Document Title	# Specification of Bit Handling Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	399
Document Classification	Standard

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

Document Change History

Date	Version	Changed by	Change Description
09.12.2011	2.0.0	AUTOSAR Administration	• 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.
24.11.2010	1.1.0	AUTOSAR Administration	• Signature for necessary Bit handling functions optimized for easy usage • Bit handling on all signed variables eliminated • Additional bit handling functions introduced
30.11.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

Table of Contents

1 Introduction and functional overview

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

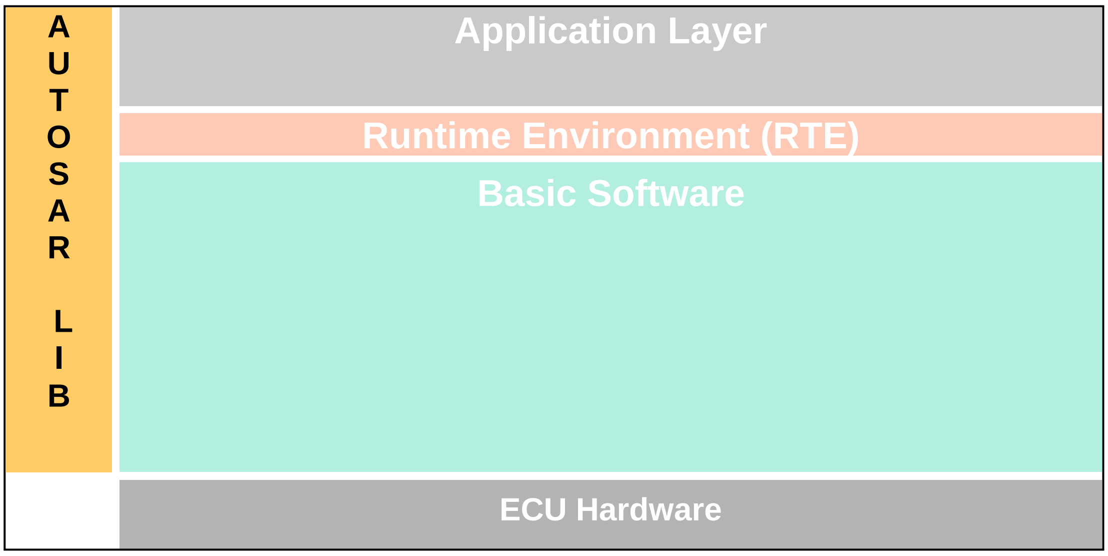

Figure: Layered architecture

Bfx routines specification specifies the functionality, API and the configuration of the AUTOSAR library for BIT functionality dedicated to fixed-point arithmetic routines

All bit functions are re-entrant and can handle several simultaneous requests from the application.

2 Acronyms and abbreviations

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

Abbreviation / Acronym:	***Description:***
BFx	Short name for Bitfield functions for fixed point
u8	Short name for uint8, specified in AUTOSAR_SWS_PlatformTypes
u16	Short name for uint16, specified in AUTOSAR_SWS_PlatformTypes
u32	Short name for uint32, specified in AUTOSAR_SWS_PlatformTypes
s8	Short name for sint8, specified in AUTOSAR_SWS_PlatformTypes
s16	Short name for sint16, specified in AUTOSAR_SWS_PlatformTypes
s32	Short name for sint32, specified in AUTOSAR_SWS_PlatformTypes
boolean	Boolean data type, specified in AUTOSAR_SWS_PlatformTypes
DET	Development Error Tracer

3 Related documentation

3.1 Input documents

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

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

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

[6] Specification of Platform Types,
AUTOSAR_SWS_PlatformTypes.pdf

[7] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[8] Requirement on Libraries,
AUTOSAR_SRS_Libraries.pdf

[9] Specification of Memory Mapping,
AUTOSAR_SWS_MemoryMapping

[10] Software Component Template,
AUTOSAR_TPS_SoftwareComponentTemplateSoftware

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

3.2 Related standards and norms

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

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

4 Constraints and assumptions

4.1 Limitations

No limitations

4.2 Applicability to car domains

No restrictions

5 Dependencies to other modules

5.1 File structure

[BFX220] ⌈ The Bfx module shall provide the following files:

