MCAL User Manual for Crc

32-bit TriCore™ AURIX™ TC3xx microcontroller

About this document

Scope and purpose

This User Manual is intended to enable users to integrate the Microcontroller Abstraction Layer (MCAL) software for the TriCore™ AURIX™ family of 32-bit microcontrollers.

This document describes responsibilities of integrator in-charge of integrating MCAL software with the basic software (BSW) stack. This document also provides detailed information on safety, configuration and functions along with examples of usage of significant features.

Note: Detailed information about package installation, safety and other generic information that are common across all modules are provided in MCAL User Manual General.

Intended audience

This document is intended for anyone using the Crc module of the TC3xx MCAL software.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, sub-menus
Italics	Denotes variable(s) and reference(s)
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
>	Indicates that a cascading sub-menu opens when you select a menu item
[cover parentID=<alpha numeric value>]	Used for traceability completeness. Reader should ignore these.

Reference documents

This User Manual should be read in conjunction with the following documents:

- AURIX™ TC3xx MCAL User Manual General
- Specification of CRC Driver, AUTOSAR_SWS_CRC_Driver, AUTOSAR Release 4.2.2
- Specification of CRC Driver, AUTOSAR_SWS_CRC_Driver, AUTOSAR Release 4.4.0

Table of contents

About this document . . . 1
Table of contents . . . 2
1 CRC driver . . . 5
1.1 User information . . . 5
1.1.1 Description . . . 5
1.1.2 Hardware-software mapping . . . 5
1.1.2.1 FCE: Primary hardware peripheral . . . 6
1.1.2.2 SCU: dependent hardware peripheral . . . 6
1.1.2.3 DMA: dependent hardware peripheral . . . 6
1.1.2.4 SRC: dependent hardware peripheral . . . 7
1.1.3 File structure . . . 7
1.1.3.1 C file structure . . . 7
1.1.3.2 Code generator plugin files . . . 9
1.1.4 Integration hints . . . 9
1.1.4.1 Integration with AUTOSAR stack . . . 10
1.1.4.2 Multicore and Resource Manager . . . 12
1.1.4.3 MCU support . . . 12
1.1.4.4 Port support . . . 12
1.1.4.5 DMA support . . . 13
1.1.4.6 Interrupt connections . . . 14
1.1.4.7 Example usage . . . 15
1.1.5 Key architectural considerations . . . 19
1.1.5.1 AUTOSAR modes of operation . . . 19
1.1.5.2 CPU CRCN instruction . . . 19
1.1.5.3 DMA based operation . . . 20
1.2 Assumptions of Use (AoU) . . . 21
1.3 Reference information . . . 22
1.3.1 Configuration interfaces . . . 22
1.3.1.1 Container: Crc . . . 22
1.3.1.2 Container: CrcGeneral . . . 22
1.3.1.2.1 Crc16ARCMode . . . 22
1.3.1.2.2 Crc16ARCReturnErrorValue . . . 23
1.3.1.2.3 Crc16Mode . . . 23
1.3.1.2.4 Crc16ReturnErrorValue . . . 24
1.3.1.2.5 Crc32Mode . . . 24
1.3.1.2.6 Crc32P4Mode . . . 25
1.3.1.2.7 Crc32P4ReturnErrorValue . . . 25
1.3.1.2.8 Crc32ReturnErrorValue . . . 26
1.3.1.2.9 Crc64Mode . . . 26

Table of contents

1.3.1.2.10 Crc64ReturnErrorValue . . . 27
1.3.1.2.11 Crc8H2FMode . . . 28
1.3.1.2.12 Crc8H2FReturnErrorValue . . . 28
1.3.1.2.13 Crc8Mode . . . 29
1.3.1.2.14 Crc8ReturnErrorValue . . . 29
1.3.1.2.15 CrcDma16bitApi . . . 30
1.3.1.2.16 CrcDma32P4bitApi . . . 30
1.3.1.2.17 CrcDma32bitApi . . . 31
1.3.1.2.18 CrcDma8bitApi . . . 31
1.3.1.2.19 CrcRuntimeApiMode . . . 32
1.3.1.2.20 CrcSafetyEnable . . . 32
1.3.1.2.21 CrcVersionInfoApi . . . 33
1.3.1.3 Container: CrcPublishedInformation . . . 34
1.3.1.3.1 CrcInitialValue16 . . . 34
1.3.1.3.2 CrcInitialValue16ARC . . . 34
1.3.1.3.3 CrcInitialValue32 . . . 35
1.3.1.3.4 CrcInitialValue32P4 . . . 35
1.3.1.3.5 CrcInitialValue64 . . . 35
1.3.1.3.6 CrcInitialValue8 . . . 36
1.3.1.3.7 CrcInitialValue8H2F . . . 36
1.3.1.4 Container: CommonPublishedInformation . . . 37
1.3.1.4.1 ARPatchVersion . . . 37
1.3.1.4.2 ArMajorVersion . . . 37
1.3.1.4.3 ArMinorVersion . . . 38
1.3.1.4.4 ModuleId . . . 38
1.3.1.4.5 SwMajorVersion . . . 39
1.3.1.4.6 SwMinorVersion . . . 39
1.3.1.4.7 SwPatchVersion . . . 40
1.3.1.4.8 VendorId . . . 40
1.3.1.5 Container: CrcChannelConfig . . . 40
1.3.1.5.1 CrcChannelId . . . 41
1.3.1.5.2 CrcDmaChannel . . . 41
1.3.1.5.3 CrcErrorNotification . . . 42
1.3.1.5.4 CrcResultNotification . . . 42
1.3.2 Functions - Type definitions . . . 43
1.3.2.1 Crc_ChannelConfigType . . . 43
1.3.2.2 Crc_DmaReturnType . . . 44
1.3.2.3 Crc_ErrNotificationPtrType . . . 45
1.3.2.4 Crc_ResNotificationPtrType . . . 45
1.3.3 Functions - APIs . . . 45
1.3.3.1 Crc_CalculateCRC8 . . . 45
1.3.3.2 Crc_CalculateCRC8H2F . . . 46

Table of contents

1.3.3.3 Crc_CalculateCRC16 ... 47
1.3.3.4 Crc_CalculateCRC16ARC ... 48
1.3.3.5 Crc_CalculateCRC32 ... 49
1.3.3.6 Crc_CalculateCRC32P4 ... 50
1.3.3.7 Crc_CalculateCRC64 ... 51
1.3.3.8 Crc_DmaCalculateCRC8 ... 52
1.3.3.9 Crc_DmaCalculateCRC16 ... 54
1.3.3.10 Crc_DmaCalculateCRC32 ... 55
1.3.3.11 Crc_DmaCalculateCRC32P4 ... 57
1.3.3.12 Crc_GetVersionInfo ... 58
1.3.4 Notifications and Callbacks ... 59
1.3.4.1 Crc_DmaErrorIsr ... 59
1.3.4.2 Crc_DmaTransferIsr ... 60
1.3.5 Scheduled functions ... 61
1.3.6 Interrupt service routines ... 61
1.3.7 Callout ... 61
1.3.8 Errors Handling ... 61
1.3.9 Deviations and limitations ... 62
1.3.9.1 Deviations ... 62
1.3.9.1.1 Software specification deviations ... 62
1.3.9.1.2 AMDC Violations ... 62
1.3.9.1.3 VSMD Violations ... 62
1.3.9.2 Limitations ... 62
Revision history ... 64
Disclaimer ... 65

1 CRC driver

1.1 User information

1.1.1 Description

The CRC driver provides APIs to calculate the CRC for 8-bit, 16-bit, 32-bit and 64-bit polynomials, prescribed by AUTOSAR. The CRC driver performs CRC calculations by using hardware, runtime, table and DMA modes. It uses the Tricore hardware instructions and FCE hardware to perform the CRC calculation. The CRC driver is developed as a pre-compile variant.

1.1.2 Hardware-software mapping

This section describes the system view of the CRC driver and peripherals administered by it.

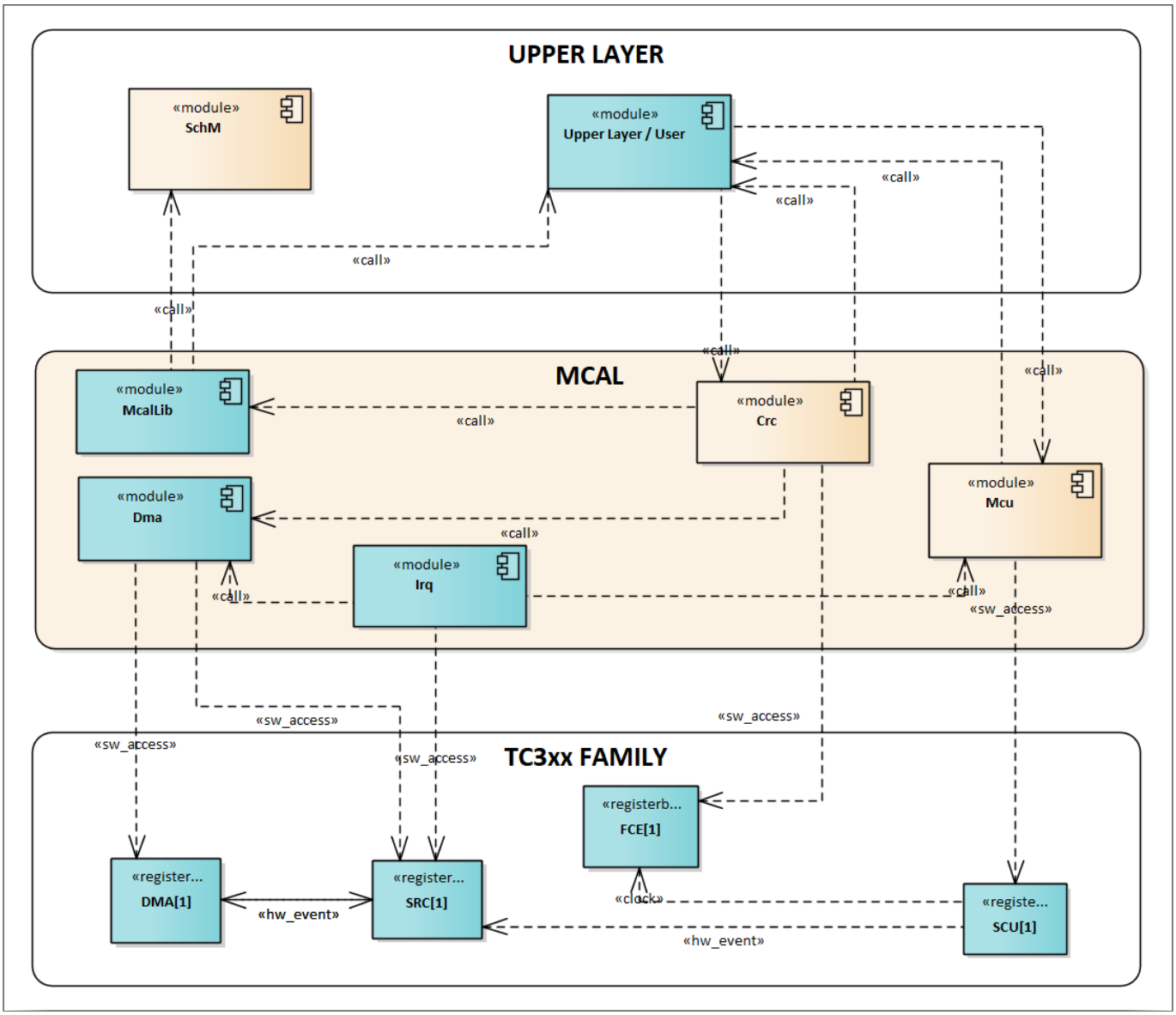

Figure 1 Mapping of hardware-software interfaces

1.1.2.1 FCE: Primary hardware peripheral

Hardware functional features

For DMA based operations, the CRC driver uses the FCE IP for calculating CRC.

CRC driver shall support the following polynomials provided by the FCE engine.

- CRC8
- CRC16
- CRC32
- CRC32P4

For DMA based operations, FCE IP is used. Each polynomial has a kernel to support the CRC calculation.

DMA channel will be exploited to transfer the input data to FCE register.

The unsupported feature of the FCE IP are:

- Automatic signature check
- Register protection and monitoring

Users of the hardware

The CRC driver exclusively utilizes the FCE IP for DMA operations.

Hardware diagnostic features

Not applicable.

Hardware events

Hardware events from the FCE IP like transient error detection or checksum failure are not used by the FCE driver.

1.1.2.2 SCU: dependent hardware peripheral

Hardware functional features

The CRC driver depends on the SCU IP for the clock functionality. The driver requires fSPB clock signal for functioning.

