Users of the hardware

The Smu_stdby in PMS is owned only by the SMU driver.

Hardware diagnostic features

None.

Hardware events

There are no hardware events.

3.1.2.2 SCU: dependent hardware peripheral

The internal alarm reactions resulting from the alarm events are interfaced to the SCU. These interface signals can be of the following types:

- NMI request
- Emergency stop request
- Reset request

Hardware functional features**Features supported:**

The clock (fSPB) is provided by SCU. Also application reset, power on reset, external EMS alarm and watchdog timeout is provided.

Features not supported:

Not applicable.

Users of the hardware

The users of the SCU are the MCAL drivers and applications.

Hardware diagnostic features

Associated SMU alarms exist, which can detect any hardware safety mechanism failures.

Hardware events

SMU sends NMI, Reset, Emergency Stop and Run State requests to SCU.

3.1.2.3 PORT: dependent hardware peripheral**Hardware functional features****Features supported:**

- The port pins are driven to GPIO or SMU mode where SMU uses the Port pins to trigger external reaction mechanism using FSP.
- The emergency stop feature is also supported by the SMU driver.

Features not supported:

SMU driver

None.

Users of the hardware

Port hardware is used by the SMU driver, MCAL and the application. The application shall setup the Port pins (P33.8 and P33.10) to be used by the SMU driver.

Hardware diagnostic features

None.

Hardware events

None.

3.1.2.4 SMU: primary hardware peripheral

The SMU is a central and modular component of the safety architecture providing a generic interface to manage the behavior of the microcontroller under the presence of faults. The SMU centralizes all the alarm signals related to the different hardware and software-based safety mechanisms. Each alarm can be individually configured to trigger internal actions and/or to notify externally the presence of faults through a fault signaling protocol.

SMU is used by the upper layer and provides an abstracted interface to the user to access the SMU peripheral.

The SMU peripheral has two parts: the SMU_core and the SMU_stdby. Both operate in different clock and power domains. One of the most important features of SMU_stdby is to monitor the correct functioning of the SMU_core. The status of any error is triggered as an alarm from SMU_core to SMU_stdby.

Hardware functional features**Features supported:**

- Alarm handling: Setting, querying, clearing of alarms and setup of alarm reactions
- FSP handling: Activation, deactivation of the FSP to request the trigger of safe state
- Port emergency handling: Activation of port emergency stop
- Recovery Timer handling: Setup, status query status of recovery timers
- SMU State Machine: Status query of the SMU state machine
- SMU configuration protection: Activation of permanent lock mechanism
- Smu_ActivatePES() software command
- Register monitoring feature for safety flip flop

Features not supported:

None.

Modes or states of SMU

- The SMU peripheral itself operates in three states, that is, START, RUN and FAULT states. The application provides the services to invoke the transitions between the states.
- The FSP driven by SMU has three states, that is, Power-on, Fault and Fault Free states.

Users of the hardware

The SMU peripheral can be accessed through the SMU complex driver which is part of complex driver layer of AUTOSAR.

Hardware diagnostic features

In case of CMD SFR the STS SFR can be read back to ensure the command has been successfully executed.

Hardware events

The error status of the hardware and software safety mechanisms is indicated by alarms. The reaction for these alarms is determined by the configuration parameters.

SMU driver

SMU is connected to the interrupt router through three service request nodes SRC_SMUx (x = 0 to 2). Each service request can trigger an interrupt on the CPU 0, 1, 2, 3, 4 or 5. The number of CPUs depend on the derivative.

3.1.2.5 SRC: dependent hardware peripheral

Hardware functional features

Internal actions resulting from an alarm event interface to the interrupt router (IR). These are SMU IR request 0, SMU IR request 1 and SMU IR request 2.

Features not supported:

The SMU driver should not access the interrupt router directly.

Users of the hardware

The service request nodes SRC_SMUx (x=0 to 2) are exclusively allocated to the SMU peripheral. The service request to be triggered is decided by SMU peripheral using the interrupt generation set selected by the alarm that is IGSC0, IGSC1 or IGSC2. Each set is a code which is a combination of the three service requests that need to be enabled that is IGSC0 = 011b implies that SRC_SMU0, SRC_SMU1 service requests are triggered when the alarm is triggered.

Hardware diagnostic features

None.

Hardware events

None.

3.1.3 File structure

3.1.3.1 C file structure

The section provides details on the C files of the SMU driver.

SMU driver

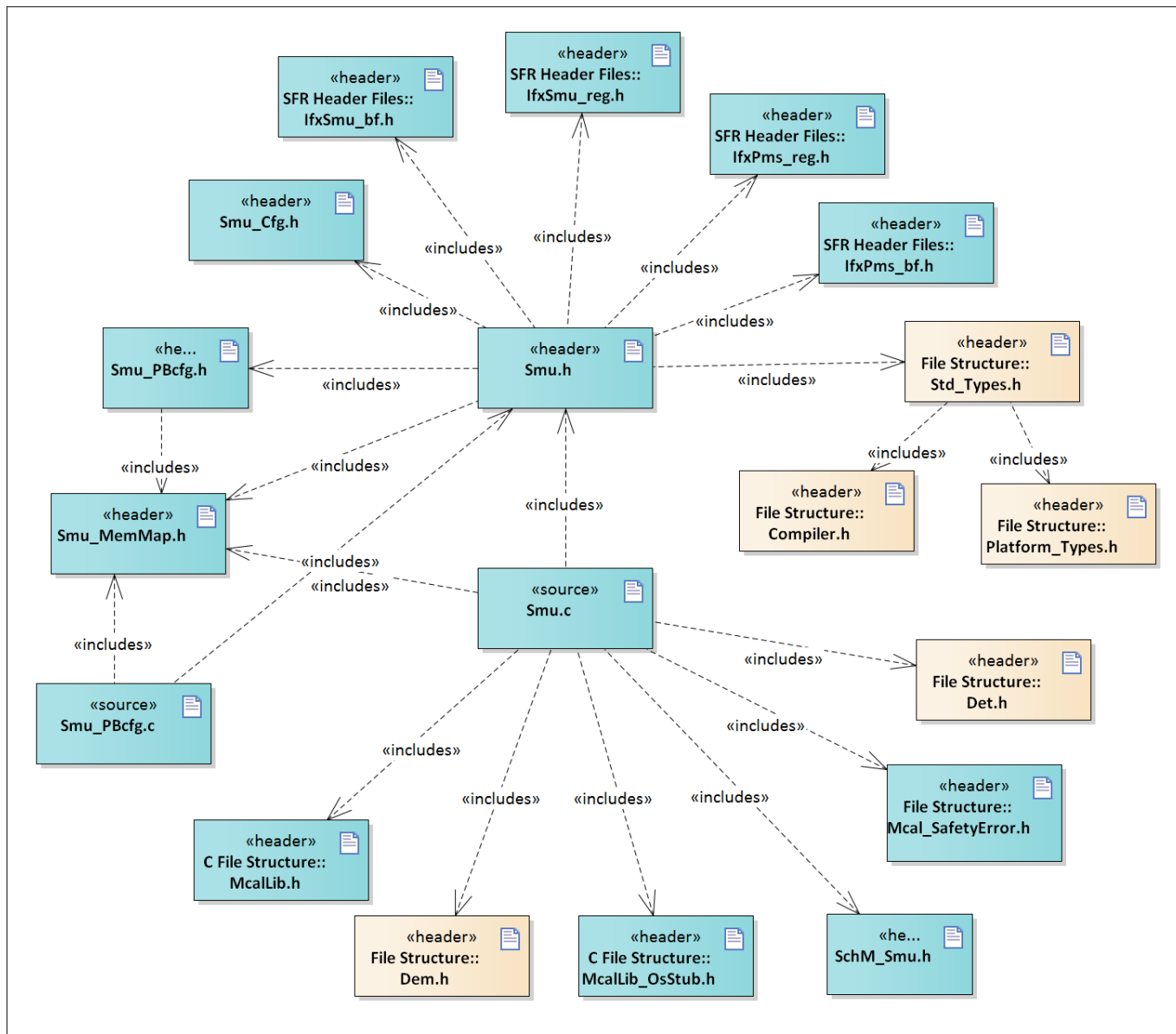


Figure 11 **C file structure**

Table 168 C file structure

File name	Description
Smu.c	Contains the source code of the SMU software module
Smu.h	Contains the data types and function prototypes to be exported
Smu_PBCfg.c	Contains the post-build configuration for the SMU driver
Smu_Cfg.h	Contains the pre-compile configuration for the SMU driver. The file implements all pre-processor directives.
Smu_MemMap.h	Contains mapping of code and data (variables, constants) to specific memory sections for the SMU driver
Dem.h	Provides the exported interfaces of Diagnostic Event Manager
Det.h	Provides the exported interfaces of Development Error Tracer

SMU driver
Table 168 C file structure (continued)

File name	Description
Std_Types.h	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.
Compiler.h	Provides abstraction from compiler specific keywords
Platform_Types.h	Platform specific type declaration file as defined by AUTOSAR
McalLib.h	Header file (Static) defining prototypes of data structures and APIs of end-init and delay services and included by McalLib.c
IfxSmu_bf.h	SFR header file for Smu
IfxSmu_reg.h	SFR header file for Smu
IfxPms_reg.h	SFR header file for Pms
IfxPms_bf.h	SFR header file for Pms
McalLib_OsStub.h	McalLib_OsStub.h provides macros to support user mode of Tricore. This should be included by other drivers to call OS APIs.
Mcal_SafetyError.h	Header file containing the prototype for the API for reporting safety-related errors
Smu_PBcfg.h	Contains generated configuration data of user
SchM_Smu.h	The header file contains the definitions of SMU critical sections

3.1.3.2 Code generator plugin files

The section provides details on the plugin files of the SMU driver.

SMU driver

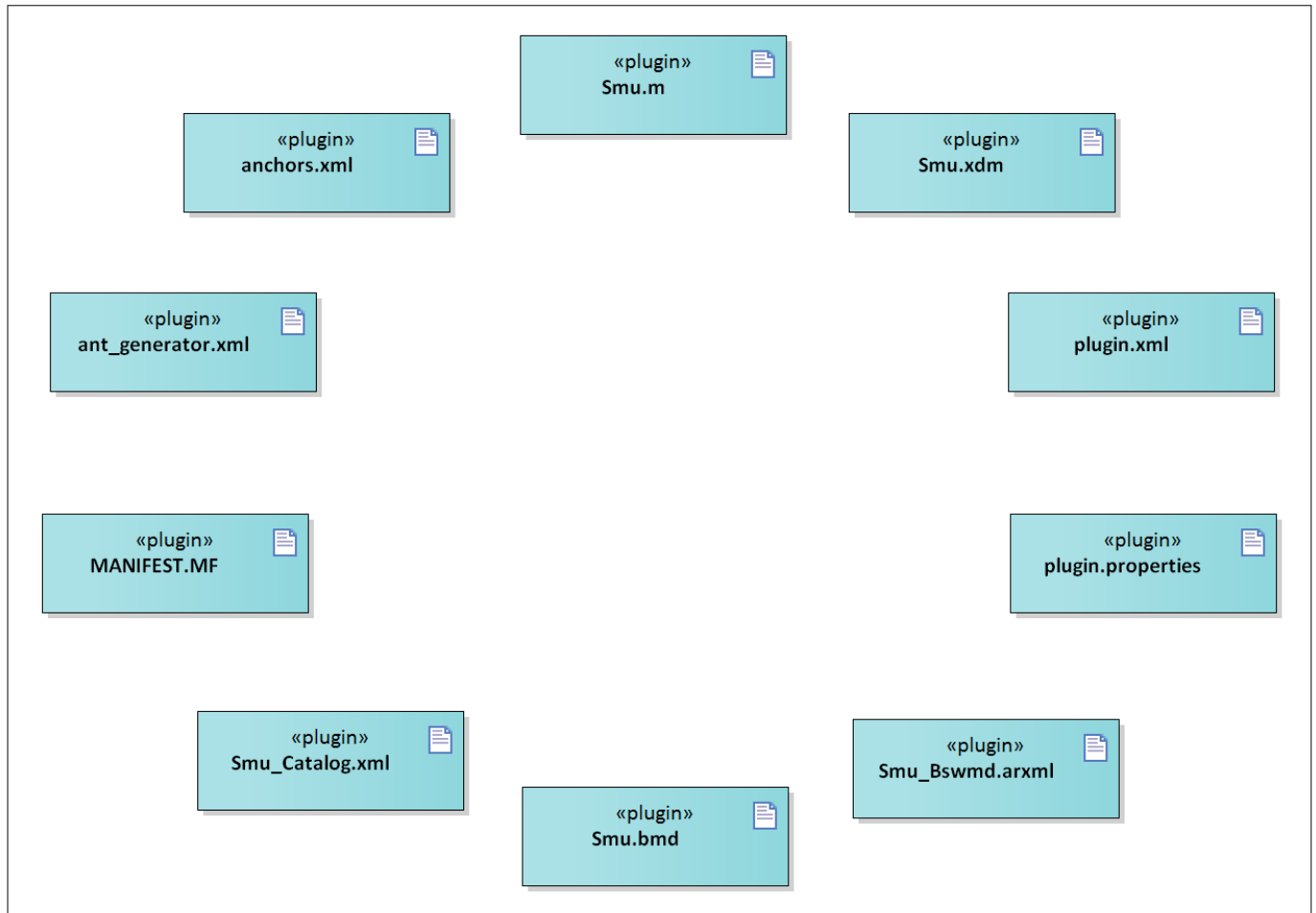


Figure 12 Code generator plugin files

Table 169 Code generator plugin files

File name	Description
Smu.m	The file is code template macro file for the SMU driver
Smu.xdm	The file is Tresos format XML data model schema file for the SMU driver
anchors.xml	The file is Tresos anchors support file for the SMU driver
ant_generator.xml	The file is a Tresos support file to generate and rename multiple post-build configurations when using variation point
MANIFEST.MF	The file is Tresos plugin support file containing the metadata for SMU driver
plugin.properties	The file is Tresos plugin support file for the SMU driver
plugin.xml	The file is Tresos plugin support file for the SMU driver
Smu.bmd	The file is an AUTOSAR format XML data model schema file for the SMU driver
Smu_Catalog.xml	The file is an AUTOSAR format catalog file for the SMU driver
Smu_Bswmd.arxml	The file is an AUTOSAR format module description file for the SMU driver

3.1.4 Integration hints

The section lists the key points that an integrator or user of the SMU driver must consider.

SMU driver**Key features**

- Alarm handling: Setting, querying, clearing of alarms and setting up alarm reactions
- FSP handling: Activation, deactivation of the FSP to request the trigger of safe state
- Port Emergency handling: Activation of port emergency stop
- Recovery Timer handling: Setup, status query status of recovery timers
- SMU State Machine: Status query of the SMU state machine
- SMU Configuration Protection: Activation of permanent lock mechanism
- Monitoring SMU_core by SMU_stdby
- Register Monitoring interface for SFF tests

3.1.4.1 Stub modules

The section lists the modules required to integrate the SMU driver.

EcuM

The SMU driver does not use EcuM.