- C files, Bfx_<name>.c used to implement the library. All C files shall be prefixed with 'Bfx'.
  Header file Bfx.h provides all public function prototypes and types defined by the BFX library specification ⌋(BWS31400005)

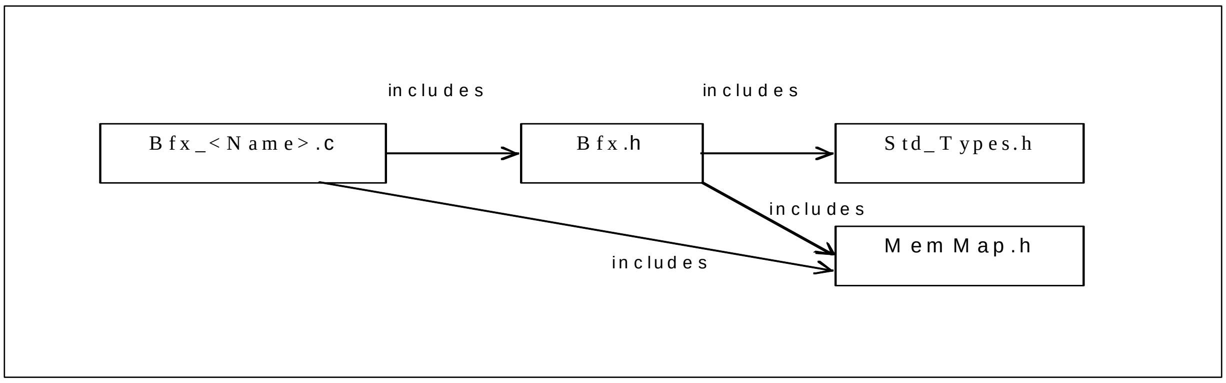

Figure 1: File structure

[BFX222]

⌈ 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.: Bfx_setbit.c etc.

Option 2 : <Name> can have common name of group of functions:
2.1 Group by object family:
eg.:Bfx_set.c, Bfx_get.c
2.2 Group by routine family:
eg.: Bfx_bit8.c,Bfx_bit16.c etc.
2.4 Group by other methods: (individual grouping allowed)

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

6 Requirements traceability

6.1 Document: Requirements on Libraries

Requirement	Description	Satisfied by
BWS003		BFX301
BWS00304	All AUTOSAR library Modules should use the AUTOSAR data types (Integers, Boolean) instead of nat...	BFX212
BWS00306	All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, u...	BFX213
BWS00318		BFX301
BWS00321		BFX301
BWS00348	Each AUTOSAR library Module implementation *.	BFX211
BWS00374		BFX301
BWS00378	All AUTOSAR library Modules should use the AUTOSAR data types (Integers, Boolean) instead of nat...	BFX212
BWS00379		BFX301
BWS00407		BFX301, BFX302
BWS00411	If source code for caller and callee of Bfx_GetVersionInfo is available, the Bfx library should ...	BFX302
BWS00436	Each AUTOSAR library Module implementation *.	BFX210
BWS007	The library, written in C programming language, should confirm to the HIS subset of the MISRA C ...	BFX209
BWS31400002	Bfx library shall not require initialization phase.	BFX200
BWS31400003	Bfx library shall not require a shutdown operation phase.	BFX201
BWS31400004	Bfx API can be directly called from BSW modules or SWC.	BFX203
BWS31400005	The Bfx module shall provide the following files:	BFX220
BWS31400006	The statement "Bfx.	BFX204
BWS31400007	Using a library should be documented.	BFX205
BWS31400012	These requirements are not applicable to this specification.	BFX999
BWS31400013	Error detection: Function should check at runtime (both in production and in development code) t...	BFX216, BFX217
BWS31400015	The Bfx library shall be implemented in a way that the code can be shared among callers in diffe...	BFX206
BWS31400017	Usage of macros must be avoided in the context of Library.	BFX207
BWS31400018	A library function shall not call any BSW module functions, e.	BFX208

