Document Title	Specification of Extended Fixed Point Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	400
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R20-11

Document Change History

Date	Release	Changed by	Change Description
2020-11-30	R20-11	AUTOSAR Release Management	**Added :** • New function added Efx_Pt1Typ1Calc with the requirements SWS_Efx_00531, SWS_Efx_00532, SWS_Efx_00533, SWS_Efx_00534, SWS_Efx_00535 • SWS_Efx_00823, SWS_Efx_00824 SWS_Efx_00825 - Requirements related to saturation has been added for the functions Efx_Add,Efx_Mul,Efx_Div **Modified :** • Updated Chapter 7.1 Error classification **Removed :** • Removal of Efx_Cast and Efx_Gt functions
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	**Modified:** • Updated the range and resolution of requirements SWS_EFX_00220,SWS_EFX_00223,SWS_EFX_00226,SWS_EFX_00229,SWS_EFX_00232,SWS_EFX_00235,SWS_EFX_00240,SWS_EFX_00243,SWS_EFX_00246,SWS_EFX_00250,SWS_EFX_00253,SWS_EFX_00256

Document Change History

Date	Release	Changed by	Change Description
2017-12-08	4.3.1	AUTOSAR Release Management	**Added:** • A note has been added in Section 8.1 of EFX specification to provide clarity in usage of mnemonic for Boolean data types. **Modified:** • The data type for Boolean has been updated in the UML of SWS_Efx_00355. • Inclusion of Pointer to Constant (P2CONST) for SWS_Efx_00355, SWS_Efx_00309, SWS_Efx_00307 & SWS_Efx_00193 and proper categorization of Parameters as InOut for SWS_Efx_00376.
2016-11-30	4.3.0	AUTOSAR Release Management	**Modified:** • Updated the correct reference to SRS_BSW_General (SRS_BSW_00437) & (SRS_BSW_00448) for SWS_Efx_00810 & SWS_Efx_00822 requirements. • Updated EFX document to support MISRA 2012 standard. (Removed redundant statements in SWS_Efx_00809 which already exist in SWS_BSW document and SWS_SRS document) • Updated SWS_Efx_00275 & SWS_Efx_00276 to provide more clarity on resolution of parameters. • Updated SWS_Efx_00278 & SWS_Efx_00279 to provide more clarity on rounding and minimum value of Param_cpcst->SlopeXXX_u32 * dT_s32. Provided the correct IT number. • Updated the section 8.5.3.1 for Structure definitions for controller routines.

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
			• Updated SWS_Efx_00240, SWS_Efx_00243, SWS_Efx_00246, SWS_Efx_00250, SWS_Efx_00253 & SWS_Efx_00256 to correct the case sensitivity for the function name. • Section 2 has been revisited to update Default Error Tracer instead of Development Error tracer. **Removed:** • Removal of Efx_ISetParam from BSW uml model which is obsolete. • Removed the duplicated trace environments for SWS_Efx_00520 & SWS_Efx_00525. • Removed the requirements that are marked as Deprecated. (8.5.1.2 Second computation, SWS_Efx_00009 - SWS_Efx_00011, SWS_Efx_00041 - SWS_Efx_00043, SWS_Efx_00295 - SWS_Efx_00302, SWS_Efx_00347 - SWS_Efx_00354, SWS_Efx_00345, SWS_Efx_00460, SWS_Efx_00461 & 8.5.14 Efx_DeadTime)
2015-07-31	4.2.2	AUTOSAR Release Management	**Modified:** • Updated the requirement ID for SWS_Efx_00033 as per the convention • Updated requirement ID SWS_Efx_00436 (UML) for OutTypeMn as per the standard convention • Updated SWS_Efx_00001 for naming convention under Section 5.1, File Structure • Updated SWS_Efx_00365 to correct the data type of input parameters

Document Change History

Date	Release	Changed by	Change Description
2014-10-31	4.2.1	AUTOSAR Release Management	**Added:** • New Variants for SWS_Efx_00412 (0xE2 - 0xE9) • Note has been added for SWS_Efx_00053, SWS_Efx_00072 & Section 8.5.3.1. • A statement has been added to clarify the formula used for Hypotenuse function just below the section 8.5.9 • A statement has been added to provide more clarity on the formula mentioned in SWS_Efx_00451 **Modified:** • Updated usage of const in a consistent manner in EFX document. (SWS_Efx_00050, SWS_Efx_00067, SWS_Efx_00085, SWS_Efx_00519, SWS_Efx_00107, SWS_Efx_00122, SWS_Efx_00146, SWS_Efx_00172, SWS_Efx_00205, SWS_Efx_00379 & SWS_Efx_00404) • Formula for TeQ_<size> has been corrected in section 8.5.3.1 and font has been updated for SWS_Efx_00071 • Condition check included for SWS_Efx_00053, SWS_Efx_00072 & Section 8.5.3.1 and corrected for SWS_Efx_00054, SWS_Efx_00073 & SWS_Efx_00504. Formula updated for SWS_Efx_00073 • Updated rounding for SWS_Efx_00071, SWS_Efx_00091, SWS_Efx_00502, SWS_Efx_00151 & SWS_Efx_00156.

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
			• Service ID[hex] for SWS_Efx_00405, SWS_Efx_00410 & SWS_Efx_00412 • Input & Output range has been modified for SWS_Efx_00187 • Statement on rounding was updated for SWS_Efx_00441 • Comment for structure element “n” has been updated for SWS_Efx_00204 & • SWS_Efx_00836. Data type of “n” has been modified for SWS_Efx_00204.
2014-03-31	4.1.3	AUTOSAR Release Management	• Modified: Rounding mechanism was updated for HpFilter, Average, Array_Average & MovingAverage functions. • Added: A note below SWS_Efx_00307 for Efx_RampGetSwitchPos function.
2013-10-31	4.1.2	AUTOSAR Release Management	• Deprecated: Efx_DeadTime function • Removed: Requirements for Efx_SlewRate, Efx_RampCalc and Efx_RampCalcJump functions • Added: SWS_Efx_00837 for Efx_RampCalc function • Modified: • Descriptions of Efx_RampCalc and Efx_RampSetParam • Requirements for Efx_RampCalc and Efx_RampCalcJump functions. • Syntax for variants of Efx_SlewRate, Efx_Div and Efx_MovingAverage functions. • Resolution of the in-parameter for Efx_Arcsin and Efx_Arccos functions. • Name "underflow" to "negative overflow" throughout the document • Editorial changes

Document Change History

Date	Release	Changed by	Change Description
2013-03-15	4.1.1	AUTOSAR Administration	• Added 8-bit and 16-bit variants for Hysteresis functions • Formulae modified for Hypotenuse functions • Second computation First-order low-pass filter functions are deprecated • Inequalities are corrected for Efx_HystLeftRight, Efx_HystDeltaRight, Efx_HystCenterHalfDelta functions • Description and requirements are modified for Efx_Div, Efx_Debounce, Efx_HystLeftDelta, Efx_SortAscend, Efx_SortDescend, Efx_EdgeBipol, Efx_Hysteresis, Efx_MovingAverage functions • Description of the in-parameter corrected for Efx_DebounceSetParam, Efx_Debounce functions • Physical range of ‘fac’ parameter is modified in LpFilter First computation • Renamed RS_FlipFlop function for removing the post-fixes • Added SWS_Efx_00823 for Integral promotion • Modified syntax for Efx_Gt, Efx_Debounce functions • Corrected for ‘DependencyOnArtifact’

Document Change History

Date	Release	Changed by	Change Description
2011-12-22	4.0.3	AUTOSAR Administration	• Initialization functionality introduced for 'Counter Routines' • Interface for Efx_CtrlSetLimit corrected • Efx_MovingAverage routine interface corrected • Efx_RampCalcSwitch routine definition and requirements updated for correct behavior • Interface for Efx_Debounce_u8_u8 routine updated • Updated parameter sequences for DT1 and PI controller routines. • Name revised for Efx_PCalc routine • Description correct for Efx_DebounceParam_Type and Efx_DebounceState_Type • Interface table corrected for Efx_Div routine • Interface table corrected for Efx_MedianSort routine • Error classification support and definition removed as DET call not supported by library • Configuration parameter description / support removed for XXX_GetVersionInfo routine. • XXX_GetVersionInfo routine name corrected.
2010-09-30	3.1.5	AUTOSAR Administration	• Introduction of additional LIMITED Functions for controllers • Ramp functions optimised for effective usage • Separation of DT1 Type 1 and Type 2 Controller functions • Introduction of additional approximative function for calculation of TeQ

Document Change History

Date	Release	Changed by	Change Description
2010-02-02	3.1.4	AUTOSAR Administration	• Initial Release

Disclaimer

Table of Contents

1 Introduction and functional overview

AUTOSAR Library routines are the part of system services in AUTOSAR architecture and below figure shows position of AUTOSAR library in layered architecture.

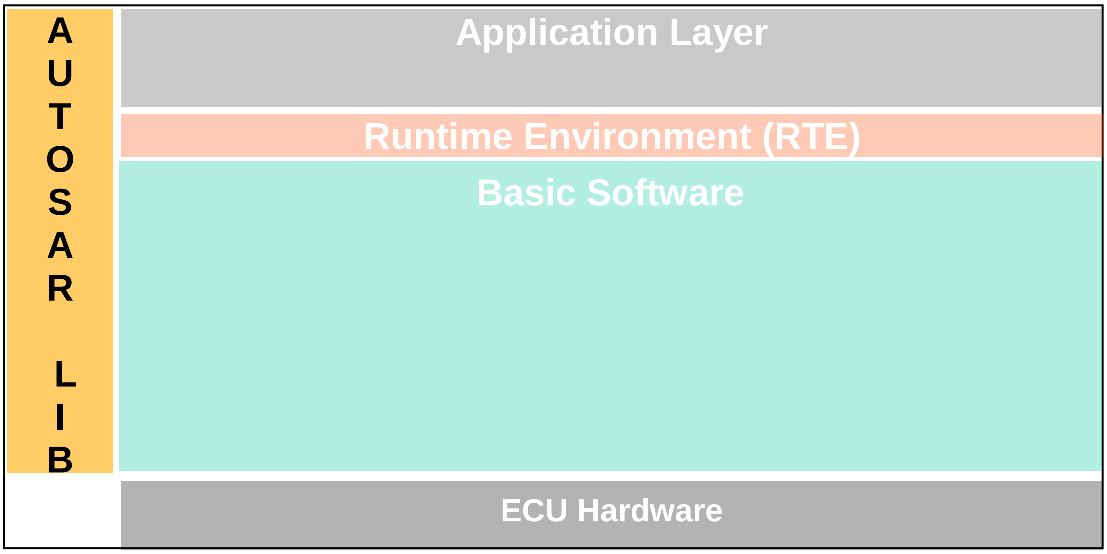

Figure : Layered architecture

This specification specifies the functionality, API and the configuration of the AUTOSAR library dedicated to extended mathematical functions for fixed-point values.

This extended mathematical library (Efx) contains the following routines:

- Moving average
- First order high pass filter
- First order low-pass filter
- Controller routines
- Square root
- Exponential
- Average
- Array Average
- Moving Average
- Hypotenuse
- Trigonometric functions
- Rate limiter functions
- Ramp routines
- Hysteresis function
- Dead Time
- Debounce
- Ascending Sort Routine
- Descending Sort Routine
- Median Sort
- Edge detection routines
- Interval routines
- Counter routines

- Flip-Flop routine
- Limiter routines
- 64 bit functions

All routines are re-entrant and can be used by multiple runnables at the same time.

2 Acronyms and abbreviations

Acronyms and abbreviations, which have a local scope and therefore are not contained in the AUTOSAR glossary, must appear in a local glossary.

Abbreviation / Acronym:	***Description:***
Arcsin	Inverse Sine
Arccos	Inverse Cosine
BSW	Basic Software
Cos	Cosine
DET	Default Error Tracer
EFX	Extended Mathematical library – Fixed point
Hypot	Hypotenuse
HpFilter	High pass filter
LpFilterFac1	Low pass filter with a factor of 1 (included in [0, 1])
LpFilter	Low pass filter
Mn	Mnemonic
Lib	Library
Sqrt	Square root
Sin	Sine
SWS	Software Specification
SRS	Software Requirement Specification
u8	Mnemonic for the uint8, specified in AUTOSAR_SWS_PlatformTypes
u16	Mnemonic for the uint16, specified in AUTOSAR_SWS_PlatformTypes
u32	Mnemonic for the uint32, specified in AUTOSAR_SWS_PlatformTypes
s8	Mnemonic for the sint8, specified in AUTOSAR_SWS_PlatformTypes
s16	Mnemonic for the sint16, specified in AUTOSAR_SWS_PlatformTypes
s32	Mnemonic for the sint32, specified in AUTOSAR_SWS_PlatformTypes
s64	Mnemonic for the sint64, specified in AUTOSAR_SWS_PlatformTypes
u64	Mnemonic for the uint64, specified in AUTOSAR_SWS_PlatformTypes

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules,
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[5] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[6] Specification of Platform Types,
AUTOSAR_SWS_PlatformTypes.pdf

[7] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[8] Requirement on Libraries,
AUTOSAR_SRS_Libraries.pdf

[9] Specification of Memory Mapping,
AUTOSAR_SWS_MemoryMapping.pdf

3.2 Related standards and norms

[10] ISO/IEC 9899:1990 Programming Language – C

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 File structure

Implementation & grouping of routines with respect to C files is recommended as per below options and there is no restriction to follow the same.

Option 1 : <Name> can be function name providing one C file per function,
eg.: Efx_Pt1_s32.c etc.

Option 2 : <Name> can have common name of group of functions:
2.1 Group by object family:
eg.:Efx_Pt1.c, Efx_Dt1.c, Efx_Pid.c
2.2 Group by routine family:
eg.: Efx_Filter.c, Efx_Controller.c, Efx_Average.c etc.
2.3 Group by method family:
eg.: Efx_Sin.c, Efx_Exp.c, Efx_Arcsin.c, etc.
2.4 Group by architecture:
eg.: Efx_Slewrate16.c, Efx_Slewrate32.c
2.5 Group by other methods: (individual grouping allowed)

Option 3 : <Name> can be removed so that single C file shall contain all Efx functions, eg.: Efx.c.
Using above options gives certain flexibility of choosing suitable granularity with reduced number of C files. Linking only on-demand is also possible in case of some options.

6 Requirements traceability

Requirement	Description	Satisfied by
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_Efx_00815
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	SWS_Efx_00809
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	SWS_Efx_00812
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Efx_00813
SRS_BSW_00318	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	SWS_Efx_00815
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_Efx_00815
SRS_BSW_00374	All Basic Software Modules shall provide a readable module vendor identification	SWS_Efx_00814
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_Efx_00812
SRS_BSW_00379	All software modules shall provide a module identifier in the header file and in the module XML description file.	SWS_Efx_00814
SRS_BSW_00402	Each module shall provide version information	SWS_Efx_00814
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Efx_00815, SWS_Efx_00816
SRS_BSW_00411	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	SWS_Efx_00816
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_Efx_00810
SRS_BSW_00448	Module SWS shall not contain requirements from Other Modules	SWS_Efx_00822
SRS_LIBS_00001	The functional behavior of each library functions shall not be configurable	SWS_Efx_00818
SRS_LIBS_00002	A library shall be operational before all BSW modules and application SW-Cs	SWS_Efx_00800
SRS_LIBS_00003	A library shall be operational until the shutdown	SWS_Efx_00801
SRS_LIBS_00015	It shall be possible to configure the microcontroller so that the library code is shared between all callers	SWS_Efx_00806
SRS_LIBS_00017	Usage of macros should be avoided	SWS_Efx_00807
SRS_LIBS_00018	A library function may only call library functions	SWS_Efx_00808

7 Functional specification

7.1 Error classification

[SWS_Efx_00821] ⌈Section 7.1 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.⌋()

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.1.1 Development Errors

There are no development errors.

7.1.2 Runtime Errors

There are no runtime errors

7.1.3 Transient Faults

There are no transient faults.

7.1.4 Production Error

There are no production errors

7.1.5 Extended Production Errors

There are no extended production errors

7.2 Initialization and shutdown

[SWS_Efx_00800] ⌈Efx library shall not require initialization phase. A Library function may be called at the very first step of ECU initialization, e.g. even by the OS or EcuM, thus the library shall be ready. ⌋ (SRS_LIBS_00002)

[SWS_Efx_00801] ⌈Efx library shall not require a shutdown operation phase. ⌋ (SRS_LIBS_00003)

7.3 Using Library API

Efx API can be directly called from BSW modules or SWC. No port definition is required. It is a pure function call.

The statement ‘Efx.h’ shall be placed by the developer or an application code generator but not by the RTE generator

Using a library should be documented. if a BSW module or a SWC uses a Library, the developer should add an Implementation-DependencyOnArtifact in the BSW/SWC template.
minVersion and maxVersion parameters correspond to the supplier version. In case of AUTOSAR library, these parameters may be left empty because a SWC or BSW module may rely on a library behaviour, not on a supplier implementation. However,

the SWC or BSW modules shall be compatible with the AUTOSAR platform where they are integrated.

7.4 library implementation

[SWS_Efx_00806] ⌈The Efx library shall be implemented in a way that the code can be shared among callers in different memory partitions. ⌋(SRS_LIBS_00015)

[SWS_Efx_00807] ⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used. ⌋(SRS_LIBS_00017)

[SWS_Efx_00808] ⌈A library function shall not call any BSW modules functions, e.g. the DET. A library function can call other library functions. Because a library function shall be re-entrant. But other BSW modules functions may not be re-entrant. ⌋(SRS_LIBS_00018)

[SWS_Efx_00809] ⌈The library, written in C programming language, should conform to the MISRA C Standard.
Please refer to SWS_BSW_00115 for more details.
⌋(SRS_BSW_00007)

[SWS_Efx_00810] ⌈Each AUTOSAR library Module implementation <library>*.c and <library>*.h shall map their code to memory sections using the AUTOSAR memory mapping mechanism. ⌋(SRS_BSW_00437)

[SWS_Efx_00812] ⌈All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of native C data types, unless this library is clearly identified to be compliant only with a platform. ⌋(SRS_BSW_00304, SRS_BSW_00378)

[SWS_Efx_00813] ⌈All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, unless this library is clearly identified to be compliant only with a platform. eg. #pragma, typeof etc. ⌋(SRS_BSW_00306)

[SWS_Efx_00823] ⌈Integral promotion has to be adhered to when implementing Efx services. Thus, to obtain maximal precision, intermediate results shall not be limited. ⌋()

8 API specification

8.1 Imported types

In this chapter, all types included from the following modules are listed:

Header file	Imported Type
Std_Types.h	boolean, sint8, uint8, sint16, uint16, sint32, uint32

It is observed that since the sizes of the integer types provided by the C language are implementation-defined, the range of values that may be represented within each of the integer types will vary between implementations.

