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

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Missing input parameter and return value description of Mfx_DivShLeft function (SWS_Mfx_00058) added • Editorial change (converted to LaTex)
2020-11-30	R20-11	AUTOSAR Release Management	• Chapter 7.1 Error sections updated
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Added requirement tracing information for SWS_Mfx_00024 • Removal of (**) from Mul variants in SWS_Mfx_00024 • Addition of (*) for 0x078 • Renamed "Development Error Tracer" to "Default Error Tracer" in abbreviations

2016-11-30	4.3.0	AUTOSAR Release Management	• The renaming of "Development Error Tracer" to "Default Error Tracer" is done in abbreviations • Removal of the requirement SWS_Mfx_00204 • Maximum shift value updated for SWS_Mfx_00064 • Updated SWS_Mfx_00073 for clarity in minmax handling • Clarifications
2015-07-31	4.2.2	AUTOSAR Release Management	• Updated SWS_Mfx_00017 for shift value of Function ID 0x200 to 0x205 from 64 to 63 • Updated SWS_Mfx_00001 under Section 5.1 File Structure
2014-10-31	4.2.1	AUTOSAR Release Management	• Minor corrections and clarifications
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial Changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Corrections and removals of duplicate functions • Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Editorial Changes
2011-12-22	4.0.3	AUTOSAR Administration	• Addition to the list of function for consistency and completeness • Fix typing errors in document
2010-09-30	3.1.5	AUTOSAR Administration	• New API created to achieve completion of the need • File structure has been detailed for what concerns naming conventions
2010-02-02	3.1.4	AUTOSAR Administration	• Initial Release

Disclaimer

Table of Contents

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 2n Scaled Integers :

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

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

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the MFXLibrary module that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	**Description:**
Abs	Absolute value
AbsDiff	Absolute value of a difference
Add	Addition
AR	Autosar
BSW	Basic Software
DET	Default 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 & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

[2] ISO/IEC 9899:1990 Programming Language - C
http://www.iso.org

[3] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[4] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[5] Requirements on Libraries
AUTOSAR_SRS_Libraries

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3, SWS BSW General], which is also valid for MFXLibrary.

Thus, the specification SWS BSW General shall be considered as additional and required specification for MFXLibrary.

4 Constraints and assumptions

4.1 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

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 File structure

[SWS_Mfx_00001] ⌈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'.

⌋*(SRS_LIBS_00005)*

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 Tracing

The following tables reference the requirements specified in [4], [5] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_Mfx_00215]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Mfx_00209]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_Mfx_00212]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Mfx_00213]
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	[SWS_Mfx_00215]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_Mfx_00215]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Mfx_00211]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_Mfx_00214]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Mfx_00212]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_Mfx_00214]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_Mfx_00214]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Mfx_00215] [SWS_Mfx_00216]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Mfx_00216]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_Mfx_00210]

Requirement	Description	Satisfied by
[SRS_LIBS_00001]	The functional behavior of each library functions shall not be configurable	[SWS_Mfx_00218]
[SRS_LIBS_00002]	A library shall be operational before all BSW modules and application SW-Cs	[SWS_Mfx_00200]
[SRS_LIBS_00003]	A library shall be operational until the shutdown	[SWS_Mfx_00201]
[SRS_LIBS_00004]	Using libraries shall not pass through a port interface	[SWS_Mfx_00203]
[SRS_LIBS_00005]	Each library shall provide one header file with its public interface	[SWS_Mfx_00001]
[SRS_LIBS_00007]	Using a library should be documented	[SWS_Mfx_00205]
[SRS_LIBS_00015]	It shall be possible to configure the microcontroller so that the library code is shared between all callers	[SWS_Mfx_00206]
[SRS_LIBS_00017]	Usage of macros should be avoided	[SWS_Mfx_00207]
[SRS_LIBS_00018]	A library function may only call library functions	[SWS_Mfx_00208]

7 Functional specification

7.1 Error classification

[SWS_Mfx_00227] ⌈Section 7.1 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.⌋*()*

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

7.1.1 Development Errors

There are no development errors.

7.1.2 Runtime Errors

There are no runtime errors.

7.1.3 Transient Faults

There are no transient faults.

7.1.4 Production Error

There are no production errors.

7.1.5 Extended Production Errors

There are no extended production errors.

7.2 Initialization and shutdown

[SWS_Mfx_00200] ⌈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.⌋*(SRS_LIBS_00002)*

[SWS_Mfx_00201] ⌈MFX library shall not require a shutdown operation phase.⌋*(SRS_LIBS_00003)*

7.3 Using Library API

[SWS_Mfx_00203] ⌈MFX API can be directly called from BSW modules or SWC. No port definition is required. It is a pure function call.⌋*(SRS_LIBS_00004)*

[SWS_Mfx_00205] ⌈Using a library should be documented. if a BSW module or a SWC uses a Library, the developer should add an Implementation-DependencyOnArtifact in the BSW/SWC template.

minVersion and maxVersion parameters correspond to the supplier version. In case of AUTOSAR library, these parameters may be left empty because a SWC or BSW module may rely on a library behaviour, not on a supplier implementation. However, the SWC or BSW modules shall be compatible with the AUTOSAR platform where they are integrated.⌋*(SRS_LIBS_00007)*

7.4 Library implementation

[SWS_Mfx_00206] ⌈The MFX library shall be implemented in a way that the code can be shared among callers in different memory partitions.⌋*(SRS_LIBS_00015)*

[SWS_Mfx_00207] ⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used.⌋*(SRS_LIBS_00017)*

[SWS_Mfx_00208] ⌈A library function shall not call any BSW modules functions, e.g. the DET. A library function can call other library functions. Because a library function shall be re-entrant. But other BSW modules functions may not be re-entrant.⌋*(SRS_LIBS_00018)*

[SWS_Mfx_00209] ⌈The library, written in C programming language, should conform to the MISRA C Standard.

Please refer to SWS_BSW_00115 for more details.⌋*(SRS_BSW_00007)*

[SWS_Mfx_00210] ⌈Each AUTOSAR library Module implementation <library>*.c and

<library>*.h shall map their code to memory sections using the AUTOSAR memory mapping mechanism.⌋*(SRS_BSW_00437)*

[SWS_Mfx_00211] ⌈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.⌋ *(SRS_BSW_00348)*

[SWS_Mfx_00212] ⌈All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of native C data types, unless this library is clearly identified to be compliant only with a platform.⌋ *(SRS_BSW_00378, SRS_BSW_00304)*

[SWS_Mfx_00213] ⌈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.⌋ *(SRS_BSW_00306)*

[SWS_Mfx_00225] ⌈Integral promotion has to be adhered to when implementing Mfx services. Thus, to obtain maximal precision, intermediate results shall not be limited.⌋ *()*

8 API specification

8.1 Imported types

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

Module	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 Platform_Types.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 8.1: 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 8.2: 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

[SWS_Mfx_00002] ⌈

Service Name	Mfx_Add_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

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

Service ID [hex]	0x001 to 0x024	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This routine makes an addition between the two arguments.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00006] ⌈This routine makes an addition between the two arguments:

Return-value = x_value + y_value⌋*()*

[SWS_Mfx_00007] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00008] ⌈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);

▽

△

Function ID[hex]	Function prototype
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);
0x019	sint16 Mfx_Add_u32s32_s16(uint32 , sint32);
0x01A	sint8 Mfx_Add_u32s32_s8(uint32 , sint32);
0x01B	uint16 Mfx_Add_u32s32_u16(uint32 , sint32);
0x01C	unit8 Mfx_Add_u32s32_u8(uint32 , sint32);
0x01D	uint16 Mfx_Add_u32u32_u16(uint32 , uint32);
0x01E	uint8 Mfx_Add_u32u32_u8(uint32 , uint32);
0x01F	sint16 Mfx_Add_u16u16_s16(uint16 , uint16);
0x020	uint8 Mfx_Add_u16u16_u8(uint16 , uint16);
0x021	uint8 Mfx_Add_u16s16_u8(uint16 , sint16);
0x022	sint8 Mfx_Add_u16u16_s8(uint16 , uint16);
0x023	sint8 Mfx_Add_u16s16_s8(uint16 , sint16);
0x024	sint8 Mfx_Add_u8u8_s8(uint8 , uint8);

8.5.2 Subtractions

[SWS_Mfx_00009] ⌈

Service Name	Mfx_Sub_<InTypeMn1><InTypeMn2>_<OutTypeMn>	
Syntax	`<OutType> Mfx_Sub_<InTypeMn1><InTypeMn2>_<OutTypeMn> ( <InType1> x_value, <InType2> y_value )`	
Service ID [hex]	0x025 to 0x054	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This routine makes a subtraction between the two arguments.	