Requirement	***Satisfied by***

Requirement	***Satisfied by***
[BSW31400001] The functional behavior of each library functions shall not be configurable	Section 10.2: **BFX314**
[BSW31400002] A library shall be operational before all BSW modules and application SWCs	Section 7.4: **BFX200**
[BSW31400003] A library shall be operational until the shutdown	Section 7.4: **BFX201**
[BSW31400004] Using libraries shall not pass through a port interface	Section 7.5: **BFX203**
[BSW31400005] Library header file	Section 5.1: **BFX220**
[BSW31400006] The header #include is placed by the SWC developer	Section 7.5: **BFX204**
[BSW31400007] Using a library should be documented	Section 7.5: **BFX205**
[BSW31400008] Backward compatibility	This is the first evolution and SWS shall considered in future evolution of this document
[BSW31400009] Re-entrancy	All APIs in section 8 are defined to be re-entrant
[BSW31400010] Specific types	Not relevant: section 8.3 does not define specific types
[BSW31400011] Naming convention for functions and types	APIs defined in section 8 are named according this requirement
[BSW31400012] Passing parameters with structure is allowed	Not applicable for this documents
[BSW31400013] Error detection	Section 7.2: **BFX216**, **BFX217**
[BSW31400015] Shared library code	Section 7.6: **BFX206**
[BSW31400016] Non AUTOSAR library	Not relevant. BFX is an AUTOSAR standard library
[BSW31400017] Usage of macros should be avoided	Section 7.6: **BFX207**
[BSW31400018] A library function can only call library functions	Section 7.6: **BFX208**

6.2 Document: Requirements on Basic SW Modules

A library is not a basic software module. Therefore, many requirements for BSW modules are not applicable and hence not listed below. However, only few relevant requirements are listed:

Requirement	***Satisfied by***
[BSW00402] Published information	Section 10.1: **BFX310**
[BSW00407] Function to read out published parameters	Section 8.8.1: **BFX301**, **BFX302**
[BSW007] HIS MISRA C	Section 7.6: **BFX209**
[BSW00411] Get version info keyword	Section 8.8.1: **BFX302**
[BSW00436] Module Header File Struc-	Section 7.6: **BFX210**

ture for the Basic Software Memory Mapping	
[BSW00348] Standard type header	Section 7.6: **BFX211**
[BSW00304] AUTOSAR integer data types	Section 7.6: **BFX212**
[BSW00378] AUTOSAR boolean type	Section 7.6: **BFX212**
[BSW00306] Avoid direct use of compiler and platform specific keywords	Section 7.6: **BFX213**
[BSW00374] Module vendor identification	Section 10.1: **BFX310, BFX301**
[BSW00379] Module identification	Section 10.1: **BFX310, BFX301**
[BSW003] Version identification	Section 8.8.1: **BFX301**
[BSW00318] Format of module version numbers	Section 8.8.1: **BFX301**
[BSW00321] Enumeration of module version numbers	Section 8.8.1: **BFX301**

7 Functional specification

7.1 Error classification

[BFX223]: No error classification definition as DET call not supported by library

7.2 Error detection

[BFX216] ⌈ Error detection: Function should check at runtime (both in production and in development code) the value of input parameters, especially cases where erroneous value can bring to fatal error or unpredictable result, if they have the values allowed by the function specification. All the error cases shall be listed in SWS and the function should return a specified value (in SWS) that is not configurable. This value is dependant of the function and the error case so it is determined case by case.
If values passed to the library routines are not in valid range, out of boundary condition and out of the function specification, then such error are not detected in the library routines ⌋(BWS31400013)

7.3 Error notification

[BFX217] ⌈ A library function can only call library functions: The functions shall not call the DET in case of error. ⌋(BWS31400013)

7.4 Initialization and shutdown

[BFX200] ⌈ Bfx 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. ⌋(BWS31400002)

[BFX201] ⌈ Bfx library shall not require a shutdown operation phase. ⌋(BWS31400003)

7.5 Using Library API

[BFX203] ⌈ Bfx API can be directly called from BSW modules or SWC. No port definition is required. It is a pure function call. ⌋(BWS31400004)

[BFX204] ⌈ The statement "Bfx.h" shall be placed by the developer or an application code generator but not by the RTE generator ⌋(BWS31400006)