Memory mapping

SMU driver

The variables, constants and configuration data of SMU driver is encapsulated in specific memory section macros defined in the `Smu_MemMap.h`. The user must place appropriate compiler pragmas to ensure the memory mapped macros are located to the correct memory space. Sample memory sections are listed below:

```
/* Data Sections ---- Global RAM data to be placed in LMU RAM NC sections*/
#if defined SMU_START_SEC_INIT_VAR_ASIL_B_GLOBAL_32
/* User Pragma to be placed here for LMU RAM NC*/
#undef SMU_START_SEC_INIT_VAR_ASIL_B_GLOBAL_32
#undef MEMMAP_ERROR
#elif defined SMU_STOP_SEC_INIT_VAR_ASIL_B_GLOBAL_32
/* User Pragma for LMU RAM NC here */
#undef SMU_STOP_SEC_INIT_VAR_ASIL_B_GLOBAL_32
#undef MEMMAP_ERROR

/*Configuration data sections-- to be placed in PFx*/
#elif defined SMU_START_SEC_CONFIG_DATA_ASIL_B_GLOBAL_UNSPECIFIED
/* User pragmas to be placed here for PF0 */
#undef SMU_START_SEC_CONFIG_DATA_ASIL_B_GLOBAL_UNSPECIFIED
#undef MEMMAP_ERROR
#elif defined SMU_STOP_SEC_CONFIG_DATA_ASIL_B_GLOBAL_UNSPECIFIED
/* User pragmas to be placed here for PFx */
#undef SMU_STOP_SEC_CONFIG_DATA_ASIL_B_GLOBAL_UNSPECIFIED
#undef MEMMAP_ERROR

/* Code Section ----- to be placed in PFx*/
#elif defined SMU_START_SEC_CODE_ASIL_B_GLOBAL
/* User Pragma to be placed here */
#undef SMU_START_SEC_CODE_ASIL_B_GLOBAL
#undef MEMMAP_ERROR
#elif defined SMU_STOP_SEC_CODE_ASIL_B_GLOBAL
/* User Pragma to be placed here */
#undef SMU_STOP_SEC_CODE_ASIL_B_GLOBAL
#undef MEMMAP_ERROR

#endif
```

DET

The complete DET module is not provided as a standard MCAL module but rather provided as a stub module. The user of the SMU driver must process all the errors reported to the DET module through the `Det_ReportError` API. The function is defined in the non-productive `Det.h` and `Det.c` files.

DEM

The complete DEM module is not provided as a standard MCAL module but rather provided as a stub module. The user of the SMU driver must process all the production errors reported to the DEM module through the `Dem_ReportErrorStatus` API. The function is defined in the non-productive `Dem.c` and `Dem.h` files.

SchM

The complete SchM module is not provided as a standard MCAL module but rather provided as a stub module. The SMU driver uses the exclusive area defined in `SchM_Smu.h` to protect SFRs and variables from concurrent accesses from different threads. The identified SchM for SMU driver are: `CmdAccess` and `DriverAccess`.

The user should implement the SchM functions defined by the SMU driver as suspend or resume of interrupts for the CPU on which the API is invoked. The SchM related functions for SMU driver are defined in non-

SMU driver

productive file `SchM_Smu.c` and `SchM_Smu.h`. A sample implementation of the SchM functions is depicted below.

```
/*Sample implementation*/
#include "IFX_Os.h"
#include "SchM_Smu.h"

void SchM_Enter_CmdAccess(void)
{
    /*Suspend all interrupts*/
    SuspendAllInterrupts();
}

void SchM_Exit_CmdAccess(void)
{
    /*Resume all interrupts*/
    ResumeAllInterrupts();
}

void SchM_Enter_DriverAccess(void)
{
    /*Suspend all interrupts*/
    SuspendAllInterrupts();
}

void SchM_Exit_DriverAccess(void)
{
    /*Resume all interrupts*/
    ResumeAllInterrupts();
}
```

Safety error

All safety-related errors are reported through `Mcal_ReportSafetyError`. Only reporting of safety error is the responsibility of the driver, the handling of the reported error must be done by the user. The function is defined in the non-productive `Mcal_SafetyError.c` and `Mcal_SafetyError.h` files.

Notification and callbacks

The SMU driver does not implement or report any notification or callback functions.

Operating system

The SMU driver does not require configuration of any interrupts. The interrupts triggered as a result of alarm action have to be handled by the user.

3.1.4.2 Multicore and Resource Manager

The multicore- and resource-related information is listed below:

- The runtime services of the SMU driver will be accessible by all cores.
- The hardware and software safety mechanism are associated with specific alarm groups and positions and cannot be reallocated by software or configuration. Hence, the SMU driver does not have any core specific resource allocation.

SMU driver

- The SMU initialization and de-initialization must be called only from the master core. In case `Smu_Init` or `Smu_DeInit` is called from any core other than the master core than the APIs will report an error `E_NOT_OK`. In case DET is enabled, a DET error `SMU_E_CORE_MISMATCH` will also be reported.

3.1.4.3 MCU support

The SMU driver does not use MCU.

3.1.4.4 Port support

The PORT driver configures the Port pins of the entire microcontroller. The user must configure port pins 33.8 and 33.10 used by the SMU driver for FSP through the PORT configuration.

3.1.4.5 DMA support

The SMU driver does not use DMA.

3.1.4.6 Interrupt connections

The SMU driver does not implement any interrupt handlers. The internal reactions configured as a result of alarm events, which trigger an interrupt need to be handled by the user.

SMU driver

3.1.4.7 Example usage

Initializing and deinitializing the SMU driver

The SMU driver initialization is done by calling the `Smu_Init` API. The user must call `Smu_Init` API only from the master core. When `Smu_Init` is called from a core other than a master core, the API returns an error. Similar is the case with `Smu_DeInit` API. The SMU driver can be initialized and deinitialized as shown below. Also in order to check the initialization values, `Smu_InitCheck` API can be used. However, the `Smu_InitCheck` API is enabled only when the parameter `SmuInitCheckApi` is enabled.

```
#include "Smu_Test.h"
/* Initialize SMU*/
Return = Smu_Init(&(Smu_Config));
/*Check for initialiazation values*/
#if(SMU_INIT_CHECK_API==STD_ON)
Return = Smu_InitCheck(&(Smu_Config));
#endif
/*Call SMU driver functions*/
/*.....*/
/*Deinitialize the driver*/
Return = Smu_DeInit();
```

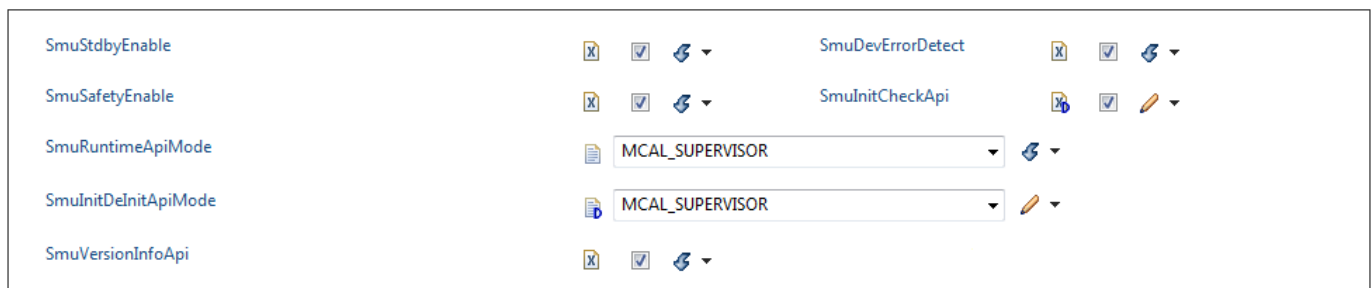


Figure 13 Example configuration for enabling `Smu_stdby`, `InitCheck` API, `DET`, version info API and user mode

FSP handling

For FSP handling, the glitch filter settings, output pin direction and enable has to set by the user (refer configuration for more details). FSP can be operated by choosing one of the three signaling modes:

- Time switching protocol
- Dual rail protocol
- Bi-stable protocol

FSP output pins can be enabled through configuration for `Smu_stdby`. PES can be enabled or disabled while using FSP. The prescaler, signaling mode, fault state duration, glitch filter settings, port enable and hardware direction can be selected as per the configuration depicted below. This includes the common glitch filter settings, port enable and hardware direction.

SMU driver

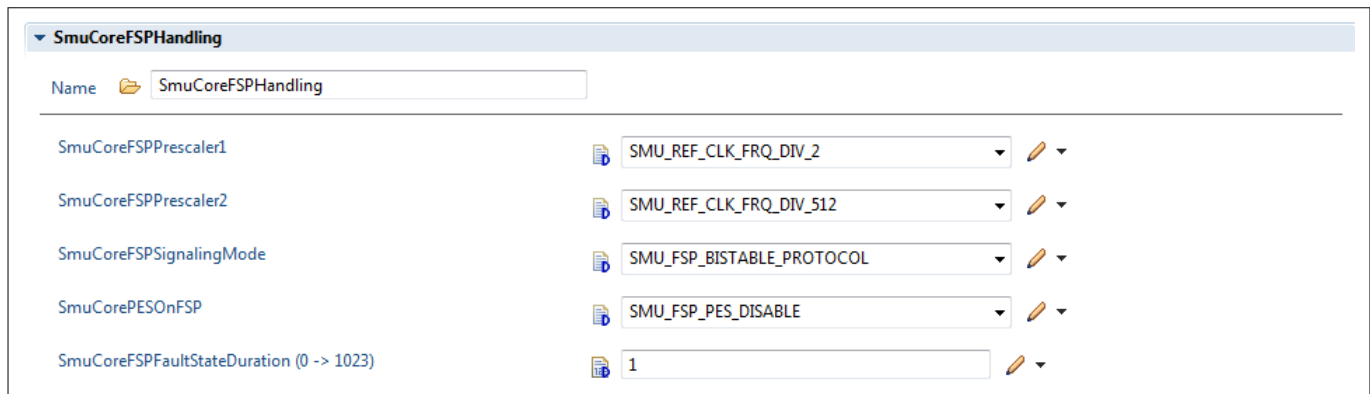


Figure 14 FSP setting for Smu_core

To enable the transition of FAULT to RUN state, the parameter `SmuCoreEnableFaultToRunState` shall be enabled. Also the external action with respect to the alarm group and position should be enabled. The internal and external reaction configuration is explained in section **Alarm action**

An example code usage:

```
Smu_CoreStateType SmuState;
/*Setup the error pin in SMU mode*/
ResultFSP = Smu_SetupErrorPin();
if(E_OK == ResultFSP)
{
    /*Set the alarm status of alarm group 10 and position 5*/
    ResultFSP = Smu_SetAlarmStatus(SMU_ALARM_GROUP10, SMU_ALARM_5);
    {
        /*Activate FSP to indicate a fault state*/
        ResultFSP = Smu_ActivateFSP();
        if(E_OK == ResultFSP)
        {
            /*Get the SMU state*/
            SmuState = Smu_GetSmuState();
            if(SMU_FAULT_STATE == SmuState)
            {
                /*In case it is FAULT state then release FSP to transition to RUN state*/
                ResultFSP = Smu_ReleaseFSP();
                if(E_OK == ResultFSP)
                {
                    /*Release error pin to transition to GPIO mode*/
                    ResultFSP = Smu_ReleaseErrorPin();
                }
            }
        }
    }
}
```

Smu_core state machine

Smu_core has three states:-START, RUN, FAULT.

Transition from START to RUN: The transition takes place by executing the `Smu_core` command to activate RUN state by calling the API `Smu_ActivateRunState`.

SMU driver

Transition from FAULT to RUN: The transition takes place by executing the Smu_core command by calling the API Smu_ReleaseFSP.

```

/*Get the Smu_core state*/
SmuState = Smu_GetSmuState();

switch (SmuState)
{
    case SMU_START_STATE:
    {
        /*Activate the RUN state*/
        Result = Smu_ActivateRunState(SMU_RUN_COMMAND);
        break;
    }
    case SMU_FAULT_STATE:
    {
        /*In case it is FAULT state then release FSP to transition to RUN state*/
        Result = Smu_ReleaseFSP();
        break;
    }
    case SMU_RUN_STATE:
    {
        Result = E_OK;
        break;
    }
    default:
    {
        break;
    }
}

```

Configuration register locking

The SMU configuration registers can be protected from unintended access in two ways:

- **Temporary lock:** To write into the SMU configuration registers, the SMU configuration register protection is disabled temporarily and then again enabled.
- **Permanent lock:** To prevent any writing into the SMU configuration register, the SMU configuration register protection is enabled and can be disabled only after application reset.

Smu_LockConfigRegs enables the permanent lock on the configuration registers. In case the API Smu_LockConfigRegs fails to turn on the permanent lock and safety error check is enabled, then a DET is reported to let the user know that the configuration registers could not be permanently locked. An example usage is shown below:

```

Std_ReturnType Result = E_NOT_OK;
ResultLockTest = Smu_LockConfigRegs();

```

Alarm status

SMU driver

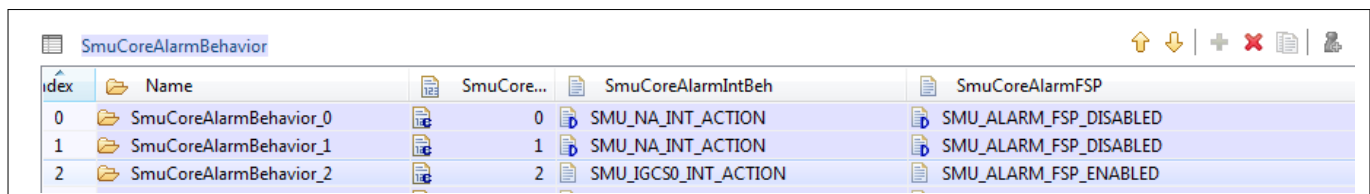
The SMU driver provides APIs to get, clear and set alarm status. An example usage is shown below.

```
Smu_SetAlarmStatus is valid only for Smu_core.
ResultAlarmStatus = Smu_ClearAlarmStatus(SMU_ALARM_GROUP10, SMU_ALARM_0);
if (E_OK == ResultAlarmStatus)
{
    ResultAlarmStatus = Smu_SetAlarmStatus(SMU_ALARM_GROUP10, SMU_ALARM_0);
    if (E_OK == ResultAlarmStatus)
    {
        ResultAlarmStatus = Smu_GetAlarmStatus(SMU_ALARM_GROUP10, &AlarmStatus);
        if ((E_OK == ResultAlarmStatus) && (0x01U == (AlarmStatus & 0x01)))
        {
            ResultAlarmStatus = Smu_ClearAlarmStatus(SMU_ALARM_GROUP10, SMU_ALARM_0);
            if (E_OK == ResultAlarmStatus)
            {
                ResultAlarmStatus = Smu_GetAlarmStatus(SMU_ALARM_GROUP10, &AlarmStatus);
            }
        }
    }
}
```

Alarm action

The SMU driver provides services to set, get and clear the alarm actions by using the `Smu_SetAlarmAction` and `Smu_GetAlarmAction` APIs.

There are two kinds of alarm action that can be configured for `Smu_core`: internal action and external action. These are configured as shown below.