The following mnemonic are used in the library routine names.

Size	Platform Type	Mnemonic	Range
unsigned 8-Bit	boolean	u8	[TRUE, FALSE]
signed 8-Bit	sint8	s8	[-128, 127]
signed 16-Bit	sint16	s16	[-32768, 32767]
signed 32-Bit	sint32	s32	[-2147483648, 2147483647]
signed 64-Bit	sint64	s64	[-9223372036854775808, 9223372036854775807]
unsigned 8-Bit	uint8	u8	[0, 255]
unsigned 16-Bit	uint16	u16	[0, 65535]
unsigned 32-Bit	uint32	u32	[0, 4294967295]
unsigned 64-Bit	uint64	u64	[0, 18446744073709551615]

Table 1: Base Types

As a convention in the rest of the document:

- mnemonics will be used in the name of the routines (using <InTypeMn1> that means Type Mnemonic for Input 1)
- the real type will be used in the description of the prototypes of the routines (using <InTypeMn1> or <OutType>).

Note:

The naming convention for the api's with boolean return type/parameter type is given as _u8 which shall be interpreted as _b. (Boolean)
If there is no boolean data type present in the return type/parameter type then _u8 shall be interpreted as _u8 only.

8.2 Type definitions

None

8.3 Comment about rounding

Two types of rounding can be applied:

Results are 'rounded off', it means:

- 0 <= X < 0.5 rounded to 0
- 0.5 <= X < 1 rounded to 1
- -0.5 < X <= 0 rounded to 0
- -1 < X <= -0.5 rounded to -1

Results are rounded towards zero.

- 0 <= X < 1 rounded to 0
- -1 < X <= 0 rounded to 0

8.4 Comment about routines optimized for target

The routines described in this library may be realized as regular routines or inline functions. For ROM optimization purposes, it is recommended that the c routines be realized as individual source files so they may be linked in on an as-needed basis.

For example, depending on the target, two types of optimization can be done:

- Some routines can be replaced by another routine using integer promotion

- Some routines can be replaced by the combination of a limiting routine and a routine with a different signature.

8.5 Mathematical functions definitions

This table describes the meaning of used symbols in below sections.

Symbols	Description
Yn	Actual output to calculate
Yn-1	Output value, one time step before
Xn	Actual input, given from the input
Xn-1	Input, one time step before
a, b0, b1	Filter dependent constants

8.5.1 First-order low-pass filter

We consider a recursive first-order low-pass filter with a transfer function :

$$H(z) = \frac{b_1}{1 + a * z^{-1}}$$

The new return value (Yn) at any point of time can be calculated given the previous value (Yn-1), the current value (Xn) and a known constant (K). The formula to calculate the same is as follows:

Yn = Yn-1 + (Xn - Yn-1) * K
Where b_1=K and a = K - 1

The filter is a convergent low-pass filter only if the average value K is included in [0,1]

8.5.1.1 First computation

[SWS_Efx_00005]⌈

Service Name	Efx_LpFilterFac1_<InTypeMn><InTypeMn><InTypeMn>_<OutTypeMn>

Syntax	`<OutType> Efx_LpFilterFac1_<InTypeMn><InTypeMn><InTypeMn>_<OutTypeMn> (`
	`  <InType> Yn-1,`
	`  <InType> Xn,`
	`  <InType> fac`
	`)`

Service ID [hex]	0x01 to 0x08
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	Yn-1	Old output value
	Xn	Current measured value
	fac	Factor value that represents the physical range [-1, 1) if signed and [0, 1) if unsigned. Only physical value [0 , 1] shall be used if the filter shall converge.
Parameters (inout)	None	
Parameters (out)	None	
Return value	<Out Type>	Result (Yn) of the calculation
Description	This service computes the output of a first order low-pass filter	
Available via	Efx.h	

⌋()

[SWS_Efx_00006]⌈
Yn = Yn-1 + (((Xn - Yn-1) * fac) >> n)
Where 'n' is a shift that depends on the types used by the functions for the factor
⌋()

[SWS_Efx_00007]⌈
In order to converge all the time, the result is corrected for value saturation using the following logic:
If (Yn == Yn-1)
If (((Xn - Yn-1) * fac) > 0)
Yn ++
Else If (((Xn - Yn-1) * fac) < 0)
Yn --
End If
Endif
⌋()

[SWS_Efx_00008] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***	***Associated shift***
0x01	sint16 Efx_LpFilterFac1_s16s16s16_s16 (sint16, sint16, sint16)	15
0x02	sint16 Efx_LpFilterFac1_s16s16u16_s16 (sint16, sint16, uint16)	16
0x03	sint32 Efx_LpFilterFac1_s32s32u16_s32 (sint32, sint32, uint16)	16
0x04	uint16 Efx_LpFilterFac1_u16u16s16_u16 (uint16, uint16, sint16)	15
0x05	uint16 Efx_LpFilterFac1_u16u16u16_u16 (uint16, uint16, uint16)	16
0x06	uint8 Efx_LpFilterFac1_u8u8u8_u8 (uint8, uint8, uint8)	8
0x07	uint32 Efx_LpFilterFac1_u32u32u32_u32 (uint32, uint32, uint32)	32
0x08	uint32 Efx_LpFilterFac1_u32u32u16_u32 (uint32, uint32, uint16)	16

⌋ ( )

8.5.1.2 Third computation

[SWS_Efx_00012]⌈

Service Name	Efx_LpFilter_<InTypeMn>_<OutTypeMn>	
Syntax	`<OutType> Efx_LpFilter_<InTypeMn>_<OutTypeMn> ( <InType> input, <InType> old_output, uint32 tau_const, uint16 recurrence, uint8 reset, <InType> init_val, uint8* started )`	
Service ID [hex]	0x0D and 0x0E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	input	Input signal
	old_output	Previous value of the output value (filtered signal)
	tau_const	Parameter Tau of the filter : the time constant (second)
	recurrence	Delta time between two executions of the function
	reset	Flag to reset the filtered signal
	init_val	Initial value of the filter
Parameters (inout)	started	Pointer to the flag to detect the first call of the function
Parameters (out)	None	
Return value	<OutType>	Return value of the filter
Description	This service computes the first one order discrete filter	
Available via	Efx.h	

⌋()

[SWS_Efx_00013]⌈

If (tau_const==0), then output = input

⌋()

[SWS_Efx_00014]⌈

If (*started==0), then output = init_val

This flag is used to indicate the filter state. *Started = 0, indicates that current function call is the first call of the function to trigger initialisation.

⌋()

[SWS_Efx_00015]⌈
This service computes the first one order discrete filter:

$$output = old\_output + (input - old\_output) * \left(1 - \exp\left(\frac{-recurrence}{tau\_const}\right)\right)$$

$$output = old\_output * \exp\left(\frac{-recurrence}{tau\_const}\right) + input * \left(1 - \exp\left(\frac{-recurrence}{tau\_const}\right)\right)$$

Formula 1

⌋()

Remark : the exponential functions can be computed with interpolations

[SWS_Efx_00016]⌈
if ((reset == 1) or (*started == 0)), then output = init_val
⌋()

[SWS_Efx_00017]⌈
if (*started == 0), then *started=1
⌋()

[SWS_Efx_00018] ⌈
Here is the list of implemented functions.

Service ID[hex]	Syntax
0x0D	uint32 Efx_LpFilter_u32_u32 (uint32, uint32, uint32, uint16, uint8, uint32, uint8*)
0x0E	sint32 Efx_LpFilter_s32_s32 (sint32, sint32, uint32, uint16, uint8, sint32, uint8 *)

⌋ ( )

[SWS_Efx_00020] ⌈input, old_output, and init_val must have the same resolution and the same physical unit. ⌋ ( )

[SWS_Efx_00021] ⌈tau_const and recurrence must have the same resolution and the same physical unit⌋ ( )

It is not recommended to call Efx_LpFilter_<InTypeMn>_<OutTypeMn> under any condition. It must be called at each recurrence, even if it is not used, If the conditions are not fulfilled then output shall be frozen to the previous value all the time.
The parameter started has to be declared as private variable by the caller and shall be initialized to 0 (default init), because the function uses the previous values of this output (so the stack mustn't be used).

8.5.2 First-order High-pass filter

We consider a recursive first-order high-pass filter with a transfer function :

$$H(z) = \frac{b_0 * z + b_1}{z + a}$$

The new return value (Yn) at any point of time can be calculated given the previous value (Yn-1), the current input (Xn), the previous input (Xn-1) and a known constant (K). The formula to calculate the same is as follows:

Yn = Yn-1 – K * Yn-1 + (Xn - Xn-1)

Where $b_0 = 1$, $b_1 = -1$ and a=K -1
The filter is a convergent high-pass filter only if the factor value m is included in [0,1]

[SWS_Efx_00022]⌈

Service Name	Efx_HpFilter_u8_s16	

Syntax	`sint16 Efx_HpFilter_u8_s16 (`
	`  sint16 Yn-1,`
	`  uint8 Xn,`
	`  uint8 Xn-1,`
	`  uint16 K`
	`)`	

Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Yn-1	Previous sint16 output Physical range: [-256 , 255.9921875] Resolution: $1/2^7$
	Xn	Present uint8 input Physical range: [0,255] Resolution: 1
	Xn-1	Previous uint8 input Physical range: [0,255] Resolution: 1
	K	Constant uint16 multiplying factor Physical range: [0,0.99998] Resolution: $1/2^{16}$
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint16	Yn : Result of the calculation Physical range: [-256 , 255.9921875] Resolution: 1/2^7
Description	This service computes the output of a first order high-Pass filter	
Available via	Efx.h	

⌋()
[SWS_Efx_00023]⌈:
Yn = Yn-1 - (K * Yn-1 / 2^{16}) + (Xn - Xn-1)*2^7
The result is rounded towards zero.
⌋()

[SWS_Efx_00024]⌈

Return value shall be saturated to boundary values in the event of negative or positive overflow.
⌋()

[SWS_Efx_00025]⌈
A saturation correction for converging output to zero is applied to the result :
If ((Yn equals Yn-1) and (Yn-1 > 0))
decrement Yn by one
If ((Yn equals Yn-1) and (Yn-1 < 0))
increment Yn by one
⌋()

[SWS_Efx_00026]⌈

Service Name	Efx_HpFilter_s8_s16	
Syntax	`sint16 Efx_HpFilter_s8_s16 ( sint16 Yn-1, sint8 Xn, sint8 Xn-1, uint16 K )`	
Service ID [hex]	0x11	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Yn-1	Previous sint16 output Physical range: [-256 , 255.9921875] Resolution: $1/2^7$
	Xn	Present sint8 input Physical range: [-128 , 127] Resolution: 1
	Xn-1	Previous sint8 input Physical range: [-128 , 127] Resolution: 1
	K	Constant uint16 multiplying factor Physical range: [0,0.99998] Resolution: $1/2^{16}$
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint16	Yn : Result of the calculation Physical range: [-256 , 255.9921875] Resolution: 1/2^7
Description	This service computes the output of a first order high-Pass filter	
Available via	Efx.h	

⌋()
[SWS_Efx_00027]⌈
$Yn = Yn\text{-}1 - (K^* Yn\text{-}1 /2^{16}) + (Xn - Xn\text{-}1)^*2^7$
The result is rounded towards zero.

⌋()

[SWS_Efx_00028]⌈
Return value shall be saturated to boundary values in the event of negative or positive overflow.
⌋()

[SWS_Efx_00029]⌈
A saturation correction for converging output to zero is applied to the result :
If ((Yn equals Yn-1) and (Yn-1 > 0))
decrement Yn by one
If ((Yn equals Yn-1) and (Yn-1 < 0))
increment Yn by one
⌋()

[SWS_Efx_00030]⌈

Service Name	Efx_HpFilter_u16_s32	

Syntax	`sint32 Efx_HpFilter_u16_s32 (`
	`  sint32 Yn-1,`
	`  uint16 Xn,`
	`  uint16 Xn-1,`
	`  uint16 K`
	`)`	

Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Yn-1	Previous sint32 output Physical range: [-65536 , 65535.99996] Resolution: $1/2^{15}$
	Xn	Present uint16 input Physical range: [0,65535] Resolution: 1
	Xn-1	Previous uint16 input Physical range: [0,65535] Resolution: 1
	K	Constant uint16 multiplying factor Physical range: [0,0.99998] Resolution: $1/2^{16}$
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Yn : Result of the calculation Physical range: [-65536 , 65535.99996] Resolution: 1/2^15
Description	This service computes the output of a first order high-Pass filter	
Available via	Efx.h	

⌋()

[SWS_Efx_00031]⌈
$Yn = Yn\text{-}1 - (K^* Yn\text{-}1 /2^{16}) + (Xn - Xn\text{-}1)^*2^{15}$
The result is rounded towards zero.
⌋()

[SWS_Efx_00032]⌈
Return value shall be saturated to boundary values in the event of negative or positive overflow.
⌋()

[SWS_Efx_00033]⌈
A saturation correction for converging output to zero is applied to the result :
If ((Yn equals Yn-1) and (Yn-1 > 0))
decrement Yn by one
If ((Yn equals Yn-1) and (Yn-1 < 0))
increment Yn by one
⌋()

[SWS_Efx_00035]⌈

Service Name	Efx_HpFilter_s16_s32	
Syntax	`sint32 Efx_HpFilter_s16_s32 ( sint32 Yn-1, sint16 Xn, sint16 Xn-1, uint16 K )`	
Service ID [hex]	0x13	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Yn-1	Previous sint32 output Physical range: [-65536 , 65535.99996] Resolution: $1/2^{15}$
	Xn	Present sint16 input Physical range: [-32768,32767] Resolution: 1
	Xn-1	Previous sint16 input Physical range: [-32768,32767] Resolution: 1
	K	Constant uint16 multiplying factor Physical range: [0,0.99998] Resolution: $1/2^{16}$
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Yn : Result of the calculation Physical range: [-65536 , 65535.99996] Resolution: 1/2^31
Description	This service computes the output of a first order high-Pass filter	
Available via	Efx.h	

⌋()

[SWS_Efx_00036]⌈
$Yn = Yn\text{-}1 - (K * Yn\text{-}1/2^{16}) + (Xn - Xn\text{-}1)*2^{15}$
The result is rounded towards zero.
⌋()

[SWS_Efx_00037]⌈
Return value shall be saturated to boundary values in the event of negative or positive overflow.
⌋()

[SWS_Efx_00038]⌈
A saturation correction for converging output to zero is applied to the result :
If ((Yn equals Yn-1) and (Yn-1> 0))
decrement Ynby one
If ((Yn equals Yn-1) and (Yn-1< 0))
increment Yn by one
⌋()

8.5.3 Controller routines

Controller routines includes P, PT1, DT1, PD, I, PI, PID governors used in control system applications. For these controllers, the required parameters are derived using Laplace-Z transformation. The following parameters are required to calculate the new controller output yn and can be represented in the following equation.

Yn = a1 * Yn-1 + b0 * Xn + b1 * Xn-1 + b2 * Xn-2 + ……. + bn-1 * X1 + bn * X0

In the equation, the following symbols are used

Symbols	***Description***
Yn	Actual output to calculate
Yn-1	Output value, one time step before
Xn	Actual input, given from the input
Xn-1	Input, one time step before
Xn-2	Input, two time steps before
X1	Input, n-1 time steps before
X0	Input, n time steps before
a1, b0, b1, b2, bn-1, bn	Controller dependent proportional parameters are used to describe the weight of the states.

8.5.3.1 Structure definitions for controller routines

System parameters are separated from time or time equivalent parameters. The system parameters are grouped in controller dependent structures Efx_Param<controller>_Type, whereas the time (equivalent) parameters are assigned directly. Systems states are grouped in a structure Efx_State<controller>_Type except the actual input value Xn which is assigned directly.

The System parameters, used in the equations are given by:

- K : Amplification factor, The amplification factor K shall have a resolution of 1/2^16.
- T1 : Decay time constant. T1 is expressed in us (micro seconds) and shall have a resolution of 1/10^6.
- Tv : Lead time. Physical unit [sec] describes the Lead time.
  Tv is expressed in us (micro seconds) and shall have a resolution of 1/(2^8 * 10^6)
  Tv range = [0.003906 us, 8388607 us] dT, with respect to [Tv_min, Tv_max]
- Tn : Follow-up time. Physical unit [sec] describes the Follow-up time.
  Tn is expressed in us and have a resolution of 1/10^6.
  Tn is given by a reciprocal value (Tnrec) to avoid a division in the implementation.
  Tnrec is scaled by the factor 2^32.
  Tnrec is given by the equation: 2^32 / (10^6 * Tn).

The time and time equivalent parameters in the equation / implementation are given by:

- dT : Time step = sampling interval. dT is expressed in us (micro seconds) and shall have a resolution of 1/10^6.

Analogous to the abbreviations above, the following abbreviations are used in the implementation:

K_<size>, K_C	: Amplification factor
T1rec_<size>	: Reciprocal delay time constant = 1/ T1. The result shall be Rounded towards Zero.
Tv _<size>, Tv_C	: Lead time
Tnrec _<size>, Tnrec_C	: Reciprocal follow-up time = 1/ Tn. The result shall be Rounded towards Zero.
dT_<size>	: Time step = sampling interval [10^{-6} seconds per increment of 1 data representation unit]
TeQ_<size>	: Time equivalent, TeQ = exp (- dT/ T1).

Herein “<size>” denotes the size of the variable, e.g _s32 stand for a sint32 bit variable.

Note:
1. Tv & Tn cannot be negative
2. Dt should always be greater than zero.

