MCAL User Manual for DEMO package

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.

Intended audience

This document is intended for anyone using the DEMO package 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	Denote variable(s) and reference(s)
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, code snippets, file/folder names, directories, command line inputs
Hyperlink	Provides quick and easy access to cross-referenced topics/sections
>	Indicates that a cascading sub-menu opens when you choose a menu item

Glossary of terms

Table 2 Glossary

Term	Description
AUTOSAR	Automotive Open System Architecture
BSW	Basic software
Channel	A channel is a software exchange medium for data that are defined with the same criteria: configuration parameters, number of data elements with same size and data pointers (source, destination) or location.
CPU	Central processing unit
DEM	Diagnostics event manager (MCAL module)
DER	Destination error
DET	Development error tracer
Driver	A driver is a BSW module located in the MCAL layer and contains the functionality to control and access an internal or an external device.
EB	Externally defined buffer for QSPI
ex_r	Exclusive read access
ex_rw	Exclusive read and write access

About this document

Table 2 Glossary (continued)

Term	Description
ex_w	Exclusive write access
I/O	Input/Output
I2C	Inter-Integrated Circuit
IFX	Infineon Technologies
ISR	Interrupt service routine
MC-ISAR	Microcontroller Infineon Software Architecture
MCU	Microcontroller unit
Module	The element is composed of various software units called modules. Typically each software driver is referred to as module. More explicitly, it is also referred to as software module.
OS	Operating system
Peripheral	Hardware module used by a driver. A driver can use one or more peripherals.
PLL	Phase lock loop
Pn_xxxxx	Port n register(xxxxx is register name)
rw	Read and write access
SchM	Scheduler manager (AUTOSAR module)
SCU	System control unit
SCL	Serial Clock Line
SDA	Serial Data Line
STM	System timer (hardware)
VariantPB	This variant allows a mix of pre-compile time, post-build time and link time configuration parameters. The intention of this variant is to optimize the parameters configuration for a re-loadable binary.
VarinatPC	This variant allows only pre-compile configuration parameters. The intention of this variant is to optimize the parameters configuration for a source code delivery.
w	Write access

Reference documents

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

- AURIX™ TC3xx MCAL User Manual for BASIC package, V1.30.0_9.0, 2019-10-10, Infineon Technologies Munich AG
- AURIX™ TC3xx User Manual, V1.2.0, 2019-04, Infineon Technologies Munich AG
- AURIX™ TC38x Appendix to User Manual, V1.2.0, 2019-04, Infineon Technologies Munich AG
- AURIX™ TC39x-B Appendix to User Manual, V1.2.0, 2019-04, Infineon Technologies Munich AG
- AURIX™ TC35x Appendix to User Manual, V1.2.0, 2019-04, Infineon Technologies Munich AG
- AURIX™ TC37xEXT Appendix to User Manual, V1.2.0, 2019-04, Infineon Technologies Munich AG
- AURIX™ TC37x Appendix to User Manual, V1.2.0, 2019-04, Infineon Technologies Munich AG
- AURIX™ TC36x Appendix to User Manual, V1.2.0, 2019-04, Infineon Technologies Munich AG
- TC35x_AA_Errata_Sheet, Rel1.2, 2019-07-19, Infineon Technologies Munich AG
- TC35x_AB_Errata_Sheet, Rel1.1, 2019-07-19, Infineon Technologies Munich AG
- TC37xEXT_AA_Errata_Sheet, Rel1.2, 2019-07-19, Infineon Technologies Munich AG
- TC37xEXT_AB_Errata_Sheet, Rel1.1, 2019-07-19, Infineon Technologies Munich AG

About this document

- TC39x_BC_Errata_Sheet, Rel1.2, 2019-07-19, Infineon Technologies Munich AG
- TC39x_BB_Errata_Sheet, Rel1.3, 2019-07-19, Infineon Technologies Munich AG
- TC39x_BA_Errata_Sheet, Rel1.5, 2019-07-19, Infineon Technologies Munich AG
- TC39x_AA_Errata_Sheet, Rel1.8, 2019-03-29, Infineon Technologies Munich AG
- TC38x_AD_Errata_Sheet, Rel1.2, 2019-07-19, Infineon Technologies Munich AG
- TC38x_AC_Errata_Sheet, Rel1.2, 2019-07-19, Infineon Technologies Munich AG
- TC38x_AB_Errata_Sheet, Rel1.3, 2019-07-19, Infineon Technologies Munich AG
- TC38x_AA_Errata_Sheet, Rel1.5, 2019-07-19, Infineon Technologies Munich AG

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

Table of contents

1 HSSL driver

1.1 User information

1.1.1 Description

The HSSL driver provides the necessary configuration parameters and APIs for the point-to-point communication of single data value and of large data blocks called streams. The HSSL driver initializes the HSCT module. The HSSL driver is implemented as a pre-compile variant. The HSSL driver does not support the read stream.

The features of HSSL are:

- Point-to-point communication between two devices
- Each kernel provides four channels to transfer data to/from target
- Each channels support direct writing of 8/16/32 bit data from initiator to the target register
- For transferring large data blocks channel 2 contains FIFO
- HSSL module implements the transport layer tasks
- HSCT module implements data link layer and physical layer services

1.1.2 Hardware-software mapping

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

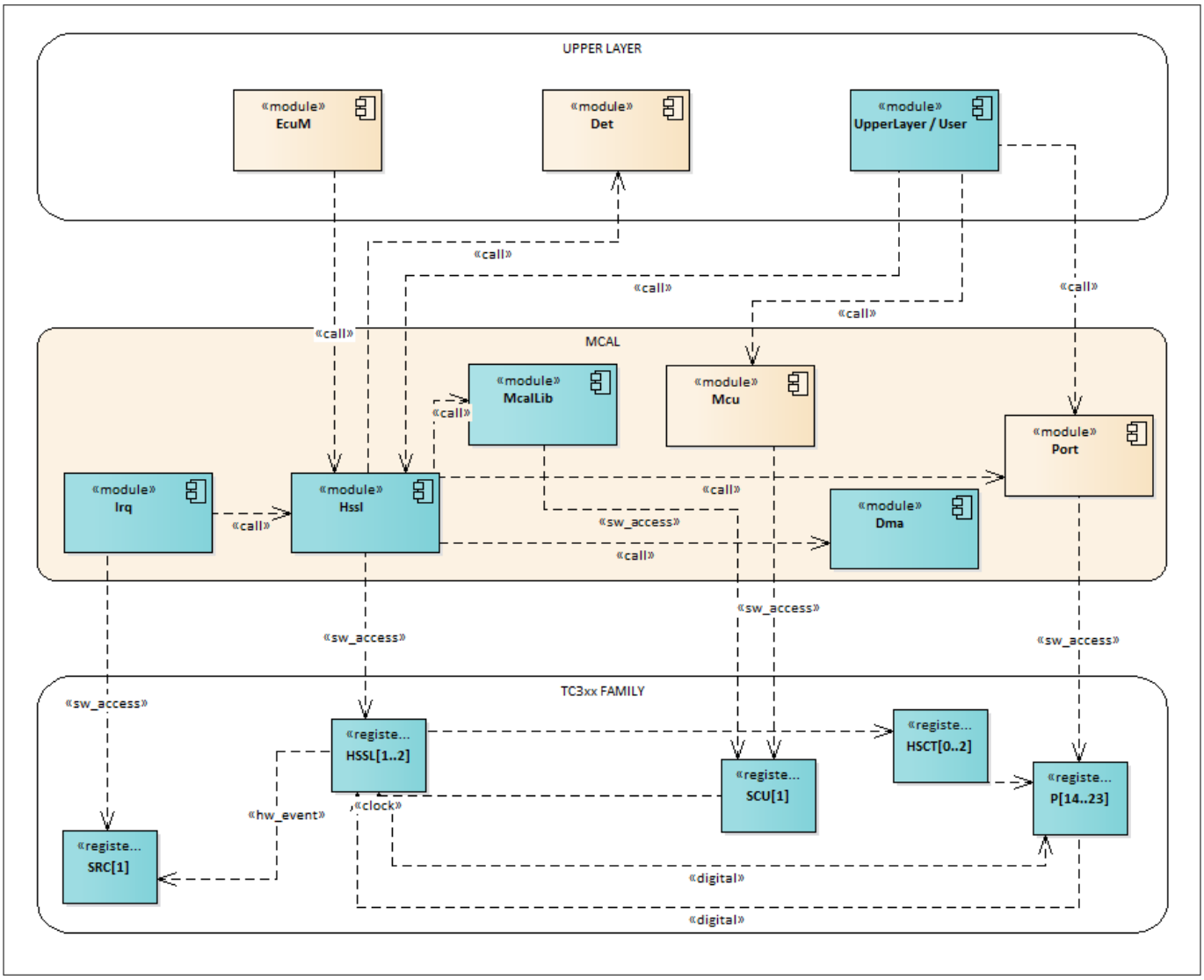


Figure 1 Mapping of hardware-software interfaces

1.1.2.1 HSSL

HSSL is a serial communication protocol driver, which enables two devices to communicate with each other.

Hardware functional features

- Writing a single 8/16/32 bit data value into the register of a target device
- Reading single data from an 8/16/32 bit register of a target device
- Support of 32-bit address range
- Transfers protected by CRC16
- Programmable time outs for detection of blocked answer transfers
- Automatic frame transfer ID generation for detection of dropped frames
- Support of DMA driven multiple register write / read transfers efficient transmission and reception of large data blocks/streams
- Acknowledge for command and stream frames to reduce latency of error detection
- Two stage FIFOs for transmitting and receiving streaming data
- Automatic FIFO flush when entering the run mode, for error handling

- Write protection by an external memory protection unit
- Remote trigger of event / interrupt in the target device by the initiator
- Identification of the target by the JTAG ID number
- Feature set identification of the HSSL module possible by using the JTAG ID number

Users of the hardware

The HSSL driver uses the HSSL and HSCT peripheral of the AURIX™ platform to realize the functionality. The HSSL driver provides APIs to initialize the complete HSSL and HSCT hardware unit to be able to read/write register, block transfer and streaming of data.

Hardware diagnostic features

None.

Hardware events

The HSSL module generates events for the following conditions.

COK – On successful receiving acknowledgment

RDI – On receiving data

ERR - On channel specific error (NACK, Transaction tag (TTE), TIMEOUT and UNEXPECTED)

TRG – On receiving the trigger frame

EXI – On receiving global errors like CRC, SRI/SPB bus access and PHY Inconsistency Error.

1.1.2.2 SCU: dependent Hardware peripheral

Hardware functional features

The HSSL driver depends on the SCU for the clock functionality.

Users of the hardware

The SCU module provides the clock for all the peripherals. It is only the MCU driver, however, that is responsible for the configuration of the clock tree.

Hardware diagnostic features

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

Hardware events

None.

1.1.2.3 SRC

Hardware functional features

The SRC registers are not updated by the HSSL driver. The application should enable the SRC as per the mode HSSL configuration.

Users of the hardware

None.

Hardware diagnostic features

None.

Hardware events

None.

1.1.2.4 Port

Hardware functional features

The HSSL driver depends on the PORT driver for configuring the LVDS port pins.

Users of the hardware

The port settings are exclusively set by the PORT driver. The HSSL driver is indirectly depended on the port functionality.

Hardware diagnostic features

None.

Hardware events

None.

1.1.2.5 DMA

Hardware functional features

The HSSL driver uses the DMA for the transmission and reception of data in the multiread and multiwrite functions. The HSSL driver uses the interface APIs provided by the DMA driver to use the DMA functionality.

Users of the hardware

The DMA channels are exclusively owned by the DMA user, but the functionality is shared by the MCAL drivers. The DMA driver is triggered for every element transmitted or received on the HSSL interface.

Hardware diagnostic features

Move engine (ME) error is enabled during the data transmission.

Hardware events

If any ME error is encountered during the data transfer, the DMA raises an error which is handled by the driver module.

1.1.3 File structure

1.1.3.1 C file structure

The following diagram explains the file structure of the HSSL driver.

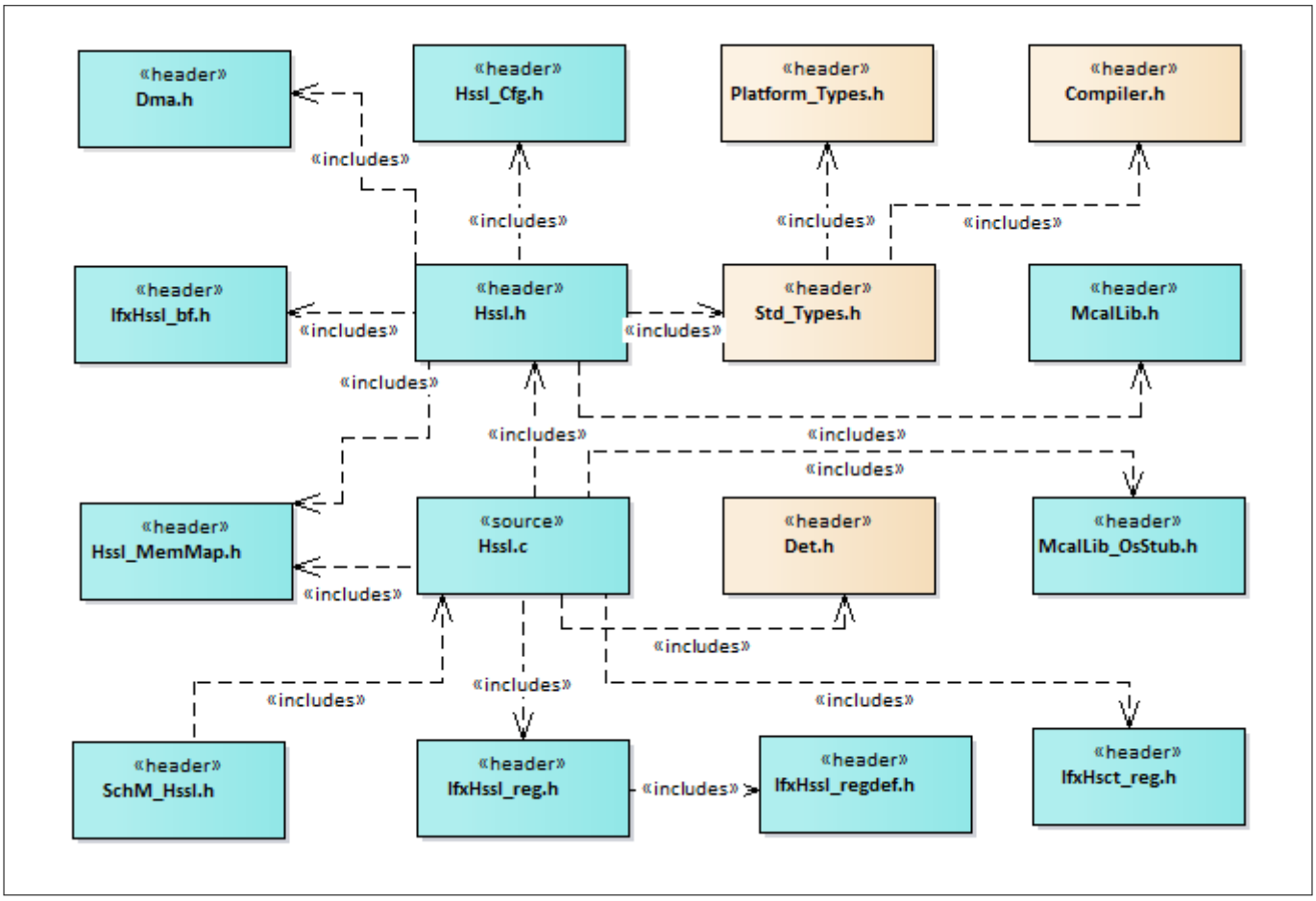


Figure 2 C file structure

Table 3 C file structure

File name	Description
Compiler.h	Provides abstraction from compiler-specific keywords
Std_Types.h	Standard data types to be used are declared here
Hssl_Cfg.h	Generated header file containing macros and configuration data of interrupt priority and interrupt service providers
Platform_Types.h	Platform-specific type declaration file as defined by AUTOSAR
Det.h	Provides the exported interface of DET
Hssl.c	File (static) containing implementation of APIs
Hssl.h	Header file (static) defining prototypes of data structures and APIs
Hssl_MemMap.h	Memmap file is used to define the section of memory to which variables or constants will be placed
Dma.h	Header file (static) defining prototypes of data structure and APIs
McalLib.h	Static header file defining prototypes of data structure and APIs exported by the MCALLIB
IfxHssl_reg.h	SFR header file for HSSL
IfxHssl_bf.h	Provides the Bit Mask, Length and Offset Macro definition for HSSL registers

Table 3 C file structure (continued)

File name	Description
IfxHssl_regdef.h	Includes the register definition file for HSSL
IfxHsct_reg.h	SFR header file for HSCT
IfxHsct_bf.h	Provides the Bit Mask, Length and Offset Macro definition for HSCT registers
IfxHsct_regdef.h	Includes the register definition file for HSCT
IfxDma_reg.h	SFR header file for the DMA
IfxDma_bf.h	SFR header file for the DMA
IfxPort_reg.h	SFR header file for the PORT
IfxPort_bf.h	SFR header file for the PORT
SchM_Hssl.h	Export Header for Schm functions of HSSL driver. Functions to protect critical sections
Hssl_Irq.h	IRQ file for handling all the HSSL interrupts
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.

1.1.3.2 Code generator plugin files

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

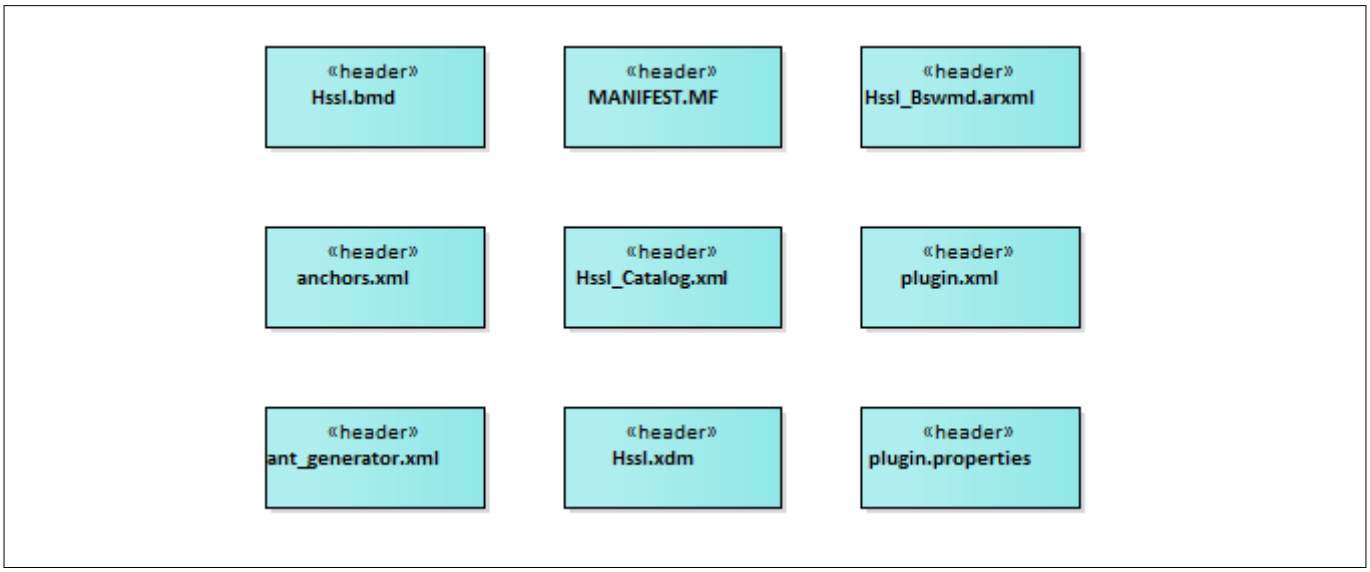


Figure 3 Code generator plugin files

Table 4 Code generator plugin files

File name	Description
anchors.xml	Tresos anchors support file for the HSSL driver
Hssl.xdm	Tresos format XML data model schema file
Hssl.bmd	AUTOSAR format XML data model schema file (for each device)

Table 4 Code generator plugin files (continued)

File name	Description
MANIFEST.MF	Tresos plugin support file containing the metadata for the HSSL driver
plugin.proprties	Tresos plugin support file for the HSSL driver
plugin.xml	Tresos plugin support file for the HSSL driver
Hssl_Bswmd.arxml	AUTOSAR format module description file
Hssl_Catalog.xml	AUTOSAR format catalog file

1.1.4 Integration hints

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

1.1.4.1 Integration with AUTOSAR stack

This sections lists the module that are not part of the MCAL, but are required to integrate the HSSL driver.

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **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 Hssl_MemMap.h file. The Hssl_MemMap.h file is provided in the MCAL package as a stub code. The integrator must place the 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 as follows.

```
/*To be used for all global or static variables.*/
#if defined HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined HSSL_START_SEC_VAR_INIT_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_INIT_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_8
    #undef MEMMAP_ERROR
#elif defined HSSL_START_SEC_VAR_INIT_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_INIT_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_START_SEC_CONST_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_START_SEC_CONST_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_CONST_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_STOP_SEC_CONST_QM_LOCAL_32
    #undef MEMMAP_ERROR
```

```
/* Code Section */
#elif defined HSSL_START_SEC_CODE_QM_LOCAL
    /* User Pragma here */
    #undef HSSL_START_SEC_CODE_QM_LOCAL
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_CODE_QM_LOCAL
     /* User Pragma here */
    #undef HSSL_STOP_SEC_CODE_QM_LOCAL
    #undef MEMMAP_ERROR
#endif
#if defined MEMMAP_ERROR
#error "Hssl_MemMap.h, wrong pragma command"
#endif
```

- **DET**

  The DET module is a part of the AUTOSAR stack that handles all the development and runtime errors reported by the BSW modules. The HSSL driver reports all the development errors to the DET module through the `Det_ReportError()` API. The user of the HSSL driver must process all the errors reported to the DET module through the `Det_ReportError()` API.

  The `Det.h` and `Det.c` files are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.

- **DEM**

  The HSSL driver does not report production errors.

- **SchM**

  The SchM module is a part of the RTE that manages the BSW scheduler. The HSSL driver uses the exclusive areas defined in SchM_Hssl.h to protect the SFRs and variables from concurrent accesses from different threads. The SchMs identified for the HSSL driver are:

  - Channel status lock
  - Activate slave
  - Deactivate slave
  - DMA operated command queues

  The `SchM_Hssl.h` and `SchM_Hssl.c` files are provided in the MCAL package as an example code and needs to updated by the integrator. The user must implement the SchM functions defined by the HSSL driver as suspend/resume of interrupts for the CPU on which the API is invoked. A sample implementation of the SchM functions is as follows:

```
/* sample implementation of SchM_Hssl.c */
void SchM_Enter_Hssl_ChannelStatusLock(void)
{
    /* Start of Critical Section */
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_ChannelStatusLock(void)
{
    /* Start of Critical Section */
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Enter_Hssl_ActivateSlave(void)
{
    /* Start of Critical Section */
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_ActivateSlave(void)
{
    /* Start of Critical Section */
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Enter_Hssl_DeactivateSlave(void)
{
    /* Start of Critical Section */
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_DeactivateSlave(void)
{
    /* Start of Critical Section */
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Enter_Hssl_DmaOperatedCmdQueue(void)
{
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_DmaOperatedCmdQueue(void)
{
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}
```

- **Safety error**

  The HSSL driver does not report any safety error.
- **Notifications and callbacks**

The HSSL driver implements the Hssl_UserNotify and Hssl_DMAUserNotify notification functions for ISR_COK, ISR_RDI, ISR_TRG, ISR_ERR, Hssl_DmaCallout and Hssl_DmaErrCallout, respectively. These notification functions can be configured by the user in the EB tresos for each ISRs separately.

- **Operating system**

  The OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application. The OS files provided by the MCAL package are only an example code and must be updated by the integrator with the actual OS files for the desired function. The HSSL driver does not program any Service Request (SR) register.

1.1.4.2 Multicore and resource manager

The driver does not support execution on multiple cores in parallel.

1.1.4.3 MCU support

The HSSL driver is dependent on the MCU driver for the clock configuration. The initialization of HSSL driver must be started only after completion of the MCU initialization. The following must be considered while configuring the MCU driver in the EB tresos:

In the MCU configuration, the following fields are to be configured in McuClockRefrencePoint

- McuClockRefSelection
- McuClockRefrencePointFrequency

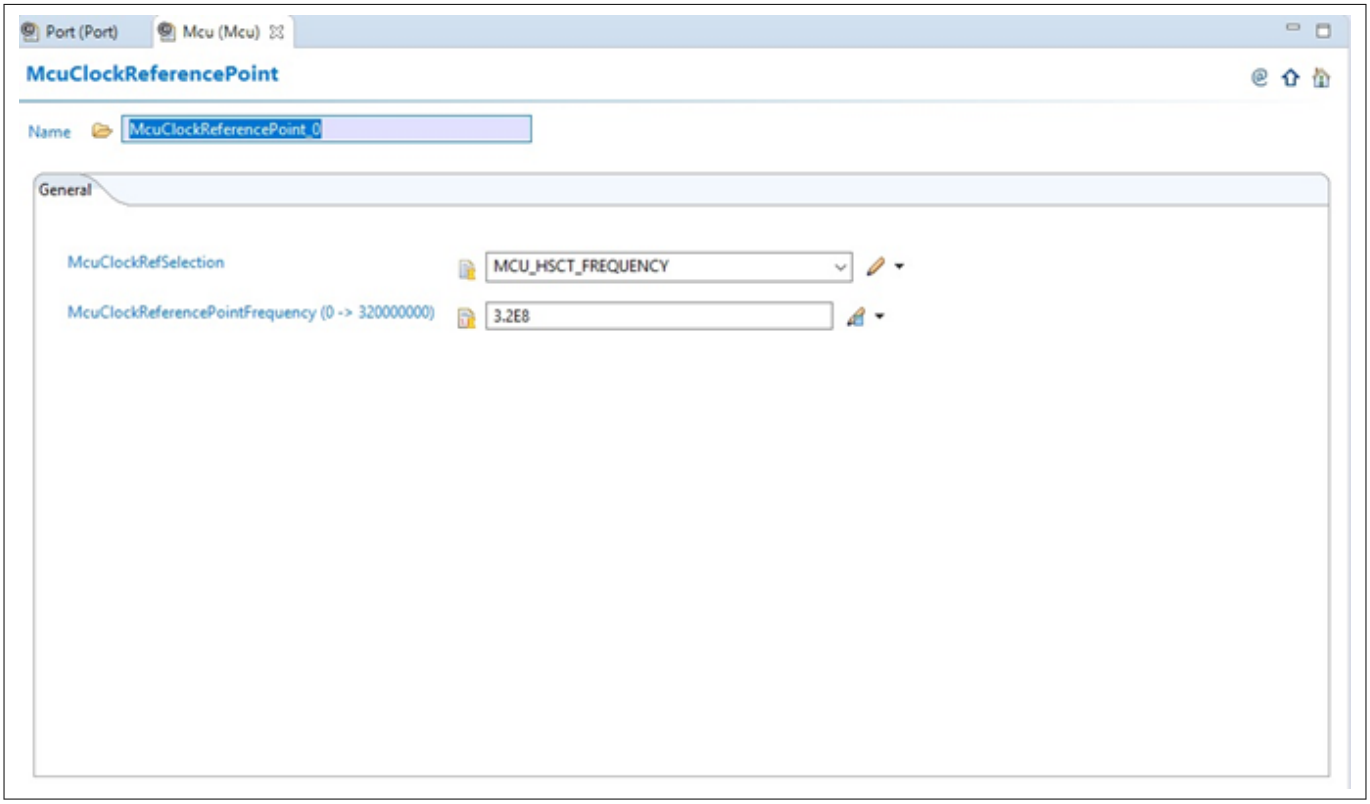


Figure 4 HSSL MCU configuration

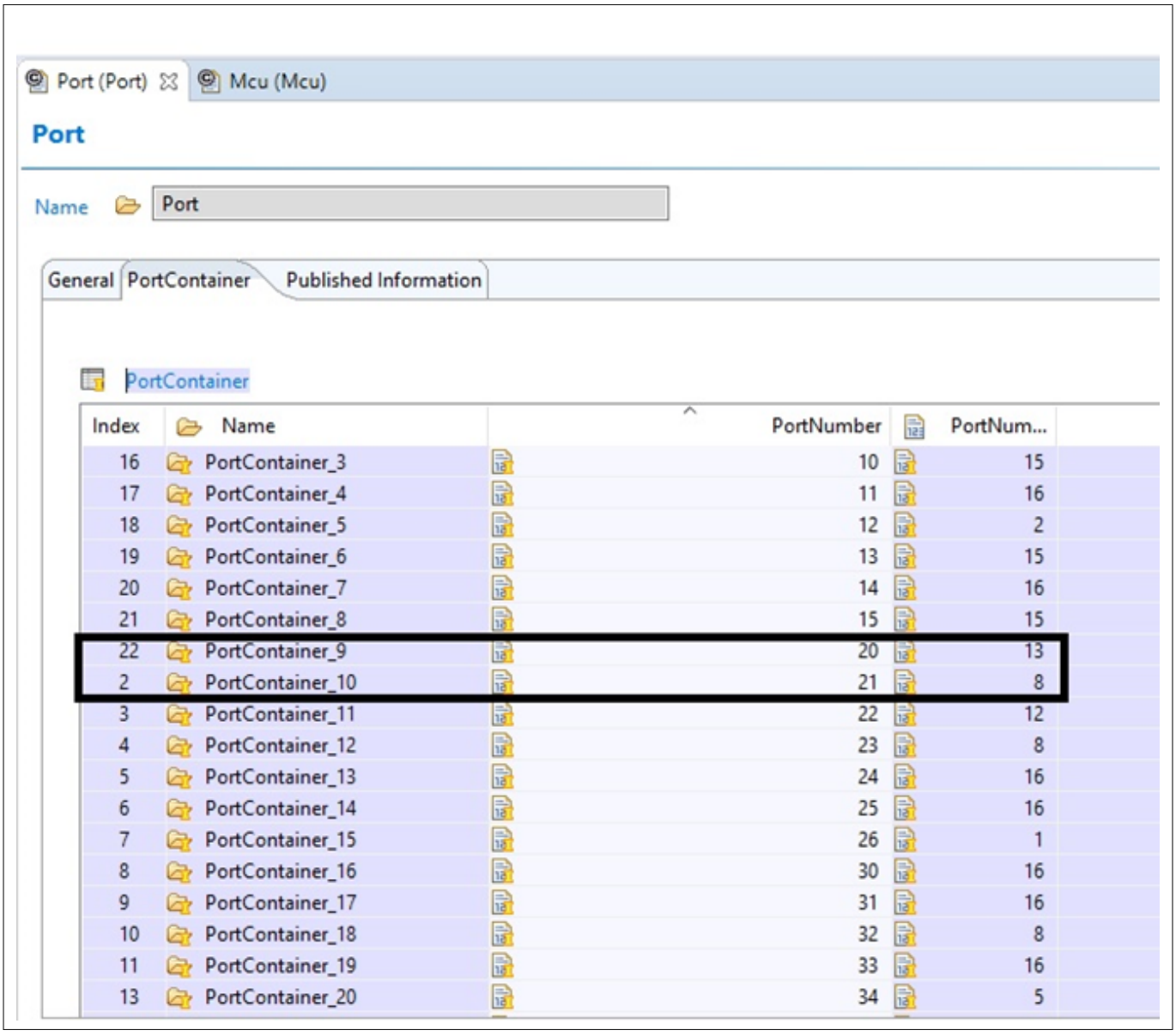
1.1.4.4 PORT support

The PORT driver configures the port pins of the entire microcontroller. The user must configure port pins used by the PORT driver through the port configuration and initialize the port LVDS pins prior to invoking the HSSL initialization.

Following port pins for the HSSL are to be configured as per the configuration:

- HSCT_SYSCLK_OUT - system clock output
- HSCT_RXDN – Rx data negative pin
- HSCT_RXDP - RX data positive pin
- HSCT_TXDN - TX data negative pin
- HSCT_TXDP _ TX data positive pin

The following images shows the example configuration of the PORT pins for HSCT for instance (HSSL0) similarly:



Index	Name	PortNumber	PortNum...
16	PortContainer_3	10	15
17	PortContainer_4	11	16
18	PortContainer_5	12	2
19	PortContainer_6	13	15
20	PortContainer_7	14	16
21	PortContainer_8	15	15
22	PortContainer_9	20	13
2	PortContainer_10	21	8
3	PortContainer_11	22	12
4	PortContainer_12	23	8
5	PortContainer_13	24	16
6	PortContainer_14	25	16
7	PortContainer_15	26	1
8	PortContainer_16	30	16
9	PortContainer_17	31	16
10	PortContainer_18	32	8
11	PortContainer_19	33	16
13	PortContainer_20	34	5

