MICROSAR CRC

Technical Reference

Version 7.00.00

Authors	virgmi, virljs, virbka, speter
Status	Released

Document Information

History

Author	Date	Version	Remarks
-	2006-12-13	1.0	Initial Version
-	2008-01-21	3.00.00	Update to ASR 2.1 Changed versioning to new notation
virmz	2008-05-19	4.00.00	Update to ASR 3 Add Crc8 calculation
virgmi	2014-11-18	4.01.00	Update to ASR 4 Add Crc8H2F calculation
virgmi	2015-05-08	4.02.00	SafeBSW Add Crc32P4 calculation
virgmi	2016-11-24	4.03.00	Add Crc64 calculation
virljs	2020-06-02 2020-06-08	5.00.00	Add slicing-by-4 feature Add slicing-by-8 feature
virbka	2022-02-22	6.00.00	Add information about memmap handling
speter	2024-02-14	7.00.00	Add Crc32PJ193976 calculation according to SAE J1939-76 Standard

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of CRC Routines	V4.2.0
[2]	AUTOSAR	Specification of CRC Routines	V4.3.0
[3]	AUTOSAR	List of Basic Software Modules	V1.6.0
[4]	IEEE	Frank L. Berry, Michael E. Kounavis, "Novel Table Lookup-Based Algorithms for High-Performance CRC Generation", IEEE Transactions on Computers, vol. 57, no. , pp. 1550-1560, November 2008, doi:10.1109/TC.2008.85	-
[5]	Vector Informatik	MICROSAR MemMap Technical Reference	See delivery
[6]	SAE International	Surface Vehicle recommended Practice	APR2020

Scope of the Document

This technical reference describes the general use of the CRC library basis software. There are no aspects which are controller specific.

> **Caution**
> We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CRC as specified in [1]. Additionally, the CRC routine specified in [6] is described.

Latest Analyzed AUTOSAR Release:	R19-11	
AUTOSAR Schema Compatibility:	R4.0.3	
Supported Configuration Variants:	pre-compile	
Vendor ID:	CRC_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CRC_MODULE_ID	201 decimal (according to ref. [3])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

This component implements service functions in ANSI C for calculating CRC checksums. The module allows pre-compile configuration of the calculation method, which is used to compute the CRC values. The seven possible methods are table based or runtime calculation of CRC values.

There are seven different CRC calculation services available:

> Two different services for checksum calculation of <u>8bit CRC</u> value from a buffer

> Checksum calculation of <u>16bit CRC</u> value from a buffer

> Three different services for checksum calculation of <u>32bit CRC</u> value from a buffer

