User Manual

for S32K3 CRC Driver

Document Number: UM34CRCASRR21-11 Rev0000R 3.0.0 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	31.03.2023	NXP RTD Team	S32K3 Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 3.0.0

Chapter 2

Introduction

This User Manual describes NXP Semiconductor AUTOSAR CRC for S32K3XX. AUTOSAR CRC driver configuration parameters and deviations from the specification are described in Driver chapter of this document. AUTOSAR CRC driver requirements and APIs are described in the AUTOSAR CRC driver software specification document.

2.1 Supported Derivatives

The software described in this document is intended to be used with the following microcontroller devices of NXP Semiconductors:

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100
- s32k322_mqfp172 / MWCT2D16S_mqfp172

- s32k324_mqfp172 / MWCT2D17S_mqfp172
- s32k324_mapbga257
- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

All of the above microcontroller devices are collectively named as S32K3.

Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power.

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR:

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

- **Boldface** style: Used for important terms, notes and warnings.
- *Italic* style: Used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
ASM	Assembler
BSMI	Basic Software Make file Interface
CAN	Controller Area Network
C/CPP	C and C++ Source Code
CS	Chip Select
CTU	Cross Trigger Unit
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DMA	Direct Memory Access
ECU	Electronic Control Unit
FIFO	First In First Out
LSB	Least Signifigant Bit
MCU	Micro Controller Unit
MIDE	Multi Integrated Development Environment
MSB	Most Significant Bit
N/A	Not Applicable
RAM	Random Access Memory
SIU	Systems Integration Unit
SWS	Software Specification
VLE	Variable Length Encoding
XML	Extensible Markup Language

2.5 Reference List

#	Title	Version
1	Specification of CRC Driver	AUTOSAR Release R21-11
2	S32K3XX Reference Manual	Rev.6, Draft B, 01/2023
3	S32K3xx Data Sheet	Rev. 6, 11/2022
4	S32K39 and S32K37 Reference Manual	Rev.2, Draft A, — 02/2023
5	S32K396 Data Sheet	Rev. 1.1 — 08/2022
6	S32M27x Reference Manual	Rev.2, Draft A, — 02/2023
7	S32M2xx Data Sheet	Rev. 2 RC — 12/2022
8	S32K358_0P14E Mask Set Errata	Rev. 28, 9/2022
9	S32K396_0P40E Mask Set Errata	Rev. DEC2022, 12/2022
10	S32K311_0P98C Mask Set Errata	Rev. 6/March/2023, 3/2023
11	S32K312: Mask Set Errata for Mask 0P09C	Rev. 25/April/2022
12	S32K342: Mask Set Errata for Mask 0P97C	Rev. 10, 11/2022
13	S32K3x4: Mask Set Errata for Mask 0P55A/1P55A	Rev. 14/Oct/2022

Chapter 3

Driver

3.1 Requirements

Requirements for this driver are detailed in the Autosar Driver Software Specification document (See Table Reference List).

For CDD: CRC is a Complex Device Driver (CDD), so there are no AUTOSAR requirements regarding this module.

It has vendor-specific requirements and implementation.

3.2 Driver Design Summary

- The CRC driver is implemented as an complex device driver. It uses the CRC hardware peripheral which provides support for implementing the CRC calculations.
- The driver offers: Software Calculation, Lookup Table Calculation and Hardware Calculation API to the upper layer that can be used to configure the CRC and initiate CRC calculations.
- Hardware and software settings can be configured using an Autosar standard configuration tool. The information required for a CRC data calculation will be configured in a data structure that will be sent as parameter to the API of the driver.

3.3 Hardware Resources

The CRC Driver consists of:

1. CRC IP

3.4 Deviations from Requirements

The driver deviates from the AUTOSAR CRC Driver software specification in some places. The table below identifies the AUTOSAR requirements that are not implemented or out of scope for the CRC Driver.

Term	Definition
N/S	Out of scope
N/I	Not implemented
N/F	Not fully implemented

Below table identifies the AUTOSAR requirements that are not fully implemented, implemented differently or out of scope for the CRC driver.

Requirement	Status	Description	Notes
CDD_CRC_085	N/S	If the CRC calculation within the function Crc_CalculateCRC64 is performed by hardware, then the CRC module's imple-menter shall ensure reentrancy of this function by implementing a (software based) locking mechanism	There are no platforms supporting Crc64 Hardware calculation (Hardware limitation).

Requirement	Status	Description	Notes
CDD_CRC_070	N/S	These requirements are not applicable to this specification: SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00170, SRS_BSW_00412, 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_00401, 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_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00347, SRS_BSW_00305, SRS_BSW_00307, SRS_BSW_00373, SRS_BSW_00327, SRS_BSW_00335, SRS_BSW_00350, SRS_BSW_00410, SRS_BSW_00314, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00302, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00304, SRS_BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00371, SRS_BSW_00358, SRS_BSW_00414, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00009, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334	Not a requirement.

3.5 Driver Limitations

The CRC Driver has the following limitations:
- Post Build Compiler not implemented.
- For each CRC type, the user can configure only one of the three methods: Hardware, Look-up-Table, Runtime(Software).
- Crc64Mode hardware not supported.

3.6 Driver usage and configuration tips

This driver is an Complex Device Driver. Complete driver functionality together with API description can be found below.

3.6.1 CRC Calculation Type

- When Calculation Type is CRC_IP_LOOKUP_TABLES_CALCULATION or CRC_IP_HARDWARE_↩ CALCULATION, the Protocol shall be selected from the supported ones.
- Autosar Library supports only the Autosar Protocols.
- Polynomial configuration is enabled when the following Protocols are selected: CRC_PROTOCOL_8BIT, CRC_PROTOCOL_16BIT, CRC_PROTOCOL_32BIT, CRC_PROTOCOL_64BIT.
- Note: Each Calculation Type supports a specific list of Protocols. If out of range there will be an error message.

3.6.1.1 Calculation Type supported by CRC_IP_LOOKUP_TABLES_CALCULATION

- CRC_PROTOCOL_8BIT_SAE_J1850
- CRC_PROTOCOL_8BIT_H2F
- CRC_PROTOCOL_16BIT_CCITT
- CRC_PROTOCOL_16BIT_ARC
- CRC_PROTOCOL_32BIT_ETHERNET
- CRC_PROTOCOL_32BIT_E2EP4
- CRC_PROTOCOL_64BIT_ECMA

3.6.1.2 Calculation Type supported by CRC_IP_SOFTWARE_CALCULATION

- CRC_PROTOCOL_8BIT_CUSTOM
- CRC_PROTOCOL_16BIT_CUSTOM
- CRC_PROTOCOL_32BIT_CUSTOM
- CRC_PROTOCOL_64BIT_CUSTOM
- CRC_PROTOCOL_8BIT_SAE_J1850
- CRC_PROTOCOL_8BIT_H2F
- CRC_PROTOCOL_16BIT_CCITT
- CRC_PROTOCOL_16BIT_ARC
- CRC_PROTOCOL_32BIT_ETHERNET
- CRC_PROTOCOL_32BIT_E2EP4
- CRC_PROTOCOL_64BIT_ECMA

3.6.1.3 Calculation Type supported by CRC_IP_HARDWARE_CALCULATION

- CRC_PROTOCOL_16BIT_CUSTOM
- CRC_PROTOCOL_32BIT_CUSTOM
- CRC_PROTOCOL_16BIT_CCITT
- CRC_PROTOCOL_32BIT_ETHERNET

3.6.2 CRC Initialization.

- The Crc_Init() function shall initialize the CRC hardware peripheral(s) and the internal driver context, according to the input configuration data. The application shall ensure that the Crc_Init() function is called first. Only the Crc_GetVersionInfo() can be called before Crc_Init().

3.6.3 CRC Channel Configuration.

- The function receives pointer to a configuration structure that shall be loaded into the Logic Channel. According to input parameters this function configures channel's CRC width, polynomial and whether Swap and/or Inversion functionality is applied on CRC input data and/or signature or not.

3.6.4 CRC Channel Calculation.

- The CRC driver provides the function of calculating the CRC of a data stream of a certain length. The CRC calculation uses the CRC width, polynomial, and previously set swap/inversion options for the channel using the Crc_SetChannelConfig() function. The CRC is computed on an input vector data pointed by pCrcData and has a length set to CrcLength. The function synchronously calculates the CRC and returns the resulting CRC.

3.6.5 CRC Channel Get Result.

- Driver provides functionality for reading CRC result from required channel. The Crc_GetChannelResult() function reads CRC Results (CRC checksum).

3.6.6 CRC Examples for using the driver

- This sub-chapter lists some examples of how to use the CRC Driver.