Figure 5 Port number used for HSSL0 module

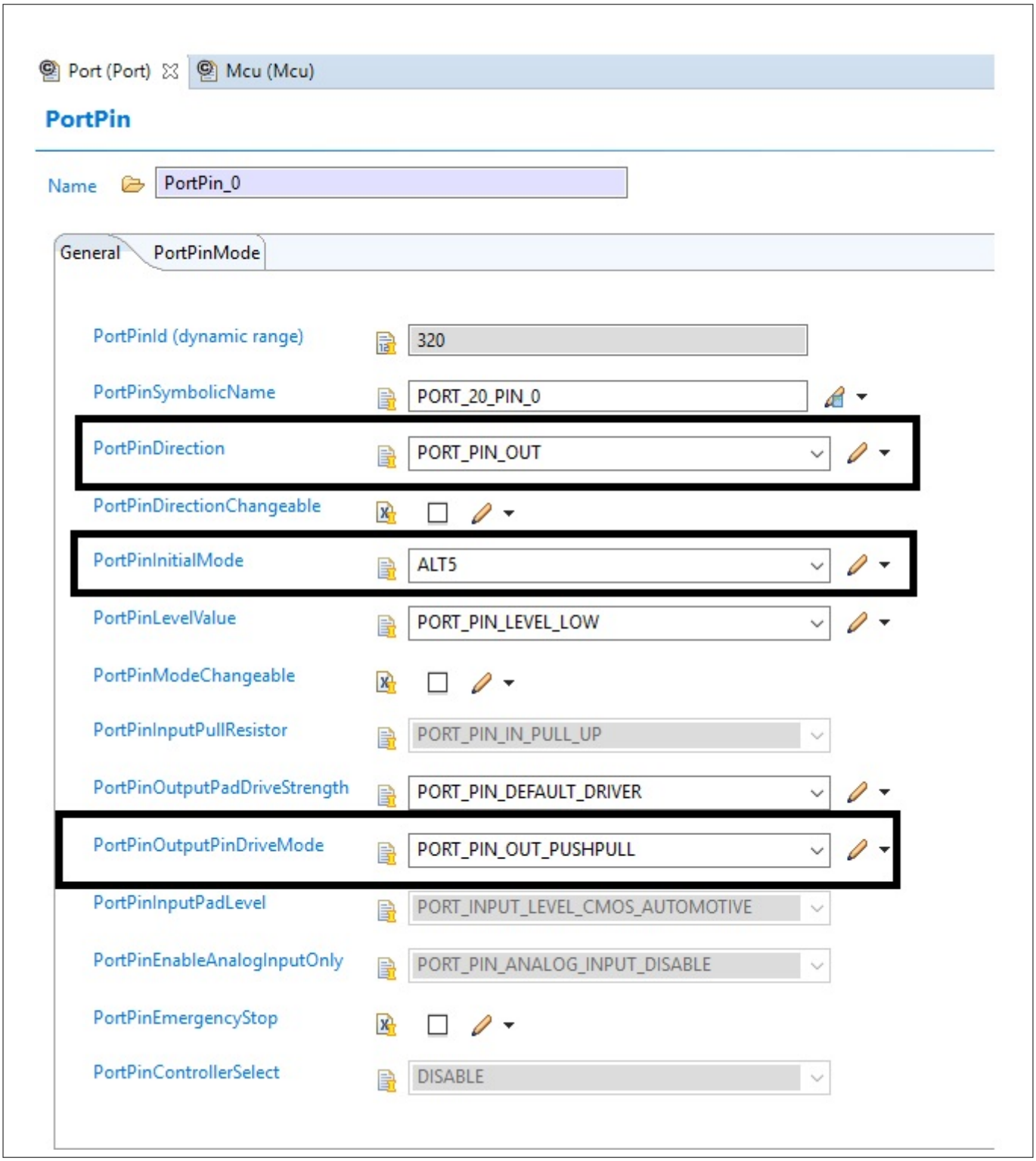


Figure 6 HSSL system clock port output pin configuration

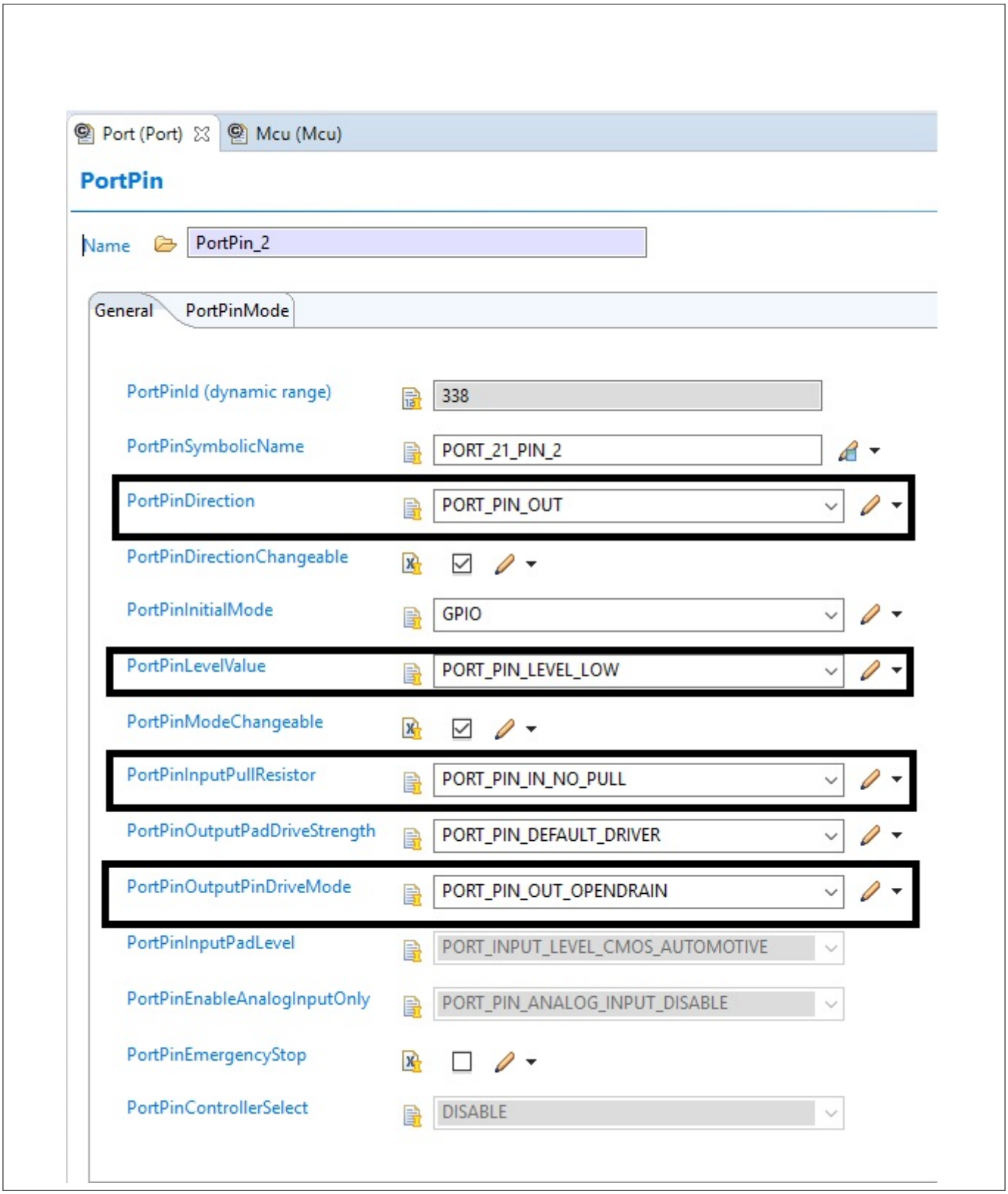


Figure 7 HSSL port pins configuration for RXDN and RXDP

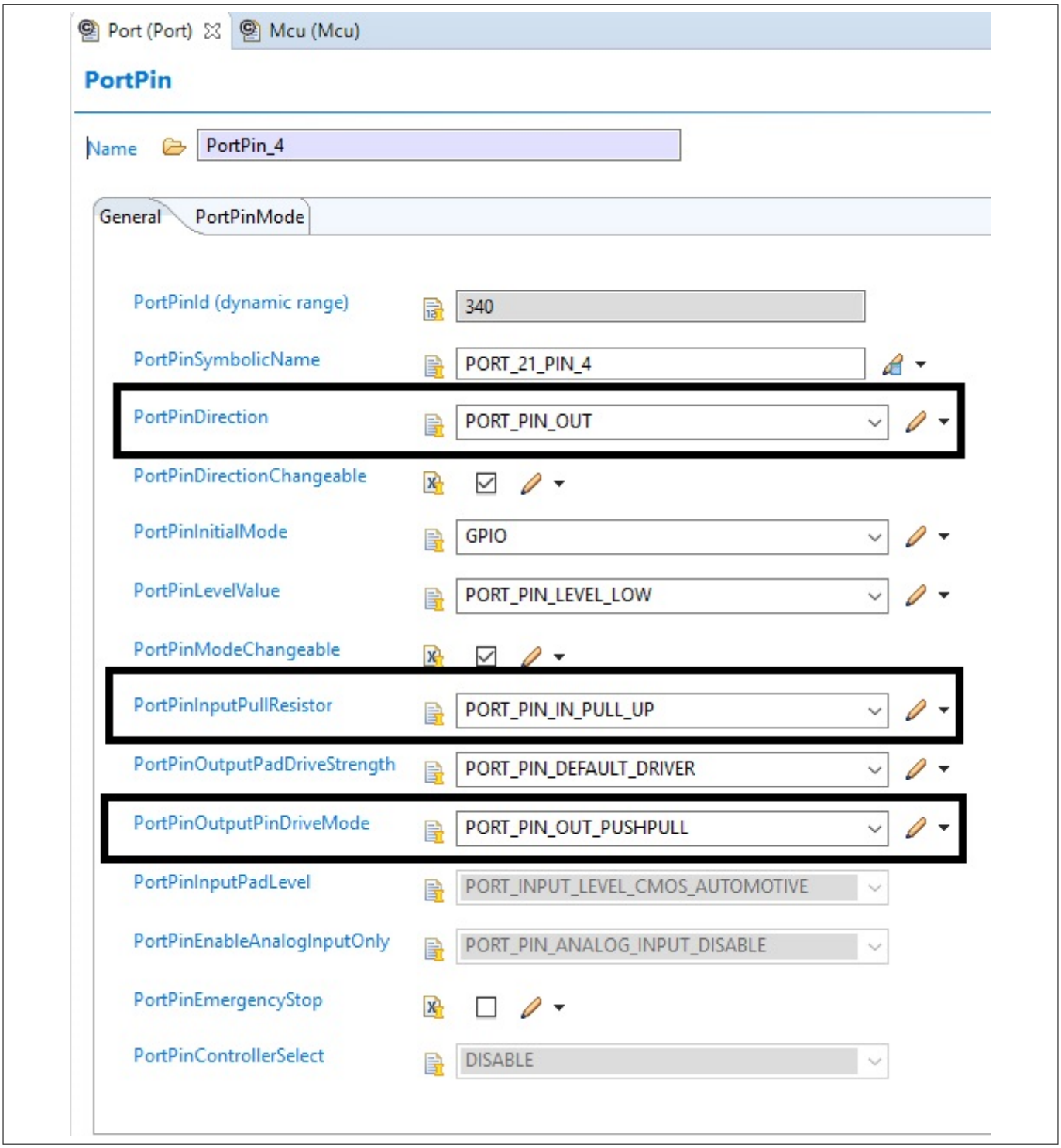


Figure 8 HSSL port configuration for TXDN and TXDP

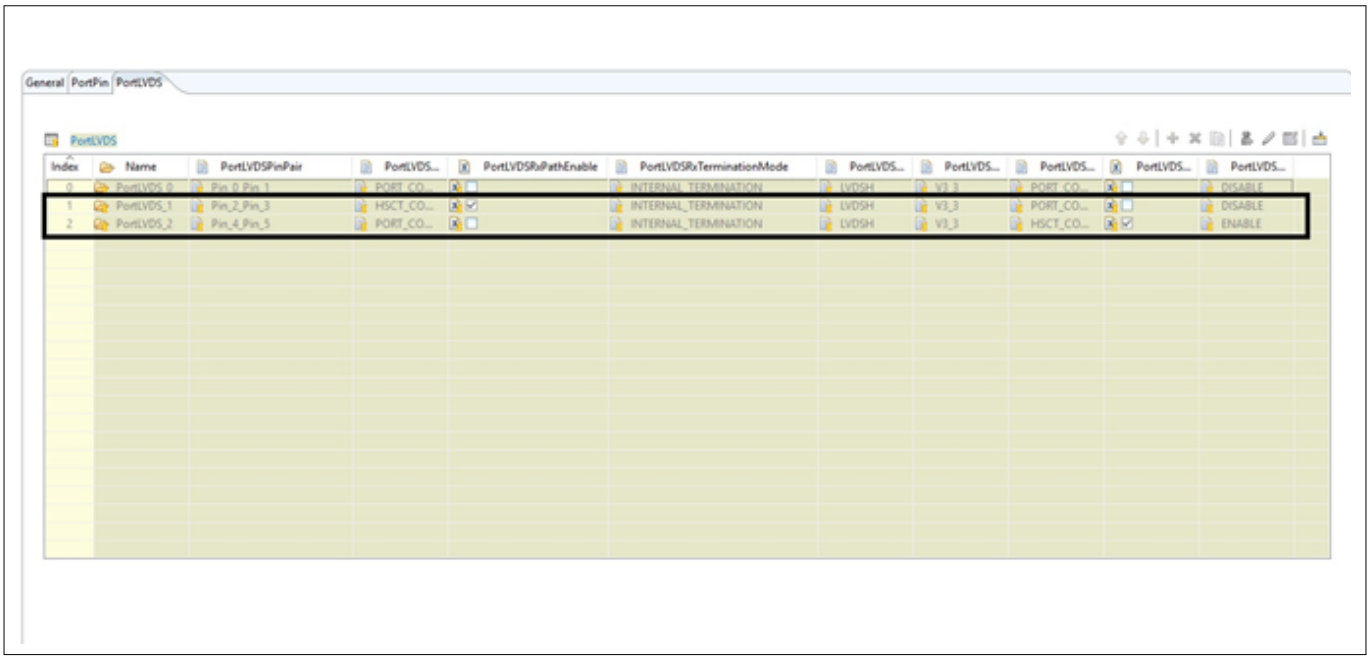


Figure 9 LVDS port configuration for HSCT

1.1.4.5 DMA support

The DMA channels should be configured to use the HSSL `Multi_Read` and `Multi_Write` APIs.
The following figures shows the general configuration of DMA.

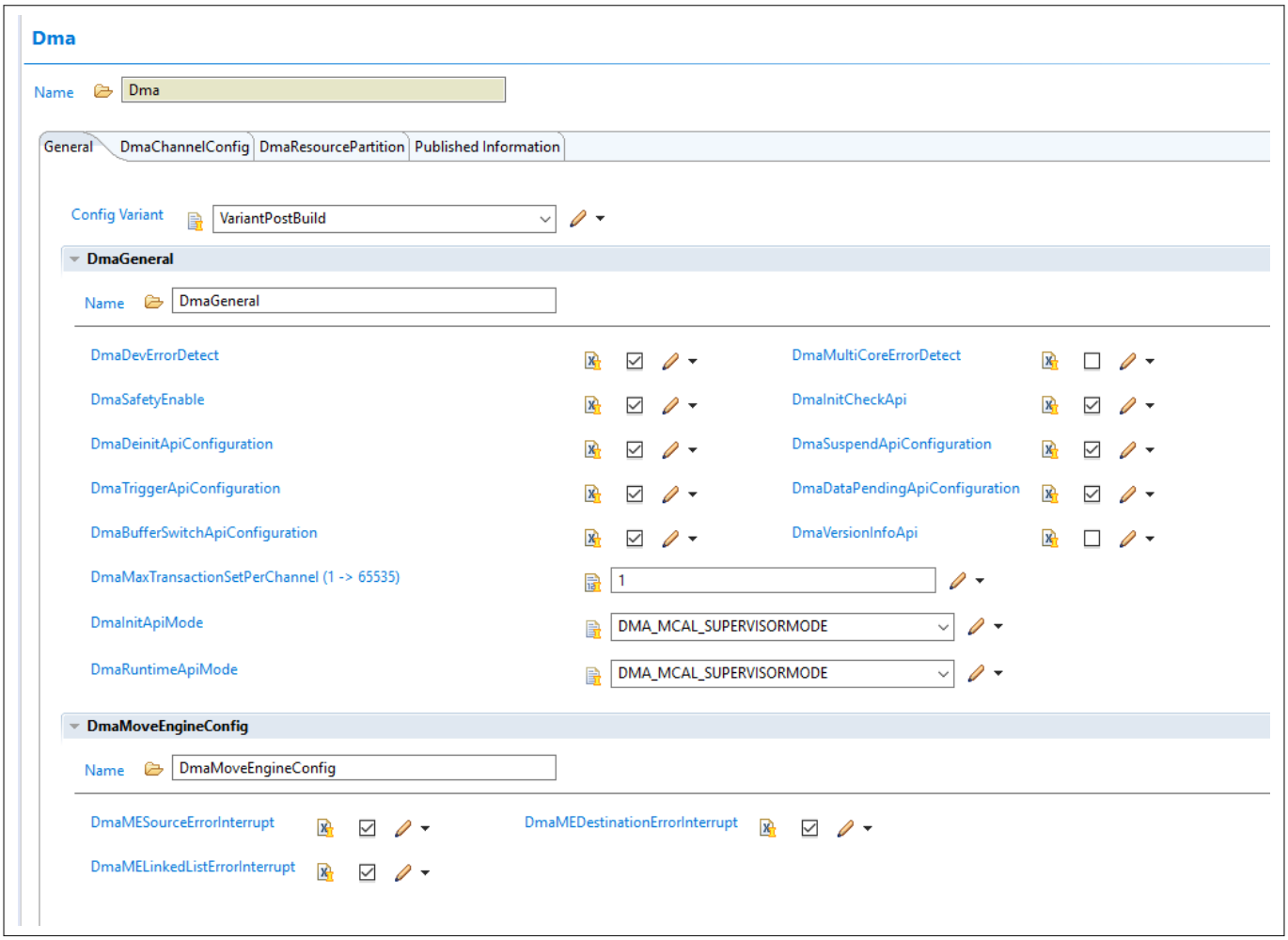


Figure 10 DMA general configuration

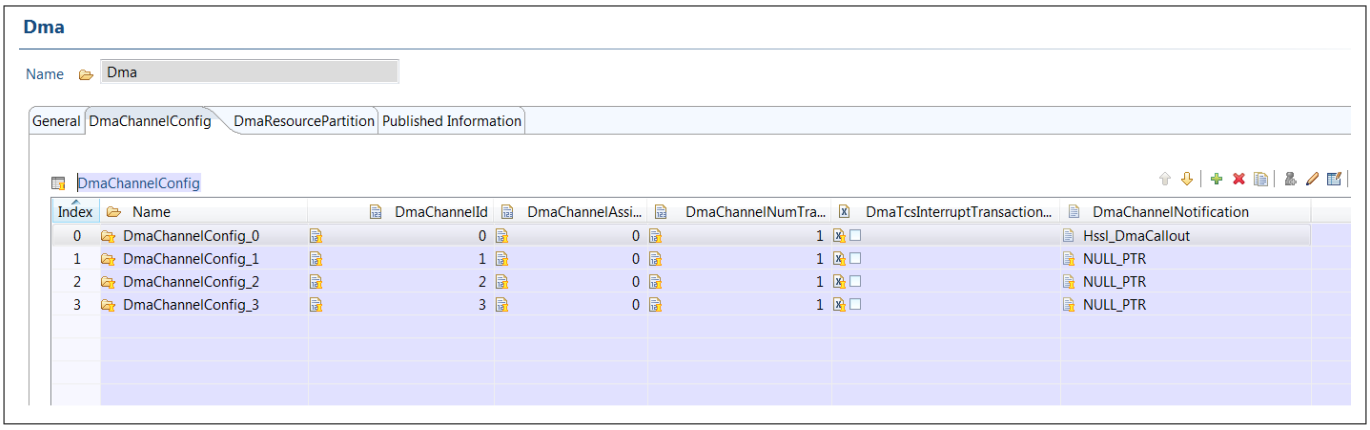


Figure 11 Configure the number of DMA channels to be used

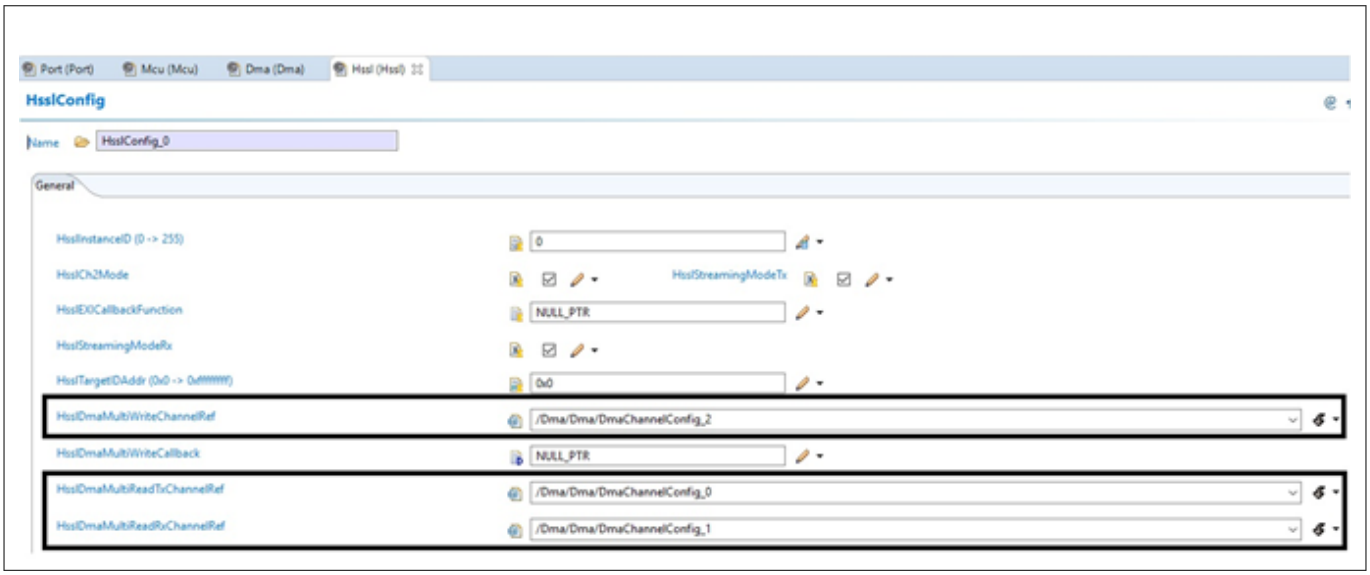


Figure 12 Selection of DMA channels in HSSL configuration

Hssl_MultiRead requires two DMA channels, one for transmission and another for reception.
The following figure shows the configuration to be used for transmit channel for multi-read operation

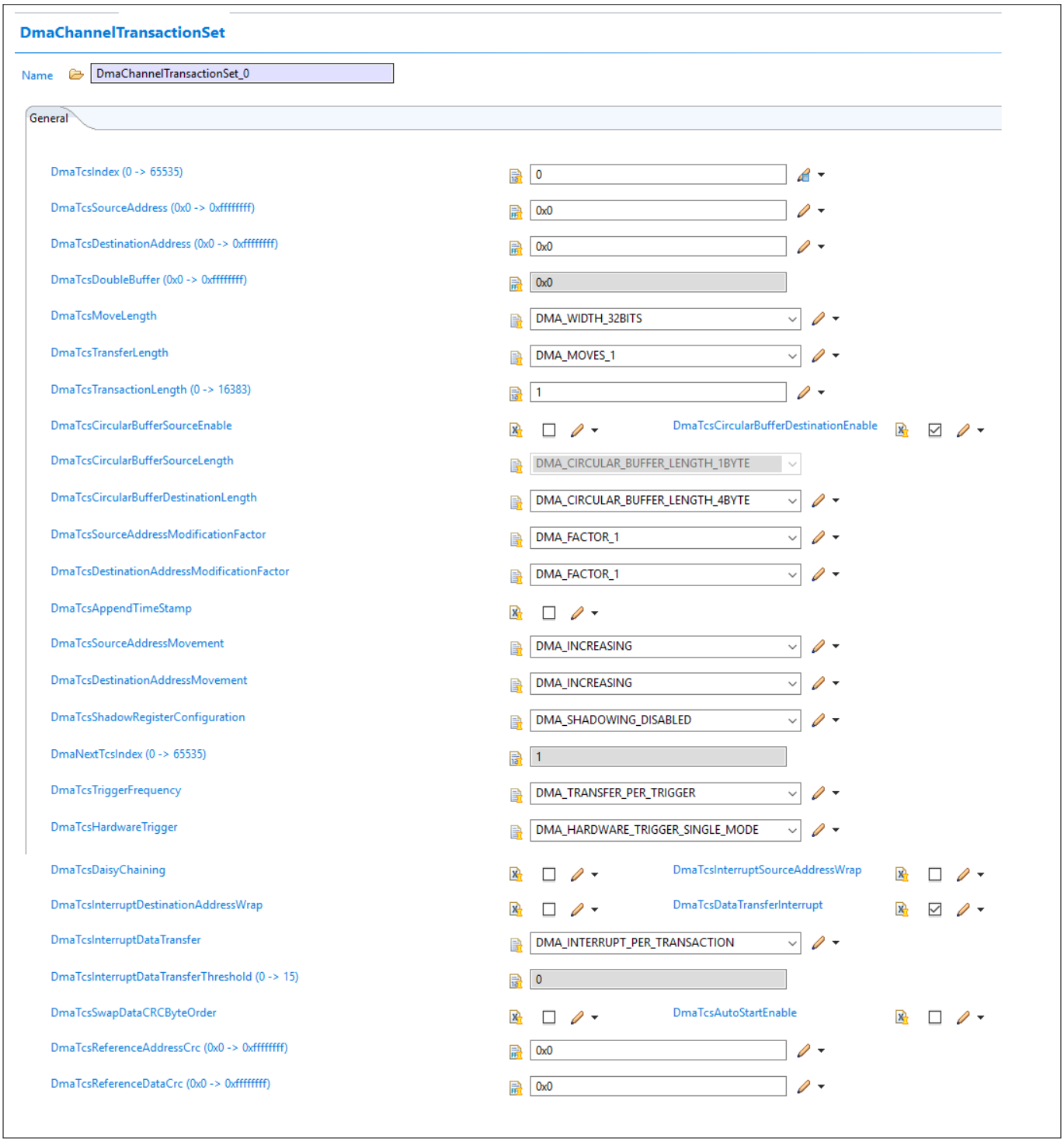


Figure 13 DMA channel transaction configuration for transmit multi-read

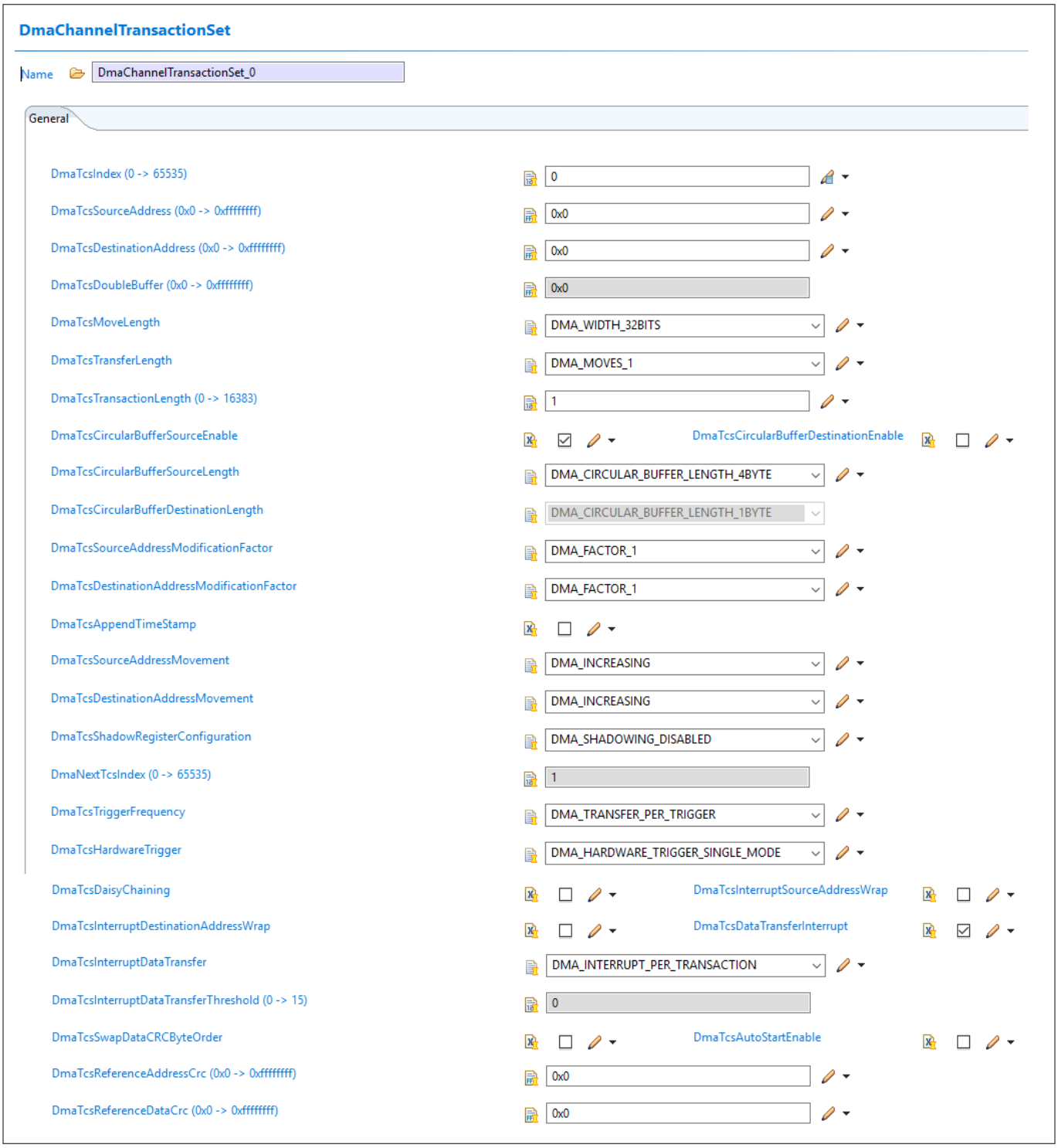


Figure 14 DMA channel transaction for receive multi-read

Hssl_MultiWrite requires one DMA channel for transmission. The following figure shows the configuration to be used for transmit channel for multi-write operation.

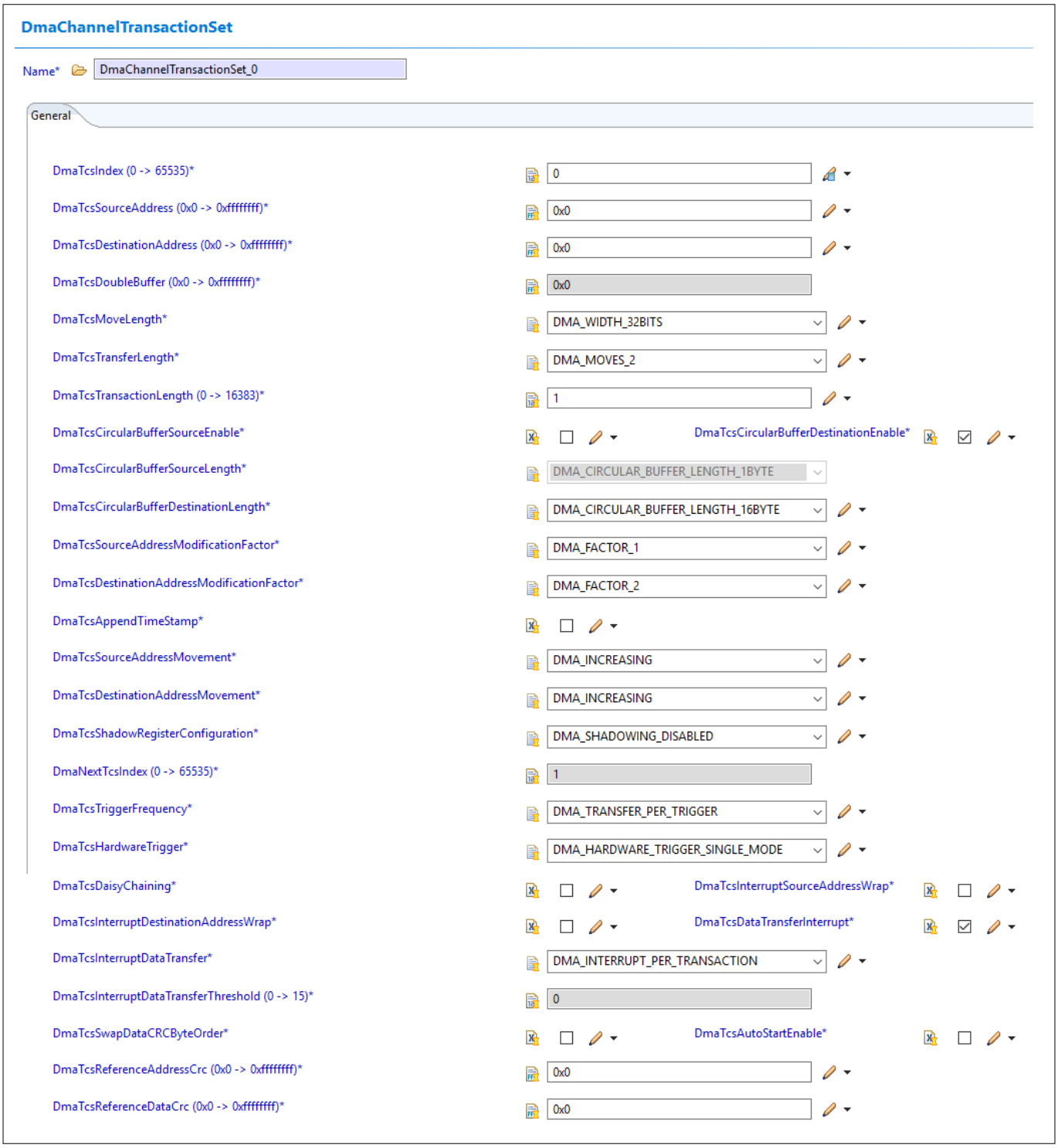


Figure 15 DMA channel transaction for multi-write

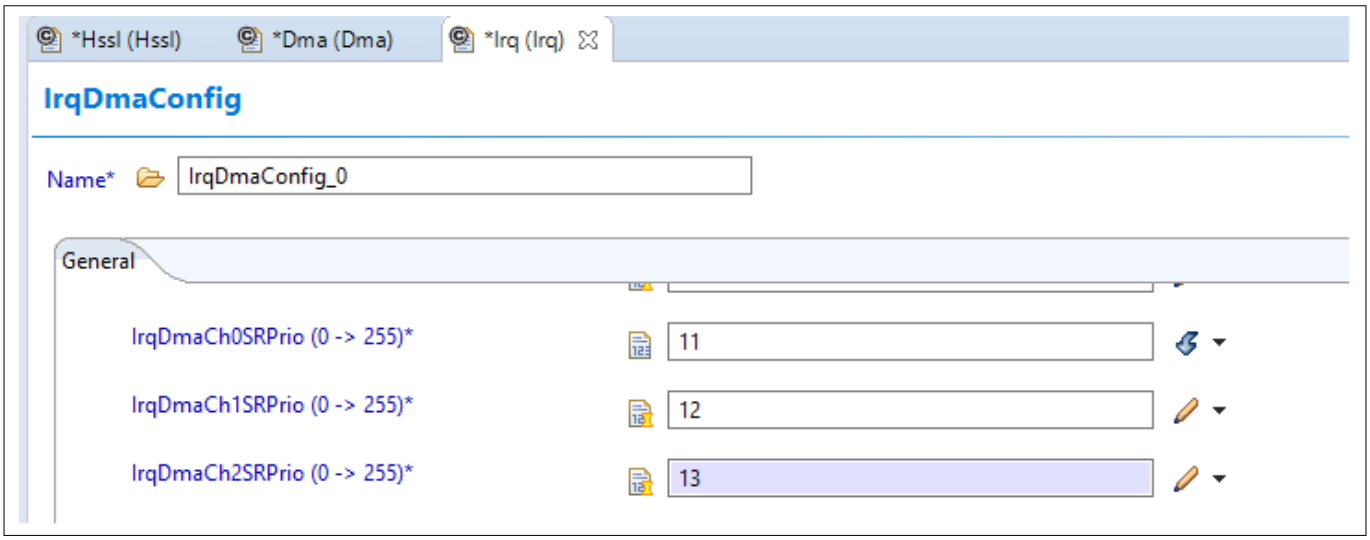


Figure 16 DMA IRQ configuration for multi-read and multi-write

A callout function Hssl_DmaCallout is registered for the each DMA channel which is invoked after every DMA channel transfer completion.

Note: Separate DMA channels to be configured and used for Hssl_MultiRead and Hssl_MultiWrite as mentioned in the above figures. Same DMA channel cannot be used for Hssl_MultiRead and Hssl_MultiWrite.

Note: For Each HSSL kernel requires three separate DMA channels (two for Hssl_MultiRead and one for Hssl_MultiWrite). DMA channels cannot be shared across the HSSL kernels.

Note: In one HSSL kernel, if a channel is performing Hssl_MultiWrite operation then other channels of the same kernel are not allowed to perform the same operation until the channel finishes the HSSL DMA multi write operation. Same applicable for Hssl_MultiRead.

1.1.4.6 Interrupt connections

The HSSL driver has seventeen interrupt lines. The names of the interrupt signals are COK_INT,RDI_INT,ERR_INT and TRG_INT.

Command OK interrupt COK

The arrival of error-free response frame triggers the COK interrupt at the initiator side. This contains four interrupt lines for four channels.

Read data interrupt RDI

The arrival of read response frame triggers RDI interrupt in addition to the COK interrupt. This contains four interrupt lines for four channels.

Error interrupt ERR

When the erroneous response frame (NACK frame) is received at the initiator side, it triggers the ERR interrupt. After an ERR interrupt, normal transmission must be resumed by the software because an optional DMA would remain not triggered and would wait for COK indefinitely. This contains four interrupt lines for four channels.

The HSSL protocol defines four types of errors:

- Time Out Error

A time out error is detected at the initiator side, if the expected ACK frame was not received within the expected time window. This can occur if a frame had been sent by the initiator, and the target detected a CRC error and did not answer with an acknowledge, or the connection between the initiator and the target is physically damaged in one or the other direction.

- Transaction tag error

A transaction tag error occurs at the initiator side, if instead the expected ACK frame with the expected TAG number, an acknowledge with an unexpected transaction tag was received. This would indicate a missing frame or missing acknowledge. Transaction tag errors generate frames that pass the CRC checking stage.

- Target error

A Target Error can occur at the target side, when accessing the target memory a bus error or memory protection error occurs. In such a case, the target responds with an NACK frame indicating the error.

Trigger interrupt TRG

The arrival of a trigger frame at the target side triggers a TRG interrupt at target side. This contains four interrupt lines for four channels.

Exception interrupt EXI

If the receive stage of the HSSL driver detects a CRC error or any inconsistency in the received data, the global EXI Interrupt is activated, which is not channel specific.

- CRC error

A CRC Error can occur:

a. at the target side, in which case:

- the CRC error flag is set
- the received command frame with a CRC error is discarded
- no acknowledge frame is sent and
- a channel unspecific EXI error interrupt is generated, if enabled.

b. at the initiator side, in which case:

- the CRC error flag is set
- the received response frame with a CRC error is discarded
- a channel unspecific EXI error interrupt is generated, if enabled

Both scenarios lead to a time out at the initiator side. In both cases the CRC error flag is set at the side receiving the erroneous frame (either initiator or target) and an interrupt is generated, if enabled.

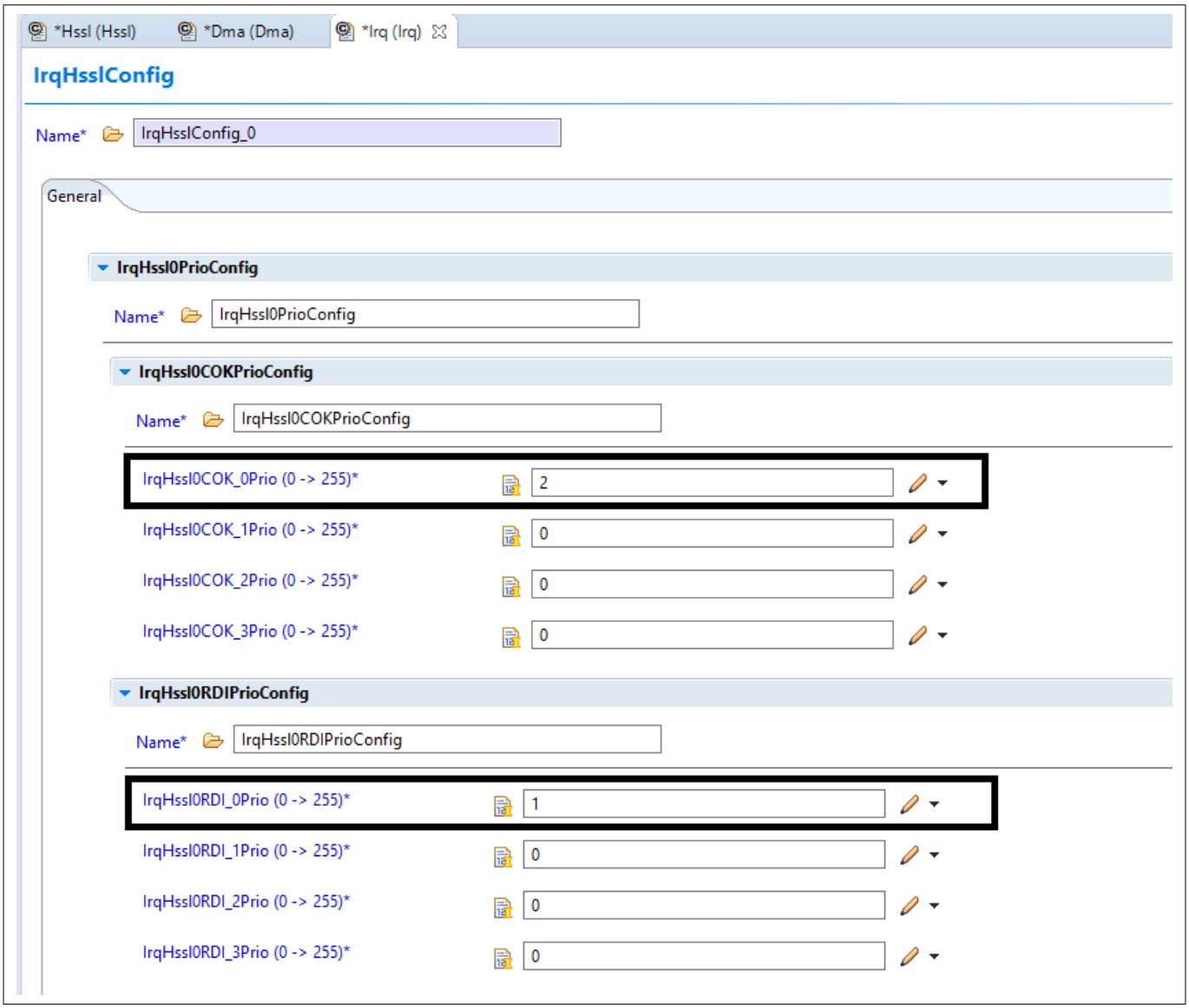


Figure 17 HSSL IRQ priority configuration for multi-read and multi-write

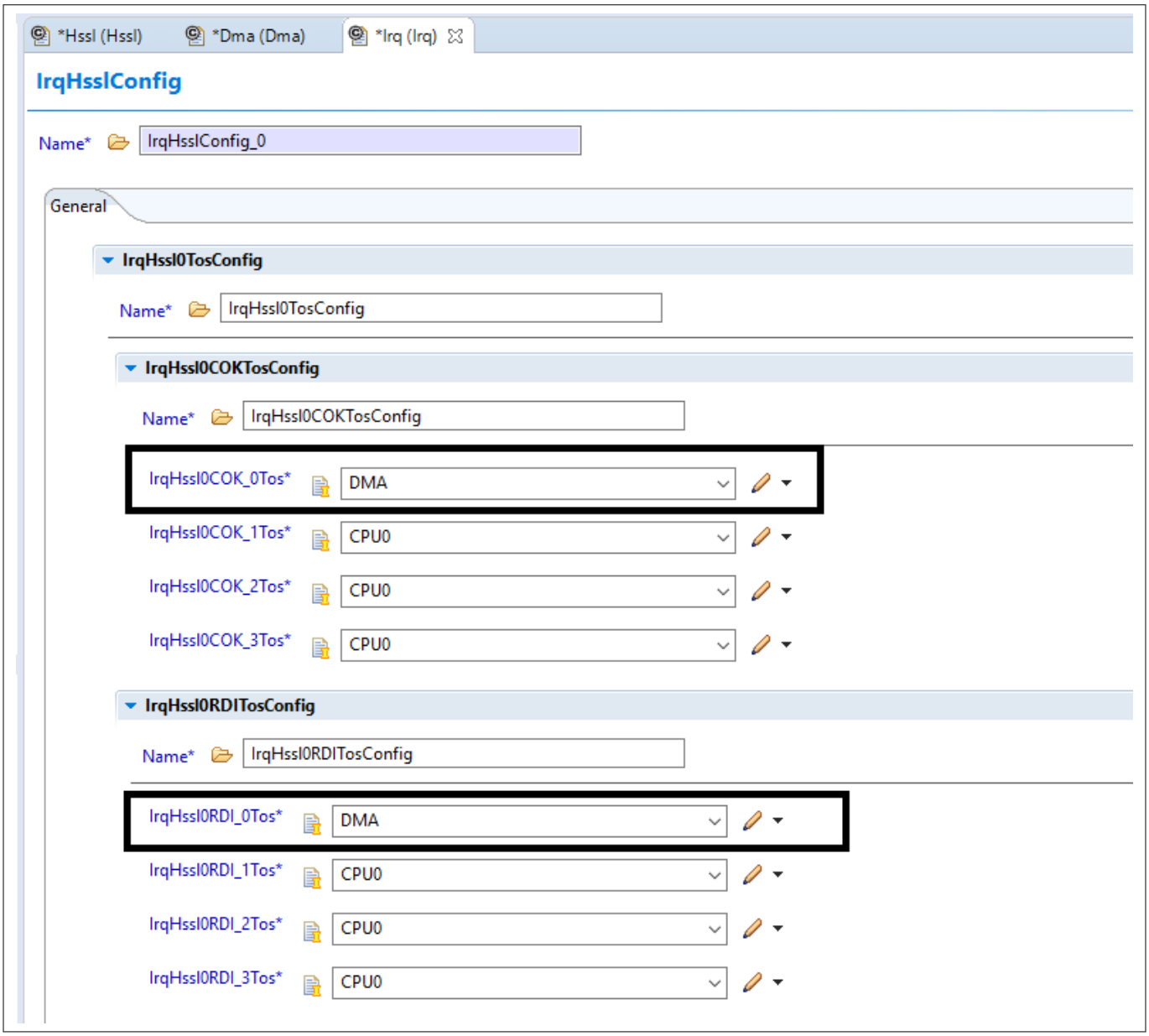


Figure 18 HSSL IRQ configuration for multi-read and multi-write

1.1.4.7 Example usage

The following are the pre-requisite for the HSSL initialization:

Note: Global that needs to be defined in the application code:

- Mcu_ConfigType Mcu_Config
- Port_ConfigType Port_Config
- Dma_ConfigType Dma_Config

Refer to the *Integration hints* and add all dependent modules from the catalog. Follow the below sequence in the application code:

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

4. Initialize the IRQ for dependent modules using the IrqDma_Init and IrqHssl_Init APIs.
5. Initialize the HSSL driver using the Hssl_Init API.