> Checksum calculation of <u>64bit CRC</u> value from a buffer

1.1 Architecture Overview

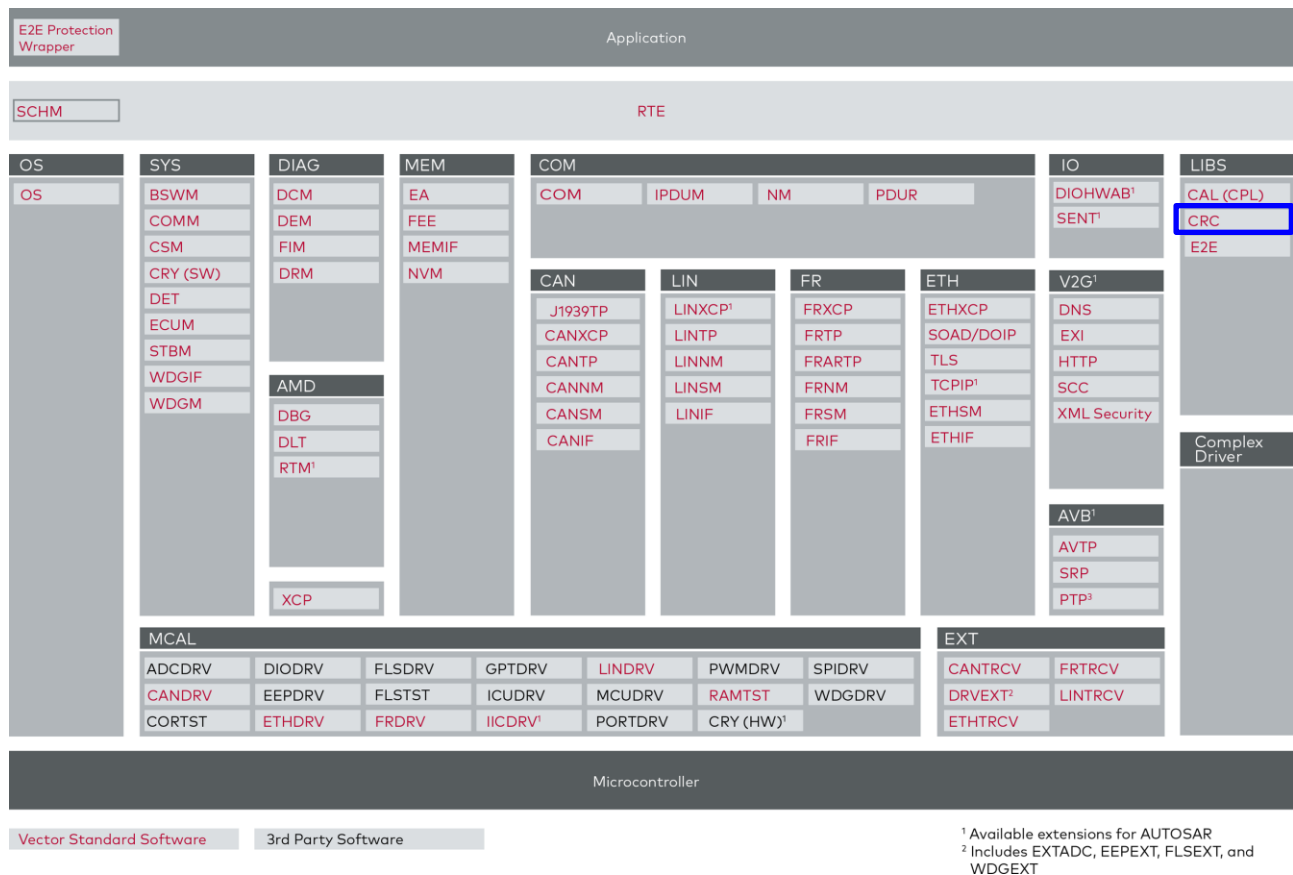The following figure shows where the CRC is located in the AUTOSAR architecture.

Figure 1-1   AUTOSAR 4.x Architecture Overview

No interface from other module is required. The CRC component is a service module. It can be called from any module or application (e.g. NVRAM Manager).

2 Functional Description

2.1 Features

The features listed in the following tables cover the complete functionality specified for the CRC.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 2-1   Supported AUTOSAR standard conform features

> Table 2-2   Not supported AUTOSAR standard conform features

Vector Informatik provides further CRC functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 2-3   Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Crc8 calculation based on the SAE-J1850 CRC8 Standard
Crc8 calculation based on 0x2F polynomial
Crc16 calculation based on the CCITT CRC16 Standard
Crc32 calculation based on the IEEE-802.3 Ethernet Standard
Crc32 calculation based on 0xF4ACFB13 polynomial (E2E Profile 4)
Crc64 calculation based on 0x42F0E1EBA9EA3693 polynomial (E2E Profile 7)
For all CRC calculations following methods can be used: > Table based calculation: results in fast execution, but increases code size (ROM table) > Runtime calculation: results in smaller code size, but slower execution time
All routines are re-entrant and can be used by multiple applications at the same time.