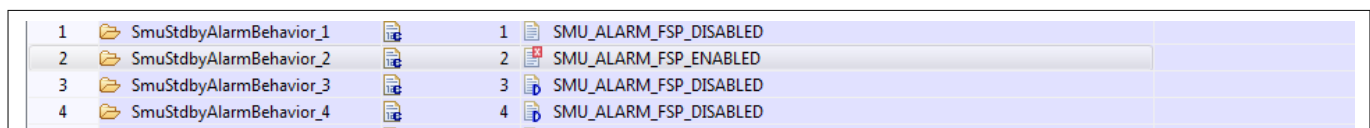


Index	Name	SmuCore...	SmuCoreAlarmIntBeh	SmuCoreAlarmFSP
0	SmuCoreAlarmBehavior_0	0	SMU_NA_INT_ACTION	SMU_ALARM_FSP_DISABLED
1	SmuCoreAlarmBehavior_1	1	SMU_NA_INT_ACTION	SMU_ALARM_FSP_DISABLED
2	SmuCoreAlarmBehavior_2	2	SMU_IGCS0_INT_ACTION	SMU_ALARM_FSP_ENABLED

Figure 15 Internal and external action settings for Smu_core

For `Smu_stdby`, the internal reaction is by default `SMU_NA_INT_ACTION`. Only external action can be configured.

For both `Smu_core` and `Smu_stdby`, in case an alarm action is tried to be configured for a reserved alarm group, then the configuration will throw an error as shown in the figure below.



1	SmuStdbyAlarmBehavior_1	1	SMU_ALARM_FSP_DISABLED	
2	SmuStdbyAlarmBehavior_2	2	SMU_ALARM_FSP_ENABLED	
3	SmuStdbyAlarmBehavior_3	3	SMU_ALARM_FSP_DISABLED	
4	SmuStdbyAlarmBehavior_4	4	SMU_ALARM_FSP_DISABLED	

Figure 16 External action setting for Smu_stdby

Smu_core recovery timer

The recovery timers are configured by first enabling the RT0 and RT1 and setting the RT duration. On enabling, the respective RT group configurations are enabled.

SMU driver

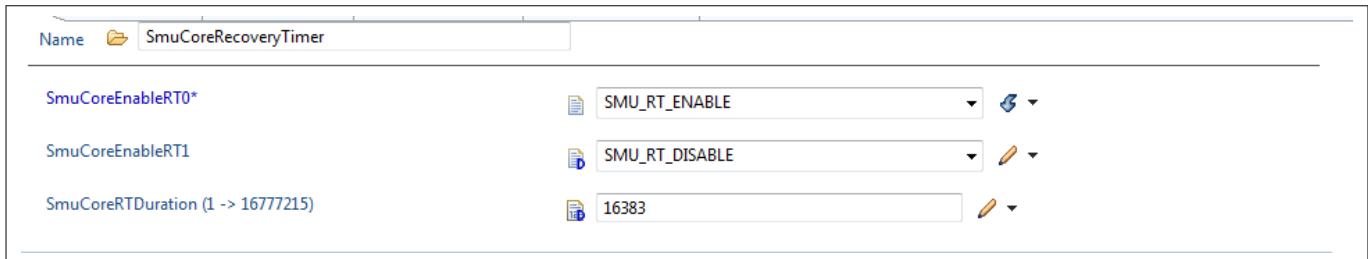
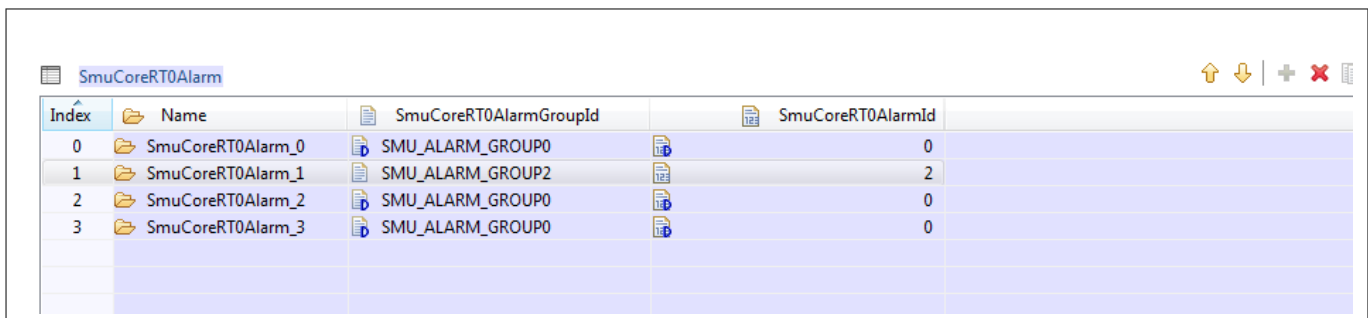


Figure 17 Enable the RT configuration

After the RT group configuration is enabled the recovery timers can be assigned to the Smu_core alarm groups and positions.



Index	Name	SmuCoreRT0AlarmGroupId	SmuCoreRT0AlarmId
0	SmuCoreRT0Alarm_0	SMU_ALARM_GROUP0	0
1	SmuCoreRT0Alarm_1	SMU_ALARM_GROUP2	2
2	SmuCoreRT0Alarm_2	SMU_ALARM_GROUP0	0
3	SmuCoreRT0Alarm_3	SMU_ALARM_GROUP0	0

Figure 18 Assigning alarm groups and positions to RT

In order to configure RT, the respective alarm group and position have to be configured for their internal action. In case it is a reserved alarm position, the user must take care of not assigning the alarm position to RT0 or RT1. The information of reserved alarm positions will be evident when an error is encountered while configuring the internal alarm action for that particular alarm group and position. Hence, the RT configuration can take reference from the error as discussed.

The SMU driver provides the services to stop the recovery timer and to detect any missed events for the configured RT. An example usage is shown below.

```
volatile uint32 DelayCount = SMU_RT_DELAY;
/*Set the alarm status for alarm group 10 and position 2*/
ResultRecoveryTimer = Smu_SetAlarmStatus(SMU_ALARM_GROUP10, SMU_ALARM_2);
if (E_OK == ResultRecoveryTimer)
{
    /*Set the alarm status for alarm group 10 and position 3*/
    ResultRecoveryTimer = Smu_SetAlarmStatus(SMU_ALARM_GROUP10, SMU_ALARM_3);
    if (E_OK == ResultRecoveryTimer)
    {
        /*capture any missed events*/
        ResultRecoveryTimer = Smu_GetRTMissedEvent(SMU_TIMER_NUM, &EventMissed);
        if (0x01 == EventMissed)
        {
            /*Stop the RT is missed events are detected*/
            ResultRecoveryTimer = Smu_RTStop(SMU_TIMER_NUM);
        }
    }
}
```

SMU driver**Register monitoring**

The `Smu_RegisterMonitor` API can be used to enable the SFF tests for protected registers of particular modules and retrieve the results. The input parameter passed to the API has to strictly follow the sequence of modules as per the bit fields of the RMCTL register. However, the user shall ensure that the module for which the SFF test is being requested is present in the derivative being used.

Smu_core alive test

The `Smu_CoreAliveTest` API enables the `Smu_core_alive` test. For the `Smu_core_alive` test to run through `Smu_core` command sequence, `Smu_stdby` must be enabled. In case `Smu_stdby` is not enabled then `Smu_CoreAliveTest` returns an error `E_OK`. In case DET is enabled, a DET error is also reported.

```
Result = Smu_CoreAliveTest();
```

Alarm execution status

The `Smu_GetAlarmExecutionStatus` API can be used to retrieve the alarm execution status. Once retrieved, the execution status can be cleared using the `Smu_ClearAlarmExecutionStatus` API. The alarm reactions do not get triggered if the alarm execution status for the particular alarm is not cleared.

```
Std_ReturnType RetVal = E_NOT_OK;
/*Get the alarm execution status*/
RetVal = Smu_GetAlarmExecutionStatus(ExecReq, &ExecStatus);

if(RetVal == E_OK)
{
    /*Clear the alarm execution status of the particular alarms as requested*/
    RetVal = Smu_ClearAlarmExecutionStatus(ExecStatusReq);
}
```

3.1.5 Key architectural considerations

3.1.5.1 Clearing alarm status during initialization

During initialization, all the alarm statuses are cleared, hence the user must ensure to keep a track of the alarm status before `Smu_Init` is called.

3.1.5.2 Initialization and de-initialization

`Smu_Init` and `Smu_Deinit` shall be called only from the master cores. In case init and deinit are called from any other core besides the master core, the sequence will return an error. There is no resource distribution across the cores and the SMU driver shall be accessible across all cores.

3.1.5.3 Smu_core state machine transitions

The `Smu_core` state machine transitions are not be verified by the driver. The user must verify the state before using it.

3.1.5.4 Recovery timer handling

While recovery timer is running, a missed alarm mapped to the same recovery timer is logged in the SFR SMU_STS. These missed alarms have to explicitly checked and cleared by the application. The missed alarm can be checked using the `Smu_GetRTMissedEvent` API.

SMU driver

3.2 Assumptions of Use (AoUs)

The AoUs for the driver are as follows:

- **Clearing of RT missed events**

The user shall explicitly clear the RT missed events detected.

[cover parentID SMU={79E76BAC-C9CD-409c-8518-B9778796261D}]

- **Init check AoU**

The user shall call the Smu_InitCheck API after initialization but before releasing vehicle safe state. Additionally user shall ensure that the Smu_LockCfgReg is invoked immediately after Smu_InitCheck without any pre-emption to avoid possible change of SMU configuration from any other software units.

[cover parentID SMU={6779F95B-9003-4e32-A5CA-4E072E2BB8CF}]

- **Non-interference check**

The user shall check that the correct config pointer has been passed and there is no interference to MCAL from other modules.

[cover parentID SMU={B5E820B9-2097-4917-BC6C-60B5A523F429}]

- **SFF test module check**

The user shall check if the module for which SFF test has been requested is available or not in the particular device.

[cover parentID SMU={9A7FBC44-B881-4469-9D42-CA6507F9A712}]

- **SMU FSP functionality**

The API Smu_Release FSP is asynchronous and the transition from FAULT to RUN state may require several cycles based on the fault state duration configured by the user, control PAD characteristics and/or recurrent fault occurrences. The API Smu_Activate FSP is asynchronous and the transition to FAULT state may require several cycles based on the control PAD characteristics. Hence there is no deterministic time frame within which the state transition can be checked by the driver. The user shall ensure the transition to the intended state has occurred in the Smu_core. The user can check this using the Smu_GetSmuState

[cover parentID SMU={D1FB1959-3FB3-4123-92D8-226B551E6129}]

SMU driver

3.3 Reference information

3.3.1 Configuration interfaces

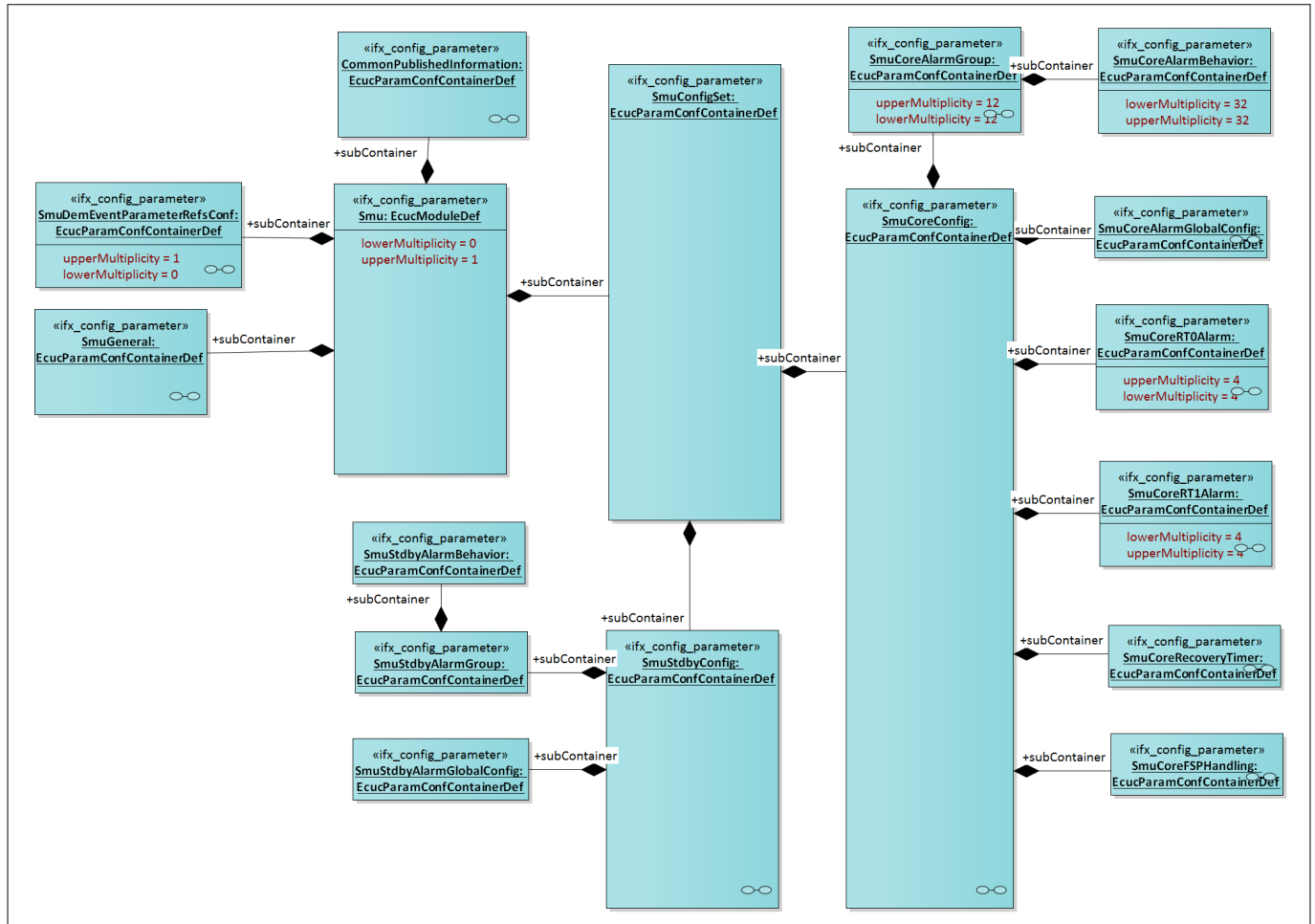


Figure 19 Container hierarchy along with their configuration parameters

3.3.1.1 Container: CommonPublishedInformation

The container gives the published information for SMU driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.1.1 ArMajorVersion

Table 170 Specification for ArMajorVersion

Name	ArMajorVersion		
Description	This parameter provides the major version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	4		

SMU driver

Table 170 **Specification for ArMajorVersion (continued)**

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.1.2 ArMinorVersion

Table 171 **Specification for ArMinorVersion**

Name	ArMinorVersion		
Description	This parameter provides the minor version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.1.3 ArPatchVersion

Table 172 **Specification for ArPatchVersion**

Name	ArPatchVersion		
Description	This parameter provides the patch version of the AUTOSAR specification.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL

SMU driver

Table 172 **Specification for ArPatchVersion (continued)**

Dependency	-
-------------------	---

3.3.1.1.4 ModuleId

Table 173 **Specification for ModuleId**

Name	ModuleId		
Description	The configuration parameter defines the module ID of SMU module from module list.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	255		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.1.5 Release

Table 174 **Specification for Release**

Name	Release		
Description	The configuration parameter defines the AURIX derivative used for the implementation.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	String		
Default value	As per HW derivative		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.1.6 SwMajorVersion

Table 175 **Specification for SwMajorVersion**

Name	SwMajorVersion
-------------	----------------

SMU driver
Table 175 **Specification for SwMajorVersion (continued)**

Description	The configuration parameter defines the major version number of the vendor specific implementation of the module. The numbering is vendor specific.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.1.7 SwMinorVersion
Table 176 **Specification for SwMinorVersion**

Name	SwMinorVersion		
Description	The configuration parameter defines the minor version number of the vendor specific implementation of the module. The numbering is vendor specific.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the driver version		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.1.8 SwPatchVersion
Table 177 **Specification for SwPatchVersion**

Name	SwPatchVersion		
Description	The configuration parameter defines the patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 255		

SMU driver
Table 177 Specification for SwPatchVersion (continued)

Default value	As per driver version		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.1.9 VendorId
Table 178 Specification for VendorId

Name	VendorId		
Description	The configuration parameter defines the vendor ID of the dedicated implementation of the module according to the AUTOSAR vendor list.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published-Information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.2 Container: Smu

The Smu container is the parent container for SMU module.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.3 Container: SmuConfigSet

The container contains SmuConfigSet configurations for the SMU driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.4 Container: SmuCoreAlarmBehavior

The container contains configuration parameters related to alarm behavior. Each alarm group has thirty two alarm configurations. Both the internal and external behavior can be configured for every alarm.