Initialization of the driver

The code sequence for initializing the HSSL driver is as follows.

```
#include "Hssl.h"
#include "Mcu.h"
#include "Port.h"
#include "Dma.h"
#include "Irq.h"

extern const Mcu_ConfigType Mcu_Config;
extern const Dma_ConfigType Dma_Config;
extern const Port_ConfigType Port_Config;

void core0_main (void)
{
    Hssl_ConfigType *cfg = NULL_PTR;

    /* Initialize all dependent modules */
    Mcu_Init(&Mcu_Config);
    Mcu_InitClock( 0 );
    while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
    Mcu_DistributePllClock();

    IrqDma_Init();
    IrqHssl_Init();
    Dma_Init(&Dma_Config);
    Port_Init(&Port_Config);

   /* Enable service request for all the configured interrupts */
    SRC_DMACH1.U |= 0x400U;
    SRC_DMACH2.U |= 0x400U;
    SRC_HSSL0COK0.B.SRE = 0x1;
    SRC_HSSL0RDI0.B.SRE = 0x1;

    Hssl_Init (cfg);
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_INIT) ;
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_RUN) ;
}
```

Sample code for single write command and single read command

The code sequence for performing single write and single read operation between the master and slave is as follows.

```
Hssl_DataTemplateType WriteData;
    uint32 DataBuffer = 0x33333333U;
    uint32 DataAddr;
    Std_ReturnType RetVal;
    Hssl_ChannelType Hssl_channel;

    WriteData.Data = &DataBuffer;
    DataAddr =  0x70003420U;
    WriteData.Address = &DataAddr;
    Hssl_channel.Number = 0U;
    Hssl_channel.Timeout=0xFFFFFFFFU;

   /* Trigger the Write command */
   RetVal = Hssl_Write ((Hssl_InstanceID)0U,&WriteData, HSSL_DATA_SIZE_32BIT,
Hssl_channel ,0U);
   if (RetVal == E_OK)
   {
       RetVal = Hssl_Read
((Hssl_InstanceID)0U,&WriteData,HSSL_DATA_SIZE_32BIT, Hssl_channel ,0U);
   }
```

Sample code for multi-write operation

The code sequence for performing the multi-write operation on the slave using the DMA is shown as follows (refer to the DMA support and interrupt connection configuration).

```
#include "Hssl.h"
#include "Mcu.h"
#include "Port.h"
#include "Dma.h"
#include "Irq.h"

extern const Mcu_ConfigType Mcu_Config;
extern const Dma_ConfigType Dma_Config;
extern const Port_ConfigType Port_Config;

Hssl_DataTemplateType WriteDataDMA[8U];
uint32 DataBufferDMA[8U];
uint8 RetVal;

/* User callback function is invoked post DMA transmission completes */
void Hssl0_Dma_Write_User_Fn(Dma_ChEventType Event)
{
    while(1);
}

void core0_main (void)
{
    Hssl_ConfigType *cfg = NULL_PTR;

    Hssl_ChannelType Hssl_channel;

      Mcu_Init(&Mcu_Config);
      Mcu_InitClock( 0 );
      while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
    Mcu_DistributePllClock();

    IrqDma_Init();
    IrqHssl_Init();
    Port_Init(&Port_Config);
       Dma_Init(&Dma_Config);

       SRC_DMACH1.U |= 0x400U;
    SRC_DMACH2.U |= 0x400U;
    SRC_HSSL0COK0.B.SRE = 0x1;
    SRC_HSSL0RDI0.B.SRE = 0x1;

    Hssl_Init (cfg);
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_INIT) ;
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_RUN) ;


      Hssl_channel.Number = 0U;
      Hssl_channel.Timeout=0xFFFFFFFFU;
```

```
      DataBufferDMA[0U] = 0xAAAAAAAAU;
      DataBufferDMA[1U] = 0x70003420U;
      DataBufferDMA[2U] = 0xBBBBBBBBU;
      DataBufferDMA[3U] = 0x70003430U;
      DataBufferDMA[4U] = 0xCCCCCCCCU;
      DataBufferDMA[5U] = 0x70003440U;
      DataBufferDMA[6U] = 0xDDDDDDDDU;
      DataBufferDMA[7U] = 0x70003450U;

     WriteDataDMA[0U].Data = &DataBufferDMA[0U];
     WriteDataDMA[1U].Address = &DataBufferDMA[1];
     WriteDataDMA[2U].Data = &DataBufferDMA[2U] ;
     WriteDataDMA[3U].Address = &DataBufferDMA[3U];
     WriteDataDMA[4U].Data = &DataBufferDMA[4U] ;
     WriteDataDMA[5].Address = &DataBufferDMA[5U];
     WriteDataDMA[6].Data = &DataBufferDMA[6U] ;
    WriteDataDMA[7].Address = &DataBufferDMA[7U];

    RetVal = Hssl_MultiWrite(0U,(Hssl_DataTemplateType *)WriteDataDMA,
HSSL_DATA_SIZE_32BIT,4U,Hssl_channel,0U);
}
```

Sample code for multi-read operation

The code sequence for performing multi-read operation from the slave using the DMA is shown as folows (refer to the DMA support and interrupt connection configuration).

```
/* User callback function is invoked post DMA transmission completes */
void Hssl0_Dma_Read_User_Fn(Dma_ChEventType Event)
{
    while(1);
}

/*Global variable declarations*/
Hssl_ReadDataTemplateType ReadDataDMA[8U];
uint32 ReaddataBuffer[4];
uint32 DataBufferDMA[8U];
uint8 RetVal;

/*Address buffer from which the data has to read*/
DataBufferDMA[0U] = 0x70003420U;
DataBufferDMA[1U] = 0x70003430U;
DataBufferDMA[2U] = 0x70003440U;
DataBufferDMA[3U] = 0x70003450U;

ReadDataDMA[0U].Address = &DataBufferDMA[0U] ;
ReadDataDMA[1U].Address = &DataBufferDMA[1U];
ReadDataDMA[2U].Address = &DataBufferDMA[2U] ;
ReadDataDMA[3U].Address = &DataBufferDMA[3U];

RetVal = Hssl_MultiRead(0U,
(Hssl_ReadDataTemplateType*)ReadDataDMA,ReadDataDMA,HSSL_DATA_SIZE_32BIT,4U,Hss
l_channel,0U);
```

Sample code for streaming operation

The code sequence for performing streaming operation.

```
uint32 DataAddress[32];
uint32 *DestinationAddressStart = &Dst_Addr;
uint8 retVal;
/*Source buffer data to be transmitted*/
for(Index = 0U; Index < 32U; Index++)
{
  databuf[Index] = 0x22222222;
}
/* Start stream operation */
retVal  = Hssl_StartStream ((Hssl_InstanceID)0U,(&DataAddress[0]),
*DestinationAddressStart,HSSL_DATA_SIZE_32BIT,0U);
```

Sample code for multi-slave operation

The code sequence for performing multi-slave operation.

```
/*Sequence for the multi slave mode*/
uint8 Slaveid = 1U;
uint8 retVal;

/* Slave must be selected before activating a slave */
retVal = Hssl_SelectSlave((Hssl_InstanceID)0U, Slaveid);
if(retVal == E_OK)
{
  retVal = Hssl_ActivateSlave((Hssl_InstanceID)0U,Slaveid,Hssl_SlaveStatusType
*Hssl_SlaveStatus);
}

/*Perform read or write or stream operation*/

/* Deactivating slave */
retVal = Hssl_DeactivateSlave((Hssl_InstanceID)0U,Slaveid,Hssl_SlaveStatusType
*Hssl_SlaveStatus)
```

1.1.5 Key architectural considerations

There are no key architectural considerations for the driver.

1.2 Assumptions of Use (AoU)

There are no AoUs for the driver.

1.3 Reference information

1.3.1 Configuration interfaces

The HSSL driver is delivered as a Variant Pre-Compile.
The following diagram depicts the hierarchy along with the extensions provided for HSSL module.

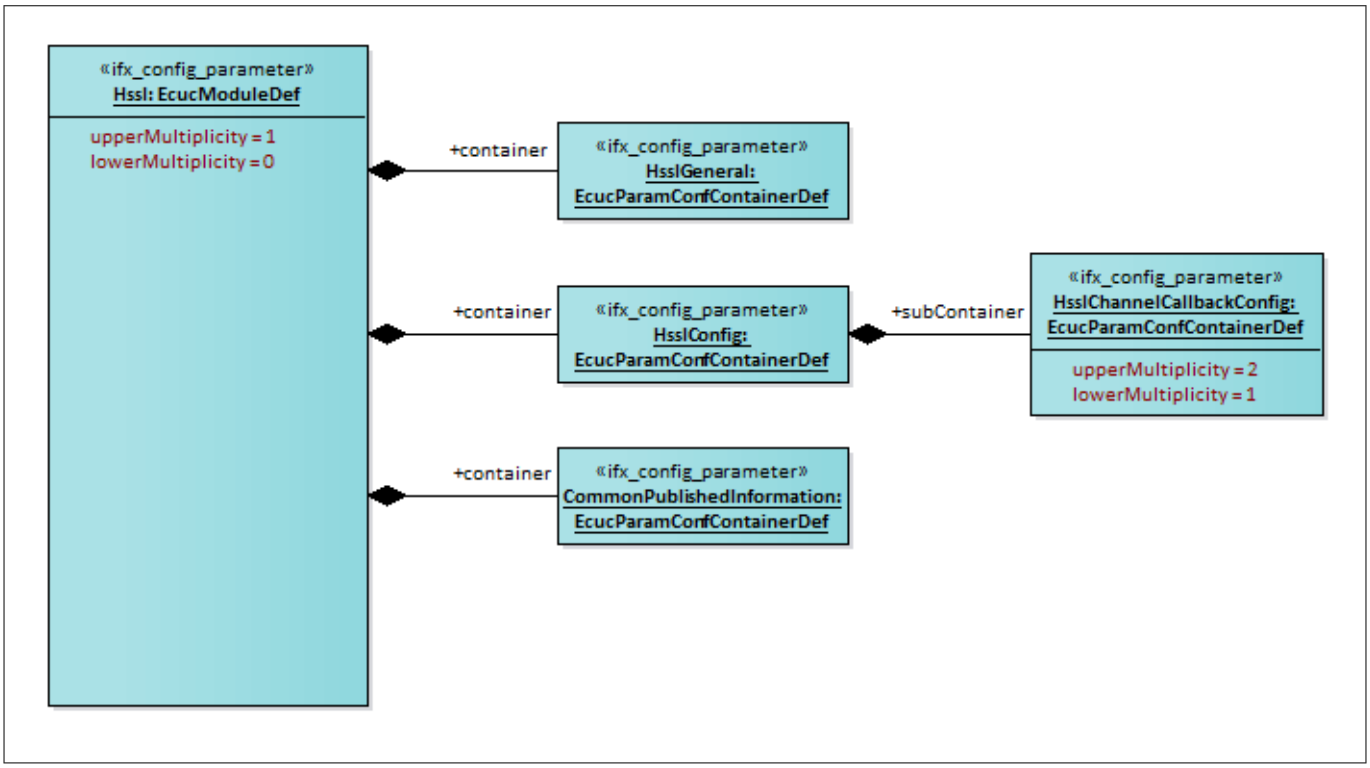


Figure 19 HSSL module configuration

1.3.1.1 Container: HsslGeneral

This container contains the general configuration parameters of the HSSL driver

1.3.1.1.1 HsslDevErrorDetect

Table 5 Specification for HsslDevErrorDetect

Name	HsslDevErrorDetect		
Description	Enables or disables the development error detection		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE: Enabled FALSE: Disabled		
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	-		

1.3.1.1.2 HsslVersionInfoApi

Table 6 Specification for HsslVersionInfoApi

Name	HsslVersionInfoApi		
Description	Enables or disables the Hssl_GetVersionInfo function		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE: Enabled FALSE: Disabled		
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	-		

1.3.1.1.3 HsslInitApimode

Table 7 Specification for HsslInitApiMode

Name	HsslInitApiMode		
Description	This configuration parameter defines the mode in which the HSSLInit API will be used		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_MCAL_SUPERVISOR HSSL_MCAL_USER		
Default value	HSSL_MCAL_SUPERVISOR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.4 HsslMultiSlaveMode

Table 8 Specification for HsslMultiSlaveMode

Name	HsslMultiSlaveMode		
Description	Enables or disables the multi slave mode		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE: Enabled		

Table 8 Specification for HsslMultiSlaveMode (continued)

	FALSE: Disabled		
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	-		

1.3.1.1.5 Hsslclockpredivider

Table 9 Specification for Hsslclockpredivider

Name	Hsslclockpredivider		
Description	This configuration parameter is used to set the clock predivider value		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x0000 0x3FFF		
Default value	0xFF		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.6 HsslInterfaceMode

Table 10 Specification for HsslInterfaceMode

Name	HsslInterfaceMode		
Description	This configuration parameter is used to select the master or slave interface		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	HSSL_MASTER HSSL_SLAVE		
Default value	HSSL_MASTER		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

Table 10 Specification for HsslInterfaceMode (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.7 HsslOperatingMode

Table 11 Specification for HsslOperatingMode

Name	HsslOperatingMode		
Description	This configuration parameter is used to select the operating mode in polling or interrupt mode		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	HSSL_POLLING_MODE HSSL_INTERUPT_MODE		
Default value	HSSL_POLLING_MODE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2 Container: HsslConfig

This container contains the module kernel specific configuration parameters.

Note: Availability of modules is based on the release notes

1.3.1.2.1 HsslInstanceID

Table 12 Specification for HsslInstanceID

Name	HsslInstanceID		
Description	This configuration parameter is used to select HSSL instance		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 : HSSL0 1: HSSL1		
Default value	0: HSSL0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

Table 12 Specification for HsslInstanceID (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.2 HsslCh2Mode

Table 13 Specification for HsslCh2Mode

Name	HsslCh2Mode		
Description	This configuration parameter is used to select channel 2 mode in streaming or command mode		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE : HSSL_CH2_STREAMING FALSE : HSSL_CH2_COMMAND		
Default value	FALSE : HSSL_CH2_COMMAND		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.3 HsslStreamingModeTx

Table 14 Specification for HsslStreamingModeTx

Name	HsslStreamingModeTx		
Description	Defines the Streaming Mode for Transmitter to be either Continuous or Streaming.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	TRUE : HSSL_STREAMING_MODE_SINGLE FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Default value	FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.4 HsslStreamingModeRx

Table 15 Specification for HsslStreamingModeRx

Name	HsslStreamingModeRx		
Description	Defines the Streaming Mode for Receiver to be either Continuous or Streaming.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE : HSSL_STREAMING_MODE_SINGLE FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Default value	FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.5 HsslTargetIDAddr

Table 16 Specification for HsslTargetIDAddr

Name	HsslTargetIDAddr		
Description	Defines the Address pointer containing the address of the memory location containing the unique ID data		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	0x0000		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.6 HsslEXICallbackFunction

Table 17 Specification for HsslEXICallbackFunction

Name	HsslEXICallbackFunction		
Description	This configuration parameter is used to define the function name for the user function for global interrupt.		
Multiplicity	1..1	Type	EcucStringParamDef

Table 17 Specification for HsslEXICallbackFunction (continued)

Range	NULL_PTR		
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	-		

1.3.1.2.7 HsslDmaMultiWriteChannelRef

Table 18 Specification for HsslDmaMultiWriteChannelRef

Name	HsslDmaMultiWriteChannelRef		
Description	This configuration parameter refers to the DmaConfiguration of DMA channel used by HSSL Multi write shall be provided as reference.		
Multiplicity	1..1	Type	EcucRefrenceParamDef
Range	Reference to parameter of type DmaChannel		
Default value	None		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.8 HsslDmaMultiWriteCallback

Table 19 Specification for HsslDmaMultiWriteCallback

Name	HsslDmaMultiWriteCallback		
Description	This configuration parameter is used to define the function name for the user function for multi write function.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

Table 19 Specification for HsslDmaMultiWriteCallback (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.9 HsslDmaMultiReadTxChannelRef

Table 20 Specification for HsslDmaMultiReadTxChannelRef

Name	HsslDmaMultiReadTxChannelRef		
Description	This configuration parameter refers to the DmaConfiguration of DMA channel used by HSSL Multi read for TX channel shall be provided as reference.		
Multiplicity	1..1	Type	EcucRefrenceParamDef
Range	Reference to parameter of type DmaChannel		
Default value	None		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.10 HsslDmaMultiReadRxChannelRef

Table 21 Specification for HsslDmaMultiReadRxChannelRef

Name	HsslDmaMultiReadRxChannelRef		
Description	This configuration parameter refers to the DmaConfiguration of DMA channel used by HSSL Multi read for Rx channel shall be provided as reference.		
Multiplicity	1..1	Type	EcucRefrenceParamDef
Range	Reference to parameter of type DmaChannel		
Default value	None		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.11 HsslDmaMultiReadCallback

Table 22 Specification for HsslDmaMultiReadCallback

Name	HsslDmaMultiReadCallback		
Description	This configuration parameter is used to define the function name for the user function for multi read function.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
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	-		

1.3.1.2.12 HsslAcessWindowStartAddrx

Table 23 Specification for HsslAcessWindowStartAddrx

Name	HsslAcessWindowStartAddrx (x = 0-3)		
Description	Defines the upper 24 bits of the start address of the corresponding access window		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 16384		
Default value	0x0000		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.13 HsslAcessWindowEndAddrx

Table 24 Specification for HsslAcessWindowEndAddrx

Name	HsslAcessWindowEndAddrx (x = 0-3)		
Description	Defines the upper 24 bits of the End address of the corresponding access window		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 16384		
Default value	0x0000		

Table 24 Specification for HsslAcessWindowEndAddrx (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.14 HsslAcessRuleWindowx

Table 25 Specification for HsslAcessRuleWindowx

Name	HsslAcessRuleWindowx (x = 0-3)		
Description	This configuration parameter represents the Access Rules for Window(x)		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	0 - HSSL_NO_ACCESS 1 - HSSL_READ_ACCESS 2 - HSSL_WRITE_ACCESS 3 - HSSL_READ_WRITE		
Default value	0 - HSSL_NO_ACCESS		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.15 HsslReferenceClock

Table 26 Specification for HsslReferenceClock

Name	HsslReferenceClock		
Description	This configuration parameter is used to select the reference clock frequency		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	0 – HSSL_10MHZ 1 – HSSL_20MHZ 2 – HSSL_40MHZ		
Default value	1 – HSSL_20MHZ		
Post-build variant value	FALSE	Post-build variant multiplicity	-

Table 26 Specification for HsslReferenceClock (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.16 HsslSystemClockDivider

Table 27 Specification for HsslSystemClockDivider

Name	HsslSystemClockDivider		
Description	This configuration parameter represents the system clock frequency divider		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_SYSCLK_DIV_1 HSSL_SYSCLK_DIV_2 HSSL_SYSCLK_DIV_4		
Default value	HSSL_SYSCLK_DIV_1		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.17 HsslMasterTxSpeed

Table 28 Specification for HsslMasterTxSpeed

Name	HsslMasterTxSpeed		
Description	This configuration parameter is used to select HSSL master transmitter speed		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_TX_LOW_SPEED HSSL_TX_HIGH_SPEED		
Default value	HSSL_TX_LOW_SPEED		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.18 HsslMasterRxSpeed

Table 29 Specification for HsslMasterRxSpeed

Name	HsslMasterRxSpeed		
Description	This configuration parameter is used to select HSSL master receiver speed		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_RX_LOW_SPEED HSSL_RX_MEDIUM_SPEED HSSL_RX_HIGH_SPEED		
Default value	HSSL_RX_LOW_SPEED		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.19 Container: HsslChannelCallbackConfig

This container contains callback user notification functions for the channel specific interrupts.

1.3.1.2.20 HsslChxCOKCallbackFunction

Table 30 Specification for HsslChxCOKCallbackFunction

Name	HsslChxCOKCallbackFunction (x = 0-3)		
Description	This configuration parameter is used to define the function name for the user call back notification function for command ok interrupt, where x represents the channel number.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
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	-		

1.3.1.2.21 HsslChxRDICallbackFunction

Table 31 Specification for HsslChxRDICallbackFunction

Name	HsslChxRDICallbackFunction (x = 0-3)		
Description	This configuration parameter is used to define the function name for the user call back notification function for read data interrupt, where x represents the channel number.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
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	-		

1.3.1.2.22 HsslChxTRGCallbackFunction

Table 32 Specification for HsslChxTRGCallbackFunction

Name	HsslChxTRGCallbackFunction (x = 0-3)		
Description	This configuration parameter is used to define the function name for the user call back notification function for trigger interrupt, triggered by the trigger command frame, where x represents the channel number.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
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	-		

1.3.1.2.23 HsslChxERRCallbackFunction

Table 33 Specification for HsslChxERRCallbackFunction

Name	HsslChxERRCallbackFunction (x = 0-3)
Description	This configuration parameter is used to define the function name for the user call back notification function for error interrupt, where x represents the channel number.

Table 33 Specification for HsslChxERRCallbackFunction (continued)

Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
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	-		

1.3.1.3 Container: CommonPublishedInformation

This container contains published information about vendor and versions.

1.3.1.3.1 ArMajorVersion

Table 34 Specification for ArMajorVersion

Name	ArMajorVersion		
Description	This parameter specifies AUTOSAR major release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	4		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.2 ArMinorVersion

Table 35 Specification for ArMinorVersion

Name	ArMinorVersion		
Description	This parameter specifies AUTOSAR minor release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	2		

Table 35 Specification for ArMinorVersion (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.3 ArPatchVersion

Table 36 Specification for ArPatchVersion

Name	ArPatchVersion		
Description	This parameter specifies AUTOSAR patch release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	2		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.4 SwMajorVersion

Table 37 Specification for SwMajorVersion

Name	SwMajorVersion		
Description	This parameter specifies software major release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.5 SwMinorVersion

Table 38 Specification for SwMinorVersion

Name	SwMinorVersion		
Description	This parameter specifies software minor release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.6 SwPatchVersion

Table 39 Specification for SwPatchVersion

Name	ArPatchVersion		
Description	This parameter specifies software patch release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.7 ModuleId

Table 40 Specification for ModuleId

Name	ModuleId		
Description	This parameter specifies module identification number.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 65535		
Default value	255		

Table 40 Specification for ModuleId (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.8 VendorId

Table 41 Specification for VendorId

Name	VendorId		
Description	This parameter specifies vendor identification number.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 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	-		

1.3.1.3.9 Release

Table 42 Specification for Release

Name	Release		
Description	This parameter indicates the TC3xx dice derivative used for implementation		
Multiplicity	1..1	Type	EcucStringParamDef
Range	String		
Default value	As per hardware derivative		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.2 Functions – Type definitions

This section describes all the type definitions that are used by APIs.

1.3.2.1 Hssl_DataTemplate

Table 43 Hssl_ DataTemplate

Name	`Hssl_ DataTemplate`	
Type	Structure	
File	Hssl.h	
Range	uint16* Data	Pointer to the data Range: [0x0..0xFFFFFFFF]
	uint32* Address	Pointer to the address Range: [0x0..0xFFFFFFFF]
Description	This Type definition is used to hold the address of read data buffer	

1.3.2.2 Hssl_channel

Table 44 Specification for Hssl_channel

Name	`Hssl_ChannelType`	
Type	Structure	
File	Hssl.h	
Range	uint8 Number	Channel number Range: [0..3]
	uint32 Timeout	Variable holding the timeout value of the channel Range: [0x0..0xFFFFFFFF]
Description	This type definition is used to hold the channel number and timeout of the channel	

1.3.2.3 Hssl_ReadDataTemplate

Table 45 Hssl_ Specification for Hssl_ReadDataTemplate

Name	`Hssl_ReadDataTemplateType`	
Type	Structure	
File	Hssl.h	
Range	uint32*	Address Range: [0x0..0xFFFFFFFF]
Description	This Type definition is used to hold the address of read data buffer	

1.3.2.4 Hssl_InstanceID

Table 46 Hssl_ InstanceID

Name	`Hssl_InstanceID`	
Type	Enumeration	
File	Hssl.h	
Range	HSSL0	HSSL instance id is 0
	HSSL1	HSSL instance id is 1
Description	This type definition is used to select the HSSL instance.	

1.3.2.5 Hssl_SlaveStatusType

Table 47 Hssl_ SlaveStatusType

Name	`Hssl_SlaveStatusType`	
Type	Enumeration	
File	Hssl.h	
Range	HSSL_SLAVE_ACTIVATED	Slave is activated
	HSSL_SLAVE_DEACTIVATED	Slave is deactivated
	HSSL_SLAVE_NOT_RESPONDING	Slave is not responding
	HSSL_SLAVE_NOT_SELECTED	Slave is not selected
Description	This type definition is used to select the status of the slave in multislave mode.	

1.3.2.6 Hssl_EventType

Table 48 Hssl_ EventType

Name	`Hssl_EventType`	
Type	Enumeration	
File	Hssl.h	
Range	HSSL_NO_EVENT	0U
	HSSL_WRITE_COMMAND_COMPLETED	0x2U
	HSSL_READ_COMMAND_COMPLETED	0x4U
	HSSL_TRIGGER_COMMAND_COMPLETED	0x8U
	HSSL_ERROR_NACK	0x10U
	HSSL_ERROR_TRANSACTION_TAG	0x20U
	HSSL_ERROR_TIMEOUT	0x40U

Table 48 Hssl_ EventType (continued)

	HSSL_ERROR_UNEXPECTED	0x80U
	HSSL_STREAM_BLOCK_TRANSMITTED	0x100U
	HSSL_STREAM_BLOCK_ERROR_OCCURED	0x200U
	HSSL_STREAM_BLOCK_RECEIVED	0x400U
	HSSL_SRI_BUS_ERROR	0x800U
	HSSL_PIE1_CHANNEL_NUMBER_CODE_ERROR	0x1000U
	HSSL_PIE2_DATA_LENGHT_ERROR	0x2000U
	HSSL_CRC_ERROR	0x4000U
Description	This type definition is used to indicate the event for notification functions.	

1.3.3 Functions - APIs

This section lists the APIs provided by HSSL driver along with a short description of the functionality.

1.3.3.1 Hssl_Init

Table 49 Specification for Hssl_Init API

Syntax	Std_ReturnType Hssl_Init (const Hssl_ConfigType* const Address)	
Service ID	0x3C	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Address	May be null pointer since it is pre compile module
Parameters (out)	None	
Parameters (in-out)	None	
Return	Std ReturnType Returns 'E_OK' if successful, 'E_NOT_OK' otherwise	
Description	Initializes the HSCT and HSSL module ,setting the Access window start and end address , access mode, target address registers and channel 2 mode	
Source	IFX	
Error handling	DET: HSSL_E_INV_PARAM: API is called with invalid input parameter. DEM: None Safety Errors: None	

Table 49 Specification for Hssl_Init API (continued)

	Runtime Errors: None
Configuration dependencies	-

1.3.3.2 Hssl_InitChannel

Table 50 Specification for Hssl_Initchannel API

Syntax	Std_ReturnType Hssl_InitChannel (const Hssl_InstanceID id, Hssl_ChannelType Channel, uint8 TimeoutErr, uint8 TransID, uint8 AckErr)	
Service ID	0x3D	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	channel	HSSL Channel
	TimeoutErr	Enable/Disable Timeout Error interrupt
	TransID	Enable/Disable Transaction ID Error interrupt
	AckErr	Enable/Disable Acknowledge Error interrupt
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	This API initializes the HSSL channels and also sets the interrupt.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_NOT_INITIALIZED: API is called without module initialization. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.3 Hssl-SetMode

Table 51 Specification for Hssl_SetMode API

Syntax	Std_ReturnType Hssl_SetMode (const Hssl_InstanceID id, uint8 Mode)	
Service ID	0x3A	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Mode	0 = Sleep, 1 = Initialize , 2 =Run
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	This API sets the mode of the HSSL module to the required mode.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_NOT_INITIALIZED: API is called without module initialization. HSSL_E_INV_PARAM: API is called with invalid mode argument. HSSL_E_INV_MODE: API is called with invalid mode for mode transition. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.4 Hssl_Reset

Table 52 Specification for Hssl_Reset API

Syntax	Std_ReturnType Hssl_Reset (const Hssl_InstanceID id)	
Service ID	0x3B	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)

Table 52 Specification for Hssl_Reset API (continued)

Parameters (out)	None
Parameters (in-out)	None
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful
Description	This API resets the HSCT and HSSL kernel and clears the status and error resisters.
Source	IFX
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_NOT_INITIALIZED: API is called without module initialization. DEM: None Safety Errors: None Runtime Errors: None
Configuration dependencies	-

1.3.3.5 Hssl_Write

Table 53 Specification for Hssl_Write API

Syntax	Std_ReturnType Hssl_Write (const Hssl_InstanceID id, const Hssl_DataTemplateType *WriteData, uint16 DataSize, Hssl_ChannelType Channel, InjectedError)	
Service ID	0x3E	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	writedata	Pointer to Hssl_Datatemplatetype structure which includes write address and data to be written
	Datasize	Size of the data to be written
	Channel	HSSL channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	

Table 53 Specification for Hssl_Write API (continued)

Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful
Description	Triggers the write command. In case of polling mode, Hssl_WriteAck API must be called to wait for acknowledgement. In case of interrupt mode, a notification is given to user after successful reception of acknowledgement
Source	IFX
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number, invalid data size. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL pointer argument. DEM: None Safety Errors: None Runtime Errors: None
Configuration dependencies	-

1.3.3.6 Hssl_WriteAck

Table 54 Specification for Hssl_WriteAck API

Syntax	Std_ReturnType Hssl_WriteAck (const Hssl_InstanceID id, Hssl_ChannelType Channel)	
Service ID	0x3F	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Waits for acknowledgment. The rationale behind adding separate polling function to poll for the acknowledgment: This reduces the blocking time of Hssl_Write API for back to back triggers for other HSSL channels.	
Source	IFX	
Error handling	DET:	

Table 54 Specification for Hssl_WriteAck API (continued)

	HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. DEM: None Safety Errors: None Runtime Errors: None
Configuration dependencies	-

1.3.3.7 Hssl_Read

Table 55 Specification for Hssl_Read API

Syntax	Std_ReturnType Hssl_Read (const Hssl_InstanceID id, const Hssl_DataTemplateType *DataAddress, uint16 DataSize, Hssl_ChannelType Channel, uint16 InjectedError)	
Service ID	0x40	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	dataAddress	Pointer to Hssl_DataTemplateType structure which includes read address
	dataSize	Size of data to be read
	Channel	HSSL channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' upon successful triggering of read command,otherwise 'E_NOT_OK' if unsuccessful.	
Description	Triggers the read command. In case of polling mode, The response for the read command can be obtained by calling Hssl_ReadRply API. In case of interrupt mode, a notification is given to user after successful read response is received.	
Source	IFX	

Table 55 Specification for Hssl_Read API (continued)

Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number, invalid data size. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL pointer argument. DEM: None Safety Errors: None Runtime Errors: None
Configuration dependencies	-

1.3.3.8 Hssl_ReadRply

Table 56 Specification for Hssl_ReadRply API

Syntax	Std_ReturnType Hssl_ReadRply (const Hssl_InstanceID id, Hssl_ChannelType Channel)	
Service ID	0x41	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	channel	HSSL Channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK', if response is received. E_NOT_OK, any error occurred.	
Description	Reads the response (data) for the read command triggered and updates the data buffer which is passed in Hssl_Read API. The rationale behind adding separate polling function to poll for the response: This reduces the blocking time of Hssl_Write API for back to back triggers for other HSSL channels.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. DEM: None Safety Errors: None	

Table 56 Specification for Hssl_ReadRply API (continued)

	Runtime Errors: None
Configuration dependencies	-

1.3.3.9 Hssl_Id

Table 57 Specification for Hssl_Id API

Syntax	Std_ReturnType Hssl_Id (const Hssl_InstanceID id, uint32 *StoreAddress, Hssl_ChannelType Channel)	
Service ID	0x42	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	StoreAddress	Pointer to the Address location/variable to store the ID received from target
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Sends ID Request Frame to target device. The received data (JTAG_ID) is used to identify the device capabilities.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.10 Hssl_Trigger

Table 58 Specification for Hssl_Trigger API

Syntax	Std_ReturnType Hssl_Trigger (const Hssl_InstanceID id, Hssl_ChannelType Channel)	
Service ID	0x4D	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	This API triggers the Trigger interrupt at the Target side	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.11 Hssl_StartStream

Table 59 Specification for Hssl_StartStream API

Syntax	Std_ReturnType Hssl_StartStream (const Hssl_InstanceID id, uint32 *SourceAddressStart, const uint32 *DestinationAddressStart, uint16 DataSize, uint16 InjectedError)
Service ID	0x43

Table 59 Specification for Hssl_StartStream API (continued)

Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	Hssl Instance Id (0:HSSL0 and 1: HSSL1)
	SourceAddressstart	Pointer to address containing start of data to be streamed. Note: The source address must be aligned to 256 bit.
	DestinationAddressstart	Pointer to address containing Destination start address of target. Note: The source address must be aligned to 256 bit.
	dataSize	Indicates the number of stream frames to transmit. Note: Each frame length is 256 bit.
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK', on successful stream operation trigger. E_NOT_OK, in case of any error.	
Description	Perform write stream operation. Read stream is not possible due to hardware limitation. Polling mode for stream operation is not supported. User must enable the interrupts to get the notification about the streaming completion.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid data size. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL arguments. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.12 Hssl_StopStream

Table 60 Specification for Hssl_StopStream API

Syntax	Std_ReturnType Hssl_StopStream

Table 60 Specification for Hssl_StopStream API (continued)

	(const Hssl_InstanceID id)	
Service ID	0x44	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Stops the ongoing streaming	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.13 Hssl_MultiWrite

Table 61 Specification for Hssl_MultiWrite API

Syntax	Std_ReturnType Hssl_MultiWrite (const Hssl_InstanceID id, const Hssl_DataTemplateType *WriteArray, uint16 DataSize, uint16 NumCmd, Hssl_ChannelType Channel, uint16 InjectedError)	
Service ID	0x45	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)

Table 61 Specification for Hssl_MultiWrite API (continued)

	WriteArray	Hssl_DataTemplateType structure which includes array containing write Address and Data to be written for each array record
	DataSIze	Size of data to be written
	NumCmd	Number of address / data pair to be transmitted. Note: the maximum size must not be greater than 2048.
	Channel	HSSL Channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Performs the Multi Write transfer using DMA. User must configure the notification function for configuration parameter “HsslDmaMultiWriteCallback” in order to get notified.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number and Invalid data size. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL arguments. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.14 Hssl_MultiRead

Table 62 Specification for Hssl_MultiRead API

Syntax	Std_ReturnType Hssl_MultiRead (const Hssl_InstanceID id, const Hssl_ReadDataTemplateType *ReadArray, const uint32 *Buffer, uint16 DataSize, uint16 NumCmd, Hssl_ChannelType Channel,

Table 62 Specification for Hssl_MultiRead API (continued)

	uint16 InjectedError)	
Service ID	0x46	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	Hssl Instance Id (0:HSSL0 and 1: HSSL1)
	ReadArray	Pointer to Hssl_ReadDataTemplateType structure which includes read Address
	Buffer	Store address
	dataSize	Size of data to be written
	NumCmd	Number of address / data pair to be transmitted. Note: the maximum size must not be greater than 2048.
	Channel	HSSL Channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Performs the Multi read transfer using DMA. User must configure the notification function for configuration parameter “HsslDmaMultiReadCallback” in order to get notified.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number and Invalid data size. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL arguments. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.15 Hssl_ActivateSlave

Table 63 Specification for Hssl_ActivateSlave API

Syntax	Std_ReturnType Hssl_ActivateSlave

Table 63 Specification for Hssl_ActivateSlave API (continued)

	(const Hssl_InstanceID id, uint8 Hssl_SlaveID, Hssl_SlaveStatusType *Hssl_SlaveStatus)	
Service ID	0x49	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_SlaveID	Select the slave based on the slave id in multi-slave mode
	Hssl_SlaveStatus	Status of the slave
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Activates the slave in multi slave mode. API Hssl_SelectSlave must be called before calling this API to select the slave. Once slave is selected, this API is necessary to call to activate the slave. Once slave is activated, any other operation can be performed.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL arguments. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.16 Hssl_DeactivateSlave

Table 64 Specification for Hssl_DeactivateSlave API

Syntax	Std_ReturnType Hssl_DeactivateSlave (const Hssl_InstanceID id, uint8 Hssl_SlaveID, Hssl_SlaveStatusType *Hssl_SlaveStatus)
Service ID	0x4a

Table 64 Specification for Hssl_DeactivateSlave API (continued)

Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_SlaveID	Select the slave based on the slave id in multi-slave mode
	Hssl_SlaveStatus	Status of the slave
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Deactivates the slave in multi slave mode. Once the slave is deactivated, It is must to call Hssl_SelectSlave and Hssl_ActivateSlave respectively before calling any other API.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL arguments. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.17 Hssl_SelectSlave

Table 65 Specification for Hssl_SelectSlave API

Syntax	Std_ReturnType Hssl_SelectSlave (const Hssl_InstanceID id, uint8 Hssl_SlaveID)	
Service ID	0x4B	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_SlaveID	Select the slave based on the slave id in multi-slave mode
Parameters (out)	None	
Parameters (in-out)	None	

Table 65 Specification for Hssl_SelectSlave API (continued)

Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful
Description	Selects the slave in multi slave mode. This API must be called before calling Hssl_ActivateSlave.
Source	IFX
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_PARAM: API is called with invalid slaveID. DEM: None Safety Errors: None Runtime Errors: None
Configuration dependencies	-

1.3.3.18 Hssl_GetGlobalError

Table 66 Specification for Hssl_GetGlobalError API

Syntax	Std_ReturnType Hssl_GetGlobalError (const Hssl_InstanceID id, uint32 *Hssl_GlobalErrFlg)	
Service ID	0X47	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_GlobalErrFlg	Pointer to store Hssl Global error flags value
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Reads the global error.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. HSSL_E_INV_POINTER: API is called with NULL pointer. DEM: None Safety Errors: None	

Table 66 Specification for Hssl_GetGlobalError API (continued)

	Runtime Errors: None
Configuration dependencies	-

1.3.3.19 Hssl_GetChannelError

Table 67 Specification for Hssl_GetChannelError API

Syntax	Std_ReturnType Hssl_GetChannelError (const Hssl_InstanceID id, const Hssl_ChannelType Channel, Hssl_ChannelErrorType *const ChannelError)	
Service ID	0X4C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
Parameters (out)	Channel	HSSL channel number
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Returns the channel error occurred for a specific channel.	
Source	IFX	
Error handling	DET: HSSL_E_INSTANCE_NOT_CONFIGURED: API is called with invalid HSSL instance ID. HSSL_E_INV_PARAM: API is called with invalid HSSL channel number HSSL_E_INV_MODE: API is called when the driver is not in RUN mode. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.3.20 Hssl_GetVersionInfo

Table 68 Specification for Hssl_GetVersionInfo API

Syntax	void Hssl_GetVersionInfo (

Table 68 Specification for Hssl_GetVersionInfo API (continued)

	Std_VersionInfoType *versioninfo)	
Service ID	0X48	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Versioninfo	Pointer to store the version information of this module
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	This service returns the version information of module.	
Source	IFX	
Error handling	DET: HSSL_E_INV_POINTER: API is called with NULL pointer. DEM: None Safety Errors: None Runtime Errors: None	
Configuration dependencies	-	

1.3.4 Notifications and callbacks

1.3.4.1 Hssl_DmaCallout

Table 69 Specification for Hssl_DmaCallout

Syntax	void Hssl_DmaCallout (const uint32 Channel, const Dma_ChEventType Event)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	HSSL channel to use
	Dma_ChEventType	An enumeration for representing the various channel events
Parameters (out)	None	
Parameters (in-out)	None	

Table 69 Specification for Hssl_DmaCallout (continued)

Return	None
Description	Dma callback is called after the successful transmission.
Source	IFX
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None
Configuration dependencies	-

1.3.4.2 Hssl_DmaErrCallout

Table 70 Specification for Hssl_DmaCallout

Syntax	void Hssl_IsrCOK (const Hssl_InstanceID id, uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	This function is called when the error is occurred during DMA transaction.	
Source	IFX	
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None	
Configuration dependencies	-	

1.3.5 Scheduled functions

The driver does not support any scheduled functions.

