Document Title	Specification of Fixed Point Math Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	394
Document Classification	Standard

Document Version	1.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
12.12.2011	1.2.0	AUTOSAR Administration	• Addition to the list of function for consistency and completeness • Fix typing errors in document
15.11.2010	1.1.0	AUTOSAR Administration	• New API created to achieve completion of the need • File structure has been detailed for what concerns naming conventions
30.11.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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

Known Limitations

- No requirements on Service library can be implemented in multiple ways. Many small routines can be combined into one implementation file. For bigger routines, one file shall contain one routine implementation. Generally one routine per object file is recommended from linker optimization point of view. For Bit handling routines more routines can contribute to form one object file. This kind of grouping is not achieved in Release 4.0, Rev001 and will be addressed in Release 4.0, rev002

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR library dedicated to arithmetic routines for fixed point values.

This mathematical library (MFX) contains the following routines :

- addition
- subtraction
- absolute value
- absolute value of differences
- multiplication
- division
- combination of multiplication and division
- combination of multiplication and shift right
- combination of division and shift left
- modulo
- limitation

Some of these functions are proposed too for 2^n Scaled Integers :

- addition
- subtraction
- absolute value
- absolute value of differences
- multiplication
- division
- conversion (specific to 2^n Scaled Integers)

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:***
Abs	Absolute value
AbsDiff	Absolute value of a difference
Add	Addition
AR	Autosar
BSW	Basic Software
DET	Development Error Tracer
Div	Division
DivShLeft	Combination of division and shift left
ECU	Electronic Control Unit
Limit	Limitation routine
Max	Maximum
MFX/Mfx	Math – Fixed Point library
Min	Minimum
Minmax	Limitation with only one value for min and max
Mod	Modulo routine
Mul	Multiplication
MulDiv	Combination of multiplication and division
MulShRight	Combination of multiplication and shift right
s16	Mnemonic for the sint16, specified in AUTOSAR_SWS_PlatformTypes
s32	Mnemonic for the sint32, specified in AUTOSAR_SWS_PlatformTypes
s8	Mnemonic for the sint8, specified in AUTOSAR_SWS_PlatformTypes
Sub	Subtraction
SWS	Software Specification
u16	Mnemonic for the uint16, specified in AUTOSAR_SWS_PlatformTypes
u32	Mnemonic for the uint32, specified in AUTOSAR_SWS_PlatformTypes
u8	Mnemonic for the uint8, 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] AUTOSAR Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[6] Specification of Platform Types,
AUTOSAR_SWS_PlatformTypes.pdf

[7] Requirement on Libraries,
AUTOSAR_SRS_Libraries.pdf

[8] Specification of C Implementation Rules,
AUTOSAR_TR_CImplementationRules.pdf

[9] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.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

[MFX001] ⌈The MFX module shall provide the following files:

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

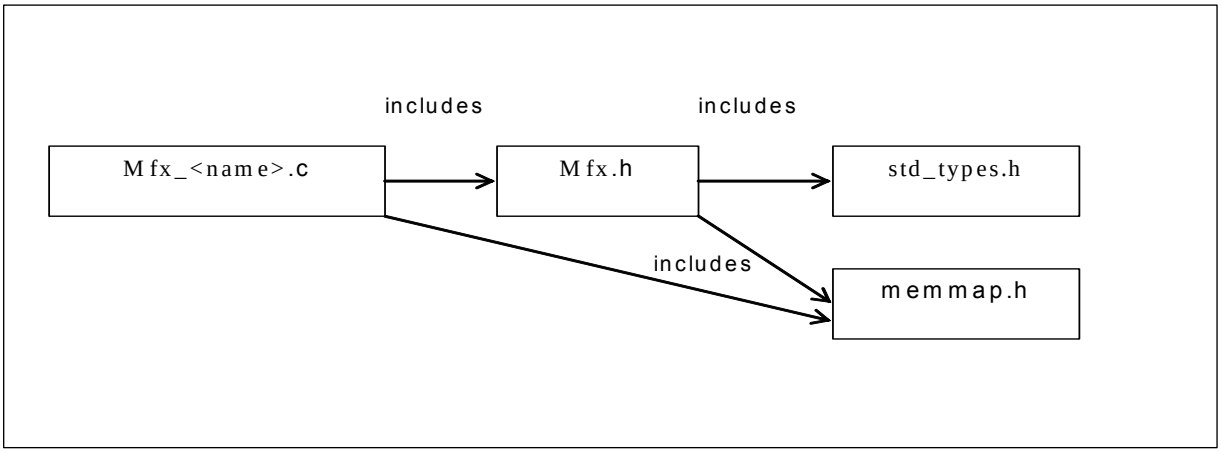

Figure 1: File structure

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

Option 1 : <Name> can be function name providing one C file per function,
eg.: Mfx_Add_u8u8_u8.c etc.

Option 2 : <Name> can have common name of group of functions:
2.1 Group by object family:
eg.: Mfx_NomMath.c, Mfx_ScaledMath.c
2.2 Group by routine family:
eg.: Mfx_Add.c
2.3 Group by method family: if it makes sense
2.4 Group by architecture:
eg.: Mfx_Add8.c
2.5 Group by other methods: (individual grouping allowed)

Option 3 : <Name> can be removed so that single C file shall contain all MFX functions, eg.: Mfx.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		MFX215
BSW00304	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	MFX212
BSW00306	All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, un...	MFX213
BSW00318		MFX215
BSW00321		MFX215
BSW00348	Each AUTOSAR library Module implementation *.	MFX211
BSW00374	The standardized common published parameters as required by BSW00402 in the SRS General on Basic ...	MFX214
BSW00378	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	MFX212
BSW00379	The standardized common published parameters as required by BSW00402 in the SRS General on Basic ...	MFX214
BSW00402	The standardized common published parameters as required by BSW00402 in the SRS General on Basic ...	MFX214
BSW00407		MFX215, MFX216
BSW00411		MFX216
BSW00436	Each AUTOSAR library Module implementation *.	MFX210
BSW007	The library, written in C programming language, should conform to the HIS subset of the MISRA C S...	MFX209
BSW31400001	The MFX library shall not have any configuration options that may affect the functional behavior ...	MFX218
BSW31400002	MFX library shall not require initialization phase.	MFX200
BSW31400003	MFX library shall not require a shutdown operation phase.	MFX201
BSW31400004	MFX API can be directly called from BSW modules or SWC.	MFX203
BSW31400005	The MFX module shall provide the following files:	MFX001
BSW31400006	The statement "Mfx.	MFX204
BSW31400007	Using a library should be documented.	MFX205
BSW31400013	Error detection: Function should check at runtime (both in production and development code) the v...	MFX220, MFX217
BSW31400015	The MFX library shall be implemented in a way that the code can be shared among callers in differ...	MFX206
BSW31400017	Usage of macros should be avoided.	MFX207
BSW31400018	A library function shall not call any BSW modules functions, e.	MFX208

7 Functional specification

7.1 Error classification

[MFX219] ⌈The Mfx module shall be able to detect the following errors and exceptions depending on its configuration (development/production)⌋( )

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API Service called with wrong parameter	Development	MFX_E_PARAM_POINTER	0x01

7.2 Error detection

[MFX220] ⌈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

[MFX217] ⌈The functions shall not call the DET for error notification.⌋(BSW31400013)

7.4 Initialization and shutdown

[MFX200] ⌈MFX 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)

[MFX201] ⌈MFX library shall not require a shutdown operation phase.⌋(BSW31400003)

7.5 Using Library API

[MFX203] ⌈MFX API can be directly called from BSW modules or SWC. No port definition is required. It is a pure function call.⌋(BSW31400004)

[MFX204] ⌈The statement "Mfx.h" shall be placed by the developer or an application code generator but not by the RTE generator⌋(BSW31400006)

[MFX205] ⌈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.⌋(BSW31400007)

7.6 Library implementation

[MFX206] ⌈The MFX library shall be implemented in a way that the code can be shared among callers in different memory partitions.⌋(BSW31400015)

[MFX207] ⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used.⌋(BSW31400017)

[MFX208] ⌈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)

[MFX209] ⌈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)

[MFX210] ⌈Each AUTOSAR library Module implementation <library>*.c and <library>*.h shall map their code to memory sections using the AUTOSAR memory mapping mechanism.⌋(BSW00436)

[MFX211] ⌈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)

[MFX212] ⌈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. ⌋(BSW00378, BSW00304)

[MFX213] ⌈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. ⌋(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	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 mnemonics are used in the library routine names.

Size	Platform Type	Mnemonic
signed 8-Bit	sint8	s8
signed 16-Bit	sint16	s16
signed 32-Bit	sint32	s32
unsigned 8-Bit	uint8	u8
unsigned 16-Bit	uint16	u16
unsigned 32-Bit	uint32	u32

Table 1: Base Types

As described in [6], the ranges for each of the base types are shown in Table 2.

Base Type	Range
uint8	[0, 255]
sint8	[-128, 127]
uint16	[0, 65535]
sint16	[-32768, 32767]
uint32	[0, 4294967295]
sint32	[-2147483648, 2147483647]

Table 2: Ranges for 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 <InType1> 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 optimization

8.4.1 Optimized with constants

For optimization purpose, in some routines, it is mandatory that an argument of the function “must be constant”.

The requirement is that the expression must be fully evaluated at compile time. It may be a constant literal, macro, or arithmetic expression that can be computed at compile time. It may not contain a variable or function call.

For example, the parameters for the radix points are constant expressions so that they may be eliminated after the pre-process phase of compilation. When implemented properly as an inline function or macro, the calculations for the number of shifts necessary are done at compile time, not at run time. There is a ROM/throughput penalty when constant expressions are not used.

8.5 Mathematical routines definitions

8.5.1 Additions

[MFX002] ⌈

Service name:	Mfx_Add_<InTypeMn1><InTypeMn2>_<OutTypeMn>

Syntax:	`<OutType> Mfx_Add_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x_value,`
	`  <InType2> y_value`
	`)`

Service ID[hex]:	0x01 to 0x18
Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX006:** This routine makes an addition between the two arguments: Return-value = x_value + y_value **MFX007:** Return-value shall be saturated to boundary values in the event of underflow or overflow.	

( )

[MFX008]
Here is the list of implemented functions.