Users of the hardware

The SCU IP supplies clock for all the peripherals and the MCU driver is responsible for configuring the clock tree. To avoid conflicts due to simultaneous writes, update to all the ENDINIT protected registers is performed using the MCALLIB APIs.

Hardware diagnostic features

The SMU alarms configured for the SCU IP are not monitored by the CRC driver.

Hardware events

Hardware events from the SCU are not used by the CRC driver.

1.1.2.3 DMA: dependent hardware peripheral

Hardware functional features

The CRC driver depends on the DMA IP for transferring the data to the FCE IR register.

Each core is assigned with only one of the DMA channels which are linked to one of the FCE channels.

Users of the hardware

The DMA channels used for the CRC driver must be reserved and configured by the application through configurations provided by RM and DMA. The reserved DMA channels are exclusively used by CRC.

DMA based CRC APIs shall re-configure DMA channel settings during runtime based on the input parameters. The DMA source address shall be the address of the passed data pointer, DMA destination address shall be the address of the linked FCE channel's IR register and the DMA transfer width shall be the width of the polynomial being calculated.

Hardware diagnostic features

- SMU alarms configured for the DMA are not monitored by the CRC driver

Hardware events

- DMA error interrupt is enabled during data transmission and routed to the CRC driver by the DMA driver.
- DMA's successful transfer completion interrupts for the reserved channel is routed to the CRC driver by the DMA driver.

1.1.2.4 SRC: dependent hardware peripheral

Hardware functional features

The CRC driver depends on the interrupt router for raising an interrupt to the CPU or DMA based on data transfer which indicates the status of data transfer.

Users of the hardware

- The interrupt router is configured either by the IRQ driver or the user software.

Hardware diagnostic features

- The SMU alarms configured for the interrupt router are not monitored by the CRC driver.

Hardware events

- The interrupt events raised by the interrupt router are serviced by the CPU or DMA. The CRC driver provides interrupt handlers as software interfaces, which must be invoked from the ISR.

1.1.3 File structure

1.1.3.1 C file structure

This section provides details of the C files of the CRC driver.

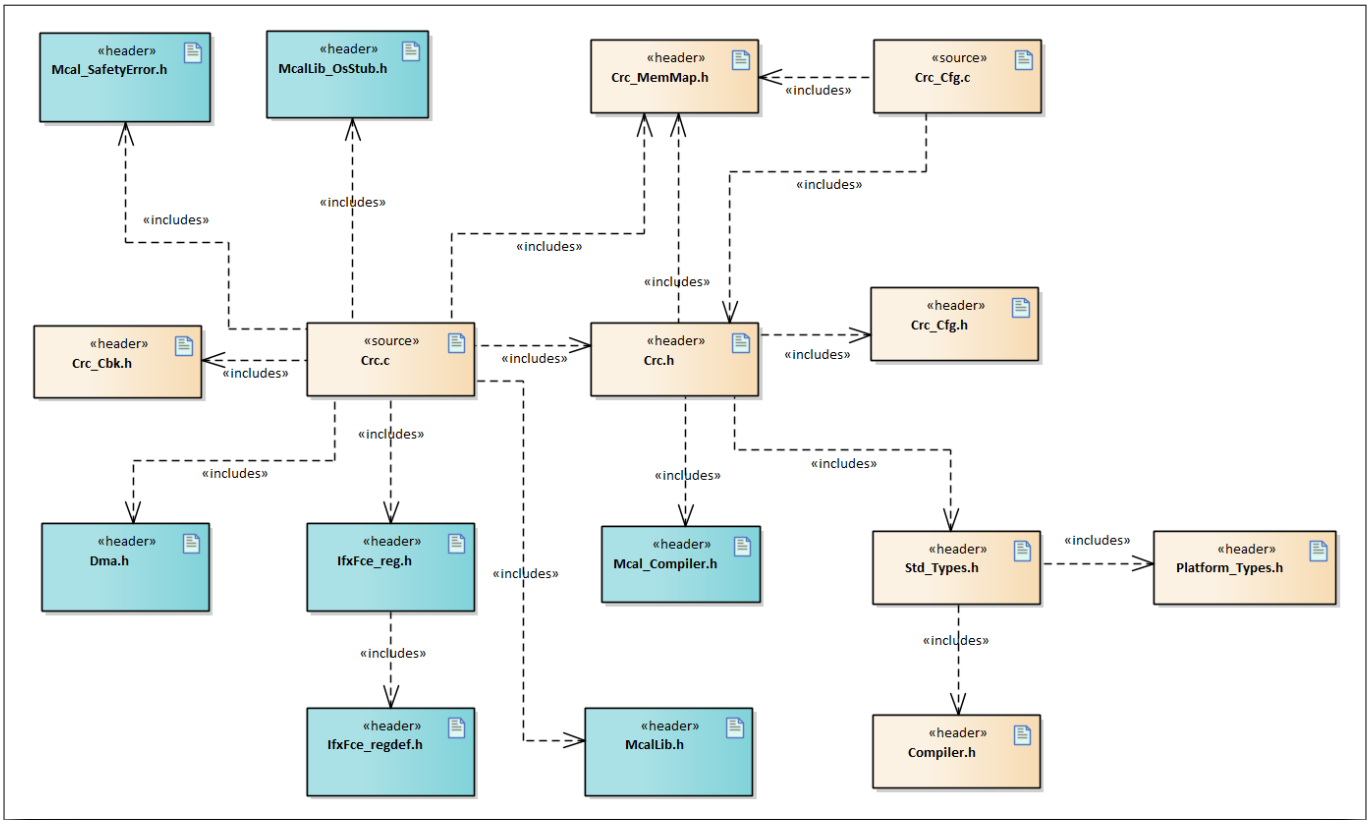

Figure 2 **Crc_C_File_Structure-1.png**

Table 2 **C file structure**

File name	Description
`Compiler.h`	Provides abstraction from compiler-specific keywords
`Crc.c`	Contains the implementation of the CRC feature.
`Crc.h`	Provides the functional prototypes and access to the CRC driver function. This file exports only the necessary interfaces for upper layer.
`Crc_Cbk.h`	Result notification ISR on the completion of CRC and error notification ISR are declared.
`Crc_Cfg.c`	Generated header file containing configuration data of the user.
`Crc_Cfg.h`	Provides the specific parameters of FCE.
`Crc_MemMap.h`	File (Static) containing the memory section definitions used by the CRC driver.
`Dma.h`	Header file (static) defining prototypes of data structures and APIs
`IfxFce_reg.h`	SFR header file for FCE
`IfxFce_regdef.h`	SFR header file for FCE
`McalLib.h`	Static header file defining prototypes of data structure and APIs exported by the MCALLIB.
`McalLib_OsStub.h`	McalLib_OsStub.h provides macros to support user mode of Tricore. This shall be included by other drivers to call OS APIs.
`Mcal_Compiler.h`	Header file providing abstraction for TriCore™-intrinsic instruction.
`Mcal_SafetyError.h`	Header file containing the prototype of the API for reporting safety-related errors

(table continues...)

Table 2 (continued) C file structure

File name	Description
`Platform_Types.h`	Platform-specific type declaration file as defined by AUTOSAR
`Std_Types.h`	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.

1.1.3.2 Code generator plugin files

This section provides details of the code generator plugin files of the CRC driver.

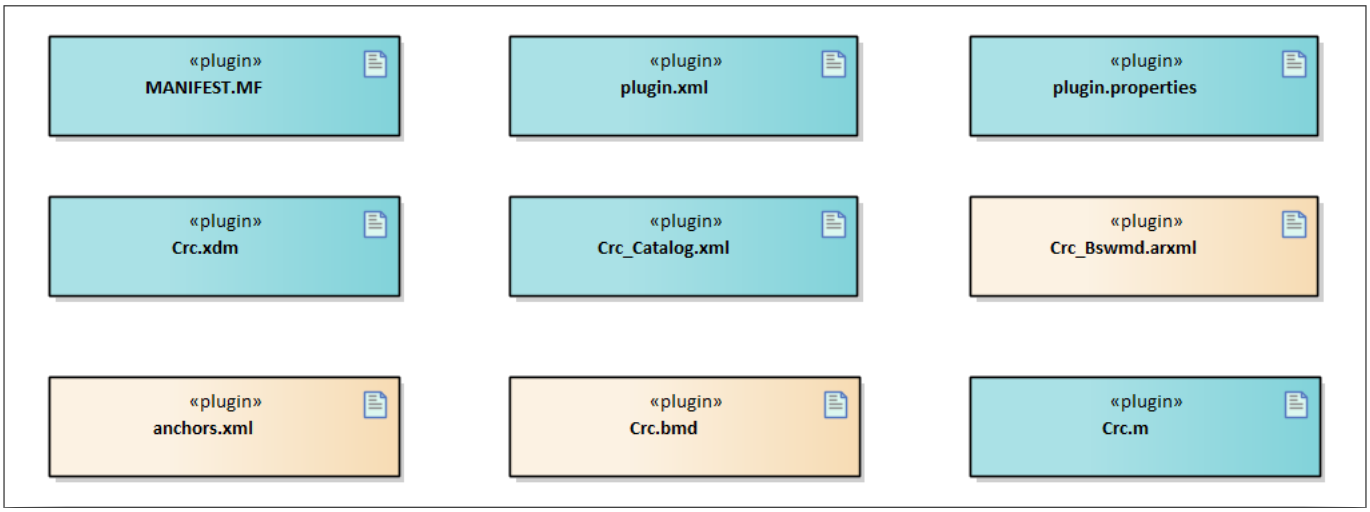

Figure 3 Crc_Code_Generator_Plugin_Files-1.png

Table 3 Code generator plugin files

File name	Description
`Crc.bmd`	Code template macro file for CRC driver.
`Crc.m`	Code template macro file for CRC driver.
`Crc.xdm`	Tresos format XML data model schema file.
`Crc_Bswmd.arxml`	AUTOSAR format module description file.
`Crc_Catalog.xml`	AUTOSAR format catalog file
`MANIFEST.MF`	Tresos plugin support file containing the metadata for CRC driver
`anchors.xml`	AUTOSAR format module description file
`plugin.properties`	Tresos plugin support file for the CRC driver
`plugin.xml`	Tresos plugin support file for the CRC driver

1.1.4 Integration hints

This section lists the key points that an integrator or user of the CRC driver must consider.

1.1.4.1 Integration with AUTOSAR stack

This section lists the modules, which are not part of the MCAL, but are required to integrate the CRC driver.

- **EcuM**

  EcuM module is not required for the integration of the CRC driver.

- **Memory mapping**

  Memory mapping is a concept from AUTOSAR that allows relocation of text, variables, constants and configuration data to user-specific memory regions. To achieve this, all the relocatable elements of the driver are encapsulated in different memory-section macros. These macros are defined in the `Crc_MemMap.h` file.

  The `Crc_MemMap.h` file is provided in the MCAL package as a stub code. The integrator must place appropriate compiler pragmas within the memory-section macros. The pragmas ensure that the elements

are relocated to the correct memory region. A sample implementation listing the memory-section macros is shown as follows.

```
#if defined CRC_START_SEC_CONST_ASIL_B_GLOBAL_8
/*****User pragmas here for PFlash*****/
 #undef CRC_START_SEC_CONST_ASIL_B_GLOBAL_8
 #undef MEMMAP_ERROR

#elif defined CRC_STOP_SEC_CONST_ASIL_B_GLOBAL_8
/*****User pragmas here for default section*****/
 #undef CRC_STOP_SEC_CONST_ASIL_B_GLOBAL_8
 #undef MEMMAP_ERROR

#elif defined CRC_START_SEC_CODE_ASIL_B_GLOBAL
/*****User pragmas here for PFlash*****/
 #undef CRC_START_SEC_CODE_ASIL_B_GLOBAL
 #undef MEMMAP_ERROR

#elif defined CRC_STOP_SEC_CODE_ASIL_B_GLOBAL
/*****User pragmas here for default section*****/
 #undef CRC_STOP_SEC_CODE_ASIL_B_GLOBAL
 #undef MEMMAP_ERROR

#elif defined CRC_START_SEC_VAR_INIT_ASIL_B_GLOBAL_8
/*****User pragmas here for non-cached LMU*****/
 #undef CRC_START_SEC_VAR_INIT_ASIL_B_GLOBAL_8
 #undef MEMMAP_ERROR

#elif defined CRC_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_8
/*****User pragmas here for default section*****/
 #undef CRC_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_8
 #undef MEMMAP_ERROR

#endif

#if defined MEMMAP_ERROR
#error "Crc_MemMap.h, wrong pragma command"
#endif
```

- **DET**

  DET module is not required for the integration of the CRC driver.
- **DEM**

  DEM module is not required for the integration of the CRC driver.
- **Safety errors**

  The CRC driver will report all the detected safety errors through the `Mcal_ReportSafetyError` API .