1.3.6 Interrupt service routines

The HSSL driver does not support any interrupts.

1.3.6.1 Hssl_IsrCOK

Table 71 Specification for Hssl_IsrCOK

Syntax	void Hssl_IsrCOK (const Hssl_InstanceID id, uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	The error free response frame triggers the COK interrupt	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslChxRDICallbackFunction (x = 0-3) Where x represents the channel number	

1.3.6.2 Hssl_IsrRDI

Table 72 Specification for Hssl_IsrRDI

Syntax	void Hssl_IsrRDI (const Hssl_InstanceID id, uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	

Table 72 Specification for Hssl_IsrRDI (continued)

Parameters (in-out)	None
Return	None
Description	The read frame triggers the RDI interrupt
Source	IFX
Error handling	-
Configuration dependencies	HsslChxRDICallbackFunction (x = 0-3) Where x represents the channel number

1.3.6.3 Hssl_IsrError

Table 73 Specification for Hssl_IsrError

Syntax	void Hssl_IsrError (const Hssl_InstanceID id, uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	The ISR gets called when the error interrupt is triggered	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslChxERRCallbackFunction (x = 0-3) Where x represents the channel number	

1.3.6.4 Hssl_IsrTrg

Table 74 Specification for Hssl_IsrTrg

Syntax	void Hssl_IsrTrg (const Hssl_InstanceID id, uint8 Channel

Table 74 Specification for Hssl_IsrTrg (continued)

	)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	ISR get called at target when trigger frame is arrived	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslChxTRGCallbackFunction (x = 0-3) Where x represents the channel number	

1.3.6.5 Hssl_IsrEXI

Table 75 Specification for Hssl_IsrEXI

Syntax	void Hssl_IsrEXI (const Hssl_InstanceID id)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	Hssl Instance Id (0:HSSL0 and 1: HSSL1)
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	ISR gets called when the global interrupt is triggered	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslEXICallbackFunction	

1.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

1.3.7.1 Development errors

The following table lists all the development errors reported by the driver.

Description	Source	Error code and value	Applicable APIs
API service is called before initialization API Hssl_Init.	IFX	HSSL_E_NOT_INITIALIZED= 0x01	Hssl_InitChannel Hssl_Write Hssl_WriteAck Hssl_Read Hssl_Id Hssl_Trigger Hssl_ReadRply Hssl_StartStream Hssl_MultiWrite Hssl_MultiRead
Service is called with NULL pointer.	IFX	HSSL_E_INV_POINTER=0x02	Hssl_Write Hssl_Read Hssl_Id Hssl_StartStream Hssl_DeactivateSlave Hssl_ActivateSlave Hssl_MultiWrite Hssl_MultiRead Hssl_GetGlobalError Hssl_GetVersionInfo
Service is called with invalid parameter.	IFX	HSSL_E_INV_PARAM=0x03	Hssl_SetMode Hssl_Init Hssl_StartStream Hssl_MultiWrite Hssl_MultiRead Hssl_SelectSlave
Service is called in an Invalid driver mode.	IFX	HSSL_E_INV_MODE=0x04	Hssl_SetMode
Service is called with unconfigured Hssl Instance.	IFX	HSSL_E_INSTANCE_NOT_CONFIGURED=0x05	Hssl_SetMode Hssl_Reset Hssl_StartStream Hssl_DeactivateSlave Hssl_ActivateSlave Hssl_GetGlobalError

1.3.7.2 Production errors

No production errors are reported in HSSL.

1.3.7.3 Safety errors

The driver does not report any safety errors.

1.3.7.4 Runtime errors

No runtime errors are reported in HSSL.

1.3.8 Deviations and limitations

1.3.8.1 Deviations

There are no deviations in the HSSL driver.

1.3.8.2 Limitations

There are no limitations for the HSSL driver.

1.3.9 Unsupported hardware features

The HSSL driver implements all the hardware features.

2 I2C driver

2.1 User information

2.1.1 Description

The I2C driver is responsible for initializing the I2C hardware module. It also provides services to write the data into the slave and read the data from the slave. It provides both synchronous (data transfer will occur without interrupt call) and asynchronous (data will be transferred by means of interrupt call) modes of read/write operation. The I2C driver is implemented as post-build variant or Variant PB.

The I2C driver does not support the Slave mode.

The driver supports:

- Master mode
- Standard mode up to 100 kbit/s (20 kbit/s - 100 kbit/s)
- Fast mode up to 400 kbit/s (100 kbit/s - 400 kbit/s)
- 7-bit I2C-bus addressing

2.1.2 Hardware-software mapping

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

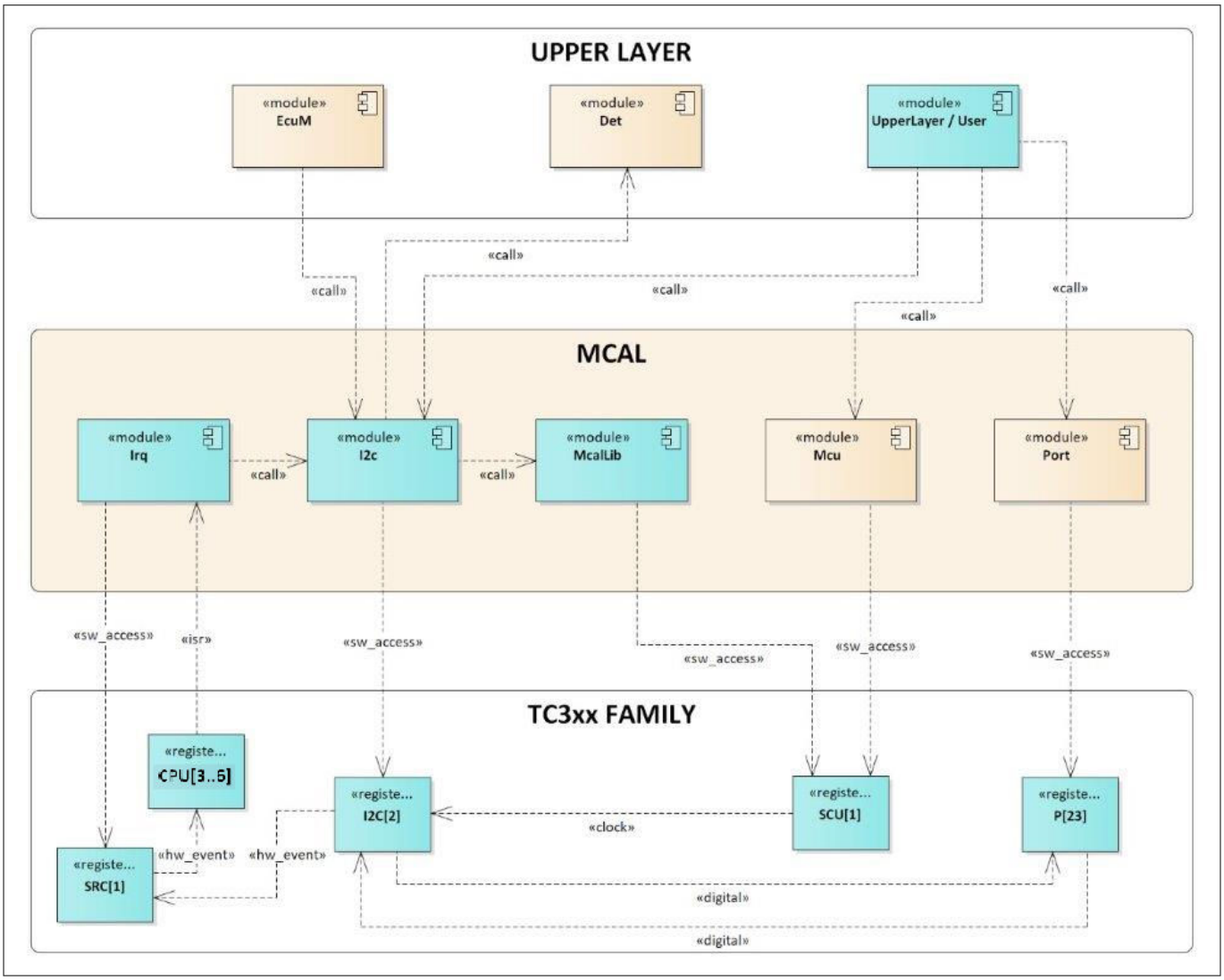


Figure 20 Mapping of hardware-software interfaces

2.1.2.1 I2C: primary hardware peripheral

Hardware functional features

The hardware features of each functional block configured by the driver are listed as follow:

- Master mode
- Standard mode up to 100 kbit/s (20kbit/s - 100kbit/s)
- Fast mode up to 400 kbit/s (100kbit/s - 400kbit/s)
- 7-bit I2C-bus addressing
- Prescaler for I2C kernel clock (from 0 to 255)
- Bit rate generation via fractional divider

Users of the hardware

The I2C driver exclusively utilizes the I2C module for its functionality.

Hardware diagnostic features

None, as there are no module specific hardware diagnostic features defined.

Hardware events

The following hardware events notified by flags are used in the I2C driver:

- TX_END flag upon transmission/reception complete
- RX flag upon switching from transmit to reception mode
- LSREQ_INT, SREQ_INT, LBREQ_INT, BREQ_INT flags for filling the FIFO with accurate number of data
- Error flags upon occurrence of errors during transmission and reception

  The module interrupt service requests are not processed by the I2C driver.

2.1.2.2 SCU: dependent hardware peripheral

Hardware functional features

The kernel_clk is set by the MCU driver from fI2C. The kernel_clk is required for maintaining the bitrate as specified by I2C protocol.

The interface_clk is directly connected to fSPB. The interface_clk is required to drive FIFO, SFR and Service Request Block.

Users of the hardware

The SCU module supplies clock for all the peripherals. However, it is only the MCU driver that is responsible for the configuration of the clock tree.

Hardware diagnostic features

The SMU alarms configured for SCU are not monitored by the I2C driver.

Hardware events

None.

2.1.2.3 Port: dependent hardware peripheral

Hardware functional features

The direction and mode selection of SCL, SDA pins of the I2C peripheral are configured by the Port driver.

Users of the hardware

The port pads are configured and used by the Port and DIO drivers.

Hardware diagnostic features

The SMU alarms configured for ports are not monitored by the I2C driver.

Hardware events

None.

2.1.2.4 SRC: dependent hardware peripheral

Hardware functional features

The I2C peripheral can trigger interrupts upon multitudes of events, varying for each I2C module. For these interrupts I2C driver depends on Interrupt Router.

Users of the hardware

No functional block of the Interrupt Router (IR) is administered by the I2C driver. The Interrupt Router is exclusively administered by the IRQ driver. The interrupt priorities and Type of Service (TOS) are configured centrally in the IRQ driver and hence the resource conflict is avoided. Individual module service request enabling is handled by the respective drivers.

Hardware diagnostic features

The SMU alarms configured for Interrupt Router are not monitored by the I2C driver.

Hardware events

None.

2.1.3 File structure

2.1.3.1 C file structure

This section provides details on the C files of the I2C driver.

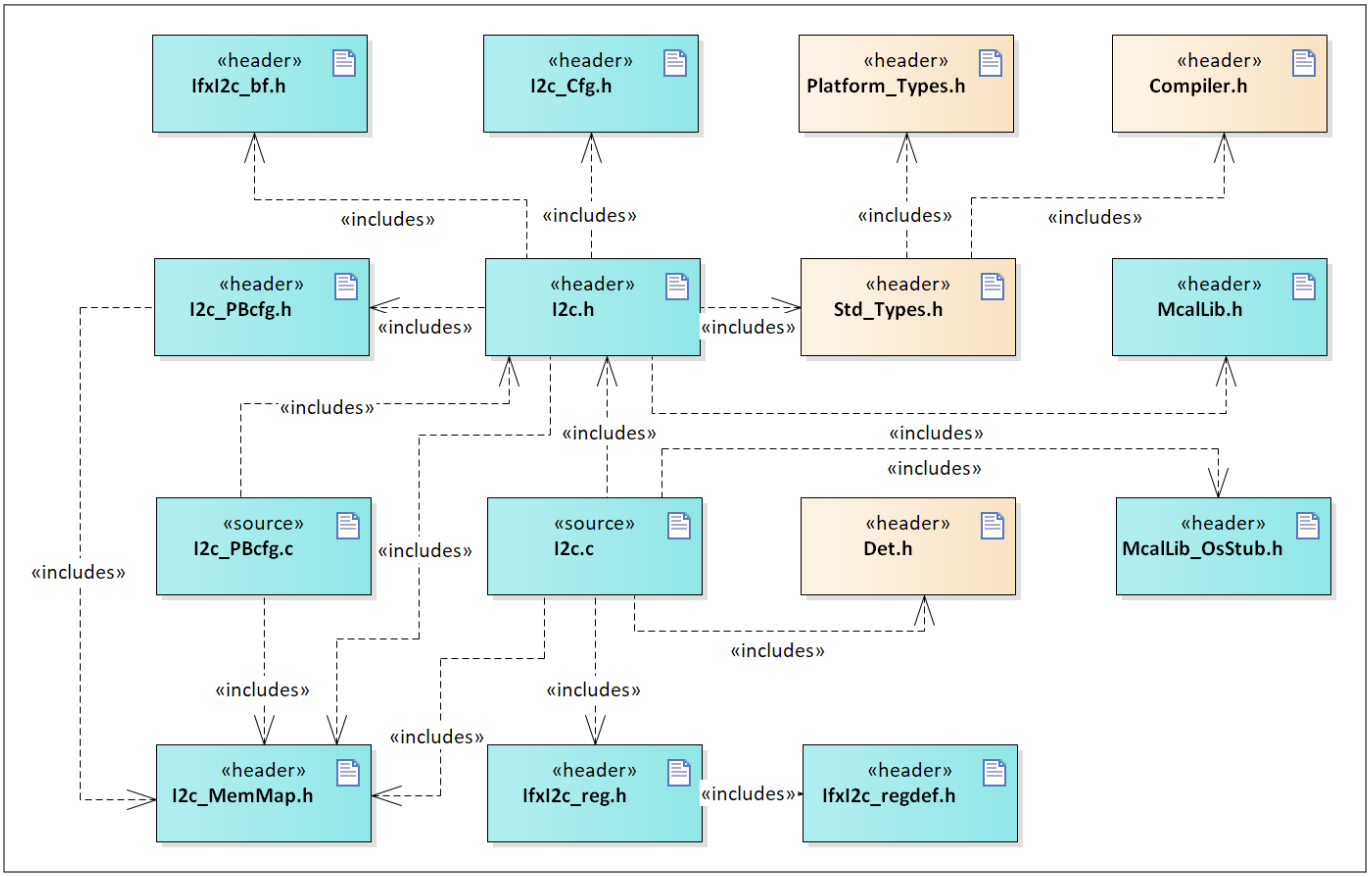


Figure 21 C file structure

Table 76 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
`Compiler.h`	Provides macros for the encapsulation of definitions and declarations
`Compiler_Cfg.h`	The file contains the module/component specific parameters (ptrclass and memclass) that are passed to the macros defined in Compiler.h
`Det.h`	Provides the exported interfaces of Development Error Tracer

Table 76 C file structure (continued)

File name	Description
`McalLib.h`	Header file (Static) defining prototypes of data structures and APIs of end-init and delay services and included by McalLib.c
`McalLib_OsStub.h`	Provides macros to support user mode of TriCore™
`I2c_MemMap.h`	Mapping of code and data (variables, constant variables) to specific memory sections
`I2c.h`	Contains macros, type definitions and function prototypes of the I2C driver
`I2c.c`	Implementation of I2C driver functionality
`I2c_Cfg.h`	The pre-compile configuration macros required for I2C driver implementation are present in this file
`I2c_PBcfg.h`	Contains I2C driver post build configuration parameter declaration
`I2c_PBcfg.c`	Contains I2C driver post build configuration parameters
`I2c_Irq.c`	IRQ file for handling all the I2C interrupts
`IfxI2c_bf.h`	Provides the Bit Mask, Length and Offset Macro definition for I2C registers
`IfxI2c_reg.h`	SFR header file for I2C
`IfxI2c_regdef.h`	Includes the register definition file for I2C

2.1.3.2 Code generator plugin files

The section provides details on the plugin files of the I2C driver.

Figure 22 Code generator plugin files

Table 77 Code generator plugin files

File name	Description
anchors.xml	Tresos anchors support file for the I2C driver
plugin.xml	Tresos plugin support file for the I2C driver
plugin.properties	Tresos plugin support file for the I2C driver
MANIFEST.MF	Tresos plugin support file containing the meta-data for I2C driver
ant_generator.xml	Tresos support file to generate and rename multiple Post-Build configuration when using variation point feature
I2c_Bswmd.arxml	AUTOSAR format module description file
I2c_Catalog.xml	AUTOSAR format catalog file
I2c.bmd	AUTOSAR format XML data model schema file (for each device)
I2c.m	Code template macro file for I2C driver
I2c.xdm	Tresos format XML data model schema file

2.1.4 Integration hints

This section lists the key points that an integrator or user of the I2C driver must consider.

2.1.4.1 Integration with AUTOSAR stack

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **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 `I2c_MemMap.h` file.

  The `I2c_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 depicted below.

```
#if defined I2C_START_SEC_VAR_CLEARED_QM_LOCAL_8
    /* User Pragma here */
    #undef I2C_START_SEC_VAR_CLEARED_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined I2C_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
    /* User Pragma here */
    #undef I2C_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined I2C_START_SEC_VAR_CLEARED_QM_LOCAL_UNSPECIFIED
    /* User Pragma here */
    #undef I2C_START_SEC_VAR_CLEARED_QM_LOCAL_UNSPECIFIED
    #undef MEMMAP_ERROR

#elif defined I2C_STOP_SEC_VAR_CLEARED_QM_LOCAL_UNSPECIFIED
    /* User Pragma here */
    #undef I2C_STOP_SEC_VAR_CLEARED_QM_LOCAL_UNSPECIFIED
    #undef MEMMAP_ERROR


#elif defined I2C_START_SEC_VAR_CLEARED_QM_LOCAL_32
    /* User Pragma here */
    #undef I2C_START_SEC_VAR_CLEARED_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined I2C_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
    /* User Pragma here */
    #undef I2C_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined I2C_START_SEC_CONST_QM_LOCAL_32
    /* User Pragma here */
    #undef I2C_START_SEC_CONST_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined I2C_STOP_SEC_CONST_QM_LOCAL_32
    /* User Pragma here */
    #undef I2C_STOP_SEC_CONST_QM_LOCAL_32
    #undef MEMMAP_ERROR

/* Code Section */
#elif defined I2C_START_SEC_CODE_QM_LOCAL
    /* User Pragma here */
    #undef I2C_START_SEC_CODE_QM_LOCAL
    #undef MEMMAP_ERROR

#elif defined I2C_STOP_SEC_CODE_QM_LOCAL
     /* User Pragma here */
    #undef I2C_STOP_SEC_CODE_QM_LOCAL
```

```
    #undef MEMMAP_ERROR
#endif
#if defined MEMMAP_ERROR
#error "I2c_MemMap.h, wrong pragma command"
#endif
```

- **DET**

  The DET module is a part of the AUTOSAR stack that handles all the development and runtime errors reported by the BSW modules. The I2C driver reports all the development errors to the DET module through the `Det_ReportError()` API. The user of the <Mod> driver must process all the errors reported to the DET module through the `Det_ReportError()` API.

  The `Det.h` and `Det.c` files are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.
- **DEM**

  DEM module is not required for the integration of the I2C driver.
- **SchM**

  SchM is not required for the integration of the I2C driver.
- **Safety error**

  I2C driver does not report any safety errors.
- **Notifications and callbacks**

  The I2C driver itself does not implement any notifications. However, the driver reports the completion of asynchronous transfers through notification functions. These notification functions can be configured by the user in Tresos for each I2cChannelConfiguration separately. Refer I2c_NotifFunctionPtrType for notification function prototype..
- **Operating system**

  The OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application.

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

2.1.4.2 Multicore and Resource Manager

I2C driver does not support execution on multiple cores in parallel.

2.1.4.3 MCU support

The I2C driver is dependent on MCU driver for clock generation. The fI2C defines the application clock frequency for the I2C Kernel. The fI2C which is derived from PLL2 (200MHz) is independent on fSPB and allows the I2C to operate at a constant baud rate (frequency). The required fI2C is 66.6MHz.But the current MCU driver supports only integer values of frequency. So the I2C driver is configured to 100MHz by considering divider value 2. This configuration can be done using `McuI2Cfrequency` in MCU module in Tresos. The frequency needs to be referenced in MCU module configuration parameter McuClockRefSelection which inturn will be referenced by I2C configuration in Tresos. Sample configuration for MCU driver is as follows:

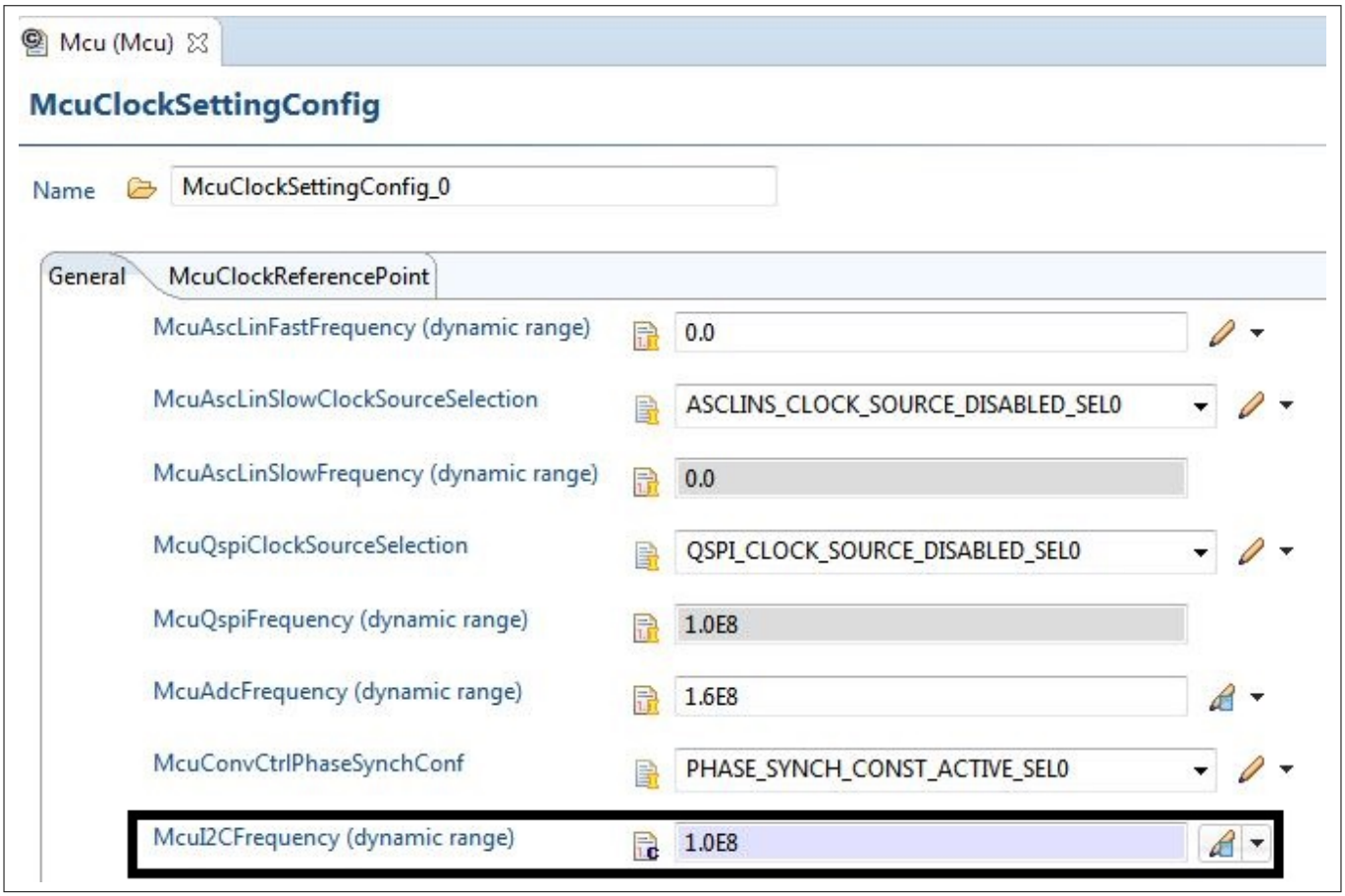
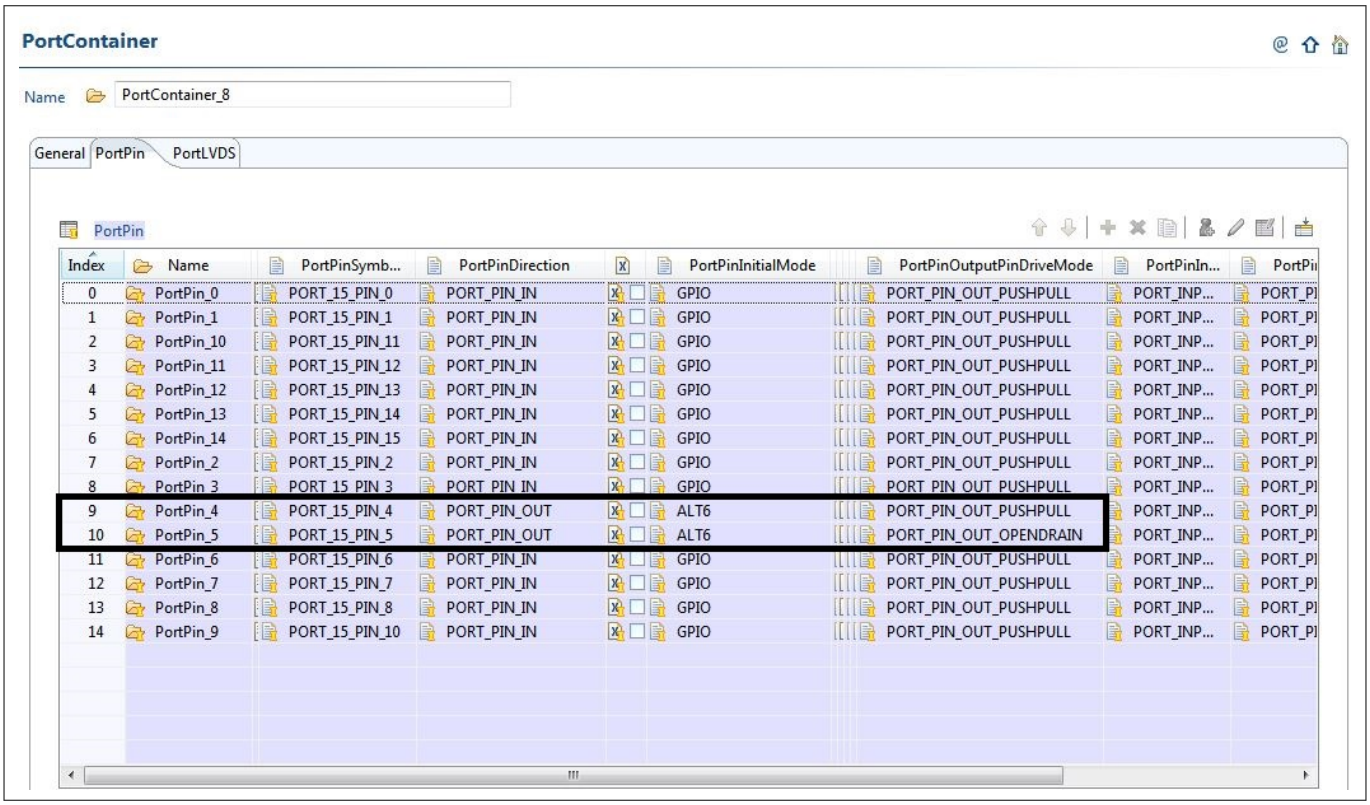


Figure 23 Mcu Configuration

2.1.4.4 Port support

The PORT driver configures the port pins of the entire microcontroller. The user must configure port pins used by the I2C driver, through the PORT configuration and initialize the port pins prior to invoking of I2C initialization

I2C driver requires two pins to be configured, SCL and SDA. SCL represents clock and SDA represents data. As I2C protocol allows multi-master, the SDA needs to be configured as Open-Drain in order to achieve wired-AND logic. Sample configuration for PORT driver is as follows:



Index	Name	PortPinSymb...	PortPinDirection		PortPinInitialMode	PortPinOutputPinDriveMode	PortPinIn...	PortPi
0	PortPin_0	PORT_15_PIN_0	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
1	PortPin_1	PORT_15_PIN_1	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
2	PortPin_10	PORT_15_PIN_11	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
3	PortPin_11	PORT_15_PIN_12	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
4	PortPin_12	PORT_15_PIN_13	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
5	PortPin_13	PORT_15_PIN_14	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
6	PortPin_14	PORT_15_PIN_15	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
7	PortPin_2	PORT_15_PIN_2	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
8	PortPin_3	PORT_15_PIN_3	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
9	PortPin_4	PORT_15_PIN_4	PORT_PIN_OUT		ALT6	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
10	PortPin_5	PORT_15_PIN_5	PORT_PIN_OUT		ALT6	PORT_PIN_OUT_OPENDRAIN	PORT_INP...	PORT_PI
11	PortPin_6	PORT_15_PIN_6	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
12	PortPin_7	PORT_15_PIN_7	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
13	PortPin_8	PORT_15_PIN_8	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI
14	PortPin_9	PORT_15_PIN_10	PORT_PIN_IN		GPIO	PORT_PIN_OUT_PUSHPULL	PORT_INP...	PORT_PI

Figure 24 Port Configuration

2.1.4.5 DMA support

I2C driver does not use any services provided by DMA driver.

2.1.4.6 Interrupt connections

The interrupt connections of the I2C driver are described in this section.

I2C module has three interrupt lines. The interrupt connections are described in this section.

Protocol Interrupt

This interrupt has seven sources. This interrupt is generated by the events transmission end, receive mode, Arbitration lost, not acknowledgement, address match, general call and master code. Service request line SRC_I2C0P is used for protocol interrupt. User must ensure that the interrupt service routine provided by I2C

driver is called when protocol interrupt occurs. A sample invocation for protocol interrupt for I2C0 is depicted as follows:

```
#if ((IRQ_I2C_P_SR0_PRIO > 0) || (IRQ_I2C_P_SR0_CAT == IRQ_CAT2))
#if ((IRQ_I2C_P_SR0_PRIO > 0) && (IRQ_I2C_P_SR0_CAT == IRQ_CAT1))
IFX_INTERRUPT(I2C0P_ISR, 0, IRQ_I2C_P_SR0_PRIO)
#elif IRQ_I2C_P_SR0_CAT == IRQ_CAT2
ISR(I2C0P_ISR)
#endif
{
  /* Enable Global Interrupts */
  ENABLE();
  /* Call Protocol interrupt funtion */
  I2c_IsrI2cProtocol(I2C_ZERO);
}
#endif
```

Error Interrupt

This interrupt has four sources. This interrupt is generated by any of the events transmit overflow, transmit underflow, receive underflow. Service request line SRC_I2C0ERR is used for error interrupt. User must ensure that the interrupt service routine provided by I2c driver is called when error interrupt occurs. A sample invocation for error interrupt for I2C0 is depicted as follows:

```
#if ((IRQ_I2C_ERR_SR0_PRIO > 0) || (IRQ_I2C_ERR_SR0_CAT == IRQ_CAT2))
#if ((IRQ_I2C_ERR_SR0_PRIO > 0) && (IRQ_I2C_ERR_SR0_CAT == IRQ_CAT1))
IFX_INTERRUPT(I2C0E_ISR, 0, IRQ_I2C_ERR_SR0_PRIO)
#elif IRQ_I2C_ERR_SR0_CAT == IRQ_CAT2
ISR(I2C0E_ISR)
#endif
{
  /* Enable Global Interrupts */
  ENABLE();
  /* Call error interrupt funtion */
  I2c_IsrI2cError(I2C_ZERO);
}
#endif
```

Data transfer Interrupt

This interrupt has four sources. This interrupt is generated by any of the events burst request, last burst request, single request, last single request. Service request line SRC_I2C0DTR is used for data transfer interrupt. User

must ensure that the interrupt service routine provided by I2C driver is called when data transfer interrupt occurs. A sample invocation for data transfer interrupt for I2C0 is depicted as follows:

```
#if ((IRQ_I2C_EXIST == STD_ON))
#if ((IRQ_I2C_DTR_SR0_PRIO > 0) || (IRQ_I2C_DTR_SR0_CAT == IRQ_CAT2))
#if ((IRQ_I2C_DTR_SR0_PRIO > 0) && (IRQ_I2C_DTR_SR0_CAT == IRQ_CAT1))
IFX_INTERRUPT(I2C0DTR_ISR, 0, IRQ_I2C_DTR_SR0_PRIO)
#elif IRQ_I2C_DTR_SR0_CAT == IRQ_CAT2
ISR(I2C0DTR_ISR)
#endif
{
  /* Enable Global Interrupts */
  ENABLE();
  /* Call data transfer interrupt funtion */
  I2c_IsrI2cDtr(I2C_ZERO);
}
#endif
```

2.1.4.7 Example usage

Examples of I2C driver API usage are as follows:

Configuring the driver

I2C driver must be configured before usage and configuration files are generated and made available during software build process.

To configure I2c driver following guidelines should be followed properly.

- In MCU driver, configure the system clock.
- In PORT driver, configure SCL and SDA lines.
- In I2C driver, select the required speed mode and addressing mode of the slave.
- For I2C to work in asynchronous mode, asynchronous communication, configure the interrupt priority, type of service and interrupt type in IRQ module.