SMU driver

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.4.1 SmuCoreAlarmFSP
Table 179 Specification for SmuCoreAlarmFSP

Name	SmuCoreAlarmFSP		
Description	The configuration parameter defines the value of the FSP configuration. The default value of the parameter is set to the reset value of corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	SMU_ALARM_FSP_DISABLED: The configuration parameter literal defines that FSP is disabled. SMU_ALARM_FSP_ENABLED: The configuration parameter literal defines that FSP is enabled.		
Default value	SMU_ALARM_FSP_DISABLED		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.4.2 SmuCoreAlarmIntBeh
Table 180 Specification for SmuCoreAlarmIntBeh

Name	SmuCoreAlarmIntBeh		
Description	The configuration parameter defines the internal behavior of an alarm event. The default value of the parameter is set to the reset value of corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef

SMU driver
Table 180 Specification for SmuCoreAlarmIntBeh (continued)

Range	SMU_CPU_RESET_INT_ACTION: The configuration parameter literal defines the internal behavior as sending CPU reset configuration request. SMU_IGCS0_INT_ACTION: The configuration parameter literal defines the internal behavior as sending an interrupt request to the interrupt system according to the interrupt generation configuration set 0. SMU_IGCS1_INT_ACTION: The configuration parameter literal defines the internal behavior as sending an interrupt request to the interrupt system according to the interrupt generation configuration set 1 SMU_IGCS2_INT_ACTION: The configuration parameter literal defines the internal behavior as sending an interrupt request to the interrupt system according to the interrupt generation configuration set 2 SMU_NA_INT_ACTION: The configuration parameter literal defines the internal behavior as no action (default value). SMU_NMI_INT_ACTION: The configuration parameter literal defines the internal behavior as sending NMI request to the SCU. SMU_RESET_INT_ACTION: The configuration parameter literal defines the internal behavior as sending reset request to the SCU.		
Default value	SMU_NA_INT_ACTION		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.4.3 SmuCoreAlmBehaviourId
Table 181 Specification for SmuCoreAlmBehaviourId

Name	SmuCoreAlmBehaviourId		
Description	The configuration parameter defines the alarm behavior ID corresponding to the particular group. First alarm behavior is selected as the default value.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 31		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

SMU driver
Table 181 Specification for SmuCoreAlmBehaviourId (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5 Container: SmuCoreAlarmGlobalConfig

The container contains the alarm global configuration parameters. The parameters are used for initializing the SMU_AGC register.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.5.1 SmuCoreCpu0ResetRequest

Table 182 Specification for SmuCoreCpu0ResetRequest

Name	SmuCoreCpu0ResetRequest		
Description	The configuration parameter is a Boolean which denotes whether the reset request to CPU0 is set or not. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.2 SmuCoreCpu1ResetRequest

Table 183 Specification for SmuCoreCpu1ResetRequest

Name	SmuCoreCpu1ResetRequest		
Description	The configuration parameter is a Boolean which denotes whether the reset request to CPU1 is set or not. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef

SMU driver
Table 183 Specification for SmuCoreCpu1ResetRequest (continued)

Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.3 SmuCoreCpu2ResetRequest
Table 184 Specification for SmuCoreCpu2ResetRequest

Name	SmuCoreCpu2ResetRequest		
Description	The configuration parameter is a Boolean which denotes whether the reset request to CPU2 is set or not. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.4 SmuCoreCpu3ResetRequest
Table 185 Specification for SmuCoreCpu3ResetRequest

Name	SmuCoreCpu3ResetRequest		
Description	The configuration parameter is a Boolean which denotes whether the reset request to CPU3 is set or not. The default value of this parameter is set to the reset value of the corresponding SFR.		

SMU driver
Table 185 Specification for SmuCoreCpu3ResetRequest (continued)

Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.5 SmuCoreCpu4ResetRequest
Table 186 Specification for SmuCoreCpu4ResetRequest

Name	SmuCoreCpu4ResetRequest		
Description	The configuration parameter is a Boolean which denotes whether the reset request to CPU4 is set or not. This is applicable only for devices which have six CPUs and should not be set for devices with four CPUs. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.6 SmuCoreCpu5ResetRequest
Table 187 Specification for SmuCoreCpu5ResetRequest

Name	SmuCoreCpu5ResetRequest		
-------------	-------------------------	--	--

SMU driver
Table 187 Specification for SmuCoreCpu5ResetRequest (continued)

Description	The configuration parameter is a Boolean which denotes whether the reset request to CPU5 is set or not.. This is applicable only for devices which have six CPUs and should not be set for devices with four CPUs. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.7 SmuCoreCpuResetActivatePES
Table 188 Specification for SmuCoreCpuResetActivatePES

Name	SmuCoreCpuResetActivatePES		
Description	The configuration parameter enables the PES on CPU reset. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

SMU driver

3.3.1.5.8 SmuCoreEnableFaultToRunState

Table 189 Specification for SmuCoreEnableFaultToRunState

Name	SmuCoreEnableFaultToRunState		
Description	<p>The configuration parameter defines whether the FAULT state to RUN state transition is enabled or disabled. The state transition is possible only when this parameter is defined with SMU_EFRST_ENABLE.</p> <p>The default value is this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	<p>SMU_EFRST_DISABLE: The configuration parameter literal defines that the FAULT state to RUN state transition is disabled.</p> <p>SMU_EFRST_ENABLE: The configuration parameter literal defines that the FAULT state to RUN state transition is enabled.</p>		
Default value	SMU_EFRST_DISABLE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.9 SmuCoreIGCS0ActivatePES

Table 190 Specification for SmuCoreIGCS0ActivatePES

Name	SmuCoreIGCS0ActivatePES		
Description	<p>The configuration parameter defines the control of the Port Emergency Stop (PES) feature for IGCS0 internal action. When an IGCS0 internal action is triggered, the hardware triggers automatically the PES, on enabling. The default value of this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	<p>TRUE</p> <p>FALSE</p>		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL

SMU driver
Table 190 **Specification for SmuCoreIGCS0ActivatePES (continued)**

Dependency	-
-------------------	---

3.3.1.5.10 **SmuCoreIGCS1ActivatePES**
Table 191 **Specification for SmuCoreIGCS1ActivatePES**

Name	SmuCoreIGCS1ActivatePES		
Description	The configuration parameter defines the control of the Port Emergency Stop (PES) feature for IGCS1 internal action. When an IGCS1 internal action is triggered, the hardware triggers automatically the PES, on enabling. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.11 **SmuCoreIGCS2ActivatePES**
Table 192 **Specification for SmuCoreIGCS2ActivatePES**

Name	SmuCoreIGCS2ActivatePES		
Description	The configuration parameter defines the control of the Port Emergency Stop (PES) feature for IGCS2 internal action. When an IGCS2 internal action is triggered, the hardware triggers automatically the PES, on enabling. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-

SMU driver
Table 192 **Specification for SmuCoreIGCS2ActivatePES (continued)**

Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.12 SmuCoreInterruptSet0
Table 193 **Specification for SmuCoreInterruptSet0**

Name	SmuCoreInterruptSet0		
Description	<p>The configuration parameter defines the output value of the interrupt request vector when the alarm configuration flag selects the interrupt configuration set 0.</p> <p>The default value is this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	<p>SMU_SELECT_INT0: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0.</p> <p>SMU_SELECT_INT0_INT1: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0 and SRC_SMU1.</p> <p>SMU_SELECT_INT0_INT1_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0, SRC_SMU1 and SRC_SMU2.</p> <p>SMU_SELECT_INT0_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0 and SRC_SMU2.</p> <p>SMU_SELECT_INT1: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU1.</p> <p>SMU_SELECT_INT1_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU1 and SRC_SMU2.</p> <p>SMU_SELECT_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU2.</p> <p>SMU_SELECT_INT_NONE: The configuration parameter literal defines the output value of the interrupt request vector as no interrupt selected.</p>		
Default value	SMU_SELECT_INT_NONE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

SMU driver
3.3.1.5.13 SmuCoreInterruptSet1
Table 194 Specification for SmuCoreInterruptSet1

Name	SmuCoreInterruptSet1		
Description	<p>The configuration parameter defines the output value of the interrupt request vector when the alarm configuration flag selects the interrupt configuration set 1.</p> <p>The default value is this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	<p>SMU_SELECT_INT0: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0.</p> <p>SMU_SELECT_INT0_INT1: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0 and SRC_SMU1.</p> <p>SMU_SELECT_INT0_INT1_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0, SRC_SMU1 and SRC_SMU2.</p> <p>SMU_SELECT_INT0_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0 and SRC_SMU2.</p> <p>SMU_SELECT_INT1: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU1.</p> <p>SMU_SELECT_INT1_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU1 and SRC_SMU2.</p> <p>SMU_SELECT_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU2.</p> <p>SMU_SELECT_INT_NONE: The configuration parameter literal defines the output value of the interrupt request vector as no interrupt selected.</p>		
Default value	SMU_SELECT_INT_NONE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.14 SmuCoreInterruptSet2
Table 195 Specification for SmuCoreInterruptSet2

Name	SmuCoreInterruptSet2		
Description	<p>The configuration parameter defines the output value of the interrupt request vector when the alarm configuration flag selects the interrupt configuration set 2.</p> <p>The default value is this parameter is set to the reset value of the corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef

SMU driver
Table 195 Specification for SmuCoreInterruptSet2 (continued)

Range	SMU_SELECT_INT0: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0. SMU_SELECT_INT0_INT1: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0 and SRC_SMU1 . SMU_SELECT_INT0_INT1_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0, SRC_SMU1 and SRC_SMU2. SMU_SELECT_INT0_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU0, SRC_SMU2. SMU_SELECT_INT1: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU1. SMU_SELECT_INT1_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU1 and SRC_SMU2. SMU_SELECT_INT2: The configuration parameter literal defines the output value of the interrupt request vector as SRC_SMU2. SMU_SELECT_INT_NONE: The configuration parameter literal defines the output value of the interrupt request vector as no interrupt selected.		
Default value	SMU_SELECT_INT_NONE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.5.15 SmuCoreNMIActivatePES
Table 196 Specification for SmuCoreNMIActivatePES

Name	SmuCoreNMIActivatePES		
Description	The configuration parameter defines the control of the Port Emergency Stop (PES) feature for NMI internal action. When an NMI internal action is triggered, the hardware triggers automatically the PES, on enabling. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	Post-build variant multiplicity	-

SMU driver
Table 196 Specification for SmuCoreNMIActivatePES (continued)

Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.6 Container: SmuCoreAlarmGroup

The container contains the configuration parameters for SMU_core alarm groups.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.6.1 SmuCoreAlmGrpId
Table 197 Specification for SmuCoreAlmGrpId

Name	SmuCoreAlmGrpId		
Description	The configuration parameter defines group ID of the SMU alarm group. The value will be assigned to the symbolic name derived from the AlarmGroup container short name. First alarm group is selected as the default value.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 11		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.7 Container: SmuCoreConfig

The container contains the configuration parameters related to SMU_core.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.8 Container: SmuCoreFSPHandling

The container contains the configuration parameters related to SMU_core FSP handling.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

SMU driver

3.3.1.8.1 SmuCoreFSPFaultStateDuration

Table 198 Specification for SmuCoreFSPFaultStateDuration

Name	SmuCoreFSPFaultStateDuration		
Description	<p>The configuration parameter enables the maximum fault state duration of FSP signal. The fault duration value is set at bit field, TFSP_HIGH of FSP Register. The configuration parameter is specified as a number of SMU_FS ticks.</p> <p>The default value is the intermediate value of the SFR.</p>		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 1023		
Default value	1		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.8.2 SmuCoreFSPPrescaler1

Table 199 Specification for SmuCoreFSPPrescaler1

Name	SmuCoreFSPPrescaler1		
Description	<p>The configuration parameter defines the dividing factor to apply to the reference clock fBACK. The divided clock is used as reference to generate the timing of the fault signaling protocol fault state. The frequency of the divided clock is F(SMU_FS).</p> <p>The default value of the parameter is set to the reset value of corresponding SFR.</p>		
Multiplicity	1..1	Type	EcucEnumerationParamDef

SMU driver
Table 199 Specification for SmuCoreFSPPrescaler1 (continued)

Range	SMU_REF_CLK_FRQ_DIV_128: The configuration parameter literal defines that the reference clock frequency is divided by 128. SMU_REF_CLK_FRQ_DIV_16: The configuration parameter literal defines that the reference clock frequency is divided by 16. SMU_REF_CLK_FRQ_DIV_256: The configuration parameter literal defines that the reference clock frequency is divided by 256. SMU_REF_CLK_FRQ_DIV_2: The configuration parameter literal defines that the reference clock frequency is divided by 2. SMU_REF_CLK_FRQ_DIV_32: The configuration parameter literal defines that the reference clock frequency is divided by 32. SMU_REF_CLK_FRQ_DIV_4: The configuration parameter literal defines that the reference clock frequency is divided by 4. SMU_REF_CLK_FRQ_DIV_64: The configuration parameter literal defines that the reference clock frequency is divided by 64. SMU_REF_CLK_FRQ_DIV_8: The configuration parameter literal defines that the reference clock frequency is divided by 8.		
Default value	SMU_REF_CLK_FRQ_DIV_2		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.8.3 SmuCoreFSPPrescaler2
Table 200 Specification for SmuCoreFSPPrescaler2

Name	SmuCoreFSPPrescaler2		
Description	The configuration parameter defines the dividing factor to apply to the reference clock fBACK. The divided clock is used as reference to generate the timing of the fault free state for the dynamic dual rail and time switching modes of the fault signaling protocol. The frequency of the divided clock is F(SMU_FFS). The default value of the parameter is set to the reset value of corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef

SMU driver
Table 200 Specification for SmuCoreFSPPrescaler2 (continued)

Range	SMU_REF_CLK_FRQ_DIV_1024: The configuration parameter literal defines that the reference clock frequency is divided by 1024. SMU_REF_CLK_FRQ_DIV_2048: The configuration parameter literal defines that the reference clock frequency is divided by 2048. SMU_REF_CLK_FRQ_DIV_4096: The configuration parameter literal defines that the reference clock frequency is divided by 4096. SMU_REF_CLK_FRQ_DIV_512: The configuration parameter literal defines that the reference clock frequency is divided by 512.		
Default value	SMU_REF_CLK_FRQ_DIV_512		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.8.4 SmuCoreFSPSignalingMode
Table 201 Specification for SmuCoreFSPSignalingMode