3.6.6.1 Example Calculate Autosar CRC16 CCITT.

To compute CRC checksum according to CCITT-FALSE CRC16 standard, please follow the steps below:

1. Open EB Tresos then create New Configuration Project

2. In Tresos GUI, add CRC Configuration in tab "Module Configuration"

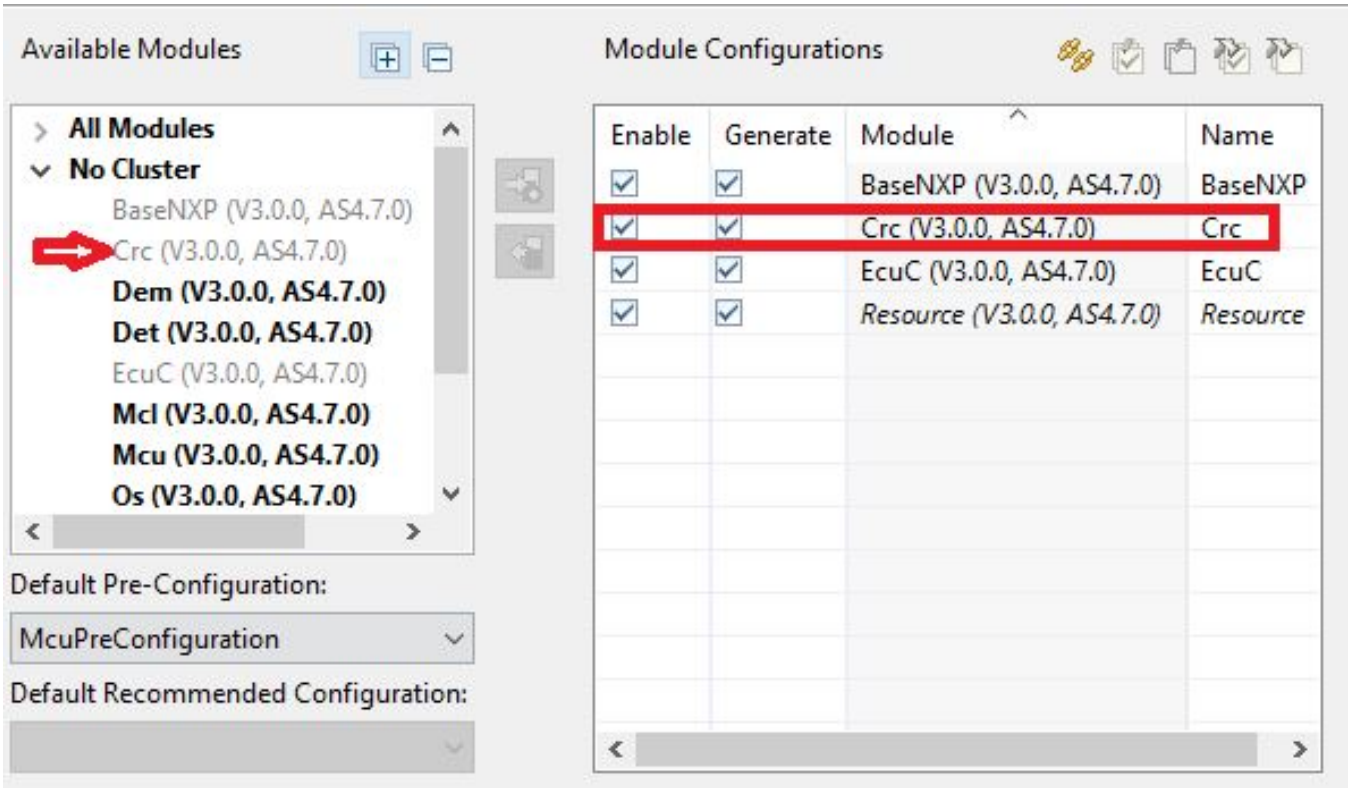

Figure 3.1 CRC add Configuration

3. In "CRC Channel Configuration", add new CRC logic channel with default config.

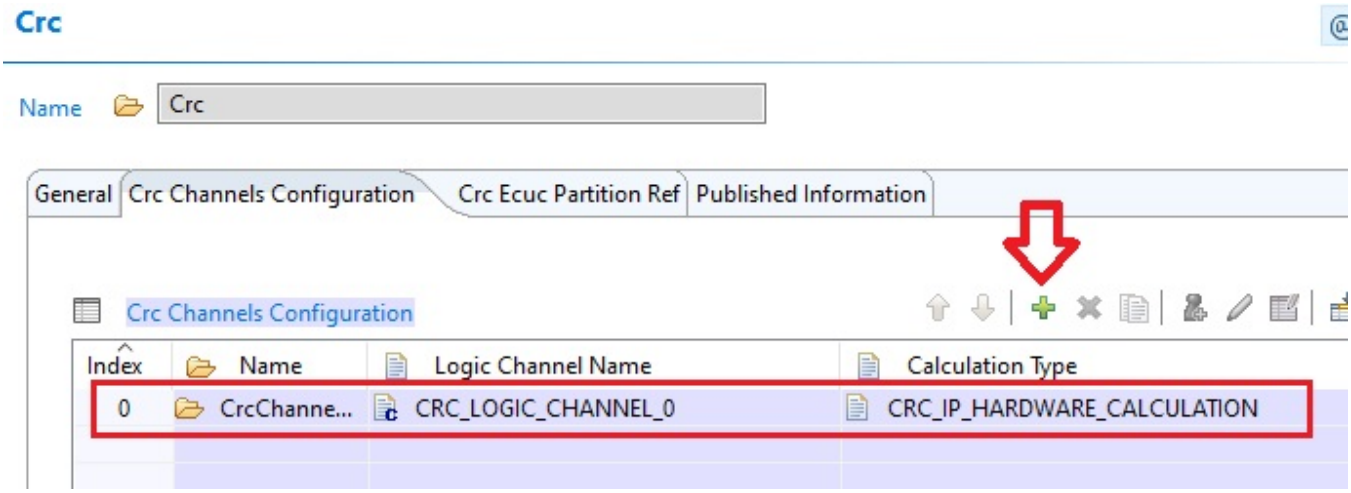

Figure 3.2 CRC add logic Channel with default config

4. In CRC logic channel configuration, using the following values for the attributes.

   - Select "Calculates Type" is CRC_IP_HARDWARE_CALCULATION
   - Set checkbox "Autosar Library Enable" is Enable.

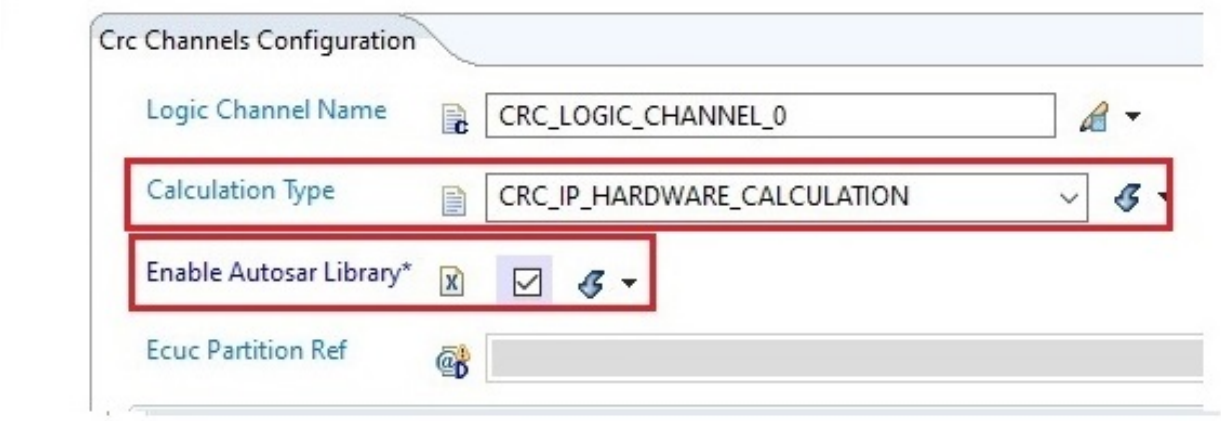

Figure 3.3 CRC config Calculates Type and Enable Autosar

   - Set "Protocol Type" is CRC_PROTOCOL_16BIT_CCITT_FALSE.

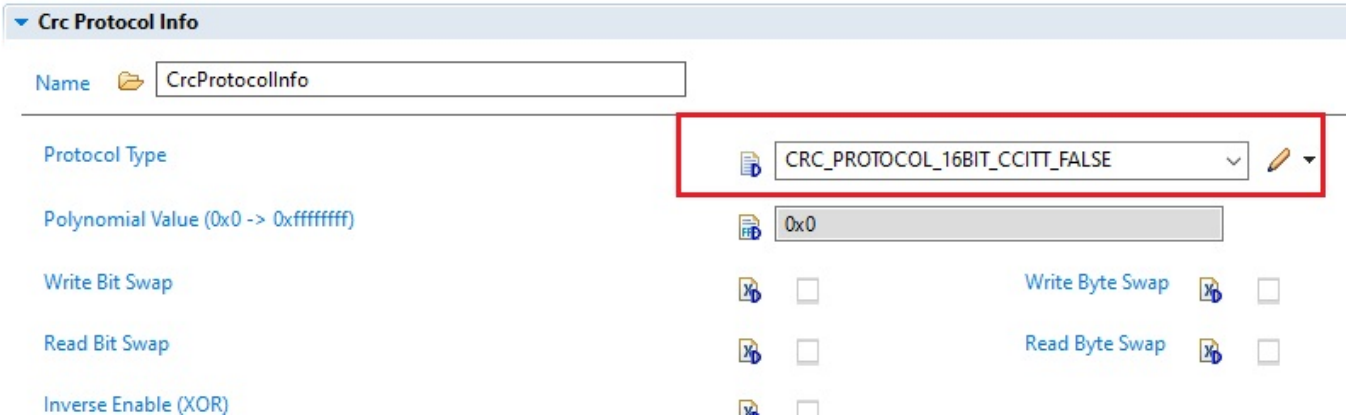

Figure 3.4 CRC config protocol is 16BIT_CCITT_FALSE

5. Generate the configuration files from Tresos without errors or warnings.

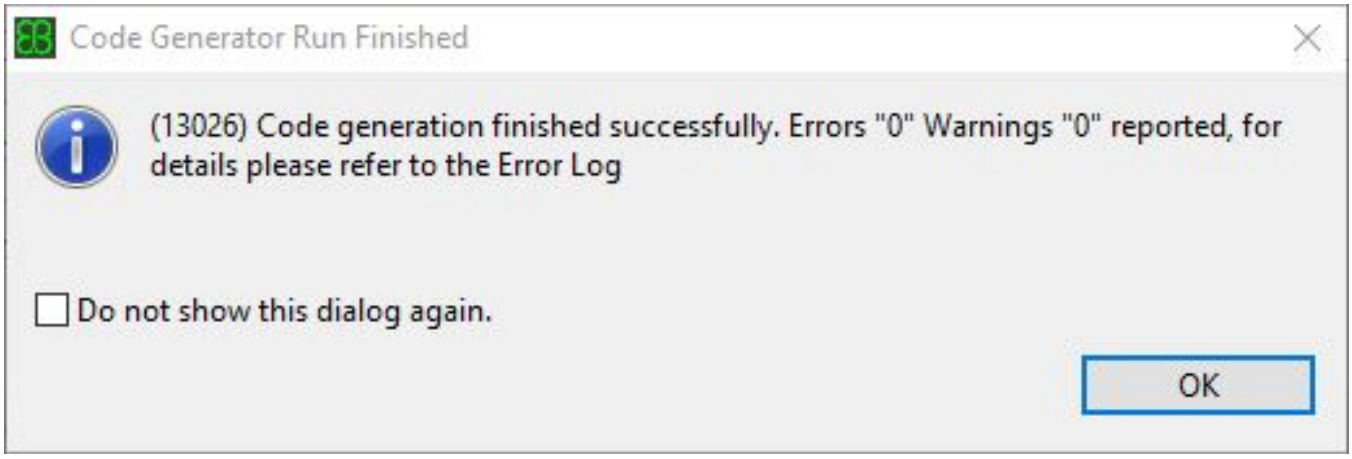

Figure 3.5 Status Generate Config

6. Write the application code, following the steps.

   - Call the function Crc_Init() providing it as parameter a NULL pointer.
   - Call the function Crc_Calculate_CRC16 () to start calculation. This function will use the pre-configured Logic Channels in Auto Mode. After the calculation is completed, it will return the results

```
/* Initialize CRC driver */
Crc_Init(NULL_PTR);
/* Start calculate then get result */
result = Crc_CalculateCRC16((const uint8_t *)CRC_data, CRC_DATA_SIZE, 0xFFFF, true);
```