Function ID[hex]	*Function prototype*
0x001	uint8 Mfx_Add_u8u8_u8(uint8 , uint8);
0x002	uint8 Mfx_Add_u8s8_u8(uint8 , sint8);
0x003	sint8 Mfx_Add_u8s8_s8(uint8 , sint8);
0x004	uint8 Mfx_Add_s8s8_u8(sint8 , sint8);
0x005	sint8 Mfx_Add_s8s8_s8(sint8 , sint8);
0x006	uint16 Mfx_Add_u16u16_u16(uint16 , uint16);
0x007	uint16 Mfx_Add_u16s16_u16(uint16 , sint16);
0x008	sint16 Mfx_Add_u16s16_s16(uint16 , sint16);
0x009	uint8 Mfx_Add_s16s16_u8(sint16 , sint16);
0x00A	sint8 Mfx_Add_s16s16_s8(sint16 , sint16);
0x00B	uint16 Mfx_Add_s16s16_u16(sint16 , sint16);
0x00C	sint16 Mfx_Add_s16s16_s16(sint16 , sint16);
0x00D	sint8 Mfx_Add_u32u32_s8(uint32 , uint32);
0x00E	sint16 Mfx_Add_u32u32_s16(uint32 , uint32);
0x00F	uint32 Mfx_Add_u32u32_u32(uint32 , uint32);
0x010	sint32 Mfx_Add_u32u32_s32(uint32 , uint32);
0x011	uint32 Mfx_Add_u32s32_u32(uint32 , sint32);
0x012	sint32 Mfx_Add_u32s32_s32(uint32 , sint32);
0x013	uint32 Mfx_Add_s32s32_u32(sint32 , sint32);
0x014	sint32 Mfx_Add_s32s32_s32(sint32 , sint32);
0x015	uint8 Mfx_Add_s32s32_u8(sint32 , sint32);
0x016	sint8 Mfx_Add_s32s32_s8(sint32 , sint32);
0x017	uint16 Mfx_Add_s32s32_u16(sint32 , sint32);
0x018	sint16 Mfx_Add_s32s32_s16(sint32 , sint32);
0xA01	sint16 Mfx_Add_u32s32_s16(uint32 , sint32);
0xA02	sint8 Mfx_Add_u32s32_s8(uint32 , sint32);
0xA03	uint16 Mfx_Add_u32s32_u16(uint32 , sint32);
0xA04	unit8 Mfx_Add_u32s32_u8(uint32 , sint32);
0xA05	uint16 Mfx_Add_u32u32_u16(uint32 , uint32);
0xA06	uint8 Mfx_Add_u32u32_u8(uint32 , uint32);
0xA07	uint16 Mfx_Add_u16u16_s16(uint16 , uint16);
0xA08	uint8 Mfx_Add_u16u16_u8(uint16 , uint16);
0xA09	uint8 Mfx_Add_u16s16_u8(uint16 , sint16);
0xA0A	sint8 Mfx_Add_u16u16_s8(uint16 , uint16);
0xA0B	sint8 Mfx_Add_u16s16_s8(uint16 , sint16);
0xA0C	sint8 Mfx_Add_u8u8_s8(uint8 , uint8);

⌋ ( )

8.5.2 Subtractions

[MFX009] ⌈

Service name:	Mfx_Sub_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_Sub_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x_value,`
	`  <InType2> y_value`
	`)`	

Service ID[hex]:	0x20 to 0x40	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX010:** This routine makes a subtraction between the two arguments : Return-value = x_value - y_value **MFX011:** Return-value shall be saturated to boundary values in the event of underflow or overflow.	

⌋ ( )

[MFX012] ⌈

Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x020	uint8 Mfx_Sub_u8u8_u8(uint8 , uint8);
0x021	sint8 Mfx_Sub_u8u8_s8(uint8 , uint8);
0x022	uint8 Mfx_Sub_u8s8_u8(uint8 , sint8);
0x023	sint8 Mfx_Sub_s8u8_s8(sint8 , uint8);
0x024	sint8 Mfx_Sub_s8s8_s8(sint8 , sint8);
0x025	uint8 Mfx_Sub_u16u16_u8(uint16 , uint16);
0x026	sint8 Mfx_Sub_u16u16_s8(uint16 , uint16);
0x027	uint8 Mfx_Sub_s16s16_u8(sint16 , sint16);
0x028	sint8 Mfx_Sub_s16s16_s8(sint16 , sint16);
0x029	uint8 Mfx_Sub_s32s32_u8(sint32 , sint32);
0x02A	sint8 Mfx_Sub_s32s32_s8(sint32 , sint32);
0x02B	uint16 Mfx_Sub_u16u16_u16(uint16 , uint16);
0x02C	uint16 Mfx_Sub_u16s16_u16(uint16 , sint16);
0x02D	sint16 Mfx_Sub_s16u16_s16(sint16 , uint16);
0x02E	sint16 Mfx_Sub_u16s16_s16(uint16 , sint16);
0x02F	uint16 Mfx_Sub_s16s16_u16(sint16 , sint16);
0x030	sint16 Mfx_Sub_u16u16_s16(uint16 , uint16);
0x031	sint16 Mfx_Sub_s16s16_s16(sint16 , sint16);
0x032	uint8 Mfx_Sub_s32u32_u8(sint32 , uint32);
0x033	sint8 Mfx_Sub_u32s32_s8(uint32 , sint32);
0x034	uint16 Mfx_Sub_s32u32_u16(sint32 , uint32);
0x035	uint16 Mfx_Sub_u32u32_u16(uint32 , uint32);
0x036	sint16 Mfx_Sub_u32u32_s16(uint32 , uint32);
0x037	uint16 Mfx_Sub_s32s32_u16(sint32 , sint32);

0x038	sint16 Mfx_Sub_s32s32_s16(sint32 , sint32);
0x039	uint32 Mfx_Sub_u32u32_u32(uint32 , uint32);
0x03A	uint32 Mfx_Sub_u32s32_u32(uint32 , sint32);
0x03B	uint32 Mfx_Sub_s32u32_u32(sint32 , uint32);
0x03C	sint32 Mfx_Sub_u32u32_s32(uint32 , uint32);
0x03D	sint32 Mfx_Sub_s32u32_s32(sint32 , uint32);
0x03E	sint32 Mfx_Sub_u32s32_s32(uint32 , sint32);
0x03F	uint32 Mfx_Sub_s32s32_u32(sint32 , sint32);
0x040	sint32 Mfx_Sub_s32s32_s32(sint32 , sint32);
0x043	sint16 Mfx_Sub_s32u32_s16(sint32 , uint32);
0x044	sint8 Mfx_Sub_s32u32_s8(sint32 , uint32);
0x045	sint16 Mfx_Sub_u32s32_s16(uint32 , sint32);
0x046	uint16 Mfx_Sub_u32s32_u16(uint32 , sint32);
0x047	uint8 Mfx_Sub_u32s32_u8(uint32 , sint32);
0x048	sint8 Mfx_Sub_u32u32_s8(uint32 , uint32);
0x049	uint8 Mfx_Sub_u32u32_u8(uint32 , uint32);
0x04A	uint16 Mfx_Sub_s16u16_u16(sint16 , uint16);
0x04B	uint8 Mfx_Sub_u16s16_u8(uint16 , sint16);
0x04C	uint8 Mfx_Sub_s16u16_u8(sint16 , uint16);
0x04D	sint8 Mfx_Sub_u16s16_s8(uint16 , sint16);
0x04E	sint8 Mfx_Sub_s16u16_s8(sint16 , uint16);
0x04F	uint8 Mfx_Sub_s8u8_u8(sint8 , uint8);
0xA15	uint8 Mfx_Sub_s8s8_u8(sint8 , sint8);
0xA16	sint8 Mfx_Sub_u8s8_s8(uint8 , sint8);

⌋ ( )

8.5.3 Absolute value

[MFX013] ⌈

Service name:	Mfx_Abs_<InTypeMn1>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_Abs_<InTypeMn1>_<OutTypeMn>(`
	`  <InType1> x_value`
	`)`	

Service ID[hex]:	0x50 to 0x57	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX014:** This routine computes the absolute value of a signed value : Return-value = | x_value | **MFX015:** Return-value shall be saturated to boundary values in the event of underflow or overflow.	

⌋ ( )

[MFX016] ⌈
Here is the list of implemented functions.

Function ID[hex]	*Function prototype*
0x050	uint8 Mfx_Abs_s8_u8(sint8);
0x051	sint8 Mfx_Abs_s8_s8(sint8);
0x052	uint8 Mfx_Abs_s32_u8(sint32);
0x053	uint16 Mfx_Abs_s16_u16(sint16);
0x054	sint16 Mfx_Abs_s16_s16(sint16);
0x055	sint16 Mfx_Abs_s32_s16(sint32);
0x056	uint32 Mfx_Abs_s32_u32(sint32);
0x057	sint32 Mfx_Abs_s32_s32(sint32);
0x058	sint8 Mfx_Abs_s32_s8(sint32);
0x059	uint16 Mfx_Abs_s32_u16(sint32);

⌋ ( )

8.5.4 Absolute value of a difference

[MFX017] ⌈

Service name:	Mfx_AbsDiff_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_AbsDiff_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x_value,`
	`  <InType2> y_value`
	`)`	

Service ID[hex]:	0x60 to 0x6a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	

Return value:	<OutType>	Result of the calculation
Description:	**MFX018:** This routine computes the absolute value of a difference between 2 values : Return-value = | x_value - y_value | **MFX019:** Return-value shall be saturated to boundary values in the event of overflow.	