  The driver performs only detection and reporting of the safety errors. The handling of the reported errors shall be done by the user. The `Mcal_ReportSafetyError` API is provided in the files `Mcal_SafetyError.c` and `Mcal_SafetyError.h` as a stub code, and must be updated by the integrator to handle the reported errors.
- **Notifications and callbacks**

The CRC driver implements notification functions Crc_DmaTransferIsr and Crc_DmaErrorIsr for notifying the completion of successful data transfer and for notifying the RP error occurred during the data transfer respectively.

These notification functions can be configured by the user in the EB Tresos tool in the DMA configuration.

- **Operating system**

  Enabling and disabling of interrupts must be managed by the OS or application.

  OS files provided by MCAL package are only an example code and must be updated by the integrator with the actual OS files for the desired function.

1.1.4.2 Multicore and Resource Manager

The CRC driver supports execution of its APIs simultaneously from all CPU cores. Each core shall be assigned with only one of the DMA channels which shall be linked to one of the FCE channels in the CRC configuration. The user shall allocate for each core a unique DMA channel for the CRC module in DMA configuration and FCE channel in CRC configuration.

Image below shows the allocation of a DMA resource to a core.

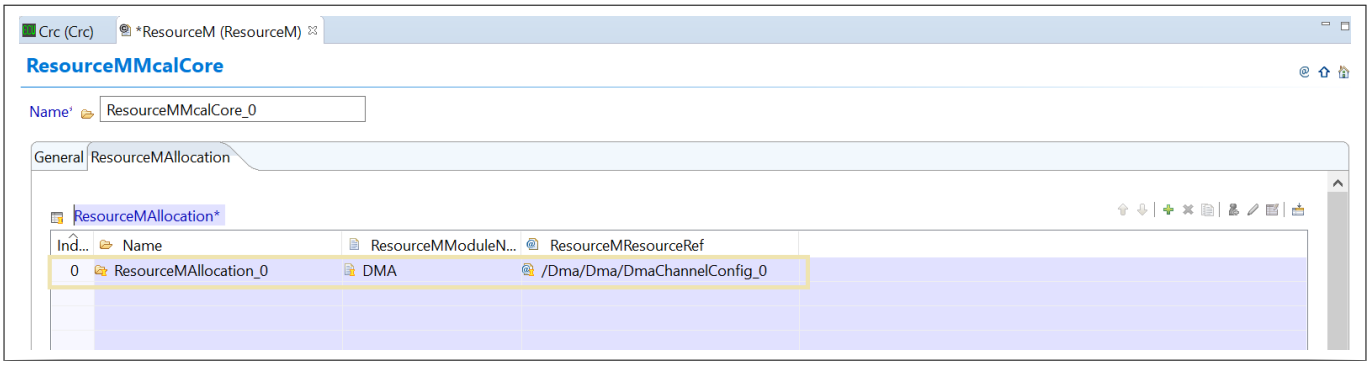

Figure 4 Configuration of DMA resource in the Resource Manager.

The following are the key points to be considered with respect to multicore in the driver:

Locating constants and variables to correct memory space should be done by the user. Memory sections are marked as GLOBAL (common to all cores). The following should be considered by the user to ensure better performance of the driver:

Code section:

The executable code of the CRC driver is placed under single MemMap section. It can be relocated to any PFlash region.

Data section:

The sections marked as global should be relocated to the non-cached LMU region.

Constants:

The sections marked as global should be relocated to the PFlash of the master core.

Note: Relocating code, data or constants to a distant memory region would impact execution timings.

1.1.4.3 MCU support

The CRC driver is dependent on MCU driver for clock configuration. The APIs of the CRC driver must be started only after completion of MCU initialization.

1.1.4.4 Port support

CRC driver does not use any services provided by the PORT driver.

1.1.4.5 DMA support

The CRC driver is dependent on the DMA IP for transferring the data to the FCE IR register. DMA channels should be configured when one/all of the DMA based CRC API is/are enables. The maximum number of required DMA channels by the CRC module depends on the number of available cores in the variant. Once the DMA channel is assigned to the CRC module, these channels must be reserved only for the CRC module and cannot be reused.

Configurations of the DMA module

1. In the DMA, in the General configuration section, enable DmaMESourceErrorInterrupt and DmaMEDestinationErrorInterrupt for receiving the move engine errors as show in Figure 5. Enable other configuration items as required by the application.

2. Add the DMA channel and configure the notification function pointers (Refer Figure 6). No other configurations are required in DMA. Transaction control set configurations for DMA are handled in CRC module and does not need any configuration in the DMA module.

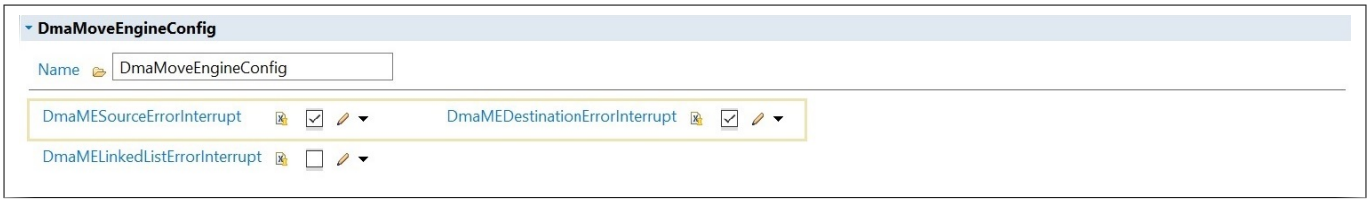

Figure 5 Enable Move engine parameter

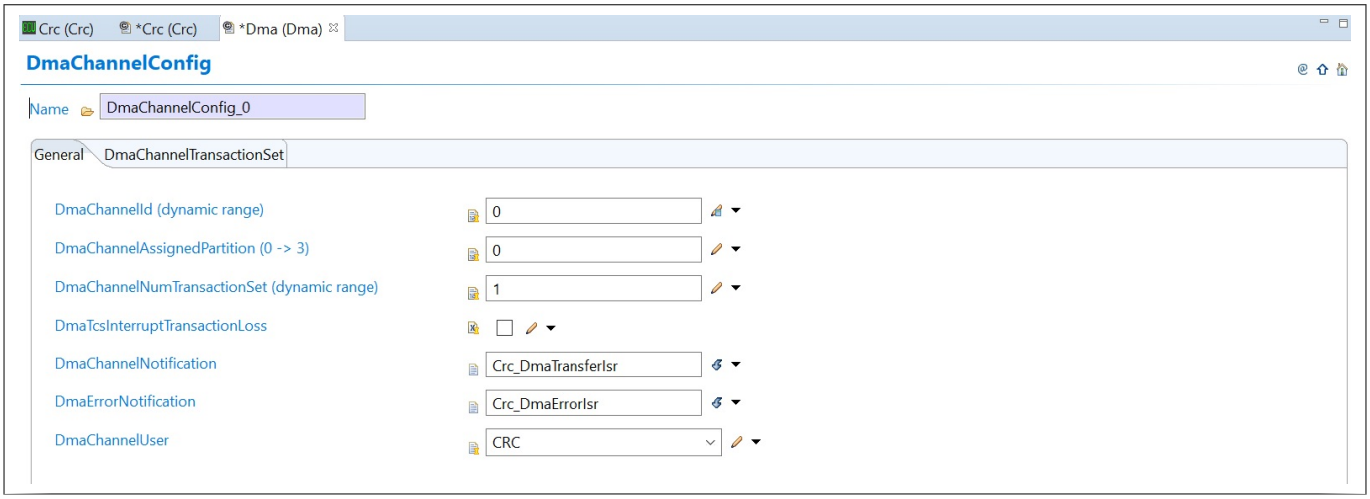

Figure 6 DMA Resource configuration

Note: If the `DmaChannelNotification` and `DmaErrorNotification` are not configured with the correct function pointers, this will result in a Codegen error.

ESM - DMA Error handling and Supervision

The following are the safety measure for the user, while using the DMA based CRC APIs:

In the event of a DMA error, the error call back function will be invoked for that CPU core. The user shall determine the DMA channel, since the DMA channel is assigned exclusively to a CPU core.

The sequence of CRC calculation would be as follows, considering the error scenario:

1. A CRC request using DMA is initiated by the user.
2. If a DMA error occurs, user shall reinitialize the DMA channel using Dma_ChInit API.
3. The same CRC request can be retriggered by the user.

1.1.4.6 Interrupt connections

The interrupt connections of the CRC driver are described in this section.

DMA would trigger a interrupt at the end of the successful channel transmission or the failed transmission. The notification and error function pointer configured during the DMA configuration will be invoked by the DMA module.

Priority should be set for the DMA channel assigned across the cores. The user shall ensure that the interrupt priority for Crc_DmaErrorIsr ISR is configured as higher than the Crc_DmaTransferIsr ISR.

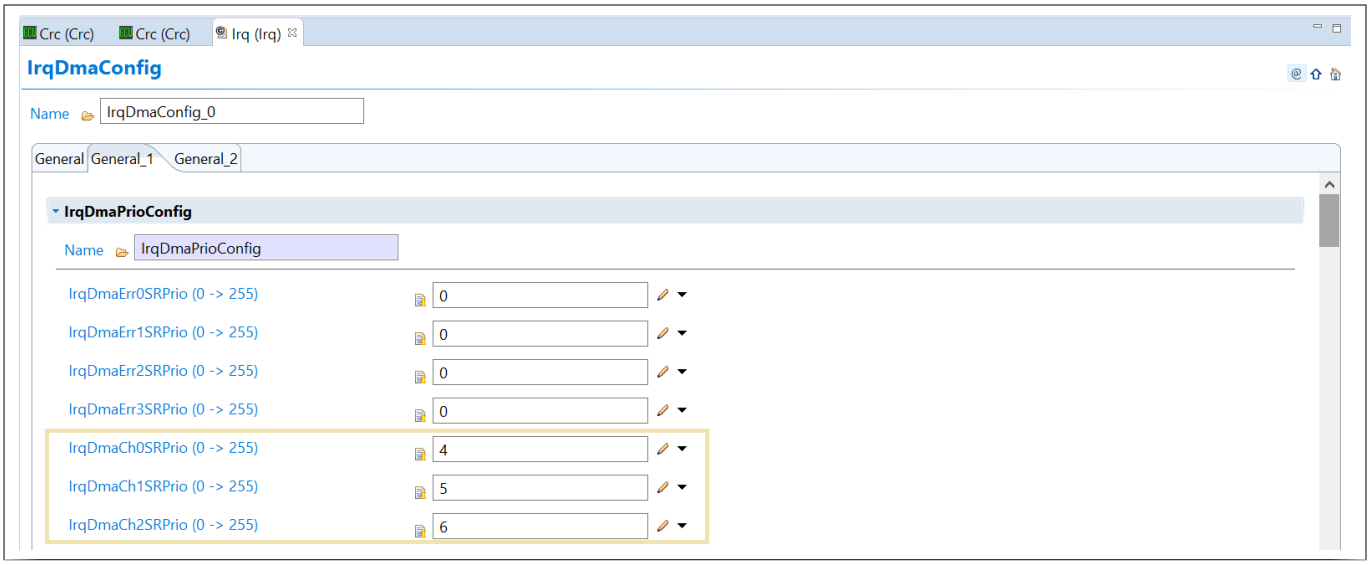

Figure 7 DMA Channel priority set in the IRQ configuration

1.1.4.7 Example usage

Configuration for AUTOSAR APIs

Each CRC polynomial can be set to any one of the given three modes with the exception of `Crc32P4Mode` and `Crc64Mode` which has only two modes. Refer the image below.

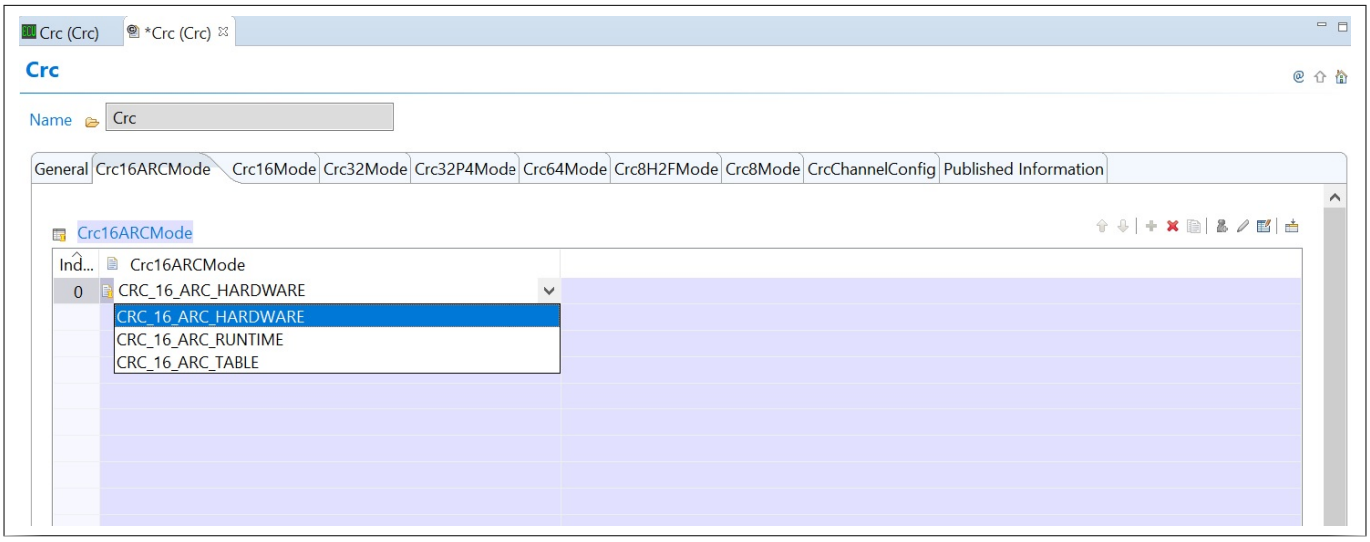

Figure 8 Mode selection for CRC calculation

For getting the expected error value on the failure of the successful CRC calculation make sure the `CrcSafetyEnable` is enabled and the return value is configured. If it is OFF, the error value parameters are disabled. The default value is zero.

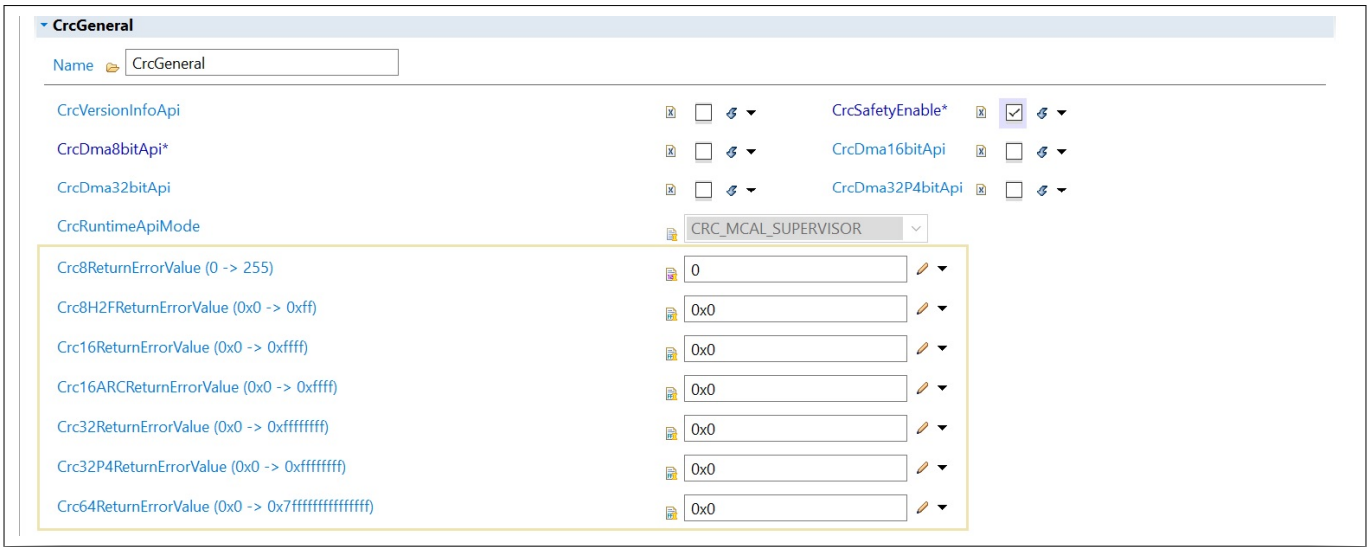

Figure 9 CrcSafetyEnable is enabled

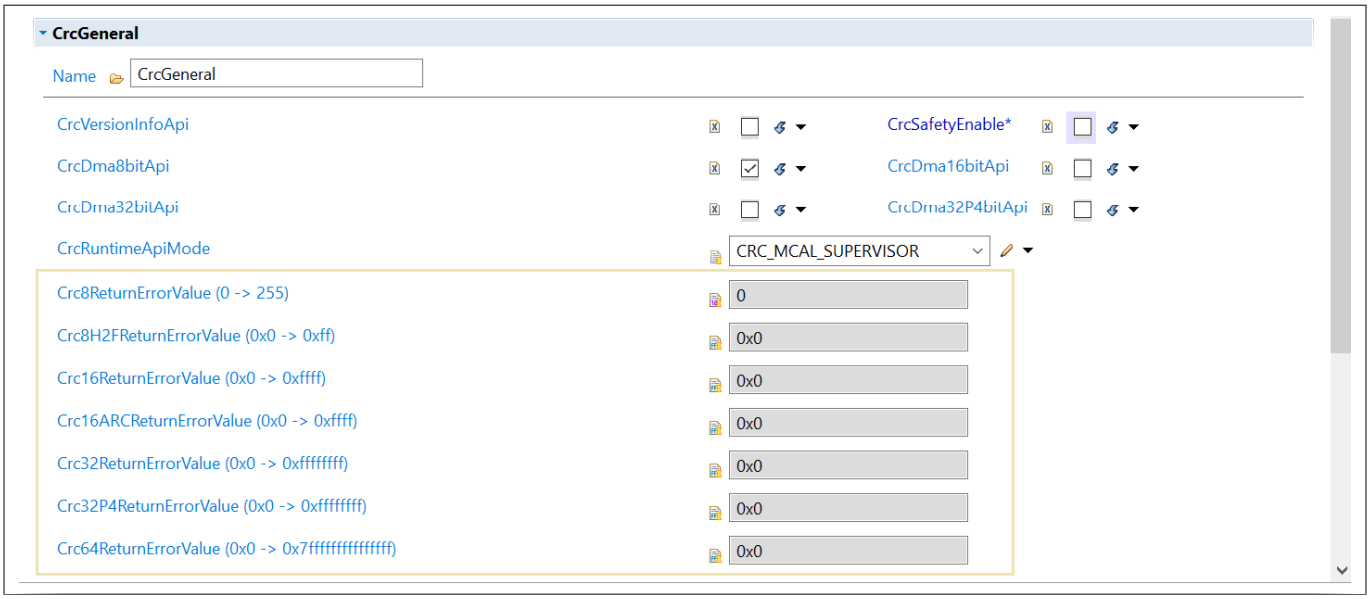

Figure 10 CrcSafetyEnable is disabled

Using the APIs

The code snippet shows an example of CRC calculation with 8-bit polynomial (0x1D).

Note: The type of each parameter varies for calculating CRC result for 8-bit, 16-bit, 32-bit, 64-bit and for the return value.

```
*Calling the CRC8 with polynomial = 0x1D */
Crc8Result = Crc_CalculateCRC8(Crc_DataPtr, Crc_Length, Crc_StartValue8,
Crc_IsFirstCall);
```

The usage of the parameters are as follows:

- `Crc_DataPtr` is the pointer to the start of the block.
- `Crc_Length` is the size of the block array.
- `Crc_StartValue8` is the start value to be used by algorithm.
- `Crc_IsFirstCall` selects the start value for CRC calculation. TRUE will select default initial value for the particular polynomial as start value for CRC calculation. FALSE will select the start value provided as argument as start value for CRC calculation.

Configuration for DMA based APIs

CRC calculation can also be obtained by invoking the DMA based CRC APIs. These APIs execute in the hardware mode where the FCE kernels are invoked for producing the desired result. DMA channels are exploited for transferring the user data input to the FCE engine for the CRC calculation.

The FCE engine has the support for the following polynomials:

- CRC8
- CRC16
- CRC32
- CRC32P4

The DMA based APIs mode can be enabled in the Tresos configuration. Following are the pre-requisite for CRC Calculation using DMA based APIs:

Note: Refer to integration hints of CRC driver and add all the dependent modules required for the configuration.

1. Enable the DMA based API:

Enable the switch of the required bit calculation in the CRC module configuration under General configuration container.

- CrcDma8bitApi is for enabling CRC8
- CrcDma16bitApi is for enabling CRC16
- CrcDma32bitApi is for enabling CRC32
- CrcDma32P4bitApi is for enabling CRC32P4

The image below shows the API for CRC8 polynomial is enabled by enabling the CrcDma8bitApi.

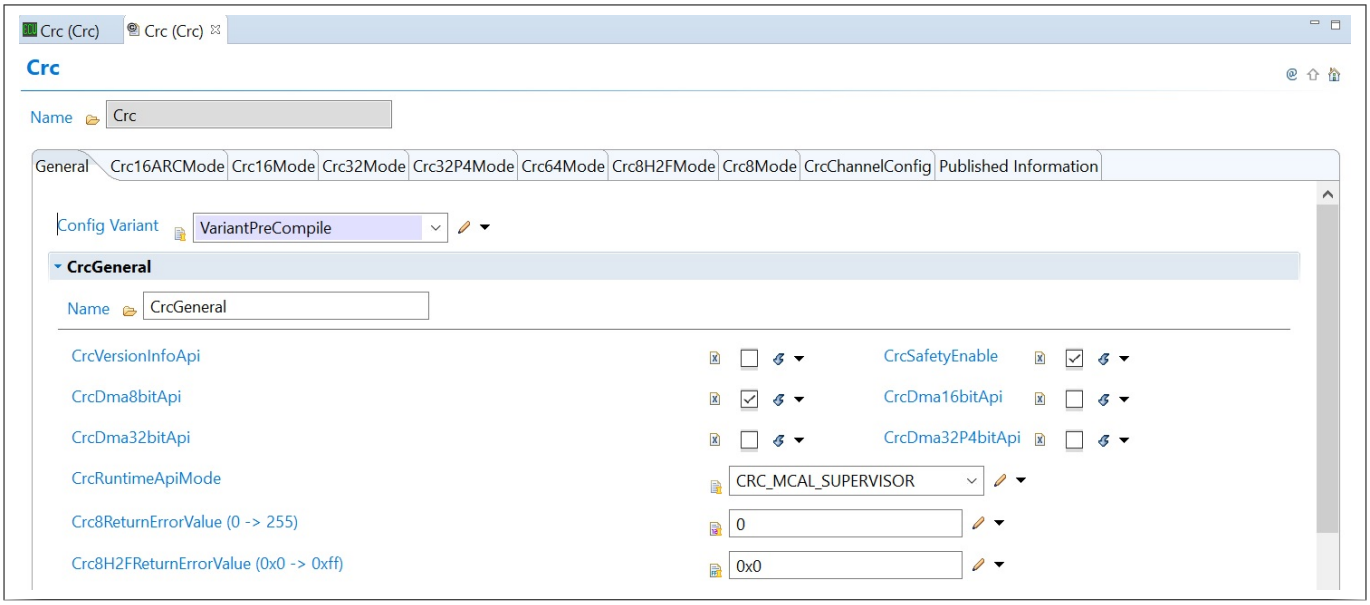

Figure 11 CrcDma8bitApi is enabled

2. Configure the DMA channel in the DMA configuration for the CRC module.

Refer the DMA support in the integration hint to configure the DMA channel for the CRC module.

3. Set the priority of the DMA channel allocated to the CRC in the IRQ configuration.

Refer the Interrupt support in the integration hint to configure the priority of the DMA channels.

4. Allocate the assigned DMA channels across the cores.

Below points shall be followed while assigning the DMA resources to the core in the Resource Manager.

- Assign only those channel in the Resource Manager which are allocated to the CRC module.
- Each core should have only one DMA channel assigned to it.

Refer the Multicore and Resource Manager in the integration hint to allocated the DMA resource in the Resource Manager.

5. Configure the FCE resource in the CRC configuration and assign the DMA channel to the FCE channel.

Configure the FCE resource under the container CrcChannelConfig as shown in the figure below

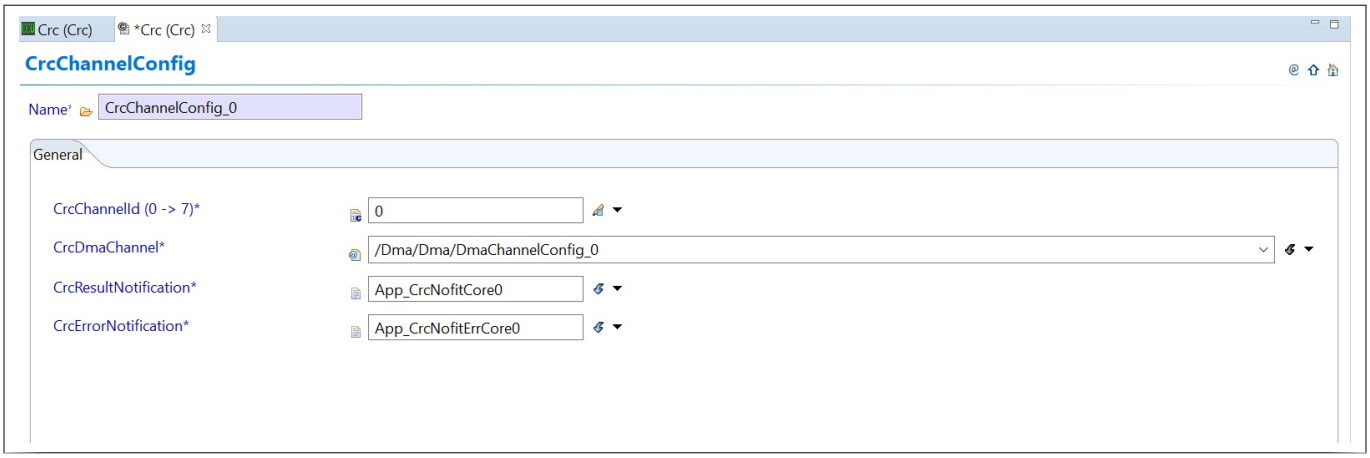

Figure 12 FCE resource allocation in the CRC configuration

Configure the `CrcResultNotification` and `CrcErrorNotification` with the user defined callback function pointer for notification of the available result and for the notification of any error occurred during the CRC calculation.

Note: If the `CrcResultNotification` and `CrcErrorNotification` are not configured with the correct function pointers, this will result in a Codegen error.

When none of the DMA based API is enabled, the CrcChannelConfig container will be disabled for editing. Make sure the container is empty when it disabled. If the CrcChannelConfig container the FCE resource and all the DMA based API is disabled, this will result in a Codegen error as this is a wastage of the DMA resource, hence unallocate all the DMA resource allocated to the CRC module in the DMA module and remove them from the CrcChannelConfig container.

The below image shows CrcChannelConfig configuration, when none of the DMA based API is enabled.

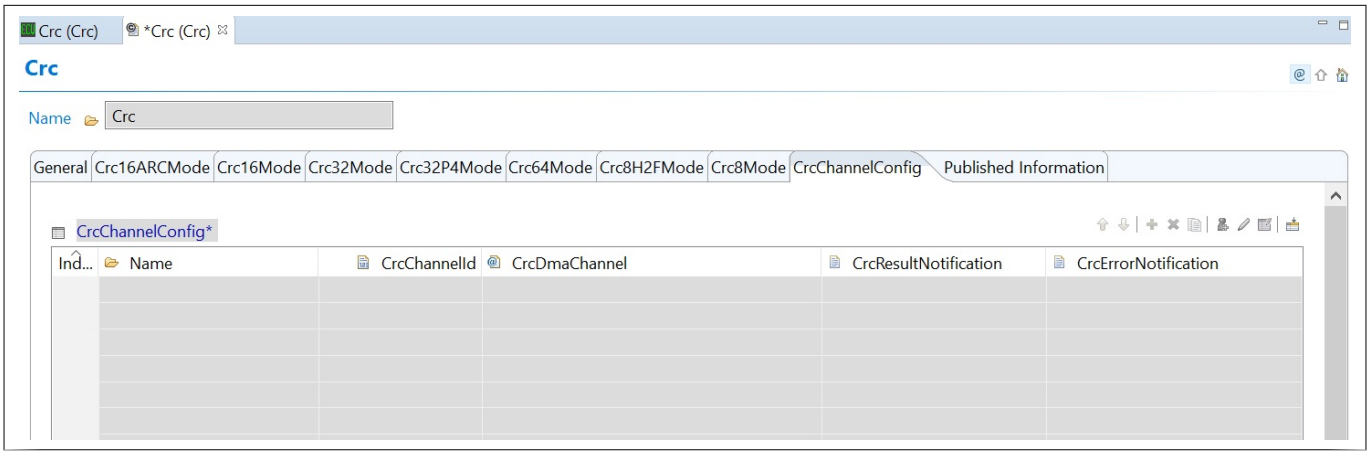

Figure 13 Empty CrcChannelConfig container

6. Once all the configurations are done successfully in the Tresos, Following the below sequence in the application code before invoking the DMA based APIs:

1. Initialize the MCU and clock using the Mcu_Init API.
2. Initialize the DMA driver using the Dma_init API.

3. Initialize the IRQ for dependent modules using the IrqDma_Init API.

```
#if (CRC_DMA_MAX_CHANNELS > 0U)
extern const Mcu_ConfigType Mcu_Config;
extern const Dma_ConfigType Dma_Config;
#endif
void core0_main (void)
{
/*your code */
#if (CRC_DMA_MAX_CHANNELS > 0U)
/*MCU initialization*/
Mcu_Init(&Mcu_Config);
Mcu_InitClock(0U);
Mcu_DistributePllClock();
/*IRQ initialization*/
IrqDma_Init();
/*DMA initialization*/
Dma_Init(&Dma_Config);
/*Enable the interrupt*/
SRC_DMACH0.U |= 0x400U;
#endif
/*your code */
}
```

1.1.5 Key architectural considerations

1.1.5.1 AUTOSAR modes of operation

Following polynomials shall support only Runtime and Table method since there is no support for equivalent CPU instruction:

- CRC32P4
- CRC64

CRC module supports Hardware, Table and Runtime mode for the following polynomial:

- CRC8
- CRC8H2F
- CRC16
- CRC16ARC
- CRC32

The mode of the above polynomials shall always use the hardware mode internally (CPU instruction). The above modes of operation simplifies the design and ensures the efficient use of hardware access.

1.1.5.2 CPU CRCN instruction

The following polynomials use CPU instruction to calculate CRC:

- CRC8
- CRC8H2F
- CRC16

- CRC16ARC
- CRC32

CRC8, CRC8H2F, CRC16, and CRC16ARC use CRCN CPU instruction and CRC32 uses CRC32.B instruction for CRC calculation.

CPU instructions do not have support for CRC32P4 polynomial.

1.1.5.3 DMA based operation

FCE engine supports CRC calculation for following polynomials:

- CRC16
- CRC32
- CRC32P4

Each core is assigned with only one of the DMA channels which are linked to one of the FCE channels. DMA the channel is exploited to transfer data to the linked FCE channel's input register.

The user shall allocate for each core a unique DMA channel for the CRC module in DMA configuration and FCE channel in CRC configuration.

In DMA configuration apart from the allocation of DMA channels to the CRC module and the callback notifications (Crc_DmaTransferIsr and Crc_DmaErrorIsr), the rest of the DMA channel configurations are ignored. Callback notification to the CRC user is configured in the CRC configuration.

DMA based CRC APIs shall re-configure DMA channel settings during runtime based on the input parameters.

The DMA source address shall be the address of the passed data pointer, DMA destination address shall be the address of the linked FCE channel's IR register and the DMA transfer width shall be the width of the polynomial being calculated.

1.2 Assumptions of Use (AoU)

The AoU for the CRC driver are as follows:

- **Error value return**

When the safety switch is ON, if the result returned by the AUTOSAR API is the configured error value and no safety error is reported, then the result is valid.
[cover parentID CRC={6AADA8A0-78EB-4432-A6AD-4862A186D2AD}]

- **FCE register protection**

User shall not modify any FCE-related register.
[cover parentID CRC={68DA99F3-38B9-47b7-BF82-BC9F32BB485B}]

- **DMA initialization for using the DMA feature**

The user shall ensure that the DMA module is initialized before using the DMA based APIs.
[cover parentID CRC={2D019E9F-173A-4e28-A7AA-993138CCBB35}]

1.3 Reference information

1.3.1 Configuration interfaces

Supported configuration variant: Pre-Compile

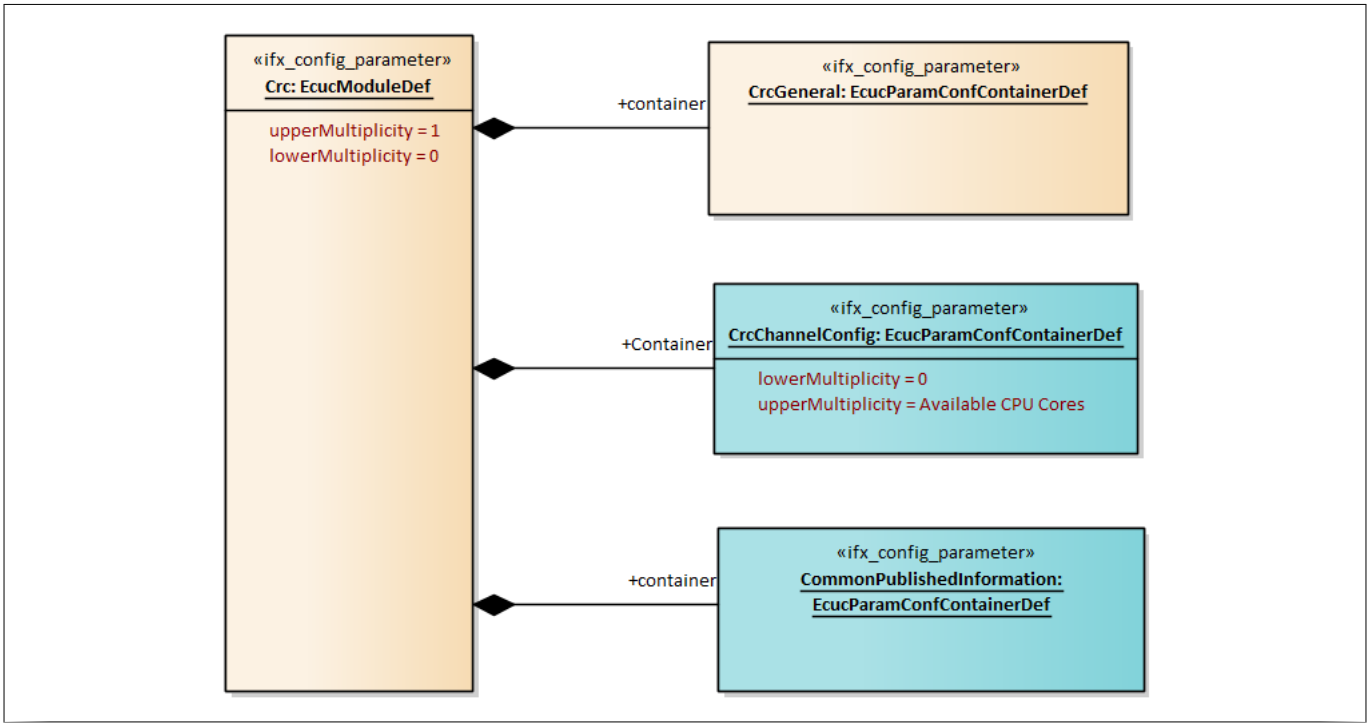

Figure 14 Container hierarchy along with their configuration parameters

1.3.1.1 Container: Crc

Configuration of the CRC driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.2 Container: CrcGeneral

General configuration of the CRC driver

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.2.1 Crc16ARCMode

Table 4 Specification for Crc16ARCMode

Name	`Crc16ARCMode`		
Description	Switch to select one of the available CRC 16-bit (0x8005h) calculation methods.		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef

(table continues...)

Table 4 (continued) Specification for Crc16ARCMode

Range	CRC_16_ARC_HARDWARE: Hardware based CRC16ARC calculation. CRC_16_ARC_RUNTIME: Runtime based CRC16ARC calculation. CRC_16_ARC_TABLE: Table based CRC16ARC calculation.		
Default value	CRC_16_ARC_HARDWARE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	IFX FOR AS4.2.2 VARIANT AND AUTOSAR_ECUC FOR AS4.4.0 VARIANT	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.2 Crc16ARCReturnErrorValue

Table 5 Specification for Crc16ARCReturnErrorValue

Name	`Crc16ARCReturnErrorValue`		
Description	The error value to be returned by the AUTOSAR API Crc_Calculate16ARC for the polynomial CRC16ARC instead of CRC value if safety check is enabled and in case of an incorrect input parameter. The default value of the parameter is the minimum value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CrcSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.3 Crc16Mode

Table 6 Specification for Crc16Mode

Name	`Crc16Mode`		
Description	Switch to select one of the available CRC 16-bit (CCITT) calculation methods.		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef

(table continues...)

Table 6 (continued) Specification for Crc16Mode

Range	CRC_16_HARDWARE: Hardware based CRC16 calculation. CRC_16_RUNTIME: Runtime based CRC16 calculation. CRC_16_TABLE: Table based CRC16 calculation.		
Default value	CRC_16_HARDWARE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.4 Crc16ReturnErrorValue

Table 7 Specification for Crc16ReturnErrorValue

Name	`Crc16ReturnErrorValue`		
Description	The error value to be returned by the AUTOSAR API Crc_CalculateCRC16 for the polynomial CRC16 instead of CRC value if safety check is enabled and in case of an incorrect input parameter. The default value of the parameter is the minimum value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CrcSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.5 Crc32Mode

Table 8 Specification for Crc32Mode

Name	`Crc32Mode`		
Description	Switch to select one of the available CRC 32-bit (IEEE-802.3 CRC32 ethernet standard) calculation methods.		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef

(table continues...)

Table 8 (continued) Specification for Crc32Mode			
Range	CRC_32_HARDWARE: Hardware based CRC32 calculation. CRC_32_RUNTIME: Runtime based CRC32 calculation. CRC_32_TABLE: Table based CRC32 calculation.		
Default value	CRC_32_HARDWARE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.6 Crc32P4Mode

Table 9 Specification for Crc32P4Mode			
Name	`Crc32P4Mode`		
Description	Switch to select one of the available CRC 32-bit E2E profile 4 calculation methods. *Note: CRC32P4 does not provide support for hardware mode (CRC_32P4_HARDWARE) in the configuration parameter. CRC for CRC32P4 in hardware mode is supported through DMA bases CRC API.*		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef
Range	CRC_32P4_RUNTIME: Runtime based CRC32P4 calculation. CRC_32P4_TABLE: Table based CRC32P4 calculation.		
Default value	CRC_32P4_TABLE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.7 Crc32P4ReturnErrorValue

Table 10 Specification for Crc32P4ReturnErrorValue	
Name	`Crc32P4ReturnErrorValue`

(table continues...)

Table 10 (continued) Specification for Crc32P4ReturnErrorValue

Description	The error value to be returned by the AUTOSAR API Crc_CalculateCRC32P4 for the polynomial CRC32P4 instead of CRC value if safety check is enabled and in case of an incorrect input parameter. The default value of the parameter is the minimum value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CrcSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.8 Crc32ReturnErrorValue

Table 11 Specification for Crc32ReturnErrorValue

Name	`Crc32ReturnErrorValue`		
Description	The error value to be returned by the AUTOSAR API Crc_CalculateCRC32 for the polynomial CRC32 instead of CRC value if safety check is enabled and in case of an incorrect input parameter. The default value of the parameter is the minimum value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CrcSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.9 Crc64Mode

Table 12 Specification for Crc64Mode

Name	`Crc64Mode`

(table continues...)

Table 12 (continued) Specification for Crc64Mode

Description	Switch to select one of the available CRC 64-bit calculation methods. *Note: CRC64 does not support CRC calculation in the hardware mode (CRC_64_HARDWARE) as the current FCE engine does not provide the kernel for CRC64.*		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef
Range	CRC_64_RUNTIME: Runtime based CRC64 calculation. CRC_64_TABLE: Table based CRC64 calculation.		
Default value	CRC_64_TABLE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	IFX FOR AS4.2.2 VARIANT AND AUTOSAR_ECUC FOR AS4.4.0 VARIANT	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.10 Crc64ReturnErrorValue

Table 13 Specification for Crc64ReturnErrorValue

Name	`Crc64ReturnErrorValue`		
Description	The error value to be returned by the AUTOSAR API Crc_CalculateCRC64 for the polynomial CRC64 instead of CRC value if safety check is enabled and in case of an incorrect input parameter. The default value of the parameter is the minimum value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 9223372036854775807		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CrcSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.11 Crc8H2FMode

Table 14 Specification for Crc8H2FMode

Name	`Crc8H2FMode`		
Description	Switch to select one of the available CRC 8-bit (2Fh polynomial) calculation methods.		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef
Range	CRC_8H2F_HARDWARE: Hardware based CRC8H2F calculation. CRC_8H2F_RUNTIME: Runtime based CRC8H2F calculation. CRC_8H2F_TABLE: Table based CRC8H2F calculation.		
Default value	CRC_8H2F_HARDWARE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.12 Crc8H2FReturnErrorValue

Table 15 Specification for Crc8H2FReturnErrorValue

Name	`Crc8H2FReturnErrorValue`		
Description	The error value to be returned by the AUTOSAR API Crc_CalculateCRC8H2F for the polynomial CRC8H2F instead of CRC value if safety check is enabled and in case of an incorrect input parameter. The default value of the parameter is the minimum value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CrcSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.13 Crc8Mode

Table 16 Specification for Crc8Mode

Name	`Crc8Mode`		
Description	Switch to select one of the available CRC 8-bit (SAE J1850) calculation methods.		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef
Range	CRC_8_HARDWARE: Hardware based CRC8 calculation. CRC_8_RUNTIME: Runtime based CRC8 calculation. CRC_8_TABLE: Table based CRC8 calculation.		
Default value	CRC_8_HARDWARE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.14 Crc8ReturnErrorValue

Table 17 Specification for Crc8ReturnErrorValue

Name	`Crc8ReturnErrorValue`		
Description	The error value to be returned by the AUTOSAR API Crc_CalculateCRC8 for polynomial the CRC8 instead of CRC value if safety check is enabled and in case of an incorrect input parameter. The default value of the parameter is the minimum value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CrcSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.15 CrcDma16bitApi

Table 18 Specification for CrcDma16bitApi

Name	`CrcDma16bitApi`		
Description	Pre-processor switch to enable / disable CrcDma16bitApi for the polynomial CRC16. True: Crc_DmaCalculateCRC16 API enable. False: Crc_DmaCalculateCRC16 API disable. The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.16 CrcDma32P4bitApi

Table 19 Specification for CrcDma32P4bitApi

Name	`CrcDma32P4bitApi`		
Description	Pre-processor switch to enable / disable CrcDma32P4bitApi for the polynomial CRC32P4. True: Crc_DmaCalculateCRC32P4 API enable. False: Crc_DmaCalculateCRC32P4 API disable. The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL

(table continues...)

Table 19 (continued) Specification for CrcDma32P4bitApi	
Dependency	-
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.2.17 CrcDma32bitApi

Table 20 Specification for CrcDma32bitApi

Name	`CrcDma32bitApi`		
Description	Pre-processor switch to enable / disable CrcDma32bitApi for the polynomial CRC32. True: Crc_DmaCalculateCRC32 API enable. False: Crc_DmaCalculateCRC32 API disable. The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.18 CrcDma8bitApi

Table 21 Specification for CrcDma8bitApi

Name	`CrcDma8bitApi`		
Description	Pre-processor switch to enable / disable CrcDma8bitApi for the polynomial CRC8. True: Crc_DmaCalculateCRC8 API enable. False: Crc_DmaCalculateCRC8 API disable. The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		

(table continues...)

Table 21 (continued) Specification for CrcDma8bitApi

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.19 CrcRuntimeApiMode

Table 22 Specification for CrcRuntimeApiMode

Name	`CrcRuntimeApiMode`		
Description	This configuration parameter gives the mode in which the runtime APIs of the CRC driver will be used. Since the CRC driver accesses the SFRs, it is more efficient to operate the CRC driver in supervisor mode. Hence, the default mode of operation is the supervisor mode. *Note: CrcRuntimeApiMode will be available only when at least one of the DMA based APIs is enabled, otherwise the parameter is editable false.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CRC_MCAL_SUPERVISOR: Operating mode used is Supervisor. CRC_MCAL_USER1: Operating mode used is USER1.		
Default value	MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.20 CrcSafetyEnable

Table 23 Specification for CrcSafetyEnable

Name	`CrcSafetyEnable`

(table continues...)

Table 23 (continued) Specification for CrcSafetyEnable

Description	Switch to enable/disable the safety check and reporting. TRUE: enables safety check and reporting FALSE: disables safety check and reporting The detection of safety related errors is enabled by default to ensure that safety issues are addressed during the product lifecycle.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.21 CrcVersionInfoApi

Table 24 Specification for CrcVersionInfoApi

Name	`CrcVersionInfoApi`		
Description	Pre-processor switch to enable / disable the API to read out the driver version information. True: Version info API enabled. False: Version info API disabled. The optional APIs are disabled by default to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3 Container: CrcPublishedInformation

Post-Build Variant Multiplicity: -
Multiplicity Configuration Class: -

1.3.1.3.1 CrcInitialValue16

Table 25 Specification for CrcInitialValue16

Name	`CrcInitialValue16`		
Description	Initial Value for this CRC calculation as specified by Autosar.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	65535 - 65535		
Default value	65535		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.2 CrcInitialValue16ARC

Table 26 Specification for CrcInitialValue16ARC

Name	`CrcInitialValue16ARC`		
Description	Initial Value for this CRC calculation as specified by Autosar.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 0		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.3 CrcInitialValue32

Table 27 Specification for CrcInitialValue32

Name	`CrcInitialValue32`		
Description	Initial Value for this CRC calculation as specified by Autosar.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	4294967295 - 4294967295		
Default value	4294967295		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.4 CrcInitialValue32P4

Table 28 Specification for CrcInitialValue32P4

Name	`CrcInitialValue32P4`		
Description	Initial Value for this CRC calculation as specified by Autosar.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	4294967295 - 4294967295		
Default value	4294967295		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.5 CrcInitialValue64

Table 29 Specification for CrcInitialValue64

Name	`CrcInitialValue64`		
Description	Initial Value for this CRC calculation as specified by Autosar.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

(table continues...)

Table 29 (continued) Specification for CrcInitialValue64			
Range	18446744073709551615 - 18446744073709551615		
Default value	18446744073709551615		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.6 CrcInitialValue8

Table 30 Specification for CrcInitialValue8			
Name	`CrcInitialValue8`		
Description	Initial Value for this CRC calculation as specified by Autosar.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	255 - 255		
Default value	255		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.7 CrcInitialValue8H2F

Table 31 Specification for CrcInitialValue8H2F			
Name	`CrcInitialValue8H2F`		
Description	Initial Value for this CRC calculation as specified by Autosar.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	255 - 255		
Default value	255		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 31 (continued) Specification for CrcInitialValue8H2F

Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4 Container: CommonPublishedInformation

Post-Build Variant Multiplicity: -
Multiplicity Configuration Class: -

1.3.1.4.1 ARPatchVersion

Table 32 Specification for ARPatchVersion

Name	`ARPatchVersion`		
Description	Patch version number of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the AUTOSAR version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.2 ArMajorVersion

Table 33 Specification for ArMajorVersion

Name	`ArMajorVersion`		
Description	Major version number of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the AUTOSAR version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 33 (continued) Specification for ArMajorVersion

Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.3 ArMinorVersion

Table 34 Specification for ArMinorVersion

Name	`ArMinorVersion`		
Description	Minor version number of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the AUTOSAR version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.4 ModuleId

Table 35 Specification for ModuleId

Name	`ModuleId`		
Description	This macro gives the Crc driver module ID as described by AUTOSAR.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	201 (0xC9)		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

(table continues...)

Table 35	(continued) Specification for ModuleId
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.4.5 SwMajorVersion

Table 36 Specification for SwMajorVersion

Name	`SwMajorVersion`		
Description	Major version number of the driver.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.6 SwMinorVersion

Table 37 Specification for SwMinorVersion

Name	`SwMinorVersion`		
Description	Minor version number of the driver.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.7 SwPatchVersion

Table 38 Specification for SwPatchVersion

Name	`SwPatchVersion`		
Description	Patch level version number of the driver. The patch version is incremented if the driver is still upwards and downwards compatible (for example, bug fix).		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.8 VendorId

Table 39 Specification for VendorId

Name	`VendorId`		
Description	Vendor ID of Infineon Technologies.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5 Container: CrcChannelConfig

This container has the FCE channel configuration.

Lower multiplicity is 0 and upper multiplicity depends on the number of cores available.

Note 1: The CrcChannelConfig container is available for update only when at least one of the DMA based APIs is enabled, otherwise the container is editable false.

Note 2: When the DMA based API is enabled and none of the channel is configured in the container, code gen error will be generated.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.5.1 CrcChannelId

Table 40 Specification for CrcChannelId

Name	`CrcChannelId`		
Description	Select one of the available FCE channels.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.2 CrcDmaChannel

Table 41 Specification for CrcDmaChannel

Name	`CrcDmaChannel`		
Description	Select the DMA resource allocated to the CRC module in the DMA configuration.		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: DmaChannelConfig		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.3 CrcErrorNotification

Table 42 Specification for CrcErrorNotification

Name	`CrcErrorNotification`		
Description	The CrcErrorNotification is used by the CRC driver to invoke user-defined function for error notification purpose. The parameter can be a name or the address (numeric value) of the notification function. Pointer to notification function should be of the type: void Function_Name (void) *Note1: By default, the notification parameter will be NULL.* *Note2: The CRC driver does not validate the configured function name or address for correctness and the responsibility falls on the user.*		
Multiplicity	1..1	**Type**	EcucFunctionNameDef
Range	String		
Default value	NULL_PTR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.4 CrcResultNotification

Table 43 Specification for CrcResultNotification

Name	`CrcResultNotification`

(table continues...)