Figure 3.6 CRC application code

3.6.6.2 Example Calculate Autosar CRC32 ETHERNET.

To compute CRC checksum according to IEEE 802.3 Ethernet standard, please follow the steps below:

1. Please follow the steps 1 to 3 from "Example Calculate Autosar CRC16 CCITT"
2. In CRC logic channel configuration, using the following values for the attributes.
   - Select "Calculates Type" is CRC_IP_HARDWARE_CALCULATION
   - Set checkbox "Autosar Library Enable" is Enable.

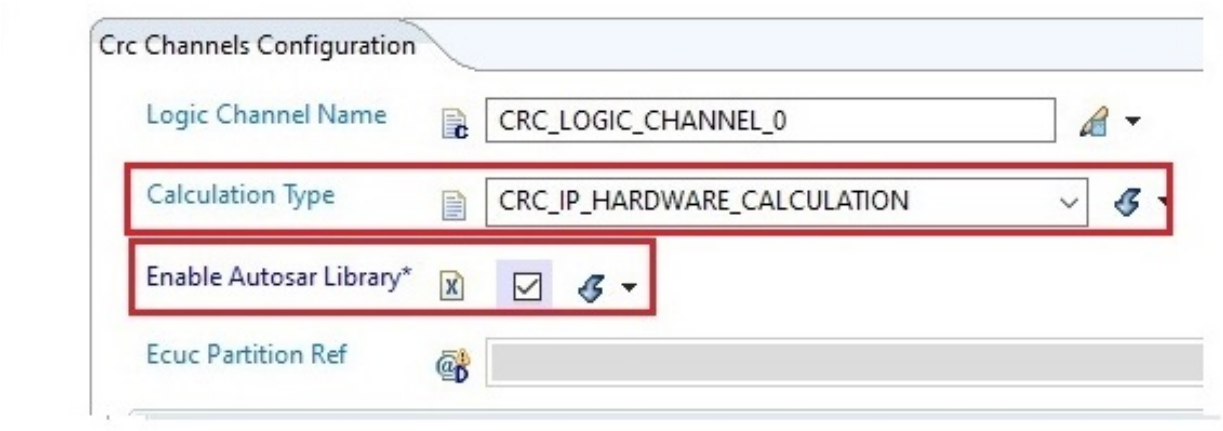

Figure 3.7 CRC config CRC config Calculates Type and Enable Autosar

   - Set "Protocol Type" is CRC_PROTOCOL_32BIT_ETHERNET.

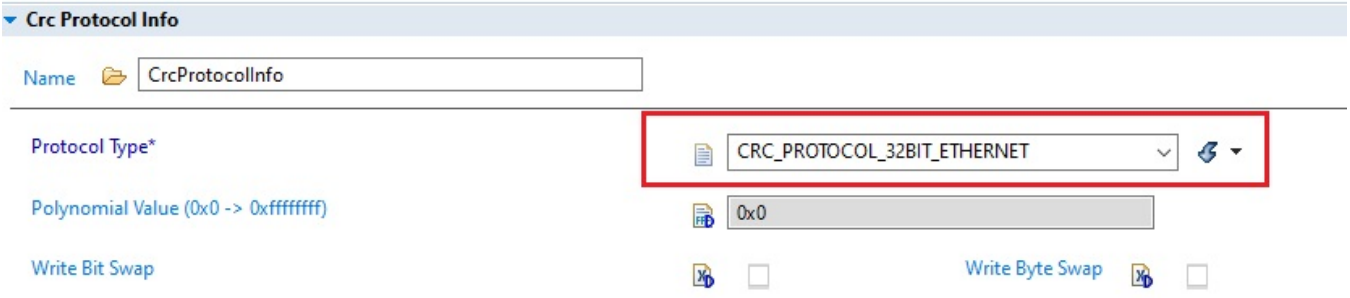

Figure 3.8 CRC config protocol is CRC_PROTOCOL_32BIT_ETHERNET

3. Generate the configuration files from Tresos without errors or warnings.
4. Write the application code, following the steps.
   - Call the function Crc_Init() providing it as parameter a NULL pointer.

- Call the function Crc_CalculateCRC32() to start calculation. This function will use the pre-configured Logic Channels in Auto Mode. After the calculation is completed, it will return the results

```
/* Initialize CRC driver */
Crc_Init(NULL_PTR);
/* Start calculate then get result */
result = Crc_CalculateCRC32((const uint8_t *)CRC_data, CRC_DATA_SIZE, 0xFFFF, true);
```

Figure 3.9 CRC application code

3.6.6.3 Example Calculate CRC CUSTOM.

To compute any CRC CUSTOM Type is CRC_PROTOCOL_8BIT_CUSTOM (or CRC_PROTOCOL_16BIT_CUSTOM, CRC_PROTOCOL_32BIT_CUSTOM, CRC_PROTOCOL_64BIT_CUSTOM), please follow the steps below:

1. -> 3. Please refer "Example Calculate Autosar CRC16 CCITT"
2. In CRC logic channel configuration, using the following values for the attributes.
   - Select "Calculates Type" is CRC_IP_SOFTWARE_CALCULATION

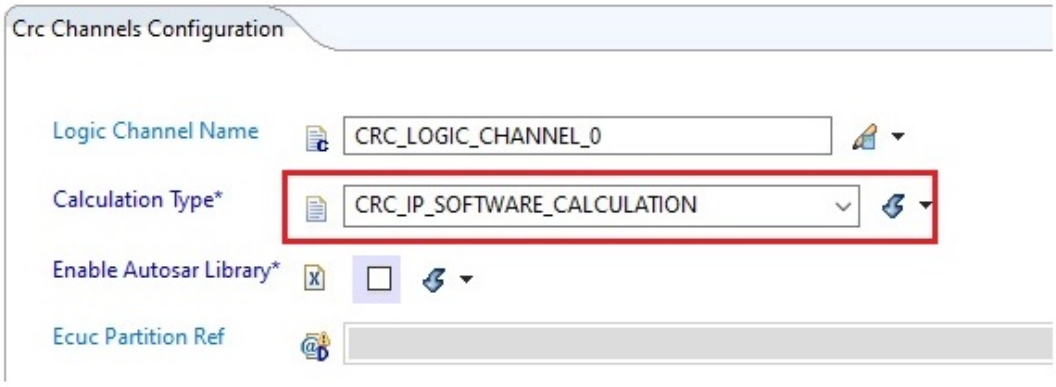

Figure 3.10 CRC Calculate Tye is CRC_IP_SOFTWARE_CALCULATION

- Set "Protocol Type" is CRC_PROTOCOL_8BIT_CUSTOM and configure Polynomial Value, Swap Option and Inverse Enable

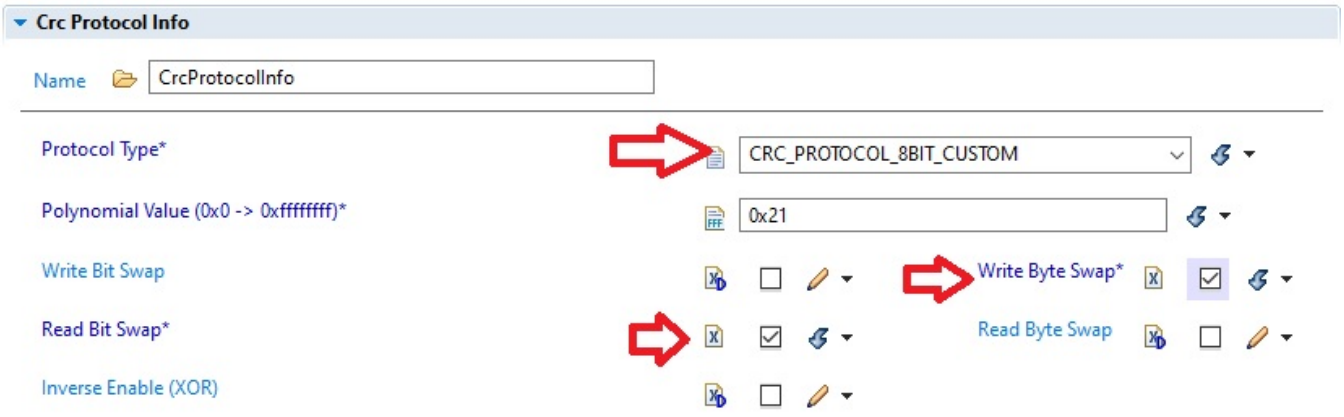

Figure 3.11 CRC config protocol CRC_PROTOCOL_8BIT_CUSTOM

3. Generate the configuration files from Tresos without errors or warnings.

4. Write the application code, following the steps.

   - Call the function Crc_Init() providing it as parameter a NULL pointer.
   - Call the function Crc_SetChannelCalculate() to start calculation. Calculation is based on User's logic channel configuration, Crc data path(pCrcData), Crc length (CrcLength) and start value (CrcStartValue). After the calculation is complete, this function will return the result

```
/* Initialize CRC driver */
Crc_Init(NULL_PTR);
/* Start calculate then get result */
result = Crc_SetChannelCalculate((const uint8_t *)CRC_data, CRC_DATA_SIZE, 0xFFFF, true);
```

Figure 3.12 CRC application code

3.6.6.4 Example Calculate CRC with DMA support (only support CRC_IP_HARDWARE_CALCULATION).

To compute CRC checksum use DMA support according to CCITT-FALSE CRC16 standard, please follow the steps below:

1. -> 3. Please refer "Example Calculate Autosar CRC16 CCITT"

2. In CRC logic channel configuration, using the following values for the attributes.

   - Note: Autosar Library Function does not support DMA

- Set checkbox "DMA support Calculate" is Enable.