Name	SmuCoreFSPSignalingMode		
Description	The configuration parameter defines the type of signal output for FSP on an alarm event. The default value of the parameter is set to the reset value of corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	SMU_FSP_BISTABLE_PROTOCOL: The configuration parameter literal defines that bistable protocol is used for FSP handling. SMU_FSP_DUAL_RAIL_PROTOCOL: The configuration parameter literal defines that dual rail protocol is used for FSP handling. SMU_FSP_TIME_SWITCHING_PROTOCOL: The configuration parameter literal defines that time switching protocol is used for FSP handling.		
Default value	SMU_FSP_BISTABLE_PROTOCOL		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

SMU driver
3.3.1.8.5 SmuCorePESOnFSP
Table 202 Specification for SmuCorePESOnFSP

Name	SmuCorePESOnFSP		
Description	The configuration parameter defines whether the PES is to be automatically requested when an alarm event configured to start the FSP is detected. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	SMU_FSP_PES_DISABLE: The configuration parameter literal defines that PES feature is disabled. SMU_FSP_PES_ENABLE: The configuration parameter literal defines that PES feature is enabled.		
Default value	SMU_FSP_PES_DISABLE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.9 Container: SmuCoreRecoveryTimer

The container contains the configuration parameters for SMU_core recovery timer.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.9.1 SmuCoreEnableRT0
Table 203 Specification for SmuCoreEnableRT0

Name	SmuCoreEnableRT0		
Description	The configuration parameter defines whether RT0 is enabled or disabled. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	SMU_RT_DISABLE: The configuration parameter literal defines that the recovery timer is disabled. Value: 0 SMU_RT_ENABLE: The configuration parameter literal defines that the recovery timer is enabled. Value: 1		
Default value	SMU_RT_DISABLE		

SMU driver
Table 203 Specification for SmuCoreEnableRT0 (continued)

Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.9.2 SmuCoreEnableRT1

Table 204 Specification for SmuCoreEnableRT1

Name	SmuCoreEnableRT1		
Description	The configuration parameter defines whether RT1 is enabled or disabled. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	SMU_RT_DISABLE: The configuration parameter literal defines that the recovery timer is disabled. Value: 0 SMU_RT_ENABLE: The configuration parameter literal defines that the recovery timer is enabled. Value: 1		
Default value	SMU_RT_DISABLE		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.9.3 SmuCoreRTDuration

Table 205 Specification for SmuCoreRTDuration

Name	SmuCoreRTDuration		
Description	The configuration parameter defines the maximum duration of SMU_core recovery timer. The maximum duration is specified as a number of the F(SMU_FS) clock ticks. The default value is the intermediate value of the tick duration.		
Multiplicity	1..1	Type	EcucIntegerParamDef

SMU driver
Table 205 **Specification for SmuCoreRTDuration (continued)**

Range	0 - 0xFFFFFFFFU		
Default value	0x3FFFU		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.10 **Container: SmuCoreRT0Alarm**

The container enables to select the alarms for RT0. Four alarms can be configured per recovery timer instance.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.10.1 **SmuCoreRT0AlarmGroupId**

Table 206 **Specification for SmuCoreRT0AlarmGroupId**

Name	SmuCoreRT0AlarmGroupId		
Description	The configuration parameter defines the alarm group ID associated with the RT0. First alarm id is selected as the default value.		
Multiplicity	1..1	Type	EcucEnumerationParamDef

SMU driver
Table 206 Specification for SmuCoreRT0AlarmGroupId (continued)

Range	SMU_ALARM_GROUP0: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 0. SMU_ALARM_GROUP10: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 10. SMU_ALARM_GROUP11: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 11. SMU_ALARM_GROUP1: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 1. SMU_ALARM_GROUP2: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 2. SMU_ALARM_GROUP3: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 3. SMU_ALARM_GROUP4: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 4. SMU_ALARM_GROUP5: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 5. SMU_ALARM_GROUP6: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 6. SMU_ALARM_GROUP7: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 7. SMU_ALARM_GROUP8: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 8. SMU_ALARM_GROUP9: The configuration parameter literal defines that the alarm group ID associated with the RT0 corresponds to alarm group 9.		
Default value	SMU_ALARM_GROUP0		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuCoreEnableRT0		

3.3.1.10.2 SmuCoreRT0AlarmId
Table 207 Specification for SmuCoreRT0AlarmId

Name	SmuCoreRT0AlarmId		
Description	The configuration parameter defines the alarm ID associated with the RT0. First alarm id is selected as the default value..		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 31		
Default value	0		

SMU driver
Table 207 **Specification for SmuCoreRT0AlarmId (continued)**

Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuCoreEnableRT0		

3.3.1.11 **Container: SmuCoreRT1Alarm**

The container enables to select the alarms for RT1. Four alarms can be configured per recovery timer instance.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.11.1 **SmuCoreRT1AlarmGroupId**

Table 208 **Specification for SmuCoreRT1AlarmGroupId**

Name	SmuCoreRT1AlarmGroupId		
Description	The configuration parameter defines the alarm group ID associated with the RT1. First alarm group is selected as the default value.		
Multiplicity	1..1	Type	EcucEnumerationParamDef

SMU driver
Table 208 Specification for SmuCoreRT1AlarmGroupId (continued)

Range	SMU_ALARM_GROUP0: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 0. SMU_ALARM_GROUP10: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 10. SMU_ALARM_GROUP11: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 11. SMU_ALARM_GROUP1: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 1. SMU_ALARM_GROUP2: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 2. SMU_ALARM_GROUP3: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 3. SMU_ALARM_GROUP4: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 4. SMU_ALARM_GROUP5: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 5. SMU_ALARM_GROUP6: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 6. SMU_ALARM_GROUP7: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 7. SMU_ALARM_GROUP8: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 8. SMU_ALARM_GROUP9: The configuration parameter literal defines that the alarm group ID associated with the RT1 corresponds to alarm group 9.		
Default value	SMU_ALARM_GROUP0		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuCoreEnableRT1		

3.3.1.11.2 SmuCoreRT1AlarmId
Table 209 Specification for SmuCoreRT1AlarmId

Name	SmuCoreRT1AlarmId		
Description	The configuration parameter defines the alarm ID associated with the RT1. First alarm id is selected as the default value.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 31		
Default value	0		

SMU driver
Table 209 Specification for SmuCoreRT1AlarmId (continued)

Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuCoreEnableRT1		

3.3.1.12 Container: SmuDemEventParameterRefsConf

The container lists down the production errors supported by the SMU driver.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Post-Build

3.3.1.12.1 SmuActivateFSPFailureNotification

Table 210 Specification for SmuActivateFSPFailureNotification

Name	SmuActivateFSPFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to failure to activate FSP is enabled or not		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.2 SmuActivatePESFailureNotification

Table 211 Specification for SmuActivatePESFailureNotification

Name	SmuActivatePESFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to failure of PES is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef

SMU driver
Table 211 Specification for SmuActivatePESFailureNotification (continued)

Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.3 SmuActivateRunStateFailureNotification
Table 212 Specification for SmuActivateRunStateFailureNotification

Name	SmuActivateRunStateFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to failure to activate RUN state is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.4 SmuClearAlarmStatusFailureNotification
Table 213 Specification for SmuClearAlarmStatusFailureNotification

Name	SmuClearAlarmStatusFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to failure to clear alarm status is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		

SMU driver
Table 213 Specification for SmuClearAlarmStatusFailureNotification (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.5 SmuCoreAliveFailureNotification
Table 214 Specification for SmuCoreAliveFailureNotification

Name	SmuCoreAliveFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to SMU_core_alive test failure is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.6 SmuRTStopFailureNotification
Table 215 Specification for SmuRTStopFailureNotification

Name	SmuRTStopFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to failure to stop recovery timer is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE

SMU driver

Table 215 **Specification for SmuRTStopFailureNotification (continued)**

Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.7 **SmuReleaseFSPFailureNotification**

Table 216 **Specification for SmuReleaseFSPFailureNotification**

Name	SmuReleaseFSPFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to failure to release FSP is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.8 **SmuSetAlarmStatusFailureNotification**

Table 217 **Specification for SmuSetAlarmStatusFailureNotification**

Name	SmuSetAlarmStatusFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to failure to set alarm status is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile

SMU driver
Table 217 Specification for SmuSetAlarmStatusFailureNotification (continued)

Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.12.9 SmuSffFailureNotification
Table 218 Specification for SmuSffFailureNotification

Name	SmuSffFailureNotification		
Description	The configuration parameter tells whether the notification for DEM related to SFF test failure is enabled or disabled.		
Multiplicity	0..1	Type	EcucSymbolicNameReferenceDef
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	Post-build variant multiplicity	FALSE
Value configuration class	Pre-Compile	Multiplicity configuration class	Pre-Compile
Origin	IFX	Scope	LOCAL
Dependency	SmuSafetyEnable		

3.3.1.13 Container: SmuGeneral

The container contains the general configurations of SMU driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.13.1 SmuAGStatusTimeout
Table 219 Specification for SmuAGStatusTimeout

Name	SmuAGStatusTimeout		
Description	The configuration parameter defines the timeout value to check for the alarm group status within the defined time frame. The default value of the timeout is the intermediate value.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	4096 - 65535		
Default value	4096		
Post-build variant value	FALSE	Post-build variant multiplicity	-

SMU driver
Table 219 Specification for SmuAGStatusTimeout (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.2 SmuCoreFSP0OutputEnable
Table 220 Specification for SmuCoreFSP0OutputEnable

Name	SmuCoreFSP0OutputEnable		
Description	The configuration parameter sets FSP[0] state to output, if true. The default value is this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.3 SmuCoreFSP0PortEnable
Table 221 Specification for SmuCoreFSP0PortEnable

Name	SmuCoreFSP0PortEnable		
Description	The configuration parameter sets FSP[0] PORT enable to true. The default value is this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-

SMU driver
Table 221 Specification for SmuCoreFSP0PortEnable (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.4 SmuCoreFSP1OutputEnable
Table 222 Specification for SmuCoreFSP1OutputEnable

Name	SmuCoreFSP1OutputEnable		
Description	The configuration parameter sets FSP[1] state to output, if true. The default value is this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.5 SmuCoreFSP1PortEnable
Table 223 Specification for SmuCoreFSP1PortEnable

Name	SmuCoreFSP1PortEnable		
Description	The configuration parameter sets FSP[1] port enable to true. The default value is this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-

SMU driver
Table 223 Specification for SmuCoreFSP1PortEnable (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.6 SmuCoreGlitchFilterSCU
Table 224 Specification for SmuCoreGlitchFilterSCU

Name	SmuCoreGlitchFilterSCU		
Description	The configuration parameter sets glitch filter for SCU to enabled state. The default value is this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.7 SmuCoreGlitchFilterSTS
Table 225 Specification for SmuCoreGlitchFilterSTS

Name	SmuCoreGlitchFilterSTS		
Description	The configuration parameter sets glitch filter for SMU_STS to be enabled. The default value is this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-

SMU driver

Table 225 **Specification for SmuCoreGlitchFilterSTS (continued)**

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.8 SmuDevErrorDetect

Table 226 **Specification for SmuDevErrorDetect**

Name	SmuDevErrorDetect		
Description	The configuration parameter enables or disables DET checks.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.9 SmuInitCheckApi

Table 227 **Specification for SmuInitCheckApi**

Name	SmuInitCheckApi		
Description	<p>The configuration parameter enables or disables the Smu_InitCheck API.</p> <p>The detection of safety related errors is enabled by default to ensure that safety issues are addressed during the product lifecycle.</p>		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-

SMU driver
Table 227 Specification for SmuInitCheckApi (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.10 SmuInitDeInitApiMode
Table 228 Specification for SmuInitDeInitApiMode

Name	SmuInitDeInitApiMode		
Description	The configuration parameter defines the mode in which the Init and Deinit API will be used. Since SMU driver accesses the SFRs, it is more efficient to operate the SMU driver in supervisor mode. Hence, the default mode of operation is supervisor.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	SMU_MCAL_SUPERVISOR: The configuration parameter implies that SUPERVISOR mode is used. The parameter takes value 0 when assigned. SMU_MCAL_USER1: The configuration parameter implies that USER1 mode is used. The parameter takes values 1 when assigned.		
Default value	SMU_MCAL_SUPERVISOR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.11 SmuRuntimeApiMode
Table 229 Specification for SmuRuntimeApiMode

Name	SmuRuntimeApiMode		
Description	The configuration parameter gives the mode in which the runtime API will be used. Since SMU driver accesses the SFRs, it is more efficient to operate the SMU driver in supervisor mode. Hence, the default mode of operation is supervisor. When the parameter is in supervisor mode, then the SmuInitDeIntMode is in supervisor mode.		
Multiplicity	1..1	Type	EcucEnumerationParamDef

SMU driver
Table 229 Specification for SmuRuntimeApiMode (continued)

Range	SMU_MCAL_SUPERVISOR: The configuration parameter implies that SUPERVISOR mode is used. The parameter takes value 0 when assigned. SMU_MCAL_USER1: The configuration parameter implies that USER1 mode is used. The parameter takes value 1 when assigned.		
Default value	SMU_MCAL_SUPERVISOR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmulnitDelnitApiMode		

3.3.1.13.12 SmuSafetyEnable

Table 230 Specification for SmuSafetyEnable

Name	SmuSafetyEnable		
Description	The configuration parameter defines whether the safety checks mandated by safety standards are enabled or disabled. The detection of safety related errors is enabled by default to ensure that safety issues are addressed during the product lifecycle.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.13 SmuStdbbyEnable

Table 231 Specification for SmuStdbbyEnable

Name	SmuStdbbyEnable		
Description	The configuration parameter defines whether the SMU_stdbby unit is enabled or disabled. The default value is this parameter is set to the reset value of the corresponding SFR.		

SMU driver
Table 231 Specification for SmuStdbyEnable (continued)

Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.13.14 SmuVersionInfoApi
Table 232 Specification for SmuVersionInfoApi

Name	SmuVersionInfoApi		
Description	The configuration parameter enables or disables the VersionInfo API. The optional features are disabled by default to minimize the executable code size.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

3.3.1.14 Container: SmuStdbyAlarmBehavior

The container contains configuration parameters corresponding to alarm behavior. The behavior type is external if FSP is enabled or no reaction.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

SMU driver
3.3.1.14.1 SmuStdbbyAlarmFSP
Table 233 Specification for SmuStdbbyAlarmFSP

Name	SmuStdbbyAlarmFSP		
Description	The configuration parameter defines whether the FSP is enabled or disabled. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	SMU_ALARM_FSP_DISABLED: The configuration parameter literal defines that FSP is disabled. SMU_ALARM_FSP_ENABLED: The configuration parameter literal defines that the FSP is enabled.		
Default value	SMU_ALARM_FSP_DISABLED		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuStdbbyEnable		

3.3.1.14.2 SmuStdbbyAlmBehaviourId
Table 234 Specification for SmuStdbbyAlmBehaviourId

Name	SmuStdbbyAlmBehaviourId		
Description	The configuration parameter defines the alarm ID corresponding to the particular group. First alarm behavior id is selected as the default value..		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 31		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuStdbbyEnable		

3.3.1.15 Container: SmuStdbbyAlarmGlobalConfig

The container contains the configuration parameters related to SMU_stdby global configurations.