Following C-structures are specially defined for the controller routines.
[SWS_Efx_00040]⌈

Name	Efx_StatePT1_Type
Kind	Structure
Elements	X1

	Type	sint32
	Comment	Input value, one time step before
	Y1	
	Type	sint32
	Comment	Output value, one time step before
Description	System State Structure for PT1 controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00824]**⌈

Name	Efx_StateDT1Typ1_Type	
Kind	Structure	
Elements	X1	
	Type	sint32
	Comment	Input value, one time step before
	X2	
	Type	sint32
	Comment	Input value, two time steps before
	Y1	
	Type	sint32
	Comment	Output value, one time step before
Description	System State Structure for DT1-Type1 controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00825]**⌈

Name	Efx_StateDT1Typ2_Type	
Kind	Structure	
Elements	X1	
	Type	sint32
	Comment	Input value, one time step before
	Y1	
	Type	sint32
	Comment	Output value, one time step before

Description	System State Structure for DT1-Type2 controller routine
Available via	Efx.h

⌋()
[SWS_Efx_00826]⌈

Name	Efx_StatePD_Type	
Kind	Structure	
Elements	X1	
	Type	sint32
	Comment	Input value, one time step before
	Y1	
	Type	sint32
	Comment	Output value, one time step before
Description	System State Structure for PD controller routine	
Available via	Efx.h	

⌋()
[SWS_Efx_00827]⌈

Name	Efx_ParamPD_Type	
Kind	Structure	
Elements	K_C	
	Type	sint32
	Comment	Amplification factor
	Tv_C	
	Type	sint32
	Comment	Lead time
Description	System and Time equivalent parameter Structure for PD controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00828]**⌈

Name	Efx_StateI_Type	
Kind	Structure	
Elements	X1	
	Type	sint32

	Comment	Input value, one time step before
	Y1	
	Type	sint32
	Comment	Output value, one time step before
Description	System State Structure for I controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00829]**⌈

Name	Efx_StatePI_Type	
Kind	Structure	
Elements	X1	
	Type	sint32
	Comment	Input value, one time step before
	Y1	
	Type	sint32
	Comment	Output value, one time step before
Description	System State Structure for PI additive (*Type1 and Type 2*) controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00830]**⌈

Name	Efx_ParamPI_Type	
Kind	Structure	
Elements	K_C	
	Type	sint32
	Comment	Amplification factor
	Tnrec_C	
	Type	sint32
	Comment	Reciprocal follow up time (1/Tn)
Description	System and Time equivalent parameter Structure for PI additive (*Type1 and Type 2*) controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00831]**⌈

Name	Efx_StatePID_Type	
Kind	Structure	
Elements	X1	
	Type	sint32
	Comment	Input value, one time step before
	X2	
	Type	sint32
	Comment	Input value, two time step before
	Y1	
	Type	sint32
	Comment	Output value, one time step before
Description	System State Structure for PID additive (*Type1 and Type 2*) controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00832]**⌈

Name	Efx_ParamPID_Type	
Kind	Structure	
Elements	K_C	
	Type	sint32
	Comment	Amplification factor
	Tv_C	
	Type	sint32
	Comment	Lead time
	Tnrec_C	
	Type	sint32
	Comment	Reciprocal follow up time (1/Tn)
Description	System and Time equivalent parameter Structure for PID additive (*Type1 and Type 2*) controller routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00833]**⌈

Name	Efx_Limits_Type

Kind	Structure	
Elements	Min_C	
	Type	sint32
	Comment	Minimum limit value
	Max_C	
	Type	sint32
	Comment	Maximum limit value
Description	Controller limit value structure	
Available via	Efx.h	

⌋()

8.5.3.2 Proportional Controller

Proportional component calculates Y(x) = Kp * X.

8.5.3.2.1 'P' Controller

[SWS_Efx_00525]⌈

Service Name	Efx_PCalc	

Syntax	`void Efx_PCalc (`
	`  sint32 X_s32,`
	`  sint32* P_ps32,`
	`  sint32 K_s32`
	`)`	

Service ID [hex]	0x14	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	input value
	K_s32	Amplification factor (Quantized with $1/2^{16}$ per increment of 1 data representation unit)
Parameters (inout)	P_ ps32	Pointer to the calculated state
Parameters (out)	None	
Return value	None	
Description	This routine computes differential equation Differential equation: Y = K * X	

Available via	Efx.h

⌋()

[SWS_Efx_00526]⌈
Calculated value *P_ps32 = (K_s32 * X_s32) >> 16
⌋()

[SWS_Efx_00527]⌈
Amplification factor is quantized with $1/2^{16}$ per increment of 1 data representation unit
⌋()

8.5.3.2.2 Set 'P' State

This routine can be realised using inline function.
[SWS_Efx_00044]⌈

Service Name	Efx_PSetState	
Syntax	`void Efx_PSetState ( sint32* P_s32, sint16 Y_s16 )`	
Service ID [hex]	0x21	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Y_s16	Input value
Parameters (inout)	P_s32	Pointer to the calculated state
Parameters (out)	None	
Return value	void	No return value
Description	The routine sets the internal state variables of a P element.	
Available via	Efx.h	

⌋()
[SWS_Efx_00045]⌈
Output value *P_s32 = Y_s16 << 16
⌋()

[SWS_Efx_00046]⌈
The internal state of the P element is stored as (Y_s16 << 16)
⌋()

8.5.3.2.3 Get 'P' output

This routine can be realised using inline function.
[SWS_Efx_00047]⌈

Service Name	Efx_POut_<OutTypeMn>	
Syntax	`<OutType> Efx_POut_<OutTypeMn> ( const sint32* P_ps32 )`	
Service ID [hex]	0x22 to 0x23	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	P_ps32	Pointer to the calculated state
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'P' controller output value
Description	This routine returns 'P' controllers output value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00048]⌈
Output value = *P_ps32 >> 16
⌋()

[SWS_Efx_00049]⌈
Return value shall be saturated to boundary values of the return data type in case of negative or positive overflow.
⌋()

[SWS_Efx_00050] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x22	sint16 Efx_POut_s16(const sint32 *)
0x23	sint8 Efx_POut_s8(const sint32 *)

⌋ ( )

8.5.3.3 Proportional controller with first order time constant

This routine calculates proportional element with first order time constant

8.5.3.3.1 'PT1' Controller

[SWS_Efx_00051]⌈

Service Name	Efx_PT1Calc	
Syntax	`void Efx_PT1Calc ( sint32 X_s32, Efx_StatePT1_Type* State_cpst, sint32 K_s32, sint32 TeQ_s32 )`	
Service ID [hex]	0x2A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the PT1 element
	K_s32	Amplification factor
	TeQ_s32	Time equivalent
Parameters (inout)	State_cpst	Pointer to PT1 state structure
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes PT1 controller output value using below difference equation Yn = exp(-dT/T1) * Yn-1+ K(1 - exp(-dT/T1)) * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00052]⌈
This equation derives implementation :
Output_value = (TeQ_s32 * State_cpst->Y1) + K_s32 * (1 - TeQ_s32) * State_cpst->X1
where TeQ_s32 = exp (-dT/T1)
⌋()

[SWS_Efx_00053]⌈
Efx_CalcTeQ_s32 shall be used for calculation of time equivalent parameter TeQ_s32 only if T1 > 0.
⌋()

Note: If T1 = 0, a PT1 controller behaves like a P controller. In this case, usage of Efx_CalcTeq_s32 should be avoided and Teq value should be passed as 0.

[SWS_Efx_00054]⌈
If (Teq = 0) then PT1 controller follows Input value,
State_cpst->Y1 = k_s32 * State_cpst->X1
⌋()

[SWS_Efx_00055]⌈
calculated Output_value and current input value shall be stored to State_cpst->Y1 and State_cpst->X1 respectively.
State_cpst->Y1 = Output_value
State_cpst->X1 = X_s32
⌋()

8.5.3.3.2 'PT1' Controller – Type1

[SWS_Efx_00531]⌈

Service Name	Efx_PT1Typ1Calc	
Syntax	`void Efx_PT1Typ1Calc ( sint32 X_s32, Efx_StatePT1_Type* State_cpst, sint32 K_s32, sint32 TeQ_s32 )`	
Service ID [hex]	0x38	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the PT1 element
	K_s32	Amplification factor
	TeQ_s32	Time equivalent
Parameters (inout)	State_cpst	Pointer to PT1 state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes PT1 controller output value using below difference equation Yn = exp(-dT/T1) * Yn-1+ K(1 - exp(-dT/T1)) * Xn	
Available via	Efx.h	

⌋()
[SWS_Efx_00532]⌈
This equation derives implementation :
Output_value = (TeQ_s32 * State_cpst->Y1) + K_s32 * (1 - TeQ_s32) * State_cpst->X1
where TeQ_s32 = exp (-dT/T1)
⌋()

[SWS_Efx_00533][
Efx_CalcTeQ_s32 shall be used for calculation of time equivalent parameter TeQ_s32 only if T1 > 0.
⌋()

Note: If T1 = 0, a PT1 controller behaves like a P controller. In this case, usage of Efx_CalcTeq_s32 should be avoided and Teq value should be passed as 0.

[SWS_Efx_00534][
If (Teq = 0) then PT1 controller follows Input value,
State_cpst->Y1 = k_s32 * State_cpst->X1
⌋()

[SWS_Efx_00535][
calculated Output_value and current input value shall be stored to State_cpst->Y1 and State_cpst->X1 respectively.
State_cpst->Y1 = Output_value
State_cpst->X1 = X_s32
⌋()

8.5.3.3.3 'PT1' Set State Value

This routine can be realised using inline function.
[SWS_Efx_00056][

Service Name	Efx_PT1SetState	

Syntax	`void Efx_PT1SetState (`
	`  Efx_StatePT1_Type* State_cpst,`
	`  sint32 X1_s32,`
	`  sint16 Y1_s16`
	`)`	

Service ID [hex]	0x2B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_s32	Initial value for input state
	Y1_s16	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to PT1 state structure
Return value	void	No return value
Description	The routine initialises internal state variables of a PT1 element.	
Available via	Efx.h	

⌋()

[SWS_Efx_00057][
Initialisation of output state variable Y1.
State_cpst->Y1 = Y1_s16 << 16
⌋()

[SWS_Efx_00058][
The internal state of the PT1 element is stored as (Y1_s16 << 16)
⌋()

[SWS_Efx_00059][
Initialisation of input state variable X1.
State_cpst->X1 = X1_s32
⌋()

8.5.3.3.4 Calculate time equivalent Value

This routine can be realised using inline function.
[SWS_Efx_00060][

Service Name	Efx_CalcTeQ_s32	

Syntax	`sint32 Efx_CalcTeQ_s32 (`
	`  sint32 T1rec_s32,`
	`  sint32 dT_s32`
	`)`	

Service ID [hex]	0x2C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	T1rec_s32	Reciprocal delay time
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Time Equivalent TeQ
Description	This routine calculates time equivalent factor	
Available via	Efx.h	

⌋()
[SWS_Efx_00061][
TeQ = exp(-T1rec_s32 * dT_s32)
⌋()

[SWS_Efx_00062]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.3.5 Calculate an approximate time equivalent Value

This routine calculates approximate time equivalent and can be realised using inline function.

[SWS_Efx_00450]⌈

Service Name	Efx_CalcTeQApp_s32	
Syntax	`sint32 Efx_CalcTeQApp_s32 ( sint32 T1rec_s32, sint32 dT_s32 )`	
Service ID [hex]	0x29	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	T1rec_s32	Reciprocal delay time
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Time Equivalent TeQ (Approximate)
Description	This routine calculates time equivalent factor	
Available via	Efx.h	

⌋()

[SWS_Efx_00451]⌈
TeQApp = 1 - (T1rec_s32 * dT_s32)
TeQApp is factorised by 2^16
This approximation is valid only when the product of the physical values of T1rec_s32 and dt_s32 is less than 1. i.e, (T1rec_s32 * dT_s32) < 1
⌋()

[SWS_Efx_00452]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.3.6 Get 'PT1' output

This routine can be realised using inline function.
[SWS_Efx_00063]⌈

Service Name	Efx_PT1Out_<OutTypeMn>	
Syntax	`<OutType> Efx_PT1Out_<OutTypeMn> ( const Efx_StatePT1_Type* State_cpst )`	
Service ID [hex]	0x2D to 0x2E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to constant state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'PT1' controller output value
Description	This routine returns 'PT1' controllers output value.	
Available via	Efx.h	

⌋()
[SWS_Efx_00064]⌈
Output value = State_cpst->Y1_s32 >> 16
⌋()

[SWS_Efx_00065]⌈
Output value shall be normalized by 16 bit right shift of internal state variable.
⌋()

[SWS_Efx_00066]⌈
Return value shall be limited by boundary values of the return data type.
⌋()

[SWS_Efx_00067] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x2D	sint16 Efx_PT1Out_s16(const Efx_StatePT1_Type *)
0x2E	sint8 Efx_PT1Out_s8(const Efx_StatePT1_Type *)

⌋ ( )

8.5.3.4 Differential component with time delay : DT1

This routine calculates differential element with first order time constant.
Routine Efx_CalcTeQ_s32, given in 8.5.3.3.4, shall be used for Efx_DT1_s32 function to calculate the time equivalent TeQ.

8.5.3.4.1 'DT1' Controller – Type1

[SWS_Efx_00070]⌈

Service Name	Efx_DT1Typ1Calc	
Syntax	`void Efx_DT1Typ1Calc ( sint32 X_s32, Efx_StateDT1Typ1_Type* State_cpst, sint32 K_s32, sint32 TeQ_s32, sint32 dT_s32 )`	
Service ID [hex]	0x30	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the DT1 controller
	K_s32	Amplification factor
	TeQ_s32	Time equivalent
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to state structure
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes DT1 controller output value using differential equation, Yn= exp(-dT/T1) * Yn-1+ K * (1- exp(-dT/T1)) * ((Xn-1 - Xn-2) / dT)	
Available via	Efx.h	

⌋()

[SWS_Efx_00071]⌈
This equation derives implementation :
Output_value = (TeQ * State_cpst->Y1) + K_s32 * (1 - TeQ) * ((State_cpst->X1 - State_cpst->X2) / dT)
where TeQ = exp(-dT/T1)
The result shall be Rounded towards Zero.
⌋()

[SWS_Efx_00072]⌈
Efx_CalcTeQ_s32 shall be used for calculation of time equivalent parameter TeQ_s32 only if T1 > 0.
⌋()

Note: If T1 = 0, a DT1 controller behaves like a D controller. In this case, usage of Efx_CalcTeq_s32 should be avoided and Teq value should be passed as 0.

[SWS_Efx_00073]⌈
If (Teq = 0), then DT1 controller follows Input value,
Output_value = k_s32 * (State_cpst->X1 - State_cpst->X2) / dT.
⌋()

[SWS_Efx_00074]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00075]⌈
Old input value State->cpst->X1 shall be stored to State_cpst->X2.
State_cpst->X2 = State_cpst->X1

Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00076]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.4.2 'DT1' Controller – Type2

[SWS_Efx_00501]⌈

Service Name	Efx_DT1Typ2Calc	

Syntax	`void Efx_DT1Typ2Calc (`
	`  sint32 X_s32,`
	`  Efx_StateDT1Typ2_Type* State_cpst,`
	`  sint32 K_s32,`
	`  sint32 TeQ_s32,`
	`  sint32 dT_s32`
	`)`	

Service ID [hex]	0x2F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the DT1 controller
	K_s32	Amplification factor
	TeQ_s32	Time equivalent
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation

		unit]
Parameters (inout)	State_cpst	Pointer to state structure
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes DT1 controller output value using differential equation, Yn= exp(-dT/T1) * Yn-1+ K * (1- exp(-dT/T1)) * ((Xn - Xn-1) / dT)	
Available via	Efx.h	

⌋()

[SWS_Efx_00502]⌈
This equation derives implementation :
Output_value = (TeQ * State_cpst->Y1) + K_s32 * (1 - TeQ) * ((X_s32 - State_cpst->X1) / dT)
where TeQ = exp(-dT/T1)
The result shall be Rounded towards Zero.
⌋()

[SWS_Efx_00503]⌈
Efx_CalcTeQ_s32 shall be used for calculation of time equivalent parameter TeQ_s32.
⌋()

[SWS_Efx_00504]⌈
If (Teq = 0), then DT1 controller follows Input value,
Output_value = k_s32 * (X_s32 - State_cpst->X1) / dT
⌋()

[SWS_Efx_00505]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00506]⌈
Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00507]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.4.3 Set ‘DT1’ State Value – Type1

This routine can be realised using inline function.
[SWS_Efx_00077]⌈

Service Name	Efx_DT1Typ1SetState	

Syntax	void Efx_DT1Typ1SetState (
	  Efx_StateDT1Typ1_Type* State_cpst,
	  sint32 X1_s32,
	  sint32 X2_s32,
	  sint16 Y1_s16
	)	

Service ID [hex]	0x31	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_s32	Initial value for the input state X1
	X2_s32	Initial value for the input state X2
	Y1_s16	Initial value for the output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	void	No return value
Description	The routine initialises internal state variables of a DT1 element.	
Available via	Efx.h	

⌋()
[SWS_Efx_00078]⌈
Initialisation of output state variable Y1.
State_cpst->Y1 = Y1_s16 << 16
⌋()

[SWS_Efx_00079]⌈
The internal state of the DT1 element is stored as (Y1_s16 << 16)
⌋()

[SWS_Efx_00080]⌈
Initialisation of input state variables X1 and X2.
State_cpst->X1 = X1_s32
State_cpst->X2 = X2_s32
⌋()

8.5.3.4.4 Set 'DT1' State Value – Type2

This routine can be realised using inline function.
[SWS_Efx_00510]⌈

Service Name	Efx_DT1Typ2SetState

Syntax	`void Efx_DT1Typ2SetState ( Efx_StateDT1Typ2_Type* State_cpst, sint32 X1_s32, sint16 Y1_s16 )`	
Service ID [hex]	0x32	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_s32	Initial value for the input state
	Y1_s16	Initial value for the output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	void	No return value
Description	The routine initialises internal state variables of a DT1 element.	
Available via	Efx.h	

⌋()
[SWS_Efx_00511]⌈
Initialisation of output state variable Y1.
State_cpst->Y1 = Y1_s16 << 16
⌋()

[SWS_Efx_00512]⌈
The internal state of the DT1 element is stored as (Y1_s16 << 16)
⌋()

[SWS_Efx_00513]⌈
Initialisation of input state variable X1.
State_cpst->X1 = X1_s32
⌋()

8.5.3.4.5 Get 'DT1' output – Type1

This routine can be realised using inline function.
[SWS_Efx_00081]⌈

Service Name	Efx_DT1Typ1Out_<OutTypeMn>
Syntax	`<OutType> Efx_DT1Typ1Out_<OutTypeMn> ( const Efx_StateDT1Typ1_Type* State_cpst )`
Service ID [hex]	0x33 to 0x34
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'DT1' controller output value
Description	This routine returns 'DT1' controller's output value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00082]⌈
Output value = State_cpst->Y1 >> 16
⌋()

[SWS_Efx_00083]⌈
Output value shall be normalized by 16 bit right shift of internal state variable.
⌋()

[SWS_Efx_00084]⌈
Return value shall be limited by boundary values of the return data type.
⌋()

[SWS_Efx_00085] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x33	sint16 Efx_DT1Typ1Out_s16(const Efx_StateDT1Typ1_Type *)
0x34	sint8 Efx_DT1Typ1Out_s8(const Efx_StateDT1Typ1_Type *)

⌋ ( )

8.5.3.4.6 Get ‘DT1’ output – Type2

This routine can be realised using inline function.
[SWS_Efx_00515]⌈

Service Name	Efx_DT1Typ2Out_<OutTypeMn>	

Syntax	`<OutType> Efx_DT1Typ2Out_<OutTypeMn> (`
	`const Efx_StateDT1Typ2_Type* State_cpst`
	`)`	

Service ID [hex]	0x35 to 0x36	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure

Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'DT1' controller output value
Description	This routine returns 'DT1' controller's output value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00516]⌈
Output value = State_cpst->Y1 >> 16
⌋()

[SWS_Efx_00517]⌈
Output value shall be normalized by 16 bit right shift of internal state variable.
⌋()

[SWS_Efx_00518]⌈
Return value shall be limited by boundary values of the return data type.
⌋()

[SWS_Efx_00519] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x35	sint16 Efx_DT1Typ2Out_s16(const Efx_StateDT1Typ2_Type *)
0x36	sint8 Efx_DT1Typ2Out_s8(const Efx_StateDT1Typ2_Type *)

⌋ ( )

8.5.3.5 Proportional and Differential controller

This routine is a combination of proportional and differential controller.