Table 2-1   Supported AUTOSAR standard conform features

2.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
Hardware based CRC calculation
Debugging concept

Table 2-2   Not supported AUTOSAR standard conform features

2.1.2   Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Slicing-by-4 and slicing-by-8 for the following Crc calculations: > Crc16 based on the CCITT CRC16 Standard > Crc32 based on 0xF4ACFB13 polynomial (E2E Profile 4) > Crc64 based on 0x42F0E1EBA9EA3693 polynomial (E2E Profile 7)
Crc32 calculation based on the SAE J1939-76 Standard

Table 2-3   Features provided beyond the AUTOSAR standard

2.1.3   Slicing-by-x

The slicing-by-x feature uses x number of precalculated tables instead of a single table. This results in an increase of the needed amount of ROM. However, since the values of the tables don't have to be calculated at runtime anymore and x tables can be used instead of one, there are performance gains.

The currently supported values for x are 4 and 8 for the profiles specified in Table 2-3.

Further information about the exact ramifications of this feature can be found in the paper referenced in document number [4].

2.2   Initialization

No initialization is necessary.

2.3   States

This component does not contain any state machine.

2.4   Main Functions

This component does not have a main function. It only contains the routines to calculate the checksums. These API functions are called from other modules, e.g. NVRAM Manager.

2.5   Error Handling

2.5.1   Development Error Reporting

Module CRC does not provide any development error detection mechanism.

2.5.2   Production Code Error Reporting

Module CRC does not provide any production error detection mechanism.

2.5.3   Parameter Checking

Module CRC's functions do not perform any parameter checks.

3 Integration

This chapter gives necessary information for the integration of the MICROSAR CRC into an application environment of an ECU.

3.1 Scope of Delivery

The delivery of the CRC contains the files which are described in the chapters 3.1.1 and 3.1.2.

3.1.1 Static Files

File Name	Description
Crc.c	Implementation of the CRC routines
Crc.h	Header file of the CRC routines
Crc_Tables.h	Header which includes all CRC tables.

Table 3-1   Static files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
Crc_Cfg.h	Configuration of the CRC routines

Table 3-2   Generated files

3.1.3 MemMap

Delivery with MICROSAR Release < R27:

Crc_MemMap.h is available as a static file.

Memory mapping sections of the CRC module are included within the global MemMap.h file (via Crc_MemMap.inc), which will be included by Crc_MemMap.h.

Delivery with MICROSAR Release >= R27:

Crc_MemMap.h is available as a dynamic file. This header file is generated by an own component "MemMap" (for more information refer to [5]).

Memory mapping sections of the CRC module are directly included within generated Crc_MemMap.h to reduce the build time.

4 API Description

4.1 Type Definitions

The types defined by the CRC are described in this chapter.

Type Name	C-Type	Description	Value Range
Crc_DataRefType	P2CONST	Pointer type for pointers to data buffers, from which the CRC value is to be calculated.	`0-255 (uint8)`

Table 4-1   Type definitions

Std_VersionInfoType

Struct Element Name	C-Type	Description	Value Range
Std_VersionInfoType	Struct	Returns the version information	`uint16 vendorID`
			`uint16 moduleID`
			`uint8 sw_major_version`
			`uint8 sw_minor_version`
			`uint8 sw_patch_version`

Table 4-2   Std_VersionInfoType

4.2 Interrupt Service Routines provided by CRC

The CRC component does not provide any interrupt service routines.

4.3 Services provided by CRC

4.3.1 Crc_CalculateCRC8

Prototype	
`uint8 Crc_CalculateCRC8 (Crc_DataRefType Crc_DataPtr, uint32 Crc_Length, uint8 Crc_StartValue8, boolean Crc_IsFirstCall )`	
Parameter	
`Crc_DataPtr`	Pointer to start address of data block to be calculated
`Crc_Length`	Length of data to be calculated (in bytes)
`Crc_StartValue8`	Value the algorithm starts with
`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.
Return code	
`uint8`	8 bit result of CRC calculation

Functional Description

This service performs a CRC8 calculation synchronously on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue8. The CRC8 routine is based on the SAE-J1850 CRC8 Standard:

SAE-J1850 CRC8 Standard	value
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

Table 4-3   SAE-J1850 CRC8 Standard

Particularities and Limitations

> If large data blocks have to be calculated (>32 bytes), the table based calculation method should be configured in order to decrease the calculation time.

