Document Title	Specification of Extended Fixed Point Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	400
Document Classification	Standard

Document Version	2.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
09.12.2011	2.0.0	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.

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
15.11.2010	1.1.0	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
07.12.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

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.

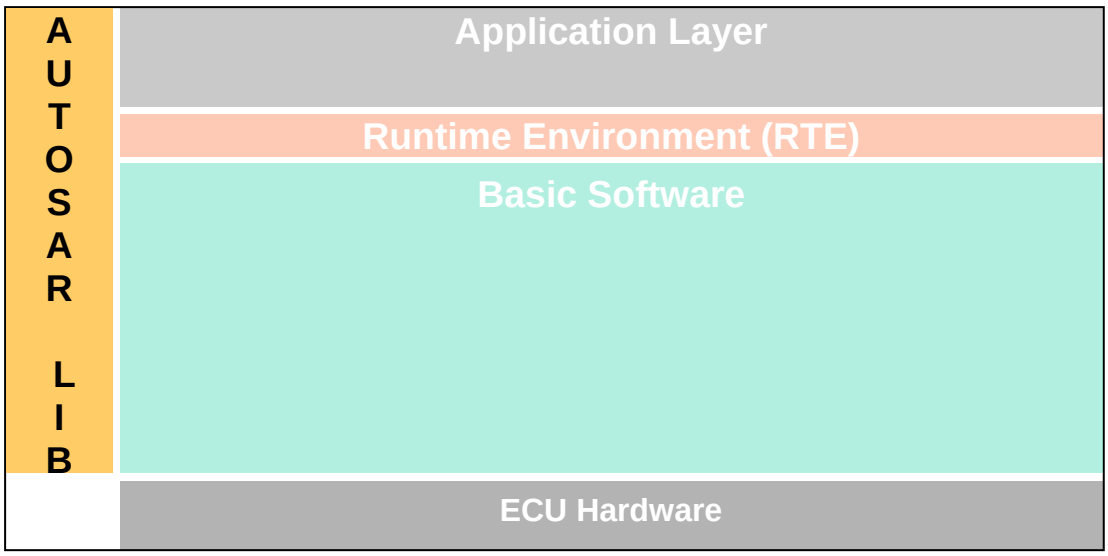

Figure : Layered architecture

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	Development 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

[11] MISRA-C 2004: Guidelines for the use of the C language in critical systems, October 2004

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

[EFX001] ⌈The Efx module shall provide the following files:

- C files, Efx_<name>.c used to implement the library. All C files shall be prefixed with 'Efx_'.
- Header file Efx.h provides all public function prototypes and types defined by the Efx library specification ⌋(BSW31400005)

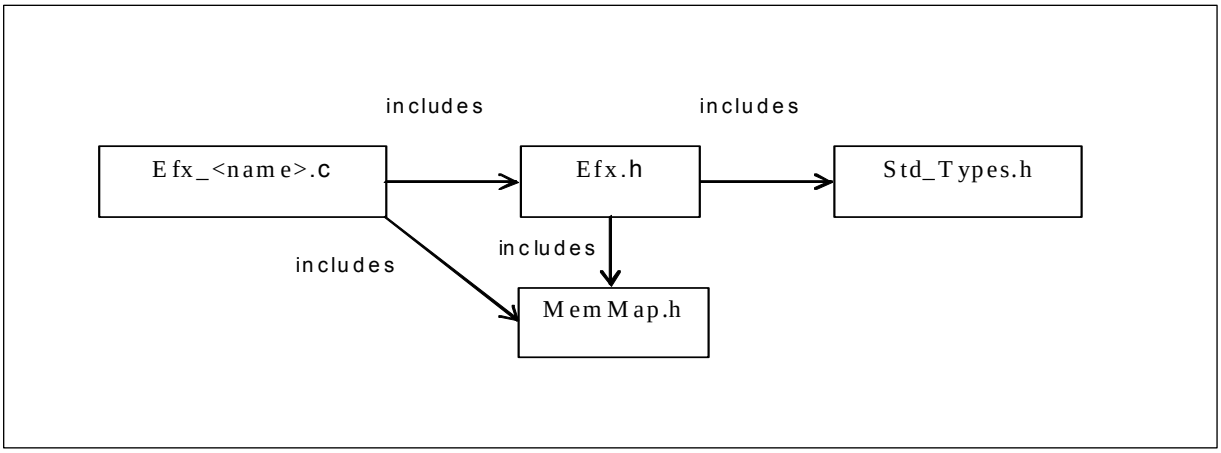

Figure : File structure

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
BSW003		EFX815
BSW00304	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	EFX812
BSW00306	All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, un...	EFX813
BSW00318		EFX815
BSW00321		EFX815
BSW00348	Each AUTOSAR library Module implementation *.	EFX811
BSW00378	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	EFX812
BSW00407		EFX815, EFX816
BSW00411		EFX816
BSW00436	Each AUTOSAR library Module implementation *.	EFX810
BSW007	The library, written in C programming language, should conform to the HIS subset of the MISRA C S...	EFX809
BSW31400002	Efx library shall not require initialization phase.	EFX800
BSW31400003	Efx library shall not require a shutdown operation phase.	EFX801
BSW31400005	The Efx module shall provide the following files:	EFX001
BSW31400013	Error detection: Function should check at runtime (both in production and development code) the v...	EFX819, EFX817
BSW31400015	The Efx library shall be implemented in a way that the code can be shared among callers in differ...	EFX806
BSW31400017	Usage of macros should be avoided.	EFX807
BSW31400018	A library function shall not call any BSW modules functions, e.	EFX808

7 Functional specification

7.1 Error classification

[EFX821] ⌈No error classification definition as DET call not supported by library

7.2 Error detection

[EFX819] ⌈Error detection: Function should check at runtime (both in production and development code) the value of input parameters, especially cases where erroneous value can bring to fatal error or unpredictable result, if they have the values allowed by the function specification. All the error cases shall be listed in SWS and the function should return a specified value (in SWS) that is not configurable. This value is dependant of the function and the error case so it is determined case by case.
If values passed to the routines are not valid and out of the function specification, then such error are not detected.
E.g. If passed value > 32 for a bit-position or a negative number of samples of an axis distribution is passed to a routine. ⌋(BSW31400013)

7.3 Error notification

[EFX817] ⌈The functions shall not call the DET for error notification. ⌋(BSW31400013)

7.4 Initialization and shutdown

[EFX800] ⌈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. ⌋(BSW31400002)

[EFX801] ⌈Efx library shall not require a shutdown operation phase. ⌋(BSW31400003)

7.5 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-DependencyOnLibrary 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.

7.6 library implementation

[EFX806] ⌈The Efx library shall be implemented in a way that the code can be shared among callers in different memory partitions. ⌋(BSW31400015)

[EFX807] ⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used. ⌋(BSW31400017)

[EFX808] ⌈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. ⌋(BSW31400018)

[EFX809] ⌈The library, written in C programming language, should conform to the HIS subset of the MISRA C Standard.
Only in technically reasonable, exceptional cases MISRA violations are permissible. Such violations against MISRA rules shall be clearly identified and documented within comments in the C source code (including rationale why MISRA rule is violated). The comment shall be placed right above the line of code which causes the violation and have the following syntax:
/ MISRA RULE XX VIOLATION: This the reason why the MISRA rule could not be followed in this special case*/*⌋(BSW007)

[EFX810] ⌈Each AUTOSAR library Module implementation <library>*.c and <library>*.h shall map their code to memory sections using the AUTOSAR memory mapping mechanism. ⌋(BSW00436)

[EFX811] ⌈Each AUTOSAR library Module implementation <library>*.c, that uses AUTOSAR integer data types and/or the standard return, shall include the header file Std_Types.h. ⌋(BSW00348)

[EFX812] ⌈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. ⌋(BSW00304, BSW00378)

[EFX813] ⌈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. ⌋(BSW00306)

8 API specification

8.1 Imported types

In this chapter, all types included from the following files 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.

Thus, in order to improve the portability of the software, these types are defined in PlatformTypes.h [6]. The following mnemonic are used in the library routine names.

Size	Platform Type	Mnemonic	Range
unsigned 8-Bit	boolean	NA	[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>).