⌋ ( )

[MFX020] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x060	uint8 Mfx_AbsDiff_u8u8_u8(uint8 , uint8);
0x061	uint16 Mfx_AbsDiff_u16u16_u16(uint16 , uint16);
0x062	uint8 Mfx_AbsDiff_s16s16_u8(sint16 , sint16);
0x063	uint16 Mfx_AbsDiff_s16s16_u16(sint16 , sint16);
0x064	uint8 Mfx_AbsDiff_u32s32_u8(uint32 , sint32);
0x065	uint16 Mfx_AbsDiff_u32s32_u16(uint32 , sint32);
0x066	uint32 Mfx_AbsDiff_u32s32_u32(uint32 , sint32);
0x067	uint32 Mfx_AbsDiff_u32u32_u32(uint32 , uint32);
0x068	uint8 Mfx_AbsDiff_s32s32_u8(sint32 , sint32);
0x069	sint16 Mfx_AbsDiff_s32s32_s16(sint32 , sint32);
0x06A	sint32 Mfx_AbsDiff_s32s32_s32(sint32 , sint32);
0x06B	sint8 Mfx_AbsDiff_s32s32_s8(sint32 , sint32);
0x06C	uint16 Mfx_AbsDiff_s32s32_u16(sint32 , sint32);
0x06D	uint32 Mfx_AbsDiff_s32s32_u32(sint32 , sint32);
0x06E	uint16 Mfx_AbsDiff_u32u32_u16(uint32 , uint32);
0x06F	uint8 Mfx_AbsDiff_u32u32_u8(uint32 , uint32);
0xA50	sint8 Mfx_Absdiff_u32u32_s8(uint32 , uint32);
0xA51	sint16 Mfx_Absdiff_u32u32_s16(uint32 , uint32);
0xA52	sint32 Mfx_Absdiff_u32u32_s32(uint32 , uint32);
0xA53	sint8 Mfx_Absdiff_u32s32_s8(uint32 , sint32);
0xA54	sint16 Mfx_Absdiff_u32s32_s16(uint32 , sint32);
0xA55	sint32 Mfx_Absdiff_u32s32_s32(uint32 , sint32);
0xA56	uint16 Mfx_AbsDiff_u16s16_u16(uint16 , sint16);
0xA57	sint16 Mfx_AbsDiff_u16u16_s16(uint16 , uint16);
0xA58	sint16 Mfx_AbsDiff_u16s16_s16(uint16 , sint16);
0xA59	sint16 Mfx_AbsDiff_s16s16_s16(sint16 , sint16);
0xA5A	uint8 Mfx_AbsDiff_u16u16_u8(uint16 , uint16);
0xA5B	uint8 Mfx_AbsDiff_u16s16_u8(uint16 , sint16);
0xA5C	sint8 Mfx_AbsDiff_u16u16_s8(uint16 , uint16);
0xA5D	sint8 Mfx_AbsDiff_u16s16_s8(uint16 , sint16);
0xA5E	sint8 Mfx_AbsDiff_s16s16_s8(sint16 , sint16);
0xA5F	uint8 Mfx_AbsDiff_u8s8_u8(uint8 , sint8);
0xA60	uint8 Mfx_AbsDiff_s8s8_u8(sint8 , sint8);
0xA61	sint8 Mfx_AbsDiff_u8u8_s8(uint8 , uint8);
0xA62	sint8 Mfx_AbsDiff_u8s8_s8(uint8 , sint8);
0xA62	sint8 Mfx_AbsDiff_s8s8_s8(sint8 , sint8);

⌋ ( )

8.5.5 Multiplications

[MFX021] ⌈

Service name:	Mfx_Mul_<InTypeMn1><InTypeMn2>_<OutTypeMn>	
Syntax:	<OutType> Mfx_Mul_<InTypeMn1><InTypeMn2>_<OutTypeMn>(<InType1> x_value, <InType2> y_value)	
Service ID[hex]:	0x70 to 0x08d	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX022:** This routine makes a multiplication between the two arguments : Return-value = x_value * y_value **MFX023:** Return-value shall be saturated to boundary values in the event of underflow or overflow.	

⌋ ( )

[MFX024] ⌈

Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x070	uint8 Mfx_Mul_u8u8_u8(uint8 , uint8);
0x071	uint8 Mfx_Mul_s8s8_u8(sint8 , sint8);
0x072**(*)**	uint8 Mfx_Mul_s8u8_u8(sint8 , uint8);
0x073**(*)**	sint8 Mfx_Mul_s8u8_s8(sint8 , uint8);
0x074	sint8 Mfx_Mul_s8s8_s8(sint8 , sint8);
0x075	uint16 Mfx_Mul_u16u16_u16(uint16 , uint16);
0x076**(*)**	uint16 Mfx_Mul_s16u16_u16(sint16 , uint16);
0x077**(*)**	sint16 Mfx_Mul_s16u16_s16(sint16 , uint16);
0x078**(**)**	sint32 Mfx_Mul_s16u16_s32(sint16 , uint16);
0x079**(**)**	sint32 Mfx_Mul_s16s16_s32(sint16 , sint16);
0x07A**(**)**	sint32 Mfx_Mul_s32u16_s32(sint32 , uint16);
0x07B**(**)**	uint32 Mfx_Mul_u16u16_u32(uint16 , uint16);
0x07C**(**)**	uint32 Mfx_Mul_u32u16_u32(uint32 , uint16);
0x07D	uint16 Mfx_Mul_s16s16_u16(sint16 , sint16);
0x07E	uint8 Mfx_Mul_s16s16_u8(sint16 , sint16);
0x07F	sint8 Mfx_Mul_s16s16_s8(sint16 , sint16);
0x080	sint16 Mfx_Mul_s16s16_s16(sint16 , sint16);
0x081	uint32 Mfx_Mul_u32u32_u32(uint32 , uint32);
0x082	sint32 Mfx_Mul_u32u32_s32(uint32 , uint32);
0x083	uint16 Mfx_Mul_s32u32_u16(sint32 , uint32);
0x084	uint32 Mfx_Mul_s32u32_u32(sint32 , uint32);
0x085	sint32 Mfx_Mul_s32u32_s32(sint32 , uint32);
0x086	uint32 Mfx_Mul_s32s32_u32(sint32 , sint32);
0x087	uint8 Mfx_Mul_s32s32_u8(sint32 , sint32);
0x088	sint8 Mfx_Mul_u32u32_s8(uint32 , uint32);
0x089	sint8 Mfx_Mul_s32s32_s8(sint32 , sint32);
0x08A	sint16 Mfx_Mul_u32u32_s16(uint32 , uint32);
0x08B	sint16 Mfx_Mul_s32s32_s16(sint32 , sint32);
0x08C	uint16 Mfx_Mul_s32s32_u16(sint32 , sint32);

0x08D	sint32 Mfx_Mul_s32s32_s32(sint32 , sint32);
0xA21	sint16 Mfx_Mul_u32s32_s16(uint32 , sint32);
0xA22	sint8 Mfx_Mul_u32s32_s8(uint32 , sint32);
0xA23	uint8 Mfx_Mul_u32s32_u8(uint32 , sint32);
0xA24	uint16 Mfx_Mul_u32u32_u16(uint32 , uint32);
0xA25	uint8 Mfx_Mul_u32u32_u8(uint32 , uint32);
0xA26	uint8 Mfx_Mul_u8s8_u8(uint8 , sint8);
0xA27	sint8 Mfx_Mul_u8s8_s8(uint8 , sint8);
0xA28	uint16 Mfx_Mul_u16s16_u16(uint16 , sint16);
0xA29	sint16 Mfx_Mul_u16s16_s16(uint16 , sint16);
0xA2A	sint32 Mfx_Mul_u16s16_s32(uint16 , sint16);
0xA2B	uint16 Mfx_Mul_u32s32_u16(uint32 , sint32);
0xA2C	uint32 Mfx_Mul_u32s32_u32(uint32 , sint32);
0xA2D	sint32 Mfx_Mul_u32s32_s32(uint32 , sint32);
0xA2E	sint16 Mfx_Mul_u16u16_s16(uint16 , uint16);
0xA2F	uint8 Mfx_Mul_u16u16_u8(uint16 , uint16);
0xA70	uint8 Mfx_Mul_u16s16_u8(uint16 , sint16);
0xA71	sint8 Mfx_Mul_u16u16_s8(uint16 , uint16);
0xA72	sint8 Mfx_Mul_u16s16_s8(uint16 , sint16);
0xA73	uint8 Mfx_Mul_u8s8_u8(uint8 , sint8);
0xA74	sint8 Mfx_Mul_u8u8_s8(uint8 , uint8);
0xA75	sint8 Mfx_Mul_u8s8_s8(uint8 , sint8);
0xA76	

Note:

- The function prototypes with the (*) in the above table, do not follow the common requirement that with mixed data type the first parameter is the unsigned one and then comes the signed one. The new added function prototypes are correcting this.
  The function prototypes with the (**) in the above table, are already covered by the 32 bit services function prototypes (e;g; Mfx_Mul_s32u16_s32 --> Mfx_Mul_s32s32_s32) or can be written as normal C line.

Therefore, all these marked function prototypes will be removed in a major future release.

⌋ ( )

8.5.6 Divisions rounded towards 0

[MFX025] ⌈

Service name:	Mfx_Div_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_Div_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x_value,`
	`  <InType2> y_value`
	`)`	

Service ID[hex]:	0x90 to 0x112	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	

Return value:	<OutType>	Result of the calculation
Description:	**MFX026:** These routines make a division between the two arguments : Return-value = x_value / y_value **MFX027:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX028:** 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 **MFX030:** The result is rounded towards 0.	

⌋ ( )

[MFX031] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x090	uint8 Mfx_Div_u8u8_u8(uint8 , uint8);
0x091	uint8 Mfx_Div_s8u8_u8(sint8 , uint8);
0x092	uint8 Mfx_Div_u8s8_u8(uint8 , sint8);
0x093	uint8 Mfx_Div_s8s8_u8(sint8 , sint8);
0x094	sint8 Mfx_Div_u8s8_s8(uint8 , sint8);
0x095	sint8 Mfx_Div_s8u8_s8(sint8 , uint8);
0x096	sint8 Mfx_Div_s8s8_s8(sint8 , sint8);
0x097	uint16 Mfx_Div_u16u16_u16(uint16 , uint16);
0x098	uint16 Mfx_Div_s16u16_u16(sint16 , uint16);
0x099	uint16 Mfx_Div_u16s16_u16(uint16 , sint16);
0x09A	sint16 Mfx_Div_u16s16_s16(uint16 , sint16);
0x09B	sint16 Mfx_Div_s16u16_s16(sint16 , uint16);
0x09C	uint16 Mfx_Div_s16s16_u16(sint16 , sint16);
0x09D	uint8 Mfx_Div_s16s16_u8(sint16 , sint16);
0x09E	sint8 Mfx_Div_s16s16_s8(sint16 , sint16);
0x09F	sint16 Mfx_Div_s16s16_s16(sint16 , sint16);
0x100**(*)**	uint16 Mfx_Div_s32u16_u16(sint32 , uint16);
0x101**(*)**	sint16 Mfx_Div_s32s16_s16(sint32 , sint16);
0x102	sint16 Mfx_Div_s32u32_s16(sint32 , uint32);
0x103**(*)**	uint32 Mfx_Div_u32u16_u32(uint32 , uint16);
0x104**(*)**	uint16 Mfx_Div_u32u16_u16(uint32 , uint16);
0x105	uint32 Mfx_Div_u32u32_u32(uint32 , uint32);
0x106	uint32 Mfx_Div_s32u32_u32(sint32 , uint32);
0x107	uint32 Mfx_Div_u32s32_u32(uint32 , sint32);
0x108	sint32 Mfx_Div_u32s32_s32(uint32 , sint32);
0x109	sint32 Mfx_Div_s32u32_s32(sint32 , uint32);
0x10A	uint32 Mfx_Div_s32s32_u32(sint32 , sint32);
0x10B	uint8 Mfx_Div_s32s32_u8(sint32 , sint32);
0x10C	sint8 Mfx_Div_s32s32_s8(sint32 , sint32);
0x10D	uint16 Mfx_Div_s32s32_u16(sint32 , sint32);
0x10E	sint16 Mfx_Div_s32s32_s16(sint32 , sint32);
0x10F	sint32 Mfx_Div_s32s32_s32(sint32 , sint32);
0x110	sint8 Mfx_Div_u32u32_s8(uint32 , uint32);
0x111	sint16 Mfx_Div_u32u32_s16(uint32 , uint32);
0x112	sint32 Mfx_Div_u32u32_s32(uint32 , uint32);
0x113	sint8 Mfx_Div_s32u32_s8(sint32 , uint32);