Expected Caller Context

> There is no specific caller. CRC can be called from anywhere.

Table 4-4   Crc_CalculateCRC8

4.3.2   Crc_CalculateCRC8H2F

Prototype

```
uint8 Crc_CalculateCRC8H2F (Crc_DataRefType Crc_DataPtr, uint32 Crc_Length,
uint8 Crc_StartValue8H2F, boolean Crc_IsFirstCall )
```

Parameter

Crc_DataPtr	Pointer to start address of data block to be calculated
Crc_Length	Length of data to be calculated (in bytes)
Crc_StartValue8H2F	Value the algorithm starts with
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.

Return code

uint8	8 bit result of CRC calculation

Functional Description

This service performs a CRC8 calculation synchronously on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue8H2F. The CRC8 routine is based on the generator polynomial 0x2F:

CRC8 routine based on 0x2F polynomial	value
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

Table 4-5   CRC calculation based on 0x2F polynomial

Particularities and Limitations

> If large data blocks have to be calculated (>32 bytes), the table based calculation method should be configured in order to decrease the calculation time.

Expected Caller Context

> There is no specific caller. CRC can be called from anywhere.

Table 4-6   Crc_CalculateCRC8H2F

4.3.3   Crc_CalculateCRC16

Prototype

```
uint16 Crc_CalculateCRC16 (Crc_DataRefType Crc_DataPtr, uint32 Crc_Length,
uint16 Crc_StartValue16, boolean Crc_IsFirstCall )
```

Parameter

Crc_DataPtr	Pointer to start address of data block to be calculated
Crc_Length	Length of data to be calculated (in bytes)
Crc_StartValue16	Value the algorithm starts with
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.

Return code

uint16	16 bit result of CRC calculation

Functional Description

This service performs a CRC16 calculation synchronously on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue16. The CRC16 routine is based on the CCITT CRC16 Standard:

CCITT CRC16 Standard	value
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

Table 4-7   CCITT CRC16 Standard

Particularities and Limitations

> If large data blocks have to be calculated (>32 bytes), the table based calculation method should be configured in order to decrease the calculation time.

> If the single table doesn't provide enough speed, the slicing-by-4 feature can be enabled to further increase the performance. This comes at the cost of an additional increase in ROM size.

Expected Caller Context

> There is no specific caller. CRC can be called from anywhere.

Table 4-8   Crc_CalculateCRC16

4.3.4   Crc_CalculateCRC32

Prototype

```
uint32 Crc_CalculateCRC32 (Crc_DataRefType Crc_DataPtr, uint32 Crc_Length,
uint32 Crc_StartValue32, boolean Crc_IsFirstCall)
```

Parameter

Crc_DataPtr	Pointer to start address of data block to be calculated
Crc_Length	Length of data to be calculated (in bytes)
Crc_StartValue32	Value the algorithm starts with
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.

Return code	
`uint32`	32 bit result of CRC calculation

Functional Description

This service performs a CRC32 calculation synchronously on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue32. The CRC32 routine is based on the IEEE-802.3 CRC32 Ethernet Standard:

IEEE-802.3 CRC32 Ethernet Standard	value
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

Table 4-9   IEEE-802.3 CRC32 Ethernet Standard

*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.

Example of magic check: calculation of IEEE-802.3 CRC32 over data bytes 31h 32h 33h 34h 35h 36h 37h 38h 39h:

> CRC generation: CRC over 31h 32h 33h 34h 35h 36h 37h 38h 39h, initial value FFFFFFFFh:

>> CRC-result: CBh F4h 39h 26h (See check value)

> CRC check: CRC over 31h 32h 33h 34h 35h 36h 37h 38h 39h 26h 39h F4h CBh (CRC-result was appended to data bytes in little endian format), initial value FFFFFFFFh:

>> CRC-result: 21h 44h DFh 1Ch

>> Magic check = CRC-result XORed with 'XOR value':

   DEBB20E3h = 2144DF1Ch xor FFFFFFFFh

Particularities and Limitations

> If large data blocks have to be calculated (>32 bytes), the table based calculation method should be configured in order to decrease the calculation time.

Expected Caller Context
> There is no specific caller. CRC can be called from anywhere.

Table 4-10   Crc_CalculateCRC32

4.3.5   Crc_CalculateCRC32P4

Prototype
uint32 **Crc_CalculateCRC32P4** (Crc_DataRefType Crc_DataPtr, uint32 Crc_Length, uint32 Crc_StartValue32, boolean Crc_IsFirstCall)

Parameter	
Crc_DataPtr	Pointer to start address of data block to be calculated
Crc_Length	Length of data to be calculated (in bytes)
Crc_StartValue32	Value the algorithm starts with
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.

Return code	
uint32	32 bit result of CRC calculation

Functional Description

This service performs a CRC32 calculation synchronously on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue32. The CRC32 routine is based on the 0xF4ACFB13 polynomial (E2E Protection Profile 4):

CRC 32 routine based on 0xF4ACFB13 polyonmial	value
CRC result width	32 bits
Polynomial	F4ACFB13h
Initial value	FFFFFFFFh
Input data reflected	Yes
Result data reflected	Yes
XOR value	FFFFFFFFh
Check	1697D06Ah
Magic check	6FB32240h

Table 4-11   CRC32 calculation based on 0xF4ACFB13 polynomial (E2E Protection Profile 4)

Particularities and Limitations

> If large data blocks have to be calculated (>32 bytes), the table based calculation method should be configured in order to decrease the calculation time.
> If the single table doesn't provide enough speed, the slicing-by-4 feature can be enabled to further increase the performance. This comes at the cost of an additional increase in ROM size.

Call context

> There is no specific caller. CRC can be called from anywhere.

Table 4-12   Crc_CalculateCRC32P4

4.3.6   Crc_CalculateCRC32PJ193976

Prototype

```
uint32 Crc_CalculateCRC32PJ193976 (Crc_DataRefType Crc_DataPtr, uint32
Crc_Length, uint32 Crc_StartValue32, boolean Crc_IsFirstCall)
```

Parameter

Crc_DataPtr	Pointer to start address of data block to be calculated
Crc_Length	Length of data to be calculated (in bytes)
Crc_StartValue32	Value the algorithm starts with
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.

Return code

uint32	32 bit result of CRC calculation

Functional Description

This service performs a CRC32 calculation synchronously on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue32. The CRC32 routine is based on the SAE J1939-76 Standard used by E2E Profile J1939-76:

SAE J1939-76 CRC32 Standard	value
CRC result width	32 bits
Polynomial	6938392Dh
Initial value	FFFFFFFFh
Input data reflected	No
Result data reflected	No
XOR value	0h

Table 4-13   SAE J1939-76 CRC32 Standard

Particularities and Limitations

> The table based calculation method is not supported.

Call context

> There is no specific caller. CRC can be called from anywhere.

Table 4-14   Crc_CalculateCRC32PJ193976

4.3.7 Crc_CalculateCRC64

Prototype
uint64 **Crc_CalculateCRC64** (Crc_DataRefType Crc_DataPtr, uint32 Crc_Length, uint64 Crc_StartValue64, boolean Crc_IsFirstCall)

Parameter	
Crc_DataPtr	Pointer to start address of data block to be calculated
Crc_Length	Length of data to be calculated (in bytes)
Crc_StartValue64	Value the algorithm starts with
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.