[BFX205] ⌈ Using a library should be documented. if a BSW module or a SWC uses a Library, the developer should add an Implementation-DependencyOnLibrary in the BSW/SWC template.
minVersion and maxVersion parameters correspond to the supplier version. In case of AUTOSAR library, these parameters may be left empty because a SWC or BSW module may rely on library behaviour, not on a supplier implementation. However, the SWC or BSW modules shall be compatible with the AUTOSAR platform where they are integrated. ⌋(BWS31400007)

7.6 Library implementation

[BFX206] ⌈ The Bfx library shall be implemented in a way that the code can be shared among callers in different memory partitions. ⌋(BWS31400015)

[BFX207] ⌈ Usage of macros must be avoided in the context of Library. The library function must be declared as function or as inline function and Macro #define should not be used. ⌋(BWS31400017)

[BFX208] ⌈ A 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 but not BSW module functions, as they may not be re-entrant ⌋(BWS31400018)

[BFX209] ⌈ The library, written in C programming language, should confirm to the HIS subset of the MISRA C Standard.
Only in technically reasonable and exceptional cases, MISRA violations are permissible. Such MISRA rules violations shall be clearly identified and documented within comments in the C source code (including rationale behind MISRA rule is violation). The comment shall be placed right above the line of code, which causes the violation and have the following syntax:

```
/* MISRA RULE XX VIOLATION: Reason why the MISRA rule could not be followed

in this special case*/
```
⌋(BWS007)

[BFX210] ⌈ Each AUTOSAR library Module implementation <library>*.c shall include the header file MemMap.h. ⌋(BWS00436)

[BFX211] ⌈ 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. ⌋(BWS00348)

[BFX212] ⌈ 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. ⌋(BWS00304, BWS00378)

[BFX213] ⌈ All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, unless this library clearly identified to be compliant only with a platform. ⌋(BWS00306)

[BFX214] ⌈ All Bit Library modules shall avoid handling user faults and values outside specified range. ⌋()

8 API specification

8.1 Imported types

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

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

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

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

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

Table 1: Base Types

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

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

Table 2: Ranges for Base Types

As a convention in the rest of the document:

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

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 a , inline functions

8.4 Bit functions definitions

8.4.1 Bfx_SetBit

[BFX 001]

⌈

Service name:	Bfx_SetBit_<TypeMn>u8	

Syntax:	`void Bfx_SetBit_<TypeMn>u8(`
	`    <Type>* const Data,`
	`    uint8 BitPn`
	`)`	

Service ID[hex]:	0x01 to 0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BitPn	Bit position
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX002:** This function shall set the logical status of input data as '1' at the requested bit position. Expected functionality: *Data = *Data | (0x01 << BitPn)	

⌋()

[BFX008]

⌈ List of implemented functions

Function ID[hex]	**Function prototype**
0x001	void Bfx_SetBit_u8u8(uint8* const, uint8)
0x002	void Bfx_SetBit_u16u8(uint16* const, uint8)
0x003	void Bfx_SetBit_u32u8(uint32* const, uint8)

⌋()

8.4.2 Bfx_ClrBit

[BFX010]⌈

Service name:	Bfx_ClrBit_<TypeMn>u8	
Syntax:	`void Bfx_ClrBit_<TypeMn>u8( <Type>* const Data, uint8 BitPn )`	
Service ID[hex]:	0x06 to 0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BitPn	Bit position
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX011:** This function shall clear the logical status of the input data to '0' at the requested bit position. Expected functionality: *Data = (*Data & ~(0x01 << BitPn))	

⌋()

[BFX015]

⌈ List of implemented functions

Function ID[hex]	***Function prototype***
0x006	void Bfx_ClrBit_u8u8(uint8* const, uint8)
0x007	void Bfx_ClrBit_u16u8(uint16* const, uint8)
0x008	void Bfx_ClrBit_u32u8(uint32* const, uint8)

⌋()

8.4.3 Bfx_GetBit

[BFX016]⌈

Service name:	Bfx_GetBit_<InTypeMn>u8_u8	
Syntax:	`boolean Bfx_GetBit_<InTypeMn>u8_u8( <InType> Data, uint8 BitPn )`	
Service ID[hex]:	0x0a to 0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Input data
	BitPn	Bit position
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	boolean	Bit Status
Description:	**BFX017:** This function shall return the logical status of the input data for the requested bit position.	