0x114	uint16 Mfx_Div_s32u32_u16(sint32 , uint32);
0x115	uint8 Mfx_Div_s32u32_u8(sint32 , uint32);
0x116	sint16 Mfx_Div_u32s32_s16(uint32 , sint32);
0x117	sint8 Mfx_Div_u32s32_s8(uint32 , sint32);
0x118	uint16 Mfx_Div_u32s32_u16(uint32 , sint32);
0x119	uint8 Mfx_Div_u32s32_u8(uint32 , sint32);
0x11A	uint16 Mfx_Div_u32u32_u16(uint32 , uint32);
0x11B	uint8 Mfx_Div_u32u32_u8(uint32 , uint32);
0x11C	sint16 Mfx_Div_u16u16_s16(uint16 , uint16);
0x11D	uint8 Mfx_Div_u16u16_u8(uint16 , uint16);
0x11E	uint8 Mfx_Div_u16s16_u8(uint16 , sint16);
0x11F	uint8 Mfx_Div_s16u16_u8(sint16 , uint16);
0xA80	sint8 Mfx_Div_u16u16_s8(uint16 , uint16);
0xA81	sint8 Mfx_Div_u16s16_s8(uint16 , sint16);
0xA82	sint8 Mfx_Div_s16u16_s8(sint16 , uint16);
0xA83	sint8 Mfx_Div_u8u8_s8(uint8 , uint8);

Note:The function prototypes with the (*) in the above table, are already covered by the 32 bit services function prototypes (e;g; Mfx_Div_s32u16_u16 --> Mfx_Div_s32u32_u16). Therefore, these functions will be removed in a major future release ⌋( )

8.5.7 Divisions rounded off

[MFX032] ⌈

Service name:	Mfx_RDiv_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_RDiv_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x_value,`
	`  <InType2> y_value`
	`)`	

Service ID[hex]:	0x120 to 0x142	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX033:** These routines make a division between the two arguments : Return-value = x_value / y_value **MFX034:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX035:** 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 **MFX037:** The result is rounded off.	

⌋ ( )

[MFX038] ⌈

Here is the list of implemented functions.

Function ID[hex]	***Function prototype : RDiv***
0x120	uint8 Mfx_RDiv_u8u8_u8(uint8 , uint8);
0x121	uint8 Mfx_RDiv_s8u8_u8(sint8 , uint8);
0x122	uint8 Mfx_RDiv_u8s8_u8(uint8 , sint8);
0x123	uint8 Mfx_RDiv_s8s8_u8(sint8 , sint8);
0x124	sint8 Mfx_RDiv_u8s8_s8(uint8 , sint8);
0x125	sint8 Mfx_RDiv_s8u8_s8(sint8 , uint8);
0x126	sint8 Mfx_RDiv_s8s8_s8(sint8 , sint8);
0x127	uint16 Mfx_RDiv_u16u16_u16(uint16 , uint16);
0x128	uint16 Mfx_RDiv_s16u16_u16(sint16 , uint16);
0x129	uint16 Mfx_RDiv_u16s16_u16(uint16 , sint16);
0x12A	sint16 Mfx_RDiv_u16s16_s16(uint16 , sint16);
0x12B	sint16 Mfx_RDiv_s16u16_s16(sint16 , uint16);
0x12C	uint16 Mfx_RDiv_s16s16_u16(sint16 , sint16);
0x12D	uint8 Mfx_RDiv_s16s16_u8(sint16 , sint16);
0x12E	sint8 Mfx_RDiv_s16s16_s8(sint16 , sint16);
0x12F	sint16 Mfx_RDiv_s16s16_s16(sint16 , sint16);
0x130**(*)**	uint16 Mfx_RDiv_s32u16_u16(sint32 , uint16);
0x131**(*)**	sint16 Mfx_RDiv_s32s16_s16(sint32 , sint16);
0x132	sint16 Mfx_RDiv_s32u32_s16(sint32 , uint32);
0x133**(*)**	uint32 Mfx_RDiv_u32u16_u32(uint32 , uint16);

0x134**(*)**	uint16 Mfx_RDiv_u32u16_u16(uint32 , uint16);
0x135	uint32 Mfx_RDiv_u32u32_u32(uint32 , uint32);
0x136	uint32 Mfx_RDiv_s32u32_u32(sint32 , uint32);
0x137	uint32 Mfx_RDiv_u32s32_u32(uint32 , sint32);
0x138	sint32 Mfx_RDiv_u32s32_s32(uint32 , sint32);
0x139	sint32 Mfx_RDiv_s32u32_s32(sint32 , uint32);
0x13A	uint32 Mfx_RDiv_s32s32_u32(sint32 , sint32);
0x13B	uint8 Mfx_RDiv_s32s32_u8(sint32 , sint32);
0x13C	sint8 Mfx_RDiv_s32s32_s8(sint32 , sint32);
0x13D	uint16 Mfx_RDiv_s32s32_u16(sint32 , sint32);
0x13E	sint16 Mfx_RDiv_s32s32_s16(sint32 , sint32);
0x13F	sint32 Mfx_RDiv_s32s32_s32(sint32 , sint32);
0x140	sint8 Mfx_RDiv_u32u32_s8(uint32 , uint32);
0x141	sint16 Mfx_RDiv_u32u32_s16(uint32 , uint32);
0x142	sint32 Mfx_RDiv_u32u32_s32(uint32 , uint32);
0x143	sint8 Mfx_RDiv_s32u32_s8(sint32 , uint32);
0x144	uint16 Mfx_RDiv_s32u32_u16(sint32 , uint32);
0x145	uint8 Mfx_RDiv_s32u32_u8(sint32 , uint32);
0x146	sint16 Mfx_RDiv_u32s32_s16(uint32 , sint32);
0x147	sint8 Mfx_RDiv_u32s32_s8(uint32 , sint32);
0x148	uint16 Mfx_RDiv_u32s32_u16(uint32 , sint32);
0x149	uint8 Mfx_RDiv_u32s32_u8(uint32 , sint32);
0x14A	uint16 Mfx_RDiv_u32u32_u16(uint32 , uint32);
0x14B	uint8 Mfx_RDiv_u32u32_u8(uint32 , uint32);
0x14C	sint16 Mfx_RDiv_u16u16_s16(uint16 , uint16);
0x14D	uint8 Mfx_RDiv_u16u16_u8(uint16 , uint16);
0x14E	uint8 Mfx_RDiv_u16s16_u8(uint16 , sint16);
0x14F	uint8 Mfx_RDiv_s16u16_u8(sint16 , uint16);
0xA70	sint8 Mfx_RDiv_u16u16_s8(uint16 , uint16);
0xA71	sint8 Mfx_RDiv_u16s16_s8(uint16 , sint16);
0xA72	sint8 Mfx_RDiv_s16u16_s8(sint16 , uint16);
0xA73	sint8 Mfx_RDiv_u8u8_s8(uint8 , uint8);

Note:The function prototypes with the (*) in the above table, are already covered by the 32 bit services function prototypes (e;g; Mfx_RDiv_s32u16_u16 --> Mfx_RDiv_s32u32_u16). Therefore, these functions will be removed in a major future release.

⌋ ( )

8.5.8 Combinations of multiplication and division rounded towards 0

[MFX039] ⌈

Service name:	Mfx_MulDiv_<InTypeMn1><InTypeMn2><InTypeMn3>_<OutTypeMn>	
Syntax:	<OutType> Mfx_MulDiv_<InTypeMn1><InTypeMn2><InTypeMn3>_<OutTypeMn>(<InType1> x_value, <InType2> y_value, <InType3> z_value)	
Service ID[hex]:	0x150 to 0x162	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument

	y_value	Second argument
	z_value	Third argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX040:** These routines make a multiplication between the two arguments and a division by the third argument : Return-value = x_value * y_value / z_value **MFX041:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX042:** The result after division by zero is defined by: * If x_value*y_value ≥ 0 then the function returns the maximum value of the output type * If x_value*y_value < 0 then the function returns the minimum value of the output type **MFX044:** The result is rounded towards 0.	

⌋ ( )

[MFX045] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype : Div***
0x150	uint16 Mfx_MulDiv_s32s32s32_u16(sint32 , sint32 , sint32);
0x151	sint16 Mfx_MulDiv_s32s32s32_s16(sint32 , sint32 , sint32);
0x152	uint16 Mfx_MulDiv_u32u32u16_u16(uint32 , uint32 , uint16);
0x153	sint16 Mfx_MulDiv_s32s32s16_s16(sint32 , sint32 , sint16);
0x154	uint16 Mfx_MulDiv_s16u16s16_u16(sint16 , uint16 , sint16);
0x155	uint16 Mfx_MulDiv_s16u16u16_u16(sint16 , uint16 , uint16);
0x156	uint16 Mfx_MulDiv_u16u16u16_u16(uint16 , uint16 , uint16);
0x157	sint16 Mfx_MulDiv_s16u16s16_s16(sint16 , uint16 , sint16);
0x158	sint16 Mfx_MulDiv_s16s16u16_s16(sint16 , sint16 , uint16);
0x159	sint16 Mfx_MulDiv_s16u16u16_s16(sint16 , uint16 , uint16);
0x15A	sint16 Mfx_MulDiv_s16s16s16_s16(sint16 , sint16 , sint16);
0x15B	uint32 Mfx_MulDiv_u32u32u32_u32(uint32 , uint32 , uint32);
0x15C	uint32 Mfx_MulDiv_u32u32s32_u32(uint32 , uint32 , sint32);
0x15D	uint32 Mfx_MulDiv_u32s32u32_u32(uint32 , sint32 , uint32);
0x15E	uint32 Mfx_MulDiv_u32s32s32_u32(uint32 , sint32 , sint32);
0x15F	sint32 Mfx_MulDiv_s32s32u32_s32(sint32 , sint32 , uint32);
0x160	sint32 Mfx_MulDiv_s32u32s32_s32(sint32 , uint32 , sint32);
0x161	sint32 Mfx_MulDiv_s32u32u32_s32(sint32 , uint32 , uint32);
0x162	sint32 Mfx_MulDiv_s32s32s32_s32(sint32 , sint32 , sint32);
0x163	uint16 Mfx_MulDiv_u32u32u32_u16(uint32 , uint32 , uint32);
0x164	uint16 Mfx_MulDiv_u16s16s16_u16(uint16 , sint16 , sint16);
0x165	uint16 Mfx_MulDiv_u16s16u16_u16(uint16 , sint16 , uint16);
0x166	sint16 Mfx_MulDiv_u16s16s16_s16(uint16 , sint16 , sint16);
0x167	sint16 Mfx_MulDiv_u16s16u16_s16(uint16 , sint16 , uint16);
0x168	sint32 Mfx_MulDiv_u32s32s32_s32(uint32 , sint32 , sint32);
0x169	sint32 Mfx_MulDiv_u32s32u32_s32(uint32 , sint32 , uint32);