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

[EFX005] ⌈

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:	<OutType>	Result (Yn) of the calculation

Description:	This service computes the output of a first order low-pass filter:
	EFX006:
	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
	EFX007:
	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	

⌋ ( )

[EFX008] ⌈
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 Second computation

[EFX009] ⌈

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]:	0x0A to 0x0B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Yn-1	Old output value
	Xn	Current measured value
	fac	Factor value that represents the physical range [0, 1]
Parameters (inout):	None	
Parameters (out):	None	
Return value:	<OutType>	Result (Yn)of the calculation

Description:	**EFX010:**
	This service computes the output of a first order low-pass filter:
	Yn = Yn-1 + ((Xn - Yn-1.HIGH) * fac)
	if ((Xn - Yn-1.HIGH) == 0)
	Yn.HIGH = Yn-1.HIGH
	Yn.LOW = 8000H
	Endif
	Yn-1 and Yn are coded with 32 bits.
	Yn-1.HIGH represents the 16 high orders bits
	Yn.LOW represents the 16 low orders bits	

⌋ ( )

[EFX011] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x0A	sint32 Efx_LpFilterFac1_s32s16u16_s32 (sint32, sint16, uint16)
0x0B	uint32 Efx_LpFilterFac1_u32u16u16_u32 (uint32, uint16, uint16)

⌋ ( )

8.5.1.3 Third computation

[EFX012] ⌈

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:	**EFX013:**
	If (tau_const==0), then output = input
	EFX014:
	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.
	EFX015:
	This service computes the first one order discrete filter:
	
	(See Formula 1 below)
	
	Remark : the exponential functions can be computed with interpolations
	EFX016:
	if ((reset == 1) or (*started == 0)), then output = init_val
	EFX017:
	if (*started == 0), then *started=1	

⌋( )

[EFX018] ⌈
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 *)

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

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

Formula 1

⌋( )

[EFX020] ⌈input, old_output, and init_val must have the same resolution and the same physical unit. ⌋( )

[EFX021] ⌈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]

[EFX022] ⌈

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:	**EFX023:**
	This service computes the output of a first order high-Pass filter:
	Yn = Yn-1 - (K * Yn-1 / 2^{16}) + (Xn - Xn-1)*2^{7}
	The division in the result is rounded off.
	EFX024:
	Return value shall be saturated to boundary values in the event of underflow or overflow.
	EFX025:
	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

⌋ ( )

[EFX026] ⌈

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:	**EFX027:**
	This service computes the output of a first order high-Pass filter:
	Yn = Yn-1 - (K* Yn-1 /2^{16}) + (Xn - Xn-1)*2^{7}
	The division in the result is rounded off.
	EFX028:
	Return value shall be saturated to boundary values in the event of underflow or overflow.
	EFX029:
	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

⌋ ( )

[EFX030] ⌈

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:	**EFX031:**
	This service computes the output of a first order high-Pass filter:
	$Yn = Yn-1 - (K* Yn-1 /2^{16}) + (Xn - Xn-1)*2^{15}$
	The division in the result is rounded off.
	EFX032:
	Return value shall be saturated to boundary values in the event of underflow or overflow.
	EFX033:
	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	

⌋ ( )

[EFX035] ⌈

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:	**EFX036:**
	This service computes the output of a first order high-Pass filter:
	Yn = Yn-1- (K* Yn-1/2^{16}) + (Xn- Xn-1)*2^{15}
	The division in the result is rounded off.
	EFX037:
	Return value shall be saturated to boundary values in the event of underflow or overflow.
	EFX038:
	A saturation correction for converging output to zero is applied to the result :
	If ((Ynequals Yn-1) and (Yn-1> 0))
	decrement Ynby one
	If ((Ynequals Yn-1) and (Yn-1< 0))
	increment Ynby 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.

$$Yn = a1 * Yn\text{-}1 + b0 * Xn + b1 * Xn\text{-}1 + b2 * Xn\text{-}2 + \ldots\ldots + bn\text{-}1 * X1 + bn * X0$$

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,	Controller dependent proportional parameters are used to describe the weight of

bn	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 description of the semantic is given in
T1 : Decay time constant
Tv : Lead time
Tn : Follow-up time

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

dT : Time step = sampling interval

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
Tv _<size>, Tv_C	: Lead time
Tnrec _<size>, Tnrec_C	: Reciprocal follow-up time = 1/ Tn.
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.

Following C-structures are specially defined for the controller routines.

[EFX040] ⌈

Name:	`Efx_StatePT1_Type`		
Type:	`Structure`		
Element:	`sint32`	`X1`	Input value, one time step before
	`sint32`	`Y1`	Output value, one time step before
Description:	System State Structure for PT1 controller routine		

Name:	`Efx_StateDT1Typ1_Type`		
Type:	`Structure`		
Element:	`sint32`	`X1`	Input value, one time step before

	sint32	X2	Input value, two time steps before
	sint32	Y1	Output value, one time step before
Description:	System State Structure for DT1-Type1 controller routine		

Name:	Efx_StateDT1Typ2_Type		
Type:	Structure		
Element:	sint32	X1	Input value, one time step before
	sint32	Y1	Output value, one time step before
Description:	System State Structure for DT1-Type2 controller routine		

Name:	Efx_StatePD_Type		
Type:	Structure		
Element:	sint32	X1	Input value, one time step before
	sint32	Y1	Output value, one time step before
Description:	System State Structure for PD controller routine		

Name:	Efx_ParamPD_Type		
Type:	Structure		
Element:	sint32	K_C	Amplification factor
	sint32	Tv_C	Lead time
Description:	System and Time equivalent parameter Structure for PD controller routine		

Name:	Efx_StateI_Type		
Type:	Structure		
Element:	sint32	X1	Input value, one time step before
	sint32	Y1	Output value, one time step before
Description:	System State Structure for I controller routine		

Name:	Efx_StatePI_Type		
Type:	Structure		
Element:	sint32	X1	Input value, one time step before
	sint32	Y1	Output value, one time step before
Description:	System State Structure for PI additive (*Type1 and Type 2*) controller routine		

Name:	Efx_ParamPI_Type		
Type:	Structure		
Element:	sint32	K_C	Amplification factor
	sint32	Tnrec_C	Reciprocal follow up time (1/Tn)
Description:	System and Time equivalent parameter Structure for PI additive (*Type1 and Type 2*) controller routine		

Name:	Efx_StatePID_Type		
Type:	Structure		
Element:	sint32	X1	Input value, one time step before
	sint32	X2	Input value, two time step before
	sint32	Y1	Output value, one time step before

Description:	System State Structure for PID additive (*Type1 and Type 2*) controller routine

Name:	`Efx_ParamPID_Type`		
Type:	`Structure`		
Element:	`sint32`	`K_C`	Amplification factor
	`sint32`	`Tv_C`	Lead time
	`sint32`	`Tnrec_C`	Reciprocal follow up time (1/Tn)
Description:	System and Time equivalent parameter Structure for PID additive (*Type1 and Type 2*) controller routine		

Name:	`Efx_Limits_Type`		
Type:	`Structure`		
Element:	`sint32`	`Min_C`	Minimum limit value
	`sint32`	`Max_C`	Maximum limit value
Description:	Controller limit value structure		

⌋ ( )

8.5.3.2 Proportional Controller

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

8.5.3.2.1 'P' Controller

[EFX041] ⌈

Service name:	Efx_PCalc_s32	

Syntax:	`void Efx_PCalc_s32(`
	`  sint32 X_s32,`
	`  sint32* P_ps32,`
	`  sint32 K_s32`
	`)`	

Service ID[hex]:	0x20	
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:	void	No return value
Description:	Differential equation: Y = K * X **EFX042:** This routine computes differential equation : Calculated value *P_ps32 = (K_s32 * X_s32) >> 16 **EFX043:** Amplification factor is quantized with $1/2^{16}$ per increment of 1 data representation unit	

⌋ ( )
Note : "This routine (Efx_PCalc_s32) is depreciated and will not be supported in future release.
Replacement routine : Efx_PCalc"

[EFX525] ⌈

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:	Differential equation: Y = K * X **EFX526:** This routine computes differential equation : Calculated value *P_ps32 = (K_s32 * X_s32) >> 16 **EFX527:** 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.

[EFX044] ⌈

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:	**EFX045 :** The routine sets the internal state variables of a P element. Output value *P_s32 = Y_s16 << 16 **EFX046 :** 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.

[EFX047] ⌈

Service name:	Efx_POut_<OutTypeMn>	

Syntax:	`<OutType> Efx_POut_<OutTypeMn>(`
	`  const sint32* const 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:	**EFX048:** This routine returns 'P' controllers output value. Output value = *P_ps32 >> 16 **EFX049:** Return value shall be saturated to boundary values of the return data type in case of underflow or overflow.	

⌋ ( )

[EFX050] ⌈

Here is the list of implemented functions.

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

⌋ ( )

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

[EFX051] ⌈

Service name:	Efx_PT1Calc	

Syntax:	`void Efx_PT1Calc(`
	`  sint32 X_s32,`
	`  Efx_StatePT1_Type* const 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:	**EFX052 :** This routine computes PT1 controller output value using below difference equation Yn = exp(-dT/T1) * Yn-1+ K(1 - exp(-dT/T1)) * Xn-1 This derives implementation : Output_value = (TeQ_s32 * State_cpst->Y1) + K_s32 * (1 - TeQ_s32) * State_cpst->X1 where TeQ_s32 = exp (-dT/T1) **EFX053 :** Efx_CalcTeQ_s32 shall be used for calculation of time equivalent parameter TeQ_s32. **EFX054 :** If (T1 = 0) then PT1 controller follows Input value, State_cpst->Y1 = k_s32 * X_s32 **EFX055:** 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' Set State Value

This routine can be realised using inline function.

