Document Title	Specification of Bit Handling Library
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	399

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R25-11

Document Change History

Date	Release	Changed by	Description
2025-11-27	R25-11	AUTOSAR Release Management	• Editorial changes
2024-11-27	R24-11	AUTOSAR Release Management	• No content change
2023-11-23	R23-11	AUTOSAR Release Management	• No content change
2022-11-24	R22-11	AUTOSAR Release Management	• New Functions added: SWS_Bfx_91002, SWS_Bfx_00134, SWS_Bfx_00135, SWS_Bfx_91003, SWS_Bfx_00137, SWS_Bfx_91004, SWS_Bfx_00139, SWS_Bfx_91005 and SWS_Bfx_00141.
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes (only converted to LaTex) • Artifact inclusion based on ArtifactAnalysis corrected
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	• Addition of 64bit handling requirement
2017-12-08	4.3.1	AUTOSAR Release Management	• Addition on mnemonic for boolean as “u8“ • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Removal of the requirement SWS_Bfx_00204 • Updation of MISRA violation comment format • Updation of unspecified value range for BitPn, BitStartPn, BitLn and ShiftCnt • Clarifications
2015-07-31	4.2.2	AUTOSAR Release Management	• Updated SWS_Bfx_00017 for the return type of Bfx_GetBit function from 1 and 0 to TRUE and FALSE • Updated chapter 8.1 for the definition of bit addressing and updated the examples of Bfx_SetBit, Bfx_ClrBit, Bfx_GetBit, Bfx_SetBits, Bfx_CopyBit, Bfx_PutBits, Bfx_PutBit • Updated SWS_Bfx_00017 for the return type of Bfx_GetBit function from 1 and 0 to TRUE and FALSE without changing the formula • Updated SWS_Bfx_00011 and SWS_Bfx_00022 for the review comments provided for the examples • In Table 2, replaced Boolean with boolean • In SWS_Bfx_00029, in example re-placed BFX_GetBits_u16u8u8_u16 with Bfx_GetBits_u16u8u8_u16
2014-10-31	4.2.1	AUTOSAR Release Management	• Correct usage of const in function declarations • Editoral changes

▽

△

2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial Changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Improve description of how to map functions to C-files • Improve the definition of error classification • Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Change return value of Test Bit API to boolean. • Improve memory map handling • Change number of parameter in Put Bit Api.
2011-12-22	4.0.3	AUTOSAR Administration	• Requirements described with more clarity for ‘Bit Shift and Rotate‘ operations • Table correction for PutBit routines • ‘Copy Bit routine‘ interfaces corrected • Error classification support and definition removed as DET call not supported by library • Configuration parameter description / support removed for XXX_ GetVersionInfo routine • Renaming of the term DET in the abbreviation to “Default Error Trace“
2009-12-18	4.0.1	AUTOSAR Administration	• Signature for necessary Bit handling functions optimized for easy usage • Bit handling on all signed variables eliminated • Additional bit handling functions introduced
2010-02-02	3.1.4	AUTOSAR Administration	• Initial Release

Disclaimer

Table of Contents

1 Introduction and functional overview

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

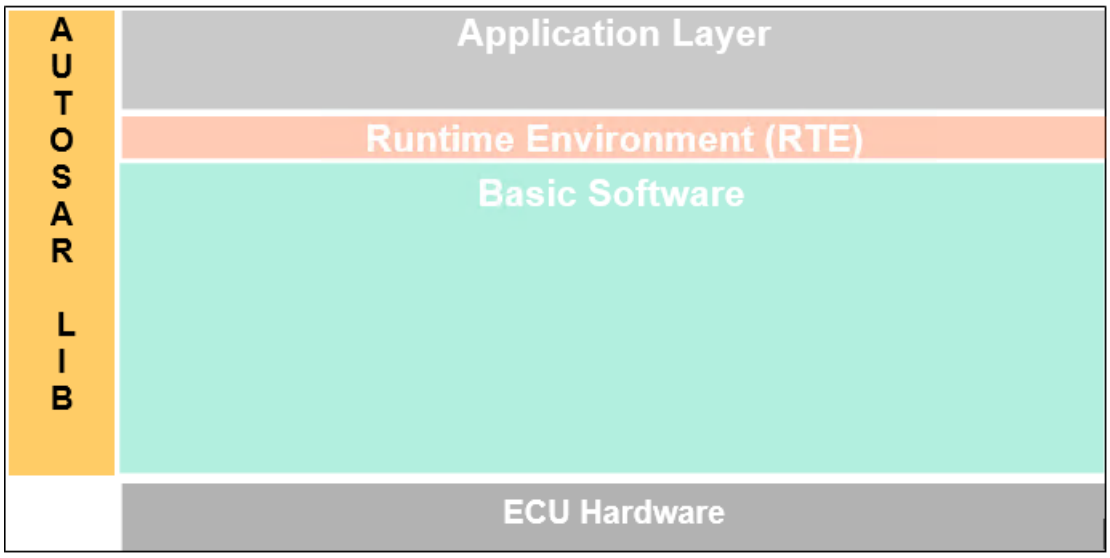

Figure 1.1: Layered Architecture

Bfx routines specification specifies the functionality and API of the AUTOSAR library for bit functionality dedicated to fixed-point arithmetic routines.

All Bfx routines are re-entrant and can handle several simultaneous requests from different users.

2 Acronyms and Abbreviations

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

Abbreviation / Acronym:	Description:
Bfx	Short name for Bitfield functions for fixed point

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

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

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[4] Requirements on Libraries
AUTOSAR_CP_RS_Libraries

[5] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[6] Specification of Platform Types for Classic Platform
AUTOSAR_CP_SWS_PlatformTypes

3.2 Related specification

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

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

4 Constraints and assumptions

4.1 Limitations

No limitations

4.2 Applicability to car domains

No restrictions

5 Dependencies to other modules

The Bfx Library has no dependency to other modules.

5.1 File structure

An implementation of the Bfx Library shall follow the naming rules for files outined in [3, SWS BSW General].

6 Requirements Tracing

The following tables reference the requirements specified in [4, RS Libraries] and [5, RS BSWGeneral] and links to the fulfillment of these.

Requirement	Description	Satisfied by
[SRS_LIBS_00005]	Each library shall provide one header file with its public interface	[SWS_Bfx_00135] [SWS_Bfx_00137] [SWS_Bfx_00139] [SWS_Bfx_00141]
[SRS_LIBS_00009]	All library functions shall be re-entrant	[SWS_Bfx_00135] [SWS_Bfx_00137] [SWS_Bfx_00139] [SWS_Bfx_00141]
[SRS_LIBS_00011]	All function names and type names shall start with "Library short name_"	[SWS_Bfx_00135] [SWS_Bfx_00137] [SWS_Bfx_00139] [SWS_Bfx_00141]
[SRS_LIBS_00015]	It shall be possible to configure the microcontroller so that the library code is shared between all callers	[SWS_Bfx_00206]
[SRS_LIBS_00017]	Usage of macros should be avoided	[SWS_Bfx_00207]
[SRS_LIBS_00018]	A library function may only call library functions	[SWS_Bfx_00208]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 Initialization and shutdown

The Bfx Library does not provide a `Init()` function nor requires an initialization phase.

A function of the Bfx library 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. The only assumption of the Bfx Library regarding their context is that it assumes a completed C startup.

Also the Bfx Library does not require a shutdown operation phase.

7.2 Using Library API

The Bfx Library functions (APIs) can be directly called from BSW modules or SWC.

Using a library should be documented. If a BSW module or a SWC uses the Bfx Library, the developer should add an Implementation-DependencyOnArtifact in the BSW/SWC template.