▽

△

Available via	Mfx.h

⌋()

[SWS_Mfx_00010] ⌈This routine makes a subtraction between the two arguments:

Return-value = x_value - y_value⌋()

[SWS_Mfx_00011] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00012] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x025	uint8 Mfx_Sub_u8u8_u8(uint8 , uint8);
0x026	sint8 Mfx_Sub_u8u8_s8(uint8 , uint8);
0x027	uint8 Mfx_Sub_u8s8_u8(uint8 , sint8);
0x028	sint8 Mfx_Sub_s8u8_s8(sint8 , uint8);
0x029	sint8 Mfx_Sub_s8s8_s8(sint8 , sint8);
0x02A	uint8 Mfx_Sub_u16u16_u8(uint16 , uint16);
0x02B	sint8 Mfx_Sub_u16u16_s8(uint16 , uint16);
0x02C	uint8 Mfx_Sub_s16s16_u8(sint16 , sint16);
0x02D	sint8 Mfx_Sub_s16s16_s8(sint16 , sint16);
0x02E	uint8 Mfx_Sub_s32s32_u8(sint32 , sint32);
0x02F	sint8 Mfx_Sub_s32s32_s8(sint32 , sint32);
0x030	uint16 Mfx_Sub_u16u16_u16(uint16 , uint16);
0x031	uint16 Mfx_Sub_u16s16_u16(uint16 , sint16);
0x032	sint16 Mfx_Sub_s16u16_s16(sint16 , uint16);
0x033	sint16 Mfx_Sub_u16s16_s16(uint16 , sint16);
0x034	uint16 Mfx_Sub_s16s16_u16(sint16 , sint16);
0x035	sint16 Mfx_Sub_u16u16_s16(uint16 , uint16);
0x036	sint16 Mfx_Sub_s16s16_s16(sint16 , sint16);
0x037	uint8 Mfx_Sub_s32u32_u8(sint32 , uint32);
0x038	sint8 Mfx_Sub_u32s32_s8(uint32 , sint32);
0x039	uint16 Mfx_Sub_s32u32_u16(sint32 , uint32);
0x03A	uint16 Mfx_Sub_u32u32_u16(uint32 , uint32);
0x03B	sint16 Mfx_Sub_u32u32_s16(uint32 , uint32);
0x03C	uint16 Mfx_Sub_s32s32_u16(sint32 , sint32);
0x03D	sint16 Mfx_Sub_s32s32_s16(sint32 , sint32);
0x03E	uint32 Mfx_Sub_u32u32_u32(uint32 , uint32);
0x03F	uint32 Mfx_Sub_u32s32_u32(uint32 , sint32);
0x040	uint32 Mfx_Sub_s32u32_u32(sint32 , uint32);
0x041	sint32 Mfx_Sub_u32u32_s32(uint32 , uint32);
0x042	sint32 Mfx_Sub_s32u32_s32(sint32 , uint32);
0x043	sint32 Mfx_Sub_u32s32_s32(uint32 , sint32);
0x044	uint32 Mfx_Sub_s32s32_u32(sint32 , sint32);

▽

△

Function ID[hex]	Function prototype
0x045	sint32 Mfx_Sub_s32s32_s32(sint32 , sint32);
0x046	sint16 Mfx_Sub_s32u32_s16(sint32 , uint32);
0x047	sint8 Mfx_Sub_s32u32_s8(sint32 , uint32);
0x048	sint16 Mfx_Sub_u32s32_s16(uint32 , sint32);
0x049	uint16 Mfx_Sub_u32s32_u16(uint32 , sint32);
0x04A	uint8 Mfx_Sub_u32s32_u8(uint32 , sint32);
0x04B	sint8 Mfx_Sub_u32u32_s8(uint32 , uint32);
0x04C	uint8 Mfx_Sub_u32u32_u8(uint32 , uint32);
0x04D	uint16 Mfx_Sub_s16u16_u16(sint16 , uint16);
0x04E	uint8 Mfx_Sub_u16s16_u8(uint16 , sint16);
0x04F	uint8 Mfx_Sub_s16u16_u8(sint16 , uint16);
0x050	sint8 Mfx_Sub_u16s16_s8(uint16 , sint16);
0x051	sint8 Mfx_Sub_s16u16_s8(sint16 , uint16);
0x052	uint8 Mfx_Sub_s8u8_u8(sint8 , uint8);
0x053	uint8 Mfx_Sub_s8s8_u8(sint8 , sint8);
0x054	sint8 Mfx_Sub_u8s8_s8(uint8 , sint8);

8.5.3 Absolute value

[SWS_Mfx_00013] ⌈

Service Name	Mfx_Abs_<InTypeMn1>_<OutTypeMn>	
Syntax	`<OutType> Mfx_Abs_<InTypeMn1>_<OutTypeMn> ( <InType1> x_value )`	
Service ID [hex]	0x055 to 0x05E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This routine computes the absolute value of a signed value.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00014] ⌈This routine computes the absolute value of a signed value:

Return-value = | x_value |⌋()

[SWS_Mfx_00015] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00016] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x055	uint8 Mfx_Abs_s8_u8(sint8);
0x056	sint8 Mfx_Abs_s8_s8(sint8);
0x057	uint8 Mfx_Abs_s32_u8(sint32);
0x058	uint16 Mfx_Abs_s16_u16(sint16);
0x059	sint16 Mfx_Abs_s16_s16(sint16);
0x05A	sint16 Mfx_Abs_s32_s16(sint32);
0x05B	uint32 Mfx_Abs_s32_u32(sint32);
0x05C	sint32 Mfx_Abs_s32_s32(sint32);
0x05D	sint8 Mfx_Abs_s32_s8(sint32);
0x05E	uint16 Mfx_Abs_s32_u16(sint32);

8.5.4 Absolute value of a difference

[SWS_Mfx_00017] ⌈

Service Name	Mfx_AbsDiff_<InTypeMn1><InTypeMn2>_<OutTypeMn>	
Syntax	`<OutType> Mfx_AbsDiff_<InTypeMn1><InTypeMn2>_<OutTypeMn> ( <InType1> x_value, <InType2> y_value )`	
Service ID [hex]	0x05F to 0x082	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This routine computes the absolute value of a difference between 2 values.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00018] ⌈This routine computes the absolute value of a difference between 2 values:

Return-value = | x_value - y_value |⌋*()*

[SWS_Mfx_00019] ⌈Return-value shall be saturated to boundary values in the event of overflow.⌋*()*

[SWS_Mfx_00020] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x05F	uint8 Mfx_AbsDiff_u8u8_u8(uint8 , uint8);
0x060	uint16 Mfx_AbsDiff_u16u16_u16(uint16 , uint16);
0x061	uint8 Mfx_AbsDiff_s16s16_u8(sint16 , sint16);
0x062	uint16 Mfx_AbsDiff_s16s16_u16(sint16 , sint16);
0x063	uint8 Mfx_AbsDiff_u32s32_u8(uint32 , sint32);
0x064	uint16 Mfx_AbsDiff_u32s32_u16(uint32 , sint32);
0x065	uint32 Mfx_AbsDiff_u32s32_u32(uint32 , sint32);
0x066	uint32 Mfx_AbsDiff_u32u32_u32(uint32 , uint32);
0x067	uint8 Mfx_AbsDiff_s32s32_u8(sint32 , sint32);
0x068	sint16 Mfx_AbsDiff_s32s32_s16(sint32 , sint32);
0x069	sint32 Mfx_AbsDiff_s32s32_s32(sint32 , sint32);
0x06A	sint8 Mfx_AbsDiff_s32s32_s8(sint32 , sint32);
0x06B	uint16 Mfx_AbsDiff_s32s32_u16(sint32 , sint32);
0x06C	uint32 Mfx_AbsDiff_s32s32_u32(sint32 , sint32);
0x06D	uint16 Mfx_AbsDiff_u32u32_u16(uint32 , uint32);
0x06E	uint8 Mfx_AbsDiff_u32u32_u8(uint32 , uint32);
0x06F	sint8 Mfx_Absdiff_u32u32_s8(uint32 , uint32);
0x070	sint16 Mfx_Absdiff_u32u32_s16(uint32 , uint32);
0x071	sint32 Mfx_Absdiff_u32u32_s32(uint32 , uint32);
0x072	sint8 Mfx_Absdiff_u32s32_s8(uint32 , sint32);
0x073	sint16 Mfx_Absdiff_u32s32_s16(uint32 , sint32);
0x074	sint32 Mfx_Absdiff_u32s32_s32(uint32 , sint32);
0x075	uint16 Mfx_AbsDiff_u16s16_u16(uint16 , sint16);
0x076	sint16 Mfx_AbsDiff_u16u16_s16(uint16 , uint16);
0x077	sint16 Mfx_AbsDiff_u16s16_s16(uint16 , sint16);
0x078	sint16 Mfx_AbsDiff_s16s16_s16(sint16 , sint16);
0x079	uint8 Mfx_AbsDiff_u16u16_u8(uint16 , uint16);
0x07A	uint8 Mfx_AbsDiff_u16s16_u8(uint16 , sint16);
0x07B	sint8 Mfx_AbsDiff_u16u16_s8(uint16 , uint16);
0x07C	sint8 Mfx_AbsDiff_u16s16_s8(uint16 , sint16);
0x07D	sint8 Mfx_AbsDiff_s16s16_s8(sint16 , sint16);
0x07E	uint8 Mfx_AbsDiff_u8s8_u8(uint8 , sint8);
0x07F	uint8 Mfx_AbsDiff_s8s8_u8(sint8 , sint8);
0x080	sint8 Mfx_AbsDiff_u8u8_s8(uint8 , uint8);
0x081	sint8 Mfx_AbsDiff_u8s8_s8(uint8 , sint8);
0x082	sint8 Mfx_AbsDiff_s8s8_s8(sint8 , sint8);

8.5.5 Multiplications

[SWS_Mfx_00021] ⌈

Service Name	Mfx_Mul_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

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

Service ID [hex]	0x083 to 0x0A7	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This routine makes a multiplication between the two arguments.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00022] ⌈This routine makes a multiplication between the two arguments:

Return-value = x_value * y_value⌋*()*

[SWS_Mfx_00023] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00024] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x083	uint8 Mfx_Mul_u8u8_u8(uint8 , uint8);
0x084	uint8 Mfx_Mul_s8s8_u8(sint8 , sint8);
0x085	sint8 Mfx_Mul_s8s8_s8(sint8 , sint8);
0x086	uint16 Mfx_Mul_u16u16_u16(uint16 , uint16);
0x087	uint16 Mfx_Mul_s16s16_u16(sint16 , sint16);
0x088	uint8 Mfx_Mul_s16s16_u8(sint16 , sint16);
0x089	sint8 Mfx_Mul_s16s16_s8(sint16 , sint16);
0x08A	sint16 Mfx_Mul_s16s16_s16(sint16 , sint16);
0x08B	uint32 Mfx_Mul_u32u32_u32(uint32 , uint32);
0x08C	sint32 Mfx_Mul_u32u32_s32(uint32 , uint32);
0x08D	uint32 Mfx_Mul_s32s32_u32(sint32 , sint32);
0x08E	uint8 Mfx_Mul_s32s32_u8(sint32 , sint32);
0x08F	sint8 Mfx_Mul_u32u32_s8(uint32 , uint32);
0x090	sint8 Mfx_Mul_s32s32_s8(sint32 , sint32);
0x091	sint16 Mfx_Mul_u32u32_s16(uint32 , uint32);
0x092	sint16 Mfx_Mul_s32s32_s16(sint32 , sint32);
0x093	uint16 Mfx_Mul_s32s32_u16(sint32 , sint32);
0x094	sint32 Mfx_Mul_s32s32_s32(sint32 , sint32);
0x095	sint16 Mfx_Mul_u32s32_s16(uint32 , sint32);
0x096	sint8 Mfx_Mul_u32s32_s8(uint32 , sint32);
0x097	uint8 Mfx_Mul_u32s32_u8(uint32 , sint32);
0x098	uint16 Mfx_Mul_u32u32_u16(uint32 , uint32);
0x099	uint8 Mfx_Mul_u32u32_u8(uint32 , uint32);
0x09A	uint8 Mfx_Mul_u8s8_u8(uint8 , sint8);
0x09B	sint8 Mfx_Mul_u8s8_s8(uint8 , sint8);
0x09C	uint16 Mfx_Mul_u16s16_u16(uint16 , sint16);
0x09D	sint16 Mfx_Mul_u16s16_s16(uint16 , sint16);
0x09E	sint32 Mfx_Mul_u16s16_s32(uint16 , sint16);
0x09F	uint16 Mfx_Mul_u32s32_u16(uint32 , sint32);
0x0A0	uint32 Mfx_Mul_u32s32_u32(uint32 , sint32);
0x0A1	sint32 Mfx_Mul_u32s32_s32(uint32 , sint32);
0x0A2	sint16 Mfx_Mul_u16u16_s16(uint16 , uint16);
0x0A3	uint8 Mfx_Mul_u16u16_u8(uint16 , uint16);
0x0A4	uint8 Mfx_Mul_u16s16_u8(uint16 , sint16);
0x0A5	sint8 Mfx_Mul_u16u16_s8(uint16 , uint16);
0x0A6	sint8 Mfx_Mul_u16s16_s8(uint16 , sint16);
0x0A7	sint8 Mfx_Mul_u8u8_s8(uint8 , uint8);

8.5.6 Divisions rounded towards 0

[SWS_Mfx_00025] ⌈

Service Name	Mfx_Div_<InTypeMn1><InTypeMn2>_<OutTypeMn>	
Syntax	`<OutType> Mfx_Div_<InTypeMn1><InTypeMn2>_<OutTypeMn> ( <InType1> x_value, <InType2> y_value )`	
Service ID [hex]	0x0A8 to 0x0D7	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	These routines make a division between the two arguments.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00026] ⌈These routines make a division between the two arguments:

Return-value = x_value / y_value⌋*()*

[SWS_Mfx_00027] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00028] ⌈The result after division by zero is defined by:

* If x_value $\geq$ 0 then the function returns the maximum value of the output type

* If x_value < 0 then the function returns the minimum value of the output type⌋*()*

[SWS_Mfx_00030] ⌈The result is rounded towards 0.⌋*()*

[SWS_Mfx_00031] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x0A8	uint8 Mfx_Div_u8u8_u8(uint8 , uint8);
0x0A9	uint8 Mfx_Div_s8u8_u8(sint8 , uint8);
0x0AA	uint8 Mfx_Div_u8s8_u8(uint8 , sint8);
0x0AB	uint8 Mfx_Div_s8s8_u8(sint8 , sint8);
0x0AC	sint8 Mfx_Div_u8s8_s8(uint8 , sint8);
0x0AD	sint8 Mfx_Div_s8u8_s8(sint8 , uint8);
0x0AE	sint8 Mfx_Div_s8s8_s8(sint8 , sint8);
0x0AF	uint16 Mfx_Div_u16u16_u16(uint16 , uint16);
0x0B0	uint16 Mfx_Div_s16u16_u16(sint16 , uint16);

▽

△

Function ID[hex]	Function prototype
0x0B1	uint16 Mfx_Div_u16s16_u16(uint16 , sint16);
0x0B2	sint16 Mfx_Div_u16s16_s16(uint16 , sint16);
0x0B3	sint16 Mfx_Div_s16u16_s16(sint16 , uint16);
0x0B4	uint16 Mfx_Div_s16s16_u16(sint16 , sint16);
0x0B5	uint8 Mfx_Div_s16s16_u8(sint16 , sint16);
0x0B6	sint8 Mfx_Div_s16s16_s8(sint16 , sint16);
0x0B7	sint16 Mfx_Div_s16s16_s16(sint16 , sint16);
0x0B8	sint16 Mfx_Div_s32u32_s16(sint32 , uint32);
0x0B9	uint32 Mfx_Div_u32u32_u32(uint32 , uint32);
0x0BA	uint32 Mfx_Div_s32u32_u32(sint32 , uint32);
0x0BB	uint32 Mfx_Div_u32s32_u32(uint32 , sint32);
0x0BC	sint32 Mfx_Div_u32s32_s32(uint32 , sint32);
0x0BD	sint32 Mfx_Div_s32u32_s32(sint32 , uint32);
0x0BE	uint32 Mfx_Div_s32s32_u32(sint32 , sint32);
0x0BF	uint8 Mfx_Div_s32s32_u8(sint32 , sint32);
0x0C0	sint8 Mfx_Div_s32s32_s8(sint32 , sint32);
0x0C1	uint16 Mfx_Div_s32s32_u16(sint32 , sint32);
0x0C2	sint16 Mfx_Div_s32s32_s16(sint32 , sint32);
0x0C3	sint32 Mfx_Div_s32s32_s32(sint32 , sint32);
0x0C4	sint8 Mfx_Div_u32u32_s8(uint32 , uint32);
0x0C5	sint16 Mfx_Div_u32u32_s16(uint32 , uint32);
0x0C6	sint32 Mfx_Div_u32u32_s32(uint32 , uint32);
0x0C7	sint8 Mfx_Div_s32u32_s8(sint32 , uint32);
0x0C8	uint16 Mfx_Div_s32u32_u16(sint32 , uint32);
0x0C9	uint8 Mfx_Div_s32u32_u8(sint32 , uint32);
0x0CA	sint16 Mfx_Div_u32s32_s16(uint32 , sint32);
0x0CB	sint8 Mfx_Div_u32s32_s8(uint32 , sint32);
0x0CC	uint16 Mfx_Div_u32s32_u16(uint32 , sint32);
0x0CD	uint8 Mfx_Div_u32s32_u8(uint32 , sint32);
0x0CE	uint16 Mfx_Div_u32u32_u16(uint32 , uint32);
0x0CF	uint8 Mfx_Div_u32u32_u8(uint32 , uint32);
0x0D0	sint16 Mfx_Div_u16u16_s16(uint16 , uint16);
0x0D1	uint8 Mfx_Div_u16u16_u8(uint16 , uint16);
0x0D2	uint8 Mfx_Div_u16s16_u8(uint16 , sint16);
0x0D3	uint8 Mfx_Div_s16u16_u8(sint16 , uint16);
0x0D4	sint8 Mfx_Div_u16u16_s8(uint16 , uint16);
0x0D5	sint8 Mfx_Div_u16s16_s8(uint16 , sint16);
0x0D6	sint8 Mfx_Div_s16u16_s8(sint16 , uint16);
0x0D7	sint8 Mfx_Div_u8u8_s8(uint8 , uint8);

8.5.7 Divisions rounded off

[SWS_Mfx_00032] ⌈

Service Name	Mfx_RDiv_<InTypeMn1><InTypeMn2>_<OutTypeMn>	

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

Service ID [hex]	0x0D8 to 0x107	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	These routines make a division between the two arguments.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00033] ⌈These routines make a division between the two arguments:

Return-value = x_value / y_value⌋()

[SWS_Mfx_00034] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00035] ⌈The result after division by zero is defined by:

* If x_value ≥ 0 then the function returns the maximum value of the output type

* If x_value < 0 then the function returns the minimum value of the output type⌋()

[SWS_Mfx_00037] ⌈The result is rounded off.⌋()

[SWS_Mfx_00038] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype : RDiv
0x0D8	uint8 Mfx_RDiv_u8u8_u8(uint8 , uint8);
0x0D9	uint8 Mfx_RDiv_s8u8_u8(sint8 , uint8);
0x0DA	uint8 Mfx_RDiv_u8s8_u8(uint8 , sint8);
0x0DB	uint8 Mfx_RDiv_s8s8_u8(sint8 , sint8);
0x0DC	sint8 Mfx_RDiv_u8s8_s8(uint8 , sint8);
0x0DD	sint8 Mfx_RDiv_s8u8_s8(sint8 , uint8);
0x0DE	sint8 Mfx_RDiv_s8s8_s8(sint8 , sint8);
0x0DF	uint16 Mfx_RDiv_u16u16_u16(uint16 , uint16);
0x0E0	uint16 Mfx_RDiv_s16u16_u16(sint16 , uint16);
0x0E1	uint16 Mfx_RDiv_u16s16_u16(uint16 , sint16);
0x0E2	sint16 Mfx_RDiv_u16s16_s16(uint16 , sint16);
0x0E3	sint16 Mfx_RDiv_s16u16_s16(sint16 , uint16);

▽

△

Function ID[hex]	Function prototype : RDiv
0x0E4	uint16 Mfx_RDiv_s16s16_u16(sint16 , sint16);
0x0E5	uint8 Mfx_RDiv_s16s16_u8(sint16 , sint16);
0x0E6	sint8 Mfx_RDiv_s16s16_s8(sint16 , sint16);
0x0E7	sint16 Mfx_RDiv_s16s16_s16(sint16 , sint16);
0x0E8	sint16 Mfx_RDiv_s32u32_s16(sint32 , uint32);
0x0E9	uint32 Mfx_RDiv_u32u32_u32(uint32 , uint32);
0x0EA	uint32 Mfx_RDiv_s32u32_u32(sint32 , uint32);
0x0EB	uint32 Mfx_RDiv_u32s32_u32(uint32 , sint32);
0x0EC	sint32 Mfx_RDiv_u32s32_s32(uint32 , sint32);
0x0ED	sint32 Mfx_RDiv_s32u32_s32(sint32 , uint32);
0x0EE	uint32 Mfx_RDiv_s32s32_u32(sint32 , sint32);
0x0EF	uint8 Mfx_RDiv_s32s32_u8(sint32 , sint32);
0x0F0	sint8 Mfx_RDiv_s32s32_s8(sint32 , sint32);
0x0F1	uint16 Mfx_RDiv_s32s32_u16(sint32 , sint32);
0x0F2	sint16 Mfx_RDiv_s32s32_s16(sint32 , sint32);
0x0F3	sint32 Mfx_RDiv_s32s32_s32(sint32 , sint32);
0x0F4	sint8 Mfx_RDiv_u32u32_s8(uint32 , uint32);
0x0F5	sint16 Mfx_RDiv_u32u32_s16(uint32 , uint32);
0x0F6	sint32 Mfx_RDiv_u32u32_s32(uint32 , uint32);
0x0F7	sint8 Mfx_RDiv_s32u32_s8(sint32 , uint32);
0x0F8	uint16 Mfx_RDiv_s32u32_u16(sint32 , uint32);
0x0F9	uint8 Mfx_RDiv_s32u32_u8(sint32 , uint32);
0x0FA	sint16 Mfx_RDiv_u32s32_s16(uint32 , sint32);
0xoFB	sint8 Mfx_RDiv_u32s32_s8(uint32 , sint32);
0x0FC	uint16 Mfx_RDiv_u32s32_u16(uint32 , sint32);
0x0FD	uint8 Mfx_RDiv_u32s32_u8(uint32 , sint32);
0x0FE	uint16 Mfx_RDiv_u32u32_u16(uint32 , uint32);
0x0FF	uint8 Mfx_RDiv_u32u32_u8(uint32 , uint32);
0x100	sint16 Mfx_RDiv_u16u16_s16(uint16 , uint16);
0x101	uint8 Mfx_RDiv_u16u16_u8(uint16 , uint16);
0x102	uint8 Mfx_RDiv_u16s16_u8(uint16 , sint16);
0x103	uint8 Mfx_RDiv_s16u16_u8(sint16 , uint16);
0x104	sint8 Mfx_RDiv_u16u16_s8(uint16 , uint16);
0x105	sint8 Mfx_RDiv_u16s16_s8(uint16 , sint16);
0x106	sint8 Mfx_RDiv_s16u16_s8(sint16 , uint16);
0x107	sint8 Mfx_RDiv_u8u8_s8(uint8 , uint8);

8.5.8 Combinations of multiplication and division rounded towards 0

[SWS_Mfx_00039] ⌈

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]	0x108 to 0x121	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
	z_value	Third argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	These routines make a multiplication between the two arguments and a division by the third argument.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00040] ⌈These routines make a multiplication between the two arguments and a division by the third argument:

Return-value = x_value * y_value / z_value⌋()

[SWS_Mfx_00041] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00042] ⌈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⌋()

[SWS_Mfx_00044] ⌈The result is rounded towards 0.⌋()

[SWS_Mfx_00045] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype : Div
0x108	uint16 Mfx_MulDiv_s32s32s32_u16(sint32 , sint32 , sint32);
0x109	sint16 Mfx_MulDiv_s32s32s32_s16(sint32 , sint32 , sint32);
0x10A	uint16 Mfx_MulDiv_u32u32u16_u16(uint32 , uint32 , uint16);
0x10B	sint16 Mfx_MulDiv_s32s32s16_s16(sint32 , sint32 , sint16);
0x10C	uint16 Mfx_MulDiv_s16u16s16_u16(sint16 , uint16 , sint16);
0x10D	uint16 Mfx_MulDiv_s16u16u16_u16(sint16 , uint16 , uint16);

▽

△

Function ID[hex]	Function prototype : Div
0x10E	uint16 Mfx_MulDiv_u16u16u16_u16(uint16 , uint16 , uint16);
0x10F	sint16 Mfx_MulDiv_s16u16s16_s16(sint16 , uint16 , sint16);
0x110	sint16 Mfx_MulDiv_s16s16u16_s16(sint16 , sint16 , uint16);
0x111	sint16 Mfx_MulDiv_s16u16u16_s16(sint16 , uint16 , uint16);
0x112	sint16 Mfx_MulDiv_s16s16s16_s16(sint16 , sint16 , sint16);
0x113	uint32 Mfx_MulDiv_u32u32u32_u32(uint32 , uint32 , uint32);
0x114	uint32 Mfx_MulDiv_u32u32s32_u32(uint32 , uint32 , sint32);
0x115	uint32 Mfx_MulDiv_u32s32u32_u32(uint32 , sint32 , uint32);
0x116	uint32 Mfx_MulDiv_u32s32s32_u32(uint32 , sint32 , sint32);
0x117	sint32 Mfx_MulDiv_s32s32u32_s32(sint32 , sint32 , uint32);
0x118	sint32 Mfx_MulDiv_s32u32s32_s32(sint32 , uint32 , sint32);
0x119	sint32 Mfx_MulDiv_s32u32u32_s32(sint32 , uint32 , uint32);
0x11A	sint32 Mfx_MulDiv_s32s32s32_s32(sint32 , sint32 , sint32);
0x11B	uint16 Mfx_MulDiv_u32u32u32_u16(uint32 , uint32 , uint32);
0x11C	uint16 Mfx_MulDiv_u16s16s16_u16(uint16 , sint16 , sint16);
0x11D	uint16 Mfx_MulDiv_u16s16u16_u16(uint16 , sint16 , uint16);
0x11E	sint16 Mfx_MulDiv_u16s16s16_s16(uint16 , sint16 , sint16);
0x11F	sint16 Mfx_MulDiv_u16s16u16_s16(uint16 , sint16 , uint16);
0x120	sint32 Mfx_MulDiv_u32s32s32_s32(uint32 , sint32 , sint32);
0x121	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

[SWS_Mfx_00046] ⌈

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]	0x122 to 0x13B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
	z_value	Third argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	These routines make a multiplication between the two arguments and a division by the third argument.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00047] ⌈These routines make a multiplication between the two arguments and a division by the third argument:

Return-value = x_value * y_value / z_value⌋*()*

[SWS_Mfx_00048] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00049] ⌈The result after division by zero is defined by:

* If x_value*y_value $\geq$ 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⌋*()*

[SWS_Mfx_00051] ⌈The result is rounded off.⌋*()*

[SWS_Mfx_00052] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype : RDiv
0x122	uint16 Mfx_RMulDiv_s32s32s32_u16(sint32 , sint32 , sint32);
0x123	sint16 Mfx_RMulDiv_s32s32s32_s16(sint32 , sint32 , sint32);

▽

△

Function ID[hex]	Function prototype : RDiv
0x124	uint16 Mfx_RMulDiv_u32u32u16_u16(uint32 , uint32 , uint16);
0x125	sint16 Mfx_RMulDiv_s32s32s16_s16(sint32 , sint32 , sint16);
0x126	uint16 Mfx_RMulDiv_s16u16s16_u16(sint16 , uint16 , sint16);
0x127	uint16 Mfx_RMulDiv_s16u16u16_u16(sint16 , uint16 , uint16);
0x128	uint16 Mfx_RMulDiv_u16u16u16_u16(uint16 , uint16 , uint16);
0x129	sint16 Mfx_RMulDiv_s16u16s16_s16(sint16 , uint16 , sint16);
0x12A	sint16 Mfx_RMulDiv_s16s16u16_s16(sint16 , sint16 , uint16);
0x12B	sint16 Mfx_RMulDiv_s16u16u16_s16(sint16 , uint16 , uint16);
0x12C	sint16 Mfx_RMulDiv_s16s16s16_s16(sint16 , sint16 , sint16);
0x12D	uint32 Mfx_RMulDiv_u32u32u32_u32(uint32 , uint32 , uint32);
0x12E	uint32 Mfx_RMulDiv_u32u32s32_u32(uint32 , uint32 , sint32);
0x12F	uint32 Mfx_RMulDiv_u32s32u32_u32(uint32 , sint32 , uint32);
0x130	uint32 Mfx_RMulDiv_u32s32s32_u32(uint32 , sint32 , sint32);
0x131	sint32 Mfx_RMulDiv_s32s32u32_s32(sint32 , sint32 , uint32);
0x132	sint32 Mfx_RMulDiv_s32u32s32_s32(sint32 , uint32 , sint32);
0x133	sint32 Mfx_RMulDiv_s32u32u32_s32(sint32 , uint32 , uint32);
0x134	sint32 Mfx_RMulDiv_s32s32s32_s32(sint32 , sint32 , sint32);
0x135	uint16 Mfx_RMulDiv_u32u32u32_u16(uint32 , uint32 , uint32);
0x136	uint16 Mfx_RMulDiv_u16s16s16_u16(uint16 , sint16 , sint16);
0x137	uint16 Mfx_RMulDiv_u16s16u16_u16(uint16 , sint16 , uint16);
0x138	sint16 Mfx_RMulDiv_u16s16s16_s16(uint16 , sint16 , sint16);
0x139	sint16 Mfx_RMulDiv_u16s16u16_s16(uint16 , sint16 , uint16);
0x13A	sint32 Mfx_RMulDiv_u32s32s32_s32(uint32 , sint32 , sint32);
0x13B	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

[SWS_Mfx_00053] ⌈

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]	0x13C to 0x151	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
	shift	Third argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Result of the calculation
Description	This routine makes a multiplication between the two arguments and applies a shift right defined by the third argument.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00054] ⌈This routine makes a multiplication between the two arguments and applies a shift right defined by the third argument:

Return-value = (x_value * y_value) >> shift⌋()

[SWS_Mfx_00055] ⌈We precise that for the shift right of a negative number, we always keep the bit of sign.⌋()

[SWS_Mfx_00056] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00057] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype	Associated maximum shift
0x13C	uint8 Mfx_MulShRight_s16s16u8_u8(sint16 , sint16 , uint8);	30
0x13D	sint8 Mfx_MulShRight_s16s16u8_s8(sint16 , sint16 , uint8);	30
0x13E	sint16 Mfx_MulShRight_s16s16u8_ s16(sint16 , sint16 , uint8);	30
0x13F	uint16 Mfx_MulShRight_s16s16u8_ u16(sint16 , sint16 , uint8);	30
0x140	uint8 Mfx_MulShRight_u32s32u8_u8(uint32 , sint32 , uint8);	63

▽

△

Function ID[hex]	Function prototype	Associated maximum shift
0x141	sint8 Mfx_MulShRight_u32s32u8_s8(uint32 , sint32 , uint8);	63
0x142	uint16 Mfx_MulShRight_u32s32u8_ u16(uint32 , sint32 , uint8);	63
0x143	sint16 Mfx_MulShRight_u32s32u8_ s16(uint32 , sint32 , uint8);	63
0x144	uint32 Mfx_MulShRight_u32s32u8_ u32(uint32 , sint32 , uint8);	63
0x145	sint32 Mfx_MulShRight_u32s32u8_ s32(uint32 , sint32 , uint8);	63
0x146	sint8 Mfx_MulShRight_s32s32u8_s8(sint32 , sint32 , uint8);	62
0x147	uint8 Mfx_MulShRight_s32s32u8_u8(sint32 , sint32 , uint8);	62
0x148	sint16 Mfx_MulShRight_s32s32u8_ s16(sint32 , sint32 , uint8);	62
0x149	uint16 Mfx_MulShRight_s32s32u8_ u16(sint32 , sint32 , uint8);	62
0x14A	uint32 Mfx_MulShRight_s32s32u8_ u32(sint32 , sint32 , uint8);	62
0x14B	sint32 Mfx_MulShRight_s32s32u8_ s32(sint32 , sint32 , uint8);	62
0x14C	uint8 Mfx_MulShRight_u32u32u8_u8(uint32 , uint32 , uint8);	63
0x14D	sint8 Mfx_MulShRight_u32u32u8_s8(uint32 , uint32 , uint8);	63
0x14E	uint16 Mfx_MulShRight_u32u32u8_ u16(uint32 , uint32 , uint8);	63
0x14F	sint16 Mfx_MulShRight_u32u32u8_ s16(uint32 , uint32 , uint8);	63
0x150	uint32 Mfx_MulShRight_u32u32u8_ u32(uint32 , uint32 , uint8);	63
0x151	sint32 Mfx_MulShRight_u32u32u8_ s32(uint32 , uint32 , uint8);	63

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

8.5.11 Combinations of division and shift left

[SWS_Mfx_00058] ⌈

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]	0x152 to 0x16E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Numerator
	y_value	Denominator
	shift	Shift left of the fixed point result. Must be a constant expression. Maximum shift according to SWS_Mfx_00064
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Quotient result
Description	This routine applies a shift left defined by the third argument to the first argument, and then makes a division by the second argument.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00059] ⌈This routine applies a shift left defined by the third argument to the first argument, and then makes a division by the second argument:

Return-value = (x_value << shift) / y_value⌋()

[SWS_Mfx_00060] ⌈We precise that for the shift left of a negative number, we always keep the bit of sign.⌋()

[SWS_Mfx_00061] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00062] ⌈The result after division by zero is defined by:

* If x_value ≥ 0 then the function returns the maximum value of the output type

* If x_value < 0 then the function returns the minimum value of the output type⌋()

[SWS_Mfx_00064] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype	Associated maximum shift
0x152	uint8 Mfx_DivShLeft_u8u8u8_u8(uint8 , uint8 , uint8);	7
0x153	uint8 Mfx_DivShLeft_u16u16u8_u8(uint16 , uint16 , uint8);	15
0x154	uint16 Mfx_DivShLeft_u16u16u8_u16(uint16 , uint16 , uint8);	15
0x155	sint16 Mfx_DivShLeft_s16s16u8_s16(sint16 , sint16 , uint8);	15
0x156	sint16 Mfx_DivShLeft_s16u16u8_s16(sint16 , uint16 , uint8);	15
0x157	uint16 Mfx_DivShLeft_u32u32u8_u16(uint32 , uint32 , uint8);	31
0x158	uint32 Mfx_DivShLeft_u32u32u8_u32(uint32 , uint32 , uint8);	31

▽

△

Function ID[hex]	Function prototype	Associated maximum shift
0x159	sint32 Mfx_DivShLeft_s32s32u8_s32(sint32 , sint32 , uint8);	31
0x15A	sint32 Mfx_DivShLeft_s32u32u8_s32(sint32 , uint32 , uint8);	31
0x15B	uint8 Mfx_DivShLeft_u32s32u8_u8(uint32 , sint32 , uint8);	31
0x15C	sint8 Mfx_DivShLeft_u32s32u8_s8(uint32 , sint32 , uint8);	31
0x15D	uint16 Mfx_DivShLeft_u32s32u8_u16(uint32 , sint32 , uint8);	31
0x15E	sint16 Mfx_DivShLeft_u32s32u8_s16(uint32 , sint32 , uint8);	31
0x15F	uint32 Mfx_DivShLeft_u32s32u8_u32(uint32 , sint32 , uint8);	31
0x160	sint32 Mfx_DivShLeft_u32s32u8_s32(uint32 , sint32 , uint8);	31
0x161	sint8 Mfx_DivShLeft_s32s32u8_s8(sint32 , sint32 , uint8);	31
0x162	uint8 Mfx_DivShLeft_s32s32u8_u8(sint32 , sint32 , uint8);	31
0x163	sint16 Mfx_DivShLeft_s32s32u8_s16(sint32 , sint32 , uint8);	31
0x164	uint16 Mfx_DivShLeft_s32s32u8_u16(sint32 , sint32 , uint8);	31
0x165	uint32 Mfx_DivShLeft_s32s32u8_u32(sint32 , sint32 , uint8);	31
0x166	uint8 Mfx_DivShLeft_u32u32u8_u8(uint32 , uint32 , uint8);	31
0x167	sint8 Mfx_DivShLeft_u32u32u8_s8(uint32 , uint32 , uint8);	31
0x168	sint16 Mfx_DivShLeft_u32u32u8_s16(uint32 , uint32 , uint8);	31
0x169	sint32 Mfx_DivShLeft_u32u32u8_s32(uint32 , uint32 , uint8);	31
0x16A	uint8 Mfx_DivShLeft_s32u32u8_u8(sint32 , uint32 , uint8);	31
0x16B	sint8 Mfx_DivShLeft_s32u32u8_s8(sint32 , uint32 , uint8);	31
0x16C	uint16 Mfx_DivShLeft_s32u32u8_u16(sint32 , uint32 , uint8);	31
0x16D	sint16 Mfx_DivShLeft_s32u32u8_s16(sint32 , uint32 , uint8);	31
0x16E	uint32 Mfx_DivShLeft_s32u32u8_u32(sint32 , uint32 , uint8);	31

If you want to see an example of the use of these functions, see subsection 8.7.2 .

8.5.12 Modulo

[SWS_Mfx_00065] ⌈

Service Name	Mfx_Mod_<TypeMn>	

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

Service ID [hex]	0x16F to 0x178	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<Type>	Result of the calculation
Description	This routine returns the remainder of the division x_value / y_value if y_value is not zero.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00066] ⌈If y_value is zero, the result is zero.⌋()

[SWS_Mfx_00068] ⌈In other cases, Return-value = x_value mod y_value⌋()

[SWS_Mfx_00069] ⌈The sign of the remainder is the same than the sign of x_value.⌋()

[SWS_Mfx_00070] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x16F	uint8 Mfx_Mod_u8(uint8 , uint8);
0x170	sint8 Mfx_Mod_s8(sint8 , sint8);
0x171	uint16 Mfx_Mod_u16(uint16 , uint16);
0x172	sint16 Mfx_Mod_s16(sint16 , sint16);
0x173	uint32 Mfx_Mod_u32(uint32 , uint32);
0x174	sint32 Mfx_Mod_s32(sint32 , sint32);
0x175	uint8 Mfx_Mod_u32u32_u8(uint32 , uint32)
0x176	sint8 Mfx_Mod_s32s32_s8(sint32 , sint32)
0x177	uint16 Mfx_Mod_u32u32_u16(uint32 , uint32)
0x178	sint16 Mfx_Mod_s32s32_s16(sint32 , sint32)

8.5.13 Limiting

[SWS_Mfx_00073] ⌈

Service Name	Mfx_Limit_<TypeMn>	

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

Service ID [hex]	0x179 to 0x17E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	input value.
	min_value	Lower Bound. min_value shall not be strictly greater than max_value.
	max_value	Upper Bound. max_value shall not be strictly lower than min_value.
Parameters (inout)	None	
Parameters (out)	None	
Return value	<Type>	Result of the calculation
Description	This routine limits the input value between Lower Bound and Upper Bound.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00074] ⌈Return-value = min_value if value < min_value⌋()

[SWS_Mfx_00075] ⌈Return-value = max_value if value > max_value⌋()

[SWS_Mfx_00076] ⌈Return-value = value in the other cases⌋()

[SWS_Mfx_00079] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x179	uint8 Mfx_Limit_u8(uint8 , uint8, uint8);
0x17A	sint8 Mfx_Limit_s8(sint8 , sint8, sint8);
0x17B	uint16 Mfx_Limit_u16(uint16 , uint16, uint16);
0x17C	sint16 Mfx_Limit_s16(sint16 , sint16, sint16);
0x17D	uint32 Mfx_Limit_u32(uint32 , uint32, uint32);
0x17E	sint32 Mfx_Limit_s32(sint32 , sint32, sint32);

8.5.14 Limitations with only one value for minimum and maximum

[SWS_Mfx_00082] ⌈

Service Name	Mfx_Minmax_<TypeMn>	

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

Service ID [hex]	0x17F to 0x184	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	First argument
	minmax_value	Second argument
Parameters (inout)	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.	
Available via	Mfx.h	

⌋() The result value is :

[SWS_Mfx_00083] ⌈minmax_value if minmax_value $\geq$ 0 and value > minmax_value⌋ ()

[SWS_Mfx_00084] ⌈minmax_value if minmax_value < 0 and value < minmax_value⌋()

[SWS_Mfx_00085] ⌈value in the other cases⌋()

[SWS_Mfx_00086] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x17F	uint8 Mfx_Minmax_u8(uint8 , uint8);
0x180	sint8 Mfx_Minmax_s8(sint8 , sint8);
0x181	uint16 Mfx_Minmax_u16(uint16 , uint16);
0x182	sint16 Mfx_Minmax_s16(sint16 , sint16);
0x183	uint32 Mfx_Minmax_u32(uint32 , uint32);
0x184	sint32 Mfx_Minmax_s32(sint32 , sint32);

8.5.15 Minimum and maximum

[SWS_Mfx_00090] ⌈

Service Name	Mfx_Min_<TypeMn>	
Syntax	`<Type> Mfx_Min_<TypeMn> ( <Type> x_value, <Type> y_value )`	
Service ID [hex]	0x185 to 0x18A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<Type>	Result of the calculation
Description	This routine returns the minimum between two values.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00091] ⌈Return-value = x_value if x_value < y_value⌋()

[SWS_Mfx_00092] ⌈Return-value = y_value in the other case⌋()

[SWS_Mfx_00093] ⌈

Service Name	Mfx_Max_<TypeMn>	
Syntax	`<Type> Mfx_Max_<TypeMn> ( <Type> x_value, <Type> y_value )`	
Service ID [hex]	0x18B to 0x190	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument
	y_value	Second argument
Parameters (inout)	None	
Parameters (out)	None	
Return value	<Type>	Result of the calculation
Description	This routine returns the maximum between two values.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00094] ⌈Return-value = x_value if x_value > y_value⌋()

[SWS_Mfx_00095] ⌈Return-value = y_value in the other case⌋()

[SWS_Mfx_00096] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x185	uint8 Mfx_Min_u8(uint8 , uint8);
0x186	sint8 Mfx_Min_s8(sint8 , sint8);
0x187	uint16 Mfx_Min_u16(uint16 , uint16);
0x188	sint16 Mfx_Min_s16(sint16 , sint16);
0x189	uint32 Mfx_Min_u32(uint32 , uint32);
0x18A	sint32 Mfx_Min_s32(sint32 , sint32);
0x18B	uint8 Mfx_Max_u8(uint8 , uint8);
0x18C	sint8 Mfx_Max_s8(sint8 , sint8);
0x18D	uint16 Mfx_Max_u16(uint16 , uint16);
0x18E	sint16 Mfx_Max_s16(sint16 , sint16);
0x18F	uint32 Mfx_Max_u32(uint32 , uint32);
0x190	sint32 Mfx_Max_s32(sint32 , sint32);

8.6 2n 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 :

- "x" 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 2n Scaled Integer Conversion

[SWS_Mfx_00100] ⌈

Service Name	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>	

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

Service ID [hex]	0x191 to 0x192	
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 (inout)	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.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00101] ⌈The function returns the integer value of the fixed point conversion (C), determined by C = 2^(c-a) * x.⌋()

[SWS_Mfx_00102] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00103] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋()

[SWS_Mfx_00104] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x191	uint8 Mfx_ConvertP2_u16_u8(uint16 x, sint16 a, sint16 c)
0x192	sint8 Mfx_ConvertP2_s16_s8(sint16 x, sint16 a, sint16 c)

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

[SWS_Mfx_00106] ⌈

Service Name	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>	
Syntax	`<OutType> Mfx_ConvertP2_<InTypeMn>_<OutTypeMn> ( <InType> x, sint16 a, sint16 c )`	
Service ID [hex]	0x193 to 0x194	
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 (inout)	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.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00107] ⌈The function returns the integer value of the fixed point conversion (C), determined by C = 2^(c-a) * x.⌋*()*

[SWS_Mfx_00108] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00109] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00110] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x193	uint16 Mfx_ConvertP2_u8_u16(uint8 x, sint16 a, sint16 c)
0x194	sint16 Mfx_ConvertP2_s8_s16 (sint8 x, sint16 a, sint16 c)

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

[SWS_Mfx_00112] ⌈

Service Name	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>	
Syntax	`<OutType> Mfx_ConvertP2_<InTypeMn>_<OutTypeMn> ( <InType> x, sint16 a, sint16 c )`	
Service ID [hex]	0x195 to 0x196	
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 (inout)	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.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00113] ⌈The function returns the integer value of the fixed point conversion (C), determined by C = 2^(c-a) * x.⌋()

[SWS_Mfx_00114] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00115] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋()

[SWS_Mfx_00116] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x195	uint16 Mfx_ConvertP2_u32_u16 (uint32 x, sint16 a, sint16 c)
0x196	sint16 Mfx_ConvertP2_s32_s16 (sint32 x, sint16 a, sint16 c)

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

[SWS_Mfx_00118] ⌈

Service Name	Mfx_ConvertP2_<InTypeMn>_<OutTypeMn>	
Syntax	`<OutType> Mfx_ConvertP2_<InTypeMn>_<OutTypeMn> ( <InType> x, sint16 a, sint16 c )`	
Service ID [hex]	0x197 to 0x198	
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 (inout)	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.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00119] ⌈The function returns the integer value of the fixed point conversion (C), determined by C = 2^(c-a) * x.⌋*()*

[SWS_Mfx_00120] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00121] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00122] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x197	uint32 Mfx_ConvertP2_u16_u32(uint16 x, sint16 a, sint16 c)
0x198	sint32 Mfx_ConvertP2_s16_s32(sint16 x, sint16 a, sint16 c)

8.6.2 Multiplication

8.6.2.1 16-Bit Multiplication of 2n Scaled Integer

[SWS_Mfx_00124] ⌈

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]	0x199 to 0x19E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00125] ⌈The function returns the integer value of the fixed point multiplication (C), determined by C = 2^(c-b-a) * [x * y].⌋*()*

[SWS_Mfx_00126] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00127] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00128] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x199	uint16 Mfx_MulP2_u16u16_u16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x19A	uint16 Mfx_MulP2_u16s16_u16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)

▽

△

Function ID[hex]	Function prototype
0x19B	uint16 Mfx_MulP2_s16s16_u16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x19C	sint16 Mfx_MulP2_u16u16_s16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x19D	sint16 Mfx_MulP2_u16s16_s16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x19E	sint16 Mfx_MulP2_s16s16_s16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)

8.6.2.2 32-Bit Multiplication of 2n Scaled Integer

[SWS_Mfx_00130] ⌈

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]	0x19F to 0x1A4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00131] ⌈The function returns the integer value of the fixed point multiplication (C), determined by C = 2^(c-b-a) * [x * y].⌋*()*

[SWS_Mfx_00132] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00133] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00134] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x19F	uint32 Mfx_MulP2_u32u32_u32(uint32 x, uint32 y, sint16 a, sint16 b, sint16 c)
0x1A0	uint32 Mfx_MulP2_u32s32_u32(uint32 x, sint32 y, sint16 a, sint16 b, sint16 c)
0x1A1	uint32 Mfx_MulP2_s32s32_u32(sint32 x, sint32 y, sint16 a, sint16 b, sint16 c)
0x1A2	sint32 Mfx_MulP2_u32u32_s32(uint32 x, uint32 y, sint16 a, sint16 b, sint16 c)
0x1A3	sint32 Mfx_MulP2_u32s32_s32(uint32 x, sint32 y, sint16 a, sint16 b, sint16 c)
0x1A4	sint32 Mfx_MulP2_s32s32_s32(sint32 x, sint32 y, sint16 a, sint16 b, sint16 c)

8.6.3 Division

8.6.3.1 16-Bit Division of 2n Scaled Integer

[SWS_Mfx_00136] ⌈

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]	0x1A5 to 0x1AC	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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.	

▽

△

Available via	Mfx.h

⌋()

[SWS_Mfx_00137] ⌈The function returns the integer value of the fixed point quotient (C), determined by C = [2^(c+b-a) * x] / y.⌋()

[SWS_Mfx_00138] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00139] ⌈If the divisor, y, is zero, the result is defined by:

* If x ≥ 0 then the function returns the maximum value of the output type

* If x < 0 then the function returns the minimum value of the output type⌋()

[SWS_Mfx_00141] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋()

[SWS_Mfx_00142] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x1A5	uint16 Mfx_DivP2_u16u16_u16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1A6	uint16 Mfx_DivP2_u16s16_u16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1A7	uint16 Mfx_DivP2_s16u16_u16(sint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1A8	uint16 Mfx_DivP2_s16s16_u16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1A9	sint16 Mfx_DivP2_u16u16_s16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1AA	sint16 Mfx_DivP2_u16s16_s16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1AB	sint16 Mfx_DivP2_s16u16_s16(sint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1AC	sint16 Mfx_DivP2_s16s16_s16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)

8.6.3.2 32-Bit Division of 2n Scaled Integer

[SWS_Mfx_00144] ⌈

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]	0x1AD to 0x1B4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00145] ⌈The function returns the integer value of the fixed point quotient (C), determined by C = [2^(c+b-a) * x] / y.⌋*()*

[SWS_Mfx_00146] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00147] ⌈If the divisor, y, is zero, the result is defined by:

* If x $\geq$ 0 then the function returns the maximum value of the output type

* If x < 0 then the function returns the minimum value of the output type⌋*()*

[SWS_Mfx_00149] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00150] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x1AD	uint32 Mfx_DivP2_u32u32_u32(uint32 x, uint32 y, sint16 a, sint16 b, sint16 c)
0x1AE	uint32 Mfx_DivP2_u32s32_u32(uint32 x, sint32 y, sint16 a, sint16 b, sint16 c)
0x1AF	uint32 Mfx_DivP2_s32u32_u32(sint32 x, uint32 y, sint16 a, sint16 b, sint16 c)
0x1B0	uint32 Mfx_DivP2_s32s32_u32(sint32 x, sint32 y, sint16 a, sint16 b, sint16 c)
0x1B1	sint32 Mfx_DivP2_u32u32_s32(uint32 x, uint32 y, sint16 a, sint16 b, sint16 c)
0x1B2	sint32 Mfx_DivP2_u32s32_s32(uint32 x, sint32 y, sint16 a, sint16 b, sint16 c)
0x1B3	sint32 Mfx_DivP2_s32u32_s32(sint32 x, uint32 y, sint16 a, sint16 b, sint16 c)
0x1B4	sint32 Mfx_DivP2_s32s32_s32(sint32 x, sint32 y, sint16 a, sint16 b, sint16 c)

8.6.4 Addition

8.6.4.1 16-Bit Addition of 2n Scaled Integer

[SWS_Mfx_00152] ⌈

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]	0x1B5 to 0x1BA	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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.
Available via	Mfx.h

⌋()

[SWS_Mfx_00153] ⌈The function returns the integer value of the fixed point sum (C), determined by

a ≥ b: C = 2^(c-a) * [x + (y * 2^(a-b))],

a < b: C = 2^(c-b) * [(x * 2^(b-a)) + y].⌋()

[SWS_Mfx_00154] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00155] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋()

[SWS_Mfx_00156] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x1B5	uint16 Mfx_AddP2_u16u16_u16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1B6	uint16 Mfx_AddP2_u16s16_u16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1B7	uint16 Mfx_AddP2_s16s16_u16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1B8	sint16 Mfx_AddP2_u16u16_s16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1B9	sint16 Mfx_AddP2_u16s16_s16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1BA	sint16 Mfx_AddP2_s16s16_s16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)

8.6.4.2 32-Bit Addition of 2n Scaled Integer

[SWS_Mfx_00158] ⌈

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]	0x1BB to 0x1C0	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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 32-bit integers with scaling factors set by input parameters.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00159] ⌈The function returns the integer value of the fixed point sum (C), determined by

a ≥ b: C = 2^(c-a) * [x + (y * 2^(a-b))],

a < b: C = 2^(c-b) * [(x * 2^(b-a)) + y]⌋*()*

[SWS_Mfx_00160] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00161] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00162] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x1BB	uint32 Mfx_AddP2_u32u32_u32(uint32 x, uint32 y, sint32 a, sint32 b, sint32 c)
0x1BC	uint32 Mfx_AddP2_u32s32_u32(uint32 x, sint32 y, sint32 a, sint32 b, sint32 c)
0x1BD	uint32 Mfx_AddP2_s32s32_u32(sint32 x, sint32 y, sint32 a, sint32 b, sint32 c)
0x1BE	sint32 Mfx_AddP2_u32u32_s32(uint32 x, uint32 y, sint32 a, sint32 b, sint32 c)
0x1BF	sint32 Mfx_AddP2_u32s32_s32(uint32 x, sint32 y, sint32 a, sint32 b, sint32 c)
0x1C0	sint32 Mfx_AddP2_s32s32_s32(sint32 x, sint32 y, sint32 a, sint32 b, sint32 c)

8.6.5 Subtraction

8.6.5.1 16-Bit Subtraction of 2n Scaled Integer

[SWS_Mfx_00164] ⌈

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]	0x1C1 to 0x1C8	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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 two 16-bit integers with scaling factors set by input parameters.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00165] ⌈The function returns the integer value of the fixed point difference (C), determined by

a ≥ b: C = 2^(c-a) * [x - (y * 2^(a-b))],

a < b: C = 2^(c-b) * [(x * 2^(b-a)) - y]⌋*()*

[SWS_Mfx_00166] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00167] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00168] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x1C1	uint16 Mfx_SubP2_u16u16_u16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1C2	uint16 Mfx_SubP2_u16s16_u16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1C3	uint16 Mfx_SubP2_s16u16_u16(sint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1C4	uint16 Mfx_SubP2_s16s16_u16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1C5	sint16 Mfx_SubP2_u16u16_s16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1C6	sint16 Mfx_SubP2_u16s16_s16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1C7	sint16 Mfx_SubP2_s16u16_s16(sint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1C8	sint16 Mfx_SubP2_s16s16_s16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)

8.6.5.2 32-Bit Subtraction of 2n Scaled Integer

[SWS_Mfx_00170] ⌈

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]	0x1C9 to 0x1D0	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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 two 32-bit integers with scaling factors set by input parameters.	
Available via	Mfx.h	

⌋()

[SWS_Mfx_00171] ⌈The function returns the integer value of the fixed point difference (C), determined by

a ≥ b: C = 2^(c-a) * [x - (y * 2^(a-b))],

a < b: C = 2^(c-b) * [(x * 2^(b-a)) - y].⌋()

[SWS_Mfx_00172] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00173] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋()

[SWS_Mfx_00174] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x1C9	uint32 Mfx_SubP2_u32u32_u32(uint32 x, uint32 y, sint32 a, sint32 b, sint32 c)
0x1CA	uint32 Mfx_SubP2_u32s32_u32(uint32 x, sint32 y, sint32 a, sint32 b, sint32 c)
0x1CB	uint32 Mfx_SubP2_s32u32_u32(sint32 x, uint32 y, sint32 a, sint32 b, sint32 c)
0x1CC	uint32 Mfx_SubP2_s32s32_u32(sint32 x, sint32 y, sint32 a, sint32 b, sint32 c)
0x1CD	sint32 Mfx_SubP2_u32u32_s32(uint32 x, uint32 y, sint32 a, sint32 b, sint32 c)
0x1CE	sint32 Mfx_SubP2_u32s32_s32(uint32 x, sint32 y, sint32 a, sint32 b, sint32 c)
0x1CF	sint32 Mfx_SubP2_s32u32_s32(sint32 x, uint32 y, sint32 a, sint32 b, sint32 c)
0x1D0	sint32 Mfx_SubP2_s32s32_s32(sint32 x, sint32 y, sint32 a, sint32 b, sint32 c)

8.6.6 Absolute Difference of 2n Scaled Integer

[SWS_Mfx_00176] ⌈

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]	0x1D1 to 0x1D6	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	y	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	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.
Available via	Mfx.h

⌋()

[SWS_Mfx_00177] ⌈The function returns the integer value of the fixed point absolute difference (C), determined by

a ≥ b: C = 2^(c-a) * |x - (y * 2^(a-b))|,

a < b: C = 2^(c-b) * |(x * 2^(b-a)) - y|.⌋()

[SWS_Mfx_00178] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋()

[SWS_Mfx_00179] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋()

[SWS_Mfx_00180] ⌈Here is the list of implemented functions.⌋()

Function ID[hex]	Function prototype
0x1D1	uint16 Mfx_AbsDiffP2_u16u16_u16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1D2	uint16 Mfx_AbsDiffP2_u16s16_u16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1D3	uint16 Mfx_AbsDiffP2_s16s16_u16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1D4	sint16 Mfx_AbsDiffP2_u16u16_s16(uint16 x, uint16 y, sint16 a, sint16 b, sint16 c)
0x1D5	sint16 Mfx_AbsDiffP2_u16s16_s16(uint16 x, sint16 y, sint16 a, sint16 b, sint16 c)
0x1D6	sint16 Mfx_AbsDiffP2_s16s16_s16(sint16 x, sint16 y, sint16 a, sint16 b, sint16 c)

8.6.7 Absolute Value

8.6.7.1 16-Bit Absolute Value of 2n Scaled Integer

[SWS_Mfx_00182] ⌈

Service Name	Mfx_AbsP2_s16_<OutTypeMn>	

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

Service ID [hex]	0x1D7 to 0x1D8	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	None	
Parameters (out)	None	
Return value	<OutType>	2^(c-a) * |x|
Description	The routine takes the absolute value of a 16-bit integer with scaling factors set by input parameters.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00183] ⌈The function returns the integer value of the fixed point absolute value (C), deter-mined by C = 2^(c-a) * |x|.⌋*()*

[SWS_Mfx_00184] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00185] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00186] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype () ()
0x1D7	uint16 Mfx_AbsP2_s16_u16(sint16 x, sint16 a, sint16 c)
0x1D8	sint16 Mfx_AbsP2_s16_s16(sint16 x, sint16 a, sint16 c)

8.6.7.2 32-Bit Absolute Value of 2n Scaled Integer

[SWS_Mfx_00188] ⌈

Service Name	Mfx_AbsP2_s32_<OutTypeMn>	

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

Service ID [hex]	0x1D9 to 0x1DA	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x	Integer value of the fixed point operand.
	a	Radix point position 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 (inout)	None	
Parameters (out)	None	
Return value	<OutType>	2^(c-a) * |x|
Description	The routine takes the absolute value of a 32-bit integer with scaling factors set by input parameters.	
Available via	Mfx.h	

⌋*()*

[SWS_Mfx_00189] ⌈The function returns the integer value of the fixed point absolute value (C), deter-mined by C = 2^(c-a) * |x|.⌋*()*

[SWS_Mfx_00190] ⌈Return-value shall be saturated to boundary values in the event of negative or positive overflow.⌋*()*

[SWS_Mfx_00191] ⌈If it is necessary to round the result of this equation, it is rounded toward zero.⌋*()*

[SWS_Mfx_00192] ⌈Here is the list of implemented functions.⌋*()*

Function ID[hex]	Function prototype
0x1D9	uint32 Mfx_AbsP2_s32_u32(sint32 x, sint16 a, sint16 c)
0x1DA	sint32 Mfx_AbsP2_s32_s32(sint32 x, 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 : 2n1

If we consider the maximum of the type used to code the factor : 2n2-1

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 (20) coded with an u8 (Max(u8)=28-1).

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 : 2n1

If we consider the maximum of the type used to code the result (factor) : 2n2-1

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 (20) coded with an u16 (Max(u16)=216-1).

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

[SWS_Mfx_00215] ⌈

Service Name	Mfx_GetVersionInfo	
Syntax	`void Mfx_GetVersionInfo ( Std_VersionInfoType* versioninfo )`	
Service ID [hex]	0x1DB	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versioninfo	Pointer to where to store the version information of this module. Format according [BSW00321]
Return value	None	
Description	Returns the version information of this library.	
Available via	Mfx.h	

⌋*(SRS_BSW_00407, SRS_BSW_00003, SRS_BSW_00318, SRS_BSW_00321)* The version information of a BSW module generally contains:

Module Id

Vendor Id

Vendor specific version numbers (SRS_BSW_00407).

[SWS_Mfx_00216] ⌈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.⌋*(SRS_BSW_00407, SRS_BSW_00411)*

8.9 Callback 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.

8.12 Service Interfaces

None.

9 Sequence diagrams

Not applicable.

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module MFXLibrary.

Chapter 10.3 specifies published information of the module MFXLibrary.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in SWS_BSWGeneral.

10.2 Containers and configuration parameters

[SWS_Mfx_00218] ⌈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.⌋*(SRS_LIBS_00001)*

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

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in SWS_BSWGeneral.

[SWS_Mfx_00214] ⌈The standardized common published parameters as required by SRS_BSW_00402 in the SRS General on Basic Software Modules [chapter 3.1] 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 [chapter 3.1] .⌋*(SRS_BSW_00402, SRS_BSW_00374, SRS_BSW_00379)*

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

A Not applicable requirements

[SWS_Mfx_00222] ⌈These requirements are not applicable to this specification.⌋*()*