Post-Build Variant Multiplicity: -

SMU driver

Multiplicity Configuration Class: -

3.3.1.15.1 SmuStdbyEnableFSP0
Table 235 Specification for SmuStdbyEnableFSP0

Name	SmuStdbyEnableFSP0		
Description	The configuration parameter defines whether the use of FSP[0] P33.8 pin is enabled or disabled for FSP handling. The default value of the parameter is the reset value of the SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuStdbyEnable		

3.3.1.15.2 SmuStdbyEnableFSP1
Table 236 Specification for SmuStdbyEnableFSP1

Name	SmuStdbyEnableFSP1		
Description	The configuration parameter defines whether the use of FSP[1] P33.10 pin is enabled or disabled for FSP handling. The default value of this parameter is set to the reset value of the corresponding SFR.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuStdbyEnable		

SMU driver
3.3.1.16 Container: SmuStdbbyAlarmGroup

The container contains the configuration parameters related to SMU_stdbby alarm groups.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.1.16.1 SmuStdbbyAlmGrpId
Table 237 Specification for SmuStdbbyAlmGrpId

Name	SmuStdbbyAlmGrpId		
Description	The configuration parameter defines the alarm group ID of the alarm group to be configured. First group id is selected as the default value.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 31		
Default value	0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	SmuStdbbyEnable		

3.3.1.17 Container: SmuStdbbyConfig

The container contains the configuration parameters related to SMU_stdbby.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

3.3.2 Functions - Type definitions
3.3.2.1 Smu_AlarmGroupId
Table 238 Specification for Smu_AlarmGroupId

Syntax	Smu_AlarmGroupId
Type	Enumeration
File	Smu.h

SMU driver
Table 238 **Specification for Smu_AlarmGroupId (continued)**

Range	0 - SMU_GROUP_0	
	1 - SMU_GROUP_1	
	2 - SMU_GROUP_2	
	3 - SMU_GROUP_3	
	4 - SMU_GROUP_4	
	5 - SMU_GROUP_5	
	6 - SMU_GROUP_6	
	7 - SMU_GROUP_7	
	8 - SMU_GROUP_8	
	9 - SMU_GROUP_9	
	10 - SMU_GROUP_10	
	11 - SMU_GROUP_11	
	20 - SMU_GROUP_20	
	21 - SMU_GROUP_21	
Description	Smu_AlarmGroupId enumeration gives the alarm group ID for each group in SMU_core and SMU_stdby.	
Source	IFX	

3.3.2.2 **Smu_AlarmIdType**

Table 239 **Specification for Smu_AlarmIdType**

Syntax	Smu_AlarmIdType
Type	Enumeration
File	Smu.h

SMU driver
Table 239 **Specification for Smu_AlarmIdType (continued)**

Range	0 - SMU_ALARM_0	
	1 - SMU_ALARM_1	
	2 - SMU_ALARM_2	
	3 - SMU_ALARM_3	
	4 - SMU_ALARM_4	
	5 - SMU_ALARM_5	
	6 - SMU_ALARM_6	
	7 - SMU_ALARM_7	
	8 - SMU_ALARM_8	
	9 - SMU_ALARM_9	
	10 - SMU_ALARM_10	
	11 - SMU_ALARM_11	
	12 - SMU_ALARM_12	
	13 - SMU_ALARM_13	
	14 - SMU_ALARM_14	
	15 - SMU_ALARM_15	
	16 - SMU_ALARM_16	
	17 - SMU_ALARM_17	
	18 - SMU_ALARM_18	
	19 - SMU_ALARM_19	
	20 - SMU_ALARM_20	
	21 - SMU_ALARM_21	
	22 - SMU_ALARM_22	
	23 - SMU_ALARM_23	
	24 - SMU_ALARM_24	
	25 - SMU_ALARM_25	
	26 - SMU_ALARM_26	
	27 - SMU_ALARM_27	
	28 - SMU_ALARM_28	
	29 - SMU_ALARM_29	
	30 - SMU_ALARM_30	
	31 - SMU_ALARM_31	
Description	Smu_AlarmIdType enumeration gives the alarm ID associated with each alarm group.	
Source	IFX	

SMU driver**3.3.2.3 Smu_ConfigType****Table 240 Specification for Smu_ConfigType**

Syntax	Smu_ConfigType	
Type	Structure	
File	Smu.h	
Range	-	-
Description	Defines the type for data structure containing the set of configuration parameters required for initializing the SMU driver.	
Source	IFX	

3.3.2.4 Smu_CoreAlarmActionType**Table 241 Specification for Smu_CoreAlarmActionType**

Syntax	Smu_CoreAlarmActionType	
Type	uint8	
File	Smu.h	

SMU driver
Table 241 **Specification for Smu_CoreAlarmActionType (continued)**

Range	0-SMU_ALARM_ACTION_NONE	SMU_NA_ALARM_CONFIG implies that no action has to be taken on receiving an alarm.
	1-SMU_ALARM_ACTION_RSVD	SMU_RSVD_ALARM_CONFIG is reserved and no action is taken. Alarm is disabled.
	2-SMU_ALARM_ACTION_IGCS0	SMU_IGCS0_ALARM_CONFIG sends an interrupt request to the interrupt system according to the Interrupt Generation Configuration Set 0 from AGC register.
	3-SMU_ALARM_ACTION_IGCS1	SMU_IGCS1_ALARM_CONFIG sends an interrupt request to the interrupt system according to the Interrupt Generation Configuration Set 1 from AGC register.
	4-SMU_ALARM_ACTION_IGCS2	SMU_IGCS2_ALARM_CONFIG sends an interrupt request to the interrupt system according to the Interrupt Generation Configuration Set 2 from AGC register.
	5-SMU_ALARM_ACTION_NMI	SMU_NMI_ALARM_CONFIG sends an NMI request to the SCU.
	6-SMU_ALARM_ACTION_RESET	SMU_RESET_ALARM_CONFIG sends a RESET request to the SCU. SCU shall be configured to generate an application or system reset.
	7-SMU_ALARM_ACTION_CPU_RESET	SMU_CPU_RST_ALARM_CONFIG sets a CPU reset using CPU Reset Configuration set from the AGC register.
	255-SMU_INVALID_ALARM_ACTION	SMU_INVALID_ALARM_CONFIG implies that alarm action is invalid.
Description	Smu_CoreAlarmActionType defines the internal action behavior for the alarms in SMU_Core.	
Source	IFX	

3.3.2.5 Smu_CoreCommandType

Table 242 **Specification for Smu_CoreCommandType**

Syntax	Smu_CoreCommandType
Type	uint8
File	Smu.h

SMU driver
Table 242 Specification for Smu_CoreCommandType (continued)

Range	0-SMU_RUN_COMMAND	SMU_RUN_COMMAND makes the SMU_core enter the RUN state.
	1-SMU_ACTIVATEFSP_COMMAND	SMU_ACTIVATEFSP_COMMAND activates FSP for SMU_core.
	2-SMU_RELEASEFSP_COMMAND	SMU_RELEASEFSP_COMMAND releases FSP for SMU_core.
	3-SMU_ACTIVATE_PES	SMU_ACTIVATE_PES activates the PES feature for SMU_core.
	4-SMU_STOPREC_COMMAND	SMU_STOPREC_COMMAND stops the recovery timer for SMU_core.
	5-SMU_ASCE_COMMAND	SMU_ASCE_COMMAND is alarm status clear enable command for SMU_core. Software shall execute SMU_ASCE_COMMAND prior to clearing of a AG _n alarm status bit. SMU_ASCE_COMMAND sets the ASCE bit in the STS register.
	6-SMU_ALARM_COMMAND	SMU_ALARM_COMMAND triggers a software-based alarm. ARG specifies the alarm index.
	7-SMU_ALIVETEST_COMMAND	SMU_ALIVETEST_COMMAND enables the testing of the smu_core_alive signal. Sending SMU_ALIVETEST_COMMAND will forward the smu_core_alive alarm to the SMU_stdby. Argument ARG shall be set to 0x05 to start the test and to 0x0A to end the test.
Description	Smu_CoreCommandType describes the SMU_core command sets.	
Source	IFX	

3.3.2.6 Smu_CoreStateType

Table 243 Specification for Smu_CoreStateType

Syntax	Smu_CoreStateType
Type	uint8
File	Smu.h

SMU driver
Table 243 **Specification for Smu_CoreStateType (continued)**

Range	0-SMU_START_STATE	SMU_START_STATE corresponds to the START state in the SSM. Value:0. The alarms shall be logged in but not processed during START state. Exception to this is the RT and Watchdog timeout alarms. Entry condition: PORST Exit condition: Releasing FSP and activating RUN state through SMU_core command.
	1-SMU_RUN_STATE	SMU_RUN_STATE corresponds to the RUN state in the SSM. Value:1 The alarms logged are processed. Entry condition: When FSP is released, RUN state is activated through SMU_core commands or FSP fault state timing is expired. Exit condition: When FSP is activated or alarm detected with FSP enabled.
	2-SMU_FAULT_STATE	SMU_FAULT_STATE corresponds to the FAULT state in the SSM. Value:2 SMU input alarm events are processed according to their configurations. FSP drives according to the configured reaction and timing. If a new FSP is detected, the FSP fault state timing is restarted. Entry condition: When alarm is detected. Exit condition: When FSP is released or FSP fault state timing is expired.
	3-SMU_UNDEFINED_STATE	SMU_UNDEFINED_STATE corresponds to the UNDEFINED state in the SSM. Value:3
Description	Smu_CoreStateType defines the various state types of SMU State Machine (SSM).	
Source	IFX	

3.3.2.7 **Smu_EnableRunStateType**

Table 244 **Specification for Smu_EnableRunStateType**

Syntax	Smu_EnableRunStateType	
Type	Enumeration	
File	Smu.h	
Range	0 - SMU_EFRST_DISABLE	The Enable Fault To RUN state is disabled
	1 - SMU_EFRST_ENABLE	The Enable Fault To RUN state is enabled.

SMU driver
Table 244 Specification for Smu_EnableRunStateType (continued)

Description	Smu_EnableRunStateType enumeration defines whether the fault to run state is enabled or disabled.
Source	IFX

3.3.2.8 Smu_FSPActionType

Table 245 Specification for Smu_FSPActionType

Syntax	Smu_FSPActionType	
Type	uint32	
File	Smu.h	
Range	0 - 1	
Description	Smu_FSPActionType defines the FSP action type for the alarm group and ID.	
Source	IFX	

3.3.2.9 Smu_SffTestResType

Table 246 Specification for Smu_SffTestResType

Syntax	Smu_SffTestResType	
Type	uint8	
File	Smu.h	
Range	0 - 255	-
Description	Smu_SffTestResType gives the SFF test results.	
Source	IFX	

3.3.3 Functions - APIs

3.3.3.1 Smu_Init

Table 247 Specification for Smu_Init API

Syntax	Std_ReturnType Smu_Init (const Smu_ConfigType * const ConfigPtr)	
Service ID	0xA8	
Sync/Async	Synchronous	
ASIL Level	B(D)	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to the SMU configuration for initialization

SMU driver
Table 247 **Specification for Smu_Init API (continued)**

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	<p>E_OK: Operation successful that is initialization of resources of AURIX SMU peripheral is successful.</p> <p>E_NOT_OK: Operation failed that is initialization of resources of AURIX SMU peripheral is not successful, for example when driver is already initialized.</p>
Description	<p>The purpose of the API is to setup the SMU peripheral based on the configuration. The SMU driver initializes the resources of the AURIX SMU peripheral, for example the error reaction and the Fault Signaling Protocol (FSP). Initialization should be done only from the master core. During initialization, all the alarm status are cleared, hence the user must ensure to keep a track of the alarm status before Smu_Init is called.</p>	
Source	IFX	
Error handling	<p>DET:</p> <p>SMU_E_ALREADY_INITIALIZED: SMU_E_ALREADY_INITIALIZED DET is reported when SMU is already initialized.</p> <p>SMU_E_INIT_FAILED: SMU_E_INIT_FAILED DET is reported when initialization of SMU driver fails due to incorrect configuration parameter.</p> <p>SMU_E_CORE_MISMATCH: SMU_E_CORE_MISMATCH DET is reported when the Init or De-Init is called from any core other than the master core.</p> <p>SMU_E_LOCKED: SMU_E_LOCKED DET is reported when the SMU is already in locked state.</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>	
Configuration dependencies	-	
User hints	None	

3.3.3.2 Smu_DeInit

Table 248 **Specification for Smu_DeInit API**

Syntax	Std_ReturnType Smu_DeInit (void)
Service ID	0xAA
Sync/Async	Synchronous
ASIL Level	B
Re-entrancy	Non Reentrant

SMU driver
Table 248 **Specification for Smu_DeInit API (continued)**

Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful: de-initialization of SMU driver by resetting the module registers is successful. E_NOT_OK: Operation failed that is de-initialization of SMU driver by resetting the module registers is not successful, for example when driver is already reset.
Description	The purpose of the API is to de-initialize the SMU driver by resetting the module registers. De-initialization shall be done only from the master core.	
Source	IFX	
Error handling	DET: SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_CORE_MISMATCH: SMU_E_CORE_MISMATCH DET is reported when the Init or De-Init is called from any core other than the master core. SMU_E_LOCKED: SMU_E_LOCKED DET is reported when the SMU is already in locked state. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.3 Smu_GetAlarmAction

Table 249 **Specification for Smu_GetAlarmAction API**

Syntax	Std_ReturnType Smu_GetAlarmAction (const Smu_AlarmGroupId AlarmGroup, const Smu_AlarmIdType AlarmPos, Smu_CoreAlarmActionType * const IntAlarmAction, Smu_FSPActionType * const FSPAction)
Service ID	0xAB
Sync/Async	Synchronous
ASIL Level	B
Re-entrancy	Reentrant

SMU driver
Table 249 **Specification for Smu_GetAlarmAction API (continued)**

Parameters (in)	AlarmGroup AlarmPos	Alarm group number (0 - 11, 20, 21) Alarm position within the requested group(0 - 31)
Parameters (out)	IntAlarmAction FSPAction	Alarm action for the requested alarm FSP action for the requested alarm.(0- Disabled , 1- Enabled)
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is retrieval of the internal alarm, FSP action currently configured for the requested alarm is successful. E_NOT_OK: Operation not successful that is retrieval of the internal alarm, FSP action currently configured for the requested alarm is not successful, for example due to invalid parameters.
Description	The purpose of the API is to provide the internal alarm, FSP action currently configured for the requested alarm.	
Source	IFX	
Error handling	DET: SMU_E_PARAM_POINTER: SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer. SMU_E_PARAM_GROUP: SMU_E_PARAM_GROUP DET is reported when the group ID or the alarm position passed as a parameter to an API is not valid. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_STDBY_DISABLED: SMU_E_STDBY_DISABLED DET is reported when any alarm action is configured or performed with respect to the standby domain of SMU without enabling the SMU standby mode. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.4 **Smu_SetAlarmAction**

Table 250 **Specification for Smu_SetAlarmAction API**

Syntax	<code>Std_ReturnType Smu_SetAlarmAction (const Smu_AlarmGroupId AlarmGroup, const Smu_AlarmIdType AlarmPos, const Smu_CoreAlarmActionType AlarmAction, const Smu_FSPActionType FSPAction)</code>
Service ID	0xAC