7.3 Library implementation

[SWS_Bfx_00206]

Upstream requirements: SRS_LIBS_00015

⌈The Bfx library shall be implemented in a way that the code can be shared among callers in different memory partitions.⌋

[SWS_Bfx_00207]

Upstream requirements: SRS_LIBS_00017

⌈Usage of C macros shall be avoided in the context of Bfx Library. The library functions shall be declared as function or as inline function.⌋

This means that the routine shall not be realized as C macro (i.e. not using a #define).

[SWS_Bfx_00208]

Upstream requirements: SRS_LIBS_00018

⌈A Bfx Library function shall not call any BSW module functions, e.g. the DET. A library function can call any other library functions since all library functions are re-entrant.⌋

[SWS_Bfx_00214] ⌈All Bfx Library functions shall avoid handling user faults and values outside specified range. Providing input values which are outside the specified range results in undefined behavior.⌋

Note: Example of such undefined behavior is calling `Bfx_SetBit_u8u8` with a value of 8 (or higher) as bit position argument.

7.4 Error Classification

Chapter [3, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of four error types which may occur in BSW modules.

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

7.4.1 Development Errors

There are no development errors.

7.4.2 Runtime Errors

There are no runtime errors

7.4.3 Production Errors

There are no production errors.

7.4.4 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

[SWS_Bfx_91001] Definition of imported datatypes of module Bfx ⌈

Module	*Header File*	*Imported Type*
Std	Std_Types.h	Std_VersionInfoType

⌋

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 CPU architectures and their compilers.

Thus, in order to improve the portability of the software, the types are defined in [6, SWS PlatformTypes] are used. The following *mnemonic*(s) are used in the library routine names to express the variablity.

Note: The naming convention for the API's with boolean return type/parameter type is given as _u8 which shall be interpreted as boolean. If there is no boolean data type present in the return type/parameter type then _u8 shall be interpreted as _uint8 only.

Size	Platform Type	Mnemonic
unsigned 8-Bit	boolean	u8
signed 8-Bit	sint8	s8
signed 16-Bit	sint16	s16
signed 32-Bit	sint32	s32
signed 64-Bit	sint64	s64
unsigned 8-Bit	uint8	u8
unsigned 16-Bit	uint16	u16
unsigned 32-Bit	uint32	u32
unsigned 64-Bit	uint64	u64

Table 8.1: Base types and their mnemonics

The ranges of these types can be found in [6, SWS PlatformTypes].

As a convention in the rest of the document:

- Mnemonics from table 8.1 will be used in the API names of the routines (using <TypeMn>)
- The real type will be used in the description of the prototypes of the routines using <Type>.

The bit addressing for the document is defined as following:

- The bit position of the lowest significant bit is defined as 0 (zero).
- The bit field length is defined as the number of bits.

8.2 Type definitions

None

8.3 Comment about functions optimized for target

The functions described in this library may be realized as regular functions or as inline functions.

8.4 Bit functions definitions

8.4.1 Bfx_SetBit

[SWS_Bfx_00001] Definition of API function Bfx_SetBit_<TypeMn>u8 ⌈

Service Name	Bfx_SetBit_<TypeMn>u8	
Syntax	`void Bfx_SetBit_<TypeMn>u8 ( <Type>* Data, uint8 BitPn )`	
Service ID [hex]	0x01 to 0x04 (see SWS_Bfx_00008)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BitPn	Bit position
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall set the logical status of input data as '1' at the requested bit position.	
Available via	Bfx.h	

⌋

Expected functionality:

```
1 *Data = *Data | (0x01 << BitPn)
```

Example:

```
1     Data = 10001010b;
2     Bfx_SetBit_u8u8(&Data, 2);
```

The `Data` will be updated to 10001110b

[SWS_Bfx_00008] List of `Bfx_SetBit` functions ⌈

Service ID[hex]	Function prototype	Valid values for parameter `BitPn`
0x01	void Bfx_SetBit_u8u8(uint8*, uint8)	0,1,2,..,7
0x02	void Bfx_SetBit_u16u8(uint16*, uint8)	0,1,2,..,15
0x03	void Bfx_SetBit_u32u8(uint32*, uint8)	0,1,2,..,31
0x04	void Bfx_SetBit_u64u8(uint64*, uint8)	0,1,2,..,63

⌋

8.4.2 Bfx_ClrBit

[SWS_Bfx_00010] Definition of API function Bfx_ClrBit_<TypeMn>u8 ⌈

Service Name	Bfx_ClrBit_<TypeMn>u8	
Syntax	`void Bfx_ClrBit_<TypeMn>u8 ( <Type>* Data, uint8 BitPn )`	
Service ID [hex]	0x06 to 0x09 (see SWS_Bfx_00015)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BitPn	Bit position
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall clear the logical status of the input data to '0' at the requested bit position.	
Available via	Bfx.h	

⌋

Expected functionality:

```
1 *Data = (*Data & ~(0x01 << BitPn))
```

Example:

```
1       Data = 10001010b;
2       Bfx_ClrBit_u8u8(&Data, 1);
```

The Data will be updated to 10001000b

[SWS_Bfx_00015] List of `Bfx_ClrBit` functions ⌈

Service ID[hex]	Function prototype	Valid values for parameter `BitPn`
0x06	void Bfx_ClrBit_u8u8(uint8*, uint8)	0,1,2,..,7
0x07	void Bfx_ClrBit_u16u8(uint16*, uint8)	0,1,2,..,15
0x08	void Bfx_ClrBit_u32u8(uint32*, uint8)	0,1,2,..,31
0x09	void Bfx_ClrBit_u64u8(uint64*, uint8)	0,1,2,..,63

⌋

8.4.3 Bfx_GetBit

[SWS_Bfx_00016] Definition of API function Bfx_GetBit_<TypeMn>u8_u8 ⌈

Service Name	Bfx_GetBit_<TypeMn>u8_u8	

Syntax	`boolean Bfx_GetBit_<TypeMn>u8_u8 (`
	`  <Type> Data,`
	`  uint8 BitPn`
	`)`	

Service ID [hex]	0x0A to 0x0D (see SWS_Bfx_00020)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input data
	BitPn	Bit position
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	Bit Status
Description	This function shall return the logical status of the input data for the requested bit position.	
Available via	Bfx.h	

⌋

Expected functionality:

```
1 if ((Data & (0x01 << BitPn)) != 0)
2     return TRUE;
3 else
4     return FALSE;
```

Example:

```
1 result = Bfx_GetBit_u8u8(10001010b, 1);
```

The `result` will be `TRUE`.

[SWS_Bfx_00020] List of `Bfx_GetBit` functions ⌈

Service ID[hex]	Function prototype	Valid values for parameter `BitPn`
0x0A	boolean Bfx_GetBit_u8u8_ u8(uint8,uint8)	0,1,2,..,7
0x0B	boolean Bfx_GetBit_u16u8_ u8(uint16,uint8)	0,1,2,..,15
0x0C	boolean Bfx_GetBit_u32u8_ u8(uint32,uint8)	0,1,2,..,31
0x0D	boolean Bfx_GetBit_u64u8_ u8(uint64,uint8)	0,1,2,..,63

⌋

8.4.4 Bfx_SetBits

[SWS_Bfx_00021] Definition of API function Bfx_SetBits_<TypeMn>u8u8u8 ⌈

Service Name	Bfx_SetBits_<TypeMn>u8u8u8	

Syntax	`void Bfx_SetBits_<TypeMn>u8u8u8 (`
	`  <Type>* Data,`
	`  uint8 BitStartPn,`
	`  uint8 BitLn,`
	`  uint8 Status`
	`)`	

Service ID [hex]	0x20 to 0x23 (see SWS_Bfx_00025)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BitStartPn	Start bit position
	BitLn	Bit field length
	Status	Status value, valid values are 0 and 1.
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall set the input data as '1' or '0' as per 'Status' value starting from 'BitStartPn' for the length 'BitLn'.	
Available via	Bfx.h	

⌋

Example:

```
1   Data = 1110100000000111b;
2   Bfx_SetBits_u16u8u8u8(&Data, 5, 5, 1);
```

Before the `Bfx_SetBits_u16u8u8u8` function call `Data` contains this value:

Position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	1	1	1	0	1	0	0	0	0	0	0	0	0	1	1	1

Figure 8.1: `Data` before `Bfx_SetBits_u16u8u8u8` call

The yellow fields mark those area which will be updated by the call (starting at bit position 5 and containing 5 bits). After the call the `Data` contains 1110101111100111b.

