Document Title	Specification of CRC Routines
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.1.1
Document Status	Final
Part of Release	2.1
Revision	0014

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
24.01.2007	2.1.1	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
15.12.2006	2.1.0	AUTOSAR Administration	• Crc_CalculateCRC16 and Crc_CalculateCRC32 APIs, Crc_DataPtr parameter : void pointer changed to uint8 pointer • Legal disclaimer revised
28.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template. • UML model introduction • Requirements traceability update • Reentrancy at calculating CRC with hardware support
31.05.2005	1.0.0	AUTOSAR Administration	Initial Release

Page left intentionally blank

Disclaimer

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module CRC.

The CRC library contains the following routines for CRC calculation:

- CRC16
- CRC32

CRC001: For both routines the following calculation options can be chosen:

- Table based calculation
  Fast execution, but larger code size (ROM table)
- Runtime calculation
  Slower execution, but small code size (no ROM table)
- Hardware supported CRC calculation (device specific)
  Fast execution, less CPU time

All routines are re-entrant and can be used by multiple applications at the same time. Hardware supported CRC calculation may be supported by some devices in the future.

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:***
CRC	Cyclic Redundancy Check
ALU	Arithmetic Logic Unit

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[4] Requirements on Memory Services,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_Mem_Services.pdf

[5] Specification of ECU Configuration,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_ECU_Configuration.pdf

[6] A Painless Guide To CRC Error Detection Algorithms, Ross N. Williams

3.2 Related standards and norms

[7] CCITT 16-bit CRC

[8] IEEE 802.3 Ethernet 32-bit CRC

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

This section describes the relations to other modules within the basic software.

5.1 File structure

5.1.1 Code file structure

The CRC library part shall consist of the following parts:

- More C file `Crc_xxx.c` containing parts of CRC code
- An API interface `Crc.h` providing the function prototypes to access the library CRC functions
- A header file `Crc_Cfg.h` providing specific parameters for the CRC.

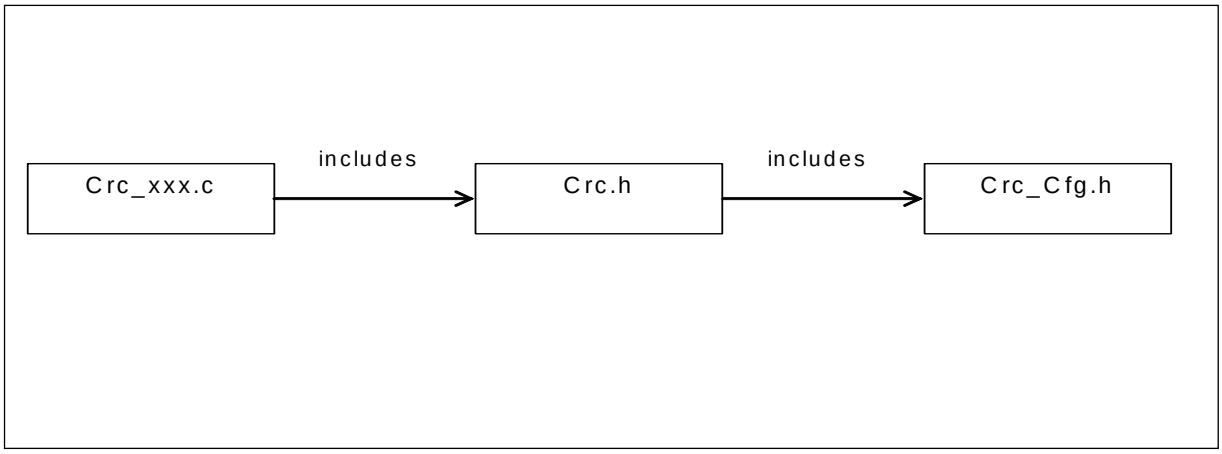


Figure 1: File structure

5.1.2 Header file structure

The include file structure shall be as follows:

- `Crc.h` shall include `Crc_Cfg.h` and `MemMap.h`
- `Crc_xxx.c` shall include `Crc.h`
- Only `Crc.h` shall be included by the customer (e.g. NVRAM Manager).

6 Requirements traceability

Document: General Requirements on Basic Software

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	Not applicable (no use case)
[BSW00404] Reference to post build time configuration	Not applicable (no use case)
[BSW00405] Reference to multiple configuration sets	Not applicable (no use case)
[BSW00345] Storage of Pre-compile-time configuration	CRC006
[BSW159] Tool-based configuration	CRC006
[BSW167] Static configuration checking	Requirement on configuration tool
[BSW171]Configurability of optional functionality	CRC006
[BSW170] Data for reconfiguration of AUTOSAR SW-components	Not applicable (no use case)
[BSW00380] Separate C-Files for configuration parameters	Not applicable (no use case)
[BSW00419] Separate C-Files for pre-compile time configuration parameters	CRC006
[BSW00381] Separate H-Files for configuration parameters	CRC006
[BSW00412] Separate H-File for configuration parameters	Not applicable (no use case)
[BSW00383] List dependencies of configuration files	Not applicable (no use case)
[BSW00384] List dependencies to other modules	Not applicable (no use case)
[BSW00387] Specify the configuration class of callback function	Not applicable (no use case)
[BSW00388] Introduce containers	Not applicable (no use case)
[BSW00389] Containers shall have names	Not applicable (no use case)
[BSW00390] Parameter content shall be unique within the module	CRC006
[BSW00391] Parameter shall have unique names	CRC006
[BSW00392] Parameters shall have a type	CRC006
[BSW00393] Parameters shall have a range	CRC006
[BSW00394] Specify the scope of the parameters	CRC006
[BSW00395] List the required parameters (per parameter)	Not applicable (no use case)
[BSW00396] Configuration classes	CRC006
[BSW00397] Pre-compile-time parameters	CRC006
[BSW00398] Link-time parameters	Not applicable (no use case)
[BSW00399] Loadable Post-build time parameters	Not applicable (no use case)
[BSW00400] Selectable Post-build time parameters	Not applicable (no use case)
[BSW00401] Creating multiplicity	Not applicable (no use case)
[BSW00402] Published information	CRC005, CRC004
[BSW00375] Notification of wake-up reason	Not applicable (no use case)

Requirement	*Satisfied by*
[BSW101] Initialization interface	Not applicable (no use case)
[BSW00416] Sequence of Initialization	Not applicable (no use case)
[BSW00406] C-Initialization routine	Not applicable (no use case)
[BSW168] Diagnostic Interface of SW components	Not applicable (no use case)
[BSW00407] Function to read out published parameters	CRC011, CRC012
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (CRC is not part of service layer)
[BSW00424] BSW main processing function task allocation	Not applicable (no use case)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (no use case)
[BSW00426] Exclusive areas in BSW modules	CRC008
[BSW00427] ISR description for BSW modules	Not applicable (no use case)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (no use case)
[BSW00429] Restricted BSW OS functionality access	Not applicable (no use case)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (no use case)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (no use case)
[BSW00433] Calling of main processing functions	Not applicable (no use case)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (to be defined in Schedule module SWS)
[BSW00336] Shutdown interface	Not applicable (no use case)
[BSW00337] Classification of errors	Not applicable (no error reported by CRC module)
[BSW00338] Detection and Reporting of development errors	Not applicable (no error reported by CRC module)
[BSW00369] Do not return development error codes via API	Not applicable (no error reported by CRC module)
[BSW00339] Reporting of production relevant errors and exceptions	Not applicable (no error reported by CRC module)
[BSW00421] Reporting of production relevant error events	Not applicable (no error reported by CRC module)
[BSW00422] Debouncing of production relevant error status	Not applicable (no error reported by CRC module)
[BSW00420] Production relevant error event rate detection	Not applicable (no error reported by CRC module)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (no error reported by CRC module)
[BSW00323] API parameter checking	Not applicable (no error reported by CRC module)
[BSW003] Version identification	CRC004
[BSW004] Version check	CRC005, CRC004
[BSW00409] Header files for production code error IDs	Not applicable (no error reported by CRC module)
[BSW00385] Configuration of error notifications	Not applicable (no error reported by CRC module)

Requirement	*Satisfied by*
[BSW00386] How errors shall be detected	Not applicable (no error reported by CRC module)
[BSW00318] Format of module version numbers	CRC004
[BSW161] Microcontroller abstraction	Not applicable (Requirement on AUTOSAR architecture, not a single module)
[BSW162] ECU layout abstraction	Not applicable (Requirement on AUTOSAR architecture, not a single module)
[BSW324] Do not use HIS I/O Library	Not applicable (architecture decision)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (Requirement on AUTOSAR architecture, not a single module)
[BSW00415] User dependent include files	Not applicable (CRC doesn't provide restricted access for several modules)
[BSW164] Implementation of interrupt service routines	Not applicable (this module doesn't implement any ISR)
[BSW00325] Runtime of interrupt service routines	Not applicable (this module doesn't implement any ISR)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (this module doesn't implement any ISR)
[BSW00342] Usage of source code and object code	Not applicable (Requirement on AUTOSAR architecture, not a single module)
[BSW00343] Specification and configuration of time	Not applicable (no configurable timings)
[BSW160] Human-readable configuration data	Not applicable (Requirement on documentation, not on specification)
[BSW007] HIS MISRA C	Not applicable (Requirement on implementation, not on specification)
[BSW00300] Module naming convention	Chapter 5.1
[BSW00413] Accessing instances of BSW modules	Conflict: This requirement will have impact on almost all BSW modules, therefore it can not be implemented within the Release 2.0 timeframe.
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (Requirement on implementation, not on specification)
[BSW00305] Self-defined data types naming convention	Not applicable (no data type definition in this specification)
[BSW00307] Global variables naming convention	Not applicable (Requirement on implementation, not on specification)
[BSW00310] API naming convention	Chapter 8.3
[BSW00373] Main processing function naming convention	Not applicable (no main function available)
[BSW00327] Error values naming convention	Not applicable (no error reported by CRC module)
[BSW00335] Status values naming convention	Not applicable (no status values available)
[BSW00350] Development error detection keyword	Not applicable (no error reported by CRC module)
[BSW00408] Configuration parameter naming convention	Chapter 10

Requirement	*Satisfied by*
[BSW00410] Compiler switches shall have defined values	Not applicable (this module doesn't implement any compiler switch)
[BSW00411] Get version info keyword	CRC011, Chapter 10
[BSW00346] Basic set of module files	Chapter 5.1
[BSW158] Separation of configuration from implementation	Chapter 5.1
[BSW00314] Separation of interrupt frames and service routines	Not applicable (this module doesn't implement any ISR)
[BSW00370] Separation of callback interface from API	Not applicable (this module doesn't implement any callback function)
[BSW00348] Standard type header	Not applicable (this module simply includes the standard type header via the module header file)
[BSW00353] Platform specific type header	Not applicable (this module simply includes the standard type header via the module header file)
[BSW00361] Compiler specific language extension header	Not applicable (this module simply includes the standard type header via the module header file)
[BSW00301] Limit imported information	Chapter 5.1
[BSW00302] Limit exported information	Not applicable (Requirement on the implementation, not on the specification)
[BSW00328] Avoid duplication of code	Not applicable (Requirement on the implementation, not on the specification)
[BSW00312] Shared code shall be reentrant	Not applicable (Requirement on the implementation, not on the specification)
[BSW006] Platform independency	Not applicable (Requirement on the implementation, not on the specification)
[BSW00357] Standard API return type	Chapters 8.2, 8.1
[BSW00377] Module specific API return types	Chapter 8.2
[BSW00304] AUTOSAR integer data types	Not applicable (Requirement on implementation, not for specification)
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (Requirement on implementation, not for specification)
[BSW00378] AUTOSAR boolean type	Not applicable (Requirement on implementation, not for specification)
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (Requirement on implementation, not for specification)
[BSW00308] Definition of global data	Not applicable (Requirement on implementation, not for specification)
[BSW00309] Global data with read-only constraint	Not applicable (Requirement on implementation, not for specification)
[BSW00371] Do not pass function pointers via API	Not applicable (no function pointers in this specification)
[BSW00358] Return type of init() functions	Not applicable (no init function in this specification)

Requirement	Satisfied by
[BSW00414] Parameter of init function	Not applicable (no init function in this specification)
[BSW00376] Return type and parameters of main processing functions7	Not applicable (no main processing function in this specification)
[BSW00359] Return type of callback functions	Not applicable (no callback function in this specification)
[BSW00360] Parameters of callback functions	Not applicable (no callback function in this specification)
[BSW00329] Avoidance of generic interfaces	Chapter 8.3
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable Requirement on implementation, not for specification)
[BSW00331] Separation of error and status values	Not applicable (no error and status values)
[BSW009] Module User Documentation	Not applicable (Requirement on documentation, not on specification)
[BSW00401] Documentation of multiple instances of configuration parameters	Chapter 10
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (no scheduling in this specification)
[BSW010] Memory resource documentation	Not applicable (Requirement on documentation, not on specification)
[BSW00333] Documentation of callback function context	Not applicable (no callback function in this specification)
[BSW00374] Module vendor identification	CRC004
[BSW00379] Module identification	CRC004
[BSW003] Version identification	CRC004
[BSW00318] Format of module version numbers	CRC004
[BSW00321] Enumeration of module version numbers	Not applicable (Requirement on implementation, not for specification)
[BSW00341] Microcontroller compatibility documentation	Not applicable (Requirement on documentation, not on specification)
[BSW00334] Provision of XML file	Not applicable (Requirement on documentation, not on specification)
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	Chapter 5

Document: Requirements on Memory Services

Requirement	Satisfied by
[BSW08518] CRC calculation method complexity	CRC001
[BSW08520] CRC routine reentrancy	CRC008
[BSW08521] CRC routine schedulability	CRC008
[BSW08524] Error detection	CRC007
[BSW08525] Support of standard polynomials	CRC002, CRC003
[BSW08526] CRC Standard calculation methods	CRC001

7 Functional specification

7.1 Basic Concepts of CRC Codes

7.1.1 Mathematical Description

Let D be a bitwise representation of data with a total number of n bit, i.e.

$D = (d_{n-1}, d_{n-2}, d_{n-3}, \ldots , d_1, d_0)$,

with $d_0, d_1, \ldots = 0b, 1b$. The corresponding *Redundant Code* C is represented by n+k bit as

$C = (D, R) = (d_{n-1}, d_{n-2}, d_{n-3}, \ldots , d_2, d_1, d_0, r_{k-1}, \ldots , r_2, r_1, r_0)$

with $r_0, r_1, \ldots = 0b,1b$. and $R = (r_{k-1}, \ldots , r_2, r_1, r_0)$ The code is simply a concatenation of the data and the redundant part. (For our application, we will chose k = 16, 32 and n as a multiple of 16 resp. 32).

CRC-Algorithms are related to *polynomials* with coefficients in the finite *field of two element*, using arithmetic operations ⊕ and * according to the following tables.

The ⊕ operation is identified as the binary operation *exclusive-or*, that is usually available in the ALU of any CPU.

⊕	*0b*	*1b*
0b	0b	1b
1b	1b	0b

*	*0b*	*1b*
0b	0b	0b
1b	0b	1b

For simplicity we will write ab instead of a*b

We introduce some Examples for *polynomials* with coefficients in the *field of two elements* and give the simplified notation of it.

(ex. 1) $p_1(X) = 1b\ X^3 + 0b\ X^2 + 1b\ X^1 + 0b\ X^0 = X^3 + X$

(ex. 2) $p_2(X) = 1b\ X^2 + 1b\ X^1 + 1b\ X^0 = X^2 + X^1 + 1b$

Any code word, represented by n+k bit can be mapped to a polynomial of order n+k-1 with coefficients in the field of two elements. We use the intuitive mapping of the bits i.e.

$C(X) = d_{n-1}X^{k+n-1} + d_{n-2}X^{k+n-2} + \ldots + d_2X^{k+2} + d_1X^{k+1} + d_0 X^k$
$+ r_{k-1}X^{k-1} + r_{k-2}Xk^{-2} + \ldots r_1 X + r_0$

$C(X) = X^k(d_{n-1}X^{n-1} + d_{n-2}X^{n-2} + \ldots + d_2X^2 + d_1X^1 + d_0)$

- $+ r_{k-1}X^{k-1} + r_{k-2}Xk^{-2} + \ldots r_1 X + r_0$

- $C(X) = X^k D(X) \oplus R(X)$

This mapping is one-to-one.

A certain space CRC_G of *Cyclic Redundant Code Polynomials* is defined to be a multiple of a given *Generator Polynomial* $G(X) = X^k + g_{k-1} X^{k-1} + g_{k-2} X^{k-2} + \ldots + g_2 X^2 + g_1 X + g_0$. By definition, for any code polynomial C(X) in CRC_G there is a polynomial M(X) with

$$C(X) = G(X) M(X)$$

For a fixed irreducible (i.e. prime-) polynomial G(X), the mapping M(X) → C(X) is one-to-one. Now, how are data of a given codeword verified? This is basically a division of polynomials, using the *Euclidian Algorithm*. In practice, we are not interested in M(X), but in the *remainder* of the division, C(X) mod G(X). For a correct code word C, this remainder has to be *zero,* C(X) mod G(X) = 0. If this is not the case – there is an error in the codeword. Given G(X) has some additional algebraic properties, one can determine the error-location and correct the codeword.

Calculating the code word from the data can also be done with the Euclidian Algorithm. For a given data polynomial $D(x) = d_{n-1}X^{n-1} + d_{n-2}X^{n-2} + \ldots + d_1X^1 + d_0$ and the corresponding code polynomial C(X) we have

$$C(X) = X^k D(X) \oplus R(X) = M(X) G(X)$$

Performing the operation "mod G(X)" on both sides, one obtains

$$0 = C(X) \bmod G(X) = [X^k D(X)] \bmod G(X) \oplus R(X) \bmod G(X) \quad (*)$$

We denote, that the order of the Polynomial R(X) is less than the order of G(X), so the modulo division gives zero with remainder R(X):

$$R(X) \bmod G(X) = R(X).$$

For polynomial R(X) with coefficients in the finite field with two elements we have the remarkable property R(X) + R(X) = 0. If we add R(X) on both sides of equation (*****) we obtain

$$R(X) = X^k D(X) \bmod G(X).$$

The important implication is, that the redundant part of the requested code can be determined by using the Euclidian Algorithm for polynomials. At present any CRC calculation method is a more or less sophisticated variation of this basic algorithm.

Up to this point a summary of the propositions about CRC Code:

1. The construction principle of CRC Codes is based on polynomials with coefficients in the finite field of two elements. The ⊕ operation of this field is identical to the binary operation "xor" (exclusive or)
2. There is a natural mapping of bit-sequences into this space of polynomials.

3. Both calculation and verification of the CRC code polynomial is based on division modulo a given generator polynomial.
4. This generator polynomial has to have certain algebraic properties, in order to achieve error-detection and eventually error-correction.

7.1.2 Euclidian Algorithm for Binary Polynomials and Bit-Sequences.

Given a Polynomial $P_n(X) = p_nX^n + p_{n-1}X^{n-1} + \ldots + p_2X^2 + p_1X + p_0$ with coefficients in the finite field of two elements. Let $Q(X) = X^k + q_{k-1}X^{k-1} + q_{k-2}X^{k-2} + \ldots + q_2X^2 + q_1X + q_0$ be another polynomial of exact order k>0. Let $R_n(X)$ be the remainder of the polynomial division,of maximum order k-1 and $M_n(X)$ corresponding so that

$R_n(X) \oplus M_n(X)\, Q(X) = P_n(X)$.

Euclidian Algorithm - Recursive

(Termination of recursion)
If n < k, then chose $R_n(X) = P_n(X)$ and $M_n = 0$.

(Recursion n+1 → n)
Let $P_{n+1}(X)$ be of maximum order n+1.
If n+1>= k calculate $P_n(X) = P_{n+1}(X) - p_{n+1}\, Q(X)\, X^{n-k+1}$. This polynomial is of maximum order n. Then

$$P_{n+1}(X) \bmod Q(X) = P_n(X) \bmod Q(X).$$

- Proof of recursion

Chose $R_{n+1}(X) = P_{n+1}(X) \bmod Q(X)$ and $M_{n+1}(X)$ so that

$$R_{n+1}(X) \oplus M_{n+1}(X)\, Q(X) = P_{n+1}(X)$$

Then $R_{n+1}(X) - R_n(X) = P_{n+1}(X) - M_{n+1}(X)\, Q(X) - P_n(X) \oplus M_n(X)\, Q(X)$

With $P_{n+1}(X) - P_n(X) = p_{n+1}\, Q(X)\, X^{n-k+1}$ we obtain

$R_{n+1}(X) - R_n(X) = p_{n+1}\, Q(X)\, X^{n-k+1} + M_n(X)\, Q(X) - M_{n+1}(X)\, Q(X)$

$R_{n+1}(X) - R_n(X) = Q(X)\, [p_{n+1}\, X^{n-k+1} + M_n(X) - M_{n+1}(X)]$

On the left side, there is a polynomial of maximum order k-1, on the right side Q(X) is of exact order k. This implies, that both sides are trivial and equal to zero. One obtains

$$R_{n+1}(X) = R_n(X) \qquad (1)$$
$$M_{n+1}(X) = M_n(X) + p_{n+1}\, X^{n-k+1} \qquad (2)$$

(end of proof)

Example
$P(X) = P_4(X) = X^4 + X^2 + X + 1b$; $Q(X) = X^2 + X + 1b$; n = 4; k = 2
$P_3(X) = X^4 + X^2 + X + 1b - 1b(X^2 + X + 1b)\ X^2 = X^3 + X + 1b.$
$P_2(X) = X^3 + X + 1b - 1b\ X\ (X^2 + X + 1b) = X^2 + 1b.$
$P_1(X) = X^2 + 1 - 1b\ (X^2 + X + 1) = X$
$R(X) = P(X) \bmod Q(X) = R_1(X) = P_1(X) = X.$

7.1.3 CRC calculation, Variations and Parameter

CRC007: Based on the Euclidian Algorithm, some variations have been developed, in order to improve the performance of calculation. All those variations do not improve the capability to detect or correct errors – the so-called Hamming Distance of the resulting code is determined only by generator polynomial. Variations simply optimize for different implementing ALUs. We give a model to get grip of the variations that are currently known. CRC-Calculation methods are characterized as follows:

1. Rule for Mapping of Data to a bit sequence (d_{n-1}, d_{n-2}, d_{n-3}, … , d_1, d_0) and the corresponding data polynomial D(X). (Standard or reflected data)
2. Generator polynomial G(X)
3. Start value and corresponding Polynomial S(X)
4. Appendix A(X), also called XOR-value for modifying the final result.
5. Rule for mapping the resulting CRC-remainder R(X) to codeword. (Standard or reflected data)

The calculation itself is organized in the steps

- Map Data to D(X).
- Perform Euclidian Algorithm on $X^k\ D(X) + X^{n-k-1}\ S(X) + A(X)$ and determine

$R(X) = [\ X^k\ D(X) + X^{n-k-1}\ S(X) + A(X)] \bmod G(X)$.

- Map D(X), R(X) to codeword.

7.2 Standard parameters

This chapter gives a rough overview of the standard parameters which are commonly used for 16-bit and 32-bit CRC calculation.

- CRC result width: Defines the result data width of the CRC calculation.
- Polynomial: Defines the generator polynomial which is used for the CRC algorithm.
- Initial value: Defines the start condition for the CRC algorithm.
- Input data reflected: Defines whether the bits of each input byte are reflected before being processed.
- Result data reflected: Similar to “Input data reflected” this parameter

defines whether the bits of the CRC result are reflected.

- XOR value: This Value is XORed to the final register value before the value is returned as the official checksum.
- Check: This field is a check value that can be used as a weak validator of implementations of the algorithm. The field contains the checksum obtained when the ASCII string "123456789" is fed through the specified algorithm.

7.2.1 16-bit CRC calculation

CRC002: The CRC16 routine is based on the CCITT CRC16 Standard:

CRC result width:	16 bits
Polynomial:	1021h
Initial value:	FFFFh
Input data reflected:	No
Result data reflected:	No
XOR value:	0000h
Check:	29B1h

7.2.2 32-bit CRC calculation

CRC003: The CRC32 routine is based on the IEEE-802.3 CRC32 Ethernet Standard:

CRC result width:	32 bits
Polynomial:	04C11DB7h
Initial value:	FFFFFFFFh
Input data reflected:	Yes
Result data reflected:	Yes
XOR value:	FFFFFFFFh
Check:	CBF43926h

7.3 General behavior

Data blocks are passed to the CRC routines by start address, size and start value. The return value is the CRC result.

7.4 Error classification

The CRC library functions do not provide any error classification. CRC recalculation and comparison must be done by each module in the upper layer (e.g. NVRAM Manager).

7.5 Error detection

The CRC library functions do not provide any error detection mechanism.

7.6 Error notification

The CRC library functions do not provide any error notification.

7.7 Version check

CRC005: Crc.c shall check if the correct version of Crc.h is included. This shall be done by a preprocessor check of the version number `CRC_MAJOR_VERSION` and `CRC_MINOR_VERSION`.

8 API specification

8.1 Imported types

8.1.1 Standard types

- Std_VersionInfoType

8.2 Type definitions

None.

8.3 Function definitions

CRC013: If CRC routines are used as a library, the module shall be developed in a way that only the parts of code that are used by other modules are linked into the final binary.

CRC014: When calculating CRC over multiple blocks, the first call should use CRC_INITIAL_VALUEXX as start value, and then for next calls the start value shall be the result of previous call.

CRC008: API specification

8.3.1 16-bit CRC Calculation

Service name:	Crc_CalculateCRC16	
Syntax:	`uint16 Crc_CalculateCRC16 ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint16 Crc_StartValue16 )`	
Service ID [hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	`Crc_DataPtr`	Pointer to start address of data block to be calculated.
	`Crc_Length`	Length of data block to be calculated in bytes.
	`Crc_StartValue16`	Initial value when the algorithm starts.
Parameters (out):	None	--
Return value:	`16 bit result of CRC calculation.`	
Description:	**CRC009:** If CRC calculation is performed by hardware then the reentrancy shall be insured by the software, by implementing locking mechanism. **CRC015:** Performs a CRC16 calculation on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue16.`	

Caveats:	If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured in order to decrease the calculation time.
Configuration:	CRC006

8.3.2 32-bit CRC Calculation

Service name:	Crc_CalculateCRC32	
Syntax:	`uint32 Crc_CalculateCRC32 ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint32 Crc_StartValue32 )`	
Service ID [hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	`Crc_DataPtr`	Pointer to start address of data block to be calculated.
	`Crc_Length`	Length of data block to be calculated in bytes.
	`Crc_StartValue32`	Initial value when the algorithm starts.
Parameters (out):	None	--
Return value:	`32 bit result of CRC calculation.`	
Description:	**CRC010:** If CRC calculation is performed by hardware then the reentrancy shall be insured by the software, by implementing locking mechanism. **CRC016:** Performs a CRC32 calculation on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue32.`	
Caveats:	If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured in order to decrease the calculation time.	
Configuration:	CRC006	

8.3.2.1 Crc_GetVersionInfo

Service name:	Crc_GetVersionInfo	
Syntax:	`void Crc_GetVersionInfo { Std_VersionInfoType *versioninfo }`	
Service ID [hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	non reentrant	
Parameters (in):	`none`	
Parameters (out):	`versioninfo`	Pointer to where to store the version information of this module.
Return value:	none	--
Description:	**CRC011:** This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). **CRC012:** This function shall be pre compile time configurable `On/Off` by the	

	configuration parameter: `CRC_VERSION_INFO_API` Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.
Caveats:	--
Configuration:	This service is only available if enabled by the pre-processor switch `CRC_VERSION_INFO_API`.

8.4 Call-back notifications

None.

8.5 Scheduled functions

None.

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

None

8.6.2 Optional Interfaces

None.

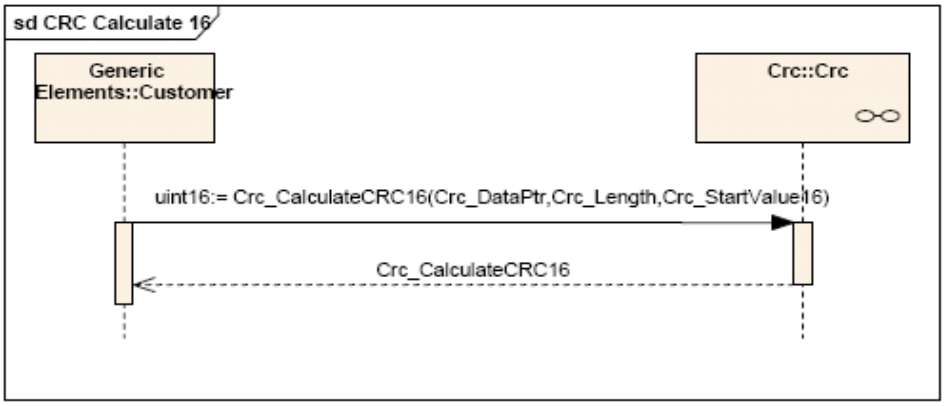
8.6.3 Configurable interfaces

None.

9 Sequence diagrams

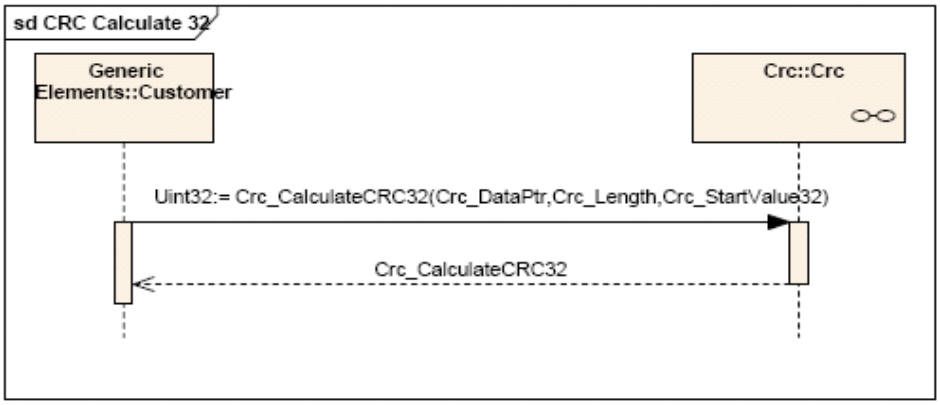
9.1 Crc_CalculateCRC16()

The following diagram shows the synchronous function call `Crc_CalculateCRC16`.



9.2 Crc_CalculateCRC32()

The following diagram shows the synchronous function call `Crc_CalculateCRC32`.



10 Configuration specification

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Architecture [2]
- AUTOSAR ECU Configuration Specification [5]: This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

10.2.1 Variants

No variants specified.

10.2.2 CRC_COMMON

SWS Item	**CRC006:** The following table specifies parameters that shall be definable in the module's configuration file (CRC_Cfg.h).
Container Name	CRC_COMMON
Description	CRC calculation selection for the CRC module
Configuration Parameters	

Name	CRC_16_MODE		
Description	Preprocessor switch to select one of the available CRC16 calculation methods.		
Type or Unit	enum		
Range	CRC_16_TABLE	table based CRC16 calculation Default selection	
	CRC_16_RUNTIME	runtime based CRC16 calculation	
	CRC_16_HARDWARE	hardware based CRC16 calculation	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	--		

Name	CRC_32_MODE		
Description	Preprocessor switch to select one of the available CRC32 calculation methods.		
Type or Unit	enum		
Range	CRC_32_TABLE	table based CRC32 calculation Default selection	
	CRC_32_RUNTIME	runtime based CRC32 calculation	
	CRC_32_HARDWARE	hardware based CRC32 calculation	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	--		

Name	CRC_VERSION_INFO_API
Description	Pre-processor switch to enable / disable the API to read out the modules version information.
Type	#define
Unit	--

Range	STD_ON	Version info API enabled.	
	STD_OFF	Version info API disabled,	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
none	--	--

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

SWS Item	**CRC004:** The following table specifies parameters that shall be published in the header file of the module.	
Information elements		
Information element name	***Type / Range***	***Information element description***
CRC_VENDOR_ID	#define/ uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
CRC_MODULE_ID	#define/ uint8	Module ID of this module from Module List
CRC_AR_MAJOR_VERSION	#define/ uint8	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
CRC_AR_MINOR_VERSION	#define/ uint8	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
CRC_AR_PATCH_VERSION	#define/ uint8	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
CRC_SW_MAJOR_VERSION	#define/ uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
CRC_SW_MINOR_VERSION	#define/ uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
CRC_SW_PATCH_VERSION	#define/ uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
CRC_INITIAL_VALUE16	#define/ uint16	Initial value for the 16-bit CRC calculation
CRC_INITIAL_VALUE32	#define/ uint32	Initial value for the 32-bit CRC calculation

11 Changes to Release 1

11.1 Deleted SWS Items

No deleted SWS items to Release 1.

11.2 Replaced SWS Items

No replaced SWS items to Release 1.

11.3 Changed SWS Items

No changed SWS items to Release 1.

11.4 Added SWS Items

SWS Item	*Rationale*
CRC009	Bugfix #5825
CRC010	Bugfix #5825
CRC011	Added due to adaptation to new SWS template.
CRC012	Added due to adaptation to new SWS template.
CRC013	Bugfix #8839
CRC014	RfC #13416
CRC015	RfC #13416
CRC016	RfC #13416