Note : The redundancy due to commutativity will be reduced in the next version

⌋ ( )

8.5.9 Combinations of multiplication and division rounded off

[MFX046] ⌈

Service name:	Mfx_RMulDiv_<InTypeMn1><InTypeMn2><InTypeMn3>_<OutTypeMn>	
Syntax:	<OutType> Mfx_RMulDiv_<InTypeMn1><InTypeMn2><InTypeMn3>_<OutTypeMn>(<InType1> x_value, <InType2> y_value, <InType3> z_value)	
Service ID[hex]:	0x170 to 0x182	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
	z_value	Third argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX047:** These routines make a multiplication between the two arguments and a division by the third argument : Return-value = x_value * y_value / z_value **MFX048:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX049:** The result after division by zero is defined by: * If x_value*y_value ≥ 0 then the function returns the maximum value of the output type * If x_value*y_value < 0 then the function returns the minimum value of the output type **MFX051:** The result is rounded off.	

⌋ ( )

[MFX052] ⌈

Here is the list of implemented functions.

Function ID[hex]	***Function prototype : RDiv***
0x170	uint16 Mfx_RMulDiv_s32s32s32_u16(sint32 , sint32 , sint32);
0x171	sint16 Mfx_RMulDiv_s32s32s32_s16(sint32 , sint32 , sint32);
0x172	uint16 Mfx_RMulDiv_u32u32u16_u16(uint32 , uint32 , uint16);
0x173	sint16 Mfx_RMulDiv_s32s32s16_s16(sint32 , sint32 , sint16);
0x174	uint16 Mfx_RMulDiv_s16u16s16_u16(sint16 , uint16 , sint16);
0x175	uint16 Mfx_RMulDiv_s16u16u16_u16(sint16 , uint16 , uint16);
0x176	uint16 Mfx_RMulDiv_u16u16u16_u16(uint16 , uint16 , uint16);
0x177	sint16 Mfx_RMulDiv_s16u16s16_s16(sint16 , uint16 , sint16);
0x178	sint16 Mfx_RMulDiv_s16s16u16_s16(sint16 , sint16 , uint16);
0x179	sint16 Mfx_RMulDiv_s16u16u16_s16(sint16 , uint16 , uint16);
0x17A	sint16 Mfx_RMulDiv_s16s16s16_s16(sint16 , sint16 , sint16);
0x17B	uint32 Mfx_RMulDiv_u32u32u32_u32(uint32 , uint32 , uint32);
0x17C	uint32 Mfx_RMulDiv_u32u32s32_u32(uint32 , uint32 , sint32);
0x17D	uint32 Mfx_RMulDiv_u32s32u32_u32(uint32 , sint32 , uint32);

0x17E	uint32 Mfx_RMulDiv_u32s32s32_u32(uint32 , sint32 , sint32);
0x17F	sint32 Mfx_RMulDiv_s32s32u32_s32(sint32 , sint32 , uint32);
0x180	sint32 Mfx_RMulDiv_s32u32s32_s32(sint32 , uint32 , sint32);
0x181	sint32 Mfx_RMulDiv_s32u32u32_s32(sint32 , uint32 , uint32);
0x182	sint32 Mfx_RMulDiv_s32s32s32_s32(sint32 , sint32 , sint32);
0x183	uint16 Mfx_RMulDiv_u32u32u32_u16(uint32 , uint32 , uint32);
0x184	uint16 Mfx_RMulDiv_u16s16s16_u16(uint16 , sint16 , sint16);
0x185	uint16 Mfx_RMulDiv_u16s16u16_u16(uint16 , sint16 , uint16);
0x186	sint16 Mfx_RMulDiv_u16s16s16_s16(uint16 , sint16 , sint16);
0x187	sint16 Mfx_RMulDiv_u16s16u16_s16(uint16 , sint16 , uint16);
0x188	sint32 Mfx_RMulDiv_u32s32s32_s32(uint32 , sint32 , sint32);
0x189	sint32 Mfx_RMulDiv_u32s32u32_s32(uint32 , sint32 , uint32);

Note : The redundancy due to commutativity will be reduced in the next version

⌋ ( )

8.5.10 Combinations of multiplication and shift right

[MFX053] ⌈

Service name:	Mfx_MulShRight_<InTypeMn1><InTypeMn2>_<OutTypeMn>	
Syntax:	<OutType> Mfx_MulShRight_<InTypeMn1><InTypeMn2>_<OutTypeMn>(<InType1> x_value, <InType2> y_value, uint8 shift)	
Service ID[hex]:	0x190 to 0x205	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
	shift	Third argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX054:** This routine makes a multiplication between the two arguments and apply a shift right defined by the third argument : Return-value = (x_value * y_value) >> shift **MFX055:** We precise that for the shift right of a negative number, we always keep the bit of sign. **MFX056:** Return-value shall be saturated to boundary values in the event of underflow or overflow.	

⌋ ( )

[MFX057] ⌈

Here is the list of implemented functions.

Function ID[hex]	*Function prototype*
0x190	uint8 Mfx_MulShRight_s16s16u8_u8(sint16 , sint16 , uint8);
0x191	sint8 Mfx_MulShRight_s16s16u8_s8(sint16 , sint16 , uint8);
0x192	sint16 Mfx_MulShRight_s16s16u8_s16(sint16 , sint16 , uint8);
0x193	uint16 Mfx_MulShRight_s16s16u8_u16(sint16 , sint16 , uint8);
0x194	uint8 Mfx_MulShRight_u32s32u8_u8(uint32 , sint32 , uint8);
0x195	sint8 Mfx_MulShRight_u32s32u8_s8(uint32 , sint32 , uint8);

0x196	uint16 Mfx_MulShRight_u32s32u8_u16(uint32 , sint32 , uint8);
0x197	sint16 Mfx_MulShRight_u32s32u8_s16(uint32 , sint32 , uint8);
0x198	uint32 Mfx_MulShRight_u32s32u8_u32(uint32 , sint32 , uint8);
0x199	sint32 Mfx_MulShRight_u32s32u8_s32(uint32 , sint32 , uint8);
0x19A	sint8 Mfx_MulShRight_s32s32u8_s8(sint32 , sint32 , uint8);
0x19B	uint8 Mfx_MulShRight_s32s32u8_u8(sint32 , sint32 , uint8);
0x19C	sint16 Mfx_MulShRight_s32s32u8_s16(sint32 , sint32 , uint8);
0x19D	uint16 Mfx_MulShRight_s32s32u8_u16(sint32 , sint32 , uint8);
0x19E	uint32 Mfx_MulShRight_s32s32u8_u32(sint32 , sint32 , uint8);
0x19F	sint32 Mfx_MulShRight_s32s32u8_s32(sint32 , sint32 , uint8);
0x200	uint8 Mfx_MulShRight_u32u32u8_u8(uint32 , uint32 , uint8);
0x201	sint8 Mfx_MulShRight_u32u32u8_s8(uint32 , uint32 , uint8);
0x202	uint16 Mfx_MulShRight_u32u32u8_u16(uint32 , uint32 , uint8);
0x203	sint16 Mfx_MulShRight_u32u32u8_s16(uint32 , uint32 , uint8);
0x204	uint32 Mfx_MulShRight_u32u32u8_u32(uint32 , uint32 , uint8);
0x205	sint32 Mfx_MulShRight_u32u32u8_s32(uint32 , uint32 , uint8);

If you want to see an example of the use of these functions, see 8.7.1 .

⌋ ( )

8.5.11 Combinations of division and shift left

[MFX058] ⌈

Service name:	Mfx_DivShLeft_<InTypeMn1><InTypeMn2>u8_<OutTypeMn>	

Syntax:	`<OutType> Mfx_DivShLeft_<InTypeMn1><InTypeMn2>u8_<OutTypeMn>(`
	`  <InType1> x_value,`
	`  <InType2> y_value,`
	`  uint8 shift`
	`)`	

Service ID[hex]:	0x210 to 0x218	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
	shift	Third argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	Result of the calculation
Description:	**MFX059:** This routine apply a shift left defined by the third argument to the first argument, and then makes a division by the second arguments: Return-value = (x_value << shift) / y_value **MFX060:** We precise that for the shift left of a negative number, we always keep the bit of sign. **MFX061:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX062:** 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	

⌋ ( )

[MFX064] ⌈
Here is the list of implemented functions.