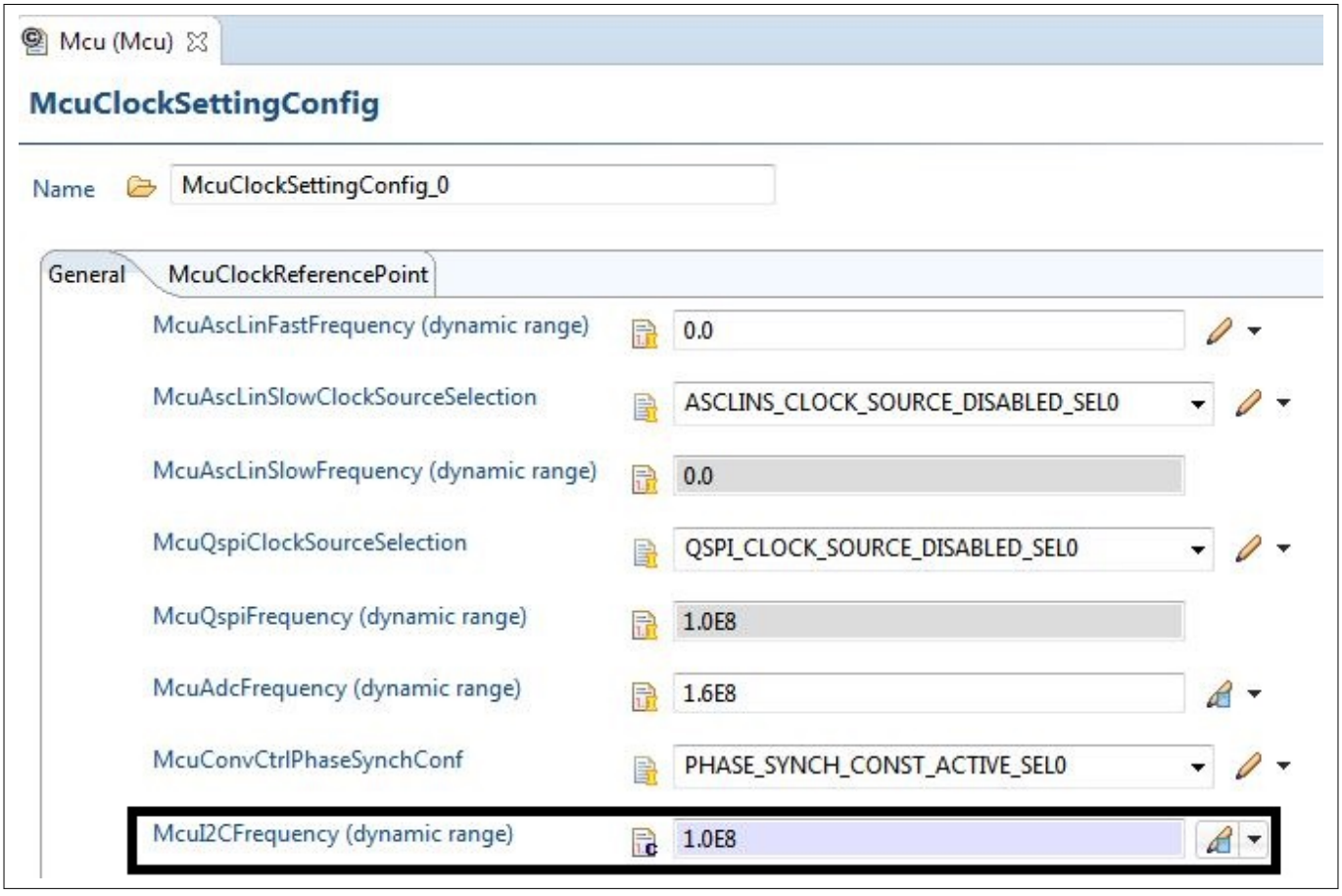


Figure 25 Mcu Configuration

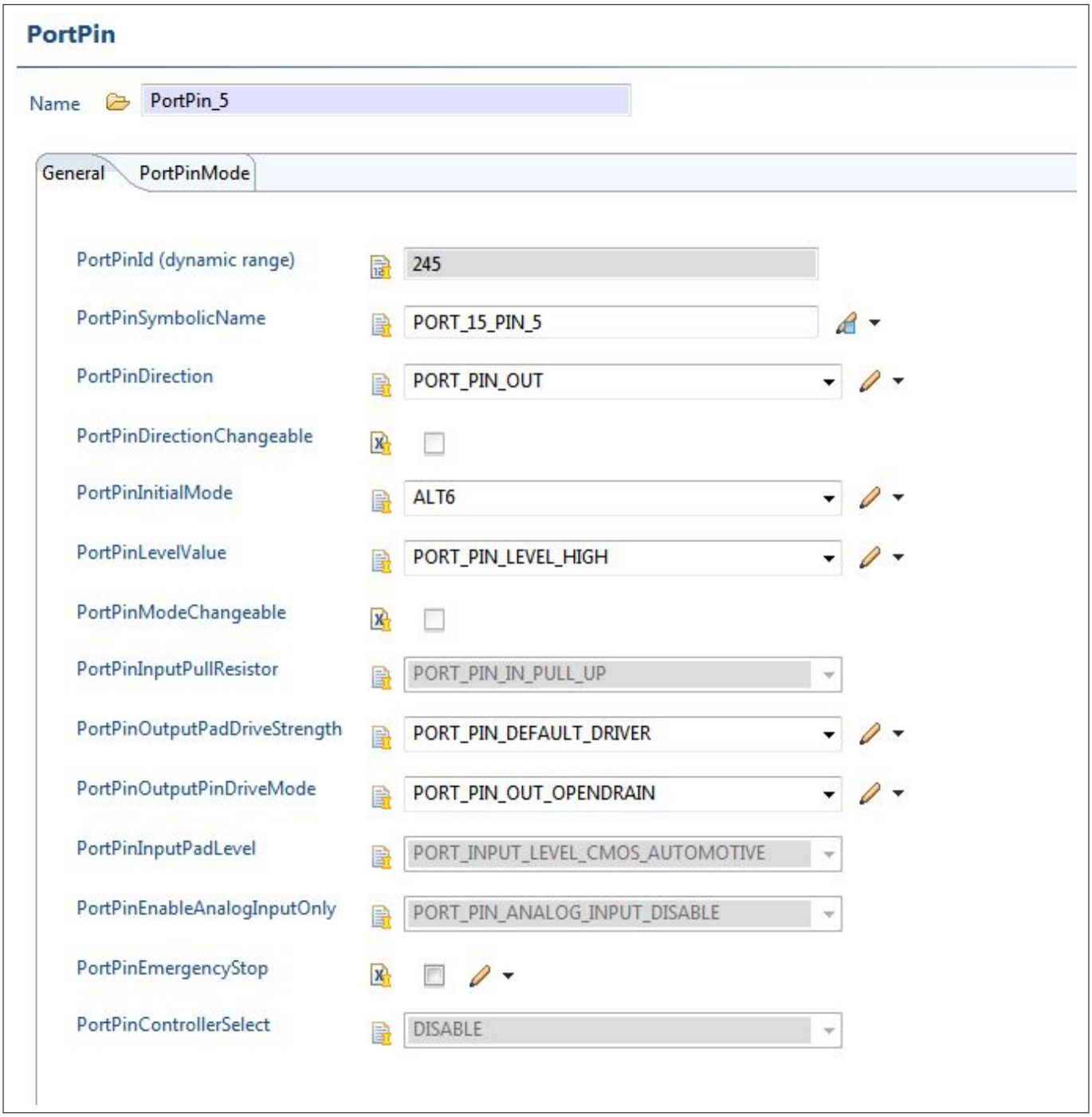


Figure 26 Port Pin Configuration

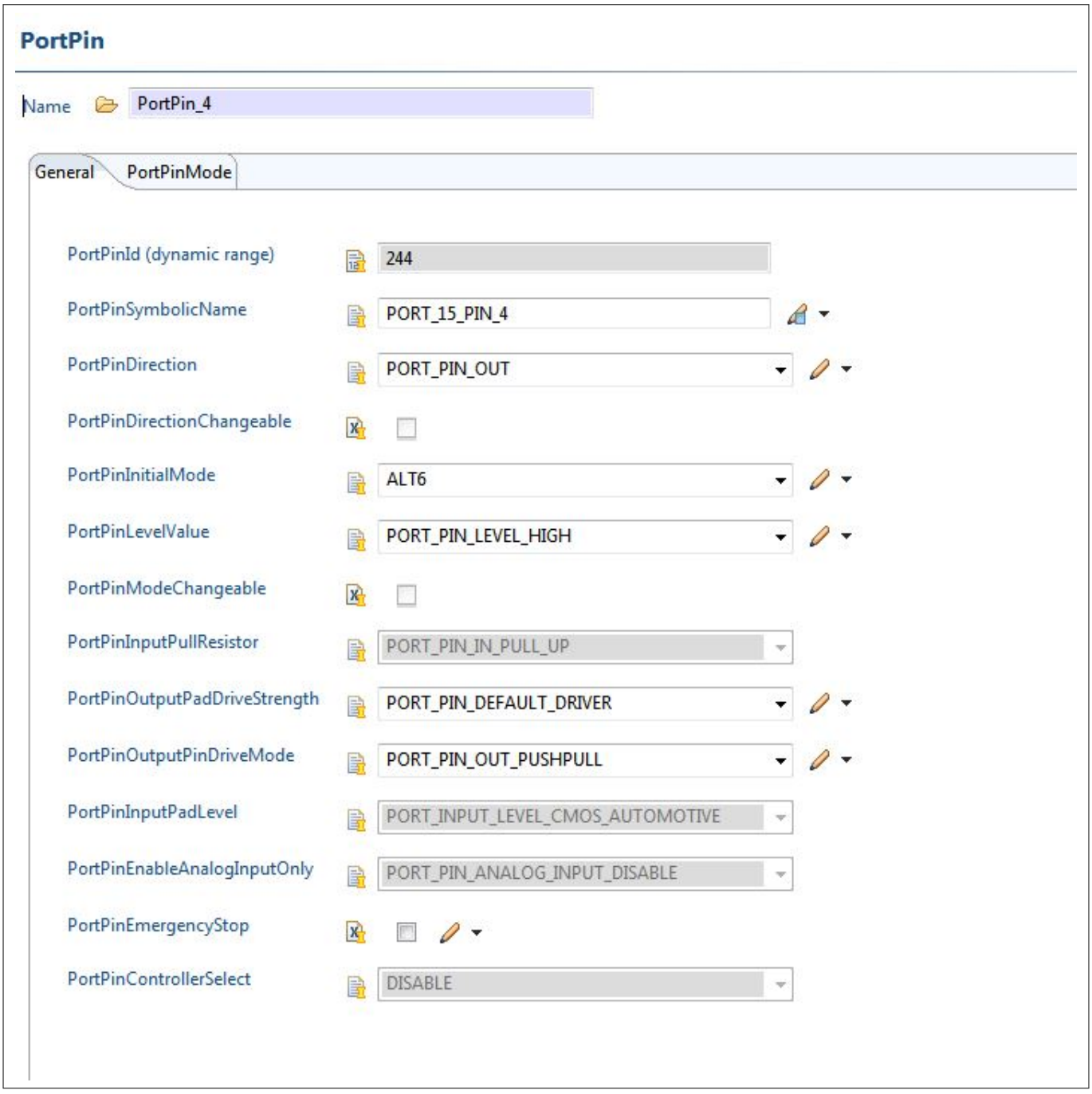


Figure 27 Port Pin Configuration

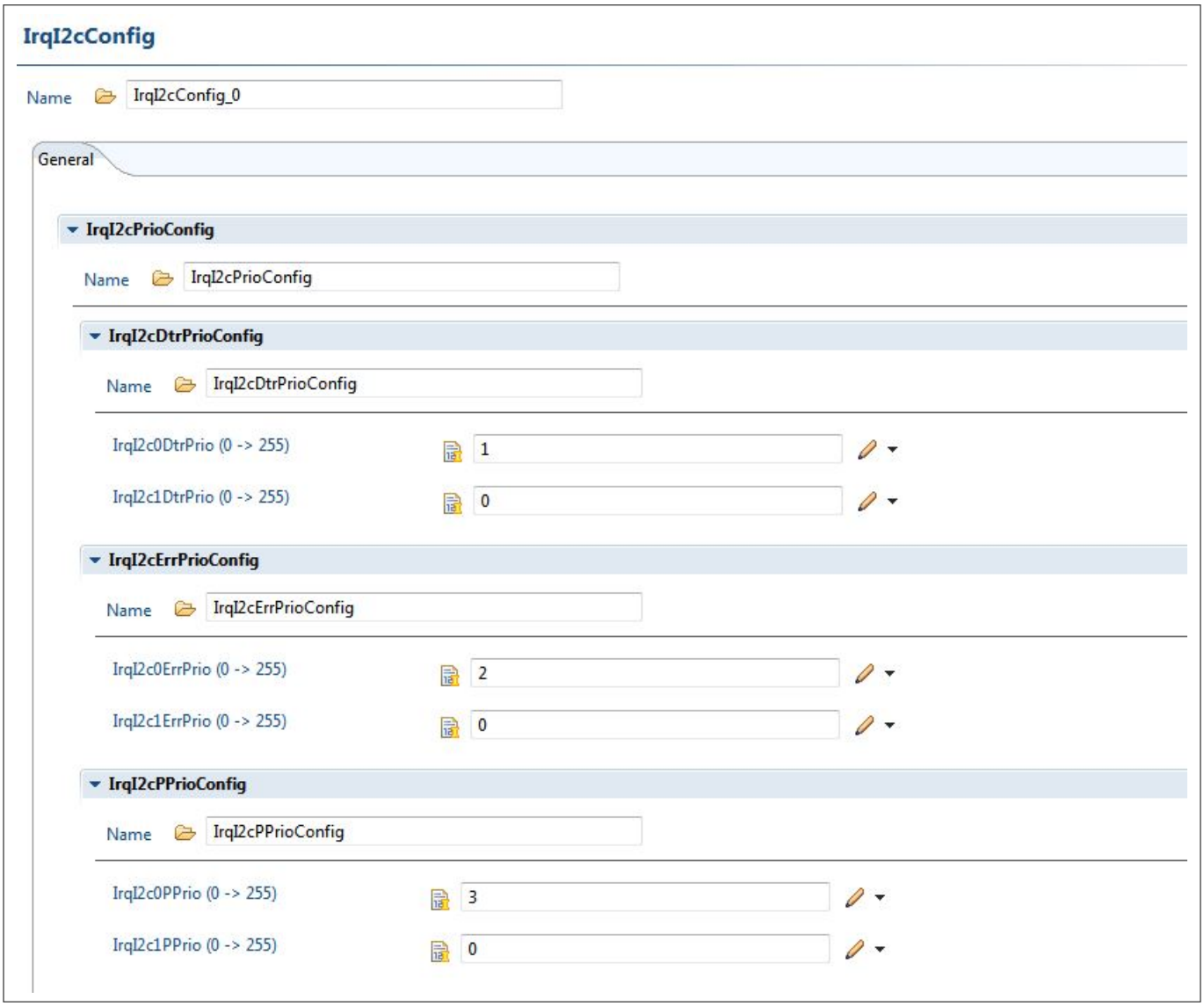


Figure 28 Irq Configuration

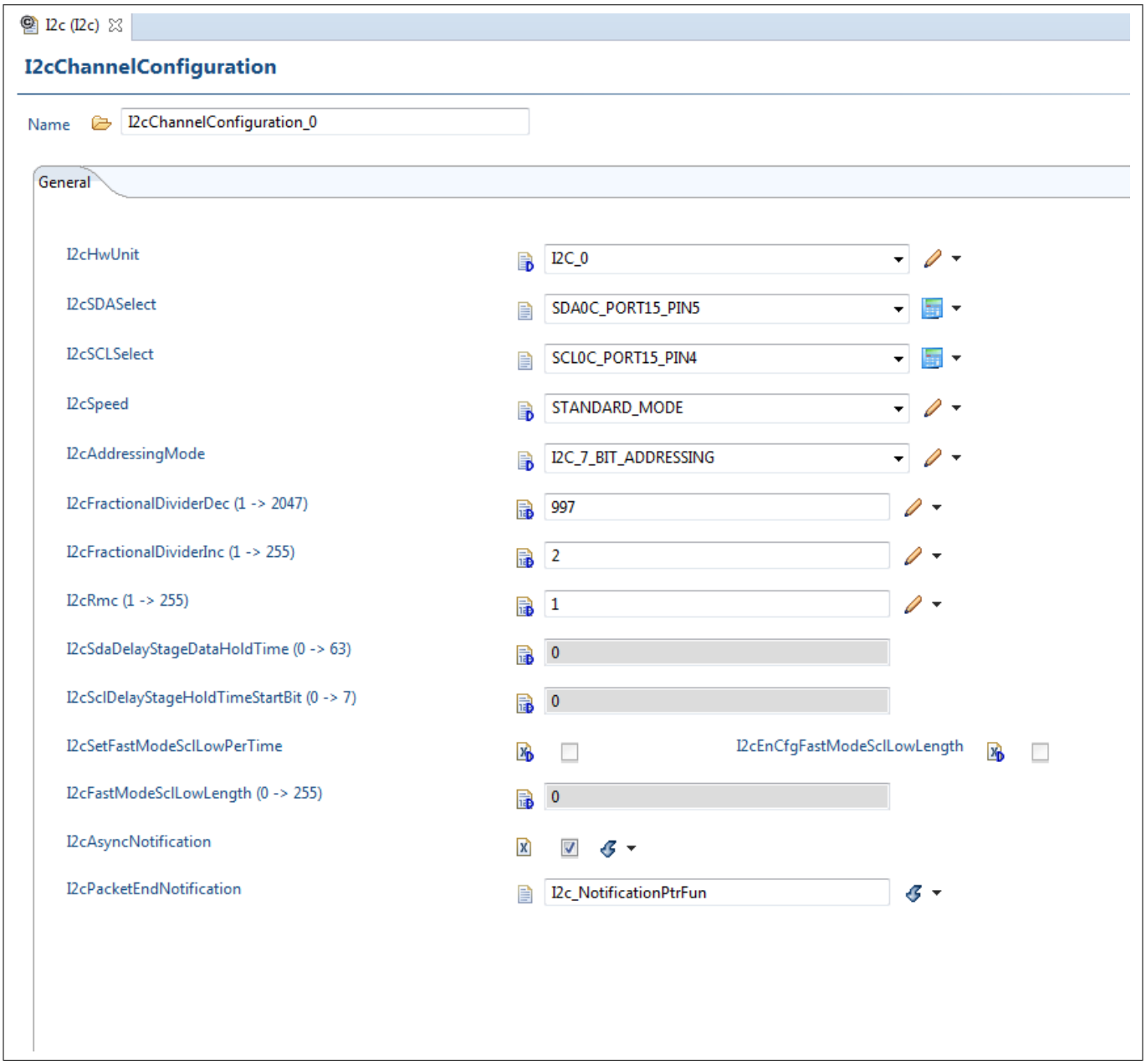


Figure 29 I2c Channel Configuration

Initializing the driver

The code sequence for initializing I2C driver is as follows.

```
#include "McalLib.h"
#include "I2c.h"
#include "Mcu.h"
#inclu"e "Port.h"
#include "IfxSrc_reg.h"
#include "Irq.h"
/* Mcu initialization */
 Mcu_Init(&Mcu_Config);
 Mcu_InitClock( 0 );
 while(Mcu_GetPllStatus() != MCU_PLL_LOCKED)
 {
 };
 Mcu_DistributePllClock();

 /* Port initialization */
 Port_Init(&Port_Config);

 /* I2c initialization */
 I2c_Init(&I2c_Config);
```

Enabling interrupt for Asynchronous mode

The code sequence for enabling interrupts for I2c driver is as follows.

```
IrqI2c_Init();
SRC_I2C0DTR.B.SRE=0x1;
SRC_I2C0ERR.B.SRE=0x1;
SRC_I2C0P.B.SRE=0x1;
```

Transmitting data - Synchronous mode

The code sequence for transmitting data through I2C bus is as follows.

```
#define I2C_NUMBER_OF_BYTES  24
uint8 Buffer[I2C_NUMBER_OF_BYTES];
uint8 Adderss_Buffer[1];
uint16 LoopCount;
uint8 data = 0;
Adderss_Buffer[0]= 0x0;
for (LoopCount=I2C_ZERO;LoopCount<I2C_NUMBER_OF_BYTES;LoopCount++)
{
  Buffer[LoopCount] = data;
  data++;
}
/* Initialize the driver */
I2c_Init(&I2c_Config);
/* Transmit data */
I2c_SyncWrite(0x0U,Buffer,I2C_NUMBER_OF_BYTES,0x50U);
I2c_SyncWrite(0x0U, Adderss_Buffer,0x1U,0x50U);
```

Receiving data - Synchronous mode

The code sequence for receiving data through I2C bus is as follows.

```
#define I2C_NUMBER_OF_BYTES  24
uint8 Buffer[I2C_NUMBER_OF_BYTES];
uint8 BufferRead[I2C_NUMBER_OF_BYTES];
uint8 Adderss_Buffer[1];
uint16 LoopCount;
uint8 data = 0;
Adderss_Buffer[0]= 0x0;
for (LoopCount=I2C_ZERO;LoopCount<I2C_NUMBER_OF_BYTES;LoopCount++)
{
  Buffer[LoopCount] = data;
  data++;
}
/* Initialize the driver */
I2c_Init(&I2c_Config);
/* Transmit data */
I2c_SyncWrite(0x0U,Buffer,I2C_NUMBER_OF_BYTES,0x50U);
I2c_SyncWrite(0x0U, Adderss_Buffer,0x1U,0x50U);
/* Receive data */
I2c_SyncRead(0x0U,BufferRead,I2C_NUMBER_OF_BYTES,0x50U);
```

Transmitting data - Asynchronous mode

The code sequence for transmitting data through I2C bus is as follows.

```
volatile uint32 count = 0;
#define I2C_NUMBER_OF_BYTES  24

/* notification function */
void I2c_NotifFunctionPtrfun(I2c_ErrorType ErrorId)
{
  count++;
}
uint8 Buffer[I2C_NUMBER_OF_BYTES];
uint8 Adderss_Buffer[1];
uint16 LoopCount;
uint8 data = 0;
Adderss_Buffer[0]= 0x0;
for (LoopCount=I2C_ZERO;LoopCount<I2C_NUMBER_OF_BYTES;LoopCount++)
{
  Buffer[LoopCount] = data;
  data++;
}
/* Initialize the driver */
I2c_Init(&I2c_Config);
/* Transmit data */
I2c_AsyncWrite(0x0U,Buffer,I2C_NUMBER_OF_BYTES,0x50U);
While(count == 0);
I2c_AsyncWrite(0x0U, Adderss_Buffer,0x1U,0x50U);
```

Receiving data - Asynchronous mode

The code sequence for receiving data through I2C bus is as follows.

```
volatile uint32 count = 0;
#define I2C_NUMBER_OF_BYTES  24

/* notification function */
void I2c_NotifFunctionPtrfun(I2c_ErrorType ErrorId)
{
  count++;
}
uint8 Buffer[I2C_NUMBER_OF_BYTES];
uint8 Adderss_Buffer[1];
uint16 LoopCount;
uint8 data = 0;
Adderss_Buffer[0]= 0x0;
for (LoopCount=I2C_ZERO;LoopCount<I2C_NUMBER_OF_BYTES;LoopCount++)
{
  Buffer[LoopCount] = data;
  data++;
}
/* Initialize the driver */
I2c_Init(&I2c_Config);
/* Transmit data */
I2c_AsyncWrite(0x0U,Buffer,I2C_NUMBER_OF_BYTES,0x50U);
While(count == 0);
I2c_AsyncWrite(0x0U, Adderss_Buffer,0x1U,0x50U);
While(count == 1);
I2c_AsyncRead(0x0U,BufferRead,I2C_NUMBER_OF_BYTES,0x50U);
While(1);
```

Notification Function

When I2C is communicating asynchronously it will provide a notification with error id, if notification is configured by user in Tresos.

The code sequence for notification function is as follows.

```
void I2c_NotifFunctionPtrfun(I2c_ErrorType ErrorId)
{
 /*User Code Here*/
}
```

2.1.5 Key architectural considerations

The key architectural considerations are as follows:

2.1.5.1 FIFO configuration

I2C uses a FIFO for temporary storing of data. The FIFO is 8 level 32 bit FIFO. The FIFO can be configured for burst mode or single request mode. The current I2C driver uses the burst mode and the burst is configured to be

4 words with a word alignment of 1 byte. Therefore, an interrupt will be generated every time FIFO is emptied by 4 words. For data size which is less than burst size single request interrupt will be generated. In case of Asynchronous operation, this interrupt is serviced through ISR and in case of Synchronous operation, polling for the status of the request is to be done. To issue burst and single requests, the FIFO needs to be configured as flow control, that is, if the peripheral is configured as flow control, then automatically a signal is generated as soon as FIFO has empty space of configured burst size. The CRBC (Clear Request Behavior Configuration) bit is disabled as the I2C driver uses burst mode. Disabling this bit signifies that driver will clear the burst requests that are generated.

2.1.5.2 Peripheral configuration

The SONA (Stop On Not Acknowledgement) bit is enabled by default. This bit signifies that the I2C kernel will put a STOP condition when not acknowledged. The SOPE (Stop On Packet End) bit is enabled by default. This bit signifies that the I2C kernel will put a STOP condition when transmitted. In both cases the kernel will change its state to LISTENING.

2.2 Assumptions of Use (AoU)

There are no AoUs for the driver.

2.3 Reference information

2.3.1 Configuration interfaces

This section details the configuration container hierarchy along with their configuration parameters.

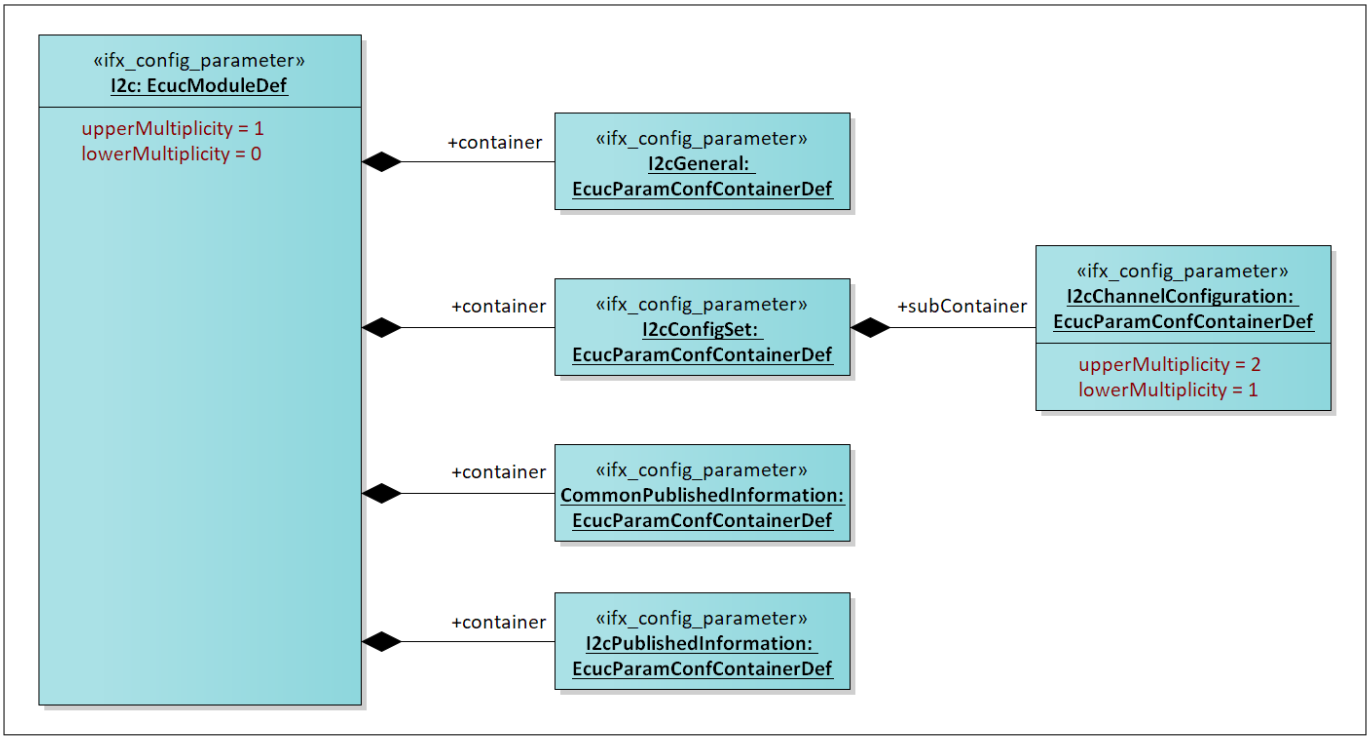


Figure 30 Container hierarchy along with their configuration parameters

2.3.1.1 Container: I2c

Configuration of the I2C (I2c driver) module

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.2 Container: I2cPublishedInformation

This container contains the published information of the I2C driver, that is, the maximum hardware units available in the configured silicon.

Post-Build Variant Multiplicity: FALSE

Multiplicuration Class: Pre-Compile

2.3.1.2.1 I2cMaxHwUnit

Table 78 Specification for I2cMaxHwUnit

Name	`I2cMaxHwUnit`		
Description	The parameter represents maximum supported I2c hardware units.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

Table 78 Specification for I2cMaxHwUnit (continued)

Range	0 - 255		
Default value	Reference to number of available hardware unit.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.3 Container: I2cConfigSet

This container contains the Channel configuration of the I2C driver. This container is a Multiple Configuration Container, i.e. this container and its sub-containers exist once per configuration set.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

2.3.1.4 Container: I2cChannelConfiguration

This sub-container contains configuration for individual channel. This channel contains the information required for required baud rate, port pin selection and I2c Speed selection.

Post-Build Variant Multiplicity: FALSE

Multipliciration Class: Pre-Compile

2.3.1.4.1 I2cHwUnit

Table 79 Specification for I2cHwUnit

Name	`I2cHwUnit`		
Description	This parameter selects the hardware unit that is to be assigned to the channel.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	I2C_0 I2C_1		
Default value	I2C_0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.2 I2cSpeed

Table 80 Specification for I2cSpeed

Name	`I2cSpeed`		
Description	This parameter defines the data transfer speed of the external device.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	STANDARD_MODE FAST_MODE HIGH_SPEED_MODE		
Default value	STANDARD_MODE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.3 I2cAddressingMode

Table 81 Specification for I2cAddressingMode

Name	`I2cAddressingMode`		
Description	This parameter defines the Addressing mode (7/10 bit) required to address the slave.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	I2C_7_BIT_ADDRESSING I2C_10_BIT_ADDRESSING		
Default value	I2C_7_BIT_ADDRESSING		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.4 I2cFractionalDividerDec

Table 82 Specification for I2cFractionalDividerDec

Name	I2cFractionalDividerDec		
Description	This parameter contains DEC value of the fractional divider.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1-2047		
Default value	997		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.5 I2cFractionalDividerInc

Table 83 Specification for I2cFractionalDividerInc

Name	I2cFractionalDividerInc		
Description	This parameter defines the data transfer speed of the external device.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1-255		
Default value	2		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.6 I2cRmc

Table 84 Specification for I2cRmc

Name	I2cRmc		
Description	This parameter contains Rmc value of the CLC1 register.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1-255		
Default value	1		

Table 84 Specification for I2cRmc (continued)

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.7 I2cSclDelayStageHoldTimeStartBit

Table 85 Specification for I2cSclDelayStageHoldTimeStartBit

Name	`I2cSclDelayStageHoldTimeStartBit`		
Description	This parameter contains SCL delay stages for Hold time start (Restart) bit.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-7		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.8 I2cSdaDelayStageDataHoldTime

Table 86 Specification for I2cSdaDelayStageDataHoldTime

Name	`I2cSdaDelayStageDataHoldTime`		
Description	This parameter contains SDA delay stage for data hold time.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-63		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.9 I2cSetFastModeSclLowPerTime

Table 87 Specification for I2cSetFastModeSclLowPerTime

Name	I2cSetFastModeSclLowPerTime		
Description	This parameter enables Standard or Fast mode SCL Low period timing.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.10 I2cFastModeSclLowLength

Table 88 Specification for I2cFastModeSclLowLength

Name	I2cFastModeSclLowLength		
Description	This parameter contains SCL Low Period Length in Fast Mode.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-255		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	I2cEnCfgFastModeSclLowLength		

2.3.1.4.11 I2cEnCfgFastModeSclLowLength

Table 89 Specification for I2cEnCfgFastModeSclLowLength

Name	I2cEnCfgFastModeSclLowLength		
Description	This parameter enables Direct Configuration of SCL Low Period Length in Fast Mode.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE		

Table 89 Specification for I2cEnCfgFastModeSclLowLength (continued)

	FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	I2cSpeed		

2.3.1.4.12 I2cAsyncNotification

Table 90 Specification for I2cAsyncNotification

Name	`I2cAsyncNotification`		
Description	Switches Asynchronous notification ON or OFF.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.13 I2cPacketEndNotification

Table 91 Specification for I2cPacketEndNotification

Name	`I2cPacketEndNotification`		
Description	This parameter is a reference to a notification function.		
Multiplicity	1..1	**Type**	EcucFunctionNameDef
Range	String		
Default value	NULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-

Table 91 Specification for I2cPacketEndNotification (continued)

Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.14 I2cTxTimeOut

Table 92 Specification for I2cTxTimeOut

Name	I2cTxTimeOut		
Description	This parameter contains timeout value for the write operation.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	50-4294967295		
Default value	65535		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.15 I2cRxTimeOut

Table 93 Specification for I2cRxTimeOut

Name	I2cRxTimeOut		
Description	This parameter contains timeout value for the read operation.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	50-4294967295		
Default value	65535		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.16 I2cSDASelect

Table 94 Specification for I2cSDASelect

Name	I2cSDASelect

Table 94 Specification for I2cSDASelect (continued)

Description	This parameter selects the port pin for SDA line. Refer DS for the list of pins applicable for specific I2C, format of pin description is as below: SDAxy_PORTz_PINk x - represents 0 to 1 based on the AURIX variant y - represents A, B, C, DN, DP, CN z - represents the port number k - represents the pin number Respective Alt-x function to be selected from the configuration. This parameter is IFX specific to make use of Hardware provided capability for selecting the right SDA pins.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	Depends on Micro variant		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.4.17 I2cSCLSelect

Table 95 Specification for I2cSCLSelect

Name	`I2cSCLSelect`		
Description	This parameter selects the port pin for SCL line. Refer DS for the list of pins applicable for specific I2C, format of pin description is as below: SCLxy_PORTz_PINk x - represents 0 to 1 based on the AURIX variant y - represents A, B, C, DN, DP, CN z - represents the port number k - represents the pin number Respective Alt-x function to be selected from the configuration. This parameter is IFX specific to make use of Hardware provided capability for selecting the right SCL pins.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	Depends on Micro variant		

Table 95 Specification for I2cSCLSelect (continued)

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5 Container: CommonPublishedInformation

This section describes the parameters published by the I2C driver.

Post-Build Variant Multiplicity: -

Configuration Class: -

2.3.1.5.1 ArPatchVersion

Table 96 Specification for ArPatchVersion

Name	`ArPatchVersion`		
Description	Patch version number of AUTOSAR specification on which the appropriate implementation is based upon.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-255		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5.2 ArMajorVersion

Table 97 Specification for ArMajorVersion

Name	`ArMajorVersion`		
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based upon.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-255		
Default value	4		

Table 97 Specification for ArMajorVersion (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5.3 ArMinorVersion

Table 98 Specification for ArMinorVersion

Name	`ArMinorVersion`		
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based upon.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-255		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5.4 SwMajorVersion

Table 99 Specification for SwMajorVersion

Name	`SwMajorVersion`		
Description	Major version number of the vendor specific implementation of the module.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-255		
Default value	As per driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5.5 SwMinorVersion

Table 100 Specification for SwMinorVersion

Name	SwMinorVersion		
Description	Minor version number of the vendor specific implementation of the module.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-255		
Default value	As per driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5.6 SwPatchVersion

Table 101 Specification for SwPatchVersion

Name	SwPatchVersion		
Description	Patch level version number of the vendor specific implementation of the module.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-255		
Default value	As per driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5.7 ModuleId

Table 102 Specification for ModuleId

Name	ModuleId		
Description	ModI ID of this module from Module List.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-65535		
Default value	255		

Table 102 Specification for ModuleId (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.5.8 VendorId

Table 103 Specification for VendorId

Name	`VendorId`		
Description	Vendor ID of the dedicated implementation of this mode according to the AUTOSAR vendor list.		
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	-		

2.3.1.5.9 Release

Table 104 Specification for Release

Name	`Release`		
Description	This parameter indicates the TC3xx device derivative used for the implementation.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	As per hardware derivative		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.6 Container: I2cGeneral

General configuration of I2C driver module.
Post-Build Variant Multiplicity: -
Configuration Class: -

2.3.1.6.1 I2cDevErrorDetect

Table 105 Specification for I2cDevErrorDetect

Name	I2cDevErrorDetect		
Description	Switches the Development Error Detection and Notification ON or OFF true: enabled (ON). false: disabled (OFF).		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
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	-		

2.3.1.6.2 I2cVersionInfoApi

Table 106 Specification for I2cVersionInfoApi

Name	I2cVersionInfoApi		
Description	Switches the I2c_GetVersionInfo function ON or OFF		
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	-		

2.3.1.6.3 I2cInitDeInitApiMode

Table 107 Specification for I2cInitDeInitApiMode

Name	`I2cInitDeInitApiMode`		
Description	This configuration parameter defines the mode in which the I2C Init and I2C DeInit API will be used.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	I2C_MCAL_SUPERVISOR I2C_MCAL_USER		
Default value	I2C_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.1.6.4 I2cSystemClock

Table 108 Specification for I2cSystemClock

Name	`I2cSystemClock`		
Description	This parameter refers to the System clock configured in MCU module.		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: McuClockReferencePointConfig		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

2.3.2 Functions - Type definitions

This section describes all the type definitions used by APIs.

2.3.2.1 I2c_ConfigType

Table 109 Specification for I2c_ConfigType

Syntax	`I2c_ConfigType`	
Type	Structure	
File	I2c.h	
Range	-	The elements of the data structure are specific to the microcontroller.
Description	This type contains the implementation-specific post build configuration structure of the I2C driver.	
Source	IFX	

2.3.2.2 I2c_ChannelType

Table 110 Specification for I2c_ChannelType

Syntax	`I2c_ChannelType`	
Type	uint8	
File	I2c.h	
Range	0-1	Represents the channel id
Description	This type contains the possible channel identifier types.	
Source	IFX	

2.3.2.3 I2c_ChannelConfigType

Table 111 Specification for I2c_ChannelConfigType

Syntax	`I2c_ChannelConfigType`	
Type	Structure	
File	I2c.h	
Range	-	The elements of the data structure are specific to the microcontroller.
Description	This type contains the implementation-specific to Channel configuration structure of the I2C driver.	
Source	IFX	

2.3.2.4 I2c_AddressingModeType

Table 112 Specification for I2c_AddressingModeType

Syntax	`I2c_AddressingModeType`

Table 112 Specification for I2c_AddressingModeType (continued)

Type	Enumeration	
File	I2c.h	
Range	I2C_7_BIT_ADDRESSING	Represents 7-bit addressing mode
	I2C_10_BIT_ADDRESSING	Represents 10-bit addressing mode
Description	This type contains the return type information which is used in various functions.	
Source	IFX	

2.3.2.5 I2c_NotifFunctionPtrType

Table 113 Specification for I2c_NotifFunctionPtrType

Syntax	`I2c_NotifFunctionPtrType`	
Type	typedef void(*I2c_NotifFunctionPtrType)(I2c_ErrorType ErrorId);	
File	I2c.h	
Range	-	-
Description	Represents the prototype for notification functions.	
Source	IFX	

2.3.2.6 I2c_ErrorType

Table 114 Specification for I2c_OperationType

Syntax	`I2c_ErrorType`	
Type	Enumeration	
File	I2c.h	
Range	I2C_NO_ERR	Returns when no error
	I2C_TX_UNDERFLOW	Returns when transmission underflow
	I2C_TX_OVERFLOW	Returns when receive overflow
	I2C_RX_UNDERFLOW	Returns when receive underflow
	I2C_RX_OVERFLOW	Returns when receive overflow
	I2C_NO_ACK	Returns when no acknowledgement
	I2C_ARBITRATION_LOST	Returns when arbitration lost
	I2C_INVALID_CHANNEL	Returns when channel invalid
	I2C_INVALID_SIZE	Returns when size invalid
	I2C_INVALID_ADDRESS	Returns when address invalid
	I2C_NULL_PTR	Returns when pointer is NULL
	I2C_IS_UNINIT	Returns when driver is uninitialized

Table 114 Specification for I2c_OperationType (continued)

	I2C_IS_BUSY	Returns when driver is busy
	I2C_ERR_OTHER	Other errors
Description	This type contains the return type information which is used in various functions.	
Source	IFX	

2.3.2.7 I2c_SizeType

Table 115 Specification for I2c_SizeType

Syntax	`I2c_SizeType`	
Type	uint16	
File	I2c.h	
Range	0-16383	Data size in bytes
Description	This type contains the possible channel status types.	
Source	IFX	

2.3.2.8 I2c_DataType

Table 116 Specification for I2c_DataType

Syntax	`I2c_DataType`	
Type	uint8	
File	I2c.h	
Range	0-255	-
Description	This type is used to hold the data to be sent or received.	
Source	IFX	

2.3.2.9 I2c_SlaveAddrType

Table 117 Specification for I2c_SlaveAddrType

Syntax	`I2c_SlaveAddrType`	
Type	uint16	
File	I2c.h	
Range	0-1023	-
Description	This type contains the possible slave address.	
Source	IFX	

2.3.2.10 I2c_ChannelStatusType

Table 118 Specification for I2c_ChannelStatusType

Syntax	`I2c_ChannelStatusType`	
Type	Enumeration	
File	I2c.h	
Range	I2C_UNINIT	Driver uninitialized
	I2C_IDLE	Bus idle
	I2C_BUSY	Bus busy
Description	This type contains the possible channel status types.	
Source	IFX	

2.3.3 Functions - APIs

This section lists the APIs provided along with a short description of the functionality.

2.3.3.1 I2c_Init

Table 119 Specification for `I2c_Init` API

Syntax	void I2c_Init (const I2c_ConfigType* const ConfigPtr)	
Service ID	0x4F	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration set.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This function will initialize all relevant registers of I2C peripheral with the values of structure ConfigPtr. This API needs to be invoked before invoking any other I2C APIs.	
Source	IFX	
Error handling	DET:	

Table 119 Specification for `I2c_Init` API (continued)

	I2C_E_ALREADY_INITIALIZED: This error is reported if, initialization is requested when kernel is already in initialized state. I2C_E_INIT_FAILED: This error is reported if the API is been invoked with invalid configuration pointer (null pointer).
Configuration dependencies	-
User hints	None

2.3.3.2 I2c_DeInit

Table 120 Specification for `I2c_DeInit` API

Syntax	Std_ReturnType I2c_DeInit (void)	
Service ID	0x50	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: de-initialization command has been accepted. E_NOT_OK: de-initialization command has not been accepted.
Description	This function will de-initialize the driver. It will reset all the I2C SFRs that were configured during the initialization of the driver	
Source	IFX	
Error handling	DET: I2C_E_UNINIT: This error is reported if de-initialization is requested when kernel is already in de-initialized state.	
Configuration dependencies	-	
User hints	None	

2.3.3.3 I2c_GetStatus

Table 121 Specification for `I2c_GetStatus` API

Syntax	I2c_ChannelStatusType I2c_GetStatus (const I2c_ChannelType ChannelId)	
Service ID	0x55	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant	
Parameters (in)	ChannelId	I2C channel identifier
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	I2c_ChannelStatusType	I2C_UNINIT : I2C module is uninitialized I2C_IDLE: I2C module is idle I2C_BUSY: I2C module is busy
Description	This API returns the status of the specified I2C module. The API I2c_GetStatus() is called to know if the specified I2C module is in I2C_UNINIT, I2C_IDLE or I2C_BUSY state.	
Source	IFX	
Error handling	DET: I2C_E_UNINIT: This error is reported if the API is been invoked when driver is not initialized. Note: When the driver is uninitialized and I2c_GetStatus is called, the API will raise I2C_E_UNINIT DET and will also return the I2C_UNINIT status. I2C_E_INVALID_CHANNEL: This error is reported if the API is been invoked with invalid channel.	
Configuration dependencies	-	
User hints	None	

2.3.3.4 I2c_SyncWrite

Table 122 Specification for `I2c_SyncWrite` API

Syntax	I2c_ErrorType I2c_SyncWrite (const I2c_ChannelType ChannelId, I2c_DataType *const DataPtr,

Table 122 Specification for `I2c_SyncWrite` API (continued)

	const I2c_SizeType Size, const I2c_SlaveAddrType SlaveAddress)	
Service ID	0x51	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant (for same channel)	
Parameters (in)	ChannelId	I2C channel identifier
	DataPtr	Pointer to data that needs to be transmitted
	Size	Size of data to be transmitted in bytes
	SlaveAddress	Address of slave
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	I2c_ReturnType	I2C_OK: Operation success I2C_NOT_OK: Operation not success I2C_IS_BUSY: Bus busy
Description	The service I2c_Write() is called to perform Write operation.	
Source	IFX	
Error handling	DET: I2C_E_UNINIT: This error is reported if, write is requested when kernel is in uninitialized state. I2C_E_INVALID_CHANNEL: This error is reported if the API is been invoked with invalid channel id. I2C_E_PARAM_POINTER: This error is reported if the API is been invoked with invalid (null) data pointer. I2C_E_INVALID_SIZE: This error is reported if the API is been invoked with invalid data size. I2C_E_INVALID_SLAVE_ADDRESS: This error is reported if the API is been invoked with invalid slave address. I2C_E_HW_UNIT_BUSY: This error is reported if the API is been invoked when bus is busy.	
Configuration dependencies	-	
User hints	None	

2.3.3.5 I2c_SyncRead

Table 123 Specification for `I2c_SyncRead` API

Syntax	I2c_ErrorType I2c_SyncRead

Table 123 Specification for `I2c_SyncRead` API (continued)

	(const I2c_ChannelType ChannelId, I2c_DataType *const DataPtr, const I2c_SizeType Size, const I2c_SlaveAddrType SlaveAddress)	
Service ID	0x52	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant (for same channel)	
Parameters (in)	ChannelId	I2C channel identifier
	Size	Size of data to be received in Bytes
	SlaveAddress	Address of slave
Parameters (out)	DataPtr	Pointer to data that is received.
Parameters (in - out)	-	-
Return	I2c_ReturnType	I2C_OK : Operation Success I2C_NOT_OK: Operation not success I2C_IS_BUSY: Bus busy
Description	The service I2c_Read() is called to perform Read operation.	
Source	IFX	
Error handling	DET: I2C_E_UNINIT: This error is reported if, write is requested when kernel is in uninitialized state. I2C_E_INVALID_CHANNEL: This error is reported if the API is been invoked with invalid channel id. I2C_E_PARAM_POINTER: This error is reported if the API is been invoked with invalid (null) data pointer. I2C_E_INVALID_SIZE : This error is reported if the API is been invoked with invalid data size. I2C_E_INVALID_SLAVE_ADDRESS: This error is reported if the API is been invoked with invalid slave address. I2C_E_HW_UNIT_BUSY: This error is reported if the API is been invoked when bus is busy.	
Configuration dependencies	-	
User hints	None	

2.3.3.6 I2c_AsyncWrite

Table 124 Specification for `I2c_AsyncWrite` API

Syntax	I2c_ErrorType I2c_AsyncWrite (const I2c_ChannelType ChannelId, I2c_DataType *const DataPtr, const I2c_SizeType Size, const I2c_SlaveAddrType SlaveAddress)	
Service ID	0x53	
Sync/Async	Asynchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant (for same channel)	
Parameters (in)	ChannelId	I2C channel identifier
	DataPtr	Pointer to data that needs to be transmitted
	Size	Size of data to be transmitted in bytes
	SlaveAddress	Address of slave
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	I2C_NO_ERR	Returns when no error
	I2C_INVALID_CHANNEL	Returns when channel invalid
	I2C_INVALID_SIZE	Returns when size invalid
	I2C_INVALID_ADDRESS	Returns when address invalid
	I2C_NULL_PTR	Returns when pointer is NULL
	I2C_IS_UNINIT	Returns when driver is uninitialized
	I2C_IS_BUSY	Returns when driver is busy
	I2C_ERR_OTHER	Other errors
Description	The service I2c_AsyncWrite() is called to perform Write operation.	
Source	IFX	
Error handling	DET: I2C_E_UNINIT: This error is reported if, write is requested when kernel is in uninitialized state. I2C_E_INVALID_CHANNEL: This error is reported if the API is been invoked with invalid channel id. I2C_E_PARAM_POINTER: This error is reported if the API is been invoked with invalid (null) data pointer.	

Table 124 Specification for `I2c_AsyncWrite` API (continued)

	I2C_E_INVALID_SIZE: This error is reported if the API is been invoked with invalid data size. I2C_E_INVALID_SLAVE_ADDRESS: This error is reported if the API is been invoked with invalid slave address. I2C_E_HW_UNIT_BUSY: This error is reported if the API is been invoked when bus is busy.
Configuration dependencies	-
User hints	None

2.3.3.7 I2c_AsyncRead

Table 125 Specification for `I2c_AsyncRead` API

Syntax	I2c_ErrorType I2c_AsyncRead (const I2c_ChannelType ChannelId, I2c_DataType *const DataPtr, const I2c_SizeType Size, const I2c_SlaveAddrType SlaveAddress)	
Service ID	0x54	
Sync/Async	Asynchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant (for same channel)	
Parameters (in)	ChannelId	I2C channel identifier
	DataPtr	Pointer to data that needs to be transmitted
	Size	Size of data to be transmitted in bytes
	SlaveAddress	Address of slave
Parameters (out)	DataPtr	
Parameters (in - out)	-	
Return	I2C_NO_ERR	Returns when no error
	I2C_INVALID_CHANNEL	Returns when channel invalid
	I2C_INVALID_SIZE	Returns when size invalid
	I2C_INVALID_ADDRESS	Returns when address invalid
	I2C_NULL_PTR	Returns when pointer is NULL
	I2C_IS_UNINIT	Returns when driver is uninitialized

Table 125 Specification for `I2c_AsyncRead` API (continued)

	I2C_IS_BUSY	Returns when driver is busy
	I2C_ERR_OTHER	Other errors
Description	The service I2c_AsyncRead() is called to perform Read operation.	
Source	IFX	
Error handling	DET: I2C_E_UNINIT: This error is reported if, write is requested when kernel is in uninitialized state. I2C_E_INVALID_CHANNEL: This error is reported if the API is been invoked with invalid channel id. I2C_E_PARAM_POINTER: This error is reported if the API is been invoked with invalid (null) data pointer. I2C_E_INVALID_SIZE: This error is reported if the API is been invoked with invalid data size. I2C_E_INVALID_SLAVE_ADDRESS: This error is reported if the API is been invoked with invalid slave address. I2C_E_HW_UNIT_BUSY: This error is reported if the API is been invoked when bus is busy.	
Configuration dependencies	-	
User hints	None	

2.3.3.8 I2c_CancelOperation

Table 126 Specification for `I2c_CancelOperation` API

Syntax	Std_ReturnType I2c_CancelOperation (const I2c_ChannelType ChannelId, I2c_SizeType *const TransmittedDataSize)	
Service ID	0x56	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non-Reentrant	
Parameters (in)	ChannelId	I2C channel id
Parameters (out)	TransmittedDataSize	Size transmitted before cancel (in bytes)
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Operation successful E_NOT_OK: Operation unsuccessful

Table 126 Specification for `I2c_CancelOperation` API (continued)

Description	This service cancels the ongoing operation and returns the total data transmitted through I2c channel before it is canceled. The API can be invoked only when the communication is in asynchronous mode.
Source	IFX
Error handling	DET: I2C_E_UNINIT: This error is reported if, write is requested when kernel is in uninitialized state. I2C_E_INVALID_CHANNEL: This error is reported if the API is been invoked with invalid channel id. I2C_E_PARAM_POINTER: This error is reported if the API is been invoked with TransmittedDataSize as null pointer.
Configuration dependencies	I2cAsyncReadWriteEnable
User hints	None

2.3.3.9 I2c_GetVersionInfo

Table 127 Specification for `I2c_GetVersionInfo` API

Syntax	void I2c_GetVersionInfo (Std_VersionInfoType * const VersionInfoPtr)	
Service ID	0x57	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	VersionInfoPtr	Address where the version information of the I2C module must be stored.
Parameters (in - out)	-	-
Return	void	-
Description	This API returns the version information of this module. Note: This API is available only when I2cVersionInfoApi is configured as true.	
Source	IFX	
Error handling	DET: I2C_E_PARAM_POINTER: This error is reported if, API is invoked with VersionInfoPtr as null pointer.	

Table 127 Specification for `I2c_GetVersionInfo` API (continued)

Configuration dependencies	I2cVersionInfoApi
User hints	None

2.3.4 Notifications and callbacks

The driver does not support any notification and callbacks.

2.3.5 Scheduled functions

The driver does not support any scheduled functions.

2.3.6 Interrupt service routines

The driver implements the following interrupt service routines.

2.3.6.1 I2c_IsrI2cDtr

Table 128 Specification for `I2c_IsrI2cDtr`

Syntax	void I2c_IsrI2cDtr (const uint8 HwUnit)	
Service ID	NA	
Sync/Async	Asynchronous	
ASIL level	QM	
Re-entrancy	Reentrant (for different channels)	
Parameters (in)	HwUnit	HW unit index
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Handles the burst data interrupts passed from I2C kernel.	
Source	IFX	
Error handling	DET: None	
Configuration dependencies	I2cAsyncReadWriteEnable	
User hints	None	

2.3.6.2 I2c_IsrI2cProtocol

Table 129 Specification for `I2c_IsrI2cProtocol`

Syntax	void I2c_IsrI2cProtocol (const uint8 HwUnit)	
Service ID	NA	
Sync/Async	Asynchronous	
ASIL level	QM	
Re-entrancy	Reentrant (for different channels)	
Parameters (in)	HwUnit	HW unit index
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Handles the protocol interrupts passed from I2C kernel.	
Source	IFX	
Error handling	DET: None	
Configuration dependencies	I2cAsyncReadWriteEnable	
User hints	None	

2.3.6.3 I2c_IsrI2cError

Table 130 Specification for `I2c_IsrI2cError`

Syntax	void I2c_IsrI2cError (const uint8 HwUnit)	
Service ID	NA	
Sync/Async	Asynchronous	
ASIL level	QM	
Re-entrancy	Reentrant	
Parameters (in)	HwUnit	HW unit index

Table 130 Specification for `I2c_IsrI2cError` (continued)

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Handles the error interrupts passed from I2C kernel.	
Source	IFX	
Error handling	DET: None	
Configuration dependencies	I2cAsyncReadWriteEnable	
User hints	None	

2.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

2.3.7.1 Production errors

The I2C driver does not have any production errors.

2.3.7.2 Development errors

The following table lists the development errors reported by the driver.

Table 131 Description of development errors reported

Description	Source	Error code and value	Applicable APIs
This error is reported if API Service called with NULL parameter.	IFX	I2C_E_PARAM_POINTER = 0x00	I2c_Write I2c_Read I2c_CancelOperation I2c_GetVersionInfo
This error is reported if API Service used without initialization.	IFX	I2C_E _ UNINIT = 0x01	I2c_DeInit I2c_GetStatus I2c_Write I2c_Read I2c_CancelOperation
This error is reported if transmission service called at invalid channel.	IFX	I2C_E_INVALID_ CHANNEL = 0x02	I2c_GetStatus I2c_Write I2c_Read

Table 131 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
			I2c_CancelOperation
This error is reported if I2C driver is already initialized.	IFX	I2C_E_ALREADY_INITIALIZED = 0x03	I2c_Init
This error is reported if I2C peripheral is busy.	IFX	I2C_E_HW_UNIT_BUSY = 0x04	I2c_Write I2c_Read
This error is reported if I2C invalid slave address.	IFX	I2C_E_INVALID_SLAVE_ADDRESS = 0x05	I2c_Write I2c_Read
This error is reported if, I2C driver is provided with invalid data size.	IFX	I2C_E_INVALID_SIZE = 0x06	I2c_Write I2c_Read
This error is reported if, I2C driver is provided with NULL config pointer.	IFX	I2C_E_INIT_FAILED = 0x07	I2c_Init

2.3.7.3 Safety errors

The I2C driver does not have any safety errors.

2.3.7.4 Runtime errors

The I2C driver does not have any runtime errors.

2.3.8 Deviations and limitations

The section describes the deviations and limitations from software specification.

2.3.8.1 Deviations

Not applicable for I2C.

2.3.8.2 Limitations

Not applicable for I2C.

2.3.9 Unsupported hardware features

The following hardware features of I2C are not supported

- I2C bus as Slave
- FIFO as flow controller(DMA flow)

3 IOM driver

3.1 User information

3.1.1 Description

The Input-Output Monitor (IOM) driver serves as a comparison unit, checking the correct operation of the system peripherals output that may serve as input to the monitoring function. The monitoring function should be achieved by configuring the IOM hardware. It generates global system event to the SMU.

The IOM driver initializes and controls the IOM unit of the microcontroller. The driver also provides services for the user to initialize and set the threshold values for the internal units of the IOM. It should also provide services to reset the IOM kernel.

The service should be provided to combine individual or multiple local events in order to generate a single global system event. The IOM driver is heavily dependent on initialization and configurations.

3.1.2 Hardware-software mapping

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

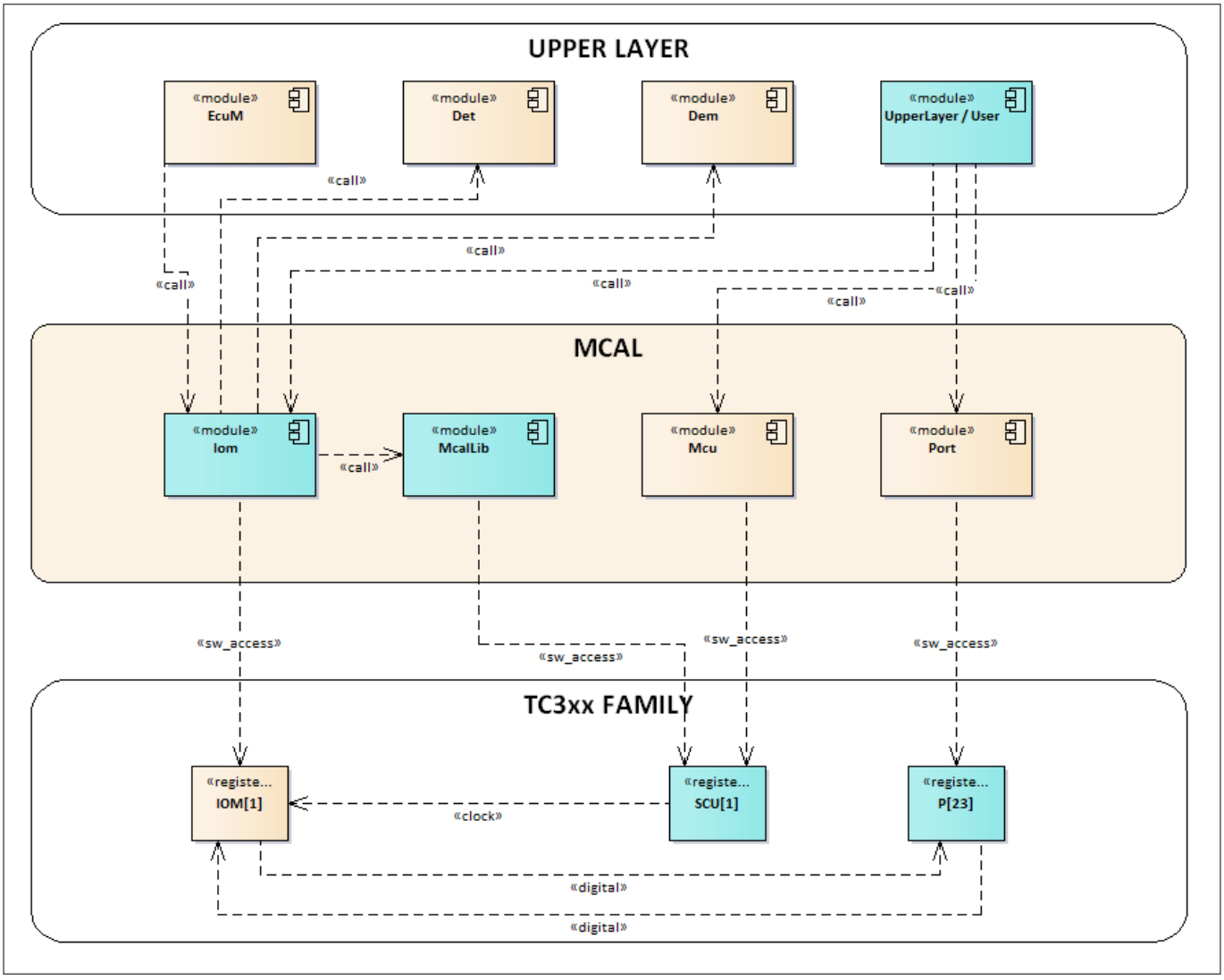


Figure 31 Mapping of hardware-software interfaces

3.1.2.1 IOM: primary hardware peripheral

Hardware functional features

The IOM driver is needed for the input output monitoring of signals.

Users of the hardware

The IOM driver exclusively utilizes the IOM IP for its functionality.

Hardware diagnostic features

None.

Hardware events

None.

3.1.2.2 SCU: primary hardware peripheral

The SCU is needed for the CLOCK for the registers, and ENDINIT functionality is used to update certain registers.

Hardware functional features

The IOM driver depends on the SCU for the clock, ENDINIT and reset functionalities.

Users of the hardware

The SCU module supplies the clock for all the peripherals and the MCU driver is responsible for configuring the clock tree. In order to avoid conflicts, update to the ENDINIT protected registers is performed using the MCALLIB.

Hardware diagnostic features

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

Hardware events

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

3.1.2.3 Port: dependent hardware peripheral

Hardware functional features

The PORT driver controls all access to the pins required by the IOM for input and output configuration.

Users of the hardware

The port pads are configured and used by the PORT and DIO drivers.

Hardware diagnostic features

None.

Hardware events

None.

3.1.3 File structure

3.1.3.1 C file structure

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

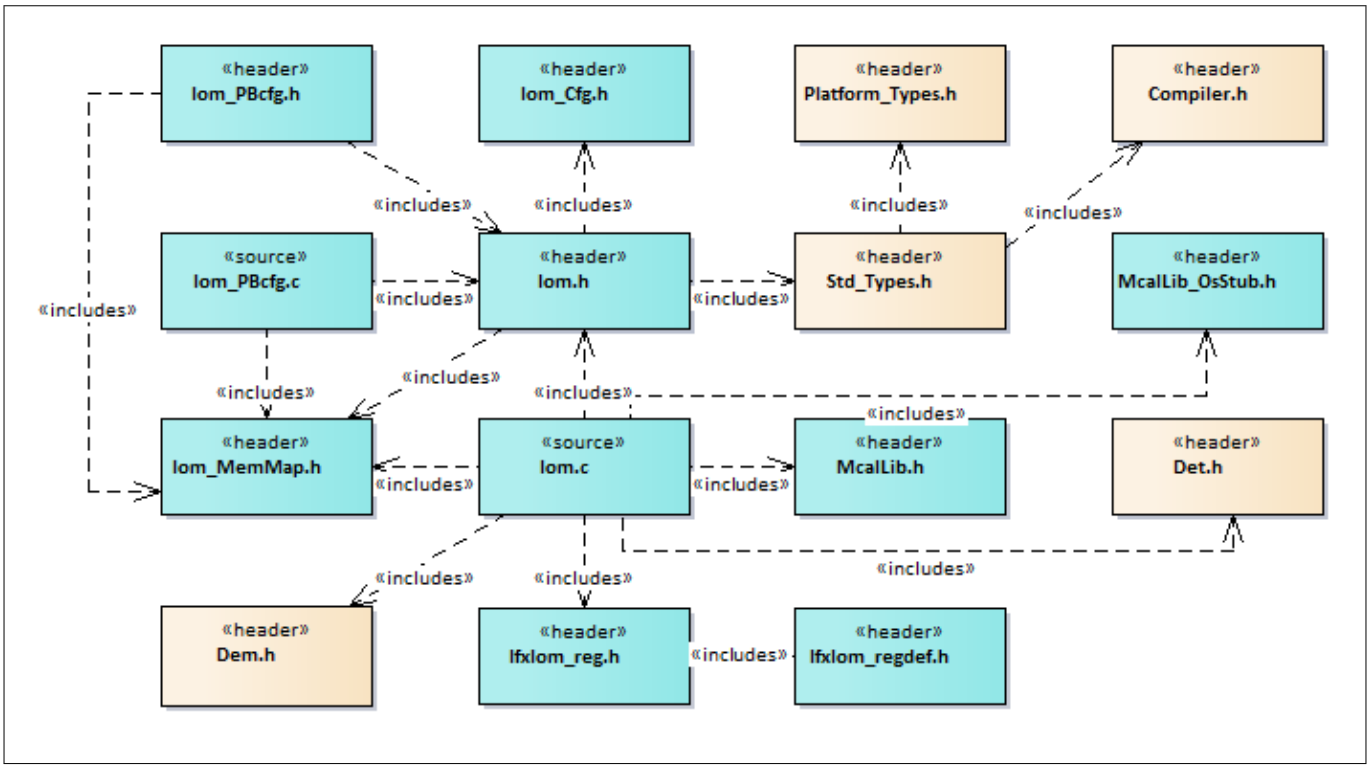


Figure 32 C file structure

Table 132 C file structure

Filename	Description
`Std_Types.h`	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.
`Compiler.h`	Provides macros for the encapsulation of definitions and declarations
`Platform_Types.h`	Platform-specific type declaration file as defined by AUTOSAR
`IfxIom_reg.h`	SFR header file for the IOM
`Det.h`	Provides the exported interfaces of the DET
`McalLib_OsStub.h`	McalLib_OsStub.h provides macros to support user mode of the TriCore™.
`Iom_MemMap.h`	File (Static) containing the memory section definitions used by the IOM driver
`Iom_Cfg.h`	Header file (Generated) containing constants and pre-processor macros as #defines
`Iom.c`	File (Static) containing implementation of the APIs
`Iom_PBCfg.h`	File (Generated) containing declaration of the post-build configuration data structures
`Iom_PBCfg.c`	File (Generated) containing a definition of the configuration data structures
`McalLib.h`	The header file (Static) defining prototypes of data structures and APIs of end-init and delay services and included by McalLib.c

3.1.3.2 Code generator plugin files

The section provides details of the plugin files of the IOM driver.

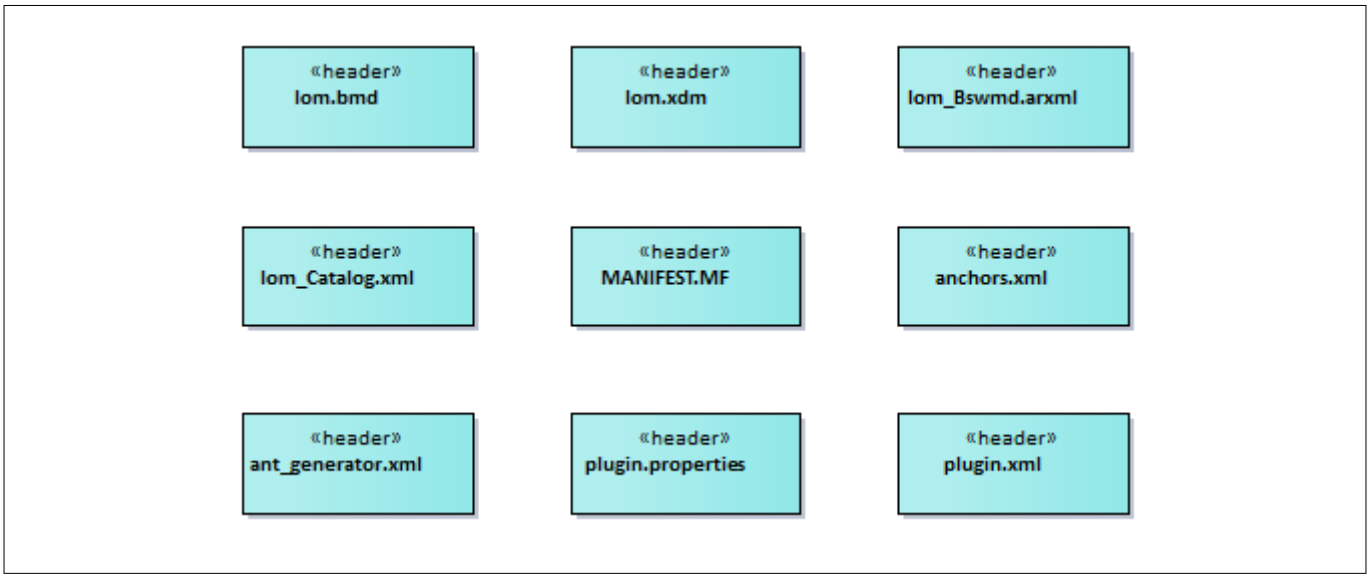


Figure 33 Code genearator plugin files

Table 133 Code generator plugin files

File name	Description
`anchors.xml`	Tresos anchors support file for the IOM driver
`Iom.xdm`	Iom.xdm Tresos format XML data model schema file
`Iom.bmd`	AUTOSAR format XML data model schema file (for each device)
`Iom_Catalog.xml`	AUTOSAR format catalog file
`Iom_Bswmd.arxml`	AUTOSAR format module description file
`MANIFEST.MF`	Tresos plugin support file containing the metadata for the IOM driver
`plugin.xml`	Tresos plugin support file for the IOM driver
`plugin.properties`	Tresos plugin support file for the IOM driver
`ant_generator.xml`	Tresos support file to generate and rename multiple post-build configurations when using the variation point

3.1.4 Integration hints

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

3.1.4.1 Integration with AUTOSAR Stack

- **ECuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **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 `Iom_MemMap.h` file. The `Iom_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 IOM_START_SEC_VAR_CLEARED_QM_LOCAL_32
  #ifdef _TASKING_C_TRICORE_
  /*****User pragmas here*****/
  #undef IOM_START_SEC_VAR_CLEARED_QM_LOCAL_32
  #undef MEMMAP_ERROR
  #elif defined IOM_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
  #ifdef _TASKING_C_TRICORE_
  /*****User pragmas here*****/
  #undef IOM_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
  #undef MEMMAP_ERROR

  /*****  CONFIG DATA  *****/
  #elif defined IOM_START_SEC_CONFIG_DATA_QM_LOCAL_UNSPECIFIED
  #ifdef _TASKING_C_TRICORE_
  /*****User pragmas here*****/
  #undef IOM_START_SEC_CONFIG_DATA_QM_LOCAL_UNSPECIFIED
  #undef MEMMAP_ERROR
  #elif defined IOM_STOP_SEC_CONFIG_DATA_QM_LOCAL_UNSPECIFIED
  #ifdef _TASKING_C_TRICORE_
  /*****User pragmas here*****/
  #undef IOM_STOP_ SEC_CONFIG_DATA_QM_LOCAL_UNSPECIFIED
  #undef MEMMAP_ERROR

  /*****  CODE  *****/
  #elif defined IOM_START_SEC_CODE_QM_LOCAL
  #ifdef _TASKING_C_TRICORE_
  /*****User pragmas here*****/
  #undef IOM_START_SEC_CODE_QM_LOCAL
  #undef MEMMAP_ERROR
  #elif defined IOM_STOP_SEC_CODE_QM_LOCAL
  #ifdef _TASKING_C_TRICORE_
  /*****User pragmas here*****/
  #undef IOM_STOP_SEC_CODE_QM_LOCAL
  #undef MEMMAP_ERROR

  #endif

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

- **DET**

  The DET module is a part of the AUTOSAR stack that handles all the development and runtime errors reported by the BSW modules. The IOM driver reports all the development errors to the DET module

through the `Det_ReportError()` API. The user of the IOM driver must process all the errors reported to the DET module through the `Det_ReportError()` API. The `Det.h` and `Det.c` files are provided in the MCAL package as a stub code and need to be replaced with a complete DET module during the integration phase.

- **DEM**

  The DEM module is a part of the AUTOSAR stack that handles all the production errors reported by the BSW modules. The MCU driver reports all the production errors to the DEM modules through the `Dem_ReportErrorStatus()` API. The user of the IOM driver must process all the production errors (fail/pass) reported to the DEM module through the `Dem_ReportErrorStatus()` API. The `Dem.h` and `Dem.c` files are provided in the MCAL package as a stub code and need to be replaced with a complete DEM module during the integration phase.

- **Schm**

  The SchM is not required for the integration of the IOM driver.

- **Safety error**

  The IOM library does not report any safety errors.

- **Notification and callbacks**

  The IOM driver does not provide any callbacks or notifications.

- **Operating system**

  The IOM driver does not program any Service Request(SR). The OS or the application must ensure the correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or the application.

3.1.4.2 Multicore and Resource Manager

The IOM driver does not support execution on multiple cores in parallel.

3.1.4.3 MCU support

The system clock is set up through the MCU driver. The MCU initialization should be performed before using the IOM APIs to ensure the clock supply to the IOM hardware.

3.1.4.4 Port support

The PORT driver configures the port pins of the entire microcontroller. The user must configure the port pins used by the IOM driver through the PORT configuration and initialize the port pins prior to invoking the IOM initialization.

3.1.4.5 DMA support

The IOM driver does not use any services provided by the DMA driver.

3.1.4.6 Interrupt connections

The IOM driver does not use any interrupt source.

3.1.4.7 Example usage

This section explains one of the example usage of the IOM driver for a nominal case.

Configuration of the driver

The IOM driver is configured before usage and the configuration files are generated and made available during the software build process.

Initialization of the driver

The code sequence for initializing the IOM driver is as follows:

```
#include "Iom.h"
#include "Mcu.h"
#include "Port.h"

extern const Iom_ConfigType Iom_Config;

/* MCU Initialization */
Mcu_Init(&Mcu_Config);
   Mcu_InitClock( 0 );
   while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
   Mcu_DistributePllClock();

  /* Port Initialization */
Port_Init(&Port_Config);

/* Iom Initialization */
Iom_Init(&Iom_Config);


/* Further APIs of IOM driver can be called now */
```

The following code snippet shows call to Iom_ClrResetStatus(() and Iom_ResetKernel() APIs.

```
/* To Reset the Kernel */
Iom_ResetKernel();

/* To Clear the Reset status */
Iom_ClrResetStatus();
```

The following code is used to set values using for the Iom_SetLamThreshold () and Iom_ Iom_SetLamConfig() APIs.

```
/* To set the Threshold value for Lam */
Iom_SetLamThreshold(LamNo,ThresholdValue);

/*To update the configuration of Lam unit */
Iom_SetLamConfig(LamNo,ConfigurationValue);
```

The following code is used to read values from the Iom_GetResetStatus(),Iom_GetLamThreshold() and Iom_GetEcmThresVal() APIs.

```
/* To read threshold value of the counter in Ecm */
ThresVal = Iom_GetEcmThresVal(CounterNo)
/* CounterNo = Counter number in ECM  */

/* To read the Lam threshold value */
      status32 = Iom_GetLamThreshold(LamNo);
      /* LamNo = LAM unit number */

      /* read the kernel reset status bit */
      status8 = Iom_GetResetStatus();
```

Deinitialization of the driver

The following code is used to de-initialize IOM the driver.

```
/* Iom De-Initialization */
Iom_DeInit();
```

3.1.5 Key architectural considerations

There are no key architectural considerations for the driver.

3.2 Assumptions of Use (AoU)

There are no AoUs for the driver.

3.3 Reference information

3.3.1 Configuration interfaces

The following diagram depicts the hierarchy along with their configuration parameters.

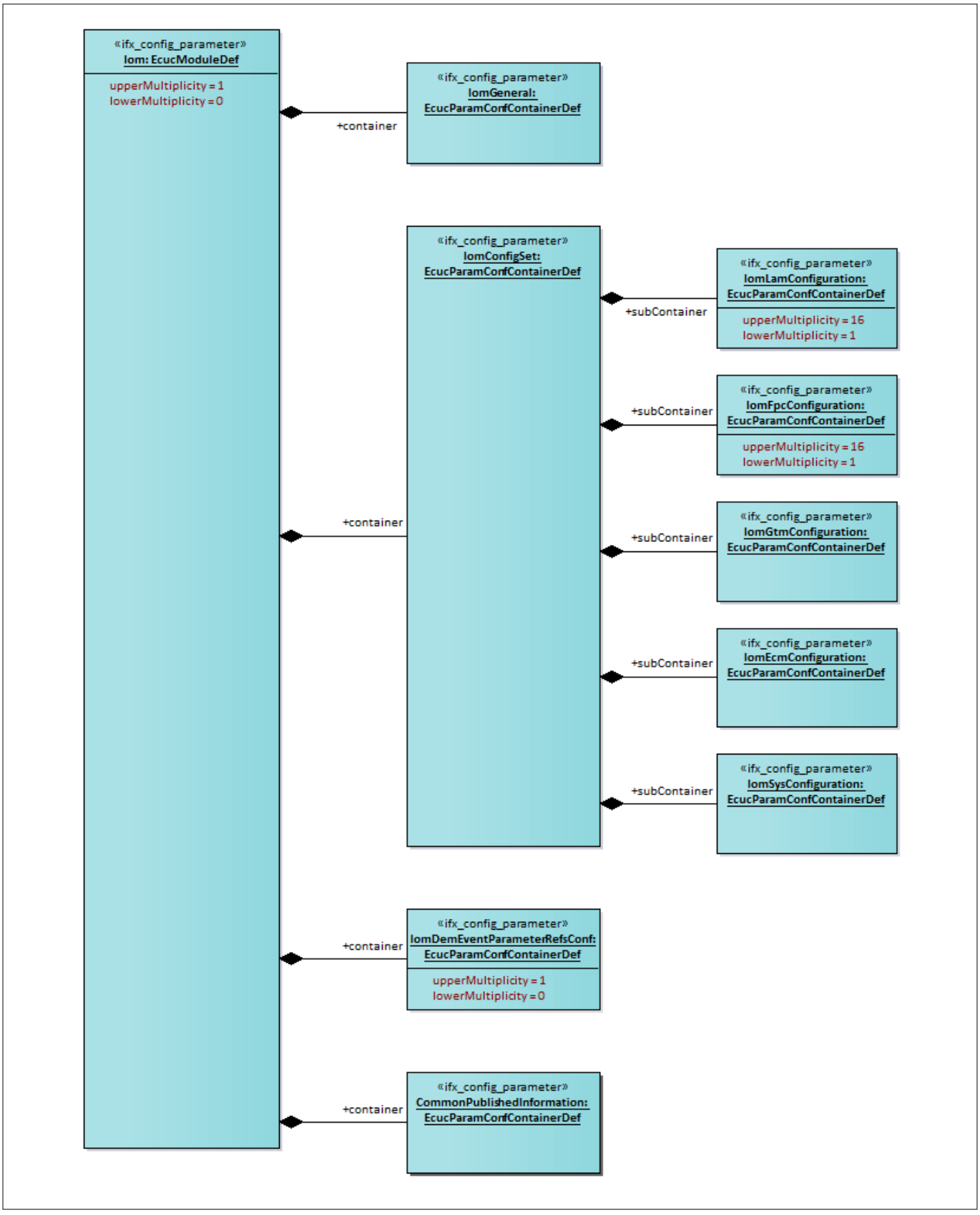


Figure 34 Configuration container relationship

3.3.1.1 Container: CommonPublishedInformation

Multiplicity Configuration Class: -

3.3.1.1.1 ArMajorVersion

Table 134 Specification for ArMajorVersion

Name	ArMajorVersion		
Description	This parameter provides the major version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.1.2 ArMinorVersion

Table 135 Specification for ArMinorVersion

Name	ArMinorVersion		
Description	This parameter provides the minor version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.1.3 ArPatchVersion

Table 136 Specification for ArPatchVersion

Name	ArPatchVersion		
Description	This parameter provides the patch version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

Table 136 Specification for ArPatchVersion (continued)

Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.1.4 ModuleId

Table 137 Specification for ModuleId

Name	ModuleId		
Description	This parameter provides the module ID of IOM.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
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	-		

3.3.1.1.5 Release

Table 138 Specification for Release

Name	Release		
Description	This parameter indicates the TC3xx device derivative used for the implementation.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	As per hardware derivative		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL

Table 138	Specification for Release (continued)
Dependency	-

3.3.1.1.6 SwMajorVersion

Table 139	Specification for SwMajorVersion		
Name	SwMajorVersion		
Description	This parameter provides the major version of the software.		
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	-		

3.3.1.1.7 SwMinorVersion

Table 140	Specification for SwMajorVersion		
Name	SwMinorVersion		
Description	This parameter provides the minor version of the software.		
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	-		

3.3.1.1.8 SwPatchVersion

Table 141	Specification for SwMajorVersion
Name	SwPatchVersion
Description	This parameter provides the patch version of the software.

Table 141 Specification for SwMajorVersion (continued)

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	-		

3.3.1.1.9 VendorId

Table 142 Specification for VendorId

Name	VendorId		
Description	This parameter provides the vendor ID.		
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	-		

3.3.1.2 Container: IomGtmConfiguration

This container holds the Lam Configuration.

Multiplicity Configuration Class: -

3.3.1.2.1 IomGtmInputx

Table 143 Specification for IomGtmInputx

Name	IomGtmInputx
Description	Disables/Enables the GTM input signal x to be included in EXOR combiner. x varies from 0 to 7. IOM_DISABLE_GTM_INPUT – disables the selected GTM input signal. IOM_ENABLE_GTM_INPUT – enables the selected GTM input signal.

Table 143 Specification for IomGtmInputx (continued)

Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_DISABLE_GTM_INPUT IOM_ENABLE_GTM_INPUT		
Default value	IOM_DISABLE_GTM_INPUT		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.3 Container: IomEcmConfiguration

This container holds the ECM Configuration.
Multiplicity Configuration Class: -

3.3.1.3.1 IomEcmThresholdx

Table 144 Specification for IomEcmThresholdx

Name	IomEcmThresholdx		
Description	Indicates threshold count value for the counter x (varies from 0 to 3) of the ECM module. Upon counter meet this value, the counter event output becomes high for one clock cycle. If the count is set to zero, the counter is disabled		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 to 15		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.3.2 IomEcmEventSelx

Table 145 Specification for IomEcmEventSelx

Name	IomEcmEventSelx
Description	Determines which LAM channel event output is routed to counter x(varies from 0 to 3) of ECM module.

Table 145 Specification for IomEcmEventSelx (continued)

Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 to 15		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.4 Container: IomEventCombModGlobalSel

This container holds the ECM Configuration.

Multiplicity Configuration Class: -

3.3.1.4.1 IomEcmEventCombSelx

Table 146 Specification for IomEcmEventCombSelx

Name	IomEcmEventCombSelx x varies from 0 to 15		
Description	Add/Remove LAMx (x varies from 0 to 15) output event in global event generation. IOM_DISABLE_CHANNEL_EVENT- disables LAM output event in global event generation. IOM_ENABLE_CHANNEL_EVENT - enables LAM output event in global event generation.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	IOM_DISABLE_CHANNEL_EVENT IOM_ENABLE_CHANNEL_EVENT		
Default value	IOM_DISABLE_CHANNEL_EVENT		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency			

3.3.1.4.2 IomEcmAccEventCombSelx

Table 147 Specification for IomEcmAccEventCombSelx

Name	IomEcmAccEventCombSelx x varies from 0 to 3		
Description	Add/Remove counter x output event in global event generation. IOM_DISABLE_COUNT_EVENT- disables counter x output event in global event generation. IOM_ENABLE_COUNT_EVENT - enables counter x output event in global event generation. Note : x varies from 0 to 3		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	IOM_DISABLE_COUNT_EVENT IOM_ENABLE_COUNT_EVENT		
Default value	IOM_DISABLE_COUNT_EVENT		
Post-build variant value	TRUE	Post-build variant multiplicity	-
Value configuration class	Post-Build	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency			

3.3.1.5 Container: IomSysConfiguration

This container holds the ECM Configuration.

Multiplicity Configuration Class: -

3.3.1.5.1 IomClcSleepModeEn

Table 148 Specification for IomClcSleepModeEn

Name	IomClcSleepModeEn		
Description	Used to enable or disable the sleep mode of the module. FALSE – disable module sleep mode TRUE – enable module sleep mode		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

Table 148 Specification for IomClcSleepModeEn (continued)

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.5.2 IomClcRmcVal

Table 149 Specification for IomClcRmcVal

Name	IomClcRmcVal		
Description	Determines 8 bit clock divider value in the RUN mode.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 to 255		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.6 Container: IomGeneral

The container contains all the general configuration parameters for the IOM driver.
Multiplicity Configuration Class: -

3.3.1.6.1 IomVersionInfoApi

Table 150 Specification for IomVersionInfoApi

Name	IomVersionInfoApi		
Description	Parameter adds or removes the Iom_GetVersionInfo() API from the code. The default value of this parameter is set to false 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**	-

Table 150 Specification for IomVersionInfoApi (continued)

Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.6.2 IomDeInitApi

Table 151 Specification for IomDeInitApi

Name	IomDeInitApi		
Description	Parameter adds or removes the Iom_DeInit () API from the code. The default value of this parameter is set to false 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	-		

3.3.1.6.3 IomDevErrorDetect

Table 152 Specification for IomDevErrorDetect

Name	IomDevErrorDetect		
Description	Parameter enables or disables the Default Error Tracer (DET) detection and reporting.		
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	-		

3.3.1.6.4 IomIndex

Table 153 Specification for IomIndex

Name	IomIndex		
Description	Specifies instance id for this module instance.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 to 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	-		

3.3.1.6.5 IomRuntimeApiMode

Table 154 Specification for IomRuntimeApiMode

Name	IomRuntimeApiMode		
Description	The parameter defines the privilege mode in which the runtime APIs would operate. Since IOM driver accesses the SFRs, it is more efficient to operate the IOM driver in supervisor mode. Hence, the default mode of operation is a supervisor.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_MCAL_SUPERVISOR IOM_MCAL_USER1		
Default value	IOM_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.6.6 IomInitDeInitApiMode

Table 155 Specification for IomInitDeInitApiMode

Name	IomInitDeInitApiMode
Description	Configuration parameter defines the privilege mode in which the initialization and deinitialization APIs would operate.

Table 155 Specification for IomInitDeInitApiMode (continued)

	Since IOM driver accesses the SFRs, it is more efficient to operate the IOM driver in supervisor mode. Hence, the default mode of operation is a supervisor.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_MCAL_SUPERVISOR IOM_MCAL_USER1		
Default value	IOM_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.7 Container: IomDemEventParameterRefsConf

This container holds the ECM Configuration.
Multiplicity Configuration Class: - Pre-Compile

3.3.1.7.1 IomClcFailureNotification

Table 156 Specification for IomClcFailureNotification

Name	IomClcFailureNotification		
Description	The parameter defines whether CLC failure DEM notification is enabled or not.		
Multiplicity	0..1	**Type**	
Range	Reference to Node: DemEventParameter		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.8 Container: IomFpcConfiguration

This container holds the Fpc Configuration.
Multiplicity Configuration Class: - Pre-Compile

3.3.1.8.1 IomFpcHwUnit

Table 157 Specification for IomFpcHwUnit

Name	IomFpcHwUnit		
Description	Identification number for Fpc unit.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-15		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.8.2 IomFpcCompareVal

Table 158 Specification for IomFpcCompareVal

Name	IomFpcCompareVal		
Description	This parameter is used to set the compare value of Fpc.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-65535		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.8.3 IomFpcMode

Table 159 Specification for IomFpcMode

Name	IomFpcMode
Description	Used to select a mode of operation for FPC. IOM_MOD_0_BOTHEDGES_DD – FPC is configured to operate in delayed debounce filter mode on both edges IOM_MOD_1_BOTHEDGES_ID – FPC is configured to operate in immediate debounce filter mode on both edges

Table 159 Specification for IomFpcMode (continued)

	IOM_MOD_2_RISINGEDGE_ID – FPC is configured to operate in Delayed debounce filter mode on the rising edge and no filtering on falling edge IOM_MOD_3_FALLINGEDGE_ID - FPC is configured to operate in immediate debounce filter mode on falling edge and no filtering on rising edge IOM_MOD_4_RISING_DD_FALLING_ID - FPC is configured to operate in delayed debounce filter mode on the rising edge and immediate debounce filter mode on falling edge IOM_MOD_5_RISING_ID_FALLING_DD - FPC is configured to operate in immediate debounce filter mode on the rising edge and delayed debounce filter mode on the falling edge. IOM_MOD_6_RISINGEDGE_PRESCALER – prescaler mode is triggered on the rising edge IOM_MOD_7_FALLINGEDGE_PRESCALER – prescaler mode is triggered on a falling edge.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_MOD_0_BOTHEDGES_DD IOM_MOD_1_BOTHEDGES_ID IOM_MOD_2_RISINGEDGE_ID IOM_MOD_3_FALLINGEDGE_ID IOM_MOD_4_RISING_DD_FALLING_ID IOM_MOD_5_RISING_ID_FALLING_DD IOM_MOD_6_RISINGEDGE_PRESCALER IOM_MOD_7_FALLINGEDGE_PRESCALER		
Default value	IOM_MOD_0_BOTHEDGES_DD		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.8.4 IomFpcMonInputSel

Table 160 Specification for IomFpcMonInputSel

Name	IomFpcMonInputSel		
Description	This parameter is used to select the monitor input signal. IOM_PNIN_0 – signal input from port logic is selected IOM_MON0_1 – monitor input signal 0 is selected IOM_MON1_2 - monitor input signal 1 is selected IOM_MON2_3 - monitor input signal 2 is selected		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef

Table 160 Specification for IomFpcMonInputSel (continued)

Range	IOM_PNIN_0 IOM_MON0_1 IOM_MON1_2 IOM_MON2_3		
Default value	IOM_PNIN_0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.8.5 IomFpcReferInputSel

Table 161 Specification for IomFpcReferInputSel

Name	IomFpcReferInputSel		
Description	This parameter is used to select the reference input signal. IOM_PNIN_0 – signal input from port logic is selected IOM_REF0_1 – reference input signal 0 is selected IOM_REF1_2 – reference input signal 1 is selected IOM_REF2_3 – reference input signal 2 is selected. IOM_GTMC_4-referenece input 3 is selected		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_PNIN_0 IOM_REF0_1 IOM_REF1_2 IOM_REF2_3 IOM_GTMC_4		
Default value	IOM_PNIN_0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.8.6 IomFpcResetTimer

Table 162 Specification for IomFpcResetTimer

Name	IomFpcResetTimer		
Description	Indicates whether FPC reset timer should be decremented or cleared on the glitch. IOM_TIMER_DECREMENT – Timer FPCk is decremented on the glitch. IOM_TIMER_CLEAR – Timer FPCk is cleared on the glitch.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_TIMER_DECREMENT IOM_TIMER_CLEAR		
Default value	IOM_TIMER_DECREMENT		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9 Container: IomLamConfiguration

This container holds the Lam Configuration.
Multiplicity Configuration Class: - Pre-Compile

3.3.1.9.1 IomLamHwUnit

Table 163 Specification for IomLamHwUnit

Name	IomLamHwUnit		
Description	Identification number for LAM unit.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-15		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.2 IomLamThreshold

Table 164 Specification for IomLamThreshold

Name	IomLamThreshold		
Description	This parameter is used to set the threshold value for event window counter from which an event is generated.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0-16777215		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.3 IomLamInvReferSignal

Table 165 Specification for IomLamInvReferSignal

Name	IomLamInvReferSignal		
Description	This parameter is used to enable/disable inversion of the reference signal to LAM FALSE – disables inversion of the reference signal to selected LAM module. TRUE – enables inversion of the reference signal to selected LAM module.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.4 IomLamInvMonSignal

Table 166 Specification for IomLamInvMonSignal

Name	IomLamInvMonSignal
Description	This parameter is used to enable/disable inversion of monitor signal to LAM

Table 166 Specification for IomLamInvMonSignal (continued)

	FALSE – disables inversion of monitor signal to selected LAM module. TRUE – enables inversion of monitor signal to selected LAM module.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.5 IomLamInvEventWin

Table 167 Specification for IomLamInvEventWin

Name	IomLamInvEventWin		
Description	This parameter is used to enable/disable inversion of event window in the LAM module FALSE – disables inversion of event window signal in selected LAM module. TRUE – enables inversion of event window signal in selected LAM module.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.6 IomLamMonSrcSelect

Table 168 Specification for IomLamMonSrcSelect

Name	IomLamMonSrcSelect
Description	The parameter defines whether monitor signal sourced directly or EXOR'd with a reference signal.

Table 168 Specification for IomLamMonSrcSelect (continued)

	IOM_MON_SIGNAL_FPCM – monitor signal is sourced directly from FPC monitor signal IOM_MON_SIGNAL_EXOR_FPCM – monitor signal is EXOR'd with FPC reference signal		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_MON_SIGNAL_FPCM IOM_MON_SIGNAL_EXOR_FPCM		
Default value	IOM_MON_SIGNAL_FPCM		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.7 IomLamRunMode

Table 169 Specification for IomLamRunMode

Name	IomLamRunMode		
Description	The parameter defines whether event window generation is free running or gated with monitor or reference signal. IOM_EVENT_WINDOW_FREE_RUNNING – event window generation is free running. IOM_EVENT_WINDOW_GATED – event window generation is gated with monitor or reference signal.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_EVENT_WINDOW_FREE_RUNNING IOM_EVENT_WINDOW_GATED		
Default value	IOM_EVENT_WINDOW_FREE_RUNNING		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.8 IomLamEventWinSelect

Table 170 Specification for IomLamEventWinSelect

Name	IomLamEventWinSelect

Table 170 Specification for IomLamEventWinSelect (continued)

Description	The parameter defines whether event window generation is from monitor signal or reference signal. IOM_EVENT_WIN_GEN_REFER – event window generation is determined from the reference signal. IOM_EVENT_WIN_GEN_MON – event window generation is determined from the monitor signal.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_EVENT_WIN_GEN_REFER IOM_EVENT_WIN_GEN_MON		
Default value	IOM_EVENT_WIN_GEN_REFER		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.9 IomLamDisableEvents

Table 171 Specification for IomLamDisableEvents

Name	IomLamDisableEvents		
Description	The parameter defines whether to suppress alarm outputs from LAM block to the ECM. FALSE – disables alarm output from LAM to ECM. TRUE – enables alarm output from LAM to ECM.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.10 IomLamEveWinActiveEdgeSelect

Table 172 Specification for IomLamEveWinActiveEdgeSelect

Name	IomLamEveWinActiveEdgeSelect		
Description	The parameter defines which active edges of reference and monitor signals are used for event window generation. IOM_NEITHER_CLR_NEITHER_GATE – neither edge used to clear event window counter and gate event generation. IOM_NEITHER_CLR_POS_GATE - neither edge used to clear event window counter and positive edge used to gate event generation IOM_NEITHER_CLR_NEG_GATE - neither edge used to clear event window counter and negative edge used to gate event generation IOM_NEITHER_CLR_EITHER_GATE - neither edge used to clear the event window counter and either edge used to gate event generation. IOM_POS_CLR_NEITHER_GATE - positive edge used to clear event window counter and neither edge used to gate event generation. IOM_POS_CLR_POS_GATE - positive edge used to clear event window counter and gate event generation. IOM_POS_CLR_NEG_GATE - positive edge used to clear event window counter and negative edge used to gate event generation. IOM_POS_CLR_EITHER_GATE - positive edge used to clear event window counter and either edge used to gate event generation. IOM_NEG_CLR_NEITHER_GATE - negative edge used to clear event window counter and neither edge used to gate event generation. IOM_NEG_CLR_POS_GATE – negative edge used to clear event window counter and positive edge used to gate event generation. IOM_NEG_CLR_NEG_GATE - negative edge used to clear event window counter and to gate event generation. IOM_NEG_CLR_EITHER_GATE - negative edge used to clear event window counter and either edge used to gate event generation. IOM_EITHER_CLR_NEITHER_GATE - either edge used to clear event window counter and neither edge used to gate event generation. IOM_EITHER_CLR_POS_GATE - either edge used to clear event window counter and positive edge used to gate event generation. IOM_EITHER_CLR_NEG_GATE - either edge used to clear event window counter and negative edge used to gate event generation. IOM_EITHER_CLR_EITHER_GATE - either edge used to clear event window counter and to gate event generation.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_NEITHER_ CLR_NEITHER_ GATE IOM_NEITHER_CLR_POS_GATE IOM_NEITHER_CLR_NEG_GATE IOM_NEITHER_CLR_EITHER_GATE IOM_POS_CLR_NEITHER_GATE IOM_POS_CLR_POS_GATE		

Table 172 Specification for IomLamEveWinActiveEdgeSelect (continued)

	IOM_POS_CLR_NEG_GATE IOM_POS_CLR_EITHER_GATE IOM_NEG_CLR_NEITHER_GATE IOM_NEG_CLR_POS_GATE IOM_NEG_CLR_NEG_GATE IOM_NEG_CLR_EITHER_GATE IOM_EITHER_CLR_NEITHER_GATE IOM_EITHER_CLR_POS_GATE IOM_EITHER_CLR_NEG_GATE IOM_EITHER_CLR_EITHER_GATE		
Default value	IOM_NEITHER_ CLR_NEITHER_ GATE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.11 IomLamMonInputSel

Table 173 Specification for IomLamMonInputSel

Name	IomLamMonInputSel		
Description	A parameter to select the monitor output signal from FPC block to LAM block.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_MONITOR_SIGNAL_FPCx x varies from 00 to 15		
Default value	IOM_MONITOR_SIGNAL_FPC00		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.9.12 IomLamRefInputSel

Table 174 Specification for IomLamRefInputSel

Name	IomLamRefInputSel

Table 174 Specification for IomLamRefInputSel (continued)

Description	A parameter to select the reference output signal from FPC block to LAM block.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	IOM_REFER_SIGNAL_FPCx x varies from 00 to 15		
Default value	IOM_ REFER _SIGNAL_FPC00		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.10 Container: IomClcConfiguration

This container holds the Clc Configuration.
Multiplicity Configuration Class: - Pre-Compile

3.3.1.10.1 IomClcSleepModeEn

Table 175 Specification for IomClcSleepModeEn

Name	IomClcSleepModeEn		
Description	Used to enable or disable sleep mode of the module.		
Multiplicity	0..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.10.2 IomClcRmcVal

Table 176 IomClcRmcVal

Name	IomClcRmcVal
Description	Determines 8 bit clock divider value in the RUN mode.

Table 176 IomClcRmcVal (continued)

Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1-255		
Default value	1		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.1.11 Container: Iom

Configuration of the Iom(Input Output Manager)

Multiplicity Configuration Class: -

3.3.1.11.1 Config Variant

Table 177 Specification of Config Variant

Name	Config Variant		
Description	-		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	Variant Post Build: Post Build Support		
Default value	Variant Post Build		
Post-build variant value	False	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

3.3.2 Functions – Type definitions

This section describes all the type definitions used by APIs.

3.3.2.1 Iom_RstStatusType

Table 178 Specification for Iom_RstStatusType

Syntax	Iom_RstStatusType
Type	uint8

Table 178	Specification for Iom_RstStatusType (continued)	
File	Iom.h	
Range	0	No kernel reset was executed
	1	Kernel reset was executed
	255	Indicates invalid value
Description	Indicates the reset status of the kernel.	
Source	IFX	

3.3.2.2 Iom_Ecm_ThresType

Table 179	Specification for Iom_Ecm_ThresType	
Syntax	Iom_Ecm_ThresType	
Type	uint8	
File	Iom.h	
Range	0-15	Threshold count value
	255	Indicates invalid value
Description	Indicates the threshold value of the counter in ECM.	
Source	IFX	

3.3.2.3 Iom_Fpc_CompareType

Table 180	Specification for Iom_Fpc_CompareType	
Syntax	Iom_Fpc_CompareType	
Type	uint32	
File	Iom.h	
Range	0x0 – 0xFFFF	Fpc compare value
	0xFFFFFFFF	Indicates invalid value
Description	Indicates the compare value of the FPC.	
Source	IFX	

3.3.2.4 Iom_FpcStatusType

Table 181	Specification for Iom_FpcStatusType	
Syntax	Iom_FpcStatusType	
Type	uint 8	
File	Iom.h	
Range	0-3	Fpc edge status

Table 181	Specification for Iom_FpcStatusType (continued)	
	255	Indicates invalid value
Description	Indicates the value of the FPC edge status	
Source	IFX	

3.3.2.5 Iom_Ecm_EveHistype

Table 182	Specification for Iom_Ecm_EveHisType	
Syntax	Iom_Ecm_EveHisType	
Type	uint32	
File	Iom.h	
Range	0x0 – 0xFFFF	ECM event trigger history
	0xFFFFFFFF	Indicates invalid value
Description	Indicates the ECM event history	
Source	IFX	

3.3.2.6 Iom_Lam_Configtype

Table 183	Specification for Iom_Lam_ConfigType
Syntax	Iom_Lam_ConfigType
Type	uint32
File	Iom.h
Range	0x0 - 0xFFFFFFFFu
Description	Indicates to the Lam Configuration Value
Source	IFX

3.3.2.7 Iom_Lam_ThresType

Table 184	Specification for Iom_Lam_ThresType
Syntax	Iom_Lam_ThresType
Type	Uint32
File	Iom.h
Range	0-0xFFFFFFFFu
Description	Indicates the threshold value of the Lam.
Source	IFX

3.3.2.8 Iom_Lam_CountType

Table 185 Specification for Iom_Lam_CountType

Syntax	Iom_Lam_CountType	
Type	uint32	
File	Iom.h	
Range	0x0-0xFFFFFF	LAM count
	0xFFFFFFFF	Indicates invalid value
Description	Indicates to the count value of the Lam event.	
Source	IFX	

3.3.2.9 Iom_Ecm_EveSelType

Table 186 Specification for Iom_Ecm_EveSelType

Syntax	Iom_Ecm_EveSelType	
Type	Uint32	
File	Iom.h	
Range	0-0xFFFFFu	ECM global event selection
	0xFFFFFFFF	Indicates invalid value
Description	Indicates the value of the global event selection register.	
Source	IFX	

3.3.2.10 Iom_EventHistory

Table 187 Specification for Iom_EventHistory

Syntax	Iom_EventHistory
Type	Enumeration
File	Iom.h
Range	IOM_EVETRIG_HISTORY_A = 0U, IOM_EVETRIG_HISTORY_B = 1U, IOM_EVETRIG_HISTORY_C = 2U, IOM_EVETRIG_HISTORY_D = 3U,
Description	Selects the history of the events recorded.
Source	IFX

3.3.2.11 Iom_FpcConfigType

Table 188 Specification for Iom_FpcConfigType

Syntax	Iom_FpcConfigType	
Type	Structure	
File	Iom.h	
Range	uint32 FpcCfg	FPC control value and compare value
	Uint16 FpcUnitNo	FPC unit Id
Description	Type for the definition of Fpc Module	
Source	IFX	

3.3.2.12 Iom_LamConfigType

Table 189 Specification for Iom_LamConfigType

Syntax	Iom_LamConfigType	
Type	Structure	
File	Iom.h	
Range	uint32 LamentWinCount	LAM event window threshold
	uint32 LamCfg	LAM configuration register value
	uint16 LamNo	LAM unit Id
Description	Type definition of the Lam module.	
Source	IFX	

3.3.2.13 Iom_EcmConfigType

Table 190 Specification for Iom_EcmConfigType

Syntax	Iom_EcmConfigType	
Type	structure	
File	Iom.h	
Range	uint32 EcmCountConfig	ECM counter configuration register value
	uint32 EcmGlobEntSel	ECM global event selection register value
Description	Type definition for the ECM module.	
Source	IFX	

3.3.2.14 Iom_ConfigType

Table 191 Specification for Iom_ConfigType

Syntax	Iom_ConfigType

Table 191	Specification for Iom_ConfigType (continued)
Type	Structure
File	Iom.h
Description	Defines the type for data structure containing the set of configuration parameters required for initializing the IOM driver.
Source	IFX

3.3.3 Functions - APIs

This section lists the APIs provided along with a short description of the functionality.

3.3.3.1 Iom_Init

Table 192	Specification for Iom_Init API	
Syntax	void Iom_Init (const Iom_ConfigType * const ConfigPtr)	
Service ID	0x5F	
Sync/Async	Synchronous	
ASIL Level	QM	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration set
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This API initializes the IOM driver. This function will initialize all relevant registers of IOM hardware with the values of structure referenced by the parameter ConfigPtr. The IOM initialization status is set at the end of the Initialization function execution.	
Source	IFX	
Error handling	DET: IOM_E_INIT: IOM driver initialization requested despite the driver already initialized. IOM_E_PARAM_CONFIG: This error is reported if the API is been invoked with invalid configuration pointer. DEM: IOM_E_CLC_ENABLE_ERR: This error is reported when enabling of CLC (module clock) fails.	
Configuration dependencies	-	

3.3.3.2 Iom_DeInit

Table 193 Specification for Iom_DeInit API

Syntax	void Iom_DeInit (void)	
Service ID	0x60	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
ASIL Level	QM	
Parameters (in)	-	-
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This API deinitializes the IOM driver. Service for deinitializing all hardware registers to their power on reset state. This API is only available when IomDeInitApi is configured as true	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error reported when API service used without module initialization.	
Configuration dependencies	-	

3.3.3.3 Iom_ResetKernel

Table 194 Specification for Iom_ResetKernel API

Syntax	void Iom_ResetKernel (void)	
Service ID	0x61	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	-	-
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	
Description	This API resets the IOM module kernel.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization.	
Configuration dependencies	-	

3.3.3.4 Iom_GetResetStatus

Table 195 Specification for Iom_GetResetStatus API

Syntax	Iom_RstStatusType Iom_GetResetStatus (void)	
Service ID	0x62	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	-	-
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_RstStatusType	Reset status for IOM kernel.
Description	This API returns the reset status for IOM kernel.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization.	
Configuration dependencies	-	

3.3.3.5 Iom_ClrResetStatus

Table 196 Specification for Iom_ClrResetStatus API

Syntax	void Iom_ClrResetStatus (void)	
Service ID	0x63	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	-	-
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This service clear the kernel reset status bit.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization.	
Configuration dependencies	-	

3.3.3.6 Iom_ClrFpcEdgeStatus

Table 197 Specification for Iom_ClrFpcEdgeStatus API

Syntax	void Iom_ClrFpcEdgeStatus (const uint8 FpcNo, const uint8 Edge)	
Service ID	0x64	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	FpcNo	FPC unit number
	Edge	Indicates rising edge or falling edge or both edges to be cleared.
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This API provides service to clear rising, falling or both edge.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_FPC: Error Reported when API service called with invalid FPC number. IOM_E_PARAM_EDGE: Error Reported when API service called with an invalid edge.	
Configuration dependencies	-	

3.3.3.7 Iom_GetFpcEdgeStatus

Table 198 Specification for Iom_GetFpcEdgeStatus API

Syntax	Iom_FpcStatusType Iom_GetFpcEdgeStatus (const uint8 FpcNo, const uint8 Edge)	
Service ID	0x65	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	FpcNo	FPC unit number
	Edge	Indicates rising edge or falling edge or both edges to be cleared
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_FpcStatusType	Indicates the value of the FPC edge status
Description	This API provides service to read and return the FPC edge status register value.	

Table 198 Specification for Iom_GetFpcEdgeStatus API (continued)

Source	IFX
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_FPC: Error Reported when API service called with invalid FPC number. IOM_E_PARAM_EDGE: Error Reported when API service called with an invalid edge.
Configuration dependencies	-

3.3.3.8 Iom_SetFpcCompare

Table 199 Specification for Iom_SetFpcCompare API

Syntax	void Iom_SetFpcCompare (const uint8 FpcNo, const uint16 CompVal)	
Service ID	0x66	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	FpcNo	FPC unit number
	Edge	Compare value of the FPC unit
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This API provides service to set FPC compare value.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_FPC: Error Reported when API service called with invalid FPC number.	
Configuration dependencies	-	

3.3.3.9 Iom_GetFpcCompare

Table 200 Specification for Iom_GetFpcCompare API

Syntax	Iom_Fpc_CompareType Iom_GetFpcCompare (const uint8 FpcNo)
Service ID	0x67
Sync/Async	Synchronous
Reentrancy	Reentrant
ASIL Level	QM

Table 200 Specification for Iom_GetFpcCompare API (continued)

Parameters (in)	FpcNo	Fpc unit number
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_Fpc_CompareType	Indicates the compare value of the Fpc
Description	This API provides service to set FPC compare value.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_FPC: Error Reported when API service called with invalid FPC number.	
Configuration dependencies	-	

3.3.3.10 Iom_SetLamConfig

Table 201 Specification for Iom_SetLamConfig API

Syntax	void Iom_SetLamConfig (const uint8 LamNo, const uint32 ConfigVal)	
Service ID	0x68	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	LamNo	LAM unit number
	ConfigVal	LAM configuration value
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	
Description	This API provides service to set LAM configuration.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_LAM: Error Reported when API service called with invalid LAM number.	
Configuration dependencies	-	

3.3.3.11 Iom_GetLamConfig

Table 202 Specification for Iom_GetLamConfig API

Syntax	Iom_Lam_ConfigType Iom_GetLamConfig (const uint8 LamNo)	
Service ID	0x69	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	LamNo	Lam unit number
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_Lam_ConfigType	Definition for Iom_Lam_ConfigType
Description	This API provides service to get LAM configuration.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_LAM: Error Reported when API service called with invalid LAM number.	
Configuration dependencies	-	

3.3.3.12 Iom_SetLamThreshold

Table 203 Specification for Iom_SetLamThreshold API

Syntax	void Iom_SetLamThreshold(const uint8 LamNo, const uint32 ThresVal)	
Service ID	0x6A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	LamNo	LAM unit number
	ThresVal	The threshold value of the LAM unit
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This API provides service to set the threshold value of the LAM unit.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization.	

Table 203	Specification for Iom_SetLamThreshold API (continued)
	IOM_E_PARAM_LAM: Error Reported when API service called with invalid LAM number. IOM_E_PARAM_THRES: Error Reported when API service called with an invalid threshold value.
Configuration dependencies	-

3.3.3.13 Iom_GetLamThreshold

Table 204	Specification for Iom_GetLamThreshold API	
Syntax	Iom_Lam_ThresType Iom_GetLamThreshold(const uint8 LamNo)	
Service ID	0x6B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	LamNo	LAM unit number
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_Lam_ThresType	Indicates the threshold value of the Lam
Description	This service is provided to read and return the selected LAM unit threshold value.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_LAM: Error Reported when API service called with invalid LAM number.	
Configuration dependencies	-	

3.3.3.14 Iom_GetLamEntWinCount

Table 205	Specification for Iom_GetLamEntWinCount API	
Syntax	Iom_Lam_CountType Iom_GetLamEntWinCount(const uint8 LamNo)	
Service ID	0x6C	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
ASIL Level	QM	
Parameters (in)	LamNo	LAM unit number
Parameters (out)	-	

Table 205 Specification for Iom_GetLamEntWinCount API (continued)

Parameters (in-out)	-	
Return	Iom_Lam_CountType	Indicates the Count value of the Lam event
Description	This service is provided to read and return LAM unit event window count register value.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_LAM: Error Reported when API service called with invalid LAM number.	
Configuration dependencies	-	

3.3.3.15 Iom_SetEcmGlobalEveSel

Table 206 Specification for Iom_SetEcmGlobalEveSel API

Syntax	void Iom_SetEcmGlobalEveSel(const uint32 EventSel)	
Service ID	0x6D	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	EventSel	Value to change ECM global event selection register.
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This service is provided to set/change ECM global event selection register.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_EVESEL: Error Reported when API service called with invalid event selection value.	
Configuration dependencies	-	

3.3.3.16 Iom_GetEcmGlobalEveSel

Table 207 Specification for Iom_GetEcmGlobalEveSel API

Syntax	Iom_Ecm_EveSelType Iom_GetEcmGlobalEveSel(void)

Table 207 Specification for Iom_GetEcmGlobalEveSel API (continued)

Service ID	0x6E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	void	-
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_Ecm_EveSelType	Indicates the value of the global event selection register.
Description		
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization.	
Configuration dependencies	-	

3.3.3.17 Iom_SetEcmThresVal

Table 208 Specification for Iom_SetEcmThresVal API

Syntax	void Iom_SetEcmThresVal(const uint8 CounterNo, const uint8 CountVal, const uint8 SelInput)	
Service ID	0x6F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	CounterNo	Counter number
	CounterVal	The threshold value of the selected counter
	SelInput	LAM channel output is routed to counter
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	
Description		
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization.	

Table 208 Specification for Iom_SetEcmThresVal API (continued)

	IOM_E_PARAM_CNT: Error Reported when API service called with the invalid counter value. IOM_E_PARAM_THRES: Error Reported when API service called with an invalid threshold value.
Configuration dependencies	-

3.3.3.18 Iom_GetEcmThresVal

Table 209 Specification for Iom_GetEcmThresVal API

Syntax	Iom_Ecm_ThresType Iom_GetEcmThresVal(const uint8 CounterNo)	
Service ID	0x70	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	CounterNo	Counter number in Ecm
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_Ecm_ThresType	Indicates the threshold value of the counter in ECMs
Description	This service is provided to read and return threshold value of the selected ECM counter.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_CNT: Error Reported when API service called with invalid ECM counter value.	
Configuration dependencies	-	

3.3.3.19 Iom_GetEcmEveTrigHis

Table 210 Specification for Iom_GetEcmEveTrigHis API

Syntax	Iom_Ecm_EveHisType Iom_GetEcmEveTrigHis(const Iom_EventHistory EveHistory)
Service ID	0x71
Sync/Async	Synchronous
Reentrancy	Reentrant
ASIL Level	QM

Table 210 Specification for Iom_GetEcmEveTrigHis API (continued)

Parameters (in)	EveHistory	Event trigger history recorded in ETA, ETB,ETC and ETD
Parameters (out)	-	
Parameters (in-out)	-	
Return	Iom_Ecm_EveHisType	Ecm event trigger history
Description	This service is provided to read and return the ECM event trigger history.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization. IOM_E_PARAM_EVEHSTRY: Error Reported when API service called with invalid ECM history.	
Configuration dependencies	-	

3.3.3.20 Iom_ClrEcmStatusHistory

Table 211 Specification for Iom_ClrEcmStatusHistory API

Syntax	void Iom_ClrEcmStatusHistory(void)	
Service ID	0x72	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
ASIL Level	QM	
Parameters (in)	void	-
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	This service will reset the ECM event trigger status history.	
Source	IFX	
Error handling	DET: IOM_E_UNINIT: Error Reported when API service used without module initialization.	
Configuration dependencies	-	

3.3.3.21 Iom_GetVersionInfo

Table 212 Specification for Iom_GetVersionInfo API

Syntax	void Iom_GetVersionInfo (Std_VersionInfoType * const versioninfo)

Table 212 Specification for Iom_GetVersionInfo API (continued)

Service ID	0x73	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
ASIL Level	QM	
Parameters (in)	versioninfo	Pointer to where to store the version information of the IOM driver
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	-
Description	API returns the version information of the IOM module. Note: This API is available only when IomVersionInfoApi is configured as true.	
Source	IFX	
Error handling	DET: IOM_E_PARAM_INVALID: Error is reported if API is invoked with null-pointer as a parameter.	
Configuration dependencies	IomVersionInfoApi	

3.3.4 Notifications and callbacks

The driver does not support any notifications and callbacks.

3.3.5 Scheduled functions

The driver does not support any scheduled functions.

3.3.6 Interrupt service routines

The driver does not support any interrupt handlers.

3.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

3.3.7.1 Development errors

The following table lists the development errors reported by the driver.

Table 213 Description of development errors reported

Description	Source	Error code and value	Applicable APIs
An API called before invocation of Iom_Init.	IFX	IOM_E_UNINIT= 0x11	Iom_DeInit Iom_ResetKernel Iom_GetResetStatus

Table 213 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
			Iom_ClrResetStatus Iom_ClrFpcEdgeStatus Iom_GetFpcEdgeStatus Iom_SetFpcCompare Iom_GetFpcCompare Iom_SetLamConfig Iom_GetLamConfig Iom_SetLamThreshold Iom_GetLamThreshold Iom_GetLamEntWinCount Iom_SetEcmGlobalEveSel Iom_GetEcmGlobalEveSel Iom_SetEcmThresVal Iom_GetEcmThresVal Iom_GetEcmEveTrigHis Iom_ClrEcmStatusHistory
API Iom_Init service called while the IOM a driver has already been initialized.	IFX	IOM_E_INIT= 0x10	Iom_Init
The error is reported if API is invoked with a null pointer.	IFX	IOM_E_PARAM_CONFIG=0x12	Iom_Init
The error is reported if API is invoked with null-pointer as a parameter.	IFX	IOM_E_PARAM_INVALID=0x13	Iom_GetVersionInfo
The error is reported if API is invoked with wrong FPC number.	IFX	IOM_E_PARAM_FPC=0x14	Iom_ClrFpcEdgeStatus Iom_GetFpcEdgeStatus Iom_SetFpcCompare Iom_GetFpcCompare
The error is reported if API is invoked with wrong LAM number.	IFX	IOM_E_PARAM_LAM=0x15	Iom_SetLamConfig Iom_GetLamConfig Iom_SetLamThreshold Iom_GetLamThreshold Iom_GetLamEntWinCount
The error is reported if API is invoked with wrong edge number.	IFX	IOM_E_PARAM_EDGE=0x16	Iom_ClrFpcEdgeStatus Iom_GetFpcEdgeStatus
The error is reported if API is invoked with an invalid threshold value.	IFX	IOM_E_PARAM_THRES=0x17	Iom_SetLamThreshold Iom_SetEcmThresVal

Table 213 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
The error is reported if API is invoked with invalid global event selection value.	IFX	IOM_E_PARAM_EVESEL=0x18	Iom_SetEcmGlobalEveSel
The error is reported if API is invoked with the invalid counter value.	IFX	IOM_E_PARAM_CNT=0x19	Iom_SetEcmThresVal Iom_GetEcmThresVal
The error is reported if API is invoked with invalid channel select value.	IFX	IOM_E_PARAM_CHNLSEL=0x20	Iom_SetEcmThresVal
The error is reported if API is invoked with invalid event history value.	IFX	IOM_E_PARAM_EVEHSTRY=0x21	Iom_GetEcmEveTrigHis

3.3.7.2 Production errors

Table 214 Description of production errors reported

Description	Source	Error code and value	Applicable APIs
This error is reported when enabling of CLC (module clock) fails.	IFX	IOM_E_CLC_ENABLE_ERR= Assigned by DEM	Iom_Init

3.3.7.3 Safety errors

The driver does not report any safety errors.

