Document Title	Specification of Platform Types
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	48

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	• Chapter 7.6 "Error classification added" • "VoidPtr" and "ConstVoidPtr" added • Document converted from Word to LaTeX
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes. • Wrong "Available via" references fixed. • Changed Document Status from Final to published.
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes. • Clarifications.
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes.
2016-11-30	4.3.0	AUTOSAR Release Management	• Support for 64 bit MCU's added. • Editorial changes.
2015-07-31	4.2.2	AUTOSAR Release Management	• Float types shall follow the appropriate binary interchange format of IEEE 754-2008. • Editorial changes.

2014-10-31	4.2.1	AUTOSAR Release Management	• Removed SWS_Platform_00063 as the influence of Post-build time configuration parameters on header files is already specified in SWS_BSWGeneral.
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial changes.
2013-10-31	4.1.2	AUTOSAR Release Management	• Types `uint64` and `sint64` added. • Editorial changes. • Removed chapter(s) on change documentation.
2013-03-15	4.1.1	AUTOSAR Administration	• Editorial changes.
2011-12-22	4.0.3	AUTOSAR Administration	• Clarified use of operators for `boolean` variables. • Implemented new traceability mechanism.
2010-09-30	3.1.5	AUTOSAR Administration	• Detailed published parameter names (module names) in chapter 10. The previous definition was ambigous across several releases. • Changed "Module Short Name" (MSN) to "Module Abbreviation" (MAB) for the use of API service prefixes such as "CanIf".
2010-02-02	3.1.4	AUTOSAR Administration	• Restored PLATFORM012. • Clarified endian support. • Clarified support for variable register width architectures. • Legal disclaimer revised.
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised.

2007-12-21	3.0.1	AUTOSAR Administration	• Chapter 8.2: "AUTOSAR supports for compiler and target implementation only 2 complement arithmetic". • Chapter 12.10: Changed the basic type for `*_least` types (optimized types) from `int` to `long` for SHx processors. • Removal the explicit cast to `boolean` in the precompile definition (`#define`) for macros `TRUE` and `FALSE` ("`#define TRUE ( (boolean) 1)`" has become "`#define TRUE 1`"). • Document meta information extended. • Small layout adaptations made.
2007-01-24	2.1.15	AUTOSAR Administration	• `boolean` type has been defined as an eight bit `long unsigned integer`. • Legal disclaimer revised. • Release Notes added. • "Advice for users" revised. • "Revision Information" added.
2006-05-16	2.0	AUTOSAR Administration	• Second release.
2005-05-31	1.0	AUTOSAR Administration	• First release.

Disclaimer

Table of Contents

1 Introduction and functional overview 7

2 Acronyms and Abbreviations 8

3 Related documentation 9

3.1 Input documents & related standards and norms 9
3.2 Related specification 10

4 Constraints and assumptions 11

4.1 Limitations 11
4.2 Applicability to car domains 11
4.3 Applicability to safety related environments 11

5 Dependencies to other modules 12

5.1 File structure 12
5.1.1 Code file structure 12
5.1.2 Header file structure 12

6 Requirements Tracing 13

7 Functional specification 20

7.1 General issues 20
7.2 CPU Type 20
7.3 Endianess 20
7.3.1 Bit Ordering (Register) 20
7.3.2 Byte Ordering (Memory) 21
7.4 Optimized integer data types 23
7.5 Boolean data type 23
7.6 Error classification 24
7.6.1 Development Errors 24
7.6.2 Runtime Errors 24
7.6.3 Transient Faults 24
7.6.4 Production Errors 24
7.6.5 Extended Production Errors 24

8 API specification 25

8.1 Imported types 25
8.2 Type definitions 25
8.2.1 boolean 25
8.2.2 uint8 25
8.2.3 uint16 26
8.2.4 uint32 26
8.2.5 uint64 26
8.2.6 sint8 27
8.2.7 sint16 27

8.2.8 sint32 . . . 27
8.2.9 sint64 . . . 28
8.2.10 uint8_least . . . 28
8.2.11 uint16_least . . . 28
8.2.12 uint32_least . . . 29
8.2.13 sint8_least . . . 29
8.2.14 sint16_least . . . 30
8.2.15 sint32_least . . . 30
8.2.16 float32 . . . 30
8.2.17 float64 . . . 31
8.2.18 VoidPtr . . . 31
8.2.19 ConstVoidPtr . . . 31
8.3 Symbol definitions . . . 32
8.3.1 CPU_TYPE . . . 32
8.3.2 CPU_BIT_ORDER . . . 32
8.3.3 CPU_BYTE_ORDER . . . 32
8.3.4 TRUE, FALSE . . . 33
8.4 Function definitions . . . 33
8.5 Call-back notifications . . . 33
8.6 Scheduled functions . . . 34
8.7 Expected Interfaces . . . 34
9 Sequence diagrams 35
10 Configuration specification 36
10.1 Published parameters . . . 36
A Annex 37
A.1 Type definitions - general . . . 37
A.2 Type definitions - S12X . . . 37
A.3 Type definitions - ST10 . . . 37
A.4 Type definitions - ST30 . . . 38
A.5 Type definitions - V850 . . . 39
A.6 Type definitions - MPC5554 . . . 39
A.7 Type definitions - TC1796/TC1766 . . . 40
A.8 Type definitions - MB91F . . . 40
A.9 Type definitions - M16C/M32C . . . 41
A.10 Type definitions - SHx . . . 42
A.11 Type definitions - ARM Cortex A53 . . . 42
B Not applicable requirements 44

1 Introduction and functional overview

This document specifies the AUTOSAR platform types header file. It contains all platform dependent types and symbols. Those types must be abstracted in order to become platform and compiler independent.

It is required that all platform types files are unique within the AUTOSAR community to guarantee unique types per platform and to avoid type changes when moving a software module from platform A to B.

2 Acronyms and Abbreviations

Acronyms and abbreviations that have a local scope are not contained in the AUTOSAR glossary. These must appear in a local glossary.

Acronym	*Description*
Rollover mechanism	The following example sequence is called 'rollover': • An `unsigned char` has the value of 255. • It is incremented by 1. • The result is 0.
SDU	Service Data Unit (payload)

Abbreviation	*Description*
int	Integer

3 Related documentation

3.1 Input documents & related standards and norms

[1] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[2] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[3] ISO/IEC 9899:1990
http://www.iso.org

[4] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[5] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[6] Cosmic C Cross Compiler User's Guide for Motorola MC68HC12, V4.5

[7] Metrowerks CodeWarrior 4.0 for Freescale HC9S12X/XGATE (V5.0.25)
Motorola HC12 Assembler, 2.6.2004

[8] Metrowerks CodeWarrior 4.0 for Freescale HC9S12X/XGATE (V5.0.25)
Motorola HC12 Compiler, 2.6.2004

[9] Metrowerks CodeWarrior 4.0 for Freescale HC9S12X/XGATE (V5.0.25)
Smart Linker, 2.4.2004

[10] TASKING for ST10 V8.5
C166/ST10 v8.5 C Cross-Compiler User's Manual, V5.16

[11] TASKING for ST10 V8.5
C166/ST10 v8.5 C Cross-Assembler, Linker/Locator, Utilities User's Manual, V5.16

[12] GreenHills MULTI for V850 V4.0.5
Building Applications for Embedded V800, V4.0, 30.1.2004

[13] Wind River (Diab Data) for PowerPC Version 5.2.1
Wind River Compiler for Power PC - Getting Started, Edition 2, 8.5.2004

[14] Wind River (Diab Data) for PowerPC Version 5.2.1
Wind River Compiler for Power PC - User's Guide,Edition 2, 11.5.2004

[15] TASKING for TriCore TC1796 V2.1R1
TriCore v2.0 C Cross-Compiler, Assembler, Linker User's Guide V1.2

[16] ARM ADS compiler manual

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules (see [4]), which is also valid for Platform Types. Thus, the specification "General Specification on Basic Software modules" [4] shall be considered as additional and required specification for Platform Types.

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

4.3 Applicability to safety related environments

The AUTOSAR `boolean` type may be used if the correct usage (see [SWS_Platform_00027]) is proven by a formal code review or a static analysis by a validated static analysis tool.

The optimized AUTOSAR integer data types (`*_least`) may be used if the correct usage (see chapter 7.4) is proven by a formal code review or a static analysis by a validated static analysis tool.

5 Dependencies to other modules

None.

5.1 File structure

5.1.1 Code file structure

None

5.1.2 Header file structure

Two header file structures are applicable. One is depending on communication related basic software modules and the second is depending on non-communication related basic software modules.

6 Requirements Tracing

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

Requirement	Description	Satisfied by
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_Platform_00063]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Platform_00063]
[SRS_BSW_00009]	All Basic SW Modules shall be documented according to a common standard.	[SWS_Platform_00063]
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_Platform_00063]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Platform_00063]
[SRS_BSW_00158]	No description	[SWS_Platform_00063]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_Platform_00063]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_Platform_00063]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_Platform_00063]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_Platform_00063]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_Platform_00063]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Platform_00063]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_Platform_00063]

Requirement	Description	Satisfied by
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_Platform_00063]
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_Platform_00063]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_Platform_00063]
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	[SWS_Platform_00063]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_Platform_00063]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_Platform_00063]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	[SWS_Platform_00013] [SWS_Platform_00014] [SWS_Platform_00015] [SWS_Platform_00016] [SWS_Platform_00017] [SWS_Platform_00018] [SWS_Platform_00020] [SWS_Platform_00021] [SWS_Platform_00022] [SWS_Platform_00023] [SWS_Platform_00024] [SWS_Platform_00025]
[SRS_BSW_00305]	Data types naming convention	[SWS_Platform_00063]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Platform_00063]
[SRS_BSW_00307]	Global variables naming convention	[SWS_Platform_00063]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Platform_00063]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_Platform_00063]
[SRS_BSW_00310]	API naming convention	[SWS_Platform_00063]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Platform_00063]

Requirement	Description	Satisfied by
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_Platform_00063]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_Platform_00063]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Platform_00063]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_Platform_00063]
[SRS_BSW_00326]	No description	[SWS_Platform_00063]
[SRS_BSW_00327]	Error values naming convention	[SWS_Platform_00063]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_Platform_00063]
[SRS_BSW_00329]	No description	[SWS_Platform_00063]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Platform_00063]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_Platform_00063]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_Platform_00063]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_Platform_00063]
[SRS_BSW_00335]	Status values naming convention	[SWS_Platform_00063]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Platform_00063]
[SRS_BSW_00337]	Classification of development errors	[SWS_Platform_00063]
[SRS_BSW_00338]	No description	[SWS_Platform_00063]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_Platform_00063]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_Platform_00063]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_Platform_00063]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_Platform_00063]

Requirement	Description	Satisfied by
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Platform_00063]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_Platform_00063]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_Platform_00063]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Platform_00063]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Platform_00063]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors.	[SWS_Platform_00063]
[SRS_BSW_00355]	No description	[SWS_Platform_00063]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_Platform_00063]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_Platform_00063]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_Platform_00063]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_Platform_00063]
[SRS_BSW_00361]	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	[SWS_Platform_00063]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Platform_00063]
[SRS_BSW_00370]	No description	[SWS_Platform_00063]
[SRS_BSW_00371]	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	[SWS_Platform_00063]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_Platform_00063]

Requirement	Description	Satisfied by
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_Platform_00063]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_Platform_00063]
[SRS_BSW_00376]	No description	[SWS_Platform_00063]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_Platform_00063]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Platform_00026] [SWS_Platform_00027] [SWS_Platform_00034]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_Platform_00063]
[SRS_BSW_00381]	No description	[SWS_Platform_00063]
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	[SWS_Platform_00063]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_Platform_00063]
[SRS_BSW_00385]	List possible error notifications	[SWS_Platform_00063]
[SRS_BSW_00386]	The BSW shall specify the configuration for detecting an error	[SWS_Platform_00063]
[SRS_BSW_00387]	No description	[SWS_Platform_00063]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_Platform_00063]
[SRS_BSW_00389]	Containers shall have names	[SWS_Platform_00063]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_Platform_00063]
[SRS_BSW_00391]	No description	[SWS_Platform_00063]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_Platform_00063]
[SRS_BSW_00393]	Parameters shall have a range	[SWS_Platform_00063]
[SRS_BSW_00394]	The Basic Software Module specifications shall specify the scope of the configuration parameters	[SWS_Platform_00063]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[SWS_Platform_00063]
[SRS_BSW_00396]	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/ container	[SWS_Platform_00063]

Requirement	Description	Satisfied by
[SRS_BSW_00397]	The configuration parameters in pre-compile time are fixed before compilation starts	[SWS_Platform_00063]
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	[SWS_Platform_00063]
[SRS_BSW_00399]	Parameter-sets shall be located in a separate segment and shall be loaded after the code	[SWS_Platform_00063]
[SRS_BSW_00400]	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	[SWS_Platform_00063]
[SRS_BSW_00401]	Documentation of multiple instances of configuration parameters shall be available	[SWS_Platform_00063]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Platform_00063]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Platform_00063]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_Platform_00063]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Platform_00063]
[SRS_BSW_00408]	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	[SWS_Platform_00063]
[SRS_BSW_00409]	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	[SWS_Platform_00063]
[SRS_BSW_00410]	Compiler switches shall have defined values	[SWS_Platform_00063]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Platform_00063]
[SRS_BSW_00412]	No description	[SWS_Platform_00063]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_Platform_00063]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Platform_00063]

Requirement	Description	Satisfied by
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_Platform_00063]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_Platform_00063]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	[SWS_Platform_00063]
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	[SWS_Platform_00063]
[SRS_BSW_00420]	No description	[SWS_Platform_00063]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the DEM	[SWS_Platform_00063]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_Platform_00063]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_Platform_00063]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_Platform_00063]

7 Functional specification

7.1 General issues

[SWS_Platform_00002] ⌈It is not allowed to add any extension to this file. Any extension invalidates the AUTOSAR conformity.⌋*()*

7.2 CPU Type

[SWS_Platform_00044] ⌈For each platform the register width of the CPU used shall be indicated by defining CPU_TYPE.⌋*()*

[SWS_Platform_00045] ⌈According to the register width of the CPU used, CPU_TYPE shall be assigned to one of the symbols CPU_TYPE_8, CPU_TYPE_16, CPU_TYPE_32 or CPU_TYPE_64.⌋*()*

7.3 Endianess

The pattern for bit, byte and word ordering in native types, such as integers, is called *endianess*.

[SWS_Platform_00043] ⌈For each platform the appropriate bit order on register level shall be indicated in the platform types header file using the symbol CPU_BIT_ORDER.⌋*()*

[SWS_Platform_00046] ⌈For each platform the appropriate byte order on memory level shall be indicated in the platform types header file using the symbol CPU_BYTE_ORDER.⌋*()*

7.3.1 Bit Ordering (Register)

[SWS_Platform_00048] ⌈In case of Big Endian bit ordering CPU_BIT_ORDER shall be assigned to MSB_FIRST in the platform types header file.⌋*()*

[SWS_Platform_00049] ⌈In case of Little Endian bit ordering CPU_BIT_ORDER shall be assigned to LSB_FIRST in the platform types header file.⌋*()*

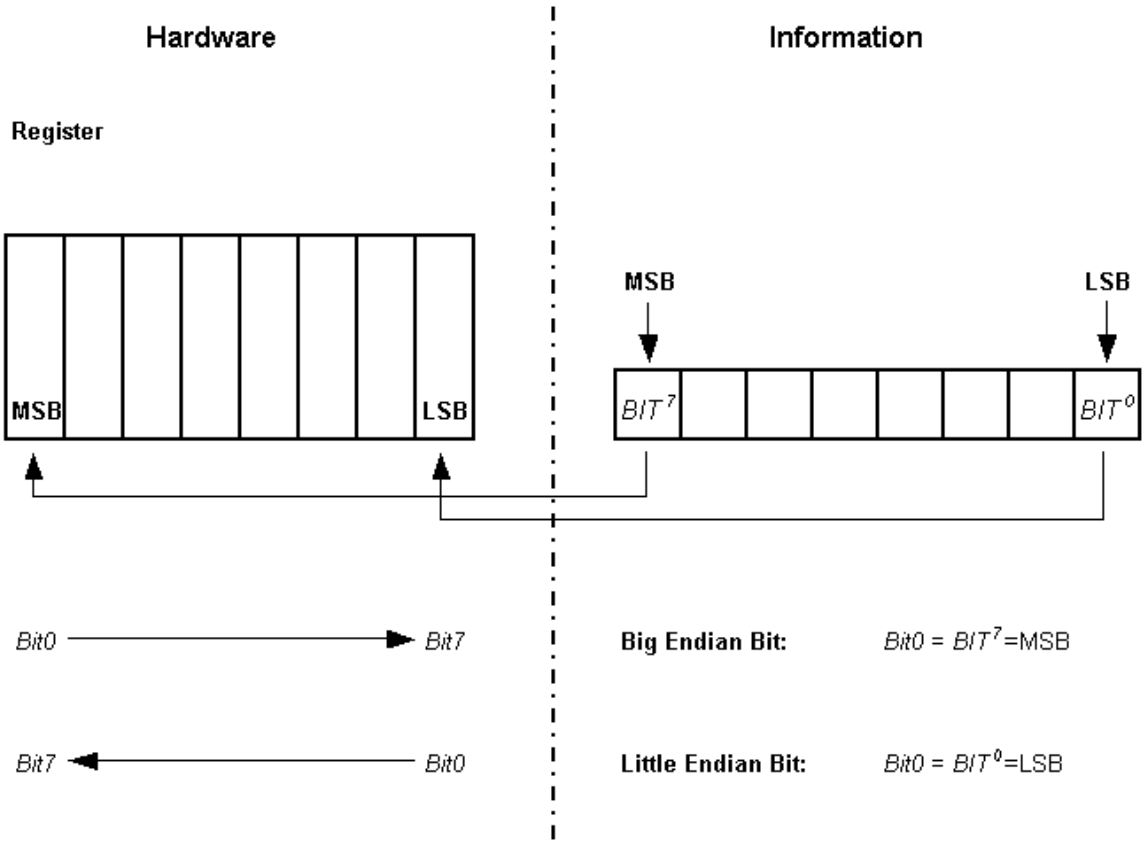

Figure 7.1: Big Endian bit ordering versus Little Endian bit ordering

Important Note:

The *naming* convention Bit0, Bit1, etc. and the bit's *significance* within a byte, word, etc. are different topics and shall not be mixed. The counting scheme of bits in Motorola[6] μC-architecture's (Big Endian Bit Order) starts with Bit0 indicating the Most Significant Bit, whereas all other μC using Little Endian Bit Order assign Bit0 to be the Least Significant Bit!

The MSB in an accumulator is always stored as the left-most bit regardless of the CPU type. Hence, Big and Little Endianess bit orders imply different bit-naming conventions.

7.3.2 Byte Ordering (Memory)

[SWS_Platform_00050] ⌈In case of Big Endian byte ordering `CPU_BYTE_ORDER` shall be assigned to `HIGH_BYTE_FIRST` in the platform types header file.⌋*()*

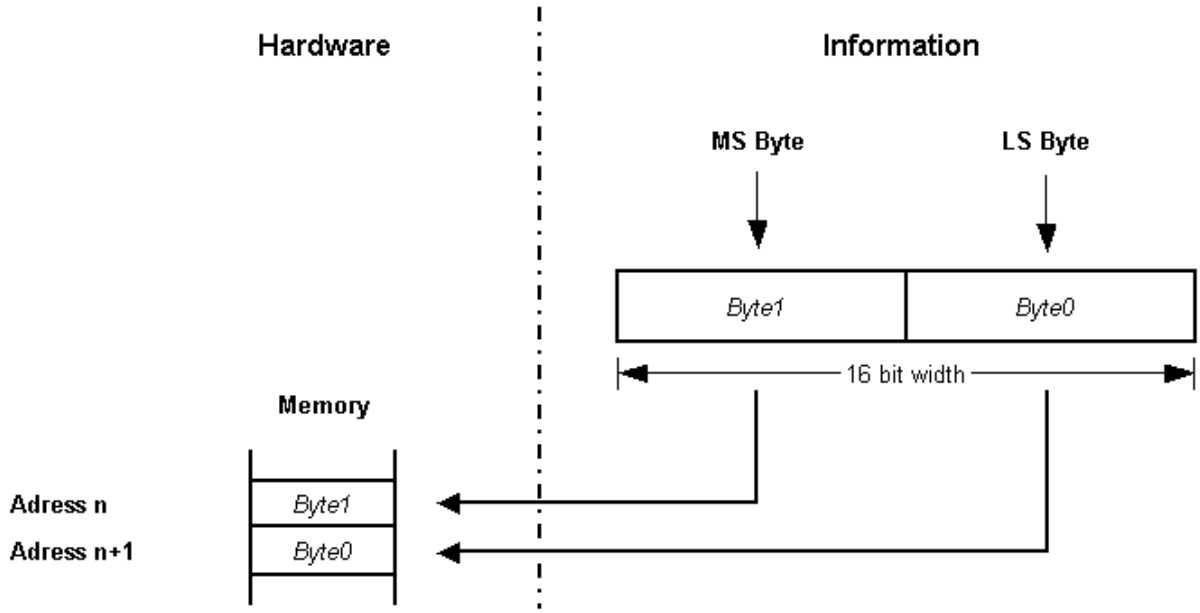

Figure 7.2: Big Endian (HIGH_BYTE_FIRST) byte ordering

Address	*Data*	*Order*
n	Byte1	Most Significant Byte (HIGH_BYTE_FIRST)
n+1	Byte0	Least Significant Byte

[SWS_Platform_00051] ⌈In case of Little Endian byte ordering CPU_BYTE_ORDER shall be assigned to LOW_BYTE_FIRST in the platform types header file.⌋ *()*

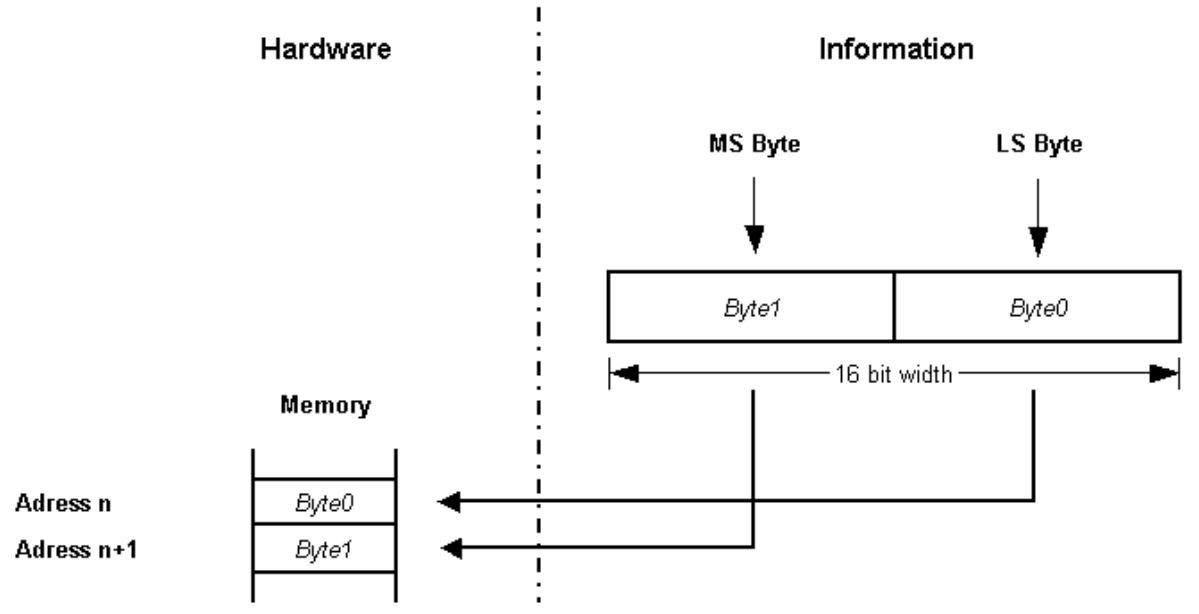

Figure 7.3: Little Endian (LOW_BYTE_FIRST) byte ordering

Address	*Data*	*Order*
n	Byte0	Least Significant Byte (LOW_BYTE_FIRST)
n+1	Byte1	Most Significant Byte

Naming convention for illustration: The Most Significant Byte within a 16 bit wide data is named *Byte1*. The Least Significant Byte within a 16 bit wide data is named *Byte0*.

Important Note: The naming convention *Byte0* and *Byte1* is not unique and may be different in the manufacturer's reference documentation for a particular μC.

7.4 Optimized integer data types

For details refer to the chapter "AUTOSAR Integer Data Types" of the document "General Requirements on Basic Software Modules" [4].

Examples of usage:

- Loop counters (e.g. maximum loop count = 124 $\Rightarrow$ use uint8_least
- Switch case arguments (e.g. maximum number of states = 17 $\Rightarrow$ use uint8_least

7.5 Boolean data type

[SWS_Platform_00027] ⌈The standard AUTOSAR type boolean shall be implemented as an unsigned integer with a bit length that is the shortest one natively supported by the platform (in general 8 bits).⌋ *(SRS_BSW_00378)*

[SWS_Platform_00034] ⌈The standard AUTOSAR type boolean shall only be used in conjunction with the standard symbols TRUE and FALSE. For value assignments of variables of type boolean no arithmetic or logical operators (+, ++, -, --, *, /, %, <<, >>, ~, &) must be used. The only allowed forms of assignment are:

```
boolean var = TRUE;
...
var = TRUE;
var = FALSE;
var = (a < b)     /* same for ">", "<=", ">=" */
var = (c && d) /* same for "!", "||" */
var = (e != f)    /* same for "==" */
```

The only allowed forms of comparison are:

```
boolean var = FALSE;
...
```

```
3 if (var == TRUE) ...
4 if (var == FALSE) ...
5 if (var != TRUE) ...
6 if (var != FALSE) ...
7 if (var) ...
8 if (!var) ...
```

⌋(*SRS_BSW_00378*)

7.6 Error classification

Section 7.2 "Error Handling" of the document "General Specification of Basic Software Modules" [4] 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.

7.6.1 Development Errors

There are no development errors.

7.6.2 Runtime Errors

There are no runtime errors.

7.6.3 Transient Faults

There are no transient faults.

7.6.4 Production Errors

There are no production errors.

7.6.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

Not applicable.

8.2 Type definitions

[SWS_Platform_00061] ⌈Concerning the signed integer types, AUTOSAR supports for compiler and target implementation only 2 complement arithmetic. This directly impacts the chosen ranges for these types.⌋*()*

8.2.1 boolean

[SWS_Platform_00026] ⌈

Name	boolean		
Kind	Type		
Range	FALSE	0	–
	TRUE	1	–
Description	This standard AUTOSAR type shall only be used together with the definitions TRUE and FALSE.		
Variation	–		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00378)*

See [SWS_Platform_00027] for implementation and usage.

[SWS_Platform_00060] ⌈The `boolean` type shall always be mapped to a platform specific type where pointers can be applied to in order to enable a passing of parameters via API.There are specific BIT types of some HW platforms which are very efficient but where no pointers can point to.⌋*()*

8.2.2 uint8

[SWS_Platform_00013] ⌈

Name	uint8		
Kind	Type		
Range	0..255	–	0x00..0xFF
Description	This standard AUTOSAR type shall be of 8 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.3 uint16

[SWS_Platform_00014] ⌈

Name	uint16		
Kind	Type		
Range	0..65535	–	0x0000..0xFFFF
Description	This standard AUTOSAR type shall be of 16 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.4 uint32

[SWS_Platform_00015] ⌈

Name	uint32		
Kind	Type		
Range	0..4294967295	–	0x00000000..0xFFFFFFFF
Description	This standard AUTOSAR type shall be 32 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.5 uint64

[SWS_Platform_00066] ⌈

Name	uint64		
Kind	Type		
Range	0..18446744073709551615	–	0x0000000000000000..0x FFFFFFFFFFFFFFFF
Description	This standard AUTOSAR type shall be 64 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋()

8.2.6 sint8

[SWS_Platform_00016] ⌈

Name	sint8		
Kind	Type		
Range	-128..+127	–	0x80..0x7F
Description	This standard AUTOSAR type shall be of 8 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.7 sint16

[SWS_Platform_00017] ⌈

Name	sint16		
Kind	Type		
Range	-32768..+32767	–	0x8000..0x7FFF
Description	This standard AUTOSAR type shall be of 16 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.8 sint32

[SWS_Platform_00018] ⌈

Name	sint32		
Kind	Type		
Range	-2147483648..+2147483647	–	0x80000000..0x7FFFFFFF
Description	This standard AUTOSAR type shall be 32 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.9 sint64

[SWS_Platform_00067] ⌈

Name	sint64		
Kind	Type		
Range	-9223372036854775808 .. 9223372036854775807	–	0x8000000000000000 .. 0x7FFFFFFFFFFFFFFF
Description	This standard AUTOSAR type shall be 64 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋()

8.2.10 uint8_least

[SWS_Platform_00020] ⌈

Name	uint8_least		
Kind	Type		
Derived from	uint		
Range	At least 0..255	–	0x00..0xFF
Description	This optimized AUTOSAR type shall be at least 8 bit unsigned.		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

See chapter 7.4 for implementation and usage.

8.2.11 uint16_least

[SWS_Platform_00021] ⌈

Name	uint16_least		
Kind	Type		
Derived from	uint		
Range	At least 0..65535	–	0x0000..0xFFFF
Description	This optimized AUTOSAR type shall be at least 16 bit unsigned.		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

See chapter 7.4 for implementation and usage.

8.2.12 uint32_least

[SWS_Platform_00022] ⌈

Name	uint32_least		
Kind	Type		
Derived from	uint		
Range	At least 0..4294967295	–	0x00000000..0xFFFFFFFF
Description	This optimized AUTOSAR type shall be at least 32 bit unsigned.		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

See chapter 7.4 for implementation and usage.

8.2.13 sint8_least

[SWS_Platform_00023] ⌈

Name	sint8_least		
Kind	Type		
Derived from	sint		
Range	At least -128..+127	–	0x80..0x7F
Description	This optimized AUTOSAR type shall be at least 8 bit signed.		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

See chapter 7.4 for implementation and usage.

8.2.14 sint16_least

[SWS_Platform_00024] ⌈

Name	sint16_least		
Kind	Type		
Derived from	sint		
Range	At least -32768..+32767	–	0x8000..0x7FFF
Description	This optimized AUTOSAR type shall be at least 16 bit signed.		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00304)*

See chapter 7.4 for implementation and usage.

8.2.15 sint32_least

[SWS_Platform_00025] ⌈

Name	sint32_least		
Kind	Type		
Derived from	sint		
Range	At least -2147483648..+2147483647	–	0x80000000..0x7FFFFFFF
Description	This optimized AUTOSAR type shall be at least 32 bit signed.		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00304)*

See chapter 7.4 for implementation and usage.

8.2.16 float32

[SWS_Platform_00041] ⌈

Name	float32		
Kind	Type		
Range	-3.4028235e+38 .. +3.4028235e+38	–	–
Description	This standard AUTOSAR type shall follow the 32-bit binary interchange format according to IEEE 754-2008 with encoding parameters specified in chapter 3.6, table 3.5, column "binary32".		
Variation	–		
Available via	Platform_Types.h		

⌋*()*

8.2.17 float64

[SWS_Platform_00042] ⌈

Name	float64		
Kind	Type		
Range	-1.7976931348623157e+308 .. +1.7976931348623157e+308	–	–
Description	This standard AUTOSAR type shall follow the 64-bit binary interchange format according to IEEE 754-2008 with encoding parameters specified in chapter 3.6, table 3.5, column "binary64".		
Available via	Platform_Types.h		

⌋()

8.2.18 VoidPtr

[SWS_Platform_91001] ⌈

Name	VoidPtr
Kind	Pointer
Type	void*
Description	This standard AUTOSAR type shall be a void pointer Note: This type shall be used for buffers that contain data returned to the caller.
Variation	–
Available via	Platform_Types.h

⌋()

8.2.19 ConstVoidPtr

[SWS_Platform_91002] ⌈

Name	ConstVoidPtr
Kind	Const Pointer
Type	const void*
Description	This standard AUTOSAR type shall be a void pointer to const. Note: This type shall be used for buffers that are passed to the callee.
Variation	–
Available via	Platform_Types.h

⌋()

8.3 Symbol definitions

8.3.1 CPU_TYPE

[SWS_Platform_00064] ⌈

Name	CPU_TYPE		
Kind	Enumeration		
Range	CPU_TYPE_8	–	Indicating a 8 bit processor
	CPU_TYPE_16	–	Indicating a 16 bit processor
	CPU_TYPE_32	–	Indicating a 32 bit processor
	CPU_TYPE_64	–	Indicating a 64 bit processor
Description	This symbol shall be defined as #define having one of the values CPU_TYPE_8, CPU_TYPE_16, CPU_TYPE_32 or CPU_TYPE_64 according to the platform.		
Available via	Platform_Types.h		

⌋()

8.3.2 CPU_BIT_ORDER

[SWS_Platform_00038] ⌈

Name	CPU_BIT_ORDER		
Kind	Enumeration		
Range	MSB_FIRST	–	The most significant bit is the first bit of the bit sequence.
	LSB_FIRST	–	The least significant bit is the first bit of the bit sequence.
Description	This symbol shall be defined as #define having one of the values MSB_FIRST or LSB_FIRST according to the platform.		
Available via	Platform_Types.h		

⌋()

8.3.3 CPU_BYTE_ORDER

[SWS_Platform_00039] ⌈

Name	CPU_BYTE_ORDER		
Kind	Enumeration		
Range	HIGH_BYTE_FIRST	–	Within uint16, the high byte is located before the low byte.

▽

△

	LOW_BYTE_FIRST	–	Within uint16, the low byte is located before the high byte.
Description	This symbol shall be defined as #define having one of the values HIGH_BYTE_FIRST or LOW_BYTE_FIRST according to the platform.		
Available via	Platform_Types.h		

⌋()

8.3.4 TRUE, FALSE

[SWS_Platform_00056] ⌈

Name	TRUE_FALSE		
Kind	Enumeration		
Range	FALSE	0x00	–
	TRUE	0x01	–
Description	The symbols TRUE and FALSE shall be defined as follows: #ifndef TRUE #define TRUE 1 #endif #ifndef FALSE #define FALSE 0 #endif		
Available via	Platform_Types.h		

⌋()

[SWS_Platform_00054] ⌈In case of in-built compiler support of the symbols, redefinitions shall be avoided using a conditional check.⌋()

[SWS_Platform_00055] ⌈These symbols shall only be used in conjunction with the `boolean` type defined in `Platform_Types.h`.⌋()

8.4 Function definitions

Not applicable.

8.5 Call-back notifications

Not applicable.

8.6 Scheduled functions

Not applicable.

8.7 Expected Interfaces

Not applicable.

9 Sequence diagrams

Not applicable.

10 Configuration specification

10.1 Published parameters

For details refer to the chapter 10.3 "Published Information" in [4].

A Annex

A.1 Type definitions - general

The platform type files for all platforms could contain the following symbols:

```
#define CPU_TYPE_8          8
#define CPU_TYPE_16         16
#define CPU_TYPE_32         32
#define CPU_TYPE_64         64
#define MSB_FIRST           0
#define LSB_FIRST           1
#define HIGH_BYTE_FIRST     0
#define LOW_BYTE_FIRST      1
```

A.2 Type definitions - S12X

The platform types for Freescale S12X[7][8][9] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_16
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef signed char         sint8_least;
typedef unsigned char       uint8_least;
typedef signed short        sint16_least;
typedef unsigned short      uint16_least;
typedef signed long         sint32_least;
typedef unsigned long       uint32_least;
typedef float               float32;
typedef double              float64;
```

A.3 Type definitions - ST10

The platform types for ST Microelectronics ST10[10][11] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_16
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef unsigned short      uint8_least;
typedef unsigned short      uint16_least;
typedef unsigned long       uint32_least;
typedef signed short        sint8_least;
typedef signed short        sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

A.4 Type definitions - ST30

The platform types for STMicroelectronics ST30 could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef unsigned long       uint8_least;
typedef unsigned long       uint16_least;
typedef unsigned long       uint32_least;
typedef signed long         sint8_least;
typedef signed long         sint16_least;
typedef signed long         sint32_least;
```

```
typedef float               float32;
typedef double              float64;
```

A.5 Type definitions - V850

The platform types for NEC V850[12] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef unsigned long       uint8_least;
typedef unsigned long       uint16_least;
typedef unsigned long       uint32_least;
typedef signed long         sint8_least;
typedef signed long         sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

A.6 Type definitions - MPC5554

The platform types for Freescale MPC5554[13][14] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       MSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
```

```
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef unsigned long       uint8_least;
typedef unsigned long       uint16_least;
typedef unsigned long       uint32_least;
typedef signed long         sint8_least;
typedef signed long         sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

A.7 Type definitions - TC1796/TC1766

The platform types for Infineon TC1796/TC1766[15] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef unsigned long       uint8_least;
typedef unsigned long       uint16_least;
typedef unsigned long       uint32_least;
typedef signed long         sint8_least;
typedef signed long         sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

A.8 Type definitions - MB91F

The platform types for Fujitsu MB91F could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef unsigned long       uint8_least;
typedef unsigned long       uint16_least;
typedef unsigned long       uint32_least;
typedef signed long         sint8_least;
typedef signed long         sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

A.9 Type definitions - M16C/M32C

The platform types for Renesas M16C and M32C could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_16
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long  uint64;
typedef unsigned short      uint8_least;
typedef unsigned short      uint16_least;
typedef unsigned long       uint32_least;
typedef signed short        sint8_least;
typedef signed short        sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

A.10 Type definitions - SHx

The platform types for Renesas SHx could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed int          sint32;
typedef signed long long    sint64;
typedef unsigned int        uint32;
typedef unsigned long long  uint64;
typedef unsigned long       uint8_least;
typedef unsigned long       uint16_least;
typedef unsigned long       uint32_least;
typedef signed long         sint8_least;
typedef signed long         sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

A.11 Type definitions - ARM Cortex A53

The platform types for ARM Cortex A53[16] in Little Endian could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_64
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef unsigned char       uint8;
typedef unsigned short      uint16;
typedef unsigned int        uint32;
typedef unsigned long long  uint64;
typedef signed char         sint8;
typedef signed short        sint16;
typedef signed int          sint32;
typedef signed long long    sint64;
typedef unsigned int        uint8_least;
```

```
11 typedef unsigned int       uint16_least;
12 typedef unsigned int       uint32_least;
13 typedef signed int         sint8_least;
14 typedef signed int         sint16_least;
15 typedef signed int         sint32_least;
16 typedef float              float32;
17 typedef double             float64;
```

B Not applicable requirements

[SWS_Platform_00063] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00345, SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00171, SRS_BSW_00170, SRS_BSW_00419, SRS_BSW_00381, SRS_BSW_00412, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00387, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00391, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00375, SRS_BSW_00101, SRS_BSW_00416, SRS_BSW_00406, SRS_BSW_00168, SRS_BSW_00407, SRS_BSW_00423, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00336, SRS_BSW_00337, SRS_BSW_00338, SRS_BSW_00369, SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00420, SRS_BSW_00417, SRS_BSW_00323, SRS_BSW_00409, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00326, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00300, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00305, SRS_BSW_00307, SRS_BSW_00310, SRS_BSW_00373, SRS_BSW_00327, SRS_BSW_00335, SRS_BSW_00350, SRS_BSW_00408, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00314, SRS_BSW_00370, SRS_BSW_00348, SRS_BSW_00361, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00357, SRS_BSW_00377, SRS_BSW_00355, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00371, SRS_BSW_00358, SRS_BSW_00414, SRS_BSW_00376, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00329, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00009, SRS_BSW_00401, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00374, SRS_BSW_00379, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334)*