Function ID[hex]	*Function prototype*
0x210	uint8 Mfx_DivShLeft_u8u8u8_u8(uint8 , uint8 , uint8);
0x211	uint8 Mfx_DivShLeft_u16u16u8_u8(uint16 , uint16 , uint8);
0x212	uint16 Mfx_DivShLeft_u16u16u8_u16(uint16 , uint16 , uint8);
0x213	sint16 Mfx_DivShLeft_s16s16u8_s16(sint16 , sint16 , uint8);
0x214	sint16 Mfx_DivShLeft_s16u16u8_s16(sint16 , uint16 , uint8);
0x215	uint16 Mfx_DivShLeft_u32u32u8_u16(uint32 , uint32 , uint8);
0x216	uint32 Mfx_DivShLeft_u32u32u8_u32(uint32 , uint32 , uint8);
0x217	sint32 Mfx_DivShLeft_s32s32u8_s32(sint32 , sint32 , uint8);
0x218	sint32 Mfx_DivShLeft_s32u32u8_s32(sint32 , uint32 , uint8);
0xA30	uint8 Mfx_DivShLeft_u32s32u8_u8(uint32 , sint32 , uint8);
0xA31	sint8 Mfx_DivShLeft_u32s32u8_s8(uint32 , sint32 , uint8);
0xA32	uint16 Mfx_DivShLeft_u32s32u8_u16(uint32 , sint32 , uint8);
0xA33	sint16 Mfx_DivShLeft_u32s32u8_s16(uint32 , sint32 , uint8);
0xA34	uint32 Mfx_DivShLeft_u32s32u8_u32(uint32 , sint32 , uint8);
0xA35	sint32 Mfx_DivShLeft_u32s32u8_s32(uint32 , sint32 , uint8);
0xA36	sint8 Mfx_DivShLeft_s32s32u8_s8(sint32 , sint32 , uint8);
0xA37	uint8 Mfx_DivShLeft_s32s32u8_u8(sint32 , sint32 , uint8);
0xA38	sint16 Mfx_DivShLeft_s32s32u8_s16(sint32 , sint32 , uint8);
0xA39	uint16 Mfx_DivShLeft_s32s32u8_u16(sint32 , sint32 , uint8);
0xA3A	uint32 Mfx_DivShLeft_s32s32u8_u32(sint32 , sint32 , uint8);
0xA3B	uint8 Mfx_DivShLeft_u32u32u8_u8(uint32 , uint32 , uint8);
0xA3C	sint8 Mfx_DivShLeft_u32u32u8_s8(uint32 , uint32 , uint8);
0xA3D	sint16 Mfx_DivShLeft_u32u32u8_s16(uint32 , uint32 , uint8);
0xA3E	sint32 Mfx_DivShLeft_u32u32u8_s32(uint32 , uint32 , uint8);
0xA3F	uint8 Mfx_DivShLeft_s32u32u8_u8(uint32 , sint32 , uint8);
0xA40	sint8 Mfx_DivShLeft_s32u32u8_s8(uint32 , sint32 , uint8);
0xA41	uint16 Mfx_DivShLeft_s32u32u8_u16(uint32 , sint32 , uint8);
0xA42	sint16 Mfx_DivShLeft_s32u32u8_s16(uint32 , sint32 , uint8);
0xA43	uint32 Mfx_DivShLeft_s32u32u8_u32(uint32 , sint32 , uint8);

If you want to see an example of the use of these functions, see 8.7.2.
⌋ ( )

8.5.12 Modulo

[MFX065] ⌈

Service name:	Mfx_Mod_<TypeMn>	

Syntax:	`<Type> Mfx_Mod_<TypeMn>(`
	`  <Type> x_value,`
	`  <Type> y_value`
	`)`	

Service ID[hex]:	0x220 to 0x225	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	

Return value:	<Type>	Result of the calculation
Description:	This routine returns the remaining of the divison x_value / y_value if y_value is not zero. **MFX066:** If y_value is zero, the result is zero. **MFX068:** In other cases, Return-value = x_value mod y_value **MFX069:** The sign of the remainder is the same than the sign of x_value.	

⌋ ( )

[MFX070] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x220	uint8 Mfx_Mod_u8(uint8 , uint8);
0x221	sint8 Mfx_Mod_s8(sint8 , sint8);
0x222	uint16 Mfx_Mod_u16(uint16 , uint16);
0x223	sint16 Mfx_Mod_s16(sint16 , sint16);
0x224	uint32 Mfx_Mod_u32(uint32 , uint32);
0x225	sint32 Mfx_Mod_s32(sint32 , sint32);

⌋ ( )

8.5.13 Limiting

[MFX073] ⌈

Service name:	Mfx_Limit_<TypeMn>	

Syntax:	`<Type> Mfx_Limit_<TypeMn>(`
	`  <Type> value,`
	`  <Type> min_value,`
	`  <Type> max_value`
	`)`	

Service ID[hex]:	0x230 to 0x235	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	value	First argument
	min_value	Second argument
	max_value	Third argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<Type>	Result of the calculation
Description:	This routine limits a value to a minimum or a maximum: **MFX074:** Return-value =min_value if value < min_value **MFX075:** Return-value =max_value if value > max_value **MFX076:** Return-value = value in the other cases **MFX077:** We shall have min_value≤max_value. If not, the default Return-value is value.	

⌋ ( )

[MFX079] ⌈
Here is the list of implemented functions.

Function ID[hex]	Function prototype
0x230	uint8 Mfx_Limit_u8(uint8 , uint8, uint8);
0x231	sint8 Mfx_Limit_s8(sint8 , sint8, sint8);
0x232	uint16 Mfx_Limit_u16(uint16 , uint16, uint16);
0x233	sint16 Mfx_Limit_s16(sint16 , sint16, sint16);
0x234	uint32 Mfx_Limit_u32(uint32 , uint32, uint32);
0x235	sint32 Mfx_Limit_s32(sint32 , sint32, sint32);

⌋ ( )

8.5.14 Limitations with only one value for minimum and maximum

[MFX082] ⌈

Service name:	Mfx_Minmax_<TypeMn>	

Syntax:	`<Type> Mfx_Minmax_<TypeMn>(`
	`  <Type> value,`
	`  <Type> minmax_value`
	`)`	

Service ID[hex]:	0x240 to 0x245	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	value	First argument
	minmax_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<Type>	Result of the calculation
Description:	The routine limits a value to a minimum or a maximum that depends on the sign of the minmax_value. The result value is : **MFX083:** minmax_value if minmax_value ≥ 0 and value > minmax_value **MFX084:** minmax_value if minmax_value < 0 and value < minmax_value **MFX085:** value in the other cases	

⌋ ( )

[MFX086] ⌈

Here is the list of implemented functions.

Function ID[hex]	Function prototype
0x240	uint8 Mfx_Minmax_u8(uint8 , uint8);
0x241	sint8 Mfx_Minmax_s8(sint8 , sint8);
0x242	uint16 Mfx_Minmax_u16(uint16 , uint16);
0x243	sint16 Mfx_Minmax_s16(sint16 , sint16);
0x244	uint32 Mfx_Minmax_u32(uint32 , uint32);
0x245	sint32 Mfx_Minmax_s32(sint32 , sint32);

⌋ ( )

8.5.15 Minimum and maximum

[MFX090] ⌈

Service name:	Mfx_Min_<TypeMn>	
Syntax:	<Type> Mfx_Min_<TypeMn>(<Type> x_value, <Type> y_value)	
Service ID[hex]:	0x250 to 0x255	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<Type>	Result of the calculation
Description:	The routine MFX_Min returns the minimum between two values : **MFX091:** Return-value =x_value if x_value < y_value **MFX092:** Return-value = y_value in the other case	

⌋ ( )

[MFX093] ⌈

Service name:	Mfx_Max_<TypeMn>	
Syntax:	<Type> Mfx_Max_<TypeMn>(<Type> x_value, <Type> y_value)	
Service ID[hex]:	0x256 to 0x25b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument
	y_value	Second argument
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<Type>	Result of the calculation
Description:	The routine MFX_Max returns the maximum between two values: **MFX094:** Return-value =x_value if x_value > y_value **MFX095:** Return-value = y_value in the other case	

⌋ ( )

[MFX096] ⌈

Here is the list of implemented functions.

Function ID[hex]	*Function prototype*
0x250	uint8 Mfx_Min_u8(uint8 , uint8);
0x251	sint8 Mfx_Min_s8(sint8 , sint8);
0x252	uint16 Mfx_Min_u16(uint16 , uint16);
0x253	sint16 Mfx_Min_s16(sint16 , sint16);
0x254	uint32 Mfx_Min_u32(uint32 , uint32);
0x255	sint32 Mfx_Min_s32(sint32 , sint32);
0x256	uint8 Mfx_Max_u8(uint8 , uint8);
0x257	sint8 Mfx_Max_s8(sint8 , sint8);
0x258	uint16 Mfx_Max_u16(uint16 , uint16);
0x259	sint16 Mfx_Max_s16(sint16 , sint16);
0x25A	uint32 Mfx_Max_u32(uint32 , uint32);
0x25B	sint32 Mfx_Max_s32(sint32 , sint32);

⌋ ( )

8.6 2^n Scaled Integer Math Functions

For all the following functions, upper case letters will be used for operands, and lower case letters will be used for radix.
For example :

- "A" is the operand, "a" is the parameter that represents its radix,
- "C" is the result, "c" is the parameter for its radix.

A Radix will always be a signed integer on 16 bits (sint16). For that reason, the mnemonic will not appear in the name of the functions in order to have shorter names.

For all operations, the valid range is given for information. Indeed, operations with parameters outside of the valid range will be saturated within the range of the output type. It can help for optimization purpose.

8.6.1 Conversion

8.6.1.1 16-Bit to 8-Bit 2^n Scaled Integer Conversion

[MFX100] ⌈

Service name:	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>(`
	`  <InType> x,`
	`  sint16 a,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x260 to 0x261	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed-point operand.
	a	Radix point position of the fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -15 ≤ (c - a) ≤ 7

Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	=2^(c-a) * x
Description:	The routine converts a scaled 16-bit integer to a scaled 8-bit integer. **MFX101:** The function returns the integer value of the fixed point conversion (C), determined according to the above return-value equation. **MFX102:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX103:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX104] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x260	uint8 Mfx_ConvertP2_u16_u8(uint16 A, sint16 a, sint16 c)
0x261	sint8 Mfx_ConvertP2_s16_s8(sint16 A, sint16 a, sint16 c)

⌋ ( )

8.6.1.2 8-Bit to 16-Bit 2^n Scaled Integer Conversion

[MFX106] ⌈

Service name:	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>(`
	`  <InType> x,`
	`  sint16 a,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x26a to 0x26b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed-point operand.
	a	Radix point position of the fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -7 ≤ (c - a) ≤ 15
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	=2^(c-a) * x
Description:	The routine converts a scaled 8-bit integer to a scaled 16-bit integer. **MFX107:** The function returns the integer value of the fixed point conversion (C), determined according to the above return-value equation. **MFX108:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX109:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX110] ⌈
Here is the list of implemented functions.

Function ID[hex]	Function prototype
0x26A	uint16 Mfx_ConvertP2_u8_u16(uint8 A, sint16 a, sint16 c)
0x26B	sint16 Mfx_ConvertP2_s8_s16 (sint8 A, sint16 a, sint16 c)

⌋ ( )

8.6.1.3 32-Bit to 16-Bit 2^n Scaled Integer Conversion

[MFX112] ⌈

Service name:	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>(`
	`  <InType> x,`
	`  sint16 a,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x270 to 0x271	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed-point operand.
	a	Radix point position of the fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -31 ≤ (c - a) ≤ 15
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	=2^(c-a) * x
Description:	The routine converts a scaled 32-bit integer to a scaled 16-bit integer. **MFX113:** The function returns the integer value of the fixed point conversion (C), determined according to the above return-value equation. **MFX114:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX115:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX116] ⌈

Here is the list of implemented functions.