Figure 3.13 CRC DMA support Calculate Enable

- Select "Calculates Type" is CRC_IP_HARDWARE_CALCULATION
- If checkbox "Autosar Library Enable" is selected and "DMA Channel Enable" is selected Tresos will show "ERROR"

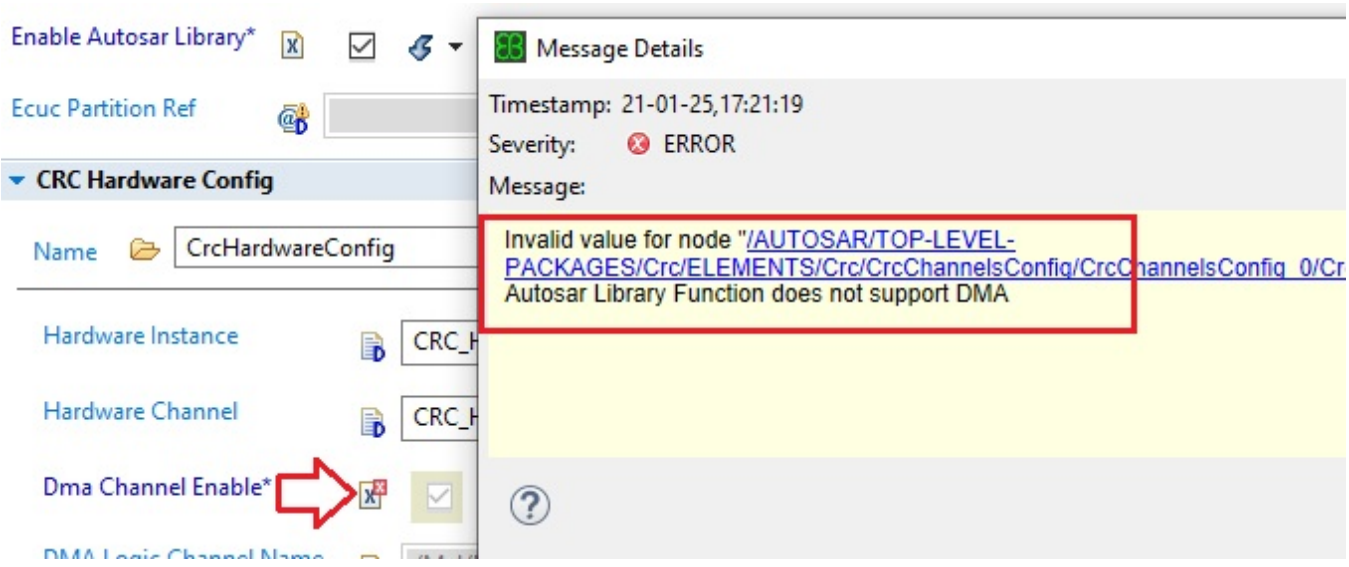

Figure 3.14 CRC config Enable "Autosar Library" and "DMA Channel"

- Set "Protocol Type" is CRC_PROTOCOL_16BIT_CCITT_FALSE.

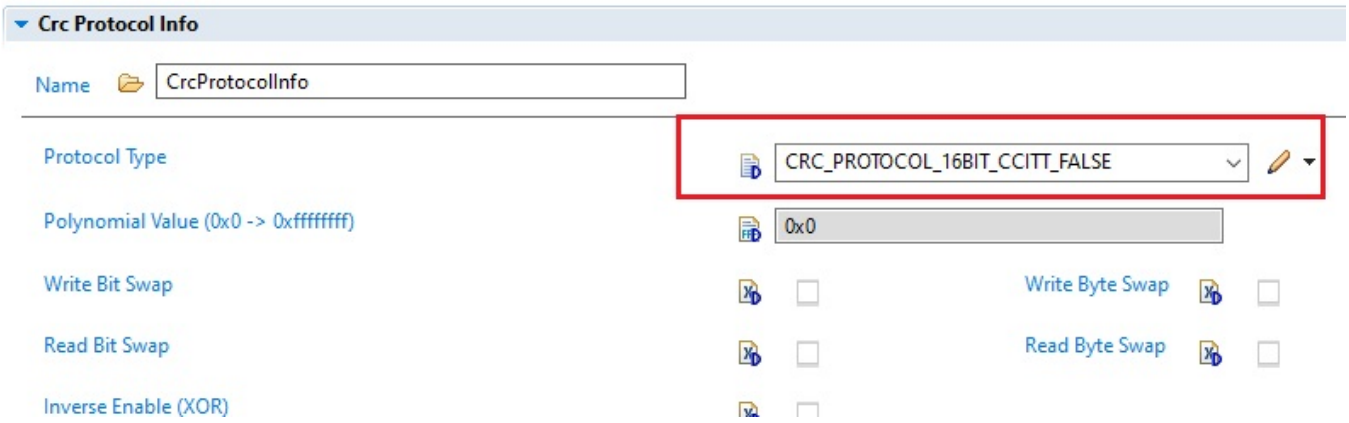

Figure 3.15 CRC config protocol is 16BIT_CCITT_FALSE

- Config hardware Channel CRC and DMA

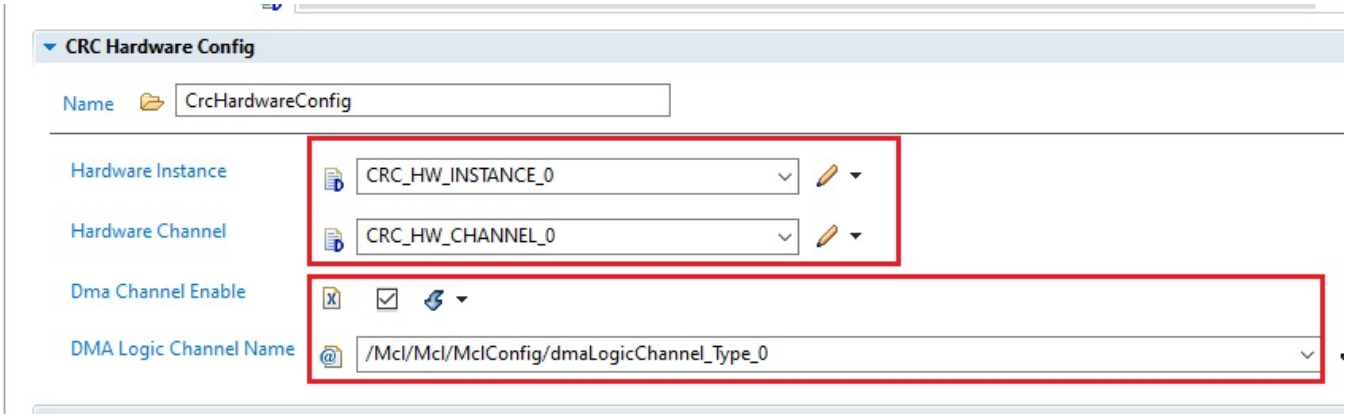

Figure 3.16 Config hardware Channel CRC and DMA

3. Generate the configuration files from Tresos without errors or warnings.

4. Write the application code, following the steps.

   - Call the function Dma_Ip_Init() to initialize the DMA driver.
   - Call the function Crc_Init() providing it as parameter a NULL pointer.
   - Call the function Crc_SetChannelCalculate() to start calculation. Calculation is based on User's logic channel configuration, Crc data path(pCrcData), Crc length (CrcLength) and start value (CrcStartValue). After the calculation is complete, this function will return the result

```
/* Initialize Mcl driver */
Mcl_Init(NULL_PTR);

/* Initialize CRC driver */
Crc_Init(NULL_PTR);

/* Start calculate with CRC_PROTOCOL_16BIT_CCITT_FALSE with SOFTWARE_CALCULATION */
CrcResult = Crc_SetChannelCalculate(CRC_LOGIC_CHANNEL_1, &CRC_data[0], CRC_DATA_SIZE, 0U, TRUE);
```

Figure 3.17 CRC application code

- Note for Crc_SetChannelCalculate(): "When DMA is used, the returned result shall be 0U if the DMA Channel did not done transferring all data."

3.7 Runtime errors

The driver generates the following DET errors at runtime.

Function	Error Code	Condition triggering the error
Crc_Init()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_SetChannelConfig()	CRC_E_INVALID_CHANNEL	API is called with invalid channel ID parameter.
Crc_SetChannelCalculate()	CRC_E_INVALID_CHANNEL	API is called with invalid channel ID parameter.
Crc_SetChannelCalculate()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_GetChannelResult()	CRC_E_INVALID_CHANNEL	API is called with invalid channel ID parameter.
Crc_CalculateCRC8()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_CalculateCRC8()	CRC_E_INVALID_WIDTH_TYPE	API is called with invalid with type support.
Crc_CalculateCRC8H2F()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_CalculateCRC8H2F()	CRC_E_INVALID_WIDTH_TYPE	API is called with invalid with type support.
Crc_CalculateCRC16()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_CalculateCRC16()	CRC_E_INVALID_WIDTH_TYPE	API is called with invalid with type support.
Crc_CalculateCRC16ARC()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_CalculateCRC16ARC()	CRC_E_INVALID_WIDTH_TYPE	API is called with invalid with type support.
Crc_CalculateCRC32()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.