8.5.3.5.1 PD Controller

[SWS_Efx_00090]⌈

Service Name	Efx_PDCalc

Syntax	`void Efx_PDCalc (`
	`  sint32 X_s32,`
	`  Efx_StatePD_Type* State_cpst,`
	`  const Efx_ParamPD_Type* Param_cpst,`
	`  sint32 dT_s32`
	`)`

Service ID [hex]	0x3A

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the PD controller
	Param_cpst	Pointer to parameter structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to internal state structure
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes proportional plus derivative controller output value using differential equation: Yn= K(1+Tv/dT) * Xn - K(Tv/dT) * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00091]⌈
This equation derives implementation :
Output_value = (Param_cpst->K_C * (1+ Param_cpst->Tv_C/dT_s32) * X_s32) - (Param_cpst->K_C * (Param_cpst->Tv_C/dT_s32) * State_cpst->X1)
The result shall be Rounded towards Zero.
⌋()

[SWS_Efx_00092]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00093]⌈
Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00094]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.5.2 PD Set State Value

This routine can be realised using inline function.
[SWS_Efx_00095]⌈

Service Name	Efx_PDSetState
Syntax	`void Efx_PDSetState (`

	`Efx_StatePD_Type* State_cpst,`
	`sint32 X1_s32,`
	`sint16 Y1_s16`
	`)`	

Service ID [hex]	0x3B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_s32	Initial value for input state
	Y1_s16	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	void	No return value
Description	The routine initialises internal state variables of a PD element.	
Available via	Efx.h	

⌋()

[SWS_Efx_00096]⌈
Initialisation of output state variable Y1.
State_cpst->Y1 = Y1_s16 << 16
⌋()

[SWS_Efx_00097]⌈
The internal state of the PD element is stored as (Y1_s16 << 16)
⌋()

[SWS_Efx_00098]⌈
Initialisation of input state variable X1.
State_cpst->X1 = X1_s32
⌋()

8.5.3.5.3 Set 'PD' Parameters

This routine can be realised using inline function.
[SWS_Efx_00100]⌈

Service Name	Efx_PDSetParam

Syntax	`void Efx_PDSetParam (`
	`Efx_ParamPD_Type* Param_cpst,`
	`sint32 K_s32,`
	`sint32 Tv_s32`
	`)`

Service ID [hex]	0x3C

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	K_s32	Amplification factor
	Tv_s32	Lead time
Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to internal parameter structure
Return value	void	No return value
Description	The routine sets the parameter structure of a PD element.	
Available via	Efx.h	

⌋()

[SWS_Efx_00101]⌈
Initialisation of amplification factor.
Param_cpst->K_C = K_s32
⌋()

[SWS_Efx_00102]⌈
Initialisation of lead time state variable
Param_cpst->Tv_C = Tv_s32
⌋()

8.5.3.5.4 Get 'PD' output

This routine can be realised using inline function.
[SWS_Efx_00103]⌈

Service Name	Efx_PDOut_<OutTypeMn>	
Syntax	<OutType> Efx_PDOut_<OutTypeMn> (const Efx_StatePD_Type* State_cpcst)	
Service ID [hex]	0x3D to 0x3E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpcst	Pointer to constant state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'PD' controller output value
Description	This routine returns 'PD' controllers output value.	

Available via	Efx.h

⌋()

[SWS_Efx_00104]⌈
Output value = State_cpst->Y1 >> 16
⌋()

[SWS_Efx_00105]⌈
Output value shall be normalized by 16 bit right shift of internal state variable.
⌋()

[SWS_Efx_00106]⌈
Return value shall be limited by boundary values of the return data type.
⌋()

[SWS_Efx_00107] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x3D	sint16 Efx_PDOut_s16(const Efx_StatePD_Type *)
0x3E	sint8 Efx_PDOut_s8(const Efx_StatePD_Type *)

⌋ ( )

8.5.3.6 Integral component

This routine calculates Integration element .

8.5.3.6.1 'I' Controller

[SWS_Efx_00110]⌈

Service Name	Efx_ICalc	
Syntax	`void Efx_ICalc ( sint32 X_s32, Efx_StateI_Type* State_cpst, sint32 K_s32, sint32 dT_s32 )`	
Service ID [hex]	0x40	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'I' controller
	K_s32	Amplification factor
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]

Parameters (inout)	State_cpst	Pointer to state variable.
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes 'I' controller output value using differential equation, Yn = Yn-1 + K * dT * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00111]⌈
This equation derives implementation :
Output_value = State_cpst->Y1 + K_s32 * dT_s32 * State_cpst->X1
⌋()

[SWS_Efx_00112]⌈
Calculated Output_value and current input value shall be stored to State_cpst->Y1 and State_cpst->X1 respectively.
State_cpst->Y1 = Output_value
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00113]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.6.2 'I' Controller with limitation

[SWS_Efx_00455]⌈

Service Name	Efx_ILimCalc	

Syntax	`void Efx_ILimCalc (`
	`  sint32 X_s32,`
	`  Efx_StateI_Type* State_cpst,`
	`  sint32 K_s32,`
	`  const Efx_Limits_Type* Limit_cpst,`
	`  sint32 dT_s32`
	`)`	

Service ID [hex]	0x3F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'I' controller
	K_s32	Amplification factor
	Limit_cpst	Pointer to limit structure

	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to state variable
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes DT1 controller output value using differential equation, Yn = Yn-1 + K * dT * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00456]⌈
This equation derives implementation :
Output_value = State_cpst->Y1 + K_s32 * dT_s32 * State_cpst->X1
⌋()

[SWS_Efx_00457]⌈
Limit output value with minimum and maximum controller limits.
If (Output value < Limit_cpst->Min_C) Then,
Output_value = Limit_cpst->Min_C
If (Output value > Limit_cpst->Max_C) Then,
Output_value = Limit_cpst->Max_C
⌋()

[SWS_Efx_00458]⌈
Calculated Output_value and current input value shall be stored to State_cpst->Y1 and State_cpst->X1 respectively.
State_cpst->Y1 = Output_value
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00459]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.6.3 Set limits for controllers

[SWS_Efx_00523]⌈

Service Name	Efx_CtrlSetLimits

Syntax	`void Efx_CtrlSetLimits (`
	`  Efx_Limits_Type* Limit_cpst,`
	`  sint32 Min_s32,`
	`  sint32 Max_s32`
	`)`

Service ID [hex]	0x97	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Min_s32	Minimum limit
	Max_s32	Maximum limit
Parameters (inout)	Limit_cpst	Pointer to limit structure
Parameters (out)	None	
Return value	None	
Description	Update limit structure	
Available via	Efx.h	

⌋()

[SWS_Efx_00524]⌈
Update limit structure
Limit_cpst->Min_C = Min_s32
Limit_cpst->Max_C = Max_s32
⌋()

8.5.3.6.4 Set 'I' State Value

This routine can be realised using inline function.
[SWS_Efx_00114]⌈

Service Name	Efx_ISetState	

Syntax	`void Efx_ISetState (`
	`  Efx_StateI_Type* State_cpst,`
	`  sint32 X1_s32,`
	`  sint16 Y1_s16`
	`)`	

Service ID [hex]	0x41	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_s32	Initial value for input state
	Y1_s16	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	void	No return value

Description	The routine initialises internal state variables of an I element.
Available via	Efx.h

⌋()

[SWS_Efx_00115]⌈
Initialisation of output state variable Y1.
State_cpst->Y1 = Y1_s16 << 16
⌋()

[SWS_Efx_00116] ⌈
The internal state of the DT1 element is stored as (Y1_s16 << 16)
⌋ ( )

[SWS_Efx_00117] ⌈
Initialisation of input state variable X1.
State_cpst->X1 = X1_s32
⌋ ( )

8.5.3.6.5 Get 'I' output

This routine can be realised using inline function.
[SWS_Efx_00118]⌈

Service Name	Efx_IOut_<OutTypeMn>	

Syntax	`<OutType> Efx_IOut_<OutTypeMn> (`
	`  const Efx_StateI_Type* State_cpst`
	`)`	

Service ID [hex]	0x43 to 0x44	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to constant state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'I' controller output value
Description	This routine returns 'I' controller's output value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00119]⌈
Output value = State_cpst->Y1 >> 16
⌋()

[SWS_Efx_00120] ⌈
Output value shall be normalized by 16 bit right shift of internal state variable.
⌋ ( )

[SWS_Efx_00121] ⌈
Return value shall be limited by boundary values of the return data type.
⌋ ( )

[SWS_Efx_00122] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0x43	sint16 Efx_IOut_s16(const Efx_StateI_Type*)
0x44	sint8 Efx_IOut_s8(const Efx_StateI_Type *)

⌋ ( )

8.5.3.7 Proportional and Integral controller

This routine is a combination of proportional and integral controller. Routine Efx_CtrlSetLimits shall be used to set limits for this controller in case of limited functionality.

8.5.3.7.1 'PI' Controller – Type1 (Implicit type)

[SWS_Efx_00125]⌈

Service Name	Efx_PITyp1Calc	
Syntax	`void Efx_PITyp1Calc ( sint32 X_s32, Efx_StatePI_Type* State_cpst, const Efx_ParamPI_Type* Param_cpst, sint32 dT_s32 )`	
Service ID [hex]	0x45	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure

	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure.
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes Proportional plus integral controller (implicit type) output value using differential equation: Yn= Yn-1+ K * Xn - K * (1 - dT/Tn) * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00126]⌈
This equation derives implementation :
Output_value = State_cpst->Y1 + (Param_cpst->K_C * X_s32) - (Param_cpst->K_C * (1 - Param_cpst->Tnrec_C * dT_s32) * State_cpst->X1)
⌋()

[SWS_Efx_00127]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00128]⌈
Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00129]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.7.2 'PI' Controller – Type1 with limitation (Implicit type)

[SWS_Efx_00465]⌈

Service Name	Efx_PITyp1LimCalc

Syntax	`void Efx_PITyp1LimCalc (`
	`  sint32 X_s32,`
	`  Efx_StatePI_Type* State_cpst,`
	`  const Efx_ParamPI_Type* Param_cpst,`
	`  const Efx_Limits_Type* Limit_cpst,`
	`  sint32 dT_s32`
	`)`

Service ID [hex]	0x35

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	Limit_cpst	Pointer to limit structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes Proportional plus integral controller (implicit type) output value using differential equation: Yn = Yn-1+ K * Xn - K * (1 - dT/Tn) * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00466]⌈
This equation derives implementation :
Output_value = State_cpst->Y1 + (Param_cpst->K_C * X_s32) - (Param_cpst->K_C * (1 - Param_cpst->Tnrec_C * dT_s32) * State_cpst->X1)
⌋()

[SWS_Efx_00467]⌈
Limit output value with minimum and maximum controller limits.
If (Output value < Limit_cpst->Min_C) Then,
Output_value = Limit_cpst->Min_C
If (Output value > Limit_cpst->Max_C) Then,
Output_value = Limit_cpst->Max_C
⌋()

[SWS_Efx_00468]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00469]⌈
Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00470]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit

⌋()

8.5.3.7.3 'PI' Controller – Type2 (Explicit type)

[SWS_Efx_00130]⌈

Service Name	Efx_PITyp2Calc	
Syntax	`void Efx_PITyp2Calc ( sint32 X_s32, Efx_StatePI_Type* State_cpst, const Efx_ParamPI_Type* Param_cpst, sint32 dT_s32 )`	
Service ID [hex]	0x46	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure.
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes Proportional plus integral controller (explicit type) output value using differential equation: Yn= Yn-1 + K * (1 + dT/Tn) * Xn - K * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00131]⌈
This equation derives implementation :
Output_value = State_cpst->Y1 + (Param_cpst->K_C * (1 + Param_cpst->Tnrec_C * dT_s32) * X_s32) - (Param_cpst->K_C * State_cpst->X1)
⌋()

[SWS_Efx_00132] ⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋ ( )

[SWS_Efx_00133] ⌈
Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋ ( )

[SWS_Efx_00134] ⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋ ( )

8.5.3.7.4 'PI' Controller – Type2 with limitation (Explicit type)

[SWS_Efx_00475]⌈

Service Name	Efx_PITyp2LimCalc	
Syntax	`void Efx_PITyp2LimCalc ( sint32 X_s32, Efx_StatePI_Type* State_cpst, const Efx_ParamPI_Type* Param_cpst, const Efx_Limits_Type* Limit_cpst, sint32 dT_s32 )`	
Service ID [hex]	0x36	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	Limit_cpst	Pointer to limit structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes Proportional plus integral controller (explicit type) output value using differential equation: Yn = Yn-1 + K * (1 + dT/Tn) * Xn - K * Xn-1	
Available via	Efx.h	

⌋()

[SWS_Efx_00476]⌈
This equation derives implementation :

Output_value = State_cpst->Y1 + (Param_cpst->K_C * (1 + Param_cpst->Tnrec_C * dT_s32) * X_s32) - (Param_cpst->K_C * State_cpst->X1)
⌋()

[SWS_Efx_00477]⌈
Limit output value with minimum and maximum controller limits.
If (Output value < Limit_cpst->Min_C) Then,
Output_value = Limit_cpst->Min_C
If (Output value > Limit_cpst->Max_C) Then,
Output_value = Limit_cpst->Max_C
⌋()

[SWS_Efx_00478]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00479]⌈
Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00480]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.7.5 Set 'PI' State Value

This routine can be realised using inline function.
[SWS_Efx_00135]⌈

Service Name	Efx_PISetState	
Syntax	`void Efx_PISetState ( Efx_StatePI_Type* State_cpst, sint32 X1_s32, sint16 Y1_s16 )`	
Service ID [hex]	0x47	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_s32	Initial value for input state
	Y1_s16	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	void	No return value

Description	The routine initialises internal state variables of a PI element.
Available via	Efx.h

⌋()
[SWS_Efx_00136]⌈
Initialisation of output state variable Y1.
State_cpst->Y1 = Y1_s16 << 16
⌋()

[SWS_Efx_00137]⌈
The internal state of the PD element is stored as (Y1_s16 << 16)
⌋()

[SWS_Efx_00138]⌈
Initialisation of input state variable X1.
State_cpst->X1 = X1_s32
⌋()

8.5.3.7.6 Set 'PI' Parameters

This routine can be realised using inline function.
[SWS_Efx_00139]⌈

Service Name	Efx_PISetParam	
Syntax	`void Efx_PISetParam ( Efx_ParamPI_Type* Param_cpst, sint32 K_s32, sint32 Tnrec )`	
Service ID [hex]	0x48	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	K_s32	Amplification factor
	Tnrec	Reciprocal follow-up time
Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to internal parameter structure
Return value	void	No return value
Description	The routine sets the parameter structure of a PI element.	
Available via	Efx.h	

⌋()

[SWS_Efx_00140]⌈
Initialisation of amplification factor.
Param_cpst->K_C = K_s32
⌋()

[SWS_Efx_00141]⌈
Initialisation of reciprocal follow up time state variable
Param_cpst->Tnrec_C = Tnrec_s32
⌋()

8.5.3.7.7 Get 'PI' output

This routine can be realised using inline function.
[SWS_Efx_00142]⌈

Service Name	Efx_PIOut_<OutTypeMn>	
Syntax	`<OutType> Efx_PIOut_<OutTypeMn> ( const Efx_StatePI_Type* State_cpst )`	
Service ID [hex]	0x49 to 0x4A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to constant state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'PI' controller output value
Description	This routine returns 'PI' controllers output value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00143]⌈
Output value = State_cpst->Y1 >> 16
⌋()

[SWS_Efx_00144]⌈
Output value shall be normalized by 16 bit right shift of internal state variable.
⌋()

[SWS_Efx_00145]⌈
Return value shall be limited by boundary values of the return data type.
⌋()

[SWS_Efx_00146] ⌈

Here is the list of implemented functions.

Service ID[hex]	Syntax
0x49	sint16 Efx_PIOut_s16(const Efx_StatePI_Type *)
0x4A	sint8 Efx_PIOut_s8(const Efx_StatePI_Type *)

⌋ ( )

8.5.3.8 Proportional, Integral and Differential controller

This routine is a combination of Proportional, integral and differential controller. Routine Efx_CtrlSetLimits shall be used to set limits for this controller in case of limited functionality.