Table 43 (continued) Specification for CrcResultNotification

Description	The CrcResultNotification is used by the CRC driver to invoke the user-defined function for result notification purpose. The parameter can be a name or the address (numeric value) of the notification function. Pointer to notification function should be of the type: void Function_Name (uint32 CrcResult) *Note1: By default, the notification parameter will be NULL.* *Note2: The CRC driver does not validate the configured function name or address for the correctness and the responsibility falls on the user.* *Note3: For 8-bit and 16-bit CRC calculation, only LSB should be read from the result passed as a parameter in the user defined notification function (value in rest of the bits are undefined).*		
Multiplicity	1..1	**Type**	EcucFunctionNameDef
Range	String		
Default value	NULL_PTR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.2 Functions - Type definitions

This section lists all the data type of the CRC driver.

1.3.2.1 Crc_ChannelConfigType

Table 44 Specification for Crc_ChannelConfigType

Syntax	`Crc_ChannelConfigType`	
Type	Structure	
File	`Crc.h`	
Range	uint8 Dma_Channel	DMA channel number in the range[0-127].
	Crc_ErrNotificationPtrType ErrNotificationPtr	Holds the address of the error notification callback function configured by the user.

(table continues...)

Table 44 (continued) Specification for Crc_ChannelConfigType

	uint8 Fce_Channel	FCE channel number in the range[0-7].
	Crc_ResNotificationPtrType ResNotificationPtr	Holds the address of the Result notification callback function configured by the user.
Description	This structure holds the configuration of the FCE resources allocated in CRC configuration.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.2 Crc_DmaReturnType

Table 45 Specification for Crc_DmaReturnType

Syntax	`Crc_DmaReturnType`	
Type	Enumeration	
File	`Crc.h`	
Range	0x01 - CRC_CHANNEL_BUSY	Return value from the DMA based APIs if the DMA channel assigned to the executing core is busy.
	0x02 - CRC_INVALID_ADDRESS	Return value from the DMA based APIs if the data pointer does not align to 16-bit address or 32-bit address while calculation CRC for 16-bit or 32-bit respectively.
	0x03 - CRC_INVALID_LENGTH	Return value from the DMA based APIs in the following cases: a. If the length of the input data is zero. b. If the 8-bit ,16-bit or 32-bit aligned input data length exceeds 16383. c. If the input data stream is not aligned to the 16-bit word boundary or 32-bit word boundary while calculating CRC for 16- bit and 32-bit respectively.
	0x04 - CRC_INVALID_POINTER	Return value from the DMA based APIs if the data pointer is NULL.
	0x00 - CRC_OK	Return value from the DMA based APIs if all the checks are successful.
	0x05 - CRC_INVALID_CORE	Return value from the DMA based APIs when none of the FCE or the DMA channel (only channels allocated to the CRC module in the DMA configuration) is assigned to the current executing core. *Note: The Crc_ChannelConfig structure would be NULL for such core.*

(table continues...)

Table 45 (continued) Specification for Crc_DmaReturnType

Description	Enumeration to hold the return values from DMA APIs.
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.3 Crc_ErrNotificationPtrType

Table 46 Specification for Crc_ErrNotificationPtrType

Syntax	`Crc_ErrNotificationPtrType`
Type	Pointer to a function of type void Function_Name (void)
File	`Crc.h`
Description	This function pointer holds the function configured by the user in CRC module, which needs to be invoked if there an error event from the move engine.
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.4 Crc_ResNotificationPtrType

Table 47 Specification for Crc_ResNotificationPtrType

Syntax	`Crc_ResNotificationPtrType`
Type	Pointer to a function of type void Function_Name (const uint32 CrcResult)
File	`Crc.h`
Description	This function pointer type would hold the address of the function configured by the user in CRC module, which has to be invoked when there is a successful completion of data by DMA channel.
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3 Functions - APIs

This section lists all the APIs of the CRC driver.

1.3.3.1 Crc_CalculateCRC8

Table 48 Specification for `Crc_CalculateCRC8` API

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

(table continues...)

Table 48 (continued) Specification for `Crc_CalculateCRC8` API

Service ID	0x01	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue8 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue8. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue8 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint8	8-bit result of CRC calculation
Description	This AUTOSAR API makes a CRC8 calculation on the number of data bytes specified by the API parameter Crc_Length with initial value as 0xFF. The API uses SAE J1850 polynomial for CRC calculation. This API supports both single-core and multi-core operations. Refer to the AUTOSAR SWS for further details.	
Source	AUTOSAR	
Error handling	CRC_E_PARAM_LENGTH, CRC_E_PARAM_POINTER	
Configuration dependencies	Crc8Mode	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.2 Crc_CalculateCRC8H2F

Table 49 Specification for `Crc_CalculateCRC8H2F` API

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

(table continues...)

Table 49 (continued) Specification for `Crc_CalculateCRC8H2F` API

Service ID	0x05	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue8H2F Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue8H2F. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue8H2F is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint8	8-bit result of CRC calculation
Description	This AUTOSAR API makes a CRC8 calculation on the number of data bytes specified by the API parameter Crc_Length with initial value as 0xFF. The API uses 0x2F polynomial for CRC calculation. This API supports both single-core and multi-core operations. Refer to the AUTOSAR SWS for further details.	
Source	AUTOSAR	
Error handling	CRC_E_PARAM_LENGTH, CRC_E_PARAM_POINTER	
Configuration dependencies	Crc8H2FMode	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.3 Crc_CalculateCRC16

Table 50 Specification for `Crc_CalculateCRC16` API

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

(table continues...)

Table 50 (continued) Specification for `Crc_CalculateCRC16` API

Service ID	0x02	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue16 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue16. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue16 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint16	16-bit result of CRC calculation
Description	This AUTOSAR API makes a CRC16 calculation on the number of data bytes specified by the API parameter Crc_Length with initial value as 0xFFFF. The API uses CCITT-FALSE CRC16 Standard for CRC calculation. This API supports both single-core and multi-core operations. Refer to the AUTOSAR SWS for further details.	
Source	AUTOSAR	
Error handling	CRC_E_PARAM_LENGTH, CRC_E_PARAM_POINTER	
Configuration dependencies	Crc16Mode	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.4 Crc_CalculateCRC16ARC

Table 51 Specification for `Crc_CalculateCRC16ARC` API

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

(table continues...)

Table 51 (continued) Specification for `Crc_CalculateCRC16ARC` API

Service ID	0x08	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue16 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore Crc_StartValue16. FALSE: Subsequent call in a call sequence. Crc_StartValue16 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint16	16-bit result of CRC calculation
Description	This AUTOSAR API makes a CRC16ARC calculation on the number of data bytes specified by the API parameter Crc_Length with initial value as 0x0000. It uses 0x8005 polynomial for CRC calculation. It supports both single-core and multi-core operations. Refer to the AUTOSAR SWS for further details.	
Source	IFX for AS4.2.2 variant and AUTOSAR for AS4.4.0 variant	
Error handling	CRC_E_PARAM_POINTER, CRC_E_PARAM_LENGTH	
Configuration dependencies	Crc16ARCMode	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.5 Crc_CalculateCRC32

Table 52 Specification for `Crc_CalculateCRC32` API

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

Service ID	0x03

(table continues...)

Table 52 (continued) Specification for `Crc_CalculateCRC32` API

Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue32 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue32. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue32 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	32-bit result of CRC calculation
Description	This AUTOSAR API makes a CRC32 calculation on the number of data bytes specified by the API parameter Crc_Length with initial value as 0xFFFFFFFF. The API uses IEEE-802.3 CRC32 Ethernet Standard for CRC calculation. This API supports both single-core and multi-core operations. Refer to the AUTOSAR SWS for further details.	
Source	AUTOSAR	
Error handling	CRC_E_PARAM_LENGTH, CRC_E_PARAM_POINTER	
Configuration dependencies	Crc32Mode	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.6 Crc_CalculateCRC32P4

Table 53 Specification for `Crc_CalculateCRC32P4` API

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

Service ID	0x06

(table continues...)

Table 53 (continued) Specification for `Crc_CalculateCRC32P4` API

Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue32 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue32. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue32 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	32-bit result of CRC calculation
Description	This AUTOSAR API makes a CRC32P4 calculation on the number of data bytes specified by the API parameter Crc_Length with initial value as 0xFFFFFFFF. The API uses 0xF4ACFB13 polynomial for CRC calculation. This API supports both single-core and multi-core operations. Refer to the AUTOSAR SWS for further details.	
Source	AUTOSAR	
Error handling	CRC_E_PARAM_LENGTH, CRC_E_PARAM_POINTER	
Configuration dependencies	Crc32P4Mode	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.7 Crc_CalculateCRC64

Table 54 Specification for `Crc_CalculateCRC64` API

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

Service ID	0x07

(table continues...)

Table 54 (continued) Specification for `Crc_CalculateCRC64` API

Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue64 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore Crc_StartValue64. FALSE: Subsequent call in a call sequence. Crc_StartValue64 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint64	64 bit result of CRC calculation.
Description	This AUTOSAR API makes a CRC64 calculation on the number of data bytes specified by the API parameter Crc_Length with initial value as 0xFFFFFFFFFFFFFFFF. The API uses 0x42F0E1EBA9EA3693 polynomial for CRC calculation. This API supports both single-core and multi-core operations. Refer to the AUTOSAR SWS for further details.	
Source	IFX for AS4.2.2 variant and AUTOSAR for AS4.4.0 variant	
Error handling	CRC_E_PARAM_POINTER, CRC_E_PARAM_LENGTH	
Configuration dependencies	Crc64Mode	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.8 Crc_DmaCalculateCRC8

Table 55 Specification for `Crc_DmaCalculateCRC8` API

Syntax	`Crc_DmaReturnType  Crc_DmaCalculateCRC8`
	`(`
	`  const uint8 * const Crc_DataPtr,`
	`  const uint32 Crc_Length,`
	`  const uint8 Crc_StartValue8,`
	`  const boolean Crc_IsFirstCall`
	`)`

Service ID	0x1A

(table continues...)

Table 55 (continued) Specification for `Crc_DmaCalculateCRC8` API

Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant for same channel	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue8 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue8. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue8 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Crc_DmaReturnType	CRC_INVALID_CORE: If the current executing core is not allocated with any FCE and DMA channel (only channels allocated to the CRC module in the DMA configuration). CRC_INVALID_POINTER: If the data pointer is NULL CRC_INVALID_LENGTH: In case, a. If the length of the input data is zero. b. If the 8-bit aligned input data length exceeds 16383 bytes. CRC_CHANNEL_BUSY: If the channel is busy. CRC_OK: If all the checks are successful.
Description	This Non AUTOSAR API makes use of DMA and FCE hardware to calculate CRC8 on the number of data bytes specified by the Crc_Length. The calculation is done asynchronously and the value is provided by the callback function from the CRC module to the user. It uses 0x2F polynomial for CRC calculation. It supports both single-core and multi-core operations.	
Source	IFX	
Error handling	-	
Configuration dependencies	CrcDma8bitApi	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(r), FCE_CLC(w), FCE_IN_CFG(w), FCE_IN_CRC(w), FCE_IN_IR(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	

(table continues...)

Table 55 (continued) Specification for `Crc_DmaCalculateCRC8` API

Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.9 Crc_DmaCalculateCRC16

Table 56 Specification for `Crc_DmaCalculateCRC16` API

Syntax	`Crc_DmaReturnType  Crc_DmaCalculateCRC16`
	`(`
	`  const uint8 * const Crc_DataPtr,`
	`  const uint32 Crc_Length,`
	`  const uint16 Crc_StartValue16,`
	`  const boolean Crc_IsFirstCall`
	`)`	

Service ID	0x1B	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant for same channel	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue16 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue16. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue16 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Crc_DmaReturnType	CRC_INVALID_CORE: If the current executing core is not allocated with any FCE and DMA channel (only channels allocated to the CRC module in the DMA configuration). CRC_INVALID_POINTER: If the data pointer is NULL. CRC_INVALID_LENGTH: In case, a. If the length of the input data is zero. b. If the 16-bit aligned input data length exceeds 16383 half words(1 half word = 2bytes). c. If the input data stream is not aligned to 16-bit word boundary. CRC_INVALID_ADDRESS: If the address of the data pointer does not align to 16-bit address boundary. CRC_CHANNEL_BUSY: If the channel is busy. CRC_OK: If all the checks are done successful.

(table continues...)

Table 56 (continued) Specification for `Crc_DmaCalculateCRC16` API

Description	This Non AUTOSAR API makes use of DMA and FCE hardware to calculate CRC16 on the number of data bytes specified by the Crc_Length. The calculation is done asynchronously and the value is provided by the callback function from the CRC module to the user. It uses CCITT-FALSE CRC16 Standard polynomial CRC calculation. It supports both single-core and multi-core operations.
Source	IFX
Error handling	-
Configuration dependencies	CrcDma16bitApi
User hints	-
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(r), FCE_CLC(w), FCE_IN_CFG(w), FCE_IN_CRC(w), FCE_IN_IR(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.10 Crc_DmaCalculateCRC32

Table 57 Specification for `Crc_DmaCalculateCRC32` API

Syntax	`Crc_DmaReturnType  Crc_DmaCalculateCRC32`
	`(`
	`  const uint8 * const Crc_DataPtr,`
	`  const uint32 Crc_Length,`
	`  const uint32 Crc_StartValue32,`
	`  const boolean Crc_IsFirstCall`
	`)`	

Service ID	0x1C	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant for same channel	
Parameters (in)	Crc_DataPtr	Pointer to start address of data block to be calculated.
	Crc_Length	Length of data block to be calculated in bytes.
	Crc_StartValue32	Start value when the algorithm starts.
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue32. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue32 is interpreted to be the return value of the previous function call.

(table continues...)

Table 57 (continued) Specification for `Crc_DmaCalculateCRC32` API

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Crc_DmaReturnType	CRC_INVALID_CORE: If the current executing core is not allocated with any FCE and DMA channel (only channels allocated to the CRC module in the DMA configuration). CRC_INVALID_POINTER: If the data pointer is NULL. CRC_INVALID_LENGTH: In case, a. If the length of the input data is zero. b. If the 32-bit aligned input data length exceeds 16383 words(1 word = 4 bytes). c. If the input data stream is not aligned to 32-bit word boundary. CRC_INVALID_ADDRESS: If the address of the data pointer does not align to 32-bit address boundary. CRC_CHANNEL_BUSY: If the channel is busy. CRC_OK: If all the checks are successful.
Description	This Non AUTOSAR API makes use of DMA and FCE hardware to calculate CRC32 on the number of data bytes specified by the Crc_Length. The calculation is done asynchronously and the value is provided by the callback function from the CRC module to the user. It uses IEEE-802.3 CRC32 Ethernet Standard for CRC calculation. It supports both single-core and multi-core operations.	
Source	IFX	
Error handling	-	
Configuration dependencies	CrcDma32bitApi	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(r), FCE_CLC(w), FCE_IN_CFG(w), FCE_IN_CRC(w), FCE_IN_IR(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.11 Crc_DmaCalculateCRC32P4

Table 58 Specification for `Crc_DmaCalculateCRC32P4` API

Syntax	`Crc_DmaReturnType Crc_DmaCalculateCRC32P4 ( const uint8 * const Crc_DataPtr, const uint32 Crc_Length, const uint32 Crc_StartValue32, const boolean Crc_IsFirstCall )`	
Service ID	0x1D	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant for same channel	
Parameters (in)	Crc_DataPtr Crc_Length Crc_StartValue32 Crc_IsFirstCall	Pointer to start address of data block to be calculated. Length of data block to be calculated in bytes. Start value when the algorithm starts. TRUE: First call in a sequence or individual CRC calculation. Start from initial value, ignore the parameter Crc_StartValue32. FALSE: Subsequent call in a call sequence. The parameter Crc_StartValue32 is interpreted to be the return value of the previous function call.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Crc_DmaReturnType	CRC_INVALID_CORE: If the current executing core is not allocated with any FCE and DMA channel (only channels allocated to the CRC module in the DMA configuration). CRC_INVALID_POINTER: If the data pointer is NULL. CRC_INVALID_LENGTH: In case, a. If the length of the input data is zero. b. If the 32-bit aligned input data length exceeds 16383 words(1 word = 4 bytes). c. If the input data stream is not aligned to 32-bit word boundary. CRC_INVALID_ADDRESS: If the address of the data pointer does not align to 32-bit address boundary. CRC_CHANNEL_BUSY: If the channel is busy. CRC_OK: If all the checks are successful.
Description	This Non AUTOSAR API makes use of DMA and FCE hardware to calculate CRC32P4 on the number of data bytes specified by the Crc_Length. The calculation is done asynchronously and the value is provided by the callback function from the CRC module to the user. It uses 0xF4ACFB13 polynomial for CRC calculation. It supports both single-core and multi-core operations.	

(table continues...)

Table 58 (continued) Specification for `Crc_DmaCalculateCRC32P4` API

Source	IFX
Error handling	-
Configuration dependencies	CrcDma32P4bitApi
User hints	-
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(r), FCE_CLC(w), FCE_IN_CFG(w), FCE_IN_CRC(w), FCE_IN_IR(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.12 Crc_GetVersionInfo

Table 59 Specification for `Crc_GetVersionInfo` API

Syntax	`void Crc_GetVersionInfo ( Std_VersionInfoType * const Versioninfo )`	
Service ID	0x04	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	Versioninfo	Pointer where the version information of this driver is stored.
Parameters (in - out)	-	-
Return	void	-
Description	This service returns the version information of the CRC driver. This API supports both single core and multi core operations.	
Source	AUTOSAR	
Error handling	CRC_E_PARAM_POINTER	
Configuration dependencies	CrcVersionInfoApi	
User hints	-	

(table continues...)

Table 59 (continued) Specification for `Crc_GetVersionInfo` API	
SFR accessed	-
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.4 Notifications and Callbacks

This section lists all the notification and callbacks of the CRC driver.

1.3.4.1 Crc_DmaErrorIsr

Table 60 Specification for `Crc_DmaErrorIsr` API		
Syntax	`void Crc_DmaErrorIsr ( const uint8 Channel, const uint32 Event )`	
Service ID	0x1F	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	Channel Event	Channel number of DMA which completed the transfer Flags indicating the event which triggered the ISR
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This callback function is called from the DMA module on detecting a move engine error during DMA transfer. While configuring the function in the DMA Tresos configuration, the parameter can be the name or the address (numeric value) of the notification function. *Note1: By default, the notification parameter will be NULL in the DMA Tresos configuration.* *Note2: The CRC driver does not validate the configured function name or address for the correctness and the responsibility falls on the user.* *Note3: User shall configure the callback function in the DMA configuration in the channel allocated to CRC to avoid codegen error.* *Note4: In the event of a DMA error, the associated DMA channel is stopped and deinitialized in the Crc_DmaErrorIsr ISR, by invoking the Dma_ChStopTransfer API.*	
Source	IFX	
Error handling	CRC_E_NOT_CONFIGURED	
Configuration dependencies	CrcChannelConfig	

(table continues...)

Table 60 (continued) Specification for `Crc_DmaErrorIsr` API

User hints	-
SFR accessed	CPU_CORE_ID(r), DMA_TSR(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.4.2 Crc_DmaTransferIsr

Table 61 Specification for `Crc_DmaTransferIsr` API

Syntax	`void Crc_DmaTransferIsr`
	`(`
	`  const uint8 Channel,`
	`  const uint32 Event`
	`)`	

Service ID	0x1E	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	Channel Event	Channel number of DMA which completed the transfer Flags indicating the event which triggered the ISR
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This is the callback function invoked by the DMA at the end of the channel transmission. While configuring the function in the DMA Tresos configuration, the parameter can be the name or the address (numeric value) of the notification function. *Note1: By default, the notification parameter will be NULL in the DMA Tresos configuration.* *Note2: The CRC driver does not validate the configured function name or address for the correctness and the responsibility falls on the user.* *Note3: User shall configure the callback function in the DMA configuration in the channel allocated to CRC to avoid codegen error.*	
Source	IFX	
Error handling	CRC_E_INVALID_ISR, CRC_E_NOT_CONFIGURED	
Configuration dependencies	CrcChannelConfig	
User hints	-	

(table continues...)

Table 61 (continued) Specification for `Crc_DmaTransferIsr` API

SFR accessed	CPU_CORE_ID(r), FCE_IN_RES(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.5 Scheduled functions

The CRC module does not provide any scheduled functions.

1.3.6 Interrupt service routines

The CRC driver does not provide any interrupt handlers.

1.3.7 Callout

The CRC driver does not provide any callout.

1.3.8 Errors Handling

This section describes the various errors reported by the CRC driver.

AUTOSAR CRC library functions do not provide any error classification. CRC recalculation and comparison must be done by each module in the upper layer.

As per the safety measures added in CRC, any failure detected in the range check for the input parameters shall be reported to the upper layer as an error value.

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
CRC_E_INVALID_ISR: Error ID for invalid event of the DMA channel. This is implemented as MCAL safety error.	IFX	0xCC	SAFETY	0xCC	SAFETY
CRC_E_NOT_CONFIGURED: Error ID for the invalid channel received in the ISR. This is implemented as MCAL safety error.	IFX	0xCB	SAFETY	0xCB	SAFETY
CRC_E_PARAM_LENGTH: Error ID for zero length check. This is implemented as MCAL safety error.	IFX	0xC9	SAFETY	0xC9	SAFETY
CRC_E_PARAM_POINTER: Error ID for NULLPTR check. This is implemented as MCAL safety error.	IFX	0xC8	SAFETY	0xC8	SAFETY

1.3.9 Deviations and limitations

The section describes the deviations and limitations of the CRC driver.

1.3.9.1 Deviations

The section describes the deviations from software specification.

1.3.9.1.1 Software specification deviations

This section describes the deviations from software specification.

Table 62 known deviations

Reference	Deviations
Autosar SWS: Ecuc_Crc_00034	Crc64Mode for 64-bit CRC calculation can not beconfigured in the Hardware mode due to FCEhardware limitation.
Autosar SWS: Ecuc_Crc_00032	Crc32P4Mode for 32-bit CRC calculation can not be configured in the Hardware mode. The 32-bit CRC for the polynomial "0xF4ACFB13" in the hardware mode can be achieved by DMA based API by enabling CrcDma32P4bitApi in the Tresos.

1.3.9.1.2 AMDC Violations

The CRC driver does not have any AMDC violations.

1.3.9.1.3 VSMD Violations

The CRC driver does not have any VSMD violations.

1.3.9.2 Limitations

The section describes the limitations of the CRC driver.

Table 63 Known limitations

Reference	Limitation
Autosar SWS: Ecuc_Crc_00034	Hardware mode is not available for usage in CRC 64-bit computation using 0x42F0E1EBA9EA3693 polynomial due to hardware limitation. However, runtime and table methods are supported.
Crc_CalculateCRC64 API	While configuring the initial value for the API Crc_CalculateCRC64 in the Tresos, the Tresos generates warning due to its limitation to represent 64-bit data. The allowed range is [0-9223372036854775807].

(table continues...)

Table 63 (continued) Known limitations

Reference	Limitation
Occurrence of safety error CRC_E_INVALID_ISR from Crc_DmaTransferIsr.	In the event of a DMA error, the Crc_DmaErrorIsr will stop and deinitialize the DMA channel being used. If the DMA channel transfer interrupt gets triggered before this channel gets stopped, the CRC_E_INVALID_ISR will get reported from Crc_DmaErrorIsr. This is expected as the DMA ME continues the transfer, inspite of the error, as documented in the DMA driver user manual (Reference: Known Limitations 'Multiple ME interrupts for source and destination errors').

Revision history

Table 64 Revision History

Date	Version	Description
2023-06-01	5.0	- Released
2023-05-23	4.1	- ASIL level field changed to Safety level with description as "refer to release notes" for all APIs under 1.3.3 Functions - APIs and 1.3.4 Notifications and Callbacks
2021-11-09	4.0	- Released
2021-11-02	3.1	Config variant attribute information is added in 'Configuration interfaces' section.
2021-03-04	3.0	- Released
2021-03-01	2.1	- Removed CRC64 polynomial limitation.
2020-11-30	2.0	- Released
2020-11-30	1.2	- Limitation section updated for CRC64 polynomial. - Crc_DmaCalculateCRC32, Crc_DmaCalculateCRC32P4 and Crc_DmaCalculateCRC16 API's return type description updated.
2020-10-13	1.1	- Section 1.1.4.5 updated for error handling and supervision. - Section 1.1.4.6 updated for ISR priority information. - Limitation section updated for safety error CRC_E_INVALID_ISR.
2020-08-14	1.0	- Released
2020-08-13	0.1	- Initial version. - Crc driver chapter moved from MCISAR_TC3xx_UM_Basic to this document. - Integration hint update for Multicore and Resource Manager for DMA resources. - DMA support updated for DMA channel allocation - Example usage updated for DMA based API. - Key architecture consideration updated. - Crc_DmaTransferIsr and Crc_DmaErrorIsr added in Notifications and Callbacks section. - Limitation and deviations updated.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-06-01

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ocr1484806431059

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.