Function	Error Code	Condition triggering the error
Crc_CalculateCRC32()	CRC_E_INVALID_WIDTH_TYPE	API is called with invalid with type support.
Crc_CalculateCRC32P4()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_CalculateCRC32P4()	CRC_E_INVALID_WIDTH_TYPE	API is called with invalid with type support.
Crc_CalculateCRC64()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.
Crc_CalculateCRC64()	CRC_E_INVALID_WIDTH_TYPE	API is called with invalid with type support.
Crc_GetVersionInfo()	CRC_E_INVALID_POINTER	API is called with a NULL pointer as parameter.

3.8 Symbolic Names Disclaimer

All containers having symbolicNameValue set to TRUE in the AUTOSAR schema will generate defines like:

#define <Mip>Conf_<Container_ShortName>_<Container_ID>

For this reason it is forbidden to duplicate the names of such containers across the RTD configurations or to use names that may trigger other compile issues (e.g. match existing `#ifdefs` arguments).

Chapter 4

Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the driver. All the parameters are described below.

- Module Crc
  - Container CrcGeneral
    - Parameter CrcDetectError
    - Parameter CrcEnableUserModeSupport
    - Parameter CrcDmaSupportEnable
    - Parameter CrcMultiCoreEnable
    - Parameter CrcVersionInfoApi
    - Parameter Crc8Mode
    - Parameter Crc8H2FMode
    - Parameter Crc16Mode
    - Parameter Crc16ARCMode
    - Parameter Crc32Mode
    - Parameter Crc32P4Mode
    - Parameter Crc64Mode
  - Container CrcChannelConfig
    - Parameter CrcLogicChannelName
    - Parameter CrcAutosarSelect
    - Parameter CrcCalculationType
    - Reference CrcPartitionRefOfChannel
    - Container CrcHardwareConfig
      - Parameter CrcHwInstance
      - Parameter CrcHwChannel
      - Parameter CrcDmaChannelEnable
      - Reference CrcDmaLogicChannelName
    - Container CrcProtocolInfo
      - Parameter CrcProtocolType
      - Parameter CrcPolynomialValue
      - Parameter CrcWriteBitSwap
      - Parameter CrcWriteByteSwap
      - Parameter CrcReadBitSwap

        - Parameter CrcReadByteSwap
        - Parameter CrcInversionEnable
    - Container CrcEcucPartitionRefArray
        - Reference CrcEcucPartitionRef
    - Container CommonPublishedInformation
        - Parameter ArReleaseMajorVersion
        - Parameter ArReleaseMinorVersion
        - Parameter ArReleaseRevisionVersion
        - Parameter ModuleId
        - Parameter SwMajorVersion
        - Parameter SwMinorVersion
        - Parameter SwPatchVersion
        - Parameter VendorApiInfix
        - Parameter VendorId

4.1 Module Crc

Vendor specific: Configuration of the Crc (Cyclic Redundancy Check) module.

Included containers:

- CrcGeneral
- CrcChannelConfig
- CrcEcucPartitionRefArray
- CommonPublishedInformation

Property	Value
type	ECUC-MODULE-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantSupport	false
supportedConfigVariants	VARIANT-PRE-COMPILE

4.2 Container CrcGeneral

Crc General

All general parameters of the Crc driver are collected here.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.3 Parameter CrcDetectError

CRC Development Error Detect

Compile switch to enable/disable development error detection for this module.

Unchecked: Crc Development error detection disabled

Checked : Crc Development error detection enabled

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	true

4.4 Parameter CrcEnableUserModeSupport

Crc User Mode Support

When this parameter is enabled, the Crc module will adapt to run from User Mode, with the following measures: Configuring REG_PROT for Crc IPs so that the registers under protection can be accessed from user mode by setting UAA bit in REG_PROT_GCR to 1

For more information and availability on this platform, please see chapter User Mode Support in IM

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF

Property	Value
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.5 Parameter CrcDmaSupportEnable

Dma Support Calculate

Check this in order to be able to use DMA in the Crc driver. Leaving this unchecked will allow the Crc driver to compile with no dependencies from the Mcl driver.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.6 Parameter CrcMultiCoreEnable

Crc Multicore Enable

This parameter globally enables the possibility to support multicore. If this parmeter is enabled, at least one EcucPartition needs to be defined (in all variants).

Note: This is an Implementation Specific Parameter.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.7 Parameter CrcVersionInfoApi

CRC VersionInfo Api

Compile switch to enable/disable the version information API.

Checked : API enabled

Unchecked: API disabled

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.8 Parameter Crc8Mode

Switch to select one of the available Crc 8-bit (SAE J1850) calculation methods

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

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	False
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_8_TABLE
literals	['CRC_8_TABLE', 'CRC_8_HARDWARE', 'CRC_8_RUNTIME']

4.9 Parameter Crc8H2FMode

Switch to select one of the available Crc 8-bit (2Fh polynomial) calculation methods

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

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	False
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_8H2F_TABLE
literals	['CRC_8H2F_TABLE', 'CRC_8H2F_HARDWARE', 'CRC_8H2F_RUNTIME']

4.10 Parameter Crc16Mode

Switch to select one of the available Crc 16-bit (CCITT-FALSE) calculation methods

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

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	False
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_16_TABLE
literals	['CRC_16_TABLE', 'CRC_16_HARDWARE', 'CRC_16_RUNTIME']

4.11 Parameter Crc16ARCMode

Switch to select one of the available CRC-16/ARC (polynomial 8005) calculation methods

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.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	False
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_16ARC_TABLE
literals	['CRC_16ARC_TABLE', 'CRC_16ARC_HARDWARE', 'CRC_16ARC_RU↩NTIME']

4.12 Parameter Crc32Mode

Switch to select one of the available Crc 32-bit (IEEE-802.3 CRC32 Ethernet Standard) calculation methods

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

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	False
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_32_TABLE
literals	['CRC_32_TABLE', 'CRC_32_HARDWARE', 'CRC_32_RUNTIME']

4.13 Parameter Crc32P4Mode

Switch to select one of the available Crc 32-bit E2E Profile 4 calculation methods.

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.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	False
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_32P4_TABLE
literals	['CRC_32P4_TABLE', 'CRC_32P4_HARDWARE', 'CRC_32P4_RUNTIME']

4.14 Parameter Crc64Mode

Switch to select one of the available Crc 64-bit calculation methods

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.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	False
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_64_TABLE
literals	['CRC_64_TABLE', 'CRC_64_HARDWARE', 'CRC_64_RUNTIME']

4.15 Container CrcChannelConfig

Crc Channels Configuration

Configuration of an individual Crc channel. Symbolic names will be generated for each channel.

Included subcontainers:

- CrcHardwareConfig
- CrcProtocolInfo

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.16 Parameter CrcLogicChannelName

Logic Channel Name

Channel used for Crc calculation

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP

Property	Value
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_LOGIC_CHANNEL_0

4.17 Parameter CrcAutosarSelect

Autosar Selection

Select the Autosar Mode to run.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	NON_AUTOSAR
literals	['NON_AUTOSAR', 'AUTOSAR_CRC_8', 'AUTOSAR_CRC_8H2F', 'AUT↩OSAR_CRC_16', 'AUTOSAR_CRC_16ARC', 'AUTOSAR_CRC_32', 'AU↩TOSAR_CRC_32P4', 'AUTOSAR_CRC_64']

4.18 Parameter CrcCalculationType

Calculation Type

Select Crc Calculation Type.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF

Property	Value
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_IP_TABLE_CALCULATION
literals	['CRC_IP_TABLE_CALCULATION', 'CRC_IP_HARDWARE_CALCUL↩ ATION', 'CRC_IP_RUNTIME_CALCULATION']

4.19 Reference CrcPartitionRefOfChannel

Partition Ref Of Channel

Maps a Crc hardware unit to zero or one ECUC partition to limit the access to this hardware unit. The ECUC partitions referenced are a subset of the ECUC partitions where the Crc driver is mapped to.

Tags: atp.Status=draft

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/EcuC/EcucPartitionCollection/EcucPartition

4.20 Container CrcHardwareConfig

This container contains the hardware configuration parameters of the Crc module.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.21 Parameter CrcHwInstance

Hardware Instance

Identifies the Crc Hardware Instance.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_HW_INSTANCE_0
literals	['CRC_HW_INSTANCE_0']

4.22 Parameter CrcHwChannel

Hardware Channel

Selects one of the Crc hardware channels available on the device.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A

Property	Value
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRC_HW_CHANNEL_0
literals	['CRC_HW_CHANNEL_0']

4.23 Parameter CrcDmaChannelEnable

Dma Channel Enable

Checked : Enabled

Unchecked: Disabled

Note: DMA mode does not support for CRC Autosar Library

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.24 Reference CrcDmaLogicChannelName

DMA Logic Channel Name

DMA Logic Channel is used for this Crc logic channel.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false

Property	Value
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Mcl/MclConfig/dmaLogicChannel_Type

4.25 Container CrcProtocolInfo

This container configuration parameters of protocol type

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.26 Parameter CrcProtocolType

Protocol Type

Identifies the Crc Protocol Type.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

Property	Value
defaultValue	CRC_PROTOCOL_8BIT_SAE_J1850
literals	['CRC_PROTOCOL_8BIT_SAE_J1850', 'CRC_PROTOCOL_8BIT_H2F', 'CRC_PROTOCOL_16BIT_CCITT_FALSE', 'CRC_PROTOCOL_16BIT↩_ARC', 'CRC_PROTOCOL_32BIT_ETHERNET', 'CRC_PROTOCOL_32↩BIT_E2E_P4', 'CRC_PROTOCOL_64BIT_ECMA', 'CRC_PROTOCOL_↩8BIT_CUSTOM', 'CRC_PROTOCOL_16BIT_CUSTOM', 'CRC_PROTO↩COL_32BIT_CUSTOM', 'CRC_PROTOCOL_64BIT_CUSTOM']

4.27 Parameter CrcPolynomialValue

Polynomial Value

Start value when the algorithm starts in process Calculate CRC

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0x

4.28 Parameter CrcWriteBitSwap

Write Bit Swap

Enumerator that defines Crc Write data bitwise functionality.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

Property	Value
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.29 Parameter CrcWriteByteSwap

Write Bit Swap

Enumerator that defines Crc Write data bytewise functionality.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.30 Parameter CrcReadBitSwap

Read Bit Swap

Enumerator that defines Crc Read data bitwise functionality.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.31 Parameter CrcReadByteSwap

Read Byte Swap

Enumerator that defines Crc Read data bytewise functionality.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.32 Parameter CrcInversionEnable

Inverse Enable (XOR)

The result shall be complement(inversion) of the actual checksum.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.33 Container CrcEcucPartitionRefArray

Crc Ecuc Partition Ref Array.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.34 Reference CrcEcucPartitionRef

Maps the Crc driver to zero or multiple ECUC partitions to make the driver API available in the according partition.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/EcuC/EcucPartitionCollection/EcucPartition

4.35 Container CommonPublishedInformation

Common Published Information

Common container, aggregated by all modules. It contains published information about vendor and versions.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.36 Parameter ArReleaseMajorVersion

AUTOSAR Release Major Version

Major version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	4
max	4
min	4

4.37 Parameter ArReleaseMinorVersion

AUTOSAR Release Minor Version

Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	7
max	7
min	7

4.38 Parameter ArReleaseRevisionVersion

AUTOSAR Release Revision Version

Revision version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.39 Parameter ModuleId

Module ID

Module ID of this module from Module List.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	201
max	201
min	201

4.40 Parameter SwMajorVersion

Software Major Version

Major version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	3
max	3
min	3

4.41 Parameter SwMinorVersion

Software Minor Version

Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.42 Parameter SwPatchVersion

Software Patch Version

Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.43 Parameter VendorApiInfix

Vendor Api Infix

In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the name of APIs is extended by the VendorId and a vendor specific name.

This parameter is used to specify the vendor specific name. In total, the implementation specific name is generated as follows:

<ModuleName>_>VendorId>_<VendorApiInfix>.

E.g. assuming that the VendorId of the implementor is 123 and the implementer chose a VendorApiInfix of "v11r456" a api name Can_Write defined in the SWS will translate to Can_123_v11r456Write.

This parameter is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	

4.44 Parameter VendorId

Vendor Id

Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	43
max	43
min	43

This chapter describes the Tresos configuration plug-in for the CRC Driver. The most of the parameters are described below.

Chapter 5

Module Index

5.1 Software Specification

Here is a list of all modules:

Chapter 6

Module Documentation

6.1 CRC HLD Driver

6.1.1 Detailed Description

Data Structures

- struct Crc_PartitionType

  CRC Partition Configuration Type. *More...*
- struct Crc_InitType

  Initialization data for the CRC driver. *More...*

Macros

- #define CDD_CRC_MODULE_ID

  Parameters that shall be published within the Crc driver header file and also in the module's description file.
- #define CRC_INSTANCE_ID

  ID of CRC Instance.
- #define CRC_CALCULATECRC8_ID

  API service ID for Crc_CalculateCRC8 function.
- #define CRC_CALCULATECRC16_ID

  API service ID for Crc_CalculateCRC16 function.
- #define CRC_CALCULATECRC32_ID

  API service ID for Crc_CalculateCRC32 function.
- #define CRC_GETVERSIONINFO_ID

  API service ID for Crc_GetVersionInfo function.
- #define CRC_CALCULATECRC8H2F_ID

  API service ID for Crc_CalculateCRC8H2F function.
- #define CRC_CALCULATECRC32P4_ID

  API service ID for Crc_CalculateCRC32P4 function.
- #define CRC_CALCULATECRC64_ID

API service ID for Crc_CalculateCRC64 function.

- #define CRC_CALCULATECRC16ARC_ID

  API service ID for Crc_CalculateCRC16ARC function.
- #define CRC_INIT_ID

  API service ID for Crc_Init function.
- #define CRC_SETCHANNELCONFIG_ID

  API service ID for Crc_SetChannelConfig function.
- #define CRC_SETCHANNELCALCULATE_ID

  API service ID for Crc_SetChannelCalculate function.
- #define CRC_GETCHANNELRESULT_ID

  API service ID for Crc_GetChannelResult function.
- #define CRC_E_INVALID_CHANNEL

  API service is called with wrong channel identifier.
- #define CRC_E_INVALID_POINTER

  API service is called with NULL pointer parameter.
- #define CRC_E_PARAM_CONFIG

  The CRC module is not properly configured.
- #define CRC_E_INIT_FAILED

  The CRC module is not properly initialized.
- #define CRC_TYPES_VENDOR_ID

  Parameters that shall be published within the Crc driver header file and also in the module's description file.

Types Reference

- typedef Crc_Ip_LogicChannelConfigType Crc_ChannelConfigType

  This type contains the CRC Channel Configuration.
- typedef Crc_Ip_ProtocolType Crc_ProtocolType

  This type contains the CRC Channel Protocol.

Function Reference

- void Crc_Init (const Crc_InitType *ConfigPtr)

  This service will store the Crc driver installation configuration based on user configuration.
- void Crc_SetChannelConfig (const uint32 Channel, const Crc_ChannelConfigType *pxChannelConfig)

  This function initializes the CRC Channel configuration by logicChannel's info.
- uint64 Crc_SetChannelCalculate (const uint32 Channel, const uint8 *pCrcData, const uint32 CrcLength, const uint64 CrcStartValue, const boolean IsFirstCall)

  This function shall start algorithm calculate CRC.
- uint64 Crc_GetChannelResult (const uint32 Channel)

  This function get result CRC after finish calculated.
- uint8 Crc_CalculateCRC8 (const uint8 *Crc_DataPtr, uint32 Crc_Length, uint8 Crc_StartValue8, boolean Crc_IsFirstCall)

  This function calculates CRC8 and returns the result.
- uint8 Crc_CalculateCRC8H2F (const uint8 *Crc_DataPtr, uint32 Crc_Length, uint8 Crc_StartValue8H2F, boolean Crc_IsFirstCall)

This function calculates CRC8H2F and returns the result.

- uint16 Crc_CalculateCRC16 (const uint8 *Crc_DataPtr, uint32 Crc_Length, uint16 Crc_StartValue16, boolean Crc_IsFirstCall)

  This function calculates CRC16 and returns the result.
- uint16 Crc_CalculateCRC16ARC (const uint8 *Crc_DataPtr, uint32 Crc_Length, uint16 Crc_StartValue16, boolean Crc_IsFirstCall)

  This function calculates CRC16 and returns the result.
- uint32 Crc_CalculateCRC32 (const uint8 *Crc_DataPtr, uint32 Crc_Length, uint32 Crc_StartValue32, boolean Crc_IsFirstCall)

  This function calculates CRC32 and returns the result.
- uint32 Crc_CalculateCRC32P4 (const uint8 *Crc_DataPtr, uint32 Crc_Length, uint32 Crc_StartValue32, boolean Crc_IsFirstCall)

  This function calculates CRC32P4 and returns the result.
- uint64 Crc_CalculateCRC64 (const uint8 *Crc_DataPtr, uint32 Crc_Length, uint64 Crc_StartValue64, boolean Crc_IsFirstCall)

  This function calculates CRC64 and returns the result.
- void Crc_GetVersionInfo (Std_VersionInfoType *pVersionInfo)

  Software module version query.

6.1.2 Data Structure Documentation

6.1.2.1 struct Crc_PartitionType

CRC Partition Configuration Type.

The Channel is identified by the following Partition

Definition at line 111 of file Crc_Types.h.

6.1.2.2 struct Crc_InitType

Initialization data for the CRC driver.

A pointer to such a structure is provided to the CRC initialization routines for configuration.

Definition at line 123 of file Crc_Types.h.

6.1.3 Macro Definition Documentation

6.1.3.1 CDD_CRC_MODULE_ID

```
#define CDD_CRC_MODULE_ID
```

Parameters that shall be published within the Crc driver header file and also in the module's description file.

Definition at line 61 of file CDD_Crc.h.

6.1.3.2 CRC_INSTANCE_ID

```
#define CRC_INSTANCE_ID
```

ID of CRC Instance.

Parameters used when raising an error/exception

Definition at line 131 of file CDD_Crc.h.

6.1.3.3 CRC_CALCULATECRC8_ID

```
#define CRC_CALCULATECRC8_ID
```

API service ID for Crc_CalculateCRC8 function.

Parameters used when raising an error/exception

Definition at line 142 of file CDD_Crc.h.

6.1.3.4 CRC_CALCULATECRC16_ID

```
#define CRC_CALCULATECRC16_ID
```

API service ID for Crc_CalculateCRC16 function.

Parameters used when raising an error/exception

Definition at line 148 of file CDD_Crc.h.

6.1.3.5 CRC_CALCULATECRC32_ID

```
#define CRC_CALCULATECRC32_ID
```

API service ID for Crc_CalculateCRC32 function.

Parameters used when raising an error/exception

Definition at line 154 of file CDD_Crc.h.

6.1.3.6 CRC_GETVERSIONINFO_ID

```
#define CRC_GETVERSIONINFO_ID
```

API service ID for Crc_GetVersionInfo function.

Parameters used when raising an error/exception

Definition at line 160 of file CDD_Crc.h.

6.1.3.7 CRC_CALCULATECRC8H2F_ID

```
#define CRC_CALCULATECRC8H2F_ID
```

API service ID for Crc_CalculateCRC8H2F function.

Parameters used when raising an error/exception

Definition at line 166 of file CDD_Crc.h.

6.1.3.8 CRC_CALCULATECRC32P4_ID

```
#define CRC_CALCULATECRC32P4_ID
```

API service ID for Crc_CalculateCRC32P4 function.

Parameters used when raising an error/exception

Definition at line 172 of file CDD_Crc.h.

6.1.3.9 CRC_CALCULATECRC64_ID

```
#define CRC_CALCULATECRC64_ID
```

API service ID for Crc_CalculateCRC64 function.

Parameters used when raising an error/exception

Definition at line 178 of file CDD_Crc.h.

6.1.3.10 CRC_CALCULATECRC16ARC_ID

```
#define CRC_CALCULATECRC16ARC_ID
```

API service ID for Crc_CalculateCRC16ARC function.

Parameters used when raising an error/exception

Definition at line 184 of file CDD_Crc.h.