3.3.7.4 Runtime errors

The driver does not report any runtime errors.

3.3.8 Deviations and limitations

The section describes deviations and limitations from software specification.

3.3.8.1 Deviations

There are no deviations for the IOM driver.

3.3.8.2 Limitations

There are no limitations for the IOM driver.

3.3.9 Unsupported hardware features

Not applicable for the IOM driver.

4 SENT driver

4.1 User information

4.1.1 Description

The SENT driver provides the necessary configuration parameters and APIs to communicate with the external sensors over single I/O line for each channel. The SENT driver is implemented as a post-build variant.

The features of the SENT are:

- SENT interface provides a serial communication link typically used to connect sensors or other peripheral devices
- Clock control, address decoding and service request control are managed by the SENT module kernel
- SENT IP-module performs communication according to the SENT specification J2716 JAN2010
- Short PWM Code (SPC) protocol enables the use of enhanced protocol functionality like synchronous, range selection and ID selection protocol mode
- Message storage consists of two 32-bit registers for each channel, representing a flexible double buffer system

4.1.2 Hardware-software mapping

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

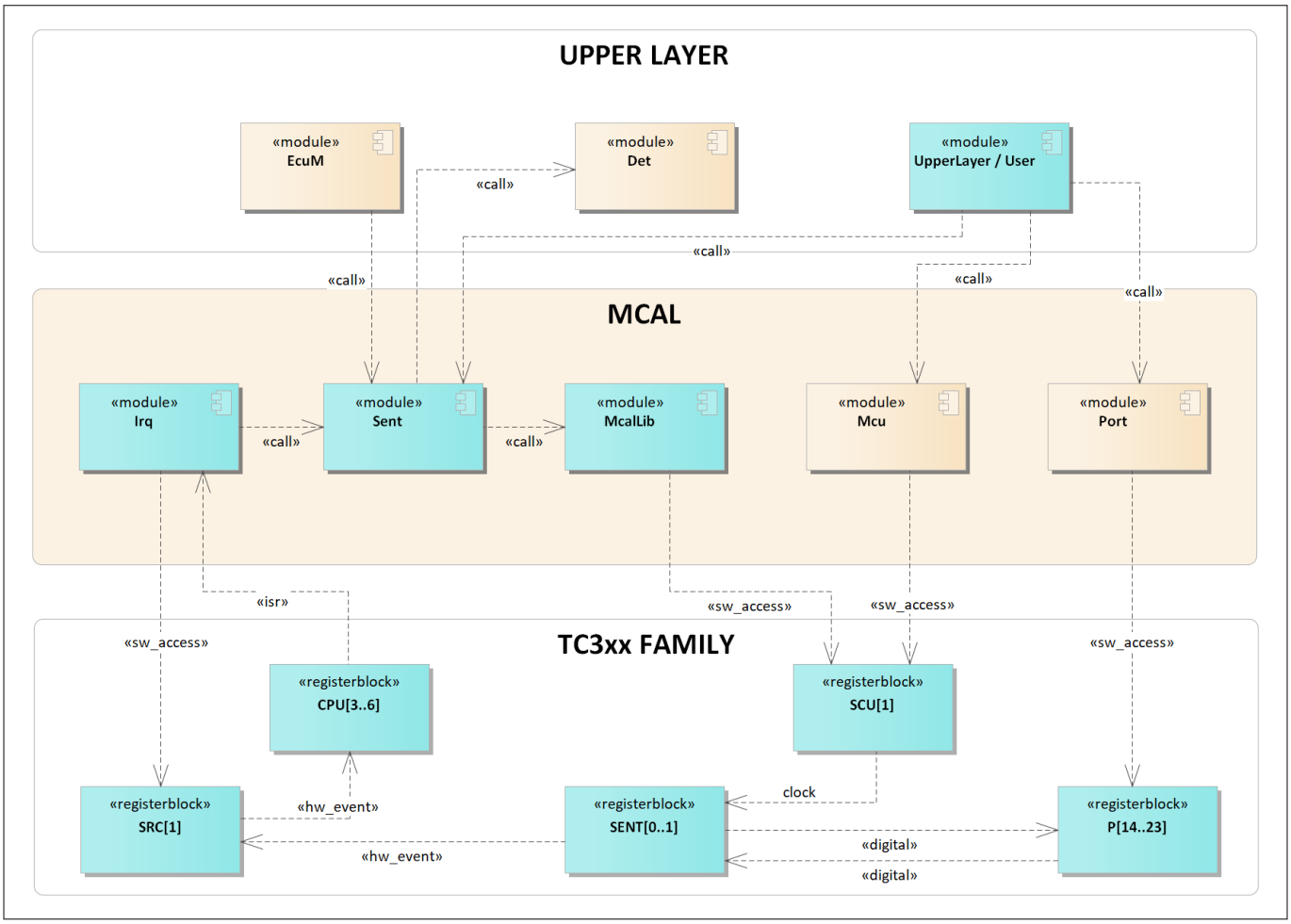


Figure 35 Mapping of hardware-software interfaces

4.1.2.1 SENT: primary hardware peripheral

Hardware functional features

The hardware features of each functional block configured by the driver are listed as follows:

- Reception of data in conformance according to the SENT standard
- Support for standard channel tick times (1 µs – 90 µs)
- Support for the SPC mode
- Digital glitch filter suppressing noise
- Time stamp generation
- Watchdog timer on incoming frames
- Interrupt generation for data reception, protocol error, buffer under-run, buffer over-run, watchdog error interrupts

Users of the hardware

The SENT driver exclusively utilizes the SENT module for its functionality.

Hardware diagnostic features

The SMU alarms configured for the SENT are not monitored by the SENT driver.

Hardware events

The SENT driver uses the following hardware events from the SENT IP:

- Receive success interrupt
- Receive data interrupt
- Receive buffer overflow interrupt
- Transfer data interrupt
- Transmit buffer underflow interrupt
- Frequency range interrupt
- Frequency drift interrupt
- Wrong number of nibble interrupt
- Nibble value out of range interrupt
- CRC error interrupt
- Wrong status and communication nibble interrupt
- Serial data receive interrupt
- Watch dog error interrupt

4.1.2.2 SCU: dependent hardware peripheral

Hardware functional features

The SENT driver depends on the SCU IP for the clock, ENDINIT and reset functionalities. The driver requires the fSPB and fSENT clock signals for functioning.

Users of the hardware

The SCU IP supplies clock for all the peripherals and the MCU driver, and 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 SENT driver.

Hardware events

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

4.1.2.3 Port: dependent hardware peripheral

Hardware functional features

The SPC data from SENT and the sensor data to the SENT and signal is routed to the SENT through the port pads. This is configured and enabled through the PORT driver

Users of the hardware

The port pads are configured by the PORT driver.

Hardware diagnostic features

Not applicable.

Hardware events

Hardware events from port pads are not used by the SENT driver.

4.1.2.4 SRC: dependent hardware peripheral

Hardware functional features

The SENT driver depends on the interrupt router for raising an interrupt to the CPU based on transmit, receive and error events, which indicates successful data transmission, reception and failure respectively.

Users of the hardware

The interrupt router is configured either by the IRQ driver or the user software. No functional block of the interrupt router is administrated by the SENT driver

Hardware diagnostic features

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

Hardware events

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

4.1.3 File structure

4.1.3.1 C file structure

The following diagram explains the file structure of the SENT driver.

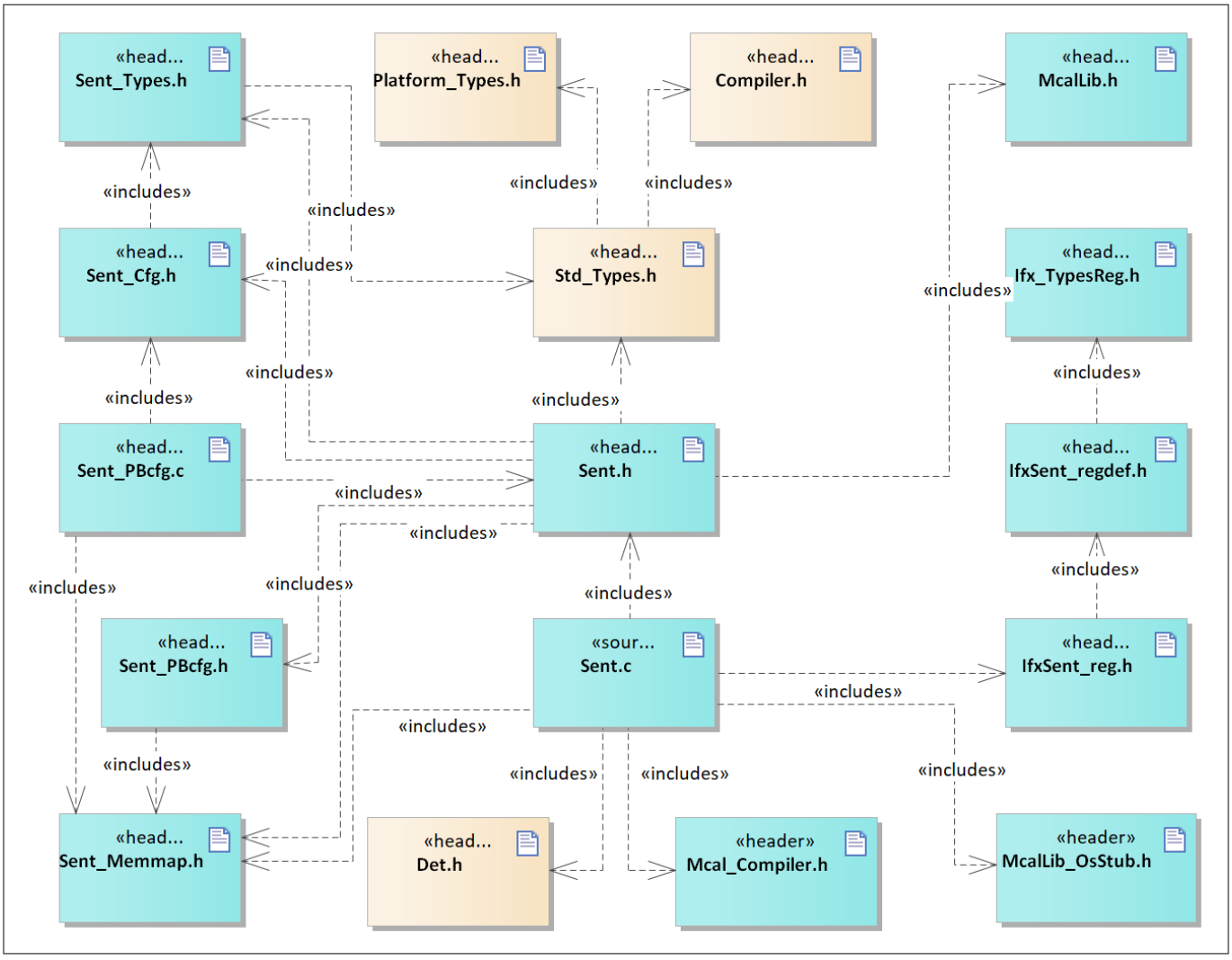


Figure 36 C file structure

Table 215 C file structure

File name	Description
`Platform_Types.h`	Platform-specific type declaration file as defined by AUTOSAR

Table 215 C file structure (continued)

File name	Description
Std_Types.h	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.
Compiler.h	Provides macros for the encapsulation of definitions and declarations
Det.h	Provides the exported interfaces of DET
McalLib.h	Header file (Static) defining prototypes of data structures and APIs of end-init and delay services and included by McalLib.c
McalLib_OsStub.h	McalLib_OsStub.h provides macros to support user mode of TriCore™
Sent_Types.h	The header file includes general LIN type declarations
Sent_MemMap.h	Mapping of code and data (variables, constant variables) to specific memory sections
Sent.h	Contains macros, type definitions and function prototypes of the SENT driver
Sent.c	Implementation of SENT driver functionality
Sent_Cfg.h	The pre-compile configuration macros required for the SENT driver implementation are present in this file
Sent_PBcfg.h	Contains SENT driver post build configuration parameter declaration
Sent_PBcfg.c	Contains SENT driver post build configuration parameters
IfxSent_reg.h	SFR header file for the SENT
IfxSent_regdef.h	Includes the register definition file for the SENT
Ifx_TypesReg.h	SFR header file

4.1.3.2 Code generator plugin files

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

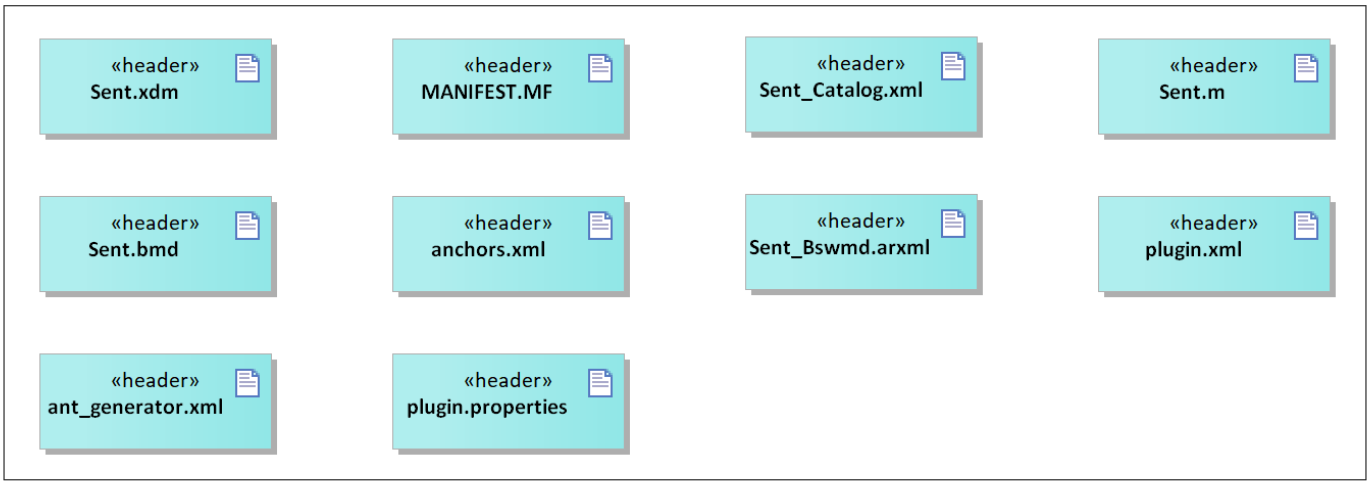


Figure 37 Code generator plugin files

Table 216 Code generator plugin files

File name	Description
`anchors.xml`	Tresos anchors support file for the SENT driver
`plugin.xml`	Tresos plugin support file for the SENT driver
`plugin.properties`	Tresos plugin support file for the SENT driver
`MANIFEST.MF`	Tresos plugin support file containing the metadata for the SENT driver
`ant_generator.xml`	Tresos support file to generate and rename multiple Post-Build configuration when using variation point feature
`Sent_Bswmd.arxml`	AUTOSAR format module description file
`Sent_Catalog.xml`	AUTOSAR format catalog file
`Sent.bmd`	AUTOSAR format XML data model schema file (for each device)
`Sent.m`	Code template macro file for the SENT driver
`Sent.xdm`	Tresos format XML data model schema file

4.1.4 Integration hints

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

4.1.4.1 Integration with AUTOSAR stack

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

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **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 `Sent_MemMap.h` file. The `Sent_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 below.

```
/***** GLOBAL RAM DATA -- NON CLEARED LMU *****/
#if defined SENT_START_SEC_VAR_CLEARED_QM_GLOBAL_8
/******User pragmas here *******/
#undef  SENT_START_SEC_VAR_CLEARED_QM_GLOBAL_8
#undef MEMMAP_ERROR
#elif defined SENT_STOP_SEC_VAR_CLEARED_QM_GLOBAL_8
/******User pragmas here *******/
#undef  SENT_STOP_SEC_VAR_CLEARED_QM_GLOBAL_8
#undef MEMMAP_ERROR
#elif defined SENT_START_SEC_VAR_CLEARED_QM_GLOBAL_32
/******User pragmas here *******/

#undef  SENT_START_SEC_VAR_CLEARED_QM_GLOBAL_32
#undef MEMMAP_ERROR
#elif defined SENT_STOP_SEC_VAR_CLEARED_QM_GLOBAL_32
/******User pragmas here *******/
#undef  SENT_STOP_SEC_VAR_CLEARED_QM_GLOBAL_32
#undef MEMMAP_ERROR
/**** CORE[x] CONFIG DATA -- PF[x] ****/ /*[x]=0..5*/
#elif defined SENT_START__SEC__CONFIG_DATA_QM_CORE[x]0_UNSPECIFIED
/******User pragmas here for PF[x] *******/
#undef SENT_START_SEC_CONFIG_DATA_QM_CORE[x]0_UNSPECIFIED
#undef MEMMAP_ERROR

#elif defined SENT_STOP_SEC_CONFIG_DATA_QM_CORE[x]0_UNSPECIFIED
/******User pragmas here for PF[x] *******/
#undef SENT_STOP_SEC_CONFIG_DATA_QM_CORE[x]0_UNSPECIFIED
#undef MEMMAP_ERROR

/***** CODE -- PF[x] *****/
#elif defined SENT_START_SEC_CODE_QM_GLOBAL
/******User pragmas here for PF[x] *******/
#undef SENT_START_SEC_CODE_QM_GLOBAL
#undef MEMMAP_ERROR

#elif defined SENT_STOP_SEC_CODE_QM_GLOBAL
/******User pragmas here for PF[x] *******/
#undef SENT_STOP_SEC_CODE_QM_GLOBAL
#undef MEMMAP_ERROR
#endif

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

- **DET**

The DET module is a part of the AUTOSAR stack that handles all the development and runtime errors reported by the BSW modules. The SENT driver reports all the development errors to the DET module through the `Det_ReportError()` API. The user of the SENT driver must process all the errors reported to the DET module through the `Det_ReportError()` API. The `Det.h` and `Det.c` files are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.

- **DEM**

  The DEM module is not required for the integration of the SENT driver.

- **SchM**

  The SchM is not required for the integration of the SENT driver.

- **Safety error**

  The SENT driver does not report any safety errors.

- **Notifications and callbacks**

  A callout function is linked uniquely with a SENT channel to be notified with the channel's interrupt events or any error/status events. The callout function prototype is defined by Sent_NotifFnPtrType. The callout functions fall under the MCAL layer and are allowed to access SENT registers if required. The application can determine the necessary action based on the event notifications. It is the responsibility of the user to define the SENT callout functions.

- **Operating system**

  OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application. The OS files provided by the MCAL package is only an example code and must be updated by the integrator with the actual OS files for the desired function.

4.1.4.2 Multicore and Resource Manager

The SENT driver supports execution of its APIs in parallel from all CPU cores. The user has to allocate resources of SENT to CPU cores at pre-compile time using the Resource Manager module. The following are the key points to be considered with respect to multicore in the driver:

- SENT channel of the SENT driver can be allocated to the CPU cores at pre-compile time.
- SENT channels that are not allocated to a CPU core shall be by default allocated to the master core.
- It must be ensured that the SENT channel ID passed as a parameter while invoking an API belongs to the same core on which the API is invoked.
- Initialization of the SENT channel must start with the master core initialization only after the successful initialization of the master core should there be a trigger for a slave core initialization. The SENT driver of the slave cores can be initialized simultaneously.
- De-initialization of the SENT driver for different slave cores can be initiated simultaneously. The master core de-initialization of the SENT driver should be carried out only after the de-initialization of the SENT driver in all the slave cores.
- DETs will be raised in case APIs are invoked with mismatch of CPU core and controller IDs or hardware object IDs.
- Interrupts raised by a hardware group must be serviced by the CPU core to which the hardware group has been allocated to.
- Locating constants, variables and configuration data to correct memory space should be done by the user. Memory sections are marked GLOBAL (common to all cores) and CORE[x](specific to a CPU core). The following should be considered by the user to ensure better performance of the driver:

  Code section

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

Data section

The RAM variable memory sections marked as specific to a core should be relocated to the DSPR/DLMU of the same core. The sections marked as global should be relocated to the non-cached LMU region.

Configuration data and constants

The configuration data sections marked as specific to a core should be relocated to the PFlash of the same core. 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.

Note: If the driver operates from a single (master) core, all the sections may be relocated to the PFlash/ DSPR/DLMU of the same CPU core.

4.1.4.3 MCU support

The SENT driver is dependent on the MCU driver for the clock configuration. The initialization of SENT driver must be started only after completion of the MCU initialization. The following must be considered while configuring the MCU driver in the EB tresos:

The fSENT defines the application clock frequency for the SENT Kernel. The fSENT which is derived from SPB (100 MHz) allows the SENT to operate at a constant baud rate (frequency). The required fSENT is 100 MHz.

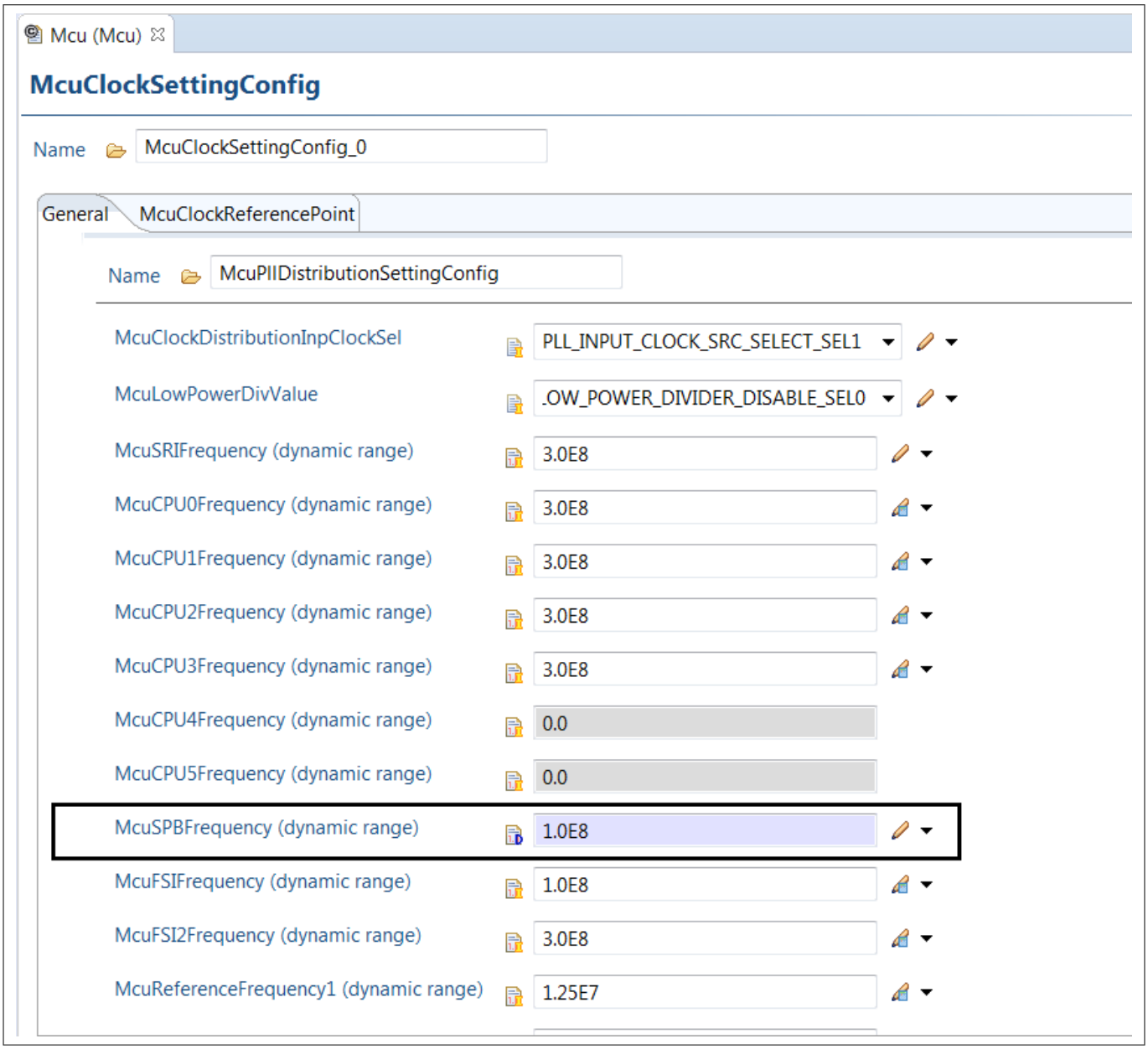


Figure 38 MCU Configuration

4.1.4.4 Port support

The PORT driver configures the port pins of the entire microcontroller. The user must configure the port pins used by the SENT driver through the port configuration and initialize the port pins prior to invoking the SENT initialization.

- **Port configuration for the Standard Sent operation**

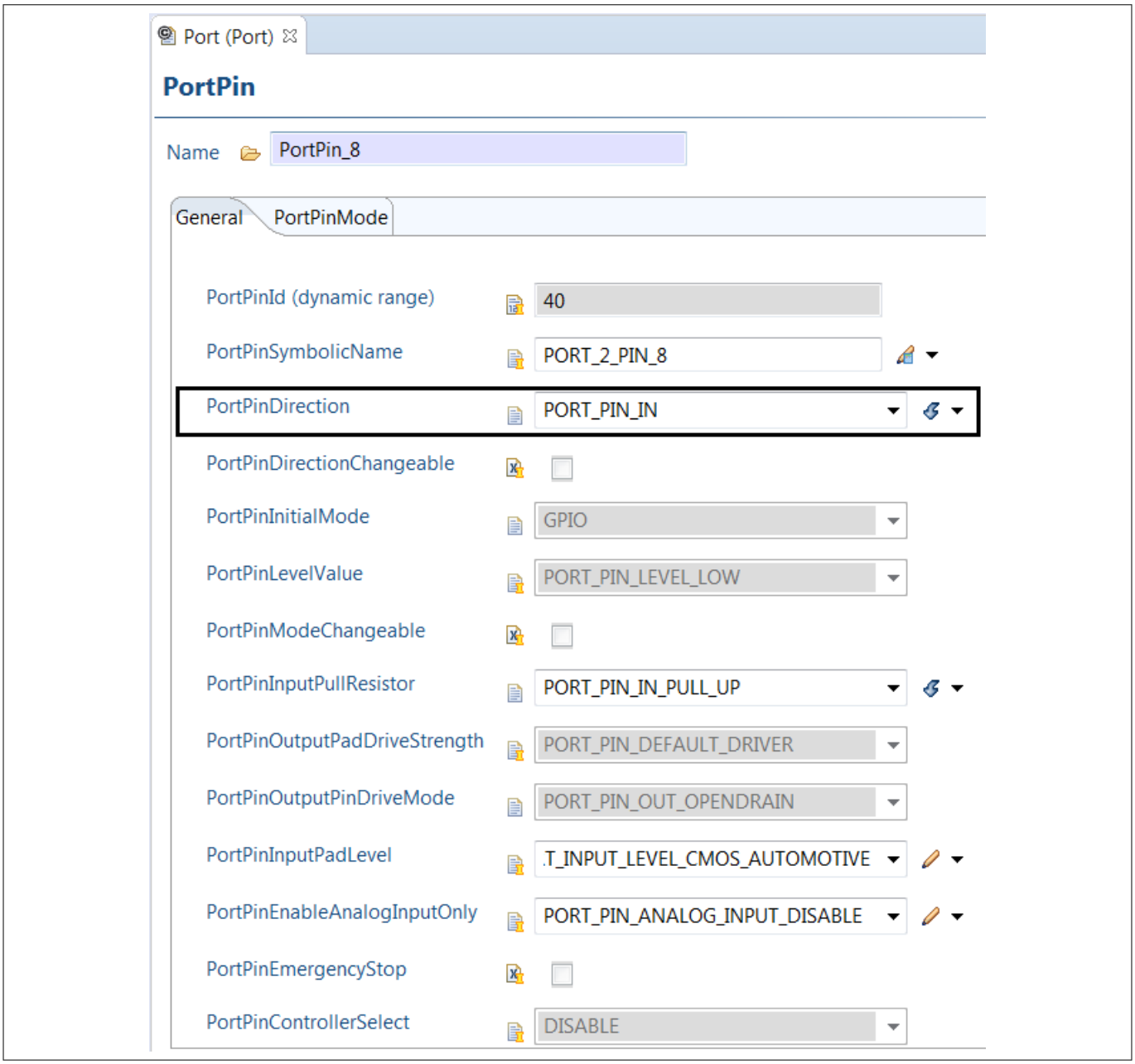


Figure 39 Port Configuration for standard mode

- **Port configuration for the SPC mode**

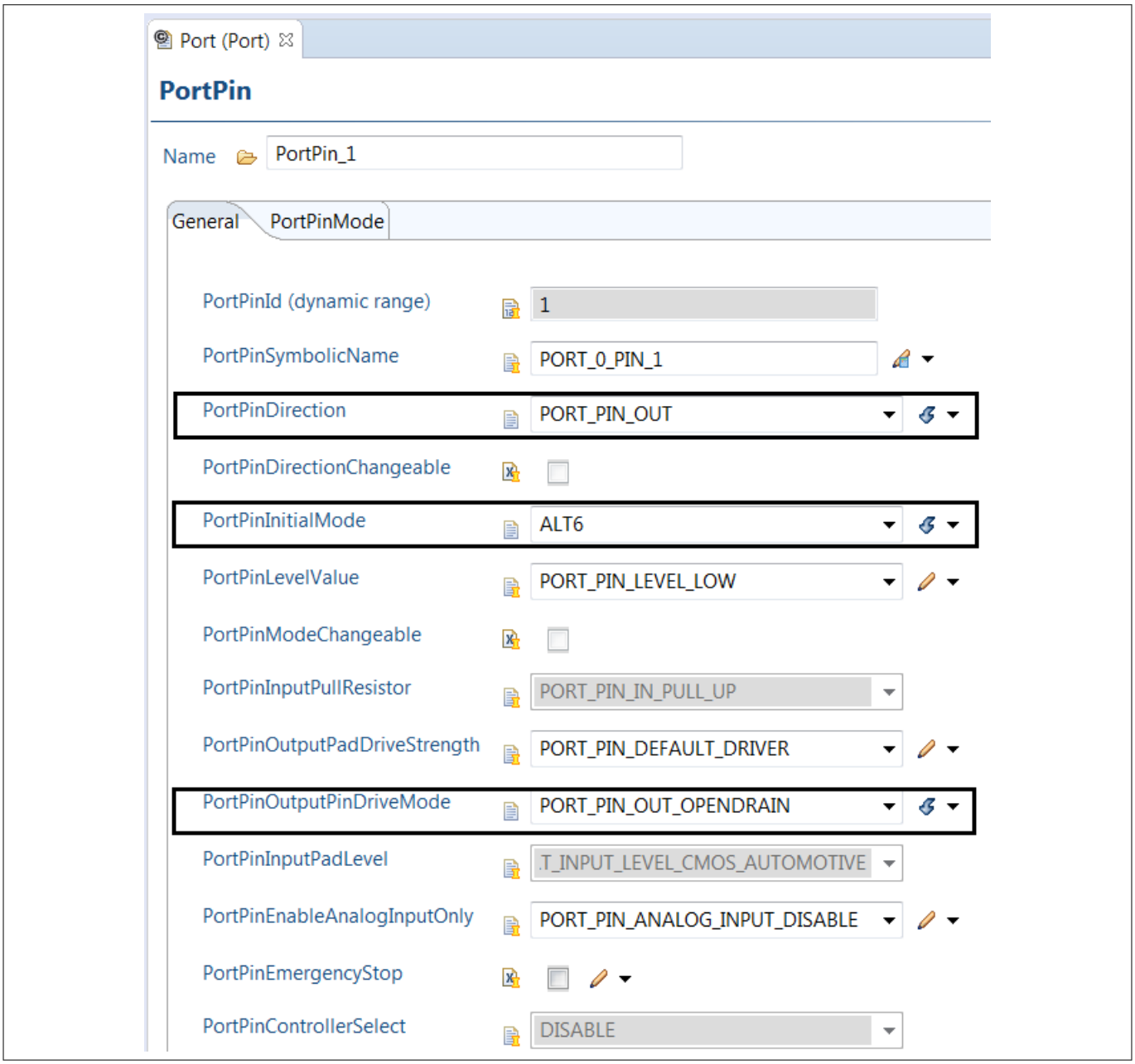


Figure 40 Port configuration for SPC mode

4.1.4.5 DMA support

The SENT driver does not use any services provided by the DMA driver.

4.1.4.6 Interrupt connections

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

The SENT driver is responsible for handling the SENT channel-specific interrupt requests and call the channel-specific registered callout function. Also, the callback functions/notifications configured should be unique for different channels. The SENT SRN interrupt handler shall invoke ISR Sent_Isr with the relevant interrupt node number. Also, each channel's interrupts are limited to a single interrupt node only. There are only 10 interrupt node available for SENT. User can configure one interrupt node to more than one SENT physical channel (for

example, SENT Physical Channel 0 linked to SRN2, SENT Physical Channel 1 linked to SRN0 and SENT Physical Channel 2 linked to SRN2 and so on).

RDI indicates a receive data interrupt. It is activated when a received frame is moved to a Receive Data Register (RDR). RSI indicates a receive frame success interrupt, that is, the CRC was successful. Both RDI and RSI will be issued together in normal use cases where the frame size is not bigger than 8 nibbles and the CRC is correct. RBI indicates a receive buffer overrun interrupt. It is activated when a new frame is transferred to a Receive Data Register RDR while the old value was still not read by the host (overwrite), that is, the kernel wants to set any of the two interrupts RSI and RDI and finds any of these two interrupts already set. TDI indicates a transmit interrupt. It is activated when data is moved from a SCR to a transmit shift register. TBI indicates a transfer buffer under run interrupt. It is set after data has been completely transferred (PLEN exceeded) and no new data was written to SCRx. In addition, the protocol error interrupts are available: FRI, FDI, NNI, NVI and CRCI. If one of the protocol interrupts is activated, data is to be treated as invalid according to SENT specification J2716 JAN2010. WSI, SDI SCRI treats the interrupts referring to the Status and Communication nibble. WDI is the Watch Dog Error Interrupt. It is issued if the time between two frames is too long.

```
#include "Sent.h"'
ISR(SENTSR0_ISR)
{
  /* Enable Global Interrupts */
  ENABLE();
Sent_Isr(0);
}
```

4.1.4.7 Example usage

Examples of SENT driver API usage are as follows:

4.1.4.7.1 Configuration of the driver

The SENT driver must be configured before usage and configuration files are generated and made available during the software build process.

To configure the SENT driver, the following guidelines shall be followed properly.

- Configuration of system clock: Before using the SENT driver, the MCU driver needs to be configured and initialized for the system clock and the system peripheral bus (SPB) clock. The SENT driver clock is derived from the SPB clock. This configuration is done using the MCU driver.
- Configuration of the port pins: For all the port pins that would be used by the SENT driver as input/output pins, configure the same in the PORT driver.
- Configuration of SENT interrupts: Configure the interrupt priority, type of service and interrupt type in the IRQ driver.
- Configuration of SENT driver: Select the required API configuration and choose channel dependent parameters like baud-rate, data length of the frame, CRC mode and so on.

Initialization of SENT driver

Refer to the *Integration hints* section and add all dependent modules. Follow the sequence in the application code:

1. Initialize the MCU and the clock `Mcu_Init` API.
2. Initialize the PORT driver using the `Port_Init` API.
3. Initialize the IRQ to enable the interrupt generation.
4. Initialize the SENT driver using the `Sent_Init` API.

Sample code for SENT driver initialization is as follows:

```
/* Mcu Initialization */
 Mcu_Init(&Mcu_Config);
 Mcu_InitClock(0U);
 while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
 Mcu_DistributePllClock ();
/* Port Initialization */
 Port_Init(&Port_Config);
/* SENT Initialization */
 Sent_Init(&Sent_Config);
/* Further APIs of SENT driver can be called now */
```

Enabling and disabling the channel

After SENT initialization the following sequence can be followed.

```
/* Enable Channel */
 Sent_SetChannel(ChannelId_0, SENT_ENABLE);
/* Disable Channel */
 Sent_SetChannel(ChannelId_0, SENT_DISABLE);
```

Reading data from standard SENT mode

```
/* Mcu Initialization */
  Mcu_Init(&Mcu_Config);
  Mcu_InitClock(0U);
  while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
  Mcu_DistributePllClock ();
 /* Port Initialization */
  Port_Init(&Port_Config);
 /* SENT Initialization */
  Sent_Init(&Sent_Config);
 /* Enable Channel */
  Sent_SetChannel(ChannelId_0, SENT_ENABLE);
  Delay(3);
  Sent_ReadData0 = Sent_ReadData(ChannelId_0);
```

Reading the data from SPC mode

```
/* Mcu Initialization */
  Mcu_Init(&Mcu_Config);
  Mcu_InitClock(0U);
  while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
  Mcu_DistributePllClock ();
 /* Port Initialization */
  Port_Init(&Port_Config);
 /* SENT Initialization */
  Sent_Init(&Sent_Config);
 /* Enable Channel */
  Sent_SetChannel(ChannelId_0, SENT_ENABLE);
  Delay(3);
  #if (SENT_SPC_USED == STD_ON)
  Sent_Spc.Mode = SYNC_MODE;
  Sent_Spc.Delay = 0;
  Sent_Spc.PulseLength = 3; /* 3 ticks */
  Sent_Spc.TimeBase = PULSE_LAST_SYNC_FREQ;
  Sent_Spc.TriggerSource = PULSE_START_IMMED;
  Sent_SpcGenPulse(ChannelId_0, &Sent_Spc);
  #endif
  Sent_ReadChannelStatus(ChannelId