8.5.3.8.1 'PID' Controller – Type1 (Implicit type)

[SWS_Efx_00150]⌈

Service Name	Efx_PIDTyp1Calc	
Syntax	`void Efx_PIDTyp1Calc ( sint32 X_s32, Efx_StatePID_Type* State_cpst, const Efx_ParamPID_Type* Param_cpst, sint32 dT_s32 )`	
Service ID [hex]	0x4B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PID' controller
	Param_cpst	Parameter structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure.
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes Proportional plus integral plus derivative controller (implicit type) output value using differential equation: Yn=Yn-1+ K * (1 + Tv/dT) * Xn- K *(1 - dT/Tn + 2Tv/dT) * Xn-1 + K * (Tv/dT) * Xn-2	
Available via	Efx.h	

⌋()

[SWS_Efx_00151] ⌈
This equation derives implementation :
calc1 = Param_cpst->K_C * (1 + t_val) * X_s32
calc2 = Param_cpst->K_C * (1 - dT_s32 * Param_cpst->Tnrec_C + 2 * t_val) * State_cpst->X1
calc3 = Param_cpst->K_C * t_val * State_cpst->X2
Output_value = State_cpst->Y1 + calc1 - calc2 + calc3
Where t_val = Param_cpst->Tv_C / dT_s32
The result shall be Rounded towards Zero.
⌋ ( )

[SWS_Efx_00152]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00153]⌈
Old input value State_cpst->X1 shall be stored to State_cpst->X2
State_cpst->X2 = State_cpst->X1

Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00154]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.8.2 'PID' Controller – Type1 with limitation (Implicit type)

[SWS_Efx_00485]⌈

Service Name	Efx_PIDTyp1LimCalc	
Syntax	`void Efx_PIDTyp1LimCalc ( sint32 X_s32, Efx_StatePID_Type* State_cpst, const Efx_ParamPID_Type* Param_cpst, const Efx_Limits_Type* Limit_cpst, sint32 dT_s32 )`	
Service ID [hex]	0x37	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure

	Limit_cpst	Pointer to limit structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure.
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes Proportional plus integral plus derivative controller (implicit type) output value using differential equation: Yn=Yn-1+ K * (1 + Tv/dT) * Xn- K *(1 - dT/Tn + 2Tv/dT) * Xn-1 + K * (Tv/dT) * Xn-2	
Available via	Efx.h	

⌋()

[SWS_Efx_00486]⌈
This equation derives implementation :
calc1 = Param_cpst->K_C * (1 + t_val) * X_s32
calc2 = Param_cpst->K_C * (1 - dT_s32 * Param_cpst->Tnrec_C + 2 * t_val) * State_cpst->X1
calc3 = Param_cpst->K_C * t_val * State_cpst->X2
Output_value = State_cpst->Y1 + calc1 - calc2 + calc3
Where t_val = Param_cpst->Tv_C / dT_s32
⌋()

[SWS_Efx_00487]⌈
Limit output value with minimum and maximum controller limits.
If (Output value < Limit_cpst->Min_C) Then,
Output_value = Limit_cpst->Min_C
If (Output value > Limit_cpst->Max_C) Then,
Output_value = Limit_cpst->Max_C
⌋()

[SWS_Efx_00488]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00489]⌈
Old input value State_cpst->X1 shall be stored to State_cpst->X2
State_cpst->X2 = State_cpst->X1

Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00490]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit

⌋()

8.5.3.8.3 'PID' Controller – Type2

[SWS_Efx_00155]⌈

Service Name	Efx_PIDTyp2Calc	
Syntax	`void Efx_PIDTyp2Calc ( sint32 X_s32, Efx_StatePID_Type* State_cpst, const Efx_ParamPID_Type* Param_cpst, sint32 dT_s32 )`	
Service ID [hex]	0x4C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PID' controller
	Param_cpst	Parameter structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure.
Parameters (out)	None	
Return value	void	No return value
Description	This routine computes Proportional plus integral plus derivative controller (explicit type) output value using differential equation: Yn = Yn-1 + K * (1 + dT/Tn+ Tv/dT) * Xn- K *(1 + 2Tv/dT) * Xn-1+ K * (Tv/dT) * Xn-2	
Available via	Efx.h	

⌋()

[SWS_Efx_00156]⌈
This equation derives implementation :
calc1 = Param_cpst->K_C * (1 + dT_s32 * Param_cpst->Tnrec_C + t_val) * X_s32
calc2 = Param_cpst->K_C * (1 + 2 * t_val) * State_cpst->X1
calc3 = Param_cpst->K_C * t_val * State_cpst->X2
Output_value = State_cpst->Y1 + calc1 - calc2 + calc3
Where t_val = Param_cpst->Tv_C / dT_s32
The result shall be Rounded towards Zero.
⌋()

[SWS_Efx_00157]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00158]⌈
Old input value State_cpst->X1 shall be stored to State_cpst->X2
State_cpst->X2 = State_cpst->X1

Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00159]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.8.4 ‘PID’ Controller – Type2 with limitation

[SWS_Efx_00495]⌈

Service Name	Efx_PIDTyp2LimCalc	

Syntax	`void Efx_PIDTyp2LimCalc (`
	`  sint32 X_s32,`
	`  Efx_StatePID_Type* State_cpst,`
	`  const Efx_ParamPID_Type* Param_cpst,`
	`  const Efx_Limits_Type* Limit_cpst,`
	`  sint32 dT_s32`
	`)`	

Service ID [hex]	0x4F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	Limit_cpst	Pointer to limit structure
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout)	State_cpst	Pointer to the internal state structure
Parameters (out)	None	
Return value	void	No return value

Description	This routine computes Proportional plus integral plus derivative controller (explicit type) output value using differential equation: Yn = Yn-1 + K * (1 + dT/Tn+ Tv/dT) * Xn- K *(1 + 2Tv/dT) * Xn-1+ K * (Tv/dT) * Xn-2
Available via	Efx.h

⌋()

[SWS_Efx_00496]⌈
This equation derives implementation :
calc1 = Param_cpst->K_C * (1 + dT_s32 * Param_cpst->Tnrec_C + t_val) * X_s32
calc2 = Param_cpst->K_C * (1 + 2 * t_val) * State_cpst->X1
calc3 = Param_cpst->K_C * t_val * State_cpst->X2
Output_value = State_cpst->Y1 + calc1 - calc2 + calc3
Where t_val = Param_cpst->Tv_C / dT_s32
⌋()

[SWS_Efx_00497]⌈
Limit output value with minimum and maximum controller limits.
If (Output value < Limit_cpst->Min_C) Then,
Output_value = Limit_cpst->Min_C
If (Output value > Limit_cpst->Max_C) Then,
Output_value = Limit_cpst->Max_C
⌋()

[SWS_Efx_00498]⌈
Calculated Output_value shall be stored to State_cpst->Y1.
State_cpst->Y1 = Output_value
⌋()

[SWS_Efx_00499]⌈
Old input value State_cpst->X1 shall be stored to State_cpst->X2
State_cpst->X2 = State_cpst->X1

Current input value X_s32 shall be stored to State_cpst->X1.
State_cpst->X1 = X_s32
⌋()

[SWS_Efx_00500]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.3.8.5 Set 'PID' State Value

This routine can be realised using inline function.
[SWS_Efx_00160]⌈

Service Name	Efx_PIDSetState
Syntax	`void Efx_PIDSetState (`

	`Efx_StatePID_Type* State_cpst,`
	`sint32 X1_s32,`
	`sint32 X2_s32,`
	`sint16 Y1_s16`
	`)`	

Service ID [hex]	0x4D	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_s32	Initial value for input state
	X2_s32	Initial value for input state
	Y1_s16	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	void	No return value
Description	The routine initialises internal state variables of a PID element.	
Available via	Efx.h	

⌋()

[SWS_Efx_00161]⌈
Initialisation of output state variable Y1.
State_cpst->Y1 = Y1_s16 << 16
⌋()

[SWS_Efx_00162]⌈
The internal state of the PD element is stored as (Y1_s16 << 16)
⌋()

[SWS_Efx_00163]⌈
Initialisation of input state variable X1.
State_cpst->X1 = X1_s32
Initialisation of input state variable X2.
State_cpst->X2 = X2_s32
⌋()

8.5.3.8.6 Set 'PID' Parameters

This routine can be realised using inline function.
[SWS_Efx_00164]⌈

Service Name	Efx_PIDSetParam
Syntax	`void Efx_PIDSetParam (` `Efx_ParamPID_Type* Param_cpst,`

	sint32 K_s32, sint32 Tv_s32, sint32 Tnrec_s32)	
Service ID [hex]	0x4E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	K_s32	Amplification factor
	Tv_s32	Lead Time
	Tnrec_s32	Reciprocal follow-up timer
Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to internal parameter structure
Return value	void	No return value
Description	The routine sets the parameter structure of a PID element.	
Available via	Efx.h	

⌋()

[SWS_Efx_00165]⌈
Initialisation of amplification factor.
Param_cpst->K_C = K_s32
⌋()

[SWS_Efx_00166] ⌈
Initialisation of lead time state variable
Param_cpst->Tv_C = Tv_s32
⌋ ( )

[SWS_Efx_00167] ⌈
Initialisation of reciprocal follow up time state variable
Param_cpst->Tnrec_C = Tnrec_s32
⌋ ( )

8.5.3.8.7 Get ‘PID’ output

This routine can be realised using inline function.
[SWS_Efx_00168]⌈

Service Name	Efx_PIDOut_<OutTypeMn>
Syntax	<OutType> Efx_PIDOut_<OutTypeMn> (const Efx_StatePID_Type* State_cpst

	)	
Service ID [hex]	0x50 to 0x51	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to constant state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return 'PID' controller output value
Description	This routine returns 'PID' controllers output value.	
Available via	Efx.h	

⌋()
[SWS_Efx_00169]⌈
Output value = State_cpst->Y1 >> 16
⌋()

[SWS_Efx_00170] ⌈
Output value shall be normalized by 16 bit right shift of internal state variable.
⌋ ( )

[SWS_Efx_00171] ⌈
Return value shall be limited by boundary values of the return data type.
⌋ ( )

[SWS_Efx_00172] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x50	sint16 Efx_PIDOut_s16(const Efx_StatePID_Type *)
0x51	sint8 Efx_PIDOut_s8(const Efx_StatePID_Type *)

⌋ ( )

8.5.4 Square root

[SWS_Efx_00175]⌈

Service Name	Efx_Sqrt_u32_u32

Syntax	`uint32 Efx_Sqrt_u32_u32 (`
	`  uint32 x_value`
	`)`

Service ID [hex]	0x52

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [0, 1] Resolution: $1/2^{32}$
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint32	Return value of the function Physical range: [0, 1] Resolution: 1/2^32
Description	This service computes the square root of a value	
Available via	Efx.h	

⌋()

[SWS_Efx_00176]⌈
Result = square_root (x_value)
⌋()

[SWS_Efx_00177]⌈
The result is rounded off.
⌋()

[SWS_Efx_00178]⌈

Service Name	Efx_Sqrt_u16_u16	

Syntax	`uint16 Efx_Sqrt_u16_u16 (`
	`  uint16 x_value`
	`)`	

Service ID [hex]	0x53	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [0, 1] Resolution: $1/2^{16}$
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint16	Return value of the function Physical range: [0, 1] Resolution: 1/2^16
Description	This service computes the square root of a value	
Available via	Efx.h	

⌋()

[SWS_Efx_00179]⌈
Result = square_root (x_value)
⌋()

[SWS_Efx_00180]⌈
The result is rounded off.
⌋()

[SWS_Efx_00181]⌈

Service Name	Efx_Sqrt_u8_u8	

Syntax	`uint8 Efx_Sqrt_u8_u8 (`
	`  uint8 x_value`
	`)`	

Service ID [hex]	0x54	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [0, 1] Resolution: $1/2^8$
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	Return value of the function Physical range: [0, 1] Resolution: 1/2^8
Description	This service computes the square root of a value	
Available via	Efx.h	

⌋()

[SWS_Efx_00182]⌈
Result = square_root (x_value)
⌋()

[SWS_Efx_00183]⌈
The result is rounded off.
⌋()

8.5.5 Exponential

[SWS_Efx_00185]⌈

Service Name	Efx_Exp_s32_s32

Syntax	`sint32 Efx_Exp_s32_s32 (`
	`  sint32 Value1`
	`)`

Service ID [hex]	0x55
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	Value1	Input value
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Return value of the function
Description	The routine returns exponential value of an input value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00186]⌈
Output = e^{-x}
where x = Value1
⌋()

[SWS_Efx_00187]⌈
Output is quantized by 2^16
Output Range = ([0.00004539....22026.4657948] * 2^16) = [2....1443526462]
Input Range = ([-10....10] * 2^16) = [0xFFF60000....0x000A0000]
⌋()

8.5.6 Average

[SWS_Efx_00190]⌈

Service Name	Efx_Average_s32_s32	

Syntax	`sint32 Efx_Average_s32_s32 (`
	`  sint32 value1,`
	`  sint32 value2`
	`)`	

Service ID [hex]	0x5A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value1	Input value1
	value2	Input value2
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Return value of the function
Description	The routine returns average value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00191]⌈
Output = (Value1 + Value2) / 2
⌋()

[SWS_Efx_00192] ⌈
The result is rounded towards zero.
⌋ ( )

8.5.7 Array Average

[SWS_Efx_00193]⌈

Service Name	Efx_Array_Average_<InTypeMn>_<OutTypeMn>	
Syntax	`<OutType> Efx_Array_Average_<InTypeMn>_<OutTypeMn> ( const <InType>* Array, uint16 Count )`	
Service ID [hex]	0x60 and 0x61	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Array	Pointer to an array
	Count	Number of array elements
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return value of the function
Description	The routine returns average value of an array.	
Available via	Efx.h	

⌋()

[SWS_Efx_00194]⌈
Output = (Array[0] + Array[1] + ... + Array[N-1]) / Count
⌋()

[SWS_Efx_00195] ⌈
The result is rounded towards zero.
⌋ ( )

[SWS_Efx_00196] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***

0x60	sint32 Efx_Array_Average_s32_s32(sint32*, uint16)
0x61	sint16 Efx_Array_Average_s16_s16(sint16*, uint16)

⌋ ( )

8.5.8 Moving Average

[SWS_Efx_00197]⌈

Service Name	Efx_MovingAverage_<InTypeMn>_<OutTypeMn>	
Syntax	`<OutType> Efx_MovingAverage_<InTypeMn>_<OutTypeMn> ( Efx_MovingAvrg<InTypeMn>_Type* state, <InType> value )`	
Service ID [hex]	0x6A to 0x6B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	Input value
Parameters (inout)	state	Pointer to sliding average structure
Parameters (out)	None	
Return value	<OutType>	Return value of the function
Description	The routine returns sliding average value of n - 1 last subsequent values of an array plus one new value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00198]⌈
state ->p_beg pointer holds start address of an array
state ->p_end pointer holds end address of an array
state ->p_act pointer holds address of an oldest entry of an array
⌋()

[SWS_Efx_00199] ⌈
state ->sum shall store total sum including 'value' & excluding oldest entry
state ->sum = state ->sum - *(state ->p_act) + value
⌋ ( )

[SWS_Efx_00200] ⌈
In every routine call state ->p_act shall be incremented with wrap around.
This increment ensures that oldest entry gets replaced with new entry.
⌋ ( )

[SWS_Efx_00201] ⌈

Output_value = state->sum / state->n
⌋ ( )

[SWS_Efx_00202] ⌈
If state ->n = 0 the result shall be zero by definition.
⌋ ( )

[SWS_Efx_00203] ⌈
The result is rounded towards zero.
⌋ ( )

Structure definition for function argument
[SWS_Efx_00204]⌈

Name	Efx_MovingAvrgS16_Type	
Kind	Structure	
Elements	sum	
	Type	sint32
	Comment	Sum of array elements
	n	
	Type	sint16
	Comment	Size of an array (only positive values)
	*p_beg	
	Type	sint16
	Comment	Pointer to the first array element
	*p_end	
	Type	sint16
	Comment	Pointer to the last array element
	*p_act	
	Type	sint16
	Comment	Pointer to the oldest entry array element
Description	Structure definition for sliding average routine for sint16 input value	
Available via	Efx.h	

⌋() **[SWS_Efx_00836]**⌈

Name	Efx_MovingAvrgS32_Type
Kind	Structure

Elements	sum	
	Type	sint64
	Comment	Sum of array elements
	n	
	Type	sint32
	Comment	Size of an array (only positive values)
	*p_beg	
	Type	sint32
	Comment	Pointer to the first array element
	*p_end	
	Type	sint32
	Comment	Pointer to the last array element
	*p_act	
	Type	sint32
	Comment	Pointer to the oldest entry array element
Description	Structure definition for sliding average routine for sint32 input value	
Available via	Efx.h	

⌋()

[SWS_Efx_00205] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x6A	sint16 Efx_MovingAverage_s16_s16(Efx_MovingAvrgS16_Type*, sint16)
0x6B	sint32 Efx_MovingAverage_s32_s32(Efx_MovingAvrgS32_Type*, sint32)

⌋ ( )

8.5.9 Hypotenuse

The formula used for calculation in the below hypotenuse requirements is,
sqrt(x_value * x_value/2 + y_value * y_value/2).
This is to achieve the specified resolution in the result.
Warning: Hypotenuse functions shall not be used directly for distance computation because the result has not the same resolution than the inputs.

[SWS_Efx_00210]⌈

Service Name	Efx_Hypot_u32u32_u32
Syntax	`uint32 Efx_Hypot_u32u32_u32 (` `  uint32 x_value,`

	`uint32 y_value` `)`	
Service ID [hex]	0x70	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument Physical range: [0, 1] Resolution: $1/2^{32}$
	y_value	Second argument Physical range: [0, 1] Resolution: $1/2^{32}$
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint32	Return value of the function Physical range: [0, sqrt(2)] Resolution: sqrt(2)/2^32
Description	This service computes the length of a vector	
Available via	Efx.h	

⌋()

[SWS_Efx_00211] ⌈
Result = sqrt(x_value * x_value/2 + y_value * y_value/2)
⌋ ( )

[SWS_Efx_00212] ⌈
The result is rounded off.
⌋ ( )

[SWS_Efx_00213]⌈

Service Name	Efx_Hypot_u16u16_u16	

Syntax	`uint16 Efx_Hypot_u16u16_u16 (`
	 `uint16 x_value,`
	 `uint16 y_value`
	 `)`	

Service ID [hex]	0x71	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument Physical range: [0, 1] Resolution: $1/2^{16}$
	y_value	Second argument Physical range: [0, 1] Resolution: $1/2^{16}$
Parameters (inout)	None	
Parameters (out)	None	

Return value	uint16	Return value of the function Physical range: [0, sqrt(2)] Resolution: sqrt(2)/2^16
Description	This service computes the length of a vector	
Available via	Efx.h	

⌋()

[SWS_Efx_00214] ⌈
Result = sqrt(x_value * x_value/2 + y_value * y_value/2)
⌋ ( )

[SWS_Efx_00215] ⌈
The result is rounded off.
⌋ ( )

[SWS_Efx_00216]⌈

Service Name	Efx_Hypot_u8u8_u8	
Syntax	`uint8 Efx_Hypot_u8u8_u8 ( uint8 x_value, uint8 y_value )`	
Service ID [hex]	0x72	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument Physical range: [0, 1] Resolution: $1/2^8$
	y_value	Second argument Physical range: [0, 1] Resolution: $1/2^8$
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	Return value of the function Physical range: [0, sqrt(2)] Resolution: sqrt(2)/2^8
Description	This service computes the length of a vector	
Available via	Efx.h	

⌋()

[SWS_Efx_00217] ⌈
Result = sqrt(x_value * x_value/2 + y_value * y_value/2)

⌋ ( )

[SWS_Efx_00218] ⌈
The result is rounded off.
⌋ ( )

8.5.10 Trigonometric functions

8.5.10.1 Sine function

[SWS_Efx_00220]⌈

Service Name	Efx_Sin_s32_s32	
Syntax	`sint32 Efx_Sin_s32_s32 ( sint32 x_value )`	
Service ID [hex]	0x75	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-PI, PI[Resolution: 2*PI/(2^32)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Return value of the function Physical range: [-1, 1[Resolution: 1/(2^31)
Description	This service computes the sine of an angle.	
Available via	Efx.h	

⌋()
[SWS_Efx_00222] ⌈
The result is rounded off.
⌋()

[SWS_Efx_00223]⌈

Service Name	Efx_Sin_s16_s16
Syntax	`sint16 Efx_Sin_s16_s16 ( sint16 x_value )`
Service ID [hex]	0x76
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-PI, PI[Resolution: 2*PI/(2^16)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint16	Return value of the function Physical range: [-1, 1[Resolution: 1/(2^15)
Description	This service computes the sine of an angle.	
Available via	Efx.h	

⌋()
[SWS_Efx_00225] ⌈
The result is rounded off.
⌋ ( )

[SWS_Efx_00226]⌈

Service Name	Efx_Sin_s8_s8	

Syntax	`sint8 Efx_Sin_s8_s8 (`
	`  sint8 x_value`
	`)`	

Service ID [hex]	0x77	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-PI, PI[Resolution: 2*PI/(2^8)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint8	Return value of the function Physical range: [-1, 1[Resolution: 1/(2^7)
Description	This service computes the sine of an angle.	
Available via	Efx.h	

⌋()
[SWS_Efx_00228] ⌈
The result is rounded off.
⌋ ( )

8.5.10.2 Cosine function

[SWS_Efx_00229]⌈

Service Name	Efx_Cos_s32_s32	

Syntax	`sint32 Efx_Cos_s32_s32 (`
	`  sint32 x_value`
	`)`	

Service ID [hex]	0x7A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-PI, PI[Resolution: 2*PI/(2^32)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Return value of the function Physical range: [-1, 1[Resolution: 1/(2^31)
Description	This service computes the cosine of an angle.	
Available via	Efx.h	

⌋()

[SWS_Efx_00231]⌈
The result is rounded off.
⌋()

[SWS_Efx_00232]⌈

Service Name	Efx_Cos_s16_s16	

Syntax	`sint16 Efx_Cos_s16_s16 (`
	`  sint16 x_value`
	`)`	

Service ID [hex]	0x7B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-PI, PI[Resolution: 2*PI/(2^16)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint16	Return value of the function Physical range: [-1, 1[Resolution: 1/(2^15)
Description	This service computes the cosine of an angle.	