6.1.3.11 CRC_INIT_ID

```
#define CRC_INIT_ID
```

API service ID for Crc_Init function.

Parameters used when raising an error/exception

Definition at line 195 of file CDD_Crc.h.

6.1.3.12 CRC_SETCHANNELCONFIG_ID

```
#define CRC_SETCHANNELCONFIG_ID
```

API service ID for Crc_SetChannelConfig function.

Parameters used when raising an error/exception

Definition at line 201 of file CDD_Crc.h.

6.1.3.13 CRC_SETCHANNELCALCULATE_ID

```
#define CRC_SETCHANNELCALCULATE_ID
```

API service ID for Crc_SetChannelCalculate function.

Parameters used when raising an error/exception

Definition at line 207 of file CDD_Crc.h.

6.1.3.14 CRC_GETCHANNELRESULT_ID

```
#define CRC_GETCHANNELRESULT_ID
```

API service ID for Crc_GetChannelResult function.

Parameters used when raising an error/exception

Definition at line 213 of file CDD_Crc.h.

6.1.3.15 CRC_E_INVALID_CHANNEL

```
#define CRC_E_INVALID_CHANNEL
```

API service is called with wrong channel identifier.

Parameter is used when raising a Det error

Definition at line 224 of file CDD_Crc.h.

6.1.3.16 CRC_E_INVALID_POINTER

```
#define CRC_E_INVALID_POINTER
```

API service is called with NULL pointer parameter.

Parameter is used when raising a Det error

Definition at line 231 of file CDD_Crc.h.

6.1.3.17 CRC_E_PARAM_CONFIG

```
#define CRC_E_PARAM_CONFIG
```

The CRC module is not properly configured.

Parameter is used when raising a Det error

Definition at line 238 of file CDD_Crc.h.

6.1.3.18 CRC_E_INIT_FAILED

```
#define CRC_E_INIT_FAILED
```

The CRC module is not properly initialized.

Parameter is used when raising a Det error

Definition at line 245 of file CDD_Crc.h.

6.1.3.19 CRC_TYPES_VENDOR_ID

```
#define CRC_TYPES_VENDOR_ID
```

Parameters that shall be published within the Crc driver header file and also in the module's description file.

Definition at line 60 of file Crc_Types.h.

6.1.4 Types Reference

6.1.4.1 Crc_ChannelConfigType

```
typedef Crc_Ip_LogicChannelConfigType Crc_ChannelConfigType
```

This type contains the CRC Channel Configuration.

The Channel is identified by the following structure configure parameter for each CRC Channel

Definition at line 260 of file CDD_Crc.h.

6.1.4.2 Crc_ProtocolType

```
typedef Crc_Ip_ProtocolType Crc_ProtocolType
```

This type contains the CRC Channel Protocol.

The Protocol shall be selected from the available list. Software CRC supports all protocols. Table based CRC support the following protocols: Hardware CRC supports implementation specific protocols.

Definition at line 136 of file Crc_Types.h.

6.1.5 Function Reference

6.1.5.1 Crc_Init()

```
void Crc_Init (
            const Crc_InitType * ConfigPtr )
```

This service will store the Crc driver installation configuration based on user configuration.

This service is a non-reentrant function that shall store user configuration. The initialization is applied for the enabled IPs, configured statically.

Parameters

in	*pxCrcInit*	- Pointer to the Crc specific configuration structure that contains static configuration.

Returns

void

6.1.5.2 Crc_SetChannelConfig()

```
void Crc_SetChannelConfig (
            const uint32 Channel,
            const Crc_ChannelConfigType * pxChannelConfig )
```

This function initializes the CRC Channel configuration by logicChannel's info.

This service is a reentrant function that shall initialize parameters list for the CRC Channel. The list is composed of an array of Crc Channel parameters settings.

Parameters

in	*Channel*	- Logic Channel Tag defined by the user.
in	*pxChannelConfig*	- Pointer to the Logic Channel Config

Returns

void

6.1.5.3 Crc_SetChannelCalculate()

```
uint64 Crc_SetChannelCalculate (
            const uint32 Channel,
            const uint8 * pCrcData,
            const uint32 CrcLength,
            const uint64 CrcStartValue,
            const boolean IsFirstCall )
```

This function shall start algorithm calculate CRC.

This service is a reentrant function that shall start the Channel to calculate the CRC using the configured algorithm.

Parameters

in	*Channel*	- Logic Channel Tag defined by the user.
in	*∗pCrcData*	- Pointer to the Crc Data Input.
in	*CrcLength*	- Length of crcDataPtr block to be calculated in bytes
in	*CrcStartValue*	- Start value (seed Value) when the algorithm starts.
in	*IsFirstCall*	- TRUE: ignore CrcStartValue the initialization value is known by the chosen algorithm. • FALSE: initialization CrcStartValue is interpreted to be the return value of the previous function

Returns

- 32 bit result of CRC calculation

6.1.5.4 Crc_GetChannelResult()

```
uint64 Crc_GetChannelResult (
             const uint32 Channel )
```

This function get result CRC after finish calculated.

This service is a function that shall get result CRC calculated by CRC Channel Allocated

Parameters

in	*Channel*	- Logic Channel Tag defined by the user.

Returns

Result of CRC calculation.

6.1.5.5 Crc_CalculateCRC8()

```
uint8 Crc_CalculateCRC8 (
             const uint8 * Crc_DataPtr,
             uint32 Crc_Length,
             uint8 Crc_StartValue8,
             boolean Crc_IsFirstCall )
```

This function calculates CRC8 and returns the result.

This service is a reentrant function that shall perform a CRC8 calculation on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue8

Parameters

in	*∗Crc_DataPtr*	- Pointer to start address of data block to be calculated.
in	*Crc_Length*	- Length of data block to be calculated in bytes.
in	*Crc_StartValue8*	- Length value of the DataPtr.
in	*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.

Returns

8 bit result of CRC calculation.

6.1.5.6 Crc_CalculateCRC8H2F()

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

This function calculates CRC8H2F and returns the result.

This service is a reentrant function that shall perform a CRC8H2F calculation on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue8H2F

Parameters

in	*∗Crc_DataPtr*	- Pointer to start address of data block to be calculated.
in	*Crc_Length*	- Length of data block to be calculated in bytes.
in	*Crc_StartValue8H2F*	- Length value of the DataPtr.
in	*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.

Returns

8 bit result of CRC calculation.

6.1.5.7 Crc_CalculateCRC16()

```
uint16 Crc_CalculateCRC16 (
            const uint8 * Crc_DataPtr,
            uint32 Crc_Length,
            uint16 Crc_StartValue16,
            boolean Crc_IsFirstCall )
```

This function calculates CRC16 and returns the result.

This service is a reentrant function that shall perform a CRC16 calculation on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue16

Parameters

in	*∗Crc_DataPtr*	- Pointer to start address of data block to be calculated.
in	*Crc_Length*	- Length of data block to be calculated in bytes.
in	*Crc_StartValue16*	- Length value of the DataPtr.
in	*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.

Returns

16 bit result of CRC calculation.

6.1.5.8 Crc_CalculateCRC16ARC()

```
uint16 Crc_CalculateCRC16ARC (
            const uint8 * Crc_DataPtr,
            uint32 Crc_Length,
            uint16 Crc_StartValue16,
            boolean Crc_IsFirstCall )
```

This function calculates CRC16 and returns the result.

This service is a reentrant function that shall perform a CRC16 calculation on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue16

Parameters

in	*∗Crc_DataPtr*	- Pointer to start address of data block to be calculated.
in	*Crc_Length*	- Length of data block to be calculated in bytes.
in	*Crc_StartValue16*	- Length value of the DataPtr.
in	*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.

Returns

16 bit result of CRC calculation.

6.1.5.9 Crc_CalculateCRC32()

```
uint32 Crc_CalculateCRC32 (
            const uint8 * Crc_DataPtr,
            uint32 Crc_Length,
            uint32 Crc_StartValue32,
            boolean Crc_IsFirstCall )
```

This function calculates CRC32 and returns the result.

This service is a reentrant function that shall perform a CRC32 calculation on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue32

Parameters

in	**Crc_DataPtr*	- Pointer to start address of data block to be calculated.
in	*Crc_Length*	- Length of data block to be calculated in bytes.
in	*Crc_StartValue32*	- Length value of the DataPtr.
in	*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.

Returns

32 bit result of CRC calculation.

6.1.5.10 Crc_CalculateCRC32P4()

```
uint32 Crc_CalculateCRC32P4 (
            const uint8 * Crc_DataPtr,
            uint32 Crc_Length,
            uint32 Crc_StartValue32,
            boolean Crc_IsFirstCall )
```

This function calculates CRC32P4 and returns the result.

This service is a reentrant function that shall perform a CRC32P4 calculation on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue32

Parameters

in	*∗Crc_DataPtr*	- Pointer to start address of data block to be calculated.
in	*Crc_Length*	- Length of data block to be calculated in bytes.
in	*Crc_StartValue32*	- Length value of the DataPtr.
in	*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.

Returns

32 bit result of CRC calculation.

6.1.5.11 Crc_CalculateCRC64()

```
uint64 Crc_CalculateCRC64 (
            const uint8 * Crc_DataPtr,
            uint32 Crc_Length,
            uint64 Crc_StartValue64,
            boolean Crc_IsFirstCall )
```

This function calculates CRC64 and returns the result.

This service is a reentrant function that shall perform a CRC64 calculation on Crc_Length data bytes, pointed to by Crc_DataPtr, with the starting value of Crc_StartValue64

Parameters

in	*∗Crc_DataPtr*	- Pointer to start address of data block to be calculated.
in	*Crc_Length*	- Length of data block to be calculated in bytes.
in	*u64Crc_StartValue*	- Length value of the DataPtr.
in	*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.

Returns

64 bit result of CRC calculation.

6.1.5.12 Crc_GetVersionInfo()

```
void Crc_GetVersionInfo (
            Std_VersionInfoType * pVersionInfo )
```

Software module version query.