[EFX056] ⌈

Service name:	Efx_PT1SetState

Syntax:	`void Efx_PT1SetState(`
	`  Efx_StatePT1_Type* const 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. **EFX057 :** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_s16 << 16 **EFX058 :** The internal state of the PT1 element is stored as (Y1_s16 << 16) **EFX059 :** Initialisation of input state variable X1. State_cpst->X1 = X1_s32	

⌋ ( )

8.5.3.3.3 Calculate time equivalent Value

This routine can be realised using inline function.

[EFX060] ⌈

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:	**EFX061 :** This routine calculates time equivalent factor TeQ = exp(-T1rec_s32 * dT_s32) **EFX062:** Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit	

⌋ ( )

8.5.3.3.4 Calculate an approximate time equivalent Value

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

[EFX450] ⌈

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:	**EFX451 :** This routine calculates time equivalent factor TeQApp = 1 - (T1rec_s32 * dT_s32) TeQApp is factorised by 2^16 **EFX452:** Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit	

⌋ ( )

8.5.3.3.5 Get ‘PT1’ output

This routine can be realised using inline function.

[EFX063] ⌈

Service name:	Efx_PT1Out_<OutTypeMn>	

Syntax:	`<OutType> Efx_PT1Out_<OutTypeMn>(`
	`  const Efx_StatePT1_Type* const State_cpst`
	`)`	

Service ID[hex]:	0x2D to 0x2E	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Constant 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. **EFX064 :** Output value = State_cpst->Y1_s32 >> 16 **EFX065 :** Output value shall be normalized by 16 bit right shift of internal state variable.	

	EFX066: Return value shall be limited by boundary values of the return data type.

⌋ ( )

[EFX067] ⌈
Here is the list of implemented functions.

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

⌋ ( )

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 8.5.3.3.3, shall be used for Efx_DT1_s32 function to calculate the time equivalent TeQ.

8.5.3.4.1 'DT1' Controller – Type1

[EFX070] ⌈

Service name:	Efx_DT1Typ1Calc	

Syntax:	`void Efx_DT1Typ1Calc(`
	`  sint32 X_s32,`
	`  Efx_StateDT1Typ1_Type* const 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:	**EFX071:** 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) This derives implementation : Output_value = (TeQ * State_cpst->Y1) + K_s32 * (1 - TeQ) * ((State_cpst->X1 - State_cpst->X2) / dT) where TeQ = exp(-dT/T1) **EFX072:**	

	Efx_CalcTeQ_s32 shall be used for calculation of time equivalent parameter TeQ_s32. **EFX073:** If (T1 = 0) then DT1 controller follows Input value, Output_value = k_s32 * (X_s32 - State_cpst->X1) / dT **EFX074:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **EFX075:** 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 **EFX076:** Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit

⌋ ( )

8.5.3.4.2 'DT1' Controller – Type2

[EFX501] ⌈

Service name:	Efx_DT1Typ2Calc	

Syntax:	`void Efx_DT1Typ2Calc(`
	`  sint32 X_s32,`
	`  Efx_StateDT1Typ2_Type* const 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:	**EFX502:** 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) This derives implementation : Output_value = (TeQ * State_cpst->Y1) + K_s32 * (1 - TeQ) * ((X_s32 - State_cpst->X1) / dT) where TeQ = exp(-dT/T1)	

	EFX503: Efx_CalcTeQ_s32 shall be used for calculation of time equivalent parameter TeQ_s32. **EFX504:** If (T1 = 0) then DT1 controller follows Input value, Output_value = k_s32 * (X_s32 - State_cpst->X1) / dT **EFX505:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **EFX506:** Current input value X_s32 shall be stored to State_cpst->X1. State_cpst->X1 = X_s32 **EFX507:** 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.

[EFX077] ⌈

Service name:	Efx_DT1Typ1SetState	

Syntax:	`void Efx_DT1Typ1SetState(`
	`  Efx_StateDT1Typ1_Type* const 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. **EFX078 :** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_s16 << 16 **EFX079 :** The internal state of the DT1 element is stored as (Y1_s16 << 16) **EFX080:** 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.

[EFX510] ⌈

Service name:	Efx_DT1Typ2SetState	

Syntax:	`void Efx_DT1Typ2SetState(`
	`  Efx_StateDT1Typ2_Type* const 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. **EFX511 :** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_s16 << 16 **EFX512 :** The internal state of the DT1 element is stored as (Y1_s16 << 16) **EFX513:** 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.

[EFX081] ⌈

Service name:	Efx_DT1Typ1Out_<OutTypeMn>	

Syntax:	`<OutType> Efx_DT1Typ1Out_<OutTypeMn>(`
	`  const Efx_StateDT1Typ1_Type* const 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. **EFX082:** Output value = State_cpst->Y1 >> 16 **EFX083:** Output value shall be normalized by 16 bit right shift of internal state variable. **EFX084:** Return value shall be limited by boundary values of the return data type.	

⌋ ( )

[EFX085] ⌈
Here is the list of implemented functions.

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

⌋ ( )

8.5.3.4.6 Get 'DT1' output – Type2

This routine can be realised using inline function.

[EFX515] ⌈

Service name:	Efx_DT1Typ2Out_<OutTypeMn>	

Syntax:	`<OutType> Efx_DT1Typ2Out_<OutTypeMn>(`
	`  const Efx_StateDT1Typ2_Type* const 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. **EFX516:** Output value = State_cpst->Y1 >> 16 **EFX517:** Output value shall be normalized by 16 bit right shift of internal state variable. **EFX518:** Return value shall be limited by boundary values of the return data type.	

⌋ ( )

[EFX519] ⌈
Here is the list of implemented functions.

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

0x35	sint16 Efx_DT1Typ2Out_s16(const Efx_StateDT1Typ2_Type * const)
0x36	sint8 Efx_DT1Typ2Out_s8(const Efx_StateDT1Typ2_Type * const)

⌋ ( )

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

[EFX090] ⌈

Service name:	Efx_PDCalc	

Syntax:	`void Efx_PDCalc(`
	`  sint32 X_s32,`
	`  Efx_StatePD_Type* const State_cpst,`
	`  const Efx_ParamPD_Type* const 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:	**EFX091 :** This routine computes proportional plus derivative controller output value using differential equation: Yn= K(1+Tv/dT) * Xn - K(Tv/dT) * Xn-1 This 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) **EFX092:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **EFX093:** Current input value X_s32 shall be stored to State_cpst->X1. State_cpst->X1 = X_s32 **EFX094 :** 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.

[EFX095] ⌈

Service name:	Efx_PDSetState	

Syntax:	`void Efx_PDSetState(`
	`  Efx_StatePD_Type* const 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. **EFX096:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_s16 << 16 **EFX097:** The internal state of the PD element is stored as (Y1_s16 << 16) **EFX098:** 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.

[EFX100] ⌈

Service name:	Efx_PDSetParam	

Syntax:	`void Efx_PDSetParam(`
	`  Efx_ParamPD_Type* const 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:	**EFX101:** The routine sets the parameter structure of a PD element. Initialisation of amplification factor. Param_cpst->K_C = K_s32 **EFX102:** 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.

[EFX103] ⌈

Service name:	Efx_PDOut_<OutTypeMn>	

Syntax:	`<OutType> Efx_PDOut_<OutTypeMn>(`
	`  const Efx_StatePD_Type* const State_cpcst`
	`)`	

Service ID[hex]:	0x3D to 0x3E	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpcst	Constant 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. **EFX104 :** Output value = State_cpst->Y1 >> 16 **EFX105 :** Output value shall be normalized by 16 bit right shift of internal state variable. **EFX106:** Return value shall be limited by boundary values of the return data type.	

⌋ ( )

[EFX107] ⌈
Here is the list of implemented functions.

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

⌋ ( )

8.5.3.6 Integral component

This routine calculates Integration element .

8.5.3.6.1 'I' Controller

[EFX110] ⌈

Service name:	Efx_ICalc	

Syntax:	`void Efx_ICalc(`
	`  sint32 X_s32,`
	`  Efx_StateI_Type* const 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:	**EFX111:** This routine computes 'I' controller output value using differential equation, Yn = Yn-1 + K * dT * Xn-1 This derives implementation : Output_value = State_cpst->Y1 + K_s32 * dT_s32 * State_cpst->X1 **EFX112:** 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 **EFX113:** Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit	

⌋ ( )

8.5.3.6.2 'I' Controller with limitation

[EFX455] ⌈

Service name:	Efx_ILimCalc

Syntax:	`void Efx_ILimCalc(`
	`  sint32 X_s32,`
	`  Efx_StateI_Type* const State_cpst,`
	`  sint32 K_s32,`
	`  const Efx_Limits_Type* const 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:	**EFX456:**
	This routine computes DT1 controller output value using differential equation,
	Yn = Yn-1 + K * dT * Xn-1
	
	This derives implementation :
	Output_value = State_cpst->Y1 + K_s32 * dT_s32 * State_cpst->X1
	
	EFX457:
	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
	
	EFX458:
	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
	
	EFX459:
	Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit	

⌋ ( )

8.5.3.6.3 Set limits for controllers

[EFX460] ⌈

Service name:	Efx_CtrlSetLimit	

Syntax:	`void Efx_CtrlSetLimit(`
	`  sint32 Min_s32,`
	`  sint32 Max_s32,`
	`  Efx_Limits_Type* const Limit_cpst`
	`)`	

Service ID[hex]:	0x42	
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:	void	No return value

Description:	**EFX461:** Update limit structure Limit_cpst->Min_C = Min_s32 Limit_cpst->Max_C = Max_s32

⌋ ( )
Note : "This routine (Efx_CtrlSetLimit) is depreciated and will not be supported in future release.
Replacement routine : Efx_CtrlSetLimits "

[EFX523] ⌈

Service name:	Efx_CtrlSetLimits	

Syntax:	`void Efx_CtrlSetLimits(`
	`  Efx_Limits_Type* const 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:	**EFX524:** 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.
[EFX114] ⌈

Service name:	Efx_ISetState	

Syntax:	`void Efx_ISetState(`
	`  Efx_StateI_Type* const 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	None	

(inout):		
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. **EFX115:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_s16 << 16 **EFX116:** The internal state of the DT1 element is stored as (Y1_s16 << 16) **EFX117:** 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.

[EFX118] ⌈

Service name:	Efx_IOut_<OutTypeMn>	

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

Service ID[hex]:	0x43 to 0x44	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Constant 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. **EFX119:** Output value = State_cpst->Y1 >> 16 **EFX120:** Output value shall be normalized by 16 bit right shift of internal state variable. **EFX121:** Return value shall be limited by boundary values of the return data type.	

⌋ ( )

[EFX122] ⌈

Here is the list of implemented functions.

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

⌋ ( )

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)

[EFX125] ⌈

Service name:	Efx_PITyp1Calc	

Syntax:	`void Efx_PITyp1Calc(`
	`  sint32 X_s32,`
	`  Efx_StatePI_Type* const State_cpst,`
	`  const Efx_ParamPI_Type* const 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:	**EFX126:** 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 This 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) **EFX127:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **EFX128:** Current input value X_s32 shall be stored to State_cpst->X1. State_cpst->X1 = X_s32 **EFX129:** 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)

[EFX465] ⌈

Service name:	Efx_PITyp1LimCalc	

Syntax:	`void Efx_PITyp1LimCalc(`
	`  sint32 X_s32,`
	`  Efx_StatePI_Type* const State_cpst,`
	`  const Efx_ParamPI_Type* const Param_cpst,`
	`  const Efx_Limits_Type* const 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:	**EFX466:**
	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
	
	This 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)
	
	EFX467:
	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
	
	EFX468:
	Calculated Output_value shall be stored to State_cpst->Y1.
	State_cpst->Y1 = Output_value
	
	EFX469:
	Current input value X_s32 shall be stored to State_cpst->X1.
	State_cpst->X1 = X_s32
	
	EFX470:
	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)

[EFX130] ⌈

Service name:	Efx_PITyp2Calc	

Syntax:	`void Efx_PITyp2Calc(`
	`  sint32 X_s32,`
	`  Efx_StatePI_Type* const State_cpst,`
	`  const Efx_ParamPI_Type* const 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:	**EFX131 :** 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 This 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) **EFX132:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **EFX133:** Current input value X_s32 shall be stored to State_cpst->X1. State_cpst->X1 = X_s32 **EFX134 :** 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)

[EFX475] ⌈

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:	**EFX476 :**
	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
	
	This 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)
	
	EFX477:
	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
	
	EFX478:
	Calculated Output_value shall be stored to State_cpst->Y1.
	State_cpst->Y1 = Output_value
	
	EFX479:
	Current input value X_s32 shall be stored to State_cpst->X1.
	State_cpst->X1 = X_s32
	
	EFX480 :
	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.

[EFX135] ⌈

Service name:	Efx_PISetState	

Syntax:	`void Efx_PISetState(`
	`  Efx_StatePI_Type* const 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	None	

(inout):		
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. **EFX136:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_s16 << 16 **EFX137 :** The internal state of the PD element is stored as (Y1_s16 << 16) **EFX138 :** 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.

[EFX139] ⌈

Service name:	Efx_PISetParam	

Syntax:	`void Efx_PISetParam(`
	`  Efx_ParamPI_Type* const 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:	**EFX140:** The routine sets the parameter structure of a PI element. Initialisation of amplification factor. Param_cpst->K_C = K_s32 **EFX141:** 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.

[EFX142] ⌈

Service name:	Efx_PIOut_<OutTypeMn>	

Syntax:	`<OutType> Efx_PIOut_<OutTypeMn>(`
	`  const Efx_StatePI_Type* const State_cpst`
	`)`	

Service ID[hex]:	0x49 to 0x4A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Constant 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. **EFX143:** Output value = State_cpst->Y1 >> 16 **EFX144:** Output value shall be normalized by 16 bit right shift of internal state variable. **EFX145:** Return value shall be limited by boundary values of the return data type.	

⌋ ( )

[EFX146] ⌈
Here is the list of implemented functions.

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

⌋ ( )

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)

[EFX150] ⌈

Service name:	Efx_PIDTyp1Calc

Syntax:	`void Efx_PIDTyp1Calc(`
	`  sint32 X_s32,`
	`  Efx_StatePID_Type* const State_cpst,`
	`  const Efx_ParamPID_Type* const 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	Constant pointer to the internal state structure.
Parameters (out):	None	
Return value:	void	No return value
Description:	**EFX151:** 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 This 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 **EFX152:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **EFX153:** 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 **EFX154:** 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)

[EFX485] ⌈

Service name:	Efx_PIDTyp1LimCalc	

Syntax:	`void Efx_PIDTyp1LimCalc(`
	`  sint32 X_s32,`
	`  Efx_StatePID_Type* const State_cpst,`
	`  const Efx_ParamPID_Type* const Param_cpst,`
	`  const Efx_Limits_Type* const 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	Constant Pointer to the internal state structure.
Parameters (out):	None	
Return value:	void	No return value

Description:	**EFX486:**
	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
	
	This 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
	
	EFX487:
	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
	
	EFX488:
	Calculated Output_value shall be stored to State_cpst->Y1.
	State_cpst->Y1 = Output_value
	
	EFX489:
	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
	
	EFX490:
	Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit	

⌋ ( )

8.5.3.8.3 'PID' Controller – Type2

[EFX155] ⌈

Service name:	Efx_PIDTyp2Calc

Syntax:	`void Efx_PIDTyp2Calc(`
	`  sint32 X_s32,`
	`  Efx_StatePID_Type* const State_cpst,`
	`  const Efx_ParamPID_Type* const 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:	**EFX156:** 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 This 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 **EFX157:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **EFX158:** 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 **EFX159:** 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

[EFX495] ⌈

Service name:	Efx_PIDTyp2LimCalc	

Syntax:	`void Efx_PIDTyp2LimCalc(`
	`  sint32 X_s32,`
	`  Efx_StatePID_Type* const State_cpst,`
	`  const Efx_ParamPID_Type* const Param_cpst,`
	`  const Efx_Limits_Type* const 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:	**EFX496:**
	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
	
	This 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
	
	EFX497:
	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
	
	EFX498:
	Calculated Output_value shall be stored to State_cpst->Y1.
	State_cpst->Y1 = Output_value
	
	EFX499:
	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
	
	EFX500:
	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.

[EFX160] ⌈

Service name:	Efx_PIDSetState

Syntax:	`void Efx_PIDSetState(`
	`  Efx_StatePID_Type* const 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	Constant pointer to internal state structure
Return value:	void	No return value
Description:	The routine initialises internal state variables of a PID element. **EFX161:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_s16 << 16 **EFX162:** The internal state of the PD element is stored as (Y1_s16 << 16) **EFX163:** 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.

[EFX164] ⌈

Service name:	Efx_PIDSetParam	

Syntax:	`void Efx_PIDSetParam(`
	`  Efx_ParamPID_Type* const 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	Constant pointer to internal parameter structure
Return value:	void	No return value
Description:	**EFX165:** The routine sets the parameter structure of a PID element. Initialisation of amplification factor. Param_cpst->K_C = K_s32 **EFX166:** Initialisation of lead time state variable Param_cpst->Tv_C = Tv_s32	

	EFX167: 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.

[EFX168] ⌈

Service name:	Efx_PIDOut_<OutTypeMn>	

Syntax:	`<OutType> Efx_PIDOut_<OutTypeMn>(`
	`  const Efx_StatePID_Type* const State_cpst`
	`)`	

Service ID[hex]:	0x50 to 0x51	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Constant 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. **EFX169:** Output value = State_cpst->Y1 >> 16 **EFX170:** Output value shall be normalized by 16 bit right shift of internal state variable. **EFX171:** Return value shall be limited by boundary values of the return data type.	

⌋ ( )

[EFX172] ⌈
Here is the list of implemented functions.

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

⌋ ( )

8.5.4 Square root

[EFX175] ⌈

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:	**EFX176:** This service computes the square root of a value Result = square_root (x_value) **EFX177:** The result is rounded off.	

⌋ ( )

[EFX178] ⌈

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:	**EFX179:** This service computes the square root of a value Result = square_root (x_value) **EFX180:** The result is rounded off.	

⌋ ( )

[EFX181] ⌈

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:	**EFX182:** This service computes the square root of a value Result = square_root (x_value) **EFX183:** The result is rounded off.	

⌋ ( )

8.5.5 Exponential

[EFX185] ⌈

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. **EFX186:** Output = e^{-x} where x = Value1 **EFX187:** Output is quantized by 2^16 Output Range = ([0.0183...1.0] * 2^16) = [1200...65535] Input Range = ([0...4] * 2^16) = [0x0...0x40000]	

⌋ ( )

8.5.6 Average

[EFX190] ⌈

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:	**EFX191:** The routine returns average value. Output = (Value1 + Value2) / 2 **EFX192:** The result is rounded off.	

⌋ ( )

8.5.7 Array Average

[EFX193] ⌈

Service name:	Efx_Array_Average_<InTypeMn>_<OutTypeMn>	

Syntax:	`<OutType> Efx_Array_Average_<InTypeMn>_<OutTypeMn>(`
	`  <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:	**EFX194:** The routine returns average value of an array. Output = (Array[0] + Array[1] + ... + Array[N-1]) / Count **EFX195:** The result is rounded off.	

⌋ ( )

[EFX196] ⌈
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

[EFX197] ⌈

Service name:	Efx_MovingAverage_<InTypeMn>_<OutTypeMn>	
Syntax:	`<OutType> Efx_MovingAverage_<InTypeMn>_<OutTypeMn>( Efx_MovingAvrg<InTypeMn>_Type* const 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:	**EFX198 :** struct ->p_beg pointer holds start address of an array struct ->p_end pointer holds end address of an array struct ->p_act pointer holds address of an oldest entry of an array **EFX199 :** struct ->sum shall store total sum including 'value' & excluding oldest entry struct ->sum = struct ->sum - *(struct ->p_act) + value **EFX200:** In every routine call struct ->p_act shall be incremented with wrap around. This increment ensures that oldest entry gets replaced with new entry. **EFX201:** The routine returns sliding average value of n - 1 last subsequent values of an array plus one new value. Array values are accessed by pointer *(struct->p_act). Output_value = struct->sum / struct->n **EFX202:** If struct ->n = 0 the result shall be zero by definition. **EFX203:** The result is rounded off.	

⌋ ( )

Structure definition for function argument

[EFX204] ⌈

Name:	`Efx_MovingAvrgS16_Type`		
Type:	`Structure`		
Element:	`sint32`	`sum`	Sum of array elements
	`sint32`	`n`	Size of an array
	`sint16`	`*p_beg`	Pointer to the first array element
	`sint16`	`*p_end`	Pointer to the last array element
	`sint16`	`*p_act`	Pointer to the oldest entry array element
Description:	Structure definition for sliding average routine for sint16 input value		

Name:	`Efx_MovingAvrgS32_Type`		
Type:	`Structure`		
Element:	`sint64`	`sum`	Sum of array elements
	`sint32`	`n`	Size of an array
	`sint32`	`*p_beg`	Pointer to the first array element
	`sint32`	`*p_end`	Pointer to the last array element
	`sint32`	`*p_act`	Pointer to the oldest entry array element
Description:	Structure definition for sliding average routine for sint32 input value		

⌋ ( )

[EFX205] ⌈
Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0x6A	sint16 Efx_Moving_Average_s16_s16(Efx_MovingAvrgS16_Type* const, sint16)
0x6B	sint32 Efx_Moving_Average_s32_s32(Efx_MovingAvrgS32_Type* const , sint32)

⌋ ( )

8.5.9 Hypotenuse

Warning: Hypotenuse functions shall not be used directly for distance computation because the result has not the same resolution than the inputs.

[EFX210] ⌈

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:	**EFX211:** This service computes the length of a vector : Result = sqrt(x_value * x_value + y_value * y_value) / sqrt(2) **EFX212:** The result is rounded off.	

⌋ ( )

[EFX213] ⌈

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:	**EFX214:** This service computes the length of a vector : Result = sqrt (x_value*x_value + y_value*y_value) / sqrt(2) **EFX215:** The result is rounded off.	

⌋ ( )

[EFX216] ⌈

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:	**EFX217:** This service computes the length of a vector : Result = sqrt (x_value * x_value + y_value * y_value) / sqrt(2) **EFX218:**	

	The result is rounded off.

⌋ ( )

8.5.10 Trigonometric functions

8.5.10.1 Sine function

[EFX220] ⌈

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: $1/((2^{31})-1)$
Description:	**EFX221:** This service computes the sine of an angle. **EFX222:** The result is rounded off.	

⌋ ( )

[EFX223] ⌈

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: $1/((2^{15})-1)$

Description:	**EFX224:** This service computes the sine of an angle. **EFX225:** The result is rounded off.

⌋ ( )

[EFX226] ⌈

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)-1)
Description:	**EFX227:** This service computes the sine of an angle. **EFX228:** The result is rounded off.	

⌋ ( )

8.5.10.2 Cosine function

[EFX229] ⌈

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: $1/((2^{31})-1)$
Description:	**EFX230:** This service computes the cosine of an angle.	

	EFX231: The result is rounded off.

⌋ ( )

[EFX232] ⌈

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: $1/((2^{15})-1)$
Description:	**EFX233:** This service computes the cosine of an angle. **EFX234:** The result is rounded off.	

⌋ ( )

[EFX235] ⌈

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: $1/((2^{7})-1)$
Description:	**EFX236:** This service computes the cosine of an angle. **EFX237:** The result is rounded off.	

⌋ ( )

8.5.10.3 Inverse Sine function

[EFX240] ⌈

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: $2*PI/((2^{31})-1)$
Parameters (inout):	None	
Parameters (out):	None	
Return value:	sint32	Return value of the function Physical range: [-PI/2 , PI/2] Resolution: $1/2^{32}$
Description:	**EFX241:** This service computes the inverse sine of a value. **EFX242:** The result is rounded off.	

⌋ ( )

[EFX243] ⌈

Service name:	Efx_Arcsin_s16_s16	

Syntax:	`sint16 Efx_Arcsin_s16_s16(`
	`  sint16 x_value`
	`)`	

Service ID[hex]:	0x81	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	Argument Physical range: [-1, 1] Resolution: $2*PI/((2^{15})-1)$
Parameters (inout):	None	
Parameters (out):	None	
Return value:	sint16	Return value of the function Physical range: [-PI/2 , PI/2] Resolution: 1/2^16
Description:	**EFX244:** This service computes the inverse sine of a value. **EFX245:** The result is rounded off.	

⌋ ( )

[EFX246] ⌈

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: $2*PI/((2^7)-1)$
Parameters (inout):	None	
Parameters (out):	None	
Return value:	sint8	Return value of the function Physical range: [-PI/2, PI/2] Resolution: $1/2^8$
Description:	**EFX247:** This service computes the inverse sine of a value. **EFX248:** The result is rounded off.	

⌋ ( )

8.5.10.4 Inverse cosine function

[EFX250] ⌈

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: $2*PI/((2^{31})-1)$
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint32	Return value of the function Physical range: [0 , PI] Resolution: $1/2^{32}$
Description:	**EFX251:** This service computes the inverse cosine of a value. **EFX252:** The result is rounded off.	

⌋ ( )

[EFX253] ⌈

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: $2*PI/((2^{15})-1)$
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint16	Return value of the function Physical range: [0 , PI] Resolution: $1/2^{16}$
Description:	**EFX254:** This service computes the inverse cosine of a value. **EFX255:** The result is rounded off.	

⌋ ( )

[EFX256] ⌈

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: $2*PI/((2^{7})-1)$
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	Return value of the function Physical range: [0 , PI] Resolution: $1/2^{8}$
Description:	**EFX257:** This service computes the inverse cosine of a value. **EFX258:** The result is rounded off.	

⌋ ( )

8.5.11 Rate limiter

[EFX261] ⌈

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. **EFX262:** If *init==0, *output=input **EFX263:** If *output==0, output shall keep this NULL value **EFX264:** Input, limit_pos, limit_neg and output must have the same resolution and the same physical unit. **EFX265:** 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. **EFX266:** 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). **EFX267:** 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. **EFX268:** At first step, when *init==0, output takes the value of input and *init will be put at 1. **EFX269:** 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 **EFX270:** Values of limit_pos and limit_neg shall be adapted to the frequency of the call of the service.	

⌋ ( )

[EFX271] ⌈
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 (sint16, uint16, uint16, sint16 *, uint8 *)
0x8D	void Efx_SlewRate_u32 (uint32, uint32, uint32, uint32 *, uint8 *)
0x8E	void Efx_SlewRate_s32 (sint32, uint32, 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

[EFX275] ⌈

Name:	`Efx_ParamRamp_Type`		
Type:	`Structure`		
Element:	`uint32`	`SlopePos_u32`	Positive slope for ramp in absolute value
	`uint32`	`SlopeNeg_u32`	Negative slope for ramp in absolute value
Description:	Structure definition for Ramp routine		

Name:	`Efx_StateRamp_Type`		
Type:	`Structure`		
Element:	`sint32`	`State_s32`	State of the ramp
	`sint8`	`Dir_s8`	Ramp direction
	`sint8`	`Switch_s8`	Position of switch
Description:	Structure definition for Ramp routine		

⌋ ( )

8.5.12.1 Ramp routine

[EFX276] ⌈

Service name:	Efx_RampCalc

Syntax:	`void Efx_RampCalc(`
	`  sint32 X_s32,`
	`  Efx_StateRamp_Type* const State_cpst,`
	`  const Efx_ParamRamp_Type* const 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]
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 > Target) or (State_cpst->State_s32 < Target).
	
	EFX278:
	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)
	
	EFX279:
	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)
	
	EFX280:
	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.
	
	EFX281:
	Comparison of State and Target decides ramp direction
	If(State_cpst->State_s32 > Target) then State_cpst->Dir_s8 = RISING
	If(State_cpst->State_s32 < Target) then State_cpst->Dir_s8 = FALLING
	If(State_cpst->State_s32 == Target) then State_cpst->Dir_s8 = END
	
	EFX282:
	This routine returns State value
	Return_value = State_cpst->State_s32
	
	EFX283:
	Calculated ramp value shall be stored to State_cpst->State_s32 variable
	
	EFX284:
	Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit	

⌋ ( )

8.5.12.2 Ramp Initialisation

[EFX285] ⌈

Service name:	Efx_RampInitState

Syntax:	`void Efx_RampInitState(`
	`  Efx_StateRamp_Type* const 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. **EFX286:** 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 **EFX442:** Initialisation of state variable State_cpst->State_s32 = Val_s32 **EFX443:** 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

[EFX287] ⌈

Service name:	Efx_RampSetParam	

Syntax:	`void Efx_RampSetParam(`
	`  Efx_ParamRamp_Type* const 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_RampParam_Type. **EFX288:** 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

[EFX289] ⌈

Service name:	Efx_RampOut_s32	

Syntax:	`sint32 Efx_RampOut_s32(`
	`  const Efx_StateRamp_Type* const 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:	**EFX290**: Returns the internal state of the ramp element. Return Value = State_cpcst->State_s32	

⌋ ( )

8.5.12.5 Ramp Jump routine

[EFX291] ⌈

Service name:	Efx_RampCalcJump	

Syntax:	`void Efx_RampCalcJump(`
	`  sint32 X_s32,`
	`  Efx_StateRamp_Type* const 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. If ramp is still rising (or falling) and target value is not reached, and then input value of ramp jumps to a lower (or higher) value of current ramp state, ramp will jump to that value immediately. This functionality is helpful if input target value of ramp changes its direction often and significantly and ramp should reach target value faster than without that functionality. If the target is reached or the target does not change its direction, the standard behaviour of ramp functionality is untouched. **EFX292:** 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 = Target
	State_cpst->Dir_s8 = END
	
	EFX293:
	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
	
	EFX303:
	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.
	
	EFX304:
	Comparison of State and Target decides ramp direction
	If(State_cpst->State_s32 > Target) then State_cpst->Dir_s8 = RISING
	If(State_cpst->State_s32 < Target) then State_cpst->Dir_s8 = FALLING
	If(State_cpst->State_s32 == Target) then State_cpst->Dir_s8 = END
	
	EFX305:
	This routine returns State value.
	Return_value = State_cpst->State_s32
	
	EFX277:
	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

[EFX295] ⌈

Service name:	Efx_RampCalcSwitch_s32	

Syntax:	`sint32 Efx_RampCalcSwitch_s32(`
	`  sint32 Xa_s32,`
	`  sint32 Xb_s32,`
	`  Efx_StateRamp_Type* const State_cpst,`
	`  const Efx_ParamRamp_Type* const Param_cpcst,`
	`  sint32 dT_s32`
	`)`	

Service ID[hex]:	0x95	
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'
	Param_cpcst	Pointer to the parameter structure which contains the positive and negative slope of the ramp
	dT_s32	Sample Time [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout):	State_cpst	Pointer to actual value of the ramp
Parameters (out):	None	
Return value:	sint32	Returns the actual state of the ramp
Description:	This routine switches ramp between two target values based on the Switch value.	

	EFX296:
	Switch decides target to select.
	If (State_cpst->Switch_s8 == TARGET_A), target = Xa_s32
	If (State_cpst->Switch_s8 == TARGET_B), target = Xb_s32
	
	EFX297:
	State_cpst->Dir_s8 hold direction information
	Ramp direction status: RISING, FALLING, END
	
	EFX298:
	If ramp is active then ramp will change to reach selected target with defined slope
	
	if (State_cpst->Dir_s8 == RISING)
	then State_cpst->State_s32 = State_cpst->State_s32 + (Param_cpcst->SlopePos_u32 * dT_s32)
	else if (State_cpst->Dir_s8 == FALLING)
	then State_cpst->State_s32 = State_cpst->State_s32 - (Param_cpcst->SlopeNeg_u32 * dT_s32)
	else if (State_cpst->Dir_s8 == END)
	State_cpst->State_s32 = target value which is decided by State_cpst->Switch_s8.
	
	EFX299:
	Once ramp value reaches the selected target value, the ramp direction status is switched to END.
	State_cpst->Dir_s8 == END
	
	EFX300:
	If the ramp has reached its destination and no change of switch occurs, the output value follows the actual target value.
	If(State_cpst->State_s32 == target value)
	Return_value = Xa_s32 (if State_cpst->Switch_s8 is TARGET_A)
	Return_value = Xb_s32 (if State_cpst->Switch_s8 is TARGET_B)
	
	EFX301:
	Calculated ramp value shall be stored to State_cpst->State_s32 variable
	
	EFX302:
	Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit

Note : "This routine (Efx_RampCalcSwitch_s32) is depreciated and will not be supported in future release.
Replacement routine : Efx_RampCalcSwitch "

[EFX520] ⌈

Service name:	Efx_RampCalcSwitch

Syntax:	`sint32 Efx_RampCalcSwitch(`
	`  sint32 Xa_s32,`
	`  sint32 Xb_s32,`
	`  boolean Switch,`
	`  Efx_StateRamp_Type* const 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. **EFX521:** 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 **EFX522:** 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. **EFX528:** Efx_RampCalcSwitch routine has to be called before Efx_RampCalc	

⌋ ( )

8.5.12.7 Get Ramp Switch position

[EFX307] ⌈

Service name:	Efx_RampGetSwitchPos	

Syntax:	`boolean Efx_RampGetSwitchPos(`
	`  Efx_StateRamp_Type* const 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:	**EFX308:** Gets the current switch position of ramp switch function. Return value = TRUE if Switch position State_cpst->Switch_s8 = TARGET_A Return value = FALSE if Switch position State_cpst->Switch_s8 = TARGET_B	

⌋ ( )

8.5.12.8 Check Ramp Activity

[EFX309] ⌈

Service name:	Efx_RampCheckActivity	

Syntax:	`boolean Efx_RampCheckActivity(`
	`  Efx_StateRamp_Type* const 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:	**EFX310:** This routine checks the status of the ramp and returns TRUE if the ramp is active, otherwise it returns FALSE. 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

[EFX311] ⌈

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. **EFX312:** If Input ≤ thresholdLow, Then return_value = Out_LowThresholdVal **EFX313:** If Input ≥ thresholdHigh, Then return_value = Out_HighThresholdVal **EFX314:** If Out_LowThresholdVal < Input < Out_HighThresholdVal, Then return_value = Out_Val **EFX315:** Input, thresholdLow and thresholdHigh must have the same resolution and the same physical unit. **EFX316:** Return_value , Out_Val, Out_LowThresholdVal and Out_HighThresholdVal must have the same resolution and the same physical unit.

⌋ ( )

[EFX317] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0x9A	uint8 Efx_Hysteresis_u8_u8 (uint8, 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

[EFX320] ⌈

Service name:	Efx_HystCenterHalfDelta_<InTypeMn>_<OutTypeMn>	

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

Service ID[hex]:	0xA0 to 0xA1	
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. **EFX321:** 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) **EFX322:** Parameters X, center and halfDelta should have the same data type. **EFX323:** State variable shall store the old boolean result.

⌋ ( )

[EFX324] ⌈
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 *)

⌋ ( )

8.5.13.3 Hysteresis left right

[EFX325] ⌈

Service name:	Efx_HystLeftRight_<InTypeMn>_<OutTypeMn>	

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

Service ID[hex]:	0xA3 to 0xA4	
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. **EFX326:** Return value = TRUE, if X ≥ Rsp (right switching point) Return value = FALSE, if TRUE if X ≤ Lsp (left switching point) Return value is former state value if Lsp > X > Rsp **EFX327:** Parameters X, Lsp and Rsp should have the same data type. **EFX328:** State variable shall store the old boolean result. **EFX329:** Rsp shall be always greater than Lsp	

⌋ ( )

[EFX330] ⌈
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 *)

⌋ ( )

8.5.13.4 Hysteresis delta right

[EFX331] ⌈

Service name:	Efx_HystDeltaRight_<InTypeMn>_<OutTypeMn>	

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

Service ID[hex]:	0xA5 to 0xA6	
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 **EFX332:** 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 **EFX333:** Parameters X, Rsp and Delta should have the same data type. **EFX334:** State variable shall store the old boolean result.	

⌋ ( )

[EFX335] ⌈
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 *)

⌋ ( )

8.5.13.5 Hysteresis left delta

[EFX336] ⌈

Service name:	Efx_HystLeftDelta_<InTypeMn>_<OutTypeMn>	

Syntax:	`boolean Efx_HystLeftDelta_<InTypeMn>_<OutTypeMn>(`
	`  <InType> X,`
	`  <InType> Lsp,`
	`  <InType> Delta,`
	`  boolean* State`
	`)`	

Service ID[hex]:	0xA7 to 0xA8	
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. **EFX337:** 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) **EFX338:** Parameters X, Lsp and Delta should have the same data type. **EFX339:** State variable shall store the old boolean result.	

⌋ ( )

[EFX340] ⌈
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 *)

⌋ ( )

8.5.14 Efx_DeadTime

[EFX345] ⌈

Service name:	Efx_DeadTime_s16_s16

Syntax:	`sint16 Efx_DeadTime_s16_s16(`
	`  sint16 X,`
	`  sint32 DelayTime,`
	`  sint32 StepTime,`
	`  Efx_DeadTimeParam_Type* Param`
	`)`

Service ID[hex]:	0xAA

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X	Input value
	DelayTime	Time to be delayed
	StepTime	Sample time
Parameters (inout):	Param	Pointer to parameter structure of type Efx_DeadTimeParam_Type
Parameters (out):	None	
Return value:	sint16	Returns the actual state of the dead time element as sint16 value

Description:	**EFX346:**
	This routine returns input value with specified delay time. Data buffer stores input samples using ring buffer algorithm,
	
	EFX347:
	Buffer size shall be configured as per the delay time range requirement.
	Hence in case of high delays, data buffer size has to be increased to reproduce original input signal at output without loss of samples.
	
	EFX348:
	Data buffer shall be allocated to use this function and *Param structure elements shall be used to store pointers to this allocated buffer.
	
	EFX349:
	StepTime is the minimum sampling time which decides signal quality of delayed signal. Param->dsintStatic stores old pending time.
	TotalTime = Param->dsintStatic + DelayTime
	while(TotalTime > StepTime) then,
	activate ring buffer with storing input value X.
	TotalTime = TotalTime - StepTime
	
	EFX350:
	Actual data pointer shall be checked for buffer size and shall be wrapped.
	If(Param->lszStatic ≤ Param->dtbufBegStatic) then,
	Param->lszStatic = Param-> dtbufEndStatic
	
	EFX351:
	Store current pointer position to Param->lszStatic.
	Store the remaining TotalTime to Param-> dsintStatic
	
	EFX352:
	Param->*dtbufBegStatic and Param->*dtbufEndStatic shall be initialised with start address and end address of data buffer respectively.
	
	EFX353:
	This routine returns present Param->*lszStatic value	

⌋ ( )

Structure definition for function argument

[EFX354] ⌈

Name:	`Efx_DeadTimeParam_Type`		
Type:	`Structure`		
Element:	`sint32`	`dsintStatic`	Time since the last pack was written
	`sint16`	`*lszStatic`	Pointer to actual buffer position
	`sint16`	`*dtbufBegStatic`	Pointer to begin of buffer
	`sint16`	`*dtbufEndStatic`	Pointer to end of buffer

Description:	Structure definition for Dead Time routine

⌋ ( )

8.5.15 Debounce routines

8.5.15.1 Efx_Debounce

[EFX355] ⌈

Service name:	Efx_Debounce_u8_u8	

Syntax:	`uint8 Efx_Debounce_u8_u8(`
	`  uint8 X,`
	`  Efx_DebounceState_Type * State,`
	`  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 [10^{-6}seconds per increment of 1 data representation unit]
Parameters (inout):	State	Pointer to state structure of type Efx_DebounceState_Type
Parameters (out):	None	
Return value:	uint8	Returns the debounced input value

Description:	**EFX356:**
	This routine debounces a digital input signal and returns the state of the signal as a boolean value.
	If(X != State->XOld) then check start debouncing.
	
	EFX357:
	If transition is from Low to High, then use Param->TimeLowHigh as debouncing time otherwise use Param->TimeHighLow
	
	EFX358 :
	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)
	State->XOld = X, and stop the timer, State->Timer = 0
	where TimePeriod = Param->TimeLowHigh or Param->TimeHighLow
	
	EFX359:
	Old value shall be returned as a output value. Current input is stored to old state.
	Return value = State->XOld
	State->XOld = X
	
	EFX360:
	Resolution of dT_s32 is 10^{-6} seconds per increment of 1 data representation unit	

⌋ ( )
Structure definition for function argument

[EFX361] ⌈

Name:	`Efx_DebounceParam_Type`		
Type:	`Structure`		
Element:	`sint16`	`TimeHighLow`	Time for a High to Low transition, given in 10ms steps
	`sint16`	`TimeLowHigh`	Time for a Low to High transition, given in 10ms steps
Description:	Structure definition for Debounce routine		

Name:	`Efx_DebounceState_Type`		
Type:	`Structure`		
Element:	`uint8`	`XOld`	Old input value from last call
	`sint32`	`Timer`	Timer for internal state
Description:	Structure definition for Debounce routine		

⌋ ( )

8.5.15.2 Efx_DebounceInit

[EFX362] ⌈

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:	**EFX363:** This routine call shall stop the debouncing timer. State->Timer = 0 **EFX364:** Sets the input state to the given init value. State->XOld = X;	

⌋ ( )

8.5.15.3 Efx_DebounceSetparam

[EFX365] ⌈

Service name:	Efx_DebounceSetParam	

Syntax:	`void Efx_DebounceSetParam(`
	`  Efx_DebounceParam_Type * Param,`
	`  sint32 THighLow,`
	`  sint32 TLowHigh`
	`)`	

Service ID[hex]:	0xB2	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	THighLow	Value for TimeHighLow of Efx_DebounceParam_Type
	TLowHigh	Value for TimeHighLow 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:	**EFX366:** This routine sets timing parameters, time for high to low transition and time for low to high for debouncing. Param-> TimeHighLow = THighLow Param-> TimeLowHigh = TLowHigh	

⌋ ( )

8.5.16 Ascending Sort Routine

[EFX370] ⌈

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:	**EFX371:** The sorting algorithm modifies the given input array and rearranges data in ascending order. 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 : uint16 Array [5] = [-42, -10, 88, 8, 15] Result : Array will be sorted to [-42, -10, 8, 15, 88]	

⌋ ( )

[EFX372] ⌈

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)

⌋ ( )

8.5.17 Descending Sort Routine

[EFX373] ⌈

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:	**EFX374:** The sorting algorithm modifies the given input array and rearranges data in descending order. 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 : uint16 Array [5] = [-42, -10, 88, 8, 15] Result : Array will be sorted to [88, 15, 8, -10, -42]	

⌋ ( )

[EFX375] ⌈

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)

⌋ ( )

8.5.18 Median sort routine

[EFX376] ⌈

Service name:	Efx_MedianSort_<InTypeMn>_<OutTypeMn>	

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

Service ID[hex]:	0xC0 to 0xC4, 0xC8	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Array	Pointer to an array
	N	Size of an array
Parameters (inout):	None	
Parameters (out):	None	
Return value:	<OutType>	Return value of the function
Description:	**EFX377:** This routine sorts values of an array in ascending order. Input array passed by the pointer shall have sorted values after this routine call. Input array [5] = [42, 10, 88, 8, 15] Sorted array[5] = [8, 10, 15, 42, 88] **EFX378:** Returns the median value of sorted array in case of N is even. Result = (Sorted_array[N/2] + Sorted_array[(N/2) - 1]) / 2 Eg. Sorted_array[4] = [8, 10, 15, 42] Result = (15 + 10) / 2 = 12 **EFX440:** Returns the median value of sorted array in case of N is odd. Return_Value = Sorted_array [N/2] = 15 Eg. Sorted_array[5] = [8, 10, 15, 42, 88] Result = 15 **EFX441:** In above calculation, N/2 shall be rounded off towards 0.	

⌋ ( )

[EFX379] ⌈

Here is the list of implemented functions.

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

⌋ ( )

8.5.19 Edge detection routines

8.5.19.1 Edge bipol detection

[EFX380] ⌈

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:	**EFX381:**
	This routine detects whether a signal has changed since the last call and returns TRUE. If signal has not changed then returns FALSE.
	if (Inp_Val != Old_Val)
	return value = TRUE
	else
	return value = FALSE.	

⌋ ( )

8.5.19.2 Edge falling detection

[EFX382] ⌈

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:	**EFX383:** 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 Return value = TRUE, If (*Old_Val == TRUE && Inp_Val == FALSE) Return value = FALSE, otherwise.	

⌋ ( )

8.5.19.3 Edge rising detection

[EFX384] ⌈

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:	**EFX385:** 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 Return value = TRUE, If (*Old_Val == FALSE && Inp_Val == TRUE) Return value = FALSE, otherwise.	

⌋ ( )

8.5.20 Interval routines

8.5.20.1 Interval Closed

[EFX386] ⌈

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:	**EFX387:** This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively. Return value = TRUE, if (MinVal ≤ InpVal ≤ MaxVal) Return value = FALSE, otherwise.

⌋ ( )

[EFX388] ⌈
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)

⌋ ( )

8.5.20.2 Interval Open

[EFX390] ⌈

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:	**EFX391:** This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively. Return value = TRUE, if (MinVal < InpVal < MaxVal) Return value = FALSE, otherwise.	

⌋ ( )

[EFX392] ⌈
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)

⌋ ( )

8.5.20.3 Interval Left Open

[EFX393] ⌈

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:	**EFX394:** This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively. Return value = TRUE, if (MinVal < InpVal ≤ MaxVal) Return value = FALSE, otherwise.	

⌋ ( )

[EFX395] ⌈
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)

⌋ ( )

8.5.20.4 Interval Right Open

[EFX396] ⌈

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:	**EFX397:** This routine compares a value 'InpVal' with lower and upper limit 'MinVal' and 'MaxVal' respectively. Return value = TRUE, if (MinVal ≤ InpVal < MaxVal) Return value = FALSE, otherwise.	

⌋ ( )

[EFX398] ⌈
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)

⌋ ( )

8.5.21 Counter routines

[EFX399] ⌈

Service name:	Efx_CounterSet_<InTypeMn>	

Syntax:	`void Efx_CounterSet_<InTypeMn>(`
	`  <InType> * const 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;	

⌋ ( )

[EFX404] ⌈
Here is the list of implemented functions.

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

⌋ ( )

[EFX400] ⌈

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. **EFX401:** The return value is the new value of the parameter CounterVal. * CounterVal ++; Return value = *CounterVal; **EFX402:** 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	

⌋ ( )

[EFX403] ⌈
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 *)

⌋ ( )

8.5.22 Flip-Flop routine

[EFX405] ⌈

Service name:	Efx_RSFlipFlop_<InTypeMn>_<OutTypeMn>	

Syntax:	`boolean Efx_RSFlipFlop_<InTypeMn>_<OutTypeMn>(`
	`  boolean R_Val,`
	`  boolean S_Val,`
	`  boolean* State_Val`
	)	

Service ID[hex]:	0xDA	
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. **EFX406:** The reset switch is higher prior than the set switch,	

	e.g. R_Val = TRUE, S_Val = TRUE Then state and return value = FALSE **EFX407:** Reset condition : R_Val = TRUE, S_Val = FALSE Then state and return value = FALSE **EFX408:** Set condition : R_Val = FALSE, S_Val = TRUE Then state and return value = TRUE **EFX409:** Invalid condition : R_Val = FALSE, S_Val = FALSE Then state and return value are unchanged

⌋ ( )

8.5.23 Limiter routines

[EFX410] ⌈

Service name:	Efx_TypeLimiter_<InTypeMn>_<OutTypeMn>	

Syntax:	`<OutType> Efx_TypeLimiter_<InTypeMn>_<OutTypeMn>(`
	`  <InType> Input_Val`
	`)`	

Service ID[hex]:	0xE0 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:	**EFX411:** 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.	

⌋ ( )

[EFX412] ⌈

Here is the list of implemented functions.

Service ID[hex]	***Syntax***
0xE0	uint8 Efx_TypeLimiter_s32_u8 (sint32)
0xE1	uint16 Efx_TypeLimiter_s32_u16 (sint32)
0xE2	uint32 Efx_TypeLimiter_s32_u32 (sint32)
0xE3	sint8 Efx_TypeLimiter_s32_s8 (sint32)
0xE4	sint16 Efx_TypeLimiter_s32_s16 (sint32)
0xE5	uint8 Efx_TypeLimiter_u32_u8 (uint32)
0xE6	uint16 Efx_TypeLimiter_u32_u16 (uint32)
0xE7	sint32 Efx_TypeLimiter_u32_s32 (uint32)

0xE8	sint8 Efx_TypeLimiter_u32_s8 (uint32)
0xE9	sint16 Efx_TypeLimiter_u32_s16 (uint32)

⌋ ( )

8.5.24 64 bits functions

8.5.24.1 General requirements

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

[EFX415] ⌈
C operators shall not be used for 64bit data (cast, arithmetic operators and comparison operators) ⌋ ( )

[EFX416] ⌈
64bit constants shall not be used. ⌋ ( )

[EFX417] ⌈
Direct affectation to and from a 64 bit type shall only be used through predefined functions of 64 bits library. ⌋ ( )

[EFX418] ⌈
Only the sint64 type is allowed (uint64 shall not be used). ⌋ ( )

[EFX419] ⌈
64bit functions do not perform saturation, even for the conversion to smaller types. ⌋ ( )

8.5.24.2 Casts

[EFX420] ⌈

Service name:	Efx_Cast_<InTypeMn>_<OutTypeMn>	

Syntax:	`<OutType> Efx_Cast_<InTypeMn>_<OutTypeMn>(`
	`  <InType> x_value`
	`)`	

Service ID[hex]:	0xEA to 0xEC	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	Argument of the function
Parameters (inout):	None	
Parameters (out):	None	

Return value:	<OutType>	Return value of the function
Description:	**EFX421:** Convert value of entry type in the value in the output type	

⌋ ( )

[EFX422] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0xEA	sint64 Efx_Cast_u32_s64(uint32)
0xEB	uint32 Efx_Cast_s64_u32(sint64)
0xEC	sint32 Efx_Cast_s64_s32(sint64)

⌋ ( )

8.5.24.3 Additions

[EFX423] ⌈

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:	**EFX424:** This service makes an addition between the two arguments : Return value = x_value + y_value The addition is not protected against the overflow.	

⌋ ( )

[EFX425] ⌈
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)

⌋ ( )

8.5.24.4 Multiplications

[EFX426] ⌈

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:	**EFX427:** This service makes a multiplication between the two arguments : Return value = x_value * y_value The multiplication is not protected against the overflow.	

⌋ ( )

[EFX428] ⌈
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)

⌋ ( )

8.5.24.5 Division

[EFX429] ⌈

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:	**EFX430:** These services make a division between the two arguments : Return value = x_value / y_value	

	EFX431: 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 **EFX432:** If y_value==0 then function returns the maximum value of the output type **EFX433:** The result is rounded towards 0.

⌋ ( )

[EFX434] ⌈
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_s64u32_s32 (sint64, uint32)
0xF8	uint32 Efx_Div_s64s32_u32 (sint64, sint32)
0xFA	sint32 Efx_Div_s64u32_s32 (sint64, uint32)
0xFB	uint32 Efx_Div_s64s32_u32 (sint64, sint32)

⌋ ( )

8.5.24.6 Comparison

[EFX436] ⌈

Service name:	Efx_Gt_<InTypeMn><InTypeMn>_<OutTypeMn>	

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

Service ID[hex]:	0xFC to 0xFD	
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:	**EFX437:** This service makes a comparison between the two arguments : Return value = (x_value > y_value)	

⌋ ( )

[EFX438] ⌈
Here is the list of implemented functions.

Service ID[hex]	*Syntax*
0xFC	sint64 Efx_Gt_s64u32_s64(sint64, uint32)
0xFD	sint64 Efx_Gt_s64s32_s64(sint64, uint32)

⌋ ( )

8.6 Examples of use of functions

None

8.7 Version API

8.7.1 Efx_GetVersionInfo

[EFX815] ⌈

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.	

The version information of a BSW module generally contains:
Module Id
Vendor Id

Vendor specific version numbers (BSW00407). ⌋ (BSW00407, BSW003, BSW00318, BSW00321)

[EFX816] ⌈
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. ⌋ (BSW00407, BSW00411)

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

[EFX814] ⌈The standardized common published parameters as required by BSW00402 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].⌋(BSW00402, BSW00374, BSW00379)

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

10.2 Configuration option

[EFX818] ⌈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.⌋(BSW31400001)

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

11 Not applicable requirements

[EFX822] ⌈These requirements are not applicable to this specification.⌋ ( )