Available via	Efx.h

⌋()
[SWS_Efx_00234]⌈
The result is rounded off.
⌋()

[SWS_Efx_00235]⌈

Service Name	Efx_Cos_s8_s8	

Syntax	`sint8 Efx_Cos_s8_s8 (`
	`  sint8 x_value`
	`)`	

Service ID [hex]	0x7C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-PI, PI[Resolution: 2*PI/(2^8
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint8	Return value of the function Physical range: [-1, 1[Resolution: 1/(2^7)
Description	This service computes the cosine of an angle.	
Available via	Efx.h	

⌋()
[SWS_Efx_00237]⌈
The result is rounded off.
⌋()

8.5.10.3 Inverse Sine function

[SWS_Efx_00240]⌈

Service Name	Efx_ArcSin_s32_s32

Syntax	`sint32 Efx_ArcSin_s32_s32 (`
	`  sint32 x_value`
	`)`

Service ID [hex]	0x80
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	x_value	Argument Physical range: [-1, 1[Resolution: 1/(2^31)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Return value of the function Physical range: [-PI/2 , PI/2[Resolution: 2*PI/(2^32)
Description	This service computes the inverse sine of a value.	
Available via	Efx.h	

⌋()
[SWS_Efx_00242]⌈
The result is rounded off.
⌋()

[SWS_Efx_00243]⌈

Service Name	Efx_ArsSin_s16_s16	
Syntax	`sint16 Efx_ArsSin_s16_s16 ( sint16 x_value )`	
Service ID [hex]	0x81	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-1, 1[Resolution: 1/(2^15)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint16	Return value of the function Physical range: [-PI/2, PI/2[Resolution: 2*PI/(2^16)
Description	This service computes the inverse sine of a value.	
Available via	Efx.h	

⌋()
[SWS_Efx_00245]⌈
The result is rounded off.
⌋()

[SWS_Efx_00246]⌈

Service Name	Efx_ArcSin_s8_s8
Syntax	`sint8 Efx_ArcSin_s8_s8 ( sint8 x_value`

	)	
Service ID [hex]	0x82	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-1, 1[Resolution: 1/(2^7)
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint8	Return value of the function Physical range: [-PI/2, PI/2[Resolution: 2*PI/(2^8)
Description	This service computes the inverse sine of a value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00248]⌈
The result is rounded off.
⌋()

8.5.10.4 Inverse cosine function

[SWS_Efx_00250]⌈

Service Name	Efx_ArcCos_s32_u32	

Syntax	`uint32 Efx_ArcCos_s32_u32 (`
	`  sint32 x_value`
	`)`	

Service ID [hex]	0x85	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-1, 1[Resolution: 1/(2^31)
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint32	Return value of the function Physical range: [0 , PI[Resolution: PI/(2^32)
Description	This service computes the inverse cosine of a value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00252]⌈
The result is rounded off.
⌋()

[SWS_Efx_00253]⌈

Service Name	Efx_ArcCos_s16_u16	
Syntax	`uint16 Efx_ArcCos_s16_u16 ( sint16 x_value )`	
Service ID [hex]	0x86	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-1, 1[Resolution: 1/(2^15)
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint16	Return value of the function Physical range: [0 , PI[Resolution: PI/(2^16)
Description	This service computes the inverse cosine of a value.	
Available via	Efx.h	

⌋()
[SWS_Efx_00255]⌈
The result is rounded off.
⌋()

[SWS_Efx_00256]⌈

Service Name	Efx_ArcCos_s8_u8	
Syntax	`uint8 Efx_ArcCos_s8_u8 ( sint8 x_value )`	
Service ID [hex]	0x87	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Argument Physical range: [-1, 1[Resolution: 1/(2^7)
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	Return value of the function Physical range: [0, PI[

		Resolution: PI/(2^8)
Description	This service computes the inverse cosine of a value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00258]⌈
The result is rounded off.
⌋()

8.5.11 Rate limiter

[SWS_Efx_00261]⌈

Service Name	Efx_SlewRate_<InTypeMn>	

Syntax	`void Efx_SlewRate_<InTypeMn> (`
	`  <InType> limit_pos,`
	`  <InType> input,`
	`  <InType> limit_neg,`
	`  <InType>* output,`
	`  uint8* init`
	`)`	

Service ID [hex]	0x8B to 0x8E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	limit_pos	positive slope
	input	Input signal
	limit_neg	negative slope
Parameters (inout)	output	Output signal
	init	Pointer on a flag used to detect the first call of the API
Parameters (out)	None	
Return value	void	No return value
Description	The routine limits the increase and the decrease of the Input entry by using tunable slopes.	
Available via	Efx.h	

⌋()

[SWS_Efx_00262]⌈
If *init==0, *output=input
⌋()

[SWS_Efx_00264]⌈
Input, limit_pos, limit_neg and output must have the same resolution and the same

physical unit.
⌋()

[SWS_Efx_00265]⌈
If the result of the Efx_SlewRate is only computed when some conditions are fulfilled, do not call the slew rate under the condition, but systematically! The slew rate must be called at each recurrence, even if it is not used, because otherwise, the output will be frozen to the previous value all the time, if conditions are not fulfilled.
⌋()

[SWS_Efx_00266]⌈
The parameters given for output and init, for which we receive the addresses, must be declared by the caller as private variables and will be initialized at 0, because the function uses the previous values of these outputs (so the stack must not be used).
⌋()

[SWS_Efx_00267]⌈
Physical values of limit_pos and limit_neg are positive. Internally limit_pos is added to output value and limit_neg is substracted from output value to get upper and lower limit band within which output value is limited.
⌋()

[SWS_Efx_00268]⌈
At first step, when *init==0, output takes the value of input and *init will be put at 1.
⌋()

[SWS_Efx_00269]⌈
limit_pos is added to the output and it becomes the maximum value of the new output
limit_neg is deducted from the output and it becomes the minimum value of the new output.
If input is outside this range, output is limited to these values, in the other case, output takes the value of input
⌋()

[SWS_Efx_00270]⌈
Values of limit_pos and limit_neg shall be adapted to the frequency of the call of the service.
⌋()

[SWS_Efx_00271] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0x8B	void Efx_SlewRate_u16 (uint16, uint16, uint16, uint16 *, uint8 *)
0x8C	void Efx_SlewRate_s16 (uint16, sint16, uint16, sint16 *, uint8 *)
0x8D	void Efx_SlewRate_u32 (uint32, uint32, uint32, uint32 *, uint8 *)
0x8E	void Efx_SlewRate_s32 (uint32, sint32, uint32, sint32 *, uint8 *)

⌋ ( )

8.5.12 Ramp routines

In case of a change of the input value, the ramp output value follows the input value with a specified limited slope.

Efx_ParamRamp_Type and Efx_StateRamp_Type are the data types for storing ramp parameters. Usage of Switch-Routine and Jump-Routine is optional based on the functionality requirement. Usage of Switch-Routine, Jump-Routine, Calc-Routine and Out-Method have the following precondition concerning the sequence of the calls.

- Efx_RampCalcSwitch
- Efx_RampCalcJump
- Efx_RampCalc
- Efx_RampOut_S32

Structure definition for function argument

[SWS_Efx_00275]⌈

Name	Efx_ParamRamp_Type	
Kind	Structure	
Elements	SlopePos_u32	
	Type	uint32
	Comment	Positive slope for ramp in absolute value. The resolution of SlopePos_u32 shall be 1/2^16.
	SlopeNeg_u32	
	Type	uint32
	Comment	Negative slope for ramp in absolute value. The resolution of SlopeNeg_u32 shall be 1/2^16.
Description	Structure definition for Ramp routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00834]**⌈

Name	Efx_StateRamp_Type	
Kind	Structure	
Elements	State_s32	
	Type	sint32
	Comment	State of the ramp
	Dir_s8	
	Type	sint8
	Comment	Ramp direction
	Switch_s8	

	Type	sint8
	Comment	Position of switch
Description	Structure definition for Ramp routine	
Available via	Efx.h	

⌋()

8.5.12.1 Ramp routine

[SWS_Efx_00276]⌈

Service Name	Efx_RampCalc	

Syntax	`void Efx_RampCalc (`
	`  sint32 X_s32,`
	`  Efx_StateRamp_Type* State_cpst,`
	`  const Efx_ParamRamp_Type* Param_cpcst,`
	`  sint32 dT_s32`
	`)`	

Service ID [hex]	0x90	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Target value for the ramp to reach
	Param_cpcst	Pointer to parameter structure
	dT_s32	Sample Time [10^{-6} seconds per increment of 1 data representation unit]. dT_s32 shall be > 0.
Parameters (inout)	State_cpst	Pointer to state structure
Parameters (out)	None	
Return value	None	
Description	The ramp output value increases or decreases a value with slope * dT_s32 depending if (State_cpst->State_s32 < X_s32) or (State_cpst->State_s32 > X_s32).	
Available via	Efx.h	

⌋()

[SWS_Efx_00837]⌈

If the ramp state State_cpst->State_s32 has reached or crossed the target value X_s32 while the direction of the ramp had been RISING/FALLING, then set State_cpst->State_s32 = X_s32

⌋()

[SWS_Efx_00278]⌈
If ramp direction is rising then ramp increases a value with slope * dT_s32
if (State_cpst->Dir_s8 == RISING)
State_cpst->State_s32 = State_cpst->State_s32 + (Param_cpcst->SlopePos_u32 * dT_s32)

The minimum value of Param_cpcst->SlopePos_u32 * dT_s32 shall be 1, when Param->SlopePos > 0.
The intermediate results shall be rounded off.
Ex: minimum increment of Param_cpcst->SlopePos_u32 * dT_s32 = 1/(2^16*10^6)
⌋()

[SWS_Efx_00279]⌈
If ramp direction is falling then ramp decreases a value with slope * dT_s32
if (State_cpst->Dir_s8 == FALLING)
State_cpst->State_s32 = State_cpst->State_s32 - (Param_cpcst->SlopeNeg_u32 * dT_s32)

The minimum value of Param_cpcst->SlopeNeg_u32 * dT_s32 shall be 1, when Param->SlopeNeg > 0.
The intermediate results shall be rounded off.
Ex: minimum decrement of Param_cpcst->SlopeNeg_u32 * dT_s32 = 1/(2^16*10^6)
⌋()

[SWS_Efx_00280]⌈
Direction of the ramp is stored so that a change of the target can be recognized and the output will follow immediately to the new target value.
State_cpst->Dir_s8 states are: RISING, FALLING, END.
⌋()

[SWS_Efx_00281]⌈
Comparison of State and Target decides ramp direction
If(State_cpst->State_s32 > X_s32) then State_cpst->Dir_s8 = FALLING
If(State_cpst->State_s32 < X_s32) then State_cpst->Dir_s8 = RISING
If(State_cpst->State_s32 == X_s32) then State_cpst->Dir_s8 = END
⌋()

[SWS_Efx_00284]⌈
Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit
⌋()

8.5.12.2 Ramp Initialisation

[SWS_Efx_00285]⌈

Service Name	Efx_RampInitState

Syntax	`void Efx_RampInitState (`
	`  Efx_StateRamp_Type* State_cpst,`
	`  sint32 Val_s32`

	)	
Service ID [hex]	0x91	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Val_s32	Initial value for state variable
Parameters (inout)	State_cpst	Pointer to the state structure
Parameters (out)	None	
Return value	None	
Description	Initializes the state, direction and switch parameters for the ramp.	
Available via	Efx.h	

⌋()

[SWS_Efx_00286]⌈
Ramp direction is initialised with END value. User has no possibility to change or modify ramp direction.
State_cpst->Dir_s8 = END
E.g. of ramp direction states: RISING = 1, FALLING = -1, END = 0
⌋()

[SWS_Efx_00442]⌈
Initialisation of state variable
State_cpst->State_s32 = Val_s32
⌋()

[SWS_Efx_00443]⌈
Initialisation of switch variable. User has no possibility to change or modify switch initialization value.
State_cpst->Switch_s8 = OFF
E.g. of switch states: TARGET_A = 1, TARGET_B = -1, OFF = 0
⌋()

8.5.12.3 Ramp Set Slope

[SWS_Efx_00287]⌈

Service Name	Efx_RampSetParam

Syntax	`void Efx_RampSetParam (`
	`  Efx_ParamRamp_Type* Param_cpst,`
	`  uint32 SlopePosVal_u32,`
	`  uint32 SlopeNegVal_u32`
	`)`

Service ID [hex]	0x92
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	SlopePosVal_u32	Positive slope value
	SlopeNegVal_u32	Negative slope value
Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to parameter structure
Return value	None	
Description	Sets the slope parameter for the ramp provided by the structure Efx_Param Ramp_Type.	
Available via	Efx.h	

⌋()
[SWS_Efx_00288]⌈
Sets positive and negative ramp slopes.
Param_cpst->SlopePos_u32 = SlopePosVal_u32
Param_cpst ->SlopeNeg_u32 = SlopeNegVal_u32
⌋()

8.5.12.4 Ramp out routines

[SWS_Efx_00289]⌈

Service Name	Efx_RampOut_s32	
Syntax	`sint32 Efx_RampOut_s32 ( const Efx_StateRamp_Type* State_cpcst )`	
Service ID [hex]	0x93	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpcst	Pointer to the state value
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint32	Internal state of the ramp element
Description	Returns the internal state of the ramp element.	
Available via	Efx.h	

⌋()
[SWS_Efx_00290]⌈
Return Value = State_cpcst->State_s32

⌋()

8.5.12.5 Ramp Jump routine

[SWS_Efx_00291]⌈

Service Name	Efx_RampCalcJump	
Syntax	`void Efx_RampCalcJump ( sint32 X_s32, Efx_StateRamp_Type* State_cpst )`	
Service ID [hex]	0x94	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_s32	Target value for ramp to jump
Parameters (inout)	State_cpst	Pointer to the state value
Parameters (out)	None	
Return value	None	
Description	This routine works in addition to main ramp function Efx_RampCalc to provide a faster adaption to target value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00292]⌈
If target value changes to a value contrary to current ramp direction and ramp has not reached its old target value then ramp state jumps to new target value immediately.
State_cpst->State_s32 = X_s32
State_cpst->Dir_s8 = END
⌋()

[SWS_Efx_00293] ⌈
If target value is changed to new value and ramp has reached its old target value then normal ramp behavior is maintained.
State_cpst->Dir_s8 = END
⌋ ( )

[SWS_Efx_00303] ⌈
Direction of the ramp is stored so that a change of the target can be recognized and the output will follow immediately to the new target value.
State_cpst->Dir_s8 states are: RISING, FALLING, END.

⌋ ( )

[SWS_Efx_00304] ⌈
Comparison of State and Target decides ramp direction
If(State_cpst->State_s32 > X_s32) then State_cpst->Dir_s8 = FALLING
If(State_cpst->State_s32 < X_s32) then State_cpst->Dir_s8 = RISING
If(State_cpst->State_s32 == X_s32) then State_cpst->Dir_s8 = END
⌋ ( )

[SWS_Efx_00277] ⌈
This routine decided if jump has to be done or not in case of change in target.
Efx_RampCalc function shall be called after this function that a jump or the standard ramp behaviour is executed.
⌋ ( )

8.5.12.6 Ramp switch routine

[SWS_Efx_00520]⌈

Service Name	Efx_RampCalcSwitch	

Syntax	`sint32 Efx_RampCalcSwitch (`
	`  sint32 Xa_s32,`
	`  sint32 Xb_s32,`
	`  boolean Switch,`
	`  Efx_StateRamp_Type* State_cpst`
	`)`	

Service ID [hex]	0x96	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Xa_s32	Target value for the ramp to reach if switch is in position 'A'
	Xb_s32	Target value for the ramp to reach if switch is in position 'B'
	Switch	Switch to decide target value
Parameters (inout)	State_cpst	Pointer to StateRamp structure
Parameters (out)	None	
Return value	sint32	Returns the selected target value
Description	This routine switches between two target values for a ramp service based on a Switch parameter.	
Available via	Efx.h	

⌋()

[SWS_Efx_00521]⌈
Parameter Switch decides which target value is selected.

If Switch = TRUE, then Xa_s32 is selected.
State_cpst->Switch_s8 is set to TARGET_A
Return value = Xa_s32

If Switch = FALSE, then Xb_s32 is selected.
State_cpst->Switch_s8 is set to TARGET_B
Return value = Xb_s32
⌋()

[SWS_Efx_00522]⌈
State_cpst->Dir_s8 hold direction information
State_cpst->Dir_s8 shall be set to END to reset direction information in case of target switch.
⌋()

[SWS_Efx_00528]⌈
Efx_RampCalcSwitch routine has to be called before Efx_RampCalc
⌋()

8.5.12.7 Get Ramp Switch position

[SWS_Efx_00307]⌈

Service Name	Efx_RampGetSwitchPos	

Syntax	`boolean Efx_RampGetSwitchPos (`
	`const Efx_StateRamp_Type* State_cpst`
	`)`	

Service ID [hex]	0x98	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to the state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	return value TRUE or FALSE
Description	Gets the current switch position of ramp switch function.	
Available via	Efx.h	

⌋()

[SWS_Efx_00308]⌈
Return value = TRUE if Switch position State_cpst->Switch_s8 = TARGET_A
Return value = FALSE if Switch position State_cpst->Switch_s8 = TARGET_B
⌋()

Note: The function “Efx_RampGetSwitchPos” should be called only after calling the function “Efx_RampCalcSwitch” or "Efx_RampCalc".

8.5.12.8 Check Ramp Activity

[SWS_Efx_00309]⌈

Service Name	Efx_RampCheckActivity	
Syntax	`boolean Efx_RampCheckActivity ( const Efx_StateRamp_Type* State_cpst )`	
Service ID [hex]	0x99	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to the state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	return value TRUE or FALSE
Description	This routine checks the status of the ramp and returns TRUE if the ramp is active, otherwise it returns FALSE.	
Available via	Efx.h	

⌋()
[SWS_Efx_00310]⌈
return value = TRUE, if Ramp is active (State_cpst->Dir_s8 != END)
return value = FALSE, if Ramp is inactive (State_cpst->Dir_s8 == END)
⌋()

8.5.13 Hysteresis routines

8.5.13.1 Hysteresis

[SWS_Efx_00311]⌈

Service Name	Efx_Hysteresis_<InTypeMn>_<OutTypeMn>
Syntax	`<OutType> Efx_Hysteresis_<InTypeMn>_<OutTypeMn> ( <InType> input,`

	<InType> thresholdLow, <InType> thresholdHigh, <InType> Out_Val, <InType> Out_LowThresholdVal, <InType> Out_HighThresholdVal)	
Service ID [hex]	0x9A to 0x9F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	input	Input signal
	thresholdLow	First threshold used to compute the output
	thresholdHigh	Second threshold used to compute the output
	Out_Val	Output value between the threshold
	Out_LowThresholdVal	Output value for Low Threshold trigger
	Out_HighThresholdVal	Output value for High Threshold trigger
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Return value of the function
Description	The routine estimates the output of the hysteresis.	
Available via	Efx.h	

⌋()

[SWS_Efx_00312]⌈
If Input < thresholdLow, Then return_value = Out_LowThresholdVal
⌋()

[SWS_Efx_00313]⌈
If Input > thresholdHigh, Then return_value = Out_HighThresholdVal
⌋()

[SWS_Efx_00314]⌈
If thresholdLow ≤ Input ≤ thresholdHigh, then return_value = Out_Val
⌋()

[SWS_Efx_00315]⌈
Input, thresholdLow and thresholdHigh must have the same resolution and the same physical unit.
⌋()

[SWS_Efx_00316]⌈
Return_value , Out_Val, Out_LowThresholdVal and Out_HighThresholdVal must

have the same resolution and the same physical unit.
⌋()

[SWS_Efx_00317] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0x9A	uint8 Efx_Hysteresis_u8_u8 (uint8, uint8, uint8, uint8, uint8)
0x9B	uint16 Efx_Hysteresis_u16_u16(uint16, uint16, uint16, uint16, uint16,uint16)
0x9C	uint32 Efx_Hysteresis_u32_u32 (uint32, uint32,uint32,uint32,uint32,uint32)
0x9D	sint8 Efx_Hysteresis_s8_s8 (sint8,sint8,sint8,sint8,sint8,sint8)
0x9E	sint16 Efx_Hysteresis_s16_s16 (sint16,sint16,sint16,sint16,sint16,sint16)
0x9F	sint32 Efx_Hysteresis_s32_s32 (sint32,sint32,sint32,sint32,sint32,sint32)

⌋ ( )

8.5.13.2 Hysteresis center half delta

[SWS_Efx_00320]⌈

Service Name	Efx_HystCenterHalfDelta_<InTypeMn>_<OutTypeMn>	

Syntax	`boolean Efx_HystCenterHalfDelta_<InTypeMn>_<OutTypeMn> (`
	`<InType> X,`
	`<InType> center,`
	`<InType> halfDelta,`
	`boolean* State`
	`)`	

Service ID [hex]	see SWS_Efx_00324	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	center	Center of hysteresis range
	halfDelta	Half width of hysteresis range
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with center and left and right side halfDelta switching point.	
Available via	Efx.h	

⌋()
[SWS_Efx_00321]⌈
Return value = TRUE, if X > center + halfDelta
Return value = FALSE, if X < center - halfDelta
Return value is former state value if
(center - halfDelta) ≤ X ≤ (center + halfDelta)

⌋()

[SWS_Efx_00322]⌈
Parameters X, center and halfDelta should have the same data type.
⌋()

[SWS_Efx_00323]⌈
State variable shall store the old boolean result.
⌋()

[SWS_Efx_00324] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0xA0	boolean Efx_HystCenterHalfDelta_s32_u8(sint32, sint32, sint32, boolean *)
0xA1	boolean Efx_HystCenterHalfDelta_u32_u8 (uint32, uint32, uint32, boolean *)
0x100	boolean Efx_HystCenterHalfDelta_s8_u8 (sint8, sint8, sint8, boolean *)
0x101	boolean Efx_HystCenterHalfDelta_u8_u8 (uint8, uint8, uint8, boolean *)
0x102	boolean Efx_HystCenterHalfDelta_s16_u8(sint16, sint16, sint16, boolean *)
0x103	boolean Efx_HystCenterHalfDelta_u16_u8(uint16, uint16, uint16, boolean *)

⌋ ( )

8.5.13.3 Hysteresis left right

[SWS_Efx_00325]⌈

Service Name	Efx_HystLeftRight_<InTypeMn>_<OutTypeMn>	

Syntax	`boolean Efx_HystLeftRight_<InTypeMn>_<OutTypeMn> (`
	`  <InType> X,`
	`  <InType> Lsp,`
	`  <InType> Rsp,`
	`  boolean* State`
	`)`	

Service ID [hex]	see SWS_Efx_00330	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Lsp	Left switching point
	Rsp	Right switching point
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with left and right switching point.	

Available via	Efx.h

⌋()

[SWS_Efx_00326]⌈
Return value = TRUE, if X > Rsp (right switching point)
Return value = FALSE, if X < Lsp (left switching point)
Return value is former state value if Lsp ≤ X ≤ Rsp
⌋()

[SWS_Efx_00327] ⌈
Parameters X, Lsp and Rsp should have the same data type.
⌋ ( )

[SWS_Efx_00328] ⌈
State variable shall store the old boolean result.
⌋ ( )

[SWS_Efx_00329] ⌈
Rsp shall be always greater than Lsp
⌋ ( )

[SWS_Efx_00330] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xA3	boolean Efx_HystLeftRight_s32_u8 (sint32, sint32, sint32, boolean *)
0xA4	boolean Efx_HystLeftRight_u32_u8 (uint32, uint32, uint32, boolean *)
0x104	boolean Efx_HystLeftRight_s8_u8 (sint8, sint8, sint8, boolean *)
0x105	boolean Efx_HystLeftRight_u8_u8 (uint8, uint8, uint8, boolean *)
0x106	boolean Efx_HystLeftRight_s16_u8(sint16, sint16, sint16, boolean *)
0x107	boolean Efx_HystLeftRight_u16_u8(uint16, uint16, uint16, boolean *)

⌋()

8.5.13.4 Hysteresis delta right

[SWS_Efx_00331]⌈

Service Name	Efx_HystDeltaRight_<InTypeMn>_<OutTypeMn>

Syntax	`boolean Efx_HystDeltaRight_<InTypeMn>_<OutTypeMn> (`
	`  <InType> X,`
	`  <InType> Delta,`
	`  <InType> Rsp,`
	`  boolean* State`
	`)`

Service ID [hex]	see SWS_Efx_00335
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Delta	Left switching point = rsp - delta
	Rsp	Right switching point
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with right switching point and delta to left switching point	
Available via	Efx.h	

⌋()

[SWS_Efx_00332]⌈
Return value = TRUE if X > Rsp (right switching point)
Return value = FALSE if X < (Rsp - Delta)
Return value is former state value if (Rsp - Delta) ≤ X ≤ Rsp
⌋()

[SWS_Efx_00333] ⌈
Parameters X, Rsp and Delta should have the same data type.
⌋ ( )

[SWS_Efx_00334] ⌈
State variable shall store the old boolean result.
⌋ ( )

[SWS_Efx_00335] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xA5	boolean Efx_HystDeltaRight_s32_u8 (sint32, sint32, sint32, boolean *)
0xA6	boolean Efx_HystDeltaRight_u32_u8 (uint32, uint32, uint32, boolean *)
0x108	boolean Efx_HystDeltaRight_s8_u8 (sint8, sint8, sint8, boolean *)
0x109	boolean Efx_HystDeltaRight_u8_u8 (uint8, uint8, uint8, boolean *)
0x10A	boolean Efx_HystDeltaRight_s16_u8(sint16, sint16, sint16, boolean *)
0x10B	boolean Efx_HystDeltaRight_u16_u8(uint16, uint16, uint16, boolean *)

⌋()

8.5.13.5 Hysteresis left delta

[SWS_Efx_00336]⌈

Service Name	Efx_HystLeftDelta_<InTypeMn>_<OutTypeMn>

Syntax	`boolean Efx_HystLeftDelta_<InTypeMn>_<OutTypeMn> ( <InType> X, <InType> Lsp, <InType> Delta, boolean* State )`	
Service ID [hex]	see SWS_Efx_00340	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Lsp	Left switching point
	Delta	Right switching point = lsp + delta
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with left switching point and delta to right switching point.	
Available via	Efx.h	

⌋()

[SWS_Efx_00337]⌈
Return value is TRUE if X > (Lsp + Delta)
Return value is FALSE if X < Lsp
Return value is former state value if Lsp ≤ X ≤ (Lsp + Delta)
⌋()

[SWS_Efx_00338] ⌈
Parameters X, Lsp and Delta should have the same data type.
⌋ ( )

[SWS_Efx_00339] ⌈
State variable shall store the old boolean result.
⌋ ( )

[SWS_Efx_00340] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0xA7	boolean Efx_HystLeftDelta_s32_u8 (sint32, sint32, sint32, boolean *)
0xA8	boolean Efx_HystLeftDelta_u32_u8 (uint32, uint32, uint32, boolean *)
0x10C	boolean Efx_HystLeftDelta_s8_u8 (sint8, sint8, sint8, boolean *)
0x10D	boolean Efx_HystLeftDelta_u8_u8 (uint8, uint8, uint8, boolean *)
0x10E	boolean Efx_HystLeftDelta_s16_u8(sint16, sint16, sint16, boolean *)

0x10F	boolean Efx_HystLeftDelta_u16_u8(uint16, uint16, uint16, boolean *)

⌋()

8.5.14 Debounce routines

8.5.14.1 Efx_Debounce

[SWS_Efx_00355]⌈

Service Name	Efx_Debounce_u8_u8	

Syntax	`boolean Efx_Debounce_u8_u8 (`
	`  boolean X,`
	`  Efx_DebounceState_Type * State,`
	`  const Efx_DebounceParam_Type * Param,`
	`  sint32 dT`
	`)`	

Service ID [hex]	0xB0	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Param	Pointer to state structure of type Efx_DebounceParam_Type
	dT	Sample Time
Parameters (inout)	State	Pointer to state structure of type Efx_DebounceState_Type
Parameters (out)	None	
Return value	boolean	Returns the debounced input value
Description	This routine debounces a digital input signal and returns the state of the signal as a boolean value.	
Available via	Efx.h	

⌋()

[SWS_Efx_00356]⌈
If(X != State->XOld) then check start debouncing.
⌋()

[SWS_Efx_00357] ⌈
If transition occurs from FALSE to TRUE (i.e State->XOld = FALSE and X = TRUE), then use Param->TimeLowHigh as debouncing time; otherwise use Param->TimeHighLow.
⌋ ( )

[SWS_Efx_00358] ⌈

State->Timer is incremented with sample time for debouncing input signal.
Once reached to the set period, old state is updated with X.
State->Timer += dT;
If (State->Timer ≥ (TimePeriod * 10000))
State->XOld = X, and stop the timer, State->Timer = 0
where TimePeriod = Param->TimeLowHigh or Param->TimeHighLow

⌋ ( )

[SWS_Efx_00359] ⌈
Old value shall be returned as a output value. Current input is stored to old state.
Return value = State->XOld
State->XOld = X

⌋ ( )

[SWS_Efx_00360] ⌈
Resolution of dT is 10^{-6} seconds per increment of 1 data representation unit

⌋ ( )

Structure definition for function argument
[SWS_Efx_00361]⌈

Name	Efx_DebounceParam_Type	
Kind	Structure	
Elements	TimeHighLow	
	Type	sint16
	Comment	Time for a High to Low transition, given in 10ms steps
	TimeLowHigh	
	Type	sint16
	Comment	Time for a Low to High transition, given in 10ms steps
Description	Structure definition for Debounce routine	
Available via	Efx.h	

⌋() **[SWS_Efx_00835]**⌈

Name	Efx_DebounceState_Type	
Kind	Structure	
Elements	XOld	
	Type	boolean
	Comment	Old input value from last call
	Timer	

	Type	sint32
	Comment	Timer for internal state
Description	Structure definition for Debounce routine	
Available via	Efx.h	

⌋()

8.5.14.2 Efx_DebounceInit

[SWS_Efx_00362]⌈

Service Name	Efx_DebounceInit	
Syntax	`void Efx_DebounceInit ( Efx_DebounceState_Type* State, boolean X )`	
Service ID [hex]	0xB1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Initial value for the input state
Parameters (inout)	None	
Parameters (out)	State	Pointer to state structure of type Efx_DebounceState_Type
Return value	void	No return value
Description	This routine call shall stop the debouncing timer.	
Available via	Efx.h	

⌋()

[SWS_Efx_00363]⌈
State->Timer = 0
⌋()

[SWS_Efx_00364] ⌈
Sets the input state to the given init value.
State->XOld = X;
⌋ ( )

8.5.14.3 Efx_DebounceSetparam

[SWS_Efx_00365]⌈

Service Name	Efx_DebounceSetParam	
Syntax	`void Efx_DebounceSetParam ( Efx_DebounceParam_Type * Param, sint16 THighLow, sint16 TLowHigh )`	
Service ID [hex]	0xB2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	THighLow	Value for TimeHighLow of Efx_DebounceParam_Type
	TLowHigh	Value for TimeLowHigh of Efx_DebounceParam_Type
Parameters (inout)	None	
Parameters (out)	Param	Pointer to state structure of type Efx_DebounceParam_Type
Return value	void	No return value
Description	This routine sets timing parameters, time for high to low transition and time for low to high for debouncing.	
Available via	Efx.h	

⌋()

[SWS_Efx_00366]⌈
Param-> TimeHighLow = THighLow
Param-> TimeLowHigh = TLowHigh
⌋()

8.5.15 Ascending Sort Routine

[SWS_Efx_00370]⌈

Service Name	Efx_SortAscend_<InTypeMn>	
Syntax	`void Efx_SortAscend_<InTypeMn> ( <OutType> * Array, uint16 Num )`	
Service ID [hex]	0xB4 to 0xB9	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Num	Size of an data array
Parameters (inout)	Array	Pointer to an data array
Parameters (out)	None	

Return value	void	No return value
Description	The sorting algorithm modifies the given input array and rearranges data in ascending order.	
Available via	Efx.h	

⌋()

Example for unsigned array :
Input array : uint16 Array [5] = [42, 10, 88, 8, 15]
Result : Array will be sorted to [8, 10, 15, 42, 88]

Example for signed array :
Input array : sint16 Array [5] = [-42, -10, 88, 8, 15]
Result : Array will be sorted to [-42, -10, 8, 15, 88]

[SWS_Efx_00372] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xB4	void Efx_SortAscend_s8 (sint8*, uint16)
0xB5	void Efx_SortAscend_u8 (uint8*, uint16)
0xB6	void Efx_SortAscend_u16 (uint16*, uint16)
0xB7	void Efx_SortAscend_s16 (sint16*, uint16)
0xB8	void Efx_SortAscend_u32 (uint32*, uint16)
0xB9	void Efx_SortAscend_s32 (sint32*, uint16)

⌋ ( )

8.5.16 Descending Sort Routine

[SWS_Efx_00373]⌈

Service Name	Efx_SortDescend_<InTypeMn>	
Syntax	`void Efx_SortDescend_<InTypeMn> ( <OutType> * Array, uint16 Num )`	
Service ID [hex]	0xBA to 0xBF	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Num	Size of an data array
Parameters (inout)	Array	Pointer to an data array
Parameters (out)	None	
Return value	void	No return value
Description	The sorting algorithm modifies the given input array and rearranges data in descending order.	