Returns the version information of this module

Parameters

in	*pVersionInfo*	- Pointer to get Version Information

Returns

void

6.2 CRC IPL Driver

6.2.1 Detailed Description

Data Structures

- struct Crc_Ip_CrcProtocolInfoType

  This type contains the CRC Protocol Information. More...
- struct Crc_Ip_LogicChannelStateType

  Internal State for the Logic CHannel. More...
- struct Crc_Ip_LogicChannelConfigType

  This type contains the CRC Ip Channel Configuration. More...
- struct Crc_Ip_LogicChannelType

  This type contains the Crc Ip Channel Type. More...
- struct Crc_Ip_InitType

  This type contains the Crc Ip Initialization. More...

Macros

- #define CRC_IP_DEVASSERT_VENDOR_ID

  Parameters that shall be published within the standard types header file and also in the module's description file.

Enum Reference

- enum Crc_Ip_CrcWidthType

  This type contains the CRC Width Type.
- enum Crc_Ip_CalculationType

  This type contains the CRC Ip Channel Configuration.
- enum Crc_Ip_ProtocolType

  This type contains the CRC Ip Channel Protocol.

Function Reference

- void Crc_Ip_Init (const Crc_Ip_InitType *const pxCrcIpInit)

  This function initializes the CRC Driver in IP Layer.
- void Crc_Ip_SetChannelConfig (const uint32 LogicChannel, const Crc_Ip_LogicChannelConfigType *px↩ LocLogicChannelConfig)

  This function initializes the CRC Channel configuration by LogicChannel's info.
- uint64 Crc_Ip_SetChannelCalculate (const uint32 LogicChannel, const uint8 *DataPtr, const uint32 Length, const uint64 StartValue, const boolean IsFirstCall)

  This function shall start algorithm calculate CRC in IP layer.
- uint64 Crc_Ip_GetChannelResult (const uint32 LogicChannel)

  This function get result CRC after finish calculated in IP Layer.

6.2.2 Data Structure Documentation

6.2.2.1 struct Crc_Ip_CrcProtocolInfoType

This type contains the CRC Protocol Information.

Definition at line 103 of file Crc_Ip_State.h.

6.2.2.2 struct Crc_Ip_LogicChannelStateType

Internal State for the Logic CHannel.

It shall contain the last loaded configuration by the user for the specified Logic Channel

Definition at line 120 of file Crc_Ip_State.h.

Data Fields

Type	Name	Description
Crc_Ip_ProtocolType	Protocol	Protocol Type: CRC-CCITT, CRC-32, CRC-8, CRC-8-H2F
const Crc_Ip_CrcProtocolInfoType *	CrcProtocolInfo	Crc Protocol Information
uint64	CrcResult	Stores the last CRC Result.

6.2.2.3 struct Crc_Ip_LogicChannelConfigType

This type contains the CRC Ip Channel Configuration.

The Channel is identified by the following structure Configure parammeter for each CRC Channel

Definition at line 158 of file Crc_Ip_Types.h.

Data Fields

Type	Name	Description
Crc_Ip_ProtocolType	Protocol	CRC protocol type.
uint64	PolynomValue	Polynomial input, MSBit first. Example polynomial: 0x1021U = 1_0000_0010_0001 = x^12+x^5+1
boolean	WriteBitSwap	Swap when writing CRC input data.
boolean	WriteByteSwap	Swap when writing CRC input data.
boolean	ReadBitSwap	Swap result bits after algorithm finished calculating CRC .
boolean	ReadByteSwap	Swap result bytes after algorithm finished calculating CRC .
boolean	InverseEnable	The inversion operation is a complement (or negation) of the content.
const uint8 *	LookUpTable	LookUp Table address for 8 bits

6.2.2.4 struct Crc_Ip_LogicChannelType

This type contains the Crc Ip Channel Type.

The Crc Ip Initialization contains all the information required to initialize the CRC Driver Internal driver structure.

Definition at line 175 of file Crc_Ip_Types.h.

Data Fields

Type	Name	Description
Crc_Ip_CalculationType	CalculationType	Software, Table, Hardware Calculation
uint8	HwInst	CRC hardware instance.
uint8	HwChannel	CRC hardware channel.
Crc_Ip_LogicChannelConfigType *	LogicChannelConfig	Pointer to LogicChannelConfig.

6.2.2.5 struct Crc_Ip_InitType

This type contains the Crc Ip Initialization.

The Crc Ip Initialization contains all the information required to initialize the CRC Driver Internal driver structure.

Definition at line 192 of file Crc_Ip_Types.h.

Data Fields

Type	Name	Description
const Crc_Ip_LogicChannelType *const *	LogicChannelConfigList	Pointer to list LogicChannelConfig

6.2.3 Macro Definition Documentation

6.2.3.1 CRC_IP_DEVASSERT_VENDOR_ID

```
#define CRC_IP_DEVASSERT_VENDOR_ID
```

Parameters that shall be published within the standard types header file and also in the module's description file.

Definition at line 66 of file Crc_Ip_Devassert.h.

6.2.4 Enum Reference

6.2.4.1 Crc_Ip_CrcWidthType

enum Crc_Ip_CrcWidthType

This type contains the CRC Width Type.

Select Width for each Channel to calculate CRC.

Definition at line 87 of file Crc_Ip_State.h.

6.2.4.2 Crc_Ip_CalculationType

enum Crc_Ip_CalculationType

This type contains the CRC Ip Channel Configuration.

The Channel is identified by the following structure Configure parammeter for each CRC Channel Internal driver enumeration.

Enumerator

CRC_IP_RUNTIME_CALCULATION	Slower execution (software calculate), but small code size (no ROM table)
CRC_IP_TABLE_CALCULATION	Fast execution (software calculate), but larger code size (ROM table)
CRC_IP_HARDWARE_CALCULATION	Fast execution, less CPU time

Definition at line 121 of file Crc_Ip_Types.h.

6.2.4.3 Crc_Ip_ProtocolType

enum Crc_Ip_ProtocolType

This type contains the CRC Ip Channel Protocol.

The Protocol shall be selected from the available list. Software CRC supports all protocols. Table based CRC support the following protocols: Hardware CRC supports implementation specific protocols.

Enumerator

CRC_PROTOCOL_8BIT_CUSTOM	Generate 8-bit CUSTOM CRC with user defined algorithm
CRC_PROTOCOL_16BIT_CUSTOM	Generate 16-bit CUSTOM CRC with user defined algorithm
CRC_PROTOCOL_32BIT_CUSTOM	Generate 32-bit CUSTOM CRC with user defined algorithm

Enumerator

CRC_PROTOCOL_64BIT_CUSTOM	Generate 64-bit CUSTOM CRC with user defined algorithm
CRC_PROTOCOL_8BIT_H2F	Generate 8-bit CRC: H2F, AUTOSAR 4.0 CRC-8 0x2F
CRC_PROTOCOL_8BIT_SAE_J1850	Generate 8-bit CRC: VDA CAN, SAE-J1850 CRC-8.
CRC_PROTOCOL_16BIT_ARC	Generate 16-bit CRC ARC code with Polynomial: 0x8005
CRC_PROTOCOL_16BIT_CCITT_FALSE	Generate 16-bit CRC: x.25. CCITT16 CRC-16 0x1021
CRC_PROTOCOL_32BIT_E2E_P4	Generate 32-bit CRC: E2E Profile 4 CRC-32 P4 0xF4ACFB13
CRC_PROTOCOL_32BIT_ETHERNET	Generate 32-bit CRC: Ethernet (IEEE 802,3) CCITT32 CRC-32 0x04C11DB7
CRC_PROTOCOL_64BIT_ECMA	Generate 64-bit CRC: ECMA
CRC_PROTOCOL_INVALID	Invalid protocol

Definition at line 134 of file Crc_Ip_Types.h.

6.2.5 Function Reference

6.2.5.1 Crc_Ip_Init()

```
void Crc_Ip_Init (
            const Crc_Ip_InitType *const pxCrcIpInit )
```

This function initializes the CRC Driver in IP Layer.

This service is a non-reentrant function that shall initialize the Crc driver. The user must make sure that the clock is enabled.

Parameters

in	*paxLogicChannelConfig*	- Pointer to the configuration structure.

Returns

void

6.2.5.2 Crc_Ip_SetChannelConfig()

```
void Crc_Ip_SetChannelConfig (
            const uint32 LogicChannel,
            const Crc_Ip_LogicChannelConfigType * pxLocLogicChannelConfig )
```

This function initializes the CRC Channel configuration by LogicChannel's info.

This service is a reentrant function that shall initializes parameters list for the CRC Channel. The list is composed of an array of Crc Channel parameters settings.

Parameters

in	*LogicChannel*	- Specifies the Logic Channel Tag defined by the user
in	*pxLocLogicChannelConfig*	- Specifies the Logic Channel Config defined by the user

Returns

void

6.2.5.3 Crc_Ip_SetChannelCalculate()

```
uint64 Crc_Ip_SetChannelCalculate (
            const uint32 LogicChannel,
            const uint8 * DataPtr,
            const uint32 Length,
            const uint64 StartValue,
            const boolean IsFirstCall )
```

This function shall start algorithm calculate CRC in IP layer.

This service is a reentrant function that shall start algorithm calculate in CRC Channel with LogicChannelConfig info

Parameters

in	*LogicChannel*	- Specifies the Logic Channel Tag defined by the user.
in	*∗DataPtr*	- Pointer to start address of data block to be calculated..
in	*Length*	- Length of DataPtr block to be calculated in bytes. ∗
in	*StartValue*	- Start value (seed Value) when the algorithm starts.
in	*IsFirstCall*	- TRUE: ignore StartValue the initialization value is known by the chosen algorithm. • FALSE: initialization StartValue is interpreted to be the return value of the previous function

Returns

Result of CRC calculated.

6.2.5.4 Crc_Ip_GetChannelResult()

```
uint64 Crc_Ip_GetChannelResult (
            const uint32 LogicChannel )
```

This function get result CRC after finish calculated in IP Layer.

This service is a function that shall get result from CRC Channel after finish calculated

Parameters

in	*LogicChannel*	- Specifies the Logic Channel Tag defined by the user

Returns

result of CRC calculated.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.