	Result = 1, ((Data & (0x01 << BitPn)) != 0)
	Result = 0, else

⌋()

[BFX020]

⌈ List of implemented functions

Function ID[hex]	***Function prototype***
0x00A	boolean Bfx_GetBit_u8u8_u8(uint8,uint8)
0x00B	boolean Bfx_GetBit_u16u8_u8(uint16,uint8)
0x00C	boolean Bfx_GetBit_u32u8_u8(uint32,uint8)

⌋()

8.4.4 Bfx_SetBits

[BFX021]⌈

Service name:	Bfx_SetBits_<TypeMn>u8u8u8	

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

Service ID[hex]:	0x20 to 0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BitStartPn	Start bit position
	BitLn	Bit field length
	Status	Status value
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX022:** This function shall set the input data as '1' or '0' as per 'Status' value starting from 'BitStartPn' for the length 'BitLn'. *For details see table below* For Example: Bfx_SetBits_u16u8u8u8(1110100000000111b, 5, 5, 1) function returns 1110101111100111b	

⌋()

15	14	13	12	11	10	9	-	2	1	0
1	1	1	0	1	0	0	0	0	0	0
						<BitStartPn>				
		< BitLn >								

[BFX025]

⌈ List of implemented functions:

Function	***Function prototype***

ID[hex]	
0x020	void Bfx_SetBits_u8u8u8u8(uint8* const, uint8, uint8, uint8)
0x021	void Bfx_SetBits_u16u8u8u8(uint16* const, uint8, uint8, uint8)
0x022	void Bfx_SetBits_u32u8u8u8(uint32* const, uint8, uint8, uint8)

⌋()

8.4.5 Bfx_GetBits

[BFX28] ⌈

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 0x28	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Input data
	BitStartPn	Start bit position
	BitLn	Bit field length
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	<Type>	Bit field sequence
Description:	**BFX029:** This function shall return the Bits of the input data starting from 'BitStartPn' for the length of 'BitLn'. *For details see table below* For Example: BFX_GetBits_u16u8u8_u16(1110100000000111b, 9, 5) returns 0000000000010100b	

⌋()

15	14	13	12	11	10	9	-	2	1	0
1	1	1	0	1	0	0	0	1	1	1
						BitStartPn				
		< BitLn >								

[BFX 034]

⌈ List of implemented functions:

Function ID[hex]	*Function prototype*
0x026	uint8 Bfx_GetBits_u8u8u8_u8(uint8,uint8,uint8)
0x027	uint16 Bfx_GetBits_u16u8u8_u16(uint16,uint8,uint8)
0x028	uint32 Bfx_GetBits_u32u8u8_u32(uint32,uint8,uint8)

⌋()

8.4.6 Bfx_SetBitMask

[BFX 035]⌈

Service name:	Bfx_SetBitMask_<TypeMn><TypeMn>	

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

Service ID[hex]:	0x2a to 0x2c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Mask	Mask used to set bits
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX036:** 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. Expected functionality: *Data = *Data | Mask	

⌋()

[BFX 038]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0X02A	void Bfx_SetBitMask_u8u8(uint8* const, uint8)
0X02B	void Bfx_SetBitMask_u16u16(uint16* const, uint16)
0X02C	void Bfx_SetBitMask_u32u32(uint32* const, uint32)

⌋()

8.4.7 Bfx_ClrBitMask

[BFX039]

⌈

Service name:	Bfx_ClrBitMask_<TypeMn><TypeMn>	

Syntax:	`void Bfx_ClrBitMask_<TypeMn><TypeMn>(`
	`    <Type>* const Data,`
	`    <Type> Mask`
	`)`	

Service ID[hex]:	0x30 to 0x32	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Mask	Mask value
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX040:** This function shall clear the logical status to '0' for the input data for all the bit positions as per the mask.	