SMU driver
Table 250 **Specification for Smu_SetAlarmAction API (continued)**

Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	AlarmGroup AlarmPos AlarmAction FSPAction	Alarm group number (0 - 11, 20, 21) Alarm position within the requested group (0-31) The internal alarm action for the requested alarm group and position. This parameter gives the FSP action to be set.(0- Disabled, 1- Enabled)
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: The alarm action is set. E_NOT_OK: The alarm action could not be set.
Description	The purpose of the API is to set the desired alarm action for the group and position specified.	
Source	IFX	
Error handling	DET: SMU_E_PARAM_GROUP: SMU_E_PARAM_GROUP DET is reported when the group ID or the alarm position passed as a parameter to an API is not valid. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. SMU_E_LOCKED: SMU_E_LOCKED DET is reported when the SMU is already in locked state. SMU_E_INVALID_ALARM_ACTION: SMU_E_INVALID_ALARM_ACTION DET is reported when the alarm action to be configured is not valid. SMU_E_STDBY_DISABLED: SMU_E_STDBY_DISABLED DET is reported when any alarm action is configured or performed with respect to the standby domain of SMU without enabling the SMU standby mode. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

SMU driver
3.3.3.5 Smu_ClearAlarmStatus
Table 251 Specification for Smu_ClearAlarmStatus API

Syntax	Std_ReturnType Smu_ClearAlarmStatus (const Smu_AlarmGroupId AlarmGroup, const Smu_AlarmIdType AlarmPos)	
Service ID	0xAD	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	AlarmGroup AlarmPos	Alarm group number (0 - 11, 20, 21) Alarm position within the requested group(0 - 31)
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: The alarm status is cleared successfully E_NOT_OK: The alarm status is not cleared successfully
Description	The purpose of the API is to clear SMU alarm status of the requested alarm.	
Source	IFX	
Error handling	DET: SMU_E_PARAM_GROUP: SMU_E_PARAM_GROUP DET is reported when the group ID or the alarm position passed as a parameter to an API is not valid. SMU_E_STDBY_DISABLED: SMU_E_STDBY_DISABLED DET is reported when any alarm action is configured or performed with respect to the standby domain of SMU without enabling the SMU standby mode. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. Runtime Errors: None DEM: SMU_E_CLEAR_ALARM_STATUS_FAILURE: SMU_E_CLEAR_ALARM_STATUS_FAILURE DEM is reported when the clearing of alarm status fails. Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

SMU driver
3.3.3.6 Smu_GetAlarmStatus
Table 252 Specification for Smu_GetAlarmStatus API

Syntax	Std_ReturnType Smu_GetAlarmStatus (const Smu_AlarmGroupId AlarmGroup, uint32 * const AlarmStatus)	
Service ID	0xAF	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	AlarmGroup	Group id of the alarm raised. (0 - 11, 20, 21)
Parameters (out)	AlarmStatus	Status of the alarm raised
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is retrieval of SMU alarm status of the requested alarm is successful. E_NOT_OK: Operation unsuccessful that is SMU alarm status of the requested alarm is could not be retrieved.
Description	The purpose of the API is to provide the alarm status of the requested alarm group.	
Source	IFX	
Error handling	DET: SMU_E_PARAM_POINTER: SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer. SMU_E_PARAM_GROUP: SMU_E_PARAM_GROUP DET is reported when the group ID or the alarm position passed as a parameter to an API is not valid. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_STDBY_DISABLED: SMU_E_STDBY_DISABLED DET is reported when any alarm action is configured or performed with respect to the standby domain of SMU without enabling the SMU standby mode. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

SMU driver
3.3.3.7 Smu_SetAlarmStatus
Table 253 Specification for Smu_SetAlarmStatus API

Syntax	Std_ReturnType Smu_SetAlarmStatus (const Smu_AlarmGroupId AlarmGroup, const Smu_AlarmIdType AlarmPos)	
Service ID	0xAE	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	AlarmGroup AlarmPos	Alarm group number (0 - 11) Alarm position within the requested group(0-31)
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is SMU alarm status of the requested alarm is set successfully. E_NOT_OK: Operation unsuccessful that is SMU alarm status of the requested alarm is could not be set.
Description	The purpose of the API is to set the requested alarm status. This service can be used by the user software to trigger software SMU alarm. For SMU_core during the START state of the SMU, it shall be possible to set any of the alarms. However, during the RUN state, only the software alarms shall be set. The API is applicable only for SMU_core alarm groups and positions.	
Source	IFX	
Error handling	DET: SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. SMU_E_PARAM_GROUP: SMU_E_PARAM_GROUP DET is reported when the group ID or the alarm position passed as a parameter to an API is not valid. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. Runtime Errors: None DEM: SMU_E_SET_ALARM_STATUS_FAILURE: SMU_E_SET_ALARM_STATUS_FAILURE DEM is reported when the setting of alarm status fails. Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

SMU driver
3.3.3.8 Smu_GetAlarmDebugStatus
Table 254 Specification for Smu_GetAlarmDebugStatus API

Syntax	Std_ReturnType Smu_GetAlarmDebugStatus (const Smu_AlarmGroupId AlarmGroup, uint32 * const AlarmStatus)	
Service ID	0xB0	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	AlarmGroup	Alarm group number (0 - 11)
Parameters (out)	AlarmStatus	Alarm Debug Status register value
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is retrieval of alarm status from debug register is successful. E_NOT_OK: Operation unsuccessful that is SMU alarm debug status of the requested alarm is could not be set.
Description	The purpose of the API is to provide the alarm status for the requested alarm group from the stored debug registers. The debug status is applicable only for SMU_core.	
Source	IFX	
Error handling	DET: SMU_E_PARAM_POINTER: SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer. SMU_E_PARAM_GROUP: SMU_E_PARAM_GROUP DET is reported when the group ID or the alarm position passed as a parameter to an API is not valid. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	This is required by the application to know the reason of the malfunction esp. In case the internal reaction was configured to be a reset. This is only for Smu_core.	

SMU driver
3.3.3.9 Smu_LockConfigRegs
Table 255 Specification for Smu_LockConfigRegs API

Syntax	Std_ReturnType Smu_LockConfigRegs (void)	
Service ID	0xB1	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is SMU configuration registers are locked successfully. E_NOT_OK: Operation not successful that is SMU configuration registers are not locked successfully.
Description	The purpose of the API is to permanently lock the SMU configuration registers to prevent any modification to configuration register content.	
Source	IFX	
Error handling	DET: SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_LOCKED: SMU_E_LOCKED DET is reported when the SMU is already in locked state. Runtime Errors: None DEM: None Safety Errors: SMU_E_SF_CFG_LOCKED: The safety error is reported when the configuration registers do not get locked after applying permanent lock. <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.10 Smu_ReleaseFSP
Table 256 Specification for Smu_ReleaseFSP API

Syntax	Std_ReturnType Smu_ReleaseFSP (void)
---------------	--

SMU driver
Table 256 Specification for Smu_ReleaseFSP API (continued)

Service ID	0xB2	
Sync/Async	Asynchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is setting the PCS bit is successful. E_NOT_OK: Operation not successful.
Description	The purpose of the API is to switch the SMU peripheral from the FAULT state to the RUN state. This also switches the error pin from the FAULT state to FAULT-FREE state. Additionally, this API can be used to change the FSP state from the power-on state to the Fault-free state. This is essential to setup the error pin to drive the FSP. It is also required for testing of FSP pin. The transition of states require clock cycles to reflect. The API returns before this transition is observed.	
Source	IFX	
Error handling	DET: SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. Runtime Errors: None DEM: SMU_E_RELEASE_FSP_FAILURE: SMU_E_RELEASE_FSP_FAILURE DEM is reported when the FSP cannot be released. Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.11 Smu_ActivateFSP
Table 257 Specification for Smu_ActivateFSP API

Syntax	Std_ReturnType Smu_ActivateFSP (void)
Service ID	0xB3

SMU driver
Table 257 **Specification for Smu_ActivateFSP API (continued)**

Sync/Async	Asynchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is activation of FSP is successful E_NOT_OK: Operation not successful.
Description	<p>The purpose of the API is to activate the FSP to indicate a FAULT state on the error pin to the safe state switching device. When FSP is activated the SMU reaches the fault state. This can be confirmed by reading the SMU state in hardware. Also, In the SMU START state, activation of FSP is only possible using this API as alarms are not processed.</p> <p>Additionally, this is required for the testing of the FSP timing.</p>	
Source	IFX	
Error handling	<p>DET:</p> <p>SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state.</p> <p>SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED.</p> <p>Runtime Errors: None</p> <p>DEM:</p> <p>SMU_E_ACTIVATE_FSP_FAILURE: SMU_E_ACTIVATE_FSP_FAILURE DEM is reported when activation of FSP fails.</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>	
Configuration dependencies	-	
User hints	None	

3.3.3.12 Smu_SetupErrorPin

Table 258 **Specification for Smu_SetupErrorPin API**

Syntax	Std_ReturnType Smu_SetupErrorPin (void)
Service ID	0xB4
Sync/Async	Asynchronous
ASIL Level	B

SMU driver
Table 258 Specification for Smu_SetupErrorPin API (continued)

Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is switching of the error pin from GPIO mode to SMU mode is successful. E_NOT_OK: Operation not successful.
Description	The purpose of the API is to enable the SMU to control the error pin. This API switches the error pin from GPIO mode to SMU mode. Only after switching to the SMU mode, SMU can control the error pin.	
Source	IFX	
Error handling	DET: SMU_E_LOCKED: SMU_E_LOCKED DET is reported when the SMU is already in locked state. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.13 Smu_ReleaseErrorPin
Table 259 Specification for Smu_ReleaseErrorPin API

Syntax	Std_ReturnType Smu_ReleaseErrorPin (void)	
Service ID	0xB5	
Sync/Async	Asynchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-

SMU driver
Table 259 **Specification for Smu_ReleaseErrorPin API (continued)**

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is control of error pin is successfully released from SMU E_NOT_OK: Operation not successful.
Description	The purpose of the API is to release the control of the error pin. The transition of modes require certain clock cycles to reflect. The API returns before this transition is observed.	
Source	IFX	
Error handling	DET: SMU_E_LOCKED: SMU_E_LOCKED DET is reported when the SMU is already in locked state. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.14 Smu_RTStop

Table 260 **Specification for Smu_RTStop API**

Syntax	Std_ReturnType Smu_RTStop (const uint8 TimerNum)	
Service ID	0xB6	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	TimerNum	Recovery Timer unit to be stopped.(0,1)
Parameters (out)	-	-
Parameters (in - out)	-	-

SMU driver
Table 260 Specification for Smu_RTStop API (continued)

Return	Std_ReturnType	E_OK: Operation successful that is requested recovery timer is stopped successfully E_NOT_OK: Operation not successful that is requested recovery timer could not be stopped successfully, for example due to invalid parameters
Description	The purpose of the API is to stop the requested recovery timer unit. Possible use case: when a fault occurs, error handler might be triggered. However, this error handler should setup a recovery mechanism or error mitigation mechanism within a finite interval of time to prevent the system from failing.	
Source	IFX	
Error handling	DET: SMU_E_INVALID_TIMER: SMU_E_INVALID_TIMER DET is reported when the timer value passed as a parameter to an API is not valid. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. Runtime Errors: None DEM: SMU_E_RT_STOP_FAILURE: SMU_E_RT_STOP_FAILURE DEM is reported when the Recovery timer cannot be stopped. Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.15 Smu_GetRTMissedEvent
Table 261 Specification for Smu_GetRTMissedEvent API

Syntax	Std_ReturnType Smu_GetRTMissedEvent (const uint8 TimerNum, boolean * const EventMissed)	
Service ID	0xB7	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	TimerNum	Recovery Timer unit for which the status has to be procured (0,1).

SMU driver
Table 261 Specification for Smu_GetRTMissedEvent API (continued)

Parameters (out)	EventMissed	EventMissed : TRUE: Event has been missed FALSE: Event has NOT been missed
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is check for missed events successful. E_NOT_OK: Operation not successful that is check for missed events not successful, for example due to invalid parameters
Description	The purpose of the API is to know if any alarms requiring the requested recovery timer were set while the recovery timer was running.	
Source	IFX	
Error handling	DET: SMU_E_INVALID_TIMER: SMU_E_INVALID_TIMER DET is reported when the timer value passed as a parameter to an API is not valid. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. SMU_E_PARAM_POINTER: SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.16 Smu_ActivatePES
Table 262 Specification for Smu_ActivatePES API

Syntax	Std_ReturnType Smu_ActivatePES (void)	
Service ID	0xB8	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-

SMU driver
Table 262 Specification for Smu_ActivatePES API (continued)

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is activation of the Port Emergency Stop (PES) is successful. E_NOT_OK: Operation not successful.
Description	The purpose of this API is to trigger the activation of the Port Emergency Stop (PES). The PES is also directly controlled by the SMU_core when entering the FAULT state.	
Source	IFX	
Error handling	DET: SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. Runtime Errors: None DEM: SMU_E_ACTIVATE_PES_FAILURE: SMU_E_ACTIVATE_PES_FAILURE DEM is reported when activation of the PES feature fails. Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.17 Smu_RegisterMonitor

Table 263 Specification for Smu_RegisterMonitor API

Syntax	Std_ReturnType Smu_RegisterMonitor (const uint16 * const RegMonPtr, Smu_SffTestResType * const RegMonResult)	
Service ID	0xB9	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	RegMonPtr	Pointer to the array which holds the modules which have SFF test enabled. The total number of elements is the total number of modules which can undergo SFF tests. The elements have to be as per the bits specified in RMCTL register. Additionally, the module needs to be present in the specific device.