Position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	1	1	1	0	1	0	1	1	1	1	1	0	0	1	1	1

Figure 8.2: After `Bfx_SetBits_u16u8u8u8` call

Another example setting bits to '0':

```
1   Value = 1110100000000111b;
2   Bfx_SetBits_u16u8u8u8(&Value, 0, 8, 0);
3   /* Value is now 1110100000000000 */
```

[SWS_Bfx_00025] List of `Bfx_SetBits` functions ⌈

Service ID[hex]	Function prototype	Valid values for parameter `BitLn`	Valid values for parameter `BitStartPn`	Maximum value for `BitStartPn` + `BitLn`
0x20	void Bfx_SetBits_ u8u8u8u8(uint8*, uint8, uint8, uint8)	1,2,..,8	0,1,2,..,7	8
0x21	void Bfx_SetBits_ u16u8u8u8(uint16*, uint8, uint8, uint8)	1,2,..,16	0,1,2,..,15	16
0x22	void Bfx_SetBits_ u32u8u8u8(uint32*, uint8, uint8, uint8)	1,2,..,32	0,1,2,..,31	32
0x23	void Bfx_SetBits_ u64u8u8u8(uint64*, uint8, uint8, uint8)	1,2,..,64	0,1,2,..,63	64

⌋

8.4.5 Bfx_GetBits

[SWS_Bfx_00028] Definition of API function Bfx_GetBits_<TypeMn>u8u8_<TypeMn> ⌈

Service Name	Bfx_GetBits_<TypeMn>u8u8_<TypeMn>	
Syntax	`<Type> Bfx_GetBits_<TypeMn>u8u8_<TypeMn> ( <Type> Data, uint8 BitStartPn, uint8 BitLn )`	
Service ID [hex]	0x26 to 0x29 (see SWS_Bfx_00034)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input data
	BitStartPn	Start bit position
	BitLn	Bit field length
Parameters (inout)	None	
Parameters (out)	None	
Return value	<Type>	Bit field sequence
Description	This function shall return the Bits of the input data starting from 'BitStartPn' for the length of 'Bit Ln'.	
Available via	Bfx.h	

⌋

Example:

```
result = Bfx_GetBits_u16u8u8_u16(1110100000000111b, 9, 5);
```

The `result` will be 0000000000010100b.

[SWS_Bfx_00034] List of `Bfx_GetBits` functions ⌈

Service ID[hex]	Function prototype	Valid values for parameter `BitLn`	Valid values for parameter `BitStartPn`	Maximum value for `BitStartPn` + `BitLn`
0x26	uint8 Bfx_GetBits_ u8u8u8_ u8(uint8,uint8,uint8)	1,2,..,8	0,1,2,..,7	8
0x27	uint16 Bfx_GetBits_ u16u8u8_ u16(uint16,uint8,uint8)	1,2,..,16	0,1,2,..,15	16
0x28	uint32 Bfx_GetBits_ u32u8u8_ u32(uint32,uint8,uint8)	1,2,..,32	0,1,2,..,31	32
0x29	uint64 Bfx_GetBits_ u64u8u8_ u64(uint64,uint8,uint8)	1,2,..,64	0,1,2,..,63	64

⌋

8.4.6 Bfx_SetBitMask

[SWS_Bfx_00035] Definition of API function Bfx_SetBitMask_<TypeMn><TypeMn> ⌈

Service Name	Bfx_SetBitMask_<TypeMn><TypeMn>	

Syntax	`void Bfx_SetBitMask_<TypeMn><TypeMn> (`
	`  <Type>* Data,`
	`  <Type> Mask`
	`)`	

Service ID [hex]	0x2A to 0x2D (see SWS_Bfx_00038)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Mask	Mask used to set bits
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall set the data to logical status '1' as per the corresponding Mask bits when set to value 1 and remaining bits will retain their original values.	
Available via	Bfx.h	

⌋

Expected functionality:

```
1 *Data = *Data | Mask
```

[SWS_Bfx_00038] List of `Bfx_SetBitMask` functions ⌈

Service ID[hex]	Function prototype
0x2A	void Bfx_SetBitMask_u8u8(uint8*, uint8)
0x2B	void Bfx_SetBitMask_u16u16(uint16*, uint16)
0x2C	void Bfx_SetBitMask_u32u32(uint32*, uint32)
0x2D	void Bfx_SetBitMask_u64u64(uint64*, uint64)

⌋

8.4.7 Bfx_ClrBitMask

[SWS_Bfx_00039] Definition of API function Bfx_ClrBitMask_<TypeMn><TypeMn> ⌈

Service Name	Bfx_ClrBitMask_<TypeMn><TypeMn>	
Syntax	`void Bfx_ClrBitMask_<TypeMn><TypeMn> ( <Type>* Data, <Type> Mask )`	
Service ID [hex]	0x30 to 0x33 (see SWS_Bfx_00045)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Mask	Mask value
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall clear the logical status to '0' for the input data for all the bit positions as per the mask.	
Available via	Bfx.h	

⌋

[SWS_Bfx_00040] ⌈This function shall clear the data to logical status '0' as per the corresponding mask bits value when set to 1. The remaining bits shall retain their original values.⌋

Expected functionality:

```
1 *Data = *Data & ~Mask
```

[SWS_Bfx_00045] List of `Bfx_ClrBitMask` functions ⌈

Service ID[hex]	Function prototype
0x30	void Bfx_ClrBitMask_u8u8(uint8*, uint8)
0x31	void Bfx_ClrBitMask_u16u16(uint16*, uint16)
0x32	void Bfx_ClrBitMask_u32u32(uint32*, uint32)
0x33	void Bfx_ClrBitMask_u64u64(uint64*, uint64)

⌋

8.4.8 Bfx_TstBitMask

[SWS_Bfx_00046] Definition of API function Bfx_TstBitMask_<TypeMn><TypeMn>_u8 ⌈

Service Name	Bfx_TstBitMask_<TypeMn><TypeMn>_u8	

Syntax	`boolean Bfx_TstBitMask_<TypeMn><TypeMn>_u8 (`
	`  <Type> Data,`
	`  <Type> Mask`
	`)`	

Service ID [hex]	0x36 to 0x39 (see SWS_Bfx_00050)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input data
	Mask	Mask value
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	Value
Description	This function shall return TRUE, if all bits defined in Mask value are set in the input Data value. In all other cases this function shall return FALSE.	
Available via	Bfx.h	

⌋

Expected functionality:

```
if ((Data & Mask) == Mask)
  return TRUE;
else
  return FALSE;
```

Example:

```
result = Bfx_TstBitMask_u8u8_u8(10010011b,10010000b);
```

The `result` will be `TRUE`.

[SWS_Bfx_00050] List of `Bfx_TstBitMask` functions ⌈

Service ID[hex]	Function prototype
0x36	boolean Bfx_TstBitMask_u8u8_u8(uint8,uint8)
0x37	boolean Bfx_TstBitMask_u16u16_u8(uint16,uint16)
0x38	boolean Bfx_TstBitMask_u32u32_u8(uint32,uint32)
0x39	boolean Bfx_TstBitMask_u64u64_u8(uint64,uint64)

⌋

8.4.9 Bfx_TstBitLnMask

[SWS_Bfx_00051] Definition of API function Bfx_TstBitLnMask_<TypeMn><TypeMn>_u8 ⌈

Service Name	Bfx_TstBitLnMask_<TypeMn><TypeMn>_u8	

Syntax	`boolean Bfx_TstBitLnMask_<TypeMn><TypeMn>_u8 (`
	`  <Type> Data,`
	`  <Type> Mask`
	`)`	

Service ID [hex]	0x3A to 0x3D (see SWS_Bfx_00055)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input data
	Mask	Mask value
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	Data
Description	This function makes a test on the input data and if at least one bit is set amongst the bits set in the mask, then the function shall return TRUE, otherwise it shall return FALSE.	
Available via	Bfx.h	

⌋

Example:

```
1 result = Bfx_TstBitLnMask_u8u8_u8(10010011b,10000111b);
```

The `result` will be `TRUE`.

[SWS_Bfx_00055] List of `Bfx_TstBitLnMask` functions ⌈

Service ID[hex]	Function prototype
0x3A	boolean Bfx_TstBitLnMask_u8u8_u8(uint8,uint8)
0x3B	boolean Bfx_TstBitLnMask_u16u16_u8(uint16,uint16)
0x3C	boolean Bfx_TstBitLnMask_u32u32_u8(uint32,uint32)
0x3D	boolean Bfx_TstBitLnMask_u64u64_u8(uint64,uint64)

⌋

8.4.10 Bfx_TstParityEven

[SWS_Bfx_00056] Definition of API function Bfx_TstParityEven_<TypeMn>_u8 ⌈

Service Name	Bfx_TstParityEven_<TypeMn>_u8	
Syntax	`boolean Bfx_TstParityEven_<TypeMn>_u8 ( <Type> Data )`	
Service ID [hex]	0x40 to 0x43 (see SWS_Bfx_00060)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input Data
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	Status
Description	This function tests the number of bits set to 1. If this number is even, it shall return TRUE, otherwise it returns FALSE.	
Available via	Bfx.h	

⌋

Examples when `Bfx_TstParityEven` returns `TRUE`:

```
result_a = Bfx_TstParityEven_u8_u8(10010011b);
result_b = Bfx_TstParityEven_u16_u8(0);
/* result_a == result_b == TRUE */
```

The following example calls return `FALSE`:

```
result_c = Bfx_TstParityEven_u8_u8(11100011b);
result_d = Bfx_TstParityEven_u16_u8(1);
result_e = Bfx_TstParityEven_u16_u8(1001111100101010b);
/* result_c == result_d == result_e == FALSE */
```

[SWS_Bfx_00060] List of `Bfx_TstParityEven` functions ⌈

Service ID[hex]	Function prototype
0x40	boolean Bfx_TstParityEven_u8_u8(uint8)
0x41	boolean Bfx_TstParityEven_u16_u8(uint16)
0x42	boolean Bfx_TstParityEven_u32_u8(uint32)
0x43	boolean Bfx_TstParityEven_u64_u8(uint64)

⌋

8.4.11 Bfx_ToggleBits

[SWS_Bfx_00061] Definition of API function Bfx_ToggleBits_<TypeMn> ⌈

Service Name	Bfx_ToggleBits_<TypeMn>	

Syntax	`void Bfx_ToggleBits_<TypeMn> (`
	`  <Type>* Data`
	`)`	

Service ID [hex]	0x46 to 0x49 (see SWS_Bfx_00065)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	Data	Pointer to input data
Parameters (out)	None	
Return value	None	
Description	This function toggles all the bits of data (1's Complement Data).	
Available via	Bfx.h	

⌋

Example:

```
1 data = 10010011b;
2 Bfx_ToggleBits_u8(&data);
```

The `data` will be 01101100b.

[SWS_Bfx_00065] List of `Bfx_ToggleBits` functions ⌈

Service ID[hex]	Function prototype
0x46	void Bfx_ToggleBits_u8(uint8*)
0x47	void Bfx_ToggleBits_u16(uint16*)
0x48	void Bfx_ToggleBits_u32(uint32*)
0x49	void Bfx_ToggleBits_u64(uint64*)

⌋

8.4.12 Bfx_ToggleBitMask

[SWS_Bfx_00066] Definition of API function Bfx_ToggleBitMask_<TypeMn><TypeMn> ⌈

Service Name	Bfx_ToggleBitMask_<TypeMn><TypeMn>	

Syntax	`void Bfx_ToggleBitMask_<TypeMn><TypeMn> (`
	`  <Type>* Data,`
	`  <Type> Mask`
	`)`	

Service ID [hex]	0x4A to 0x4D (see SWS_Bfx_00069)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Mask	Mask
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function toggles the bits of data when the corresponding bit of the mask is enabled and set to 1.	
Available via	Bfx.h	

⌋

Example:

```
1 data = 11110000b;
2 Bfx_ToggleBitMask_u8u8(&data, 00111100b);
```

The `data` will be 11001100b.

[SWS_Bfx_00069] List of `Bfx_ToggleBitMask` functions ⌈

Service ID[hex]	Function prototype
0x4A	void Bfx_ToggleBitMask_u8u8(uint8*, uint8)
0x4B	void Bfx_ToggleBitMask_u16u16(uint16*, uint16)
0x4C	void Bfx_ToggleBitMask_u32u32(uint32*, uint32)
0x4D	void Bfx_ToggleBitMask_u64u64(uint64*, uint64)

⌋

8.4.13 Bfx_ShiftBitRt

[SWS_Bfx_00070] Definition of API function Bfx_ShiftBitRt_<TypeMn>u8 ⌈

Service Name	Bfx_ShiftBitRt_<TypeMn>u8	

Syntax	`void Bfx_ShiftBitRt_<TypeMn>u8 (`
	`  <Type>* Data,`
	`  uint8 ShiftCnt`
	`)`	

Service ID [hex]	0x50 to 0x53 (see SWS_Bfx_00075)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ShiftCnt	Shift right count
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall shift data to the right by ShiftCnt. The most significant bit (left-most bit) is replaced by a '0' bit and the least significant bit (right-most bit) is discarded for every single bit shift cycle.	
Available via	Bfx.h	

⌋

Example:

```
1 data = 10010011b;
2 Bfx_ShiftBitRt_u8u8(&data,3);
```

The `data` will be 00010010b.

[SWS_Bfx_00075] List of `Bfx_ShiftBitRt` functions ⌈

Service ID[hex]	Function prototype	Maximum value of ShiftCnt
0x50	void Bfx_ShiftBitRt_u8u8(uint8*, uint8)	7
0x51	void Bfx_ShiftBitRt_u16u8(uint16*, uint8)	15
0x52	void Bfx_ShiftBitRt_u32u8(uint32*, uint8)	31
0x53	void Bfx_ShiftBitRt_u64u8(uint64*, uint8)	63

⌋

8.4.14 Bfx_ShiftBitLt

[SWS_Bfx_00076] Definition of API function Bfx_ShiftBitLt_<TypeMn>u8 ⌈

Service Name	Bfx_ShiftBitLt_<TypeMn>u8	
Syntax	`void Bfx_ShiftBitLt_<TypeMn>u8 ( <Type>* Data, uint8 ShiftCnt )`	
Service ID [hex]	0x56 to 0x59 (see SWS_Bfx_00080)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ShiftCnt	Shift left count
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall shift data to the left by ShiftCnt. The least significant bit (right-most bit) is replaced by a '0' bit and the most significant bit (left-most bit) is discarded for every single bit shift cycle.	
Available via	Bfx.h	

⌋

Example:

```
1 data = 10010011b;
2 Bfx_ShiftBitLt_u8u8(&data,3);
```

The `data` will be 10011000b.

[SWS_Bfx_00080] List of `Bfx_ShiftBitLt` functions ⌈

Service ID[hex]	Function prototype	Maximum value of ShiftCnt
0x56	void Bfx_ShiftBitLt_u8u8(uint8*, uint8)	7
0x57	void Bfx_ShiftBitLt_u16u8(uint16*, uint8)	15
0x58	void Bfx_ShiftBitLt_u32u8(uint32*, uint8)	31
0x59	void Bfx_ShiftBitLt_u64u8(uint64*, uint8)	63

⌋

8.4.15 Bfx_RotBitRt

[SWS_Bfx_00086] Definition of API function Bfx_RotBitRt_<TypeMn>u8 ⌈

Service Name	Bfx_RotBitRt_<TypeMn>u8	

Syntax	`void Bfx_RotBitRt_<TypeMn>u8 (`
	`  <Type>* Data,`
	`  uint8 ShiftCnt`
	`)`	

Service ID [hex]	0x5A to 0x5D (see SWS_Bfx_00090)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ShiftCnt	Shift count
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall rotate data to the right by ShiftCnt. The least significant bit is rotated to the most significant bit location for every single bit shift cycle.	
Available via	Bfx.h	

⌋

Examples:

```
1 data_a = 00010111b;
2 Bfx_RotBitRt_u8u8(&data,1);
3 data_b = 00010111b;
4 Bfx_RotBitRt_u8u8(&data,3);
```

The `data_a` will be 10001011b and `data_b` will be 11100010b.

[SWS_Bfx_00090] List of `Bfx_RotBitRt` functions ⌈

Service ID[hex]	Function prototype	Maximum value of ShiftCnt
0x5A	void Bfx_RotBitRt_u8u8(uint8*, uint8)	7
0x5B	void Bfx_RotBitRt_u16u8(uint16*, uint8)	15
0x5C	void Bfx_RotBitRt_u32u8(uint32*, uint8)	31
0x5D	void Bfx_RotBitRt_u64u8(uint64*, uint8)	63

⌋

8.4.16 Bfx_RotBitLt

[SWS_Bfx_00095] Definition of API function Bfx_RotBitLt_<TypeMn>u8 ⌈

Service Name	Bfx_RotBitLt_<TypeMn>u8	

Syntax	`void Bfx_RotBitLt_<TypeMn>u8 (`
	`  <Type>* Data,`
	`  uint8 ShiftCnt`
	`)`	

Service ID [hex]	0x60 to 0x63 (see SWS_Bfx_00098)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ShiftCnt	Shift count
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall rotate data to the left by ShiftCnt. The most significant bit is rotated to the least significant bit location for every single bit shift cycle.	
Available via	Bfx.h	

⌋

Examples:

```
1 data_a = 10110111b;
2 Bfx_RotBitLt_u8u8(&data,1);
3 data_b = 10110111b;
4 Bfx_RotBitLt_u8u8(&data,3);
```

The `data_a` will be 01101111b and `data_b` will be 10111101b.

[SWS_Bfx_00098] List of `Bfx_RotBitLt` functions ⌈

Sevice ID[hex]	Function prototype	Maximum value of ShiftCnt
0x60	void Bfx_RotBitLt_u8u8(uint8*, uint8)	7
0x61	void Bfx_RotBitLt_u16u8(uint16*, uint8)	15
0x62	void Bfx_RotBitLt_u32u8(uint32*, uint8)	31
0x63	void Bfx_RotBitLt_u64u8(uint64*, uint8)	63

⌋

8.4.17 Bfx_CopyBit

[SWS_Bfx_00101] Definition of API function Bfx_CopyBit_<TypeMn>u8<TypeMn>u8 ⌈

Service Name	Bfx_CopyBit_<TypeMn>u8<TypeMn>u8	

Syntax	`void Bfx_CopyBit_<TypeMn>u8<TypeMn>u8 (`
	`  <Type>* DestinationData,`
	`  uint8 DestinationPosition,`
	`  <Type> SourceData,`
	`  uint8 SourcePosition`
	`)`	

Service ID [hex]	0x66 to 0x69 (see SWS_Bfx_00108)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	DestinationPosition	Destination position
	SourceData	Source data
	SourcePosition	Source position
Parameters (inout)	DestinationData	Pointer to destination data
Parameters (out)	None	
Return value	None	
Description	This function shall copy a bit from source data from bit position to destination data at bit position.	
Available via	Bfx.h	

⌋

Example:

```
1 DestinationData = 10100001b;
2 Bfx_CopyBit_u8u8u8u8(&DestinationData, 6, 11011010, 1);
```

The `DestinationData` will have 11100001b.

[SWS_Bfx_00108] List of `Bfx_CopyBit` functions ⌈

Service ID[hex]	Function prototype	Maximum value for SourcePosition and DestinationPosition
0x66	void Bfx_CopyBit_u8u8u8u8(uint8*, uint8, uint8, uint8)	7
0x67	void Bfx_CopyBit_u16u8u16u8(uint16*, uint8, uint16, uint8)	15
0x68	void Bfx_CopyBit_u32u8u32u8(uint32*, uint8, uint32, uint8)	31
0x69	void Bfx_CopyBit_u64u8u64u8(uint64*, uint8, uint64, uint8)	63

⌋

8.4.18 Bfx_PutBits

[SWS_Bfx_00110] Definition of API function Bfx_PutBits_<TypeMn>u8u8<TypeMn> ⌈

Service Name	Bfx_PutBits_<TypeMn>u8u8<TypeMn>	
Syntax	`void Bfx_PutBits_<TypeMn>u8u8<TypeMn> ( <Type>* Data, uint8 BitStartPn, uint8 BitLn, <Type> Pattern )`	
Service ID [hex]	0x70 to 0x73 (see SWS_Bfx_00112)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BitStartPn	Start bit position
	BitLn	Bit field length
	Pattern	Pattern to be set
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall put bits as mentioned in Pattern to the input Data from the specified bit position.	
Available via	Bfx.h	

⌋

Example:

```
1 Data = 11110000b;
2 Bfx_PutBits_u8u8u8u8(&Data, 1, 3, 00000011b);
```

The `Data` will have 11110110b.

[SWS_Bfx_00112] List of `Bfx_PutBits` functions ⌈

Service ID[hex]	Function prototype	Maximum value of Bit Ln	Maximum value of Bit StartPn	Maximum value for BitStartPn + BitLn
0x70	void Bfx_PutBits_ u8u8u8u8(uint8*, uint8, uint8, uint8)	8	7	8
0x71	void Bfx_PutBits_ u16u8u8u16(uint16*, uint8, uint8, uint16)	16	15	16
0x72	void Bfx_PutBits_ u32u8u8u32(uint32*, uint8, uint8, uint32)	32	31	32
0x73	void Bfx_PutBits_ u64u8u8u64(uint64*, uint8, uint8, uint64)	64	63	64

⌋

8.4.19 Bfx_PutBitsMask

[SWS_Bfx_00120] Definition of API function Bfx_PutBitsMask_<TypeMn><TypeMn><TypeMn> ⌈

Service Name	Bfx_PutBitsMask_<TypeMn><TypeMn><TypeMn>	

Syntax	`void Bfx_PutBitsMask_<TypeMn><TypeMn><TypeMn> (`
	`  <Type>* Data,`
	`  <Type> Pattern,`
	`  <Type> Mask`
	`)`	

Service ID [hex]	0x80 to 0x83 (see SWS_Bfx_00124)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Pattern	Pattern to be set
	Mask	Mask value
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall put all bits defined in Pattern and for which the corresponding Mask bit is set to 1 in the input Data.	
Available via	Bfx.h	

⌋

Example:

```
1 data = 11100000b;
2 Bfx_PutBitsMask_u8u8u8(&data, 11001101b, 00001111b);
```

The `data` will contain 11101101b.

[SWS_Bfx_00124] List of `Bfx_PutBitsMask` functions ⌈

Service ID[hex]	Function prototype
0x80	void Bfx_PutBitsMask_u8u8u8(uint8*, uint8, uint8)
0x81	void Bfx_PutBitsMask_u16u16u16(uint16*, uint16, uint16)
0x82	void Bfx_PutBitsMask_u32u32u32(uint32*, uint32, uint32)
0x83	void Bfx_PutBitsMask_u64u64u64(uint64*, uint64, uint64)

⌋

8.4.20 Bfx_PutBit

[SWS_Bfx_00130] Definition of API function Bfx_PutBit_<TypeMn>u8u8 ⌈

Service Name	Bfx_PutBit_<TypeMn>u8u8	
Syntax	`void Bfx_PutBit_<TypeMn>u8u8 ( <Type>* Data, uint8 BitPn, boolean Status )`	
Service ID [hex]	0x85 to 0x88 (see SWS_Bfx_00132)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BitPn	Bit position
	Status	Status value
Parameters (inout)	Data	Pointer to data
Parameters (out)	None	
Return value	None	
Description	This function shall update the bit specified by BitPn of input data as '1' or '0' as per 'Status' value.	
Available via	Bfx.h	

⌋

Example:

```
1 InputData = 11100111b;
2 Bfx_PutBit_u8u8u8(&InputData, 4, TRUE);
```

Then `InputData` will be 11110111b.

[SWS_Bfx_00132] List of `Bfx_PutBit` functions ⌈

Service ID[hex]	Function prototype	Maximum value of BitPn
0x85	void Bfx_PutBit_u8u8u8(uint8*, uint8, boolean)	7
0x86	void Bfx_PutBit_u16u8u8(uint16*, uint8, boolean)	15
0x87	void Bfx_PutBit_u32u8u8(uint32*, uint8, boolean)	31
0x88	void Bfx_PutBit_u64u8u8(uint64*, uint8, boolean)	63

⌋

8.4.21 Bfx_ShiftBitSat

[SWS_Bfx_91002] Definition of API function Bfx_ShiftBitSat_<TypeMn>s8_<TypeMn> ⌈

Service Name	Bfx_ShiftBitSat_<TypeMn>s8_<TypeMn>	
Syntax	`<Type> Bfx_ShiftBitSat_<TypeMn>s8_<TypeMn> ( sint8 ShiftCnt, <Type> Data )`	
Service ID [hex]	0xE1 to 0xE8 (see SWS_Bfx_00135)	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ShiftCnt	Shift count (-<MaxShiftRight> ... -1: right, 1 ... <MaxShiftLeft>: left)
	Data	Input value
Parameters (inout)	None	
Parameters (out)	None	
Return value	<Type>	Shifted and saturated bit pattern.
Description	Arithmetic shift with saturation	
Available via	Bfx.h	

⌋

[SWS_Bfx_00134] ⌈If the shift count is greater than zero, then shift the value in Data by the amount specified by shift count to left.

If shift count is zero Data is returned.

For signed data an arithmetic shift is performed. The vacated bits are filled with zeros and the result is saturated if its sign bit differs from the sign bits that are shifted out.

For unsigned data a logical shift is performed. In this case the result is saturated, if the leading one bit is shifted out.

If the shift count is less than zero, right-shift the value in Data by the absolute value of the shift count. The vacated bits are filled with the sign-bit (the most significant bit) and bits shifted out are discarded.

Note that a shift right by the word width leaves all zeros or all ones in the result, depending on the sign-bit.⌋

Example of 32 bit signed integer: The range for shift count is -32 to +31, allowing a shift left up to 31 bit positions and a shift right up to 32 bit positions (a shift right by 32 bits leaves all zeros or all ones in the result, depending on the sign bit).

Here is a listing with some examples using unsigned data (uint16). The saturation of unsigned data is either zero or the maxium value of the type (here `UINT16_MAX`):

```
1 uint16 udata,uresult;
2
3 udata = 0x8F57;
4
5 uresult = Bfx_ShiftBitSat_u16s8_u16(-4,udata);
```

```
/*  uresult is now 0x08F5: A logical right shift is done
    new leftmost bits are filled with 0  */

uresult = Bfx_ShiftBitSat_u16s8_u16(8,udata);
/*  uresult is now 0xFFFF: A logical left shift is done.
    One of the "outshifted" bits was 1, indicating that
    an overflow happend. Here it was the first bit.
    Result is saturated to UINT16_MAX. */

udata = 0x3856;
uresult = Bfx_ShiftBitSat_u16s8_u16(4,udata);
/*  uresult is now 0xFFFF: A logical left shift is done.
    One of the "outshifted" bits was 1, indicating that
    an overflow happend. Here it was the third bit.
    Result is saturated to UINT16_MAX. */
```

Here are examples for signed data (using sint16). The saturation of signed data depends on the sign of the value. Neagtive values are saturated to `SINT16_MIN` value or -1. Positive values are saturated to `SINT16_MAX` or 0.

```
sint16 sdata,sresult;

sdata = -200; /* == 0xFF38 */

sresult = Bfx_ShiftBitSat_s16s8_s16(-4,sdata);
/*  sresult is now -13 (== 0xFFF3): An arithmetic
    right shift is done, the new leftmost bits are
    filled with 1. */

sresult = Bfx_ShiftBitSat_s16s8_s16(-10,sdata);
/*  sresult is now -1 (== 0xFFFF): An arithmetic
    right shift is done, the new leftmost bits are
    filled with 1 and the value saturates at -1.
*/

sresult = Bfx_ShiftBitSat_s16s8_s16(10,sdata);
/*  sresult is now SINT16_MIN (== 0x8000): An arithmetic
    left shift is done. One of the "outshifted" bits
    was 0 and indicates a negative overflow.
    So sresult is saturated to SINT16_MIN.
*/

sdata = 127; /* == 0x007F */

sresult = Bfx_ShiftBitSat_s16s8_s16(-4,sdata);
/*  sresult is now 7 (== 0x0007) */

sresult = Bfx_ShiftBitSat_s16s8_s16(-10,sdata);
/*  sresult is now 0 (== 0x0000) */

```

```
31 sresult = Bfx_ShiftBitSat_s16s8_s16(10,sdata);
32 /*  sresult is now SINT16_MAX (== 0x7FFF) */
```

[SWS_Bfx_00135] List of `Bfx_ShiftBitSat` functions

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service ID[hex]	Function prototype	Max Shift Left	Max Shift Right
0xE1	Bfx_ShiftBitSat_s8s8_s8	7	8
0xE2	Bfx_ShiftBitSat_u8s8_u8	8	8
0xE3	Bfx_ShiftBitSat_s16s8_s16	15	16
0xE4	Bfx_ShiftBitSat_u16s8_u16	16	16
0xE5	Bfx_ShiftBitSat_s32s8_s32	31	32
0xE6	Bfx_ShiftBitSat_u32s8_u32	32	32
0xE7	Bfx_ShiftBitSat_s64s8_s64	63	64
0xE8	Bfx_ShiftBitSat_u64s8_u64	64	64

⌋

8.4.22 Bfx_CountLeadingOnes

[SWS_Bfx_91003] Definition of API function Bfx_CountLeadingOnes_<TypeMn>

⌈

Service Name	Bfx_CountLeadingOnes_<TypeMn>	

Syntax	`uint8 Bfx_CountLeadingOnes_<TypeMn> (`
	`  <Type> Data`
	`)`	

Service ID [hex]	0xF0 to 0xF3 (see SWS_Bfx_00137)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input data
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	Number of leading "1"s
Description	Count the number of consecutive ones in Data starting with the most significant bit and return the result.	
Available via	Bfx.h	

⌋

[SWS_Bfx_00137] List of `Bfx_CountLeadingOnes` functions

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service ID[hex]	Function prototype	Maximum return value
0xF0	uint8 Bfx_CountLeadingOnes_u8 (uint8)	8
0xF1	uint8 Bfx_CountLeadingOnes_u16 (uint16)	16
0xF2	uint8 Bfx_CountLeadingOnes_u32 (uint32)	32
0xF3	uint8 Bfx_CountLeadingOnes_u64 (uint64)	64

⌋

8.4.23 Bfx_CountLeadingSigns

[SWS_Bfx_91004] Definition of API function Bfx_CountLeadingSigns_<TypeMn>
⌈

Service Name	Bfx_CountLeadingSigns_<TypeMn>	

Syntax	`uint8 Bfx_CountLeadingSigns_<TypeMn> (`
	`  <Type> Data`
	`)`	

Service ID [hex]	0xF4 to 0xF7 (see SWS_Bfx_00139)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input data
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	Count of leading signs
Description	Count and return the number of consecutive bits which have the same value as most significant bit in Data, starting with bit at position msb minus one.	
Available via	Bfx.h	

⌋

Here are some examples using `Bfx_CountLeadingSigns` and their results:

```
sint16 data;
uint8  result;

data = 0; /* 0x0000 */

result = Bfx_CountLeadingSigns_s16(data);
/*  result == 15: The sign bit is 0,
    all other 15 bits are 0. */

data = -200; /* 0xFF38 */

result = Bfx_CountLeadingSigns_s16(data);
/*  result == 7: The sign bit is 1 and
    followed by 7 bits which are also 1. */

data = 31000; /* 0x7918 */

result = Bfx_CountLeadingSigns_s16(data);
/*  result == 0: The sign bit is 0, but
    the next bit is already different. */
```

[SWS_Bfx_00139] List of `Bfx_CountLeadingSigns` functions

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service ID[hex]	Function prototype	Maximum return value
0xF4	uint8 Bfx_CountLeadingSigns_s8 (sint8)	7
0xF5	uint8 Bfx_CountLeadingSigns_s16 (sint16)	15
0xF6	uint8 Bfx_CountLeadingSigns_s32 (sint32)	31
0xF7	uint8 Bfx_CountLeadingSigns_s64 (sint64)	63

⌋

8.4.24 Bfx_CountLeadingZeros

[SWS_Bfx_91005] Definition of API function Bfx_CountLeadingZeros_<TypeMn> ⌈

Service Name	Bfx_CountLeadingZeros_<TypeMn>	

Syntax	`uint8 Bfx_CountLeadingZeros_<TypeMn> (`
	`  <Type> Data`
	`)`	

Service ID [hex]	0xF8 to 0xFB (see SWS_Bfx_00141)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Data	Input data
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	Number of leading "0"s
Description	Count the number of consecutive zeros in Data starting with the most significant bit and return the result.	
Available via	Bfx.h	

⌋

[SWS_Bfx_00141] List of `Bfx_CountLeadingZeros` functions

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service ID[hex]	Function prototype	Maximum return value
0xF8	uint8 Bfx_CountLeadingZeros_u8 (uint8)	8
0xF9	uint8 Bfx_CountLeadingZeros_u16 (uint16)	16
0xFA	uint8 Bfx_CountLeadingZeros_u32 (uint32)	32
0xFB	uint8 Bfx_CountLeadingZeros_u64 (uint64)	64

⌋

8.4.25 Bfx_GetVersionInfo

[SWS_Bfx_00301] Definition of API function Bfx_GetVersionInfo ⌈

Service Name	Bfx_GetVersionInfo

Syntax	`void Bfx_GetVersionInfo (`
	`  Std_VersionInfoType* Versioninfo`
	`)`

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

▽

△

Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	Versioninfo	Pointer to where to store the version information of this module. For details see also SWS_BSW_00162
Return value	None	
Description	Returns the version information of this library.	
Available via	Bfx.h	

⌋

8.5 Callback notifications

None

8.6 Scheduled functions

The Bfx library does not have scheduled functions.

8.7 Expected interfaces

None

8.7.1 Mandatory interfaces

None

8.7.2 Optional interfaces

None

8.7.3 Configurable interfaces

None

8.8 Service Interfaces

The library functions are directly usable by SWCs. This means that there is no port definition available nor required.

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 Bfx.

Chapter 10.3 specifies published information of the module Bfx.

10.1 How to read this chapter

For details refer to [3] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Containers and configuration parameters

The Bfx Library has no standardized configuration options.

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 [3] Chapter 10.3 *"Published Information"*.

A Not applicable requirements

[SWS_Bfx_NA_00001] Library is different from a BSW module

Upstream requirements: SRS_BSW_00101, SRS_BSW_00159, SRS_BSW_00323, SRS_BSW_00336, SRS_BSW_00375, SRS_BSW_00406, SRS_BSW_00416, SRS_BSW_00419, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00450, SRS_BSW_00478

⌈Bfx Library is a library and different to a basic software module, e.g. it has no main function or reports wake-up reasons.⌋

[SWS_Bfx_NA_00002] Bfx Library has no configuration

Upstream requirements: SRS_BSW_00167, SRS_BSW_00171, SRS_BSW_00344, SRS_BSW_00345, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00403, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00438

⌈Bfx Library has no configuration.⌋

[SWS_Bfx_NA_00003] No usage of other BSW modules

Upstream requirements: SRS_BSW_00337, SRS_BSW_00384, SRS_BSW_00385, SRS_BSW_00409, SRS_BSW_00417, SRS_BSW_00452, SRS_BSW_00458, SRS_BSW_00461, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00488, SRS_BSW_00489, SRS_BSW_00490, SRS_BSW_00491, SRS_BSW_00492, SRS_BSW_00493, SRS_BSW_00339

⌈Bfx Library does not use nor call other BSW modules. This includes e.g. error reporting to Det or Dem and also reporting and handling of security events.⌋

[SWS_Bfx_NA_00999]

Upstream requirements: SRS_BSW_00004, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00369, SRS_BSW_00402, SRS_BSW_00422, SRS_BSW_00429, SRS_BSW_00451, SRS_BSW_00437

⌈These requirements are not applicable to this specification.⌋

B History of Specification Items

Please note that the lists in this chapter also include specification items that have been removed from the specification in a later version. These specification items do not appear as hyperlinks in the document.

B.1 Specification Item History of this document compared to AUTOSAR R24-11

B.1.1 Added Specification Items in R25-11

none

B.1.2 Changed Specification Items in R25-11

Number	Heading
[SWS_Bfx_00001]	Definition of API function Bfx_SetBit_<TypeMn>u8
[SWS_Bfx_00008]	List of `Bfx_SetBit` functions
[SWS_Bfx_00010]	Definition of API function Bfx_ClrBit_<TypeMn>u8
[SWS_Bfx_00015]	List of `Bfx_ClrBit` functions
[SWS_Bfx_00016]	Definition of API function Bfx_GetBit_<TypeMn>u8_u8
[SWS_Bfx_00020]	List of `Bfx_GetBit` functions
[SWS_Bfx_00021]	Definition of API function Bfx_SetBits_<TypeMn>u8u8u8
[SWS_Bfx_00025]	List of `Bfx_SetBits` functions
[SWS_Bfx_00028]	Definition of API function Bfx_GetBits_<TypeMn>u8u8_<TypeMn>
[SWS_Bfx_00034]	List of `Bfx_GetBits` functions
[SWS_Bfx_00035]	Definition of API function Bfx_SetBitMask_<TypeMn><TypeMn>
[SWS_Bfx_00038]	List of `Bfx_SetBitMask` functions
[SWS_Bfx_00039]	Definition of API function Bfx_ClrBitMask_<TypeMn><TypeMn>
[SWS_Bfx_00045]	List of `Bfx_ClrBitMask` functions
[SWS_Bfx_00046]	Definition of API function Bfx_TstBitMask_<TypeMn><TypeMn>_u8
[SWS_Bfx_00050]	List of `Bfx_TstBitMask` functions
[SWS_Bfx_00051]	Definition of API function Bfx_TstBitLnMask_<TypeMn><TypeMn>_u8
[SWS_Bfx_00055]	List of `Bfx_TstBitLnMask` functions
[SWS_Bfx_00056]	Definition of API function Bfx_TstParityEven_<TypeMn>_u8
[SWS_Bfx_00060]	List of `Bfx_TstParityEven` functions
[SWS_Bfx_00061]	Definition of API function Bfx_ToggleBits_<TypeMn>
[SWS_Bfx_00065]	List of `Bfx_ToggleBits` functions

△

Number	Heading
[SWS_Bfx_00066]	Definition of API function Bfx_ToggleBitMask_<TypeMn><TypeMn>
[SWS_Bfx_00069]	List of `Bfx_ToggleBitMask` functions
[SWS_Bfx_00070]	Definition of API function Bfx_ShiftBitRt_<TypeMn>u8
[SWS_Bfx_00075]	List of `Bfx_ShiftBitRt` functions
[SWS_Bfx_00076]	Definition of API function Bfx_ShiftBitLt_<TypeMn>u8
[SWS_Bfx_00080]	List of `Bfx_ShiftBitLt` functions
[SWS_Bfx_00086]	Definition of API function Bfx_RotBitRt_<TypeMn>u8
[SWS_Bfx_00090]	List of `Bfx_RotBitRt` functions
[SWS_Bfx_00095]	Definition of API function Bfx_RotBitLt_<TypeMn>u8
[SWS_Bfx_00098]	List of `Bfx_RotBitLt` functions
[SWS_Bfx_00101]	Definition of API function Bfx_CopyBit_<TypeMn>u8<TypeMn>u8
[SWS_Bfx_00108]	List of `Bfx_CopyBit` functions
[SWS_Bfx_00110]	Definition of API function Bfx_PutBits_<TypeMn>u8u8<TypeMn>
[SWS_Bfx_00112]	List of `Bfx_PutBits` functions
[SWS_Bfx_00120]	Definition of API function Bfx_PutBitsMask_<TypeMn><TypeMn><TypeMn>
[SWS_Bfx_00124]	List of `Bfx_PutBitsMask` functions
[SWS_Bfx_00130]	Definition of API function Bfx_PutBit_<TypeMn>u8u8
[SWS_Bfx_00132]	List of `Bfx_PutBit` functions
[SWS_Bfx_00135]	List of `Bfx_ShiftBitSat` functions
[SWS_Bfx_00137]	List of `Bfx_CountLeadingOnes` functions
[SWS_Bfx_00139]	List of `Bfx_CountLeadingSigns` functions
[SWS_Bfx_00141]	List of `Bfx_CountLeadingZeros` functions
[SWS_Bfx_91002]	Definition of API function Bfx_ShiftBitSat_<TypeMn>s8_<TypeMn>
[SWS_Bfx_91003]	Definition of API function Bfx_CountLeadingOnes_<TypeMn>
[SWS_Bfx_91004]	Definition of API function Bfx_CountLeadingSigns_<TypeMn>
[SWS_Bfx_91005]	Definition of API function Bfx_CountLeadingZeros_<TypeMn>

Table B.1: Changed Specification Items in R25-11

B.1.3 Deleted Specification Items in R25-11

Number	Heading
[SWS_Bfx_00002]	
[SWS_Bfx_00011]	
[SWS_Bfx_00017]	
[SWS_Bfx_00022]	
[SWS_Bfx_00029]	

▽

△

Number	Heading
[SWS_Bfx_00036]	
[SWS_Bfx_00047]	
[SWS_Bfx_00200]	
[SWS_Bfx_00201]	
[SWS_Bfx_00203]	
[SWS_Bfx_00205]	
[SWS_Bfx_00209]	
[SWS_Bfx_00212]	
[SWS_Bfx_00213]	
[SWS_Bfx_00220]	
[SWS_Bfx_00222]	
[SWS_Bfx_00223]	:
[SWS_Bfx_00302]	
[SWS_Bfx_00314]	
[SWS_Bfx_00999]	

Table B.2: Deleted Specification Items in R25-11

B.2 Specification Item History of this document compared to AUTOSAR R23-11

B.2.1 Added Specification Items in R24-11

none

B.2.2 Changed Specification Items in R24-11

none

B.2.3 Deleted Specification Items in R24-11

none

B.3 Specification Item History of this document compared to AUTOSAR R22-11

B.3.1 Added Specification Items in R23-11

none

B.3.2 Changed Specification Items in R23-11

Number	Heading
[SWS_Bfx_00008]	
[SWS_Bfx_00015]	
[SWS_Bfx_00016]	Definition of API function Bfx_GetBit_<InTypeMn>u8_u8
[SWS_Bfx_00020]	
[SWS_Bfx_00025]	
[SWS_Bfx_00034]	
[SWS_Bfx_00038]	
[SWS_Bfx_00045]	
[SWS_Bfx_00046]	Definition of API function Bfx_TstBitMask_<InTypeMn><InTypeMn>_u8
[SWS_Bfx_00050]	
[SWS_Bfx_00051]	Definition of API function Bfx_TstBitLnMask_<InTypeMn><InTypeMn>_u8
[SWS_Bfx_00055]	
[SWS_Bfx_00056]	Definition of API function Bfx_TstParityEven_<InTypeMn>_u8
[SWS_Bfx_00060]	
[SWS_Bfx_00065]	
[SWS_Bfx_00069]	
[SWS_Bfx_00075]	
[SWS_Bfx_00080]	
[SWS_Bfx_00090]	
[SWS_Bfx_00098]	
[SWS_Bfx_00108]	
[SWS_Bfx_00112]	
[SWS_Bfx_00124]	
[SWS_Bfx_00132]	
[SWS_Bfx_00135]	
[SWS_Bfx_00137]	
[SWS_Bfx_00139]	
[SWS_Bfx_00141]	
[SWS_Bfx_91001]	Definition of imported datatypes of module Bfx
[SWS_Bfx_91003]	Definition of API function Bfx_CountLeadingOnes_<TypeMn>
[SWS_Bfx_91004]	Definition of API function Bfx_CountLeadingSigns_<TypeMn>
[SWS_Bfx_91005]	Definition of API function Bfx_CountLeadingZeros_<TypeMn>

Table B.3: Changed Specification Items in R23-11

B.3.3 Deleted Specification Items in R23-11

none