	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: *Data = *Data & ~Mask

⌋()

[BFX045]

⌈ List of implemented functions:

Function ID[hex]	Function prototype
0x030	void Bfx_ClrBitMask_u8u8(uint8* const, uint8)
0x031	void Bfx_ClrBitMask_u16u16(uint16* const, uint16)
0x032	void Bfx_ClrBitMask_u32u32(uint32* const, uint32)

⌋()

8.4.8 Bfx_TstBitMask

[BFX046]

⌈

Service name:	Bfx_TstBitMask_<InTypeMn><InTypeMn>_u8	

Syntax:	`boolean Bfx_TstBitMask_<InTypeMn><InTypeMn>_u8(`
	`    <InType> Data,`
	`    <InType> Mask`
	`)`	

Service ID[hex]:	0x36 to 0x38	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Input data
	Mask	Mask value
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	boolean	Value

Description:	**BFX047:**
	This function makes a test on the input variable and if all the bits are set as per the mask value, function shall return 1 or else 0.
	Result = 0, ((Value & Mask)== 0)
	Result = 1, else by default	

⌋()

[BFX050]

⌈ List of implemented functions:

Function ID[hex]	Function prototype
0x036	boolean Bfx_TstBitMask_u8u8_u8(uint8,uint8)
0x037	boolean Bfx_TstBitMask_u16u16_u8(uint16,uint16)
0x038	boolean Bfx_TstBitMask_u32u32_u8(uint32,uint32)

⌋()

8.4.9 Bfx_TstBitLnMask

[BFX051] ⌈

Service name:	Bfx_TstBitLnMask_<InTypeMn><InTypeMn>_u8	

Syntax:	`boolean Bfx_TstBitLnMask_<InTypeMn><InTypeMn>_u8(`
	`    <InType> Data,`
	`    <InType> Mask`
	`)`	

Service ID[hex]:	0x3a to 0x3c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Input data
	Mask	Mask value
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	boolean	Data
Description:	**BFX052**: This function makes a test on the input data and if at least one bit is set as per the mask, then the function shall return '1' or else '0'.	

⌋()

[BFX055]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0x03A	boolean Bfx_TstBitLnMask_u8u8_u8(uint8,uint8,uint8)
0x03B	boolean Bfx_TstBitLnMask_u16u16_u8(uint16,uint16,uint8)
0x03C	boolean Bfx_TstBitLnMask_u32u32_u8(uint32,uint32,uint8)

⌋()

8.4.10 Bfx_TstParityEven

[BFX056] ⌈

Service name:	Bfx_ParityTstEven_<InTypeMn>_u8	

Syntax:	`boolean Bfx_ParityTstEven_<InTypeMn>_u8(`
	`    <InTypeMn> Data`
	`)`	

Service ID[hex]:	0x40 to 0x42	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Input Data
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	boolean	Status
Description:	**BFX057:** This function tests the number of bits set to 1. If this number is even, it shall return 1, otherwise it returns 0.	

⌋()

[BFX060]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0x040	boolean Bfx_TstParityEven_u8_u8(uint8)
0x041	boolean Bfx_TstParityEven_u16_u8(uint16)
0x042	boolean Bfx_TstParityEven_u32_u8(uint32)

⌋()

8.4.11 Bfx_ToggleBits

[BFX061]⌈

Service name:	Bfx_ToggleBits_<TypeMn>	

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

Service ID[hex]:	0x46 to 0x48	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX062**: This function toggles all the bits of data (1's Complement Data).	

⌋()

[BFX065]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0x046	void Bfx_ToggleBits_u8(uint8* const)
0x047	void Bfx_ToggleBits_u16(uint16* const)
0x048	void Bfx_ToggleBits_u32(uint32* const)

⌋()

8.4.12 Bfx_ToggleBitMask

[BFX066]⌈

Service name:	Bfx_ToggleBitMask_<TypeMn><TypeMn>

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

Service ID[hex]:	0x4a to 0x4c
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	Mask	Mask
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX067:** This function toggles the bits of data when the corresponding bit of the mask is enabled and set to 1.	

⌋()

[BFX069]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0x04A	void Bfx_ToggleBitMask_u8u8(uint8* const, uint8)
0x04B	void Bfx_ToggleBitMask_u16u16(uint16* const, uint16)
0x04C	void Bfx_ToggleBitMask_u32u32(uint32* const, uint32)

⌋()

8.4.13 Bfx_ShiftBitRt

[BFX070]

⌈

Service name:	Bfx_ShiftBitRt_<TypeMn>u8	

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

Service ID[hex]:	0x50 to 0x52	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ShiftCnt	Shift right count
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX071:** 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.	

⌋()

BFX075:
List of implemented functions:

Function ID[hex]	***Function prototype***
0X050	void Bfx_ShiftBitRt_u8u8(uint8* const, uint8)
0X051	void Bfx_ShiftBitRt_u16u8(uint16* const, uint8)
0X052	void Bfx_ShiftBitRt_u32u8(uint32* const, uint8)

8.4.14 Bfx_ShiftBitLt

[BFX076]

⌈

Service name:	Bfx_ShiftBitLt_<TypeMn>u8	
Syntax:	`void Bfx_ShiftBitLt_<TypeMn>u8( <Type>* const Data, uint8 ShiftCnt )`	
Service ID[hex]:	0x56 to 0x58	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ShiftCnt	Shift left count
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX077:** 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 dis-carded for every single bit shift cycle.	

⌋()

[BFX080]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0X056	void Bfx_ShiftBitLt_u8u8(uint8* const, uint8)
0X057	void Bfx_ShiftBitLt_u16u8(uint16* const, uint8)
0X058	void Bfx_ShiftBitLt_u32u8(uint32* const, uint8)

⌋()

8.4.15 Bfx_RotBitRt

[BFX086]

⌈

Service name:	Bfx_RotBitRt_<TypeMn>u8	
Syntax:	`void Bfx_RotBitRt_<TypeMn>u8( <Type>* const Data, uint8 ShiftCnt )`	
Service ID[hex]:	0x5a to 0x5c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ShiftCnt	Shift count
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX087:**	

	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. e.g. If ShiftCnt = 1 then, uint8 Data = 0001 0111 (before rotate right) Data = 1000 1011 (after rotate right) e.g. If ShiftCnt = 3 then, uint8 Data = 0001 0111 (before rotate right) Data = 1110 0010 (after rotate right)

⌋()

[BFX090]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0X05A	void Bfx_RotBitRt_u8u8(uint8* const, uint8)
0X05B	void Bfx_RotBitRt_u16u8(uint16* const, uint8)
0X05C	void Bfx_RotBitRt_u32u8(uint32* const,uint8)

⌋()

8.4.16 Bfx_RotBitLt

[BFX095]

⌈

Service name:	Bfx_RotBitLt_<TypeMn>u8	

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

Service ID[hex]:	0x60 to 0x62	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ShiftCnt	Shift count
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX096:** 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. e.g. If ShiftCnt = 1 then, uint8 Data = 1011 0111 (before rotate left) Data = 0110 1111 (after rotate left) e.g. If ShiftCnt = 3 then, uint8 Data = 1011 0111 (before rotate left) Data = 1011 1101 (after rotate left)	

⌋()

[BFX098]

⌈ List of implemented functions:

Function	***Function prototype***

ID[hex]	
0X060	void Bfx_RotBitLt_u8u8(uint8* const, uint8)
0X061	void Bfx_RotBitLt_u16u8(uint16* const,uint8)
0X062	void Bfx_RotBitLt_u32u8(uint32* const,uint8)

⌋()

8.4.17 Bfx_CopyBit

[BFX101]

⌈

Service name:	Bfx_CopyBit_<TypeMn>u8<TypeMn>u8	

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

Service ID[hex]:	0x66 to 0x68	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	DestinationPosition	Destination position
	SourceData	Source data
	SourcePosition	Source position
Parameters (in-out):	DestinationData	Pointer to destination data
Parameters (out):	None	
Return value:	None	
Description:	**BFX102:** This function shall copy a bit from source data from bit position to destination data at bit position.	

⌋()

[BFX108]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0X066	void Bfx_CopyBit_u8u8u8u8(uint8* const, uint8, uint8, uint8)
0X067	void Bfx_CopyBit_u16u8u16u8(uint16* const , uint8, uint16, uint8)
0X068	void Bfx_CopyBit_u32u8u32u8(uint32* const , uint8, uint32, uint8)

⌋()

8.4.18 Bfx_PutBits

[BFX110]

⌈

Service name:	Bfx_PutBits_<TypeMn>u8u8<TypeMn>

Syntax:	`void Bfx_PutBits_<TypeMn>u8u8<TypeMn>(`
	`    <Type>* const Data,`

	`    uint8 BitStartPn,`
	`    uint8 BitLn,`
	`    <Type> Pattern`
	`)`	

Service ID[hex]:	0x70 to 0x72	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BitStartPn	Start bit position
	BitLn	Bit field length
	Pattern	Pattern to be set
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX111:** This function shall put bits as mentioned in Pattern to the input Data from the specified bit position. For Example: Bfx_PutBits_u8u8u8u8(11110000b, 1, 3, 00000011b) results in *Data = 11110110b	

⌋()

[BFX112]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0x070	void Bfx_PutBits_u8u8u8u8(uint8* const, uint8, uint8, uint8)
0x071	void Bfx_PutBits_u16u8u8u16(uint16* const, uint8, uint8, uint16)
0x072	void Bfx_PutBits_u32u8u8u32(uint32* const, uint8, uint8, uint32)

⌋()

8.4.19 Bfx_PutBitsMask

[BFX120]

⌈

Service name:	Bfx_PutBitsMask_<TypeMn><TypeMn><TypeMn>	

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

Service ID[hex]:	0x80 to 0x82	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Pattern	Pattern to be set
	Mask	Mask value
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX122:**	

	This function shall put bits as mentioned in Pattern to the input Data when corresponding Mask bits enabled and set to 1. For Example: Bfx_PutBitsMask_u8u8u8(11100000b, 11001101b, 00001111b) results in *Data = 11101101b

⌋ ()

[BFX124]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0x080	void Bfx_PutBitsMask_u8u8u8(uint8* const, uint8, uint8)
0x081	void Bfx_PutBitsMask_u16u16u16(uint16* const, uint16, uint16)
0x082	void Bfx_PutBitsMask_u32u32u32(uint32* const, uint32, uint32)

⌋()

8.4.20 Bfx_PutBit

[BFX130]⌈

Service name:	Bfx_PutBit_<TypeMn>u8u8	

Syntax:	`void Bfx_PutBit_<TypeMn>u8u8(`
	`    <Type>* const Data,`
	`    uint8 BitPn,`
	`    uint8 Status`
	`)`	

Service ID[hex]:	0x85 to 0x87	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BitPn	Bit position
	Status	Status value
Parameters (in-out):	Data	Pointer to input data
Parameters (out):	None	
Return value:	None	
Description:	**BFX131:** This function shall update the bit specified by BitPn of input data as '1' or '0' as per 'Status' value. For Example: Bfx_PutBit_u8u8u8(11100111b, 4, 1b) results in *Data = 11110111b	

⌋()

[BFX132]

⌈ List of implemented functions:

Function ID[hex]	***Function prototype***
0x085	void Bfx_PutBit_u8u8u8(uint8* const, uint8, boolean)
0x086	void Bfx_PutBit_u16u8u8(uint16* const, uint8, boolean)

0x087	void Bfx_PutBit_u32u8u8(uint32* const, uint8, boolean)

⌋()

8.5 Call-back 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 Version API

8.8.1 Bfx_GetVersionInfo

[BFX301] ⌈

Service name:	Bfx_GetVersionInfo	
Syntax:	`void Bfx_GetVersionInfo( Std_VersionInfoType* Versioninfo )`	
Service ID[hex]:	<Number of service ID, starting with 0x00. This ID is used as parameter for the error report API of Development Error Tracer>	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (in-out):	None	
Parameters (out):	Versioninfo	Pointer to where to store the version information of this module. Format according [BSW00321]
Return value:	None	

Description:	Returns the version information of this library.

⌋(BWS00407, BWS00374, BWS00379, BWS003, BWS00318, BWS00321)
The version information of a BSW module generally contains:
Module Id
Vendor Id
Vendor specific version numbers (BSW00407).

[BFX302]

⌈ If source code for caller and callee of Bfx_GetVersionInfo is available, the Bfx library should realize Bfx_GetVersionInfo as a macro defined in the module's header file. ⌋(BWS00407, BWS00411)

9 Sequence diagrams

Not applicable

10 Configuration specification

10.1 Published Information

[[BFX133]] ⌈ The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋()

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

10.2 Configuration option

[BFX314] ⌈ The Bfx 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. ⌋(BSW1400001)

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

11 Not applicable requirements

[BFX999] ⌈ These requirements are not applicable to this specification. ⌋ (BWS31400012)