Return code	
uint64	64 bit result of CRC calculation

Functional Description	

This service performs a CRC64 calculation synchronously on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue64. The CRC64 routine is based on the ECMA Standard used by E2E Profile 7:

ECMA Standard	value
CRC result width	64 bits
Polynomial	42F0E1EBA9EA3693h
Initial value	FFFFFFFFFFFFFFFFh
Input data reflected	Yes
Result data reflected	Yes
XOR value	FFFFFFFFFFFFFFFFh
Check	995DC9BBDF1939FAh
Magic check*	49958C9ABD7D353Fh

Table 4-15   ECMA CRC64 Standard

> **i**

> ***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.
>
> Example of magic check: calculation of ECMA CRC64 over data bytes 31h 32h 33h 34h 35h 36h 37h 38h 39h:
>
> > > CRC generation: CRC over 31h 32h 33h 34h 35h 36h 37h 38h 39h, initial value FFFFFFFFFFFFFFFFh:
> >
> > > > CRC-result: 99h 5Dh C9h BBh DFh19h 39h FAh (See check value)
> >
> > > CRC check: CRC over 31h 32h 33h 34h 35h 36h 37h 38h 39h FAh 39h 19h DFh BBh C9h 5Dh 99h (CRC-result was appended to data bytes in little endian format), initial value FFFFFFFFFFFFFFFFh:
> >
> > > > CRC-result: B6h 6Ah 73h 65h 42h 82h CAh C0h
> >
> > > > Magic check = CRC-result XORed with 'XOR value':
> >
> > > > 49958C9ABD7D353Fh = B66A73654282CAC0h XOR FFFFFFFFFFFFFFFFh

Particularities and Limitations
> If large data blocks have to be calculated (>32 bytes), the table based calculation method should be configured in order to decrease the calculation time.
> If the single table doesn't provide enough speed, the slicing-by-4 feature can be enabled to further increase the performance. This comes at the cost of an additional increase in ROM size.
Call context
> There is no specific caller. CRC can be called from anywhere.

Table 4-16   Crc_CalculateCRC64

4.3.8   Crc_GetVersionInfo

Prototype	
void **Crc_GetVersionInfo** (Std_VersionInfoType *Versioninfo)	
Parameter	
Versioninfo	Pointer to where to store the version information of this module
Return code	
None	--

Functional Description
This service returns the version information of the module. The version information includes:
> Module ID
> Vendor ID
> Vendor specific version numbers
This function is pre-compile time configurable On/Off by the configuration parameter CrcVersionInfoApi.

Particularities and Limitations
> None

Expected Caller Context
> Application

Table 4-17   Crc_GetVersionInfo

4.4   Services used by CRC

The CRC component does not use services from other modules.

4.5   Callback Functions

Module CRC's functions do not provide any callback functions.

4.6   Configurable Interfaces

4.6.1   Notifications

The CRC module does not provide notifications for upper layers. The CRC routines return the checksum; this value will be evaluated by application or other modules.

4.6.2   Callout Functions

The CRC module does not provide callout functions.

4.6.3   Hook Functions

The CRC module does not provide hook functions.

5 Configuration

5.1 Configuration Variants

The CRC supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the CRC parameters depend on the supported configuration variants. For their definitions please see the Crc_bswmd.arxml file.

CRC can be configured using the following tool:

> DaVinci Configurator 5 (AUTOSAR 4 packages only). Parameters are explained within the tool.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
CRC	Cyclic Redundancy Check

Table 6-1   Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CCITT	Comité Consultatif Internationale Télégrafique et Téléfonique
DEM	Diagnostic Event Manager
DET	Development Error Tracer
E2E	End to End
ECMA	European Computer Manufacturers Association
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
IEEE	Institute of Electrical and Electronics Engineers
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
NVRAM Manager	Non Volatile Random Access Memory Manager
ROM	Read Only Memory
RTE	Runtime Environment
SAE	Society of Automotive Engineers
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 6-2   Abbreviations

7 Contact

Visit our website for more information on

> News

> Products

> Demo software

> Support

> Training data

> Addresses

www.vector.com