| Document Title | Specification of CRC Library |
|---|---|
| **Document Category** | Software specification (SWS) |
| **Document UID** | 16 |
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R26-11 |

**Document Change History**

| Date | Release | Changed by | Description |
|---|---|---|---|
| 2026-11-26 | R26-11 | AUTOSAR Release Management | • Removed GetVersionInfo API<br>• Minor corrections / clarifications / editorial changes |
| 2025-11-27 | R25-11 | AUTOSAR Release Management | • Removed remaining references to CRC hardware implementation<br>• Minor corrections / clarifications / editorial changes |
| 2024-11-27 | R24-11 | AUTOSAR Release Management | • Introduction of a new CRC-32 based on SAE J1939-76 Standard<br>• Minor corrections / clarifications / editorial changes |
| 2023-11-23 | R23-11 | AUTOSAR Release Management | • Removed hardware supported CRC calculation<br>• Minor corrections / clarifications / editorial changes |
| 2022-11-24 | R22-11 | AUTOSAR Release Management | • Minor corrections / clarifications / editorial changes |
| 2021-11-25 | R21-11 | AUTOSAR Release Management | • Minor corrections / clarifications / editorial changes<br>• Changed Document Status from Final to published |
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • Improved the structure of the ’error sections’ |
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • No content changes<br>• Changed Document Status from Final to published |

▽

△

| | | | |
|---|---|---|---|
| 2018-10-31 | 4.4.0 | AUTOSAR Release Management | • Introduction of a new CRC-16 with the polynomial 0x8005<br>• Editorial changes |
| 2017-12-08 | 4.3.1 | AUTOSAR Release Management | • Editorial changes |
| 2016-11-30 | 4.3.0 | AUTOSAR Release Management | • Introduction of a new CRC-64 for E2E Profile 7<br>• Editorial changes |
| 2015-07-31 | 4.2.2 | AUTOSAR Release Management | • Corrected the magic check for the CRC32 and CRC32P4 |
| 2014-10-31 | 4.2.1 | AUTOSAR Release Management | • Introduction of a new CRC-32 with the polynomial 0xF4ACFB13<br>• Editorial changes |
| 2014-03-31 | 4.1.3 | AUTOSAR Release Management | • CRC32 IEEE 802.3 check values corrected<br>• Editorial changes |
| 2013-10-31 | 4.1.2 | AUTOSAR Release Management | • Editorial changes<br>• Removed chapter(s) on change documentation |
| 2013-03-15 | 4.1.1 | AUTOSAR Administration | • New examples on how to use CRC routines and clarifications concerning CCITT standard<br>• Removal of debugging concept |
| 2011-12-22 | 4.0.3 | AUTOSAR Administration | • The GetVersionInfo API is always available |
| 2010-09-30 | 3.1.5 | AUTOSAR Administration | • New parameter added to APIs in or-der to chain CRC computations<br>• CRC check values corrected and checked values better explained<br>• CRC magic check added |
| 2010-02-02 | 3.1.4 | AUTOSAR Administration | • Introduction of a new CRC-8 with the polynomial 2Fh<br>• CRC-8 is now compliant to SAE J1850<br>• Legal disclaimer revised |
| 2008-08-13 | 3.1.1 | AUTOSAR Administration | • Separated CRC requirements from Memory Services Requirements<br>• CRC8 management added |

▽

△

| | | | |
|---|---|---|---|
| 2008-02-01 | 3.0.2 | AUTOSAR Administration | • Separated CRC requirements from Memory Services Requirements<br>• CRC8 management added |
| 2007-12-21 | 3.0.1 | AUTOSAR Administration | • Document meta information extended<br>• Small layout adaptations made |
| 2007-01-24 | 2.1.15 | AUTOSAR Administration | • "Advice for users" revised<br>• "Revision Information" added |
| 2006-11-28 | 2.1.1 | AUTOSAR Administration | • `Crc_CalculateCRC16` and `Crc_CalculateCRC32` APIs, `Crc_DataPtr` parameter : void pointer changed to uint8 pointer<br>• Legal disclaimer revised |
| 2006-05-16 | 2.0 | AUTOSAR Administration | • Document structure adapted to common Release 2.0 SWS Template<br>• UML model introduction<br>• Requirements traceability update<br>• Reentrancy at calculating CRC with hardware support |
| 2005-05-31 | 1.0 | AUTOSAR Administration | • Initial Release |

**Disclaimer**

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

# Table of Contents

1 Introduction and functional overview 8
2 Acronyms and Abbreviations 9
3 Related documentation 10
3.1 Input documents & related standards and norms . . . 10
3.2 Related specification . . . 10
4 Constraints and assumptions 11
4.1 Limitations . . . 11
4.2 Applicability to car domains . . . 11
5 Dependencies to other modules 12
6 Requirements Tracing 13
7 Functional specification 14
7.1 Basic Concepts of CRC Codes . . . 14
7.1.1 Mathematical Description . . . 14
7.1.2 Euclidian Algorithm for Binary Polynomials and Bit-Sequences . . . 16
7.1.3 CRC calculation, Variations and Parameter . . . 17
7.1.4 Encoding of CRC polynomials . . . 18
7.2 Standard parameters . . . 18
7.2.1 8-bit CRC calculation . . . 19
7.2.1.1 8-bit SAE J1850 CRC Calculation . . . 19
7.2.1.2 8-bit 0x2F polynomial CRC Calculation . . . 20
7.2.2 16-bit CRC calculation . . . 21
7.2.2.1 16-bit CCITT-FALSE CRC16 . . . 21
7.2.2.2 16-bit 0x8005 polynomial CRC calculation . . . 22
7.2.3 32-bit CRC calculation . . . 23
7.2.3.1 32-bit Ethernet CRC Calculation . . . 23
7.2.3.2 32-bit 0xF4ACFB13 polynomial CRC calculation . . . 23
7.2.3.3 32-bit SAE J1939-76 CRC Calculation . . . 24
7.2.4 64-bit CRC calculation . . . 25
7.2.4.1 64-bit ECMA polynomial CRC calculation . . . 25
7.3 General behavior . . . 26
7.4 Debugging concept . . . 26
7.5 Error Classification . . . 27
7.5.1 Development Errors . . . 27
7.5.2 Runtime Errors . . . 27
7.5.3 Production Errors . . . 27

8 API specification 28
8.1 Imported types . . . 28
8.2 Type definitions . . . 28
8.3 Function definitions . . . 28
8.3.1 8-bit CRC Calculation . . . 32
8.3.1.1 8-bit SAE J1850 CRC Calculation . . . 32
8.3.1.2 8-bit 0x2F polynomial CRC Calculation . . . 33
8.3.2 16-bit CRC Calculation . . . 34
8.3.2.1 16-bit CCITT-FALSE CRC16 . . . 34
8.3.2.2 16-bit 0x8005 polynomial CRC calculation . . . 35
8.3.3 32-bit CRC Calculation . . . 36
8.3.3.1 32-bit Ethernet CRC Calculation . . . 36
8.3.3.2 32-bit 0xF4ACFB13 polynomial CRC calculation . . . 37
8.3.3.3 32-bit SAE J1939-76 CRC Calculation . . . 38
8.3.4 64-bit CRC Calculation . . . 39
8.3.4.1 64-bit 0x42F0E1EBA9EA3693 polynomial CRC calculation . . . 39
8.4 Scheduled functions . . . 40
8.5 Expected interfaces . . . 40
8.5.1 Mandatory interfaces . . . 40
8.5.2 Optional interfaces . . . 40
8.5.3 Callout interfaces . . . 40
8.5.4 Configurable interfaces . . . 40
8.6 Service Interfaces . . . 40
9 Sequence diagrams 41
9.1 Crc_CalculateCRC8() . . . 41
9.2 Crc_CalculateCRC8H2F() . . . 41
9.3 Crc_CalculateCRC16() . . . 42
9.4 Crc_CalculateCRC16ARC() . . . 42
9.5 Crc_CalculateCRC32() . . . 43
9.6 Crc_CalculateCRC32P4() . . . 43
9.7 Crc_CalculateCRC32P76() . . . 44
9.8 Crc_CalculateCRC64() . . . 45
10 Configuration specification 46
10.1 How to read this chapter . . . 46
10.1.1 Configuration and configuration parameters . . . 46
10.1.2 Containers . . . 46
10.2 Containers and configuration parameters . . . 48
10.2.1 Crc . . . 48
10.3 Published Information . . . 52
A Not applicable requirements 53

B Change History 54
B.1 Change History of this document according to AUTOSAR Release R22-11 54
B.1.1 Added Specification Items in R22-11 . . . . . 54
B.1.2 Changed Specification Items in R22-11 . . . . . 54
B.1.3 Deleted Specification Items in R22-11 . . . . . 54
B.2 Change History of this document according to AUTOSAR Release R23-11 54
B.2.1 Added Specification Items in R23-11 . . . . . 54
B.2.2 Changed Specification Items in R23-11 . . . . . 54
B.2.3 Deleted Specification Items in R23-11 . . . . . 55
B.3 Change History of this document according to AUTOSAR Release R24-11 55
B.3.1 Added Specification Items in R24-11 . . . . . 55
B.3.2 Changed Specification Items in R24-11 . . . . . 55
B.3.3 Deleted Specification Items in R24-11 . . . . . 55
B.4 Change History of this document according to AUTOSAR Release R25-11 55
B.4.1 Added Specification Items in R25-11 . . . . . 55
B.4.2 Changed Specification Items in R25-11 . . . . . 55
B.4.3 Deleted Specification Items in R25-11 . . . . . 55
B.5 Change History of this document according to AUTOSAR Release R26-11 56
B.5.1 Added Specification Items in R26-11 . . . . . 56
B.5.2 Changed Specification Items in R26-11 . . . . . 56
B.5.3 Deleted Specification Items in R26-11 . . . . . 56

# 1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Library CRC.

The CRC library contains the following routines for CRC calculation:

- CRC8: SAEJ1850
- CRC8H2F: CRC8 0x2F polynomial
- CRC16
- CRC32
- CRC32P4: CRC32 0xF4ACFB13 polynomial
- CRC32P76: CRC32 0x6938392D polynomial
- CRC64: CRC-64-ECMA

For all routines (CRC8, CRC8H2F, CRC16, CRC32, CRC32P4 and CRC64), the following calculation methods are possible:

- Table based calculation: Fast execution, but larger code size (ROM table)
- Runtime calculation: Slower execution, but small code size (no ROM table)

All routines are re-entrant and can be used by multiple applications at the same time.

# 2 Acronyms and Abbreviations

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

| Abbreviation / Acronym: | Description: |
|---|---|
| ALU | Arithmetic Logic Unit |

**Table 2.1: Acronyms and abbreviations**

# 3 Related documentation

## 3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[4] ITU-T Recommendation X.25: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit
http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.25-199610-I!!PDF--E&type=items

[5] 32-bit cyclic redundancy codes for Internet applications

[6] Listing of CRCs, including CRC-64-ECMA
https://en.wikipedia.org/wiki/Cyclic_redundancy_check

[7] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[8] Specification of ECU Configuration
AUTOSAR_CP_TPS_ECUConfiguration

## 3.2 Related specification

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

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

# 4 Constraints and assumptions

## 4.1 Limitations

No known limitations.

## 4.2 Applicability to car domains

No restrictions.

# 5 Dependencies to other modules

There are no dependencies to other modules.

# 6 Requirements Tracing

The following tables reference the requirements specified in [3] 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 |
|---|---|---|
| No upstream requirements specified. | | |

**Table 6.1: Requirements Tracing**

# 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 respectively 32).

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

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

| $\oplus$ | $0b$ | $1b$ |
|---|---|---|
| $0b$ | $0b$ | $1b$ |
| $1b$ | $1b$ | $0b$ |

| $*$ | $0b$ | $1b$ |
|---|---|---|
| $0b$ | $0b$ | $0b$ |
| $1b$ | $0b$ | $1b$ |

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)* $p_1(X) = 1bX^3 + 0bX^2 + 1bX^1 + 0bX^0 = X^3 + X$

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

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.

$C(X) = d_{n-1}X^{k+n-1} + d_{n-2}X^{k+n-2} + \ldots + d_2X^{k+2} + d_1X^{k+1} + d_0X^k + r_{k-1}X^{k-1} + r_{k-2}X^{k-2} + \ldots r_1X + r_0$

$C(X) = X^k(d_{n-1}X^{n-1} + d_{n-2}X^{n-2} + \ldots + d_2X^2 + d_1X^1 + d_0) + r_{k-1}X^{k-1} + r_{k-2}X^{k-2} + \ldots + r_1X + r_0$

$C(X) = X^kD(X) \oplus R(X)$

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* $G(X) = X^k + g_{k-1}X^{k-1} + g_{k-2}X^{k-2} + \ldots + g_2X^2 + g_1X + g_0$. 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 $M(X) \rightarrow C(X)$ 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, $C(X) \bmod G(X)$. For a correct code word *C*, this remainder has to be *zero*, $C(X) \bmod G(X) = 0$. 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

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

Performing the operation $\bmod G(X)$ on both sides, one obtains

$$0 = C(X) \bmod G(X) = [X^kD(X)] \bmod G(X) \oplus R(X) \bmod G(X) \tag{7.1}$$

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 7.1 we obtain

$$R(X) = X^kD(X) \bmod G(X)$$

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, the propositions on CRC Codes are summarized as follows:

1. The construction principle of CRC Codes is based on polynomials with coefficients in the finite field of two elements. The $\oplus$ 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 $P_n(X) = p_n X^n + p_{n-1} X^{n-1} + \ldots + p_2 X^2 + p_1 X + p_0$ with coefficients in the finite field of two elements. Let $Q(X) = X^k + q_{k-1} X^{k-1} + q_{k-2} X^{k-2} + \ldots + q_2 X^2 + q_1 X + q_0$ 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 choose $R_n(X) = P_n(X)$ and $M_n = 0$.

(Recursion $n+1 \rightarrow n$)

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**

Choose $R_{n+1}(X) = P_{n+1}(X) \bmod Q(X)$ 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

$$R_{n+1}(X) = R_n(X) \tag{7.2}$$

$$M_{n+1}(X) = M_n(X) + p_{n+1}X^{n-k+1} \tag{7.3}$$

(end of proof)

**Example 7.1**

$P(X) = P^4(X) = X^4 + X^2 + X + 1b; Q(X) = X^2 + X + 1b; n = 4; k = 2$

$P_3(X) = X^4 + X^2 + X + 1b - 1b(X^2 + X + 1b)X^2 = X^3 + X + 1b$

$P_2(X) = X^3 + X + 1b - 1bX(X^2 + X + 1b) = X^2 + 1b$

$P_1(X) = X^2 + 1 - 1b(X^2 + X + 1) = X$

$R(X) = P(X) \bmod Q(X) = R_1(X) = P_1(X) = X$

### 7.1.3 CRC calculation, Variations and Parameter

Based on the Euclidian Algorithm, some variations have been developed in order to improve the calculation performance. All these variations do not improve the capability to detect or correct errors - the so-called Hamming Distance of the resulting code is determined only by the generator polynomial. Variations simply optimize for different implementing ALUs.

CRC-Calculation methods are characterized as follows:

1. Rule for Mapping of Data to a bit sequence $(d_{n-1}, d_{n-2}, d_{n-3}, \ldots, d_1, d_0)$ 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 following 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

### 7.1.4 Encoding of CRC polynomials

There are three notations for encoding the polynomial, so to clarify, all three notations are shown bellow as examples for the 42’F0’E1’EB’A9’EA’36’93h polynomial:

| | | |
|---|---|---|
| 1 | Polynomial as binary | 0001’0100’0010’1111’0000’1110’0001’1110’1011’1010’1001’ 1110’1010’0011’0110’1001’0011 |
| 2 | Normal representation with high bit | 01’42’F0’E1’EB’A9’EA’36’93h |
| 3 | Normal representation | 42’F0’E1’EB’A9’EA’36’93h |
| 4 | Reversed reciprocal representation (=Koopman representation) | A1’78’70’F5’D4’F5’1B’49h |

**Table 7.1: Encoding of CRC polynomials**

Notes:

1. Normal representation with high bit = hex representation of polynomial as binary
2. Normal representation with high bit = Koopman representation * 2 + 1

Within this document (and consistently unless otherwise noted) in AUTOSAR CP the *Normal representation* (without high bit), i.e., notation 3 in the above table, is used.

## 7.2 Standard parameters

This section gives a rough overview on the standard parameters that are commonly used for 8-bit, 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 (see definition below).
- Result data reflected: Similar to "Input data reflected" this parameter defines whether the bits of the CRC result are reflected (see definition below). The result is reflected over 8-bit for a CRC8, over 16-bit for a CRC16 and over 32-bit for a CRC32.
- 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 values ’1’ ’2’ ’3’ ’4’ ’5’ ’6’ ’7’ ’8’ ’9’ corresponding to values 31h 32h 33h 34h 35h 36h 37h 38h 39h is fed through the specified algorithm.

- Magic check: The CRC checking process calculates the CRC over the entire data block, including the CRC result. An error-free data block will always result in the unique constant polynomial (magic check) - representing the CRC-result XORed with 'XOR value'- regardless of the data block content.

**Example 7.2**

Magic check calculation of SAE-J1850 CRC8 (see detailed parameters in [SWS_Crc_00030]) over data bytes 00h 00h 00h 00h:

- CRC generation: CRC over 00h 00h 00h 00h, start value FFh:
  - CRC-result = 59h
- CRC check: CRC over 00h 00h 00h 00h 59h, start value FFh:
  - CRC-result = 3Bh
  - Magic check = CRC-result XORed with 'XOR value': C4h = 3Bh xor FFh

*Data reflection*: It is a reflection on a bit basis where data bits are written in the reverse order. The formula is:

$$\text{reflect}_n(x) = \sum_{i=0}^{n-1} x_i \times 2^{n-i-1}$$

where $x$ is the data and $n$ the number of data bits.

E.g. The reflection$_{16}$ of 2D$_{16}$ (n = 8) (00101101$_2$) is B4$_{16}$ (10110100$_2$)

The reflection$_{16}$ of 12345678$_{16}$ (n = 16) (0001 0010 0011 0100 0101 0110 0111 1000$_2$) is 1E6A2C48$_{16}$ (0001 1110 0110 1010 0010 1100 0100 1000$_2$).

The reflection$_{32}$ of 123456789ABCDEF0 (n = 32) (0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000$_2$) is 0F7B3D591E6A2C48$_{16}$ (0000 1111 0111 1011 0011 1101 0101 1001 0001 1110 0110 1010 0010 1100 0100 1000$_2$).

The reflection$_8$ of 123456789ABCDEF0 (n = 8) (0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000$_2$) is 84C2A6E195D3B7F0$_{16}$ (1000 0100 1100 0010 1010 0110 1110 0001 1001 0101 1101 0011 1011 0111 1111 0000$_2$).

### 7.2.1 8-bit CRC calculation

#### 7.2.1.1 8-bit SAE J1850 CRC Calculation

**[SWS_Crc_00030]** ⌈The `Crc_CalculateCRC8` function of the CRC module shall implement the CRC8 routine based on the SAE-J1850 CRC8 Standard, according to [SWS_Crc_00073].⌋

**[SWS_Crc_00073] SAE-J1850 CRC8 Polynomial** ⌈

| | |
|---|---|
| CRC result width: | 8 bits |
| Polynomial: | 1Dh |
| Initial value: | FFh |
| Input data reflected: | No |
| Result data reflected: | No |
| XOR value: | FFh |
| Check: | 4Bh |
| Magic check: | C4h |

⌋

**[SWS_Crc_00052]** ⌈The `Crc_CalculateCRC8` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00074].⌋

**[SWS_Crc_00074] Crc_CalculateCRC8 results** ⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | 59 |
| F2 | 01 | 83 | | | | | | | 37 |
| 0F | AA | 00 | 55 | | | | | | 79 |
| 00 | FF | 55 | 11 | | | | | | B8 |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | CB |
| 92 | 6B | 55 | | | | | | | 8C |
| FF | FF | FF | FF | | | | | | 74 |

⌋

### 7.2.1.2 8-bit 0x2F polynomial CRC Calculation

**[SWS_Crc_00042]** ⌈The `Crc_CalculateCRC8H2F` function of the CRC module shall implement the CRC8 routine based on the generator polynomial 0x2F, according to [SWS_Crc_00075].⌋

**[SWS_Crc_00075] CRC8 Polynomial** ⌈

| | |
|---|---|
| CRC result width: | 8 bits |
| Polynomial: | 2Fh |
| Initial value: | FFh |
| Input data reflected: | No |
| Result data reflected: | No |
| XOR value: | FFh |
| Check: | DFh |
| Magic check: | 42h |

⌋

**[SWS_Crc_00053]** ⌈The `Crc_CalculateCRC8H2F` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00076].⌋

**[SWS_Crc_00076] Crc_CalculateCRC8H2F results** ⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | 12 |
| F2 | 01 | 83 | | | | | | | C2 |
| 0F | AA | 00 | 55 | | | | | | C6 |
| 00 | FF | 55 | 11 | | | | | | 77 |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | 11 |
| 92 | 6B | 55 | | | | | | | 33 |
| FF | FF | FF | FF | | | | | | 6C |

⌋

### 7.2.2 16-bit CRC calculation

#### 7.2.2.1 16-bit CCITT-FALSE CRC16

**[SWS_Crc_00002]** ⌈The CRC module shall implement the CRC16 routine based on the CCITT-FALSE CRC16 Standard, according to [SWS_Crc_00077].⌋

Note concerning the standard document [4]:

The computed FCS is equal to CRC16 XOR FFFFh when the frame is built (first complement of the CCITT-FALSE CRC16).

For the verification, the CRC16 (CCITT-FALSE) is computed on the same data + FCS, and the resulting value is always 1D0Fh.

Note that, if during the verification, the check would have been done on data + CRC16 (i.e. FCS XOR FFFFh) the resulting value would have been 0000h that is the CCITT-FALSE magic check.

**[SWS_Crc_00077] CCITT-FALSE CRC16 Polynomial** ⌈

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

⌋

**[SWS_Crc_00054]** ⌈The `Crc_CalculateCRC16` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00078].⌋

**[SWS_Crc_00078] Crc_CalculateCRC16 results** ⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | 84C0 |
| F2 | 01 | 83 | | | | | | | D374 |
| 0F | AA | 00 | 55 | | | | | | 2023 |
| 00 | FF | 55 | 11 | | | | | | B8F9 |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | F53F |
| 92 | 6B | 55 | | | | | | | 0745 |
| FF | FF | FF | FF | | | | | | 1D0F |

⌋

#### 7.2.2.2 16-bit 0x8005 polynomial CRC calculation

**[SWS_Crc_00067]** ⌈The CRC module shall implement the CRC16 based on the CRC-16/ARC Standard, according to [SWS_Crc_00079].⌋

**[SWS_Crc_00079] CRC-16/ARC Polynomial** ⌈

| | |
|---|---|
| CRC result width: | 16 bits |
| Polynomial: | 8005h |
| Initial value: | 0000h |
| Input data reflected: | Yes |
| Result data reflected: | Yes |
| XOR value: | 0000h |
| Check: | BB3Dh |
| Magic check: | 0000h |

⌋

**[SWS_Crc_00068]** ⌈The `Crc_CalculateCRC16ARC` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00080].⌋

**[SWS_Crc_00080] Crc_CalculateCRC16ARC results** ⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | 0000 |
| F2 | 01 | 83 | | | | | | | C2E1 |
| 0F | AA | 00 | 55 | | | | | | 0BE3 |
| 00 | FF | 55 | 11 | | | | | | 6CCF |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | AE98 |
| 92 | 6B | 55 | | | | | | | E24E |
| FF | FF | FF | FF | | | | | | 9401 |

⌋

### 7.2.3 32-bit CRC calculation

#### 7.2.3.1 32-bit Ethernet CRC Calculation

**[SWS_Crc_00003]** ⌈The CRC module shall implement the CRC32 routine based on the IEEE-802.3 CRC32 Ethernet Standard, according to [SWS_Crc_00081].⌋

**[SWS_Crc_00081] IEEE-802.3 CRC32 Ethernet Polynomial** ⌈

| | |
|---|---|
| CRC result width: | 32 bits |
| Polynomial: | 04C11DB7h |
| Initial value: | FFFFFFFFh |
| Input data reflected: | Yes |
| Result data reflected: | Yes |
| XOR value: | FFFFFFFFh |
| Check: | CBF43926h |
| Magic check*: | DEBB20E3h |

⌋

***Important note:** To match the magic check value, the CRC must be appended in little endian format, i.e. low significant byte first. This is due to the reflections of the input and the result.

**[SWS_Crc_00055]** ⌈The `Crc_CalculateCRC32` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00082]⌋

**[SWS_Crc_00082] Crc_CalculateCRC32 results** ⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | 2144DF1C |
| F2 | 01 | 83 | | | | | | | 24AB9D77 |
| 0F | AA | 00 | 55 | | | | | | B6C9B287 |
| 00 | FF | 55 | 11 | | | | | | 32A06212 |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | B0AE863D |
| 92 | 6B | 55 | | | | | | | 9CDEA29B |
| FF | FF | FF | FF | | | | | | FFFFFFFF |

⌋

#### 7.2.3.2 32-bit 0xF4ACFB13 polynomial CRC calculation

This 32-bit CRC function is described in [5]. It has an advantage with respect to the Ethernet CRC - it has a Hamming Distance of 6 up to 4kB.

**[SWS_Crc_00056]** ⌈The CRC module shall implement the CRC32 routine using the 0xF4'AC'FB'13 polynomial, according to [SWS_Crc_00083].⌋

**[SWS_Crc_00083] 0xF4'AC'FB'13 Polynomial** ⌈

| | |
|---|---|
| CRC result width: | 32 bits |
| Polynomial: | F4'AC'FB'13h |
| Initial value: | FFFFFFFFh |
| Input data reflected: | Yes |
| Result data reflected: | Yes |
| XOR value: | FFFFFFFFh |
| Check: | 16'97'D0'6Ah |
| Magic check*: | 90'4C'DD'BFh |
| Hamming distance: | 6, up to 4096 bytes (including CRC) |

⌋

***Important note:** To match the magic check value, the CRC must be appended in little endian format, i.e. low significant byte first. This is due to the reflections of the input and the result.

**[SWS_Crc_00057]** ⌈The `Crc_CalculateCRC32P4` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00084].⌋

**[SWS_Crc_00084] Crc_CalculateCRC32P4 results** ⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | 6FB32240h |
| F2 | 01 | 83 | | | | | | | 4F721A25h |
| 0F | AA | 00 | 55 | | | | | | 20662DF8h |
| 00 | FF | 55 | 11 | | | | | | 9BD7996Eh |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | A65A343Dh |
| 92 | 6B | 55 | | | | | | | EE688A78h |
| FF | FF | FF | FF | | | | | | FFFFFFFFh |

⌋

### 7.2.3.3 32-bit SAE J1939-76 CRC Calculation

**[SWS_Crc_00087] SAE J1939-76 CRC Calculation**

*Status:* DRAFT

⌈The CRC module shall implement the CRC32 routine based on the SAE J1939-76 CRC32 Standard, according to [SWS_Crc_00088].⌋

**[SWS_Crc_00088] SAE J1939-76 CRC32 Polynomial**

*Status:* DRAFT

⌈

| | |
|---|---|
| CRC result width: | 32 bits |
| Polynomial: | 6938392Dh |
| Initial value: | FFFFFFFFh |
| Input data reflected: | No |
| Result data reflected: | No |
| XOR value: | 00000000h |
| Check: | 4DB36B68 |
| Magic check: | 00000000h |

⌋

**[SWS_Crc_00089] Crc_CalculateCRC32P76 results**

*Status:* DRAFT

⌈The `Crc_CalculateCRC32P76` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00090]⌋

**[SWS_Crc_00090] Crc_CalculateCRC32P76 results**

*Status:* DRAFT

⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | FFD18D4D |
| F2 | 01 | 83 | | | | | | | 22E39B5D |
| 0F | AA | 00 | 55 | | | | | | 1E7F978E |
| 00 | FF | 55 | 11 | | | | | | E3574865 |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | 09ADD524 |
| 92 | 6B | 55 | | | | | | | 93057FCD |
| FF | FF | FF | FF | | | | | | 00000000 |

⌋

### 7.2.4 64-bit CRC calculation

#### 7.2.4.1 64-bit ECMA polynomial CRC calculation

This 64-bit CRC function is described in [6]. It has a good hamming distance of 4, for long data (see below).

**[SWS_Crc_00062]** ⌈The CRC module shall implement the CRC64 routine using the 0x42’F0’E1’EB’A9’EA’36’93 polynomial, according to [SWS_Crc_00085].⌋

**[SWS_Crc_00085] 0x42'F0'E1'EB'A9'EA'36'93 Polynomial** ⌈

| | |
|---|---|
| CRC result width: | 64 bits |
| Polynomial: | 42'F0'E1'EB'A9'EA'36'93h |
| Initial value: | FFFFFFFFFFFFFFFFh |
| Input data reflected: | Yes |
| Result data reflected: | Yes |
| XOR value: | FFFFFFFFFFFFFFFFh |
| Check: | 99'5D'C9'BB'DF'19'39'FAh |
| Magic check*: | 49'95'8C'9A'BD'7D'35'3Fh |
| Hamming distance: | 4, up to almost 8 GB |

⌋

***Important note:** To match the magic check value, the CRC must be appended in little endian format, i.e. low significant byte first. This is due to the reflections of the input and the result.

**[SWS_Crc_00063]** ⌈The `Crc_CalculateCRC64` function of the CRC module shall provide the following CRC results, according to [SWS_Crc_00086].⌋

**[SWS_Crc_00086] Crc_CalculateCRC64 results** ⌈

| Data bytes (hexadecimal) | | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | | | | | | F4A586351E1B9F4Bh |
| F2 | 01 | 83 | | | | | | | 319C27668164F1C6h |
| 0F | AA | 00 | 55 | | | | | | 54C5D0F7667C1575h |
| 00 | FF | 55 | 11 | | | | | | A63822BE7E0704E6h |
| 33 | 22 | 55 | AA | BB | CC | DD | EE | FF | 701ECEB219A8E5D5h |
| 92 | 6B | 55 | | | | | | | 5FAA96A9B59F3E4Eh |
| FF | FF | FF | FF | | | | | | FFFFFFFF00000000h |

⌋

## 7.3 General behavior

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

## 7.4 Debugging concept

None

## 7.5 Error Classification

The document [2] Chapter 7.2 *"Error Handling"* 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.

### 7.5.1 Development Errors

There are no development errors.

### 7.5.2 Runtime Errors

There are no runtime errors.

### 7.5.3 Production Errors

There are no production errors.

# 8 API specification

## 8.1 Imported types

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

**[SWS_Crc_00018] Definition of imported datatypes of module Crc** ⌈

| *Module* | *Header File* | *Imported Type* |
|---|---|---|
| There are no imported types. | | |

⌋

## 8.2 Type definitions

None.

## 8.3 Function definitions

This chapter contains all APIs provided by the module. The substructure of this chapter shall be based on possible users of the API, the users are identified via the different provided headers. If only the <Mip>.h is provided then no further substructure is needed.

The caller of a Crc API is responsible to pass valid arguments. Crc functions do not perform checks on validity of input parameters.

**[SWS_Crc_00072]** ⌈The CRC library functions shall not call any BSW modules functions (e.g. the DET).⌋

**[SWS_Crc_00014]** ⌈The CRC function (with parameter *Crc_IsFirstCall* = TRUE) shall do the following operations:

1. As 'Initial value' of the CRC computation, uses the attribute 'Initial value' of the polynomial:
$$Crc = PolynomialInitVal$$
2. If the attribute 'Input data reflected' of the polynomial is TRUE, then reflects input data (byte per byte) obtained via parameters *Crc_DataPtr* and *Crc_Length*:
$$Data = \text{reflect}_8(Data) \text{ (in the case 'Input data reflected' is TRUE)}$$
3. Compute the CRC over the data, the last CRC and the CRC polynomial:
$$Crc = f(Data, Crc, Polynomial)$$
4. Execute the XOR operation between crc and 'XOR value' of the polynomial:
$$Crc = Crc \oplus PolynomialXORVal$$

5. If the attribute 'Result data reflected' of the polynomial is TRUE, then reflect the CRC (over 8, 16, 32 or 64 bits, depending on the CRC size):

$$Crc = reflect_{Crcsize}(Crc) \text{ (in the case 'Result data reflected' is TRUE)}$$

6. The CRC is returned:

$$return\ Crc$$

Steps 2 and 3 are performed as long as data are available⌋

**[SWS_Crc_00041]** ⌈The CRC function (with parameter *Crc_IsFirstCall* = FALSE) shall do the following operations:

1. As 'Initial value' of the CRC computation, uses the parameter Crc_StartValueX (where X is 8, 8H2F, 16, 16ARC, 32, P4 or 64) that should be the CRC result of the last call. The result is then XORed with 'XOR value' and reflected if 'Result data reflected' of the polynomial is TRUE:

$$Crc = Crc_StartValueX \oplus PolynomialXORVal$$

$$Crc = \text{reflect}_{Crcsize}(Crc) \text{ (in the case 'Result data reflected' is TRUE)}$$

Steps 2 to 6 are identical to [SWS_Crc_00014].⌋

Usage of CRC functions:

For the first or the unique call the user of a CRC function shall:

1. give a pointer to the data (*Crc_DataPtr*)
2. give the number of bytes of data (*Crc_Length*)
3. give the *Crc_StartValueX* parameter a don't care value (the initialization value is known by the chosen algorithm)
4. give the *Crc_IsFirstCall* parameter the value TRUE to inform the library that it is the first or unique call
5. call the CRC function
6. get the CRC

For the subsequent calls the user has to:

1. give a pointer to the data (*Crc_DataPtr*)
2. give the number of bytes of data (*Crc_Length*)
3. give the *Crc_StartValueX* parameter (X is 8, 8H2F, 16, 16ARC, 32, P4 or 64) the *CRC result of the previous call*
4. give the *Crc_IsFirstCall* parameter the value *FALSE* to inform the library that it is not the first call

5. call the CRC function
6. get the CRC

**Example 8.1**

Calculation of CRC8: calculation of CRC8 SAEJ1850, over one of test patterns defined by SAE J1850 specification (00h, FFh, 55h, 11h results with CRC B8h)

- If done in one step:

```
uint8 Array[4] = {0x00, 0xFF, 0x55, 0x11};
uint8 ignored_val = 0x00l; /* any value, it is ignored */

uint8 resultSAE = Crc_CalculateCRC8(&Array[0], 4, ignored_val, TRUE);
```

**Listing 8.1: Single-step CRC8 SAEJ1850 calculation**

resultSAE shall be equal to B8h

- If done in several steps:

```
uint8 Array[4] = {0x00, 0xFF, 0x55, 0x11};
uint8 ignored_val = 0x00l; /* any value, it is ignored */

uint8 resultSAE = Crc_CalculateCRC8(&Array[0], 2, ignored_val, TRUE);
resultSAE = Crc_CalculateCRC8(&Array[2], 1, resultSAE, FALSE);
resultSAE = Crc_CalculateCRC8(&Array[3], 1, resultSAE, FALSE);
```

**Listing 8.2: Multi-step CRC8 SAEJ1850 calculation**

resultSAE shall be also equal to B8h

**Example 8.2**

Calculation of CRC8: calculation of that is not compatible with SAE J1850, but it is compatible with AUTOSAR releases before R4.0:

- If done in one step:

```
uint8 Array[4] = {0x00, 0xFF, 0x55, 0x11};

/* The first call also gets IsFirstCall set to FALSE, and 0xFF as start
    value, which is immediately XORed with 0xFF by CalculateCRC8,
   resulting with start value equal to 0x00. */

uint8 resultRel3 = Crc_CalculateCRC8(&Array[0], 4, 0xFF, FALSE);

/* The last XORing must be negated by the caller, to come to 0x00 XOR
   value. */

resultRel3 = resultR3 ^ 0xFF;
```

**Listing 8.3: Single-step AUTOSAR R3.2 CRC8 calculation**

resultRel3 contains the same value as computed by AUTOSAR R3.2 CRC8.

- If done in several steps:

```
uint8 Array[4] = {0x00, 0xFF, 0x55, 0x11};
```

```
/* The first call also gets IsFirstCall set to FALSE, and 0xFF as start
     value, which is immediately XORed with 0xFF by CalculateCRC8,
    resulting with start value equal to 0x00. */

uint8 resultRel3 = Crc_CalculateCRC8(&Array[0], 2, 0xFF, FALSE);
resultRel3 = Crc_CalculateCRC8(&Array[2], 1, resultRel3,FALSE);
resultRel3 = Crc_CalculateCRC8(&Array[3], 1, resultRel3, FALSE);

/* The last XORing must be negated by the caller, to come to 0x00 XOR
    value. */

resultRel3 = resultR3 ^ 0xFF; }
```

**Listing 8.4: Multi-step AUTOSAR R3.2 CRC8 calculation**

resultRel3 contains also the same value as computed by AUTOSAR R3.2 CRC8.

**Example 8.3**

Calculation of CRC32 Ethernet Standard (see detailed parameters in [SWS_Crc_00003]) over data bytes 01h 02h 03h 04h 05h 06h 07h 08h:

- In one function call, CRC over 01h 02h 03h 04h 05h 06h 07h 08h, start value FFFFFFFFh:
  - CRC-result = **3FCA88C5h** (final value)
- In two function calls:
  - CRC over 01h 02h 03h 04h, start value FFFFFFFFh:
    - CRC-result of first call = B63CFBCDh (intermediate value)
  - CRC over 05h 06h 07h 08h, start value: B63CFBCDh xor *XOR value* (FFFFFFFFh) = 49C30432h and after reflection: 4C20C392h
    - CRC-result of final call = **3FCA88C5h** (final value)

The following C-code example shows that the caller modifies the start value by using the previous result (without any rework) and indicates that it is no more the first call:

```
InterResult = Crc_CalculateCRC32(&Array12345678[0], 4, 0xFFFFFFFF, TRUE
    );
result = Crc_CalculateCRC32(&Array12345678[4], 4, InterResult, FALSE);
```

**Listing 8.5: CRC calculation using the previous result**

### 8.3.1 8-bit CRC Calculation

#### 8.3.1.1 8-bit SAE J1850 CRC Calculation

**[SWS_Crc_00031] Definition of API function Crc_CalculateCRC8** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC8 | |
| ***Syntax*** | `uint8 Crc_CalculateCRC8 ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint8 Crc_StartValue8, boolean Crc_IsFirstCall )` | |
| ***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_StartValue8 | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue8. FALSE: Subsequent call in a call sequence; Crc_StartValue8 is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint8 | 8 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC8 calculation on Crc_Length data bytes, with SAE J1850 parameters | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00032]** ⌈The function `Crc_CalculateCRC8` shall perform a CRC8 calculation using polynomial 0x1D on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue8`.⌋

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC8` in order to decrease the calculation time.

The function `Crc_CalculateCRC8` requires specification of configuration parameters defined in `Crc8Mode`.

#### 8.3.1.2 8-bit 0x2F polynomial CRC Calculation

**[SWS_Crc_00043] Definition of API function Crc_CalculateCRC8H2F** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC8H2F | |
| ***Syntax*** | `uint8 Crc_CalculateCRC8H2F ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint8 Crc_StartValue8H2F, boolean Crc_IsFirstCall )` | |
| ***Service ID [hex]*** | 0x05 | |
| ***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_StartValue8H2F | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue8H2F. FALSE: Subsequent call in a call sequence; Crc_StartValue8H2F is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint8 | 8 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC8 calculation with the Polynomial 0x2F on Crc_Length | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00044]** ⌈The function `Crc_CalculateCRC8H2F` shall perform a CRC8 calculation with the polynomial 0x2F on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue8H2F`.⌋

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC8H2F` in order to decrease the calculation time.

The function `Crc_CalculateCRC8H2F` requires specification of configuration parameters defined `Crc8H2FMode`.

### 8.3.2 16-bit CRC Calculation

#### 8.3.2.1 16-bit CCITT-FALSE CRC16

**[SWS_Crc_00019] Definition of API function Crc_CalculateCRC16** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC16 | |
| ***Syntax*** | `uint16 Crc_CalculateCRC16 ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint16 Crc_StartValue16, boolean Crc_IsFirstCall )` | |
| ***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_StartValue16 | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue16. FALSE: Subsequent call in a call sequence; Crc_StartValue16 is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint16 | 16 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC16 calculation on Crc_Length data bytes. | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00015]** ⌈The function `Crc_CalculateCRC16` shall perform a CRC16 calculation using polynomial 0x1021 on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue16`.⌋

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC16` in order to decrease the calculation time.

The function `Crc_CalculateCRC16` requires specification of configuration parameters defined in `Crc16Mode`.

#### 8.3.2.2 16-bit 0x8005 polynomial CRC calculation

**[SWS_Crc_00071] Definition of API function Crc_CalculateCRC16ARC** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC16ARC | |
| ***Syntax*** | `uint16 Crc_CalculateCRC16ARC ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint16 Crc_StartValue16, boolean Crc_IsFirstCall )` | |
| ***Service ID [hex]*** | 0x08 | |
| ***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 | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue16. FALSE: Subsequent call in a call sequence; Crc_StartValue16 is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint16 | 16 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC16 calculation on Crc_Length data bytes, using the polynomial 0x8005. | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00069]** ⌈The function `Crc_CalculateCRC16ARC` shall perform a CRC16 calculation using polynomial 0x8005 on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue16`.⌋

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC16ARC` in order to decrease the calculation time.

The function `Crc_CalculateCRC16ARC` requires specification of configuration parameters defined in `Crc16ARCMode`.

### 8.3.3 32-bit CRC Calculation

#### 8.3.3.1 32-bit Ethernet CRC Calculation

**[SWS_Crc_00020] Definition of API function Crc_CalculateCRC32** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC32 | |
| ***Syntax*** | `uint32 Crc_CalculateCRC32 ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint32 Crc_StartValue32, boolean Crc_IsFirstCall )` | |
| ***Service ID [hex]*** | 0x03 | |
| ***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 | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue32. FALSE: Subsequent call in a call sequence; Crc_StartValue32 is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint32 | 32 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC32 calculation on Crc_Length data bytes. | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00016]** ⌈The function `Crc_CalculateCRC32` shall perform a CRC32 calculation using polynomial 0x04C11DB7 on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue32`.⌋

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC32` in order to decrease the calculation time.

The function `Crc_CalculateCRC32` requires specification of configuration parameters defined in `Crc32Mode`.

#### 8.3.3.2 32-bit 0xF4ACFB13 polynomial CRC calculation

**[SWS_Crc_00058] Definition of API function Crc_CalculateCRC32P4** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC32P4 | |
| ***Syntax*** | `uint32 Crc_CalculateCRC32P4 ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint32 Crc_StartValue32, boolean Crc_IsFirstCall )` | |
| ***Service ID [hex]*** | 0x06 | |
| ***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 | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue32. FALSE: Subsequent call in a call sequence; Crc_StartValue32 is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint32 | 32 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC32 calculation on Crc_Length data bytes, using the polynomial 0xF4 ACFB13.<br>This CRC routine is used by E2E Profile 4. | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00059]** ⌈The function `Crc_CalculateCRC32P4` shall perform a CRC32 calculation using polynomial 0xF4ACFB13 on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue32`.⌋

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method, should be configured for the function `Crc_CalculateCRC32P4` in order to decrease the calculation time.

The function `Crc_CalculateCRC32P4` requires specification of configuration parameters defined in `Crc32P4Mode`.

#### 8.3.3.3 32-bit SAE J1939-76 CRC Calculation

**[SWS_Crc_00091] Definition of API function Crc_CalculateCRC32P76**

*Status:* DRAFT

⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC32P76 (draft) | |
| ***Syntax*** | `uint32 Crc_CalculateCRC32P76 ( const uint8* Crc_DataPtr, uint32 Crc_Length, uint32 Crc_StartValue32, boolean Crc_IsFirstCall )` | |
| ***Service ID [hex]*** | 0x09 | |
| ***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 | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue32. FALSE: Subsequent call in a call sequence; Crc_StartValue32 is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint32 | 32 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC32 calculation on Crc_Length data bytes, using the polynomial 0x0x6938392D.<br>**Tags:** atp.Status=draft | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00092] CRC32 calculation using polynomial 0x6938392D**

*Status:* DRAFT

⌈The function `Crc_CalculateCRC32P76` shall perform a CRC32 calculation using polynomial 0x6938392D on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue32`.⌋

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC32P76` in order to decrease the calculation time.

The function `Crc_CalculateCRC32P76` requires specification of configuration parameters defined in `Crc32P76Mode`.

### 8.3.4 64-bit CRC Calculation

#### 8.3.4.1 64-bit 0x42F0E1EBA9EA3693 polynomial CRC calculation

**[SWS_Crc_00061] Definition of API function Crc_CalculateCRC64** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Crc_CalculateCRC64 | |
| ***Syntax*** | `uint64 Crc_CalculateCRC64 (`<br>`  const uint8* Crc_DataPtr,`<br>`  uint32 Crc_Length,`<br>`  uint64 Crc_StartValue64,`<br>`  boolean Crc_IsFirstCall`<br>`)` | |
| ***Service ID [hex]*** | 0x07 | |
| ***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_StartValue64 | Start value when the algorithm starts. |
| | Crc_IsFirstCall | TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue64. FALSE: Subsequent call in a call sequence; Crc_StartValue64 is interpreted to be the return value of the previous function call. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint64 | 64 bit result of CRC calculation. |
| ***Description*** | This service makes a CRC64 calculation on Crc_Length data bytes, using the polynomial 0x42F0E1EBA9EA3693.<br>This CRC routine is used by E2E Profile 7. | |
| ***Available via*** | Crc.h | |

⌋

**[SWS_Crc_00064]** ⌈The function `Crc_CalculateCRC64` shall perform a CRC64 calculation using polynomial 0x42F0E1EBA9EA3693 on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue64`.⌋

Note: If large data blocks have to be calculated (>64 bytes, depending on performance of processor platform), the table based calculation method, should be configured for the function `Crc_CalculateCRC64` in order to decrease the calculation time.

The function `Crc_CalculateCRC64` requires specification of configuration parameters defined in `Crc64Mode`.

## 8.4 Scheduled functions

The Crc module does not have scheduled functions.

## 8.5 Expected interfaces

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

### 8.5.1 Mandatory interfaces

None

### 8.5.2 Optional interfaces

None.

### 8.5.3 Callout interfaces

None.

### 8.5.4 Configurable interfaces

None.

## 8.6 Service Interfaces

None.

# 9 Sequence diagrams

## 9.1 Crc_CalculateCRC8()

The following diagram shows the synchronous function call `Crc_CalculateCRC8`.

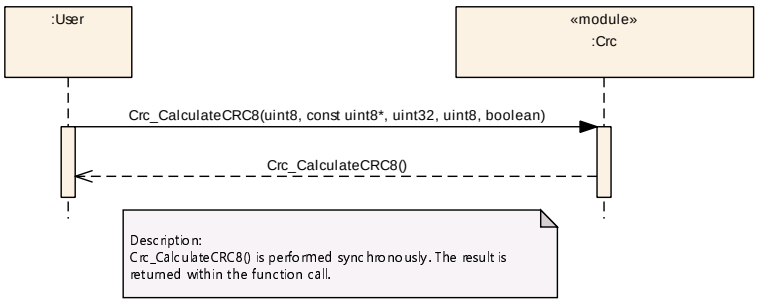



**Figure 9.1: Crc_CalculateCRC8**

## 9.2 Crc_CalculateCRC8H2F()

The following diagram shows the synchronous function call `Crc_CalculateCRC8H2F`.

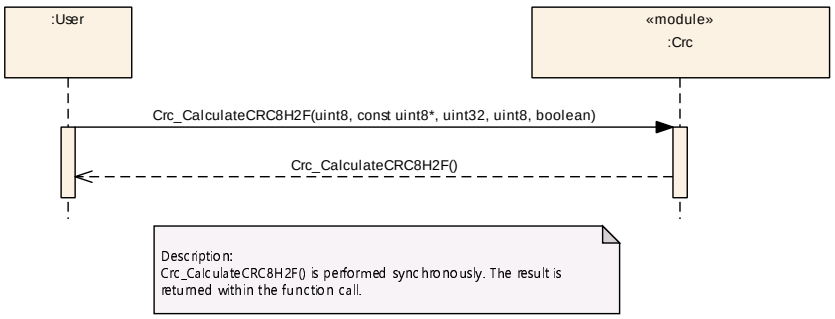



**Figure 9.2: Crc_CalculateCRC8H2F**

## 9.3 Crc_CalculateCRC16()

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

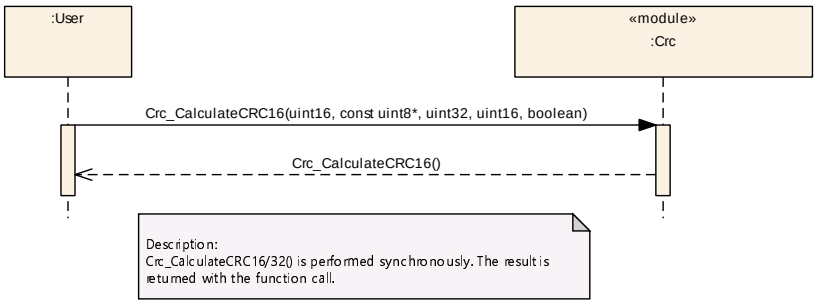



**Figure 9.3: Crc_CalculateCRC16**

## 9.4 Crc_CalculateCRC16ARC()

The following diagram shows the synchronous function call `Crc_CalculateCRC16ARC`.

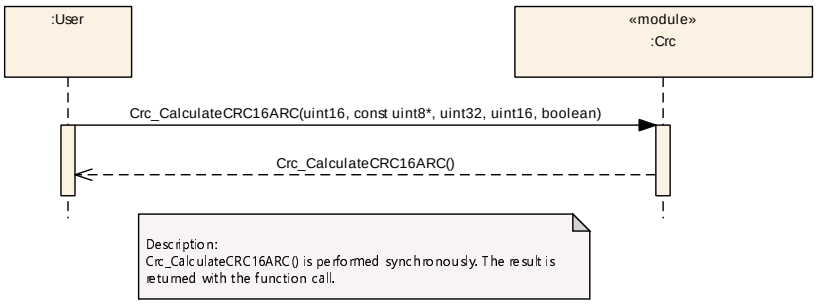



**Figure 9.4: Crc_CalculateCRC16ARC**

## 9.5 Crc_CalculateCRC32()

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

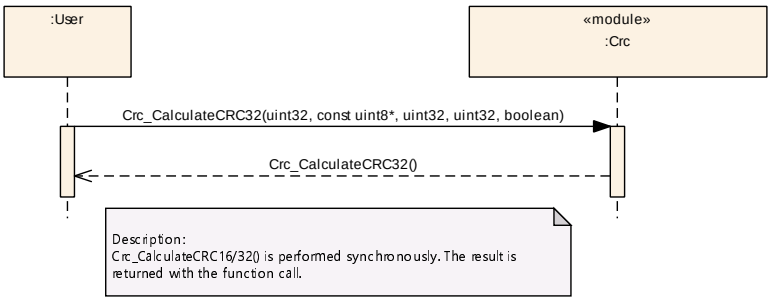



**Figure 9.5: Crc_CalculateCRC32**

## 9.6 Crc_CalculateCRC32P4()

The following diagram shows the synchronous function call `Crc_CalculateCRC32P4`.

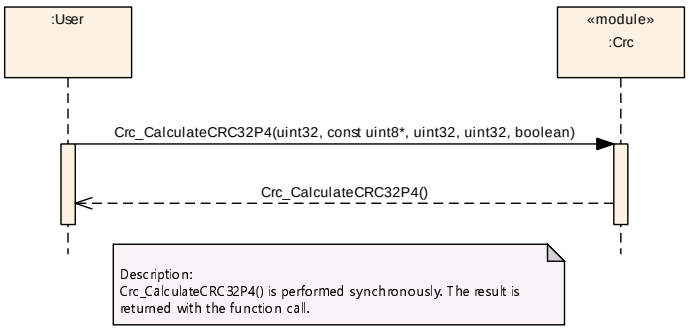



**Figure 9.6: Crc_CalculateCRC32P4**

## 9.7 Crc_CalculateCRC32P76()

The following diagram shows the synchronous function call `Crc_CalculateCRC32P76`.

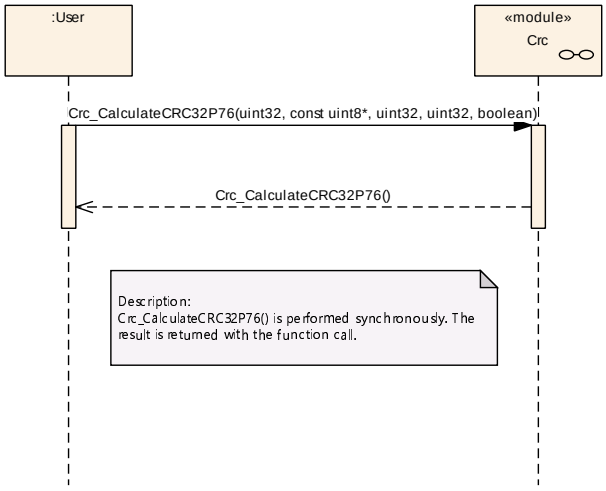



**Figure 9.7: Crc_CalculateCRC32P76**

## 9.8 Crc_CalculateCRC64()

The following diagram shows the synchronous function call `Crc_CalculateCRC64`.

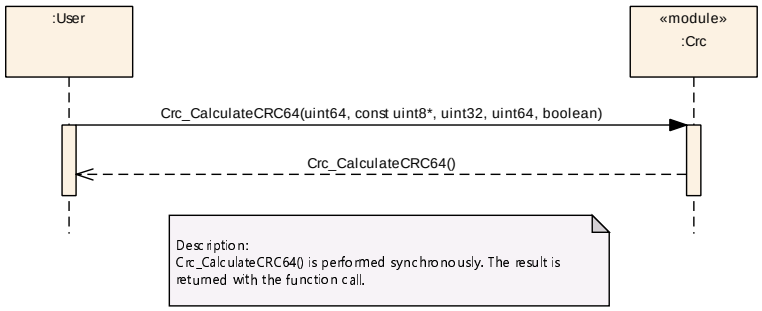



**Figure 9.8: Crc_CalculateCRC64**

# 10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module CRC.

Chapter 10.3 specifies published information of the module CRC.

## 10.1 How to read this chapter

For details refer to [2] Chapter 10.1 *"Introduction to configuration specification"*.

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

- AUTOSAR Layered Architecture [7]
- AUTOSAR ECU Configuration Specification [8]:
  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 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 are described in chapters 7 and 8.

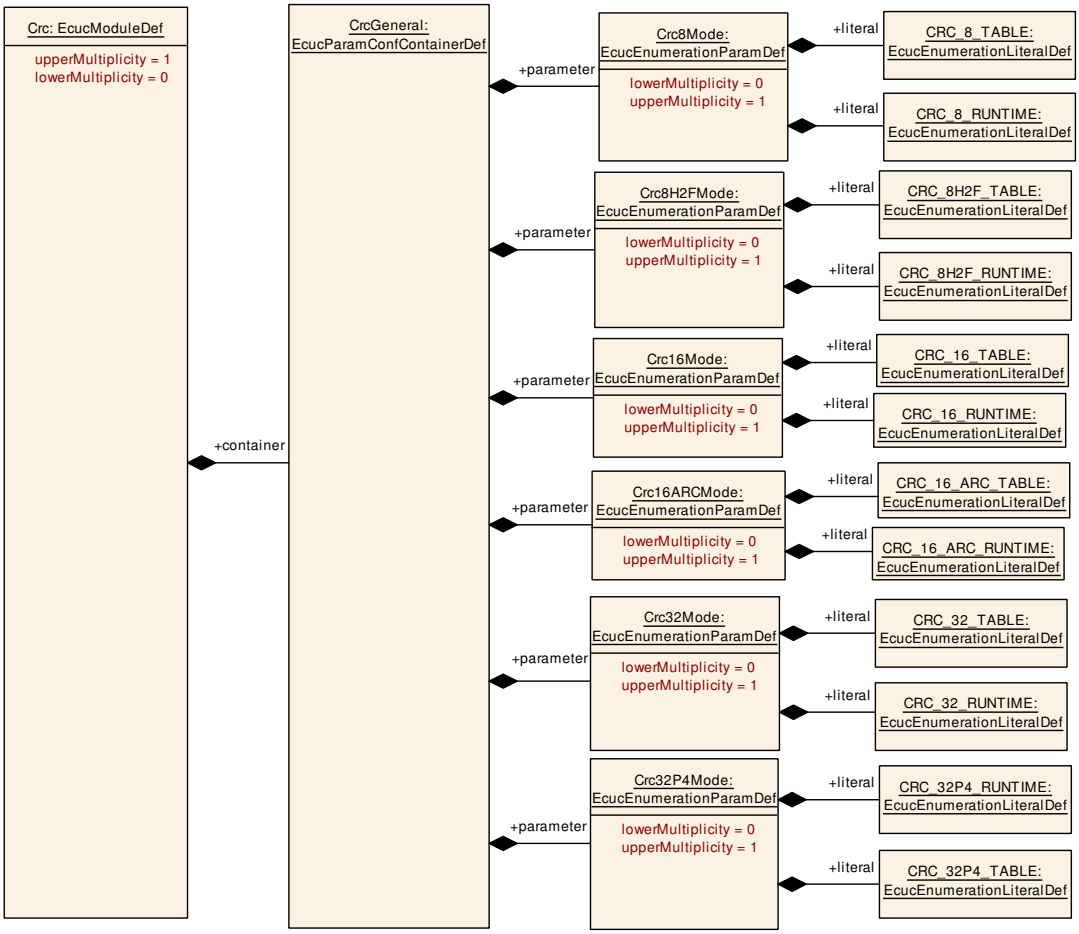



**Figure 10.1: CRC**

### 10.2.1 Crc

### [ECUC_Crc_00033] Definition of EcucModuleDef Crc ⌈

| | |
|---|---|
| **Module Name** | Crc |
| **Description** | Configuration of the Crc (Crc routines) module. |
| **Post-Build Selectable Support** | false |
| **Post-Build Loadable Support** | false |

| **Included Containers** | | |
|---|---|---|
| **Container Name** | **Multiplicity** | **Description** |
| CrcGeneral | 1 | General configuration of CRC module |

⌋

## [ECUC_Crc_00006] Definition of EcucParamConfContainerDef CrcGeneral ⌈

| | |
|---|---|
| **Container Name** | CrcGeneral |
| **Parent Container** | Crc |
| **Upstream mapping defined** | – |
| **Description** | General configuration of CRC module |
| **Multiplicity** | 1 |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| **Included Parameters** | | |
|---|---|---|
| **Parameter Name** | **Multiplicity** | **ECUC ID** |
| Crc16ARCMode | 0..1 | [ECUC_Crc_00035] |
| Crc16Mode | 0..1 | [ECUC_Crc_00025] |
| Crc32Mode | 0..1 | [ECUC_Crc_00026] |
| Crc32P4Mode | 0..1 | [ECUC_Crc_00032] |
| Crc32P76Mode | 0..1 | [ECUC_Crc_00036] |
| Crc64Mode | 0..1 | [ECUC_Crc_00034] |
| Crc8H2FMode | 0..1 | [ECUC_Crc_00031] |
| Crc8Mode | 0..1 | [ECUC_Crc_00030] |

| **No Included Containers** |
|---|

⌋

## [ECUC_Crc_00035] Definition of EcucEnumerationParamDef Crc16ARCMode ⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc16ARCMode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC-16/ARC (polynomial 8005) calculation methods | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_16_ARC_RUNTIME | runtime based CRC16 calculation |
| | CRC_16_ARC_TABLE | table based CRC16 calculation (default selection) |
| **Pre-Compile only** | true | |

⌋

## [ECUC_Crc_00025] Definition of EcucEnumerationParamDef Crc16Mode ⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc16Mode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC 16-bit (CCITT) calculation methods | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_16_RUNTIME | runtime based CRC16 calculation |

▽

△

| | | |
|---|---|---|
| | CRC_16_TABLE | table based CRC16 calculation (default selection) |
| **Pre-Compile only** | true | |

⌋

### [ECUC_Crc_00026] Definition of EcucEnumerationParamDef Crc32Mode ⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc32Mode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC 32-bit (IEEE-802.3 CRC32 Ethernet Standard) calculation methods | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_32_RUNTIME | runtime based CRC32 calculation |
| | CRC_32_TABLE | table based CRC32 calculation (default selection) |
| **Pre-Compile only** | true | |

⌋

### [ECUC_Crc_00032] Definition of EcucEnumerationParamDef Crc32P4Mode ⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc32P4Mode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC 32-bit E2E Profile 4 calculation methods. | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_32P4_RUNTIME | runtime based CRC32P4 calculation |
| | CRC_32P4_TABLE | table based CRC32P4 calculation (default selection) |
| **Pre-Compile only** | true | |

⌋

### [ECUC_Crc_00036] Definition of EcucEnumerationParamDef Crc32P76Mode

*Status:* DRAFT

⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc32P76Mode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC 32-bit SAE J1939-76 calculation methods.<br>**Tags:** atp.Status=draft | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_32P76_RUNTIME | runtime based CRC32P76 calculation<br>**Tags:** atp.Status=draft |

▽

△

| | | |
|---|---|---|
| | CRC_32P76_TABLE | table based CRC32P76 calculation<br>**Tags:** atp.Status=draft |
| **Default value** | CRC_32P76_TABLE | |
| **Pre-Compile only** | true | |

⌋

### [ECUC_Crc_00034] Definition of EcucEnumerationParamDef Crc64Mode ⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc64Mode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC 64-bit calculation methods. | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_64_RUNTIME | runtime based CRC64 calculation |
| | CRC_64_TABLE | table based CRC64 calculation (default selection) |
| **Pre-Compile only** | true | |

⌋

### [ECUC_Crc_00031] Definition of EcucEnumerationParamDef Crc8H2FMode ⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc8H2FMode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC 8-bit (2Fh polynomial) calculation methods | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_8H2F_RUNTIME | runtime based CRC8H2F calculation |
| | CRC_8H2F_TABLE | table based CRC8H2F calculation (default selection) |
| **Pre-Compile only** | true | |

⌋

### [ECUC_Crc_00030] Definition of EcucEnumerationParamDef Crc8Mode ⌈

| | | |
|---|---|---|
| **Parameter Name** | Crc8Mode | |
| **Parent Container** | CrcGeneral | |
| **Upstream mapping defined** | – | |
| **Description** | Switch to select one of the available CRC 8-bit (SAE J1850) calculation methods | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range** | CRC_8_RUNTIME | runtime based CRC8 calculation |
| | CRC_8_TABLE | table based CRC8 calculation (default selection) |
| **Pre-Compile only** | true | |

⌋

## 10.3 Published Information

For details refer to [2] Chapter 10.3 *"Published Information"*.

**[SWS_Crc_00050]** ⌈The standardized common published parameters as required by

SRS_BSW_00402 in the SRS General on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW ModuleDescription. The according module abbreviation can be found in the SWS General on Basic Software Modules [2].⌋

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

**[SWS_Crc_00048]** ⌈

| 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/ uint16 | Module ID of this module from Module List |
| CRC_AR_RELEASE_MAJOR_VERSION | #define/ uint8 | Major version number of AUTOSAR release on which the appropriate implementation is based on. |
| CRC_AR_RELEASE_MINOR_VERSION | #define/ uint8 | Minor version number of AUTOSAR release on which the appropriate implementation is based on. |
| CRC_AR_RELEASE_REVISION_VERSION | #define/ uint8 | Patch level version number of AUTOSAR release 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. |

⌋

# A Not applicable requirements

**[SWS_Crc_NA_00051]**

*Upstream requirements:* SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00170, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00395, 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_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00336, SRS_BSW_00337, SRS_BSW_00369, SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00323, SRS_BSW_00409, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00004, SRS_BSW_00461, SRS_BSW_00469, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00478, SRS_BSW_00490, SRS_BSW_00491

⌈These requirements are not applicable to this specification.⌋

# B Change History

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

## B.1 Change History of this document according to AUTOSAR Release R22-11

### B.1.1 Added Specification Items in R22-11

[SWS_Crc_NA_00051]

### B.1.2 Changed Specification Items in R22-11

[SWS_Crc_00002] [SWS_Crc_00003] [SWS_Crc_00018] [SWS_Crc_00019] [SWS_Crc_00020] [SWS_Crc_00021] [SWS_Crc_00030] [SWS_Crc_00031] [SWS_Crc_00042] [SWS_Crc_00043] [SWS_Crc_00052] [SWS_Crc_00053] [SWS_Crc_00054] [SWS_Crc_00055] [SWS_Crc_00056] [SWS_Crc_00057] [SWS_Crc_00058] [SWS_Crc_00061] [SWS_Crc_00062] [SWS_Crc_00063] [SWS_Crc_00067] [SWS_Crc_00068] [SWS_Crc_00071]

### B.1.3 Deleted Specification Items in R22-11

[SWS_Crc_00051]

## B.2 Change History of this document according to AUTOSAR Release R23-11

### B.2.1 Added Specification Items in R23-11

[SWS_Crc_00073] [SWS_Crc_00074] [SWS_Crc_00075] [SWS_Crc_00076] [SWS_Crc_00077] [SWS_Crc_00078] [SWS_Crc_00079] [SWS_Crc_00080] [SWS_Crc_00081] [SWS_Crc_00082] [SWS_Crc_00083] [SWS_Crc_00084] [SWS_Crc_00085] [SWS_Crc_00086]

### B.2.2 Changed Specification Items in R23-11

none

### B.2.3 Deleted Specification Items in R23-11

[SWS_Crc_00009] [SWS_Crc_00010] [SWS_Crc_00033] [SWS_Crc_00045] [SWS_Crc_00060] [SWS_Crc_00065] [SWS_Crc_00070]

## B.3 Change History of this document according to AUTOSAR Release R24-11

### B.3.1 Added Specification Items in R24-11

[ECUC_Crc_00036] [SWS_Crc_00087] [SWS_Crc_00088] [SWS_Crc_00089] [SWS_Crc_00090] [SWS_Crc_00091] [SWS_Crc_00092]

### B.3.2 Changed Specification Items in R24-11

[ECUC_Crc_00006] [SWS_Crc_00050]

### B.3.3 Deleted Specification Items in R24-11

none

## B.4 Change History of this document according to AUTOSAR Release R25-11

### B.4.1 Added Specification Items in R25-11

none

### B.4.2 Changed Specification Items in R25-11

[ECUC_Crc_00025] [ECUC_Crc_00026] [ECUC_Crc_00030] [ECUC_Crc_00031] [ECUC_Crc_00032] [ECUC_Crc_00034] [ECUC_Crc_00035] [ECUC_Crc_00036]

### B.4.3 Deleted Specification Items in R25-11

[SWS_Crc_00013]

## B.5 Change History of this document according to AUTOSAR Release R26-11

### B.5.1 Added Specification Items in R26-11

none

### B.5.2 Changed Specification Items in R26-11

[SWS_Crc_00018]

### B.5.3 Deleted Specification Items in R26-11

[SWS_Crc_00011] [SWS_Crc_00017] [SWS_Crc_00021]