SMU driver
Table 263 **Specification for Smu_RegisterMonitor API (continued)**

Parameters (out)	RegMonResult	Pointer to array for the SFF test results for the modules. In case a module test was not enabled but has an error recorded, it will indicate that failure as well.
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is SFF tests were successfully completed E_NOT_OK: Operation not successful that SFF tests were successfully not completed
Description	The purpose of the API is to provide the initialization, execution and termination of the Safety FlipFlop tests to be executed for different modules as per the configuration. The prerequisites like clock settings need to be taken care of additionally by the user before running the tests	
Source	IFX	
Error handling	DET: SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_LOCKED: SMU_E_LOCKED DET is reported when the SMU is already in locked state. SMU_E_PARAM_POINTER: SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer. Runtime Errors: None DEM: SMU_E_SFF_TEST_FAILURE: SMU_E_SFF_TEST_FAILURE DEM is reported when an SFF test fails. Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.18 Smu_GetSmuState

Table 264 **Specification for Smu_GetSmuState API**

Syntax	Smu_CoreStateType Smu_GetSmuState (void)
Service ID	0xBA
Sync/Async	Synchronous
ASIL Level	B
Re-entrancy	Reentrant

SMU driver
Table 264 **Specification for Smu_GetSmuState API (continued)**

Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Smu_CoreStateType	State of SMU core state machine
Description	The purpose of the API is to provide the current state of the SMU core. This is referred to as the safety status of the system as all critical faults will cause the SMU to go to the FAIL state.	
Source	IFX	
Error handling	DET: SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_NO_ERR: SMU_E_NO_ERR DET is the default value which indicates the absence of any error. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>	
Configuration dependencies	-	
User hints	None	

3.3.3.19 Smu_ActivateRunState

Table 265 **Specification for Smu_ActivateRunState API**

Syntax	Std_ReturnType Smu_ActivateRunState (const uint32 Cmd)	
Service ID	0xBB	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	Cmd	Command to switch the SMU to the RUN state
Parameters (out)	-	-
Parameters (in - out)	-	-

SMU driver
Table 265 **Specification for Smu_ActivateRunState API (continued)**

Return	Std_ReturnType	<p>E_OK: Operation successful that is activation of fault free RUN state is successful.</p> <p>E_NOT_OK: Operation not successful that is activation of fault free RUN state is not successful, for example SMU is not initially in START state.</p>
Description	<p>The purpose of the API is to allow switching the SMU peripheral into the RUN fault-free state as requested by the caller. The SMU validates the request based on integrity checks of SMU (that is check of the command value).</p>	
Source	IFX	
Error handling	<p>DET:</p> <p>SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state.</p> <p>SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED.</p> <p>Runtime Errors: None</p> <p>DEM:</p> <p>SMU_E_ACTIVATE_RUN_STATE_FAILURE: SMU_E_ACTIVATE_RUN_STATE_FAILURE DEM is reported when the activation of RUN state fails.</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>	
Configuration dependencies	-	
User hints	-	

3.3.3.20 Smu_GetVersionInfo

Table 266 **Specification for Smu_GetVersionInfo API**

Syntax	<pre>void Smu_GetVersionInfo (Std_VersionInfoType * const VersionInfoPtr)</pre>	
Service ID	0xBC	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	VersionInfoPtr	Pointer to store information about the module
Parameters (in - out)	-	-
Return	void	-

SMU driver
Table 266 **Specification for Smu_GetVersionInfo API (continued)**

Description	The purpose of the API is to return the version information of the SMU driver. The version information includes: Module ID, Vendor ID., vendor specific version numbers. This function is available only if the SMU_VERSION_INFO_API is set ON.
Source	IFX
Error handling	DET: SMU_E_PARAM_POINTER: SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	SmuVersionInfoApi
User hints	If DET is enabled and if parameter versioninfo is NULL_PTR, SMU_E_PARAM_POINTER DET is raised. Version info is read correctly, independent from the DET generation. Update Version Information in the location referenced by versioninfo

3.3.3.21 Smu_CoreAliveTest

Table 267 **Specification for Smu_CoreAliveTest API**

Syntax	Std_ReturnType Smu_CoreAliveTest (void)	
Service ID	0xBD	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: The core alive test returned success. E_NOT_OK: The core alive test returned failure
Description	The purpose of the API is to provide the means to execute the SMU_AliveTest command which checks the smu_core_alive signal. The SMU_stdby has to remain enabled to execute this command.	
Source	IFX	

SMU driver
Table 267 Specification for Smu_CoreAliveTest API (continued)

Error handling	<p>DET:</p> <p>SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED.</p> <p>SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state.</p> <p>SMU_E_STDBY_DISABLED: SMU_E_STDBY_DISABLED DET is reported when any alarm action is configured or performed with respect to the standby domain of SMU without enabling the SMU standby mode.</p> <p>Runtime Errors: None</p> <p>DEM:</p> <p>SMU_E_CORE_ALIVE_FAILURE: SMU_E_CORE_ALIVE_FAILURE DEM is reported when the core alive test of SMU core fails.</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>
Configuration dependencies	-
User hints	None

3.3.3.22 Smu_InitCheck

Table 268 Specification for Smu_InitCheck API

Syntax	Std_ReturnType Smu_InitCheck (const Smu_ConfigType * const ConfigPtr)	
Service ID	0xA9	
Sync/Async	Synchronous	
ASIL Level	B(D)	
Re-entrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to the SMU configuration for initialization
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK : Initialization comparison successful E_NOT_OK : Initialization comparison failed
Description	The purpose of the API is to check the initialization values after SMU is initialized. The API should be called after the SMU driver is initialized to check the initialization values.	
Source	IFX	

SMU driver
Table 268 **Specification for Smu_InitCheck API (continued)**

Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	SmuInitCheckApi
User hints	None

3.3.3.23 Smu_GetAlarmExecutionStatus
Table 269 **Specification for Smu_GetAlarmExecutionStatus API**

Syntax	Std_ReturnType Smu_GetAlarmExecutionStatus (const uint32 AlarmExecStatusReq, uint32 * const AlarmExecStatus)	
Service ID	0xBE	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	AlarmExecStatusReq	Alarm reaction for which execution status is requested.
Parameters (out)	AlarmExecStatus	Pointer which stores the alarm execution status result.
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful that is retrieval of the status of the alarm reactions executed is successful. E_NOT_OK: Operation not successful that is retrieval of the status of the alarm reactions executed is not successful, for example due to invalid parameters
Description	This API gives the status of the alarm reactions executed.	
Source	IFX	

SMU driver
Table 269 Specification for Smu_GetAlarmExecutionStatus API (continued)

Error handling	<p>DET:</p> <p>SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state.</p> <p>SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED.</p> <p>SMU_E_INVALID_EXECUTION_STATUS: SMU_E_INVALID_EXECUTION_STATUS DET is reported when an invalid error mechanism is requested for alarm execution status.</p> <p>SMU_E_PARAM_POINTER: SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer.</p> <p>Runtime Errors: None</p> <p>DEM: None</p> <p>Safety Errors: None</p> <p><i>Note: All DET IDs are also reported as safety errors.</i></p>
Configuration dependencies	-
User hints	None

3.3.3.24 Smu_ClearAlarmExecutionStatus
Table 270 Specification for Smu_ClearAlarmExecutionStatus API

Syntax	Std_ReturnType Smu_ClearAlarmExecutionStatus (const uint32 AlarmExecStatusReq)	
Service ID	0xBF	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	AlarmExecStatusReq	Alarm reactions for which the alarm execution status clear has been requested
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	<p>E_OK: Operation successful that is the status of the alarm reactions executed is cleared successfully</p> <p>E_NOT_OK: Operation not successful that is the status of the alarm reactions executed is not cleared successfully, for example due to invalid parameters</p>
Description	The purpose of the API is to clear the alarm reaction execution status for the requested alarm reaction.	
Source	IFX	

SMU driver
Table 270 **Specification for Smu_ClearAlarmExecutionStatus API (continued)**

Error handling	DET: SMU_E_UNINIT: SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state. SMU_E_INVALID_DRIVER_STATE: SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED. SMU_E_INVALID_EXECUTION_STATUS: SMU_E_INVALID_EXECUTION_STATUS DET is reported when an invalid error mechanism is requested for alarm execution status. Runtime Errors: None DEM: None Safety Errors: None <i>Note: All DET IDs are also reported as safety errors.</i>
Configuration dependencies	-
User hints	None

3.3.4 Notifications and callbacks

The SMU driver does not have any notifications or callbacks.

3.3.5 Scheduled functions

The SMU driver does not have any scheduled functions.

3.3.6 Interrupt service routines

Interrupt handlers are not applicable for the SMU driver.

3.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

3.3.7.1 Development errors

The following table lists all the development errors reported by the driver.

Note: *The following error IDs are also reported as safety errors.*

Table 271 **Description of development errors reported**

Description	Source	Error code and value	Applicable APIs
SMU_E_NO_ERR DET is the default value which indicates the absence of any error.	IFX	SMU_E_NO_ERR=0x00	Smu_GetSmuState

SMU driver
Table 271 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
SMU_E_UNINIT DET is reported when any API is called while the driver is not in initialized state.	IFX	SMU_E_UNINIT=0x01	Smu_ClearAlarmExecutionStatus, Smu_GetAlarmDebugStatus, Smu_GetAlarmExecutionStatus, Smu_GetRTMissedEvent, Smu_RTStop, Smu_SetAlarmStatus, Smu_GetAlarmStatus, Smu_ClearAlarmStatus, Smu_CoreAliveTest, Smu_ActivateRunState, Smu_GetSmuState, Smu_RegisterMonitor, Smu_ActivatePES, Smu_ReleaseErrorPin, Smu_SetupErrorPin, Smu_ActivateFSP, Smu_ReleaseFSP, Smu_LockConfigRegs, Smu_GetAlarmAction, Smu_SetAlarmAction, Smu_DelInit
SMU_E_ALREADY_INITIALIZED DET is reported when SMU is already initialized.	IFX	SMU_E_ALREADY_INITIALIZED=0x02	Smu_Init
SMU_E_INIT_FAILED DET is reported when initialization of SMU driver fails due to incorrect configuration parameter.	IFX	SMU_E_INIT_FAILED=0x03	Smu_Init
SMU_E_PARAM_POINTER DET is reported when the pointer passed as a parameter to an API is a NULL pointer.	IFX	SMU_E_PARAM_POINTER=0x04	Smu_GetRTMissedEvent, Smu_GetVersionInfo, Smu_RegisterMonitor, Smu_GetAlarmExecutionStatus, Smu_GetAlarmDebugStatus, Smu_GetAlarmStatus, Smu_GetAlarmAction
SMU_E_PARAM_GROUP DET is reported when the group ID or the alarm position passed as a parameter to an API is not valid.	IFX	SMU_E_PARAM_GROUP=0x05	Smu_SetAlarmStatus, Smu_ClearAlarmStatus, Smu_SetAlarmAction, Smu_GetAlarmDebugStatus, Smu_GetAlarmStatus, Smu_GetAlarmAction

SMU driver
Table 271 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
SMU_E_INVALID_DRIVER_STATE DET is reported when the SMU driver state is SMU_FAILED.	IFX	SMU_E_INVALID_DRIVER_STATE=0x06	Smu_ClearAlarmExecutionStatus, Smu_GetAlarmExecutionStatus, Smu_GetRTMissedEvent, Smu_RTStop, Smu_ClearAlarmStatus, Smu_CoreAliveTest, Smu_ActivateRunState, Smu_RegisterMonitor, Smu_ActivatePES, Smu_ReleaseErrorPin, Smu_ActivateFSP, Smu_ReleaseFSP, Smu_SetupErrorPin, Smu_LockConfigRegs, Smu_SetAlarmAction, Smu_SetAlarmStatus
SMU_E_INVALID_TIMER DET is reported when the timer value passed as a parameter to an API is not valid.	IFX	SMU_E_INVALID_TIMER=0x07	Smu_RTStop, Smu_GetRTMissedEvent
SMU_E_STDBY_DISABLED DET is reported when any alarm action is configured or performed with respect to the standby domain of SMU without enabling the SMU standby mode.	IFX	SMU_E_STDBY_DISABLED=0x08	Smu_GetAlarmStatus, Smu_SetAlarmAction, Smu_CoreAliveTest, Smu_GetAlarmAction, Smu_ClearAlarmStatus
SMU_E_LOCKED DET is reported when the SMU is already in locked state.	IFX	SMU_E_LOCKED=0x09	Smu_DeInit, Smu_Init, Smu_SetAlarmAction, Smu_RegisterMonitor, Smu_ReleaseErrorPin, Smu_SetupErrorPin, Smu_LockConfigRegs
SMU_E_INVALID_ALARM_ACTION DET is reported when the alarm action to be configured is not valid.	IFX	SMU_E_INVALID_ALARM_ACTION=0x0A	Smu_SetAlarmAction
SMU_E_INVALID_EXECUTION_STATUS DET is reported when an invalid error mechanism is requested for alarm execution status.	IFX	SMU_E_INVALID_EXECUTION_STATUS=0x0B	Smu_ClearAlarmExecutionStatus, Smu_GetAlarmExecutionStatus

SMU driver
Table 271 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
SMU_E_CORE_MISMATCH DET is reported when the Init or De-Init is called from any core other than the master core.	IFX	SMU_E_CORE_MISMATCH=0x68	Smu_DeInit, Smu_Init

3.3.7.2 Production errors

The table below lists all the production errors reported by the driver.

Table 272 Description of production errors reported

Description	Source	Error code and value	Applicable APIs
SMU_E_ACTIVATE_FSP_FAIL LURE DEM is reported when activation of FSP fails.	IFX	SMU_E_ACTIVATE_FSP_FAILURE=Assigned by DEM	Smu_ActivateFSP
SMU_E_ACTIVATE_PES_FAIL LURE DEM is reported when activation of the PES feature fails.	IFX	SMU_E_ACTIVATE_PES_FAILURE=Assigned by DEM	Smu_ActivatePES
SMU_E_ACTIVATE_RUN_STATE_FAIL LURE DEM is reported when the activation of RUN state fails.	IFX	SMU_E_ACTIVATE_RUN_STATE_FAILURE=Assigned by DEM	Smu_ActivateRunState
SMU_E_CLEAR_ALARM_STATUS_FAIL LURE DEM is reported when the clearing of alarm status fails.	IFX	SMU_E_CLEAR_ALARM_STATUS_FAILURE=Assigned by DEM	Smu_ClearAlarmStatus
SMU_E_RELEASE_FSP_FAIL LURE DEM is reported when the FSP cannot be released.	IFX	SMU_E_RELEASE_FSP_FAILURE=Assigned by DEM	Smu_ReleaseFSP
SMU_E_RT_STOP_FAILURE DEM is reported when the Recovery timer cannot be stopped.	IFX	SMU_E_RT_STOP_FAILURE=Assigned by DEM	Smu_RTStop
SMU_E_SET_ALARM_STATUS_FAIL LURE DEM is reported when the setting of alarm status fails.	IFX	SMU_E_SET_ALARM_STATUS_FAILURE=Assigned by DEM	Smu_SetAlarmStatus
SMU_E_SFF_TEST_FAILURE DEM is reported when an SFF test fails.	IFX	SMU_E_SFF_TEST_FAILURE=Assigned by DEM	Smu_RegisterMonitor

SMU driver
Table 272 Description of production errors reported (continued)

Description	Source	Error code and value	Applicable APIs
SMU_E_CORE_ALIVE_FAILURE DEM is reported when the core alive test of SMU core fails.	IFX	SMU_E_CORE_ALIVE_FAILURE=Assigned by DEM	Smu_CoreAliveTest

3.3.7.3 Safety errors

The table below lists all the safety errors reported by the driver.

Table 273 Description of safety errors reported

Description	Source	Error code and value	Applicable APIs
The safety error is reported when the configuration registers do not get locked after applying permanent lock.	IFX	SMU_E_SF_CFG_LOCKED=0xC8	Smu_LockConfigRegs

3.3.7.4 Runtime errors

The module does not report any runtime errors.

3.3.8 Deviations and limitations

3.3.8.1 Deviations

There are no deviations in the SMU driver.

3.3.8.2 Limitations

There are no limitations for the SMU driver.

3.3.9 Unsupported hardware features

The following hardware features of the SMU driver are not currently supported:

- MONBISTCTRL feature for Smu_stdby

Revision history**Revision history**

Major changes since the last revision

Date	Version	Description
2018-10-05	4.0	Updated for RC release.
2018-05-30	3.0	Updated for Beta release.
2018-02-19	2.0	Updated the version information of the TC3xx_SW_MCAL_HWErtaAnalysis.xlsx file.
2017-12-14	1.0	Initial version.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2018-10-05

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2018 Infineon Technologies AG
All Rights Reserved.

Do you have a question about any
aspect of this document?
Email: erratum@infineon.com

Document reference
IFX-mtk1519892305851

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffungsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury