Document Title	Specification of Basic Software Multicore Library
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	946

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R20-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2020-11-30	R20-11	AUTOSAR Release Management	• Improved the structure of the 'error sections' of the SWS documents • CONC_643 "BSW Multicore Distribution" finalized
2019-11-28	R19-11	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration of the AUTOSAR library for atomic routines.

This library (Bmc) contains the following routines:

- flag test and set
- flag clear
- store
- load
- exchange
- compare and exchange
- fetch and add
- fetch and subtract
- fetch and or
- fetch and xor
- fetch and and
- thread fence

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 Bmc module that are not included in the [1, AUTOSAR glossary].

Abbreviation/Acronym:	Description:
Bmc	Basic Software Multicore Library
DET	Default Error Tracer
s16	Mnemonic for `sint16`, specified in AUTOSAR_SWS_PlatformTypes
s32	Mnemonic for `sint32`, specified in AUTOSAR_SWS_PlatformTypes
s64	Mnemonic for`sint64`, specified in AUTOSAR_SWS_PlatformTypes
s8	Mnemonic for `sint8`, specified in AUTOSAR_SWS_PlatformTypes
u16	Mnemonic for `uint16`, specified in AUTOSAR_SWS_PlatformTypes
u32	Mnemonic for `uint32`, specified in AUTOSAR_SWS_PlatformTypes
u64	Mnemonic for `uint64`, specified in AUTOSAR_SWS_PlatformTypes
u8	Mnemonic for `uint8`, specified in AUTOSAR_SWS_PlatformTypes

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[4] Requirements on Libraries
AUTOSAR_SRS_Libraries

[5] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes

[6] ISO/IEC 9899:2011
http://www.iso.org

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2], which is also valid for BSWMulticoreLibrary.

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

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

[SWS_BMC_00001] ⌈The Bmc module shall provide the following files: C files `Bmc_<name>.c` used to implement the library. All C files shall be pre-fixed with 'Bmc_'. The header file `Bmc.h` provides all public function prototypes and types defined by the Bmc library specification.⌋*(SRS_LIBS_00005)*

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

Option 1: `<Name>` can be a function name providing one C file per function, e.g.: `Bmc_FlagClear.c` etc.

Option 2: `<Name>` can be a common name of a group of functions:
2.1 Group by object family:
e.g.: `Bmc_u32.c`, `Bmc_u16.c`
2.2 Group by routine family:
e.g.: `Bmc_Flag.c`, `Bmc_Fetch.c`
2.3 Group by other methods (individual grouping allowed)

Option 3: `<Name>` can be removed so that a single C file shall contain all Bmc functions, e.g.: `Bmc.c`. Using one of the 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 [3], [4] 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_00303]	No description	[SWS_BMC_00016]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	[SWS_BMC_00015]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_BMC_00044]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_BMC_00015]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_BMC_00044]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_BMC_00044]

Requirement	Description	Satisfied by
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_BMC_00043]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_BMC_00043]
[SRS_BSW_00448]	Module SWS shall not contain requirements from Other Modules	[SWS_BMC_00999]
[SRS_LIBS_00001]	The functional behavior of each library functions shall not be configurable	[SWS_BMC_00045]
[SRS_LIBS_00002]	A library shall be operational before all BSW modules and application SW-Cs	[SWS_BMC_00005]
[SRS_LIBS_00003]	A library shall be operational until the shutdown	[SWS_BMC_00006]
[SRS_LIBS_00004]	Using libraries shall not pass through a port interface	[SWS_BMC_00007]
[SRS_LIBS_00005]	Each library shall provide one header file with its public interface	[SWS_BMC_00001]
[SRS_LIBS_00007]	Using a library should be documented	[SWS_BMC_00008] [SWS_BMC_00012]
[SRS_LIBS_00015]	It shall be possible to configure the microcontroller so that the library code is shared between all callers	[SWS_BMC_00009]
[SRS_LIBS_00017]	Usage of macros should be avoided	[SWS_BMC_00010]
[SRS_LIBS_00018]	A library function may only call library functions	[SWS_BMC_00011]
[SRS_LIBS_00348]	No description	[SWS_BMC_00014]
[SRS_LIBS_00437]	No description	[SWS_BMC_00013]

7 Functional specification

7.1 Error Classification

Section "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 Errors

There are no production errors.

7.1.5 Extended Production Errors

There are no extended production errors.

7.2 Initialization and Shutdown

[SWS_BMC_00005] ⌈The Bmc library shall not require an 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_BMC_00006] ⌈The Bmc library shall not require a shutdown operation phase.⌋ *(SRS_LIBS_00003)*

7.3 Using Library API

[SWS_BMC_00007] ⌈The Bmc API can be directly called from BSW modules or SWCs. No port definition is required. It is a pure function call.⌋ *(SRS_LIBS_00004)*

[SWS_BMC_00008] ⌈Using a library should be documented. If a BSW module or a SWC uses a library, the developer should add an ImplementationDependencyOnArtifact in the BSW/SWC template. minVersion and maxVersion parameters correspond to the supplier version. In case of an AUTOSAR library, these parameters may be left empty because a SWC or BSW module may rely on a library behavior, not on a sup-

plier 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_BMC_00009] ⌈The Bmc library shall be implemented in a way that the code can be shared among callers in different memory partitions.⌋ *(SRS_LIBS_00015)*

[SWS_BMC_00010] ⌈Usage of macros should be avoided. The functions should be declared as functions or inline functions.⌋ *(SRS_LIBS_00017)*

[SWS_BMC_00011] ⌈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_BMC_00012] ⌈The library, written in the C programming language, should conform to the MISRA C Standard. Please refer to SWS_BSW_00115 for more details.⌋ *(SRS_LIBS_00007)*

[SWS_BMC_00013] ⌈Each AUTOSAR library Module implementation `<library>*.c` and `<library>*.h` shall map their code to memory sections using the AUTOSAR memory mapping mechanism.⌋ *(SRS_LIBS_00437)*

[SWS_BMC_00014] ⌈Each AUTOSAR library Module implementation `<library>*.c` that uses AUTOSAR integer data types and/or the standard return type, shall include the header file `Std_Types.h`.⌋ *(SRS_LIBS_00348)*

[SWS_BMC_00015] ⌈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 one platform.⌋ *(SRS_BSW_00378, SRS_BSW_00304)*

[SWS_BMC_00016] ⌈All AUTOSAR library Modules should avoid direct use of compiler and platform specific keywords unless this library is clearly identified to be compliant only with one platform.⌋ *(SRS_BSW_00303)*

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, sint64, uint64

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 [5]. 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
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

As described in [5], the ranges for each of the base types are shown in the table below:

Base Type	Range
boolean	[TRUE, FALSE]
uint8	[0, 255]
sint8	[-128, 127]
uint16	[0, 65535]
sint16	[-32768, 32767]
uint32	[0, 429496729]
sint32	[-2147483648, 2147483647]
uint64	[0, 18446744073709551615]
sint64	[-9223372036854775808, 9223372036854775807]

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 and macros (using <TypeMn> means Type Mnemonic and <TYPEMN> means Type Mnemonic in uppercase letters)
- the real type will be used in the description of the prototypes of the routines (using <Type>).

8.2 Type definitions

None.

8.3 Macro definitions

[SWS_BMC_00017] ⌈The Bmc module shall provide the C macros `ATOMIC_<TYPEMN>_LOCK_FREE` which shall expand to constant expressions suitable for use in `#if` preprocessing directives and which indicate the lock-free property of the corresponding atomic types. The implemented macros are listed in Table 8.3.⌋*()*

Macro
ATOMIC_BOOLEAN_LOCK_FREE
ATOMIC_U8_LOCK_FREE
ATOMIC_U16_LOCK_FREE
ATOMIC_U32_LOCK_FREE
ATOMIC_U64_LOCK_FREE
ATOMIC_S8_LOCK_FREE
ATOMIC_S16_LOCK_FREE
ATOMIC_S32_LOCK_FREE
ATOMIC_S64_LOCK_FREE

Table 8.3: Atomic lock free macros

Example: Data type `uint32` is lock free, `uint64` not. Then

```
#define ATOMIC_U32_LOCK_FREE 1
#define ATOMIC_U64_LOCK_FREE 0
```

Note: This definition is similar to the one in [6].

8.4 Function definitions

Note: All atomic operations will provide sequentially consistent ordering (see https://en.cppreference.com/w/c/atomic/memory_order#Sequentially-consistent_ordering).

Note: For all APIs which provide different API instances for different argument types (e.g. u8/16/32/64 and s8/16/32/64 variants), the implementation (for reasons of alignment) may access memory locations up to N-1 bytes before or after the pointed to memory location (where N is some platform specific constant (basically depending on the platform's alignment requirements when performing atomic operations)).

8.4.1 Flag Routines

8.4.1.1 Bmc_FlagTestAndSet

[SWS_Bmc_91003] ⌈

Service Name	Bmc_FlagTestAndSet	
Syntax	`boolean Bmc_FlagTestAndSet ( volatile boolean* Object )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	boolean	The value pointed to by Object immediately before the effects
Description	Atomically sets the value pointed to by Object to true.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00019] ⌈The function `Bmc_FlagTestAndSet` atomically sets the value pointed to by Object to `TRUE`. It returns this value before the operation, i.e., `TRUE`, if it was already set and `FALSE` otherwise.⌋()

8.4.1.2 Bmc_FlagClear

[SWS_Bmc_91004] ⌈

Service Name	Bmc_FlagClear	
Syntax	`void Bmc_FlagClear ( volatile boolean* Object )`	
Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	None	
Description	Atomically sets the value pointed to by Object to false.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00021] ⌈The function `Bmc_FlagClear` atomically sets the value pointed to by Object to `FALSE`.⌋()

8.4.2 Load and Store Routines

8.4.2.1 Bmc_Load

[SWS_Bmc_91005] ⌈

Service Name	Bmc_Load_<TypeMn>	

Syntax	`<Type> Bmc_Load_<TypeMn> (`
	`  const volatile <Type>* Object`
	`)`	

Service ID [hex]	0x10 to 0x17	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	Object	–
Parameters (out)	None	
Return value	<Type>	The value pointed to by Object
Description	Atomically loads the value pointed to by Object and returns it.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00023] ⌈The `Bmc_Load_<TypeMn>` functions atomically load the value pointed to by Object and return it. The implemented functions are listed in Table 8.4.⌋()

Service ID[hex]	***Function prototype***
0x10	uint8 Bmc_Load_u8(const volatile uint8*);
0x11	uint16 Bmc_Load_u16(const volatile uint16*);
0x12	uint32 Bmc_Load_u32(const volatile uint32*);
0x13	uint64 Bmc_Load_u64(const volatile uint64*);
0x14	sint8 Bmc_Load_s8(const volatile sint8*);
0x15	sint16 Bmc_Load_s16(const volatile sint16*);
0x16	sint32 Bmc_Load_s32(const volatile sint32*);
0x17	sint64 Bmc_Load_s64(const volatile sint64*);

Table 8.4: List of implemented functions for `Bmc_Load_<TypeMn>`

8.4.2.2 Bmc_Store

[SWS_Bmc_91006] ⌈

Service Name	Bmc_Store_<TypeMn>

▽

△

Syntax	`void Bmc_Store_<TypeMn> (`
	`  volatile <Type>* Object,`
	`  <Type> Desired`
	`)`	

Service ID [hex]	0x20 to 0x27	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Desired	Value to be stored
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	None	
Description	Atomically replaces the value pointed to by Object with the value of Desired.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00025] ⌈The `Bmc_Store_<TypeMn>` functions atomically replace the value pointed to by Object with the value of Desired. The implemented functions are listed in Table 8.5.⌋()

Service ID[hex]	*Function prototype*
0x20	uint8 Bmc_Store_u8(const volatile uint8*, uint8);
0x21	uint16 Bmc_Store_u16(const volatile uint16*, uint16);
0x22	uint32 Bmc_Store_u32(const volatile uint32*, uint32);
0x23	uint64 Bmc_Store_u64(const volatile uint64*, uint64);
0x24	sint8 Bmc_Store_s8(const volatile sint8*, sint8);
0x25	sint16 Bmc_Store_s16(const volatile sint16*, sint16);
0x26	sint32 Bmc_Store_s32(const volatile sint32*, sint32);
0x27	sint64 Bmc_Store_s64(const volatile sint64*, sint64);

Table 8.5: List of implemented functions for `Bmc_Store_<TypeMn>`

8.4.2.3 Bmc_Exchange

[SWS_Bmc_91007] ⌈

Service Name	Bmc_Exchange_<TypeMn>

Syntax	`<Type> Bmc_Exchange_<TypeMn> (`
	`  const volatile <Type>* Object,`
	`  <Type> Desired`
	`)`

Service ID [hex]	0x30 to 0x37
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	Desired	Value to be stored
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	<Type>	The value pointed to by Object immediately before the effects
Description	Atomically replaces the value pointed to by Object with the value of Desired and returns the value pointed to by Object immediately before the effects.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00027] ⌈The `Bmc_Exchange_<TypeMn>` functions atomically replace the value pointed to by Object with the value of Desired and return the value pointed to by Object immediately before the effects. The implemented functions are listed in Table 8.6.⌋()

Service ID[hex]	***Function prototype***
0x30	uint8 Bmc_Exchange_u8(volatile uint8*, uint8);
0x31	uint16 Bmc_Exchange_u16(volatile uint16*, uint16);
0x32	uint32 Bmc_Exchange_u32(volatile uint32*, uint32);
0x33	uint64 Bmc_Exchange_u64(volatile uint64*, uint64);
0x34	sint8 Bmc_Exchange_s8(volatile sint8*, sint8);
0x35	sint16 Bmc_Exchange_s16(volatile sint16*, sint16);
0x36	sint32 Bmc_Exchange_s32(volatile sint32*, sint32);
0x37	sint64 Bmc_Exchange_s64(volatile sint64*, sint64);

Table 8.6: List of implemented functions for `Bmc_Exchange_<TypeMn>`

8.4.2.4 Bmc_CompareExchange

[SWS_Bmc_91008] ⌈

Service Name	Bmc_CompareExchange_<TypeMn>	

Syntax	`boolean Bmc_CompareExchange_<TypeMn> (`
	`  volatile <Type>* Object,`
	`  <Type>* Expected,`
	`  <Type> Desired`
	`)`	

Service ID [hex]	0x40 to 0x47	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Desired	Value to be stored
Parameters (inout)	Object	Object
	Expected	Value to be stored
Parameters (out)	None	

▽

△

Return value	boolean	The result of the comparison
Description	Atomically compares the value pointed to by Object for equality with that in Expected, and if true, replaces the value pointed to by Object with Desired, and if false, updates the value in Expected with the value pointed to by Object.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00029] ⌈The `Bmc_CompareExchange_<TypeMn>` functions atomically compare the value pointed to by Object for equality with that in Expected, and if true, replace the value pointed to by Object with Desired, and if false, update the value in Expected with the value pointed to by Object. The implemented functions are listed in Table 8.7.⌋()

Service ID[hex]	***Function prototype***
0x40	boolean Bmc_CompareExchange_u8(volatile uint8*, uint8*, uint8);
0x41	boolean Bmc_CompareExchange_u16(volatile uint16*, uint16*, uint16);
0x42	boolean Bmc_CompareExchange_u32(volatile uint32*, uint32*, uint32);
0x43	boolean Bmc_CompareExchange_u64(volatile uint64*, uint64*, uint64);
0x44	boolean Bmc_CompareExchange_s8(volatile sint8*, sint8*, sint8);
0x45	boolean Bmc_CompareExchange_s16(volatile sint16*, sint16*, sint16);
0x46	boolean Bmc_CompareExchange_s32(volatile sint32*, sint32*, sint32);
0x47	boolean Bmc_CompareExchange_s64(volatile sint64*,sint64*,sint64);

Table 8.7: List of implemented functions for `Bmc_CompareExchange_<TypeMn>`

8.4.3 Fetch Routines

8.4.3.1 Bmc_FetchAdd

[SWS_Bmc_91009] ⌈

Service Name	Bmc_FetchAdd_<TypeMn>	

Syntax	`<Type> Bmc_FetchAdd_<TypeMn> (`
	`  volatile <Type>* Object,`
	`  <Type> Operand`
	`)`	

Service ID [hex]	0x50 to 0x57	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Operand	Value for the operation
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	<Type>	The value pointed to by Object immediately before the effects

▽

△

Description	Atomically replaces the value pointed to by Object with the result of the addition applied to the value pointed to by Object and the given Operand.
Available via	Bmc.h

⌋()

[SWS_BMC_00031] ⌈The `Bmc_FetchAdd_<TypeMn>` functions atomically replace the value pointed to by Object with the result of the addition applied to the value pointed to by Object and the given Operand and return the value pointed to by Object immediately before the effects. The implemented functions are listed in Table 8.8.⌋()

Service ID[hex]	***Function prototype***
0x50	uint8 Bmc_FetchAdd_u8(volatile uint8*, uint8);
0x51	uint16 Bmc_FetchAdd_u16(volatile uint16*, uint16);
0x52	uint32 Bmc_FetchAdd_u32(volatile uint32*, uint32);
0x53	uint64 Bmc_FetchAdd_u64(volatile uint64*, uint64);
0x54	sint8 Bmc_FetchAdd_u8(volatile sint8*, sint8);
0x55	sint16 Bmc_FetchAdd_u16(volatile sint16*, sint16);
0x56	sint32 Bmc_FetchAdd_u32(volatile sint32*, sint32);
0x57	sint64 Bmc_FetchAdd_u64(volatile sint64*, sint64);

Table 8.8: List of implemented functions for `Bmc_FetchAdd_<TypeMn>`

8.4.3.2 Bmc_FetchSub

[SWS_Bmc_91010] ⌈

Service Name	Bmc_FetchSub_<TypeMn>	
Syntax	`<Type> Bmc_FetchSub_<TypeMn> ( volatile <Type>* Object, <Type> Operand )`	
Service ID [hex]	0x60 to 0x67	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Operand	Value for the operation
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	<Type>	The value pointed to by Object immediately before the effects
Description	Atomically replaces the value pointed to by Object with the result of the subtraction applied to the value pointed to by Object and the given Operand.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00033] ⌈The `Bmc_FetchSub_<TypeMn>` functions atomically replace the value pointed to by Object with the result of the subtraction applied to the value

pointed to by Object and the given Operand and return the value pointed to by Object immediately before the effects. The implemented functions are listed in Table 8.9.⌋()

Service ID[hex]	Function prototype
0x60	uint8 Bmc_FetchSub_u8(volatile uint8*, uint8);
0x61	uint16 Bmc_FetchSub_u16(volatile uint16*, uint16);
0x62	uint32 Bmc_FetchSub_u32(volatile uint32*, uint32);
0x63	uint64 Bmc_FetchSub_u64(volatile uint64*, uint64);
0x64	sint8 Bmc_FetchSub_u8(volatile sint8*, sint8);
0x65	sint16 Bmc_FetchSub_u16(volatile sint16*, sint16);
0x66	sint32 Bmc_FetchSub_u32(volatile sint32*, sint32);
0x67	sint64 Bmc_FetchSub_u64(volatile sint64*, sint64);

Table 8.9: List of implemented functions for `Bmc_FetchSub_<TypeMn>`

8.4.3.3 Bmc_FetchOr

[SWS_Bmc_91011] ⌈

Service Name	Bmc_FetchOr_<TypeMn>	

Syntax	`<Type> Bmc_FetchOr_<TypeMn> (`
	`  volatile <Type>* Object,`
	`  <Type> Operand`
	`)`	

Service ID [hex]	0x70 to 0x77	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Operand	Value for the operation
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	<Type>	The value pointed to by Object immediately before the effects
Description	Atomically replaces the value pointed to by Object with the result of the or-operation applied to the value pointed to by Object and the given Operand.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00035] ⌈The `Bmc_FetchOr_<TypeMn>` functions atomically replace the value pointed to by Object with the result of the or-operation applied to the value pointed to by Object and the given Operand and return the value pointed to by Object immediately before the effects. The implemented functions are listed in Table 8.10.⌋()

Service ID[hex]	Function prototype
0x70	uint8 Bmc_FetchOr_u8(volatile uint8*, uint8);
0x71	uint16 Bmc_FetchOr_u16(volatile uint16*, uint16);
0x72	uint32 Bmc_FetchOr_u32(volatile uint32*, uint32);
0x73	uint64 Bmc_FetchOr_u64(volatile uint64*, uint64);
0x74	sint8 Bmc_FetchOr_u8(volatile sint8*, sint8);
0x75	sint16 Bmc_FetchOr_u16(volatile sint16*, sint16);
0x76	sint32 Bmc_FetchOr_u32(volatile sint32*, sint32);

Service ID[hex]	Function prototype
0x77	sint64 Bmc_FetchOr_u64(volatile sint64*, sint64);

Table 8.10: List of implemented functions for `Bmc_FetchOr_<TypeMn>`

8.4.3.4 Bmc_FetchXor

[SWS_Bmc_91012] ⌈

Service Name	Bmc_FetchXor_<TypeMn>	

Syntax	`<Type> Bmc_FetchXor_<TypeMn> (`
	`  volatile <Type>* Object,`
	`  <Type> Operand`
	`)`	

Service ID [hex]	0x80 to 0x87	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Operand	Value for the operation
Parameters (inout)	Object	Object
Parameters (out)	None	
Return value	<Type>	The value pointed to by Object immediately before the effects
Description	Atomically replaces the value pointed to by Object with the result of the xor-operation applied to the value pointed to by Object and the given Operand.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00037] ⌈The `Bmc_FetchXor_<TypeMn>` functions atomically replace the value pointed to by Object with the result of the xor-operation applied to the value pointed to by Object and the given Operand and return the value pointed to by Object immediately before the effects. The implemented functions are listed in Table 8.11.⌋()

Service ID[hex]	Function prototype
0x80	uint8 Bmc_FetchXor_u8(volatile uint8*, uint8);
0x81	uint16 Bmc_FetchXor_u16(volatile uint16*, uint16);
0x82	uint32 Bmc_FetchXor_u32(volatile uint32*, uint32);
0x83	uint64 Bmc_FetchXor_u64(volatile uint64*, uint64);
0x84	sint8 Bmc_FetchXor_u8(volatile sint8*, sint8);
0x85	sint16 Bmc_FetchXor_u16(volatile sint16*, sint16);
0x86	sint32 Bmc_FetchXor_u32(volatile sint32*, sint32);
0x87	sint64 Bmc_FetchXor_u64(volatile sint64*, sint64);

Table 8.11: List of implemented functions for `Bmc_FetchXor_<TypeMn>`

8.4.3.5 Bmc_FetchAnd

[SWS_Bmc_91013] ⌈

Service Name	Bmc_FetchAnd_<TypeMn>	

Syntax	`<Type> Bmc_FetchAnd_<TypeMn> (`
	`  volatile <Type>* Object,`
	`  <Type> Operand`
	`)`	

Service ID [hex]	0x90 to 0x97	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	Object	Object
	Operand	Value for the operation
Parameters (out)	None	
Return value	<Type>	The value pointed to by Object immediately before the effects
Description	Atomically replaces the value pointed to by Object with the result of the and-operation applied to the value pointed to by Object and the given Operand.	
Available via	Bmc.h	

⌋()

[SWS_BMC_00039] ⌈The `Bmc_FetchAnd_<TypeMn>` functions atomically replace the value pointed to by Object with the result of the and-operation applied to the value pointed to by Object and the given Operand and return the value pointed to by Object immediately before the effects. The implemented functions are listed in Table 8.12.⌋()

Service ID[hex]	***Function prototype***
0x90	uint8 Bmc_FetchAnd_u8(volatile uint8*, uint8);
0x91	uint16 Bmc_FetchAnd_u16(volatile uint16*, uint16);
0x92	uint32 Bmc_FetchAnd_u32(volatile uint32*, uint32);
0x93	uint64 Bmc_FetchAnd_u64(volatile uint64*, uint64);
0x94	sint8 Bmc_FetchAnd_u8(volatile sint8*, sint8);
0x95	sint16 Bmc_FetchAnd_u16(volatile sint16*, sint16);
0x96	sint32 Bmc_FetchAnd_u32(volatile sint32*, sint32);
0x97	sint64 Bmc_FetchAnd_u64(volatile sint64*, sint64);

Table 8.12: List of implemented functions for `Bmc_FetchAnd_<TypeMn>`

8.4.4 Fence Routines

8.4.4.1 Bmc_ThreadFence

[SWS_Bmc_91014] ⌈

Service Name	Bmc_ThreadFence
Syntax	`void Bmc_ThreadFence ( void )`
Service ID [hex]	0x03
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Creates a sequentially consistent acquire and release fence. An acquire and release fence instruction prevents the memory reordering of any read or write which precedes it in program order with any read or write which follows it in program order.
Available via	Bmc.h

⌋()

[SWS_BMC_00041] ⌈The function `Bmc_ThreadFence` creates a sequentially consistent acquire and release fence.⌋()

Note: It may also serve as a compiler barrier which stops the compiler from moving instructions across it either way for optimization purposes. Any instruction that occurs in program order before this instruction will not be reordered after this instruction. Every instruction that occurs after this instruction will not be reordered before this instruction.

8.4.5 Version API

8.4.5.1 Bmc_GetVersionInfo

[SWS_Bmc_91015] ⌈

Service Name	Bmc_GetVersionInfo	
Syntax	`void Bmc_GetVersionInfo ( Std_VersionInfoType* Versioninfo )`	
Service ID [hex]	0xFF	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	Versioninfo	Pointer to where to store the version information of this module. Format according [BSW00321]
Return value	None	

▽

△

Description	Returns the version information of this library.
Available via	Bmc.h

⌋()

[SWS_BMC_00043] ⌈If source code for caller and callee of `Bmc_GetVersionInfo` is available, the Bmc library should realize `Bmc_GetVersionInfo` as a macro defined in the module's header file.⌋(*SRS_BSW_00407, SRS_BSW_00411*)

8.5 Callback notifications

None.

8.6 Scheduled functions

The Bmc 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.

9 Sequence diagrams

Not applicable.

10 Configuration specification

10.1 Published Information

[SWS_BMC_00044] ⌈The standardized common published parameters as required by SRS_BSW_00402 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⌋ *(SRS_BSW_00402, SRS_BSW_00374, SRS_BSW_00379)*

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

10.2 Configuration Option

[SWS_BMC_00045] ⌈The Bmc 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 resource consumption optimization.

A Not applicable requirements

[SWS_BMC_00999] ⌈These requirements are not applicable to this specification.⌋ *(SRS_BSW_00448)*