Function ID[hex]	Function prototype
0x270	uint16 Mfx_ConvertP2_u32_u16 (uint32 A, sint16 a, sint16 c)
0x271	sint16 Mfx_ConvertP2_s32_s16 (sint32 A, sint16 a, sint16 c)

⌋ ( )

8.6.1.4 16-Bit to 32-Bit 2^n Scaled Integer Conversion

[MFX118] ⌈

Service name:	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>

Syntax:	`<OutType> Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>(`
	`  <InType> x,`
	`  sint16 a,`

	sint16 c)	
Service ID[hex]:	0x27a to 0x27b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed-point operand.
	a	Radix point position of the fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -15 ≤ (c - a) ≤ 31
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	=2^(c-a) * x
Description:	The routine converts a scaled 16-bit integer to a scaled 32-bit integer. **MFX119:** The function returns the integer value of the fixed point conversion (C), determined according to the above return-value equation. **MFX120:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX121:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX122] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x27A	uint32 Mfx_ConvertP2_u16_u32(uint16 A, sint16 a, sint16 c)
0x27B	sint32 Mfx_ConvertP2_s16_s32(sint16 A, sint16 a, sint16 c)

⌋ ( )

8.6.2 Multiplication

8.6.2.1 16-Bit Multiplication of 2^n Scaled Integer

[MFX124] ⌈

Service name:	Mfx_MulP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	
Syntax:	<OutType> Mfx_MulP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(<InType1> x, <InType2> y, sint16 a, sint16 b, sint16 c)	
Service ID[hex]:	0x280 to 0x285	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.

	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -31 ≤ (c - b - a) ≤ 15
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	= 2^(c-b-a) * [x * y]
Description:	The routine multiplies two 16-bit integers with scaling factors set by input parameters. **MFX125:** The function returns the integer value of the fixed point multiplication (C), determined according to the above return-value equation. **MFX126:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX127:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX128] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x280	uint16 Mfx_MulP2_u16u16_u16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x281	uint16 Mfx_MulP2_u16s16_u16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x282	uint16 Mfx_MulP2_s16s16_u16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x283	sint16 Mfx_MulP2_u16u16_s16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x284	sint16 Mfx_MulP2_u16s16_s16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x285	sint16 Mfx_MulP2_s16s16_s16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)

⌋ ( )

8.6.2.2 32-Bit Multiplication of 2^n Scaled Integer

[MFX130] ⌈

Service name:	Mfx_MulP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_MulP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x,`
	`  <InType2> y,`
	`  sint16 a,`
	`  sint16 b,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x290 to 0x295	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -63 ≤ (c - b - a) ≤ 31
Parameters (in-out):	None	

Parameters (out):	None	
Return value:	<OutType>	= 2^(c-b-a) * [x * y]
Description:	The routine multiplies two 32-bit integers with scaling factors set by input parameters. **MFX131:** The function returns the integer value of the fixed point multiplication (C), determined according to the above return-value equation. **MFX132:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX133:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX134] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x290	uint32 Mfx_MulP2_u32u32_u32(uint32 A, uint32 B, sint16 a, sint16 b, sint16 c)
0x291	uint32 Mfx_MulP2_u32s32_u32(uint32 A, sint32 B, sint16 a, sint16 b, sint16 c)
0x292	uint32 Mfx_MulP2_s32s32_u32(sint32 A, sint32 B, sint16 a, sint16 b, sint16 c)
0x293	sint32 Mfx_MulP2_u32u32_s32(uint32 A, uint32 B, sint16 a, sint16 b, sint16 c)
0x294	sint32 Mfx_MulP2_u32s32_s32(uint32 A, sint32 B, sint16 a, sint16 b, sint16 c)
0x295	sint32 Mfx_MulP2_s32s32_s32(sint32 A, sint32 B, sint16 a, sint16 b, sint16 c)

⌋ ( )

8.6.3 Division

8.6.3.1 16-Bit Division of 2^n Scaled Integer

[MFX136] ⌈

Service name:	Mfx_DivP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	<OutType> Mfx_DivP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(
	  <InType1> x,
	  <InType2> y,
	  sint16 a,
	  sint16 b,
	  sint16 c
	)	

Service ID[hex]:	0x300 to 0x307	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -15 ≤ (c + b - a) ≤ 31
Parameters (in-out):	None	
Parameters (out):	None	

Return value:	<OutType>	= [2^(c-b-a) * x] / y
Description:	The routine divides two 16-bit integers with scaling factors set by input parameters. **MFX137**: The function returns the integer value of the fixed point quotient (C), determined according to the above return-value equation. MFX138: Return-value shall be saturated to boundary values in the event of underflow or overflow**.** MFX139: If the divisor, B, is zero, the result is defined by: * If A ≥ 0 then the function returns the maximum value of the output type * If A < 0 then the function returns the minimum value of the output **type** MFX141: If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX142] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x300	uint16 Mfx_DivP2_u16u16_u16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x301	uint16 Mfx_DivP2_u16s16_u16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x302	uint16 Mfx_DivP2_s16u16_u16(sint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x303	uint16 Mfx_DivP2_s16s16_u16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x304	sint16 Mfx_DivP2_u16u16_s16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x305	sint16 Mfx_DivP2_u16s16_s16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x306	sint16 Mfx_DivP2_s16u16_s16(sint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x307	sint16 Mfx_DivP2_s16s16_s16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)

⌋ ( )

8.6.3.2 32-Bit Division of 2^n Scaled Integer

[MFX144] ⌈

Service name:	Mfx_DivP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	<OutType> Mfx_DivP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(
	  <InType1> x,
	  <InType2> y,
	  sint16 a,
	  sint16 b,
	  sint16 c
	)	

Service ID[hex]:	0x310 to 0x317	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -31 ≤ (c + b - a) ≤ 63
Parameters (in-out):	None	

Parameters (out):	None	
Return value:	<OutType>	= [2^(c-b-a) * x] / y
Description:	The routine divides two 32-bit integers with scaling factors set by input parameters. **MFX145:** The function returns the integer value of the fixed point quotient (C), determined according to the above return-value equation. **MFX146:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX147:** If the divisor, B, is zero, the result is defined by: * If A ≥ 0 then the function returns the maximum value of the output type * If A < 0 then the function returns the minimum value of the output type **MFX149:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX150] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x310	uint32 Mfx_DivP2_u32u32_u32(uint32 A, uint32 B, sint16 a, sint16 b, sint16 c)
0x311	uint32 Mfx_DivP2_u32s32_u32(uint32 A, sint32 B, sint16 a, sint16 b, sint16 c)
0x312	uint32 Mfx_DivP2_s32u32_u32(sint32 A, uint32 B, sint16 a, sint16 b, sint16 c)
0x313	uint32 Mfx_DivP2_s32s32_u32(sint32 A, sint32 B, sint16 a, sint16 b, sint16 c)
0x314	sint32 Mfx_DivP2_u32u32_s32(uint32 A, uint32 B, sint16 a, sint16 b, sint16 c)
0x315	sint32 Mfx_DivP2_u32s32_s32(uint32 A, sint32 B, sint16 a, sint16 b, sint16 c)
0x316	sint32 Mfx_DivP2_s32u32_s32(sint32 A, uint32 B, sint16 a, sint16 b, sint16 c)
0x317	sint32 Mfx_DivP2_s32s32_s32(sint32 A, sint32 B, sint16 a, sint16 b, sint16 c)

⌋ ( )

8.6.4 Addition

8.6.4.1 16-Bit Addition of 2^n Scaled Integer

[MFX152] ⌈

Service name:	Mfx_AddP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_AddP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x,`
	`  <InType2> y,`
	`  sint16 a,`
	`  sint16 b,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x320 to 0x325	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: 0 ≤ |a - b| ≤ 15

		(c - b) ≤ 15, (a - c) ≤ 15, a ≥ b (c - a) ≤ 15, (b - c) ≤ 15, a < b
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	a ≥ b: 2^(c-a) * [x + (y * 2^(a-b))], a < b: 2^(c-b) * [(x * 2^(b-a)) + y]
Description:	The routine adds two 16-bit integers with scaling factors set by input parameters. **MFX153:** The function returns the integer value of the fixed point sum (C), determined according to the above return-value equation. **MFX154:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX155:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX156] ⌈

Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x320	uint16 Mfx_AddP2_u16u16_u16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x321	uint16 Mfx_AddP2_u16s16_u16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x322	uint16 Mfx_AddP2_s16s16_u16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x323	sint16 Mfx_AddP2_u16u16_s16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x324	sint16 Mfx_AddP2_u16s16_s16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x325	sint16 Mfx_AddP2_s16s16_s16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)

⌋ ( )

8.6.4.2 32-Bit Addition of 2^n Scaled Integer

[MFX158] ⌈

Service name:	Mfx_AddP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_AddP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x,`
	`  <InType2> y,`
	`  sint16 a,`
	`  sint16 b,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x330 to 0x335	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: 0 ≤ |a - b| ≤ 31 (c - b) ≤ 31, (a - c) ≤ 31, a ≥ b (c - a) ≤ 31, (b - c) ≤ 31, a < b
Parameters (in-	None	

out):		
Parameters (out):	None	
Return value:	<OutType>	a ≥ b: 2^(c-a) * [x + (y * 2^(a-b))], a < b: 2^(c-b) * [(x * 2^(b-a)) + y]
Description:	The routine adds two 32-bit integers with scaling factors set by input parameters. **MFX159:** The function returns the integer value of the fixed point sum (C), determined according to the above return-value equation. **MFX160:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX161:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX162] ⌈

Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x330	uint32 Mfx_AddP2_u32u32_u32(uint32 A, uint32 B, sint32 a, sint32 b, sint32 c)
0x331	uint32 Mfx_AddP2_u32s32_u32(uint32 A, sint32 B, sint32 a, sint32 b, sint32 c)
0x332	uint32 Mfx_AddP2_s32s32_u32(sint32 A, sint32 B, sint32 a, sint32 b, sint32 c)
0x333	sint32 Mfx_AddP2_u32u32_s32(uint32 A, uint32 B, sint32 a, sint32 b, sint32 c)
0x334	sint32 Mfx_AddP2_u32s32_s32(uint32 A, sint32 B, sint32 a, sint32 b, sint32 c)
0x335	sint32 Mfx_AddP2_s32s32_s32(sint32 A, sint32 B, sint32 a, sint32 b, sint32 c)

⌋ ( )

8.6.5 Subtraction

8.6.5.1 16-Bit Subtraction of 2^n Scaled Integer

[MFX164] ⌈

Service name:	Mfx_SubP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_SubP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x,`
	`  <InType2> y,`
	`  sint16 a,`
	`  sint16 b,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x340 to 0x347	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: 0 ≤ |a - b| ≤ 15 (c - b) ≤ 15, (a - c) ≤ 15, a ≥ b (c - a) ≤ 15, (b - c) ≤ 15, a < b
Parameters (in-	None	

out):		
Parameters (out):	None	
Return value:	<OutType>	a ≥ b: 2^(c-a) * [x - (y * 2^(a-b))] a < b: 2^(c-b) * [(x * 2^(b-a)) - y]
Description:	The routine subtracts two 16-bit integers with scaling factors set by input parameters. **MFX165:** The function returns the integer value of the fixed point difference (C), determined according to the above return-value equation. **MFX166:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX167:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX168] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x340	uint16 Mfx_SubP2_u16u16_u16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x341	uint16 Mfx_SubP2_u16s16_u16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x342	uint16 Mfx_SubP2_s16u16_u16(sint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x343	uint16 Mfx_SubP2_s16s16_u16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x344	sint16 Mfx_SubP2_u16u16_s16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x345	sint16 Mfx_SubP2_u16s16_s16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x346	sint16 Mfx_SubP2_s16u16_s16(sint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x347	sint16 Mfx_SubP2_s16s16_s16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)

⌋ ( )

8.6.5.2 32-Bit Subtraction of 2^n Scaled Integer

[MFX170] ⌈

Service name:	Mfx_SubP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_SubP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x,`
	`  <InType2> y,`
	`  sint16 a,`
	`  sint16 b,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x350 to 0x357	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: 0 ≤ |a - b| ≤ 31 (c - b) ≤ 31, (a - c) ≤ 31, a ≥ b (c - a) ≤ 31, (b - c) ≤ 31, a < b
Parameters (in-	None	

out):		
Parameters (out):	None	
Return value:	<OutType>	a ≥ b: 2^(c-a) * [x - (y * 2^(a-b))] a < b: 2^(c-b) * [(x * 2^(b-a)) - y]
Description:	The routine subtracts two 32-bit integers with scaling factors set by input parameters. **MFX171:** The function returns the integer value of the fixed point difference (C), determined according to the above return-value equation. **MFX172:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX173:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX174] ⌈
Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x350	uint32 Mfx_SubP2_u32u32_u32(uint32 A, uint32 B, sint32 a, sint32 b, sint32 c)
0x351	uint32 Mfx_SubP2_u32s32_u32(uint32 A, sint32 B, sint32 a, sint32 b, sint32 c)
0x352	uint32 Mfx_SubP2_s32u32_u32(sint32 A, uint32 B, sint32 a, sint32 b, sint32 c)
0x353	uint32 Mfx_SubP2_s32s32_u32(sint32 A, sint32 B, sint32 a, sint32 b, sint32 c)
0x354	sint32 Mfx_SubP2_u32u32_s32(uint32 A, uint32 B, sint32 a, sint32 b, sint32 c)
0x355	sint32 Mfx_SubP2_u32s32_s32(uint32 A, sint32 B, sint32 a, sint32 b, sint32 c)
0x356	sint32 Mfx_SubP2_s32u32_s32(sint32 A, uint32 B, sint32 a, sint32 b, sint32 c)
0x357	sint32 Mfx_SubP2_s32s32_s32(sint32 A, sint32 B, sint32 a, sint32 b, sint32 c)

⌋ ( )

8.6.6 Absolute Difference of 2^n Scaled Integer

[MFX176] ⌈

Service name:	Mfx_AbsDiffP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

Syntax:	`<OutType> Mfx_AbsDiffP2_<InTypeMn1><InTypeMn2>_<OutTypeMn>(`
	`  <InType1> x,`
	`  <InType2> y,`
	`  sint16 a,`
	`  sint16 b,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x360 to 0x365	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	b	Radix point position of the second fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: 0 ≤ |a - b| ≤ 15 (c - b) ≤ 15, (a - c) ≤ 15, a ≥ b (c - a) ≤ 15, (b - c) ≤ 15, a < b

Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	a ≥ b: 2^(c-a) * |x - (y * 2^(a-b))| a < b: 2^(c-b) * |(x * 2^(b-a)) - y|
Description:	The routine subtracts and takes the absolute value of two 16-bit integers with scaling factors set by input parameters. **MFX177:** The function returns the integer value of the fixed point absolute difference (C), determined according to the above return-value equation. **MFX178:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX179:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX180] ⌈

Here is the list of implemented functions.

Function ID[hex]	***Function prototype***
0x360	uint16 Mfx_AbsDiffP2_u16u16_u16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x361	uint16 Mfx_AbsDiffP2_u16s16_u16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x362	uint16 Mfx_AbsDiffP2_s16s16_u16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x363	sint16 Mfx_AbsDiffP2_u16u16_s16(uint16 A, uint16 B, sint16 a, sint16 b, sint16 c)
0x364	sint16 Mfx_AbsDiffP2_u16s16_s16(uint16 A, sint16 B, sint16 a, sint16 b, sint16 c)
0x365	sint16 Mfx_AbsDiffP2_s16s16_s16(sint16 A, sint16 B, sint16 a, sint16 b, sint16 c)

⌋ ( )

8.6.7 Absolute Value

8.6.7.1 16-Bit Absolute Value of 2^n Scaled Integer

[MFX182] ⌈

Service name:	Mfx_AbsP2_s16_<OutTypeMn>	

Syntax:	`<OutType> Mfx_AbsP2_s16_<OutTypeMn>(`
	`  <InType1> x,`
	`  sint16 a,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x370 to 0x371	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -15 ≤ (c - a) ≤ 15
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	2^(c-a) * |x|
Description:	The routine takes the absolute value of two 16-bit integers with scaling factors set by input parameters. **MFX183:**	

	The function returns the integer value of the fixed point absolute value (C), determined according to the above return-value equation. **MFX184:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX185:** If it is necessary to round the result of this equation, it is rounded toward zero.

⌋ ( )

[MFX186] ⌈
Here is the list of implemented functions.

Function ID[hex]	*Function prototype*
0x370	uint16 Mfx_AbsP2_s16_u16(sint16 A, sint16 a, sint16 c)
0x371	sint16 Mfx_AbsP2_s16_s16(sint16 A, sint16 a, sint16 c)

⌋ ( )

8.6.7.2 32-Bit Absolute Value of 2n Scaled Integer

[MFX188] ⌈

Service name:	Mfx_AbsP2_s32_<OutTypeMn>	

Syntax:	`<OutType> Mfx_AbsP2_s32_<OutTypeMn>(`
	`  <InType1> x,`
	`  sint16 a,`
	`  sint16 c`
	`)`	

Service ID[hex]:	0x37a to 0x37b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x	Integer value of the fixed point operand.
	a	Integer value of the first fixed point operand. Must be a constant expression.
	c	Radix point position of the fixed point result. Must be a constant expression. Valid range: -31 ≤ (c - a) ≤ 31
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<OutType>	2^(c-a) * |x|
Description:	The routine takes the absolute value of two 32-bit integers with scaling factors set by input parameters. **MFX189:** The function returns the integer value of the fixed point absolute value (C), determined according to the above return-value equation. **MFX190:** Return-value shall be saturated to boundary values in the event of underflow or overflow. **MFX191:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

[MFX192] ⌈
Here is the list of implemented functions.

Function ID[hex]	Function prototype
0x37A	uint32 Mfx_AbsP2_s32_u32(sint32 A, sint16 a, sint16 c)
0x37B	sint32 Mfx_AbsP2_s32_s32(sint32 A, sint16 a, sint16 c)

⌋ ( )

8.7 Examples of use of functions

8.7.1 Combinations of multiplication and shift right

The function that multiplies an argument by a factor of a given range can be interpreted as the combination of multiplication and shift right.

If we consider the factor that is a power of two : 2^{n1}
If we consider the maximum of the type used to code the factor : 2^{n2}-1
Then, the shift right we shall apply to the result of the multiplication is given by : (n2-n1)

For example, we multiply a s8 value (argument1) by a factor of 1 (2^0) coded with an u8 (Max(u8)=2^8-1).
The physical range of the factor is [0 , 0.996]
The result is :

Mfx_MulShRight_s16s16u8_s8(argument1, factor, 8)

8.7.2 Combinations of division and shift left

In the domain of power train, the function that divides two arguments to compute a factor of a given range can be interpreted as the combination of division and shift left.

If we consider the factor that is a power of two : 2^{n1}
If we consider the maximum of the type used to code the result (factor) : 2^{n2}-1
Then, the shift left we shall apply to the result of the division is given by : (n2-n1)

For example, we divide two u16 values (argument1 and argument2) to obtain a factor of 1 (2^0) coded with an u16 (Max(u16)=2^{16}-1).
The physical range of the result is [0 , 0.999985]
The result is :

Mfx_DivShLeft_u16u16u8_u16(argument1, argument2, 16)

8.8 Version API

8.8.1 Mfx_GetVersionInfo

[MFX215] ⌈

Service name:	Mfx_GetVersionInfo

Syntax:	`void Mfx_GetVersionInfo(`
	`  Std_VersionInfoType* versioninfo`
	`)`

Service ID[hex]:	0x400	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (in-out):	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.	

」(BSW00407, BSW003, BSW00318, BSW00321)
The version information of a BSW module generally contains:
Module Id
Vendor Id
Vendor specific version numbers (BSW00407).

[MFX216] 「
If source code for caller and callee of Mfx_ GetVersionInfo is available, the Mfx library should realize Mfx_ GetVersionInfo as a macro defined in the module's header file.」(BSW00407, BSW00411)

[MFX221] 「If development error detection for the Mfx module is enabled: The function Mfx_GetVersionInfo shall raise the error MFX_E_PARAM_POINTER if the parameter versionInfo is a null pointer」( )

8.9 Call-back notifications

None.

8.10 Scheduled functions

The MFX library does not have scheduled functions.

8.11 Expected Interfaces

None.

8.11.1 Mandatory Interfaces

None.

8.11.2 Optional Interfaces

None.

8.11.3 Configurable interfaces

None.

9 Sequence diagrams

Not applicable.

10 Configuration specification

10.1 Published Information

[MFX214] ⌈The standardized common published parameters as required by BSW00402 in the SRS General 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

[MFX218] ⌈The MFX 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

[MFX222] ⌈These requirements are not applicable to this specification.⌋ ()