Available via	Efx.h

⌋()
Example for unsigned array :
Input array : uint16 Array [5] = [42, 10, 88, 8, 15]
Result : Array will be sorted to [88, 42, 15, 10, 8]

Example for signed array :
Input array : sint16 Array [5] = [-42, -10, 88, 8, 15]
Result : Array will be sorted to [88, 15, 8, -10, -42]

[SWS_Efx_00375] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xBF	void Efx_SortDescend_s8 (sint8*, uint16)
0xBA	void Efx_SortDescend_u8 (uint8*, uint16)
0xBB	void Efx_SortDescend_u16 (uint16*, uint16)
0xBC	void Efx_SortDescend_s16 (sint16*, uint16)
0xBD	void Efx_SortDescend_u32 (uint32*, uint16)
0xBE	void Efx_SortDescend_s32 (sint32*, uint16)

⌋ ( )

8.5.17 Median sort routine

[SWS_Efx_00376]⌈

Service Name	Efx_MedianSort_<InTypeMn>_<OutTypeMn>	

Syntax	`<OutType> Efx_MedianSort_<InTypeMn>_<OutTypeMn> (`
	`  <InType>* Array,`
	`  uint8 N`
	`)`	

Service ID [hex]	0xC0 to 0xC4, 0xC8	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	N	Size of an array
Parameters (inout)	Array	Pointer to an array
Parameters (out)	None	
Return value	<OutType>	Return value of the function
Description	Sort an array and return its median value	
Available via	Efx.h	

⌋()
[SWS_Efx_00377]⌈

This routine sorts values of an array in ascending order. Input array passed by the pointer shall have sorted values after this routine call.
⌋()

For example:
Input array [5] = [42, 10, 88, 8, 15]
Sorted array[5] = [8, 10, 15, 42, 88]

[SWS_Efx_00378]⌈
Returns the median value of sorted array in case of N is even.
Result = (Sorted_array[N/2] + Sorted_array[(N/2) - 1]) / 2
⌋()

For example:
Sorted_array[4] = [8, 10, 15, 42]
Result = (15 + 10) / 2 = 12

[SWS_Efx_00440]⌈
Returns the median value of sorted array in case of N is odd.
Return_Value = Sorted_array [N/2] = 15
⌋()

For example:
Sorted_array[5] = [8, 10, 15, 42, 88]
Result = 15

[SWS_Efx_00441]⌈
In above calculation, N/2 shall be rounded towards zero.
⌋()

[SWS_Efx_00379] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xC0	uint8 Efx_MedianSort_u8_u8(uint8*, uint8)
0xC1	uint16 Efx_MedianSort_u16_u16(uint16*, uint8)
0xC2	sint16 Efx_MedianSort_s16_s16(sint16*, uint8)
0xC3	sint8 Efx_MedianSort_s8_s8(sint8*, uint8)
0xC4	uint32 Efx_MedianSort_u32_u32(uint32*, uint8)
0xC8	sint32 Efx_MedianSort_s32_s32(sint32*, uint8)

⌋ ( )

8.5.18 Edge detection routines

8.5.18.1 Edge bipol detection

[SWS_Efx_00380]⌈

Service Name	Efx_EdgeBipol_u8_u8
Syntax	`boolean Efx_EdgeBipol_u8_u8 (`

	`boolean Inp_Val,`
	`boolean* Old_Val`
	`)`	

Service ID [hex]	0xC5	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Inp_Val	Actual value of the signal
Parameters (inout)	Old_Val	Pointer to the value of the signal from the last call
Parameters (out)	None	
Return value	boolean	Returns TRUE when the signal has changed since the last call
Description	This routine detects whether a signal has changed since the last call and returns TRUE. If signal has not changed then returns FALSE.	
Available via	Efx.h	

⌋()
[SWS_Efx_00381]⌈
if (Inp_Val != *Old_Val)
return value = TRUE
else
return value = FALSE.
⌋()

8.5.18.2 Edge falling detection

[SWS_Efx_00382]⌈

Service Name	Efx_EdgeFalling_u8_u8	

Syntax	`boolean Efx_EdgeFalling_u8_u8 (`
	`boolean Inp_Val,`
	`boolean* Old_Val`
	`)`	

Service ID [hex]	0xC6	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Inp_Val	Actual value of the signal
Parameters (inout)	Old_Val	Pointer to the value of the signal from the last call

Parameters (out)	None	
Return value	boolean	Returns TRUE when the signal has falling edge
Description	Returns TRUE when the signal has a falling edge, i.e. the signal was TRUE at the last call and FALSE at the actual call of this routine	
Available via	Efx.h	

⌋()

[SWS_Efx_00383]⌈
Return value = TRUE, If (*Old_Val == TRUE && Inp_Val == FALSE)
Return value = FALSE, otherwise.
⌋()

8.5.18.3 Edge rising detection

[SWS_Efx_00384]⌈

Service Name	Efx_EdgeRising_u8_u8	

Syntax	`boolean Efx_EdgeRising_u8_u8 (`
	`  boolean Inp_Val,`
	`  boolean* Old_Val`
	`)`	

Service ID [hex]	0xC7	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Inp_Val	Actual value of the signal
Parameters (inout)	Old_Val	Pointer to the value of the signal from the last call
Parameters (out)	None	
Return value	boolean	Returns TRUE when the signal has rising edge
Description	Returns TRUE when the signal has a rising edge, i.e. the signal was FALSE at the last call and TRUE at the actual call of this routine	
Available via	Efx.h	

⌋()

[SWS_Efx_00385]⌈
Return value = TRUE, If (*Old_Val == FALSE && Inp_Val == TRUE)
Return value = FALSE, otherwise.
⌋()

8.5.19 Interval routines

8.5.19.1 Interval Closed

[SWS_Efx_00386]⌈

Service Name	Efx_IntervalClosed_<InTypeMn>_<OutTypeMn>
Syntax	boolean Efx_IntervalClosed_<InTypeMn>_<OutTypeMn> (<InType> MinVal, <InType> InpVal, <InType> MaxVal)
Service ID [hex]	0xCA to 0xCB
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	MinVal: Minimum limit value InpVal: Actual value of the signal MaxVal: Maximum limit value
Parameters (inout)	None
Parameters (out)	None
Return value	boolean: Returns TRUE when MinVal <= InpVal <= MaxVal
Description	This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively.
Available via	Efx.h

⌋()

[SWS_Efx_00387]⌈
Return value = TRUE, if (MinVal ≤ InpVal ≤ MaxVal)
Return value = FALSE, otherwise.
⌋()

[SWS_Efx_00388] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xCA	boolean Efx_IntervalClosed_s32_u8(sint32, sint32, sint32)
0xCB	boolean Efx_IntervalClosed_u32_u8(uint32, uint32, uint32)

⌋ ( )

8.5.19.2 Interval Open

[SWS_Efx_00390]⌈

Service Name	Efx_IntervalOpen_<InTypeMn>_<OutTypeMn>
Syntax	boolean Efx_IntervalOpen_<InTypeMn>_<OutTypeMn> (sint32 MinVal, sint32 InpVal, sint32 MaxVal)
Service ID [hex]	0xCC to 0xCD
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	MinVal — Minimum limit value InpVal — Actual value of the signal MaxVal — Maximum limit value
Parameters (inout)	None
Parameters (out)	None
Return value	boolean — Returns TRUE when MinVal < InpVal < MaxVal
Description	This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively.
Available via	Efx.h

⌋()
[SWS_Efx_00391]⌈
Return value = TRUE, if (MinVal < InpVal < MaxVal)
Return value = FALSE, otherwise.
⌋()

[SWS_Efx_00392] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xCC	boolean Efx_IntervalOpen_s32_u8(sint32, sint32, sint32)
0xCD	boolean Efx_IntervalOpen_u32_u8(uint32, uint32, uint32)

⌋ ( )

8.5.19.3 Interval Left Open

[SWS_Efx_00393]⌈

Service Name	Efx_IntervalLeftOpen_<InTypeMn>_<OutTypeMn>

Syntax	`boolean Efx_IntervalLeftOpen_<InTypeMn>_<OutTypeMn> ( sint32 MinVal, sint32 InpVal, sint32 MaxVal )`	
Service ID [hex]	0xCE to 0xCF	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	MinVal	Minimum limit value
	InpVal	Actual value of the signal
	MaxVal	Maximum limit value
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	Returns TRUE when MinVal < InpVal <= MaxVal
Description	This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively.	
Available via	Efx.h	

⌋()

[SWS_Efx_00394]⌈
Return value = TRUE, if (MinVal < InpVal ≤ MaxVal)
Return value = FALSE, otherwise.
⌋()

[SWS_Efx_00395] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xCE	boolean Efx_IntervalLeftOpen_s32_u8(sint32, sint32, sint32)
0xCF	boolean Efx_IntervalLeftOpen_u32_u8(uint32, uint32, uint32)

⌋ ( )

8.5.19.4 Interval Right Open

[SWS_Efx_00396]⌈

Service Name	Efx_IntervalRightOpen_<InTypeMn>_<OutTypeMn>
Syntax	`boolean Efx_IntervalRightOpen_<InTypeMn>_<OutTypeMn> ( sint32 MinVal, sint32 InpVal, sint32 MaxVal )`

Service ID [hex]	0xD0 to 0xD1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	MinVal	Minimum limit value
	InpVal	Actual value of the signal
	MaxVal	Maximum limit value
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	Returns TRUE when MinVal <= InpVal < MaxVal
Description	This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively.	
Available via	Efx.h	

⌋()

[SWS_Efx_00397]⌈
Return value = TRUE, if (MinVal ≤ InpVal < MaxVal)
Return value = FALSE, otherwise.
⌋()

[SWS_Efx_00398] ⌈
Here is the list of implemented functions.

Service ID[hex]	Syntax
0xD0	boolean Efx_IntervalRightOpen_s32_u8(sint32, sint32, sint32)
0xD1	boolean Efx_IntervalRightOpen_u32_u8(uint32, uint32, uint32)

⌋ ( )

8.5.20 Counter routines

[SWS_Efx_00399]⌈

Service Name	Efx_CounterSet_<InTypeMn>	
Syntax	`void Efx_CounterSet_<InTypeMn> ( <InType>* CounterVal, <InType> Val )`	
Service ID [hex]	0xD2 to 0xD4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Val	Initial value
Parameters (inout)	CounterVal	Pointer to input value

Parameters (out)	None
Return value	None
Description	The CounterSet routines initialise counter value with initial value • CounterVal = Val;
Available via	Efx.h

⌋() **[SWS_Efx_00404]** ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0xD2	void Efx_CounterSet_u16 (uint16*, uint16)
0xD3	void Efx_CounterSet_u32 (uint32*, uint32)
0xD4	void Efx_CounterSet_u8 (uint8*, uint8)

⌋ ( )

[SWS_Efx_00400]⌈

Service Name	Efx_Counter_<InTypeMn>_<OutTypeMn>	

Syntax	`<OutType> Efx_Counter_<InTypeMn>_<OutTypeMn> (`
	`<InType> * CounterVal`
	`)`	

Service ID [hex]	0xD5 to 0xD7	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	CounterVal	Pointer to input value
Parameters (out)	None	
Return value	<OutType>	Returns value is the new value of the parameter CounterVal.
Description	The counter routines increments the value of the parameter CounterVal by 1.	
Available via	Efx.h	

⌋()
[SWS_Efx_00401]⌈
The return value is the new value of the parameter CounterVal.
* CounterVal ++;
Return value = *CounterVal;
⌋()

[SWS_Efx_00402]⌈
In case of saturation, counter value shall not be reset to 0 and shall not be incremented.
Return value = Saturated value of the counter data type

⌋()

[SWS_Efx_00403] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0xD5	uint8 Efx_Counter_u8_u8 (uint8 *)
0xD6	uint16 Efx_Counter_u16_u16 (uint16 *)
0xD7	uint32 Efx_Counter_u32_u32 (uint32 *)

⌋ ( )

8.5.21 Flip-Flop routine

[SWS_Efx_00405]⌈

Service Name	Efx_RSFlipFlop	
Syntax	`boolean Efx_RSFlipFlop ( boolean R_Val, boolean S_Val, boolean* State_Val )`	
Service ID [hex]	0xEF	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	R_Val	Reset switch - changes the flip flop state to FALSE
	S_Val	Set switch - changes the flip flop state to TRUE
Parameters (inout)	State_Val	Pointer to flip-flop state variable
Parameters (out)	None	
Return value	boolean	Returns the new state of the flip flop
Description	RS flip flop can be set and reset via input switches R_Val and S_Val.	
Available via	Efx.h	

⌋()
[SWS_Efx_00406]⌈
The reset switch is higher prior than the set switch,
e.g. R_Val = TRUE,
S_Val = TRUE
Then state and return value = FALSE
⌋()

[SWS_Efx_00407]⌈
Reset condition :
R_Val = TRUE,
S_Val = FALSE
Then state and return value = FALSE
⌋()

[SWS_Efx_00408]⌈
Set condition :
R_Val = FALSE,
S_Val = TRUE
Then state and return value = TRUE
⌋()

[SWS_Efx_00409]⌈
Invalid condition :
R_Val = FALSE,
S_Val = FALSE
Then state and return value are unchanged
⌋()

8.5.22 Limiter routines

[SWS_Efx_00410]⌈

Service Name	Efx_TypeLimiter_<InTypeMn>_<OutTypeMn>	
Syntax	`<OutType> Efx_TypeLimiter_<InTypeMn>_<OutTypeMn> ( <InType> Input_Val )`	
Service ID [hex]	0xD8 to 0xE9	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Input_Val	Input value to be limited
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Returns the limited value for input
Description	limiter routine	
Available via	Efx.h	

⌋()

[SWS_Efx_00411]⌈
Input value shall be saturated according to the data type of the return parameter. e.g. If return type is sint16 and input data range is uint32, then output value will be limited to sint16 data range.
⌋()

[SWS_Efx_00412] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xD8	uint8 Efx_TypeLimiter_s32_u8 (sint32)
0xD9	uint16 Efx_TypeLimiter_s32_u16 (sint32)
0xDA	uint32 Efx_TypeLimiter_s32_u32 (sint32)

0xDB	sint8 Efx_TypeLimiter_s32_s8 (sint32)
0xDC	sint16 Efx_TypeLimiter_s32_s16 (sint32)
0xDD	uint8 Efx_TypeLimiter_u32_u8 (uint32)
0xDE	uint16 Efx_TypeLimiter_u32_u16 (uint32)
0xDF	sint32 Efx_TypeLimiter_u32_s32 (uint32)
0xE0	sint8 Efx_TypeLimiter_u32_s8 (uint32)
0xE1	sint16 Efx_TypeLimiter_u32_s16 (uint32)
0xE2	uint8 Efx_TypeLimiter_s16_u8 (sint16)
0xE3	uint16 Efx_TypeLimiter_s16_u16 (sint16)
0xE4	sint8 Efx_TypeLimiter_s16_s8 (sint16)
0xE5	uint8 Efx_TypeLimiter_u16_u8 (uint16)
0xE6	sint8 Efx_TypeLimiter_u16_s8 (uint16)
0xE7	sint16 Efx_TypeLimiter_u16_s16 (uint16)
0xE8	uint8 Efx_TypeLimiter_s8_u8 (sint8)
0xE9	sint8 Efx_TypeLimiter_u8_s8 (uint8)

⌋ ( )

8.5.23 64 bits functions

8.5.23.1 General requirements

The usage of 64bits data must remain an exception in the code if the requirement cannot be reached by another mean.

8.5.23.2 Additions

[SWS_Efx_00423]⌈

Service Name	Efx_Add_<InTypeMn><InTypeMn>_<OutTypeMn>	

Syntax	`<OutType> Efx_Add_<InTypeMn><InTypeMn>_<OutTypeMn> (`
	`  <InType> x_value,`
	`  <InType> y_value`
	`)`	

Service ID [hex]	0xF0 to 0xF2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This service makes an addition between the two arguments The addition is not protected against the overflow.	

Available via	Efx.h

⌋()

[SWS_Efx_00424]⌈
Return value = x_value + y_value
⌋()

[SWS_Efx_00843]⌈
Return-value shall be saturated to boundary values in the event of negative or positive overflow.
⌋()

[SWS_Efx_00425] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0xF0	sint64 Efx_Add_s64s32_s64(sint64, sint32)
0xF1	sint64 Efx_Add_s64u32_s64(sint64, uint32)
0xF2	sint64 Efx_Add_s64s64_s64(sint64, sint64)

⌋ ( )

8.5.23.3 Multiplications

[SWS_Efx_00426]⌈

Service Name	Efx_Mul_<InTypeMn><InTypeMn>_<OutTypeMn>	

Syntax	`<OutType> Efx_Mul_<InTypeMn><InTypeMn>_<OutTypeMn> (`
	`  <InType> x_value,`
	`  <InType> y_value`
	`)`	

Service ID [hex]	0xF3 to 0xF5	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This service makes a multiplication between the two arguments The multiplication is not protected against the overflow.	
Available via	Efx.h	

⌋()

[SWS_Efx_00427]⌈
Return value = x_value * y_value
⌋()

[SWS_Efx_00844] ⌈
Return-value shall be saturated to boundary values in the event of negative or positive overflow.
⌋()

[SWS_Efx_00428] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xF3	sint64 Efx_Mul_s64u32_s64(sint64, uint32)
0xF4	sint64 Efx_Mul_s64s32_s64(sint64, sint32)
0xF5	sint64 Efx_Mul_s64s64_s64(sint64, sint64)

⌋ ( )

8.5.23.4 Division

[SWS_Efx_00429]⌈

Service Name	Efx_Div_<InTypeMn><InTypeMn>_<OutTypeMn>	
Syntax	<OutType> Efx_Div_<InTypeMn><InTypeMn>_<OutTypeMn> (<InType> x_value, <InType> y_value)	
Service ID [hex]	0xF6 to 0xFB	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	These services make a division between the two arguments	
Available via	Efx.h	

⌋()

[SWS_Efx_00430]⌈
Return value = x_value / y_value
⌋()

[SWS_Efx_00431]⌈
The result after division by zero is defined by:
If x_value ≥ 0 then the function returns the maximum value of the output type
If x_value < 0 then the function returns the minimum value of the output type
⌋()

[SWS_Efx_00433]⌈
The result is rounded towards 0.
⌋()

[SWS_Efx_00845]⌈
Return-value shall be saturated to boundary values in the event of negative or positive overflow.
⌋()

[SWS_Efx_00434] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xF6	sint64 Efx_Div_s64u32_s64(sint64, uint32)
0xF7	sint64 Efx_Div_s64s32_s64(sint64, sint32)
0xF8	sint32 Efx_Div_s64s32_s32 (sint64, sint32)
0xF9	uint32 Efx_Div_s64s32_u32 (sint64, sint32)
0xFA	sint32 Efx_Div_s64u32_s32 (sint64, uint32)
0xFB	uint32 Efx_Div_s64u32_u32 (sint64, uint32)

⌋ ( )

8.6 Examples of use of functions

None

8.7 Version API

8.7.1 Efx_GetVersionInfo

[SWS_Efx_00815]⌈

Service Name	Efx_GetVersionInfo

Syntax	`void Efx_GetVersionInfo (`
	`  Std_VersionInfoType* versioninfo`
	`)`

Service ID [hex]	0xff
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versioninfo	Pointer to where to store the version information of this module. Format according [BSW00321]
Return value	None	
Description	Returns the version information of this library.	
Available via	Efx.h	

⌋(SRS_BSW_00407, SRS_BSW_00003, SRS_BSW_00318, SRS_BSW_00321)
The version information of a BSW module generally contains:
Module Id
Vendor Id
Vendor specific version numbers (SRS_BSW_00407).

[SWS_Efx_00816] ⌈
If source code for caller and callee of Efx_GetVersionInfo is available, the Efx library should realize Efx_GetVersionInfo as a macro defined in the module's header file. ⌋ (SRS_BSW_00407, SRS_BSW_00411)

8.8 Call-back notifications

None

8.9 Scheduled functions

The EfX library does not have scheduled functions.

8.10 Expected Interfaces

None

8.10.1 Mandatory Interfaces

None

8.10.2 Optional Interfaces

None

8.10.3 Configurable interfaces

None

9 Sequence diagrams

Not applicable.

10 Configuration specification

10.1 Published Information

[SWS_Efx_00814] ⌈The standardized common published parameters as required by SRS_BSW_00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋ (SRS_BSW_00402, SRS_BSW_00374, SRS_BSW_00379)

Additional module-specific published parameters are listed below if applicable.

10.2 Configuration option

[SWS_Efx_00818] ⌈The Efx library shall not have any configuration options that may affect the functional behavior of the routines. I.e. for a given set of input parameters, the outputs shall be always the same. For example, the returned value in case of error shall not be configurable. ⌋ (SRS_LIBS_00001)

However, a library vendor is allowed to add specific configuration options concerning library implementation, e.g. for resources consumption optimization.

11 Not applicable requirements

[SWS_Efx_00822]⌈
These requirements are not applicable to this specification.
⌋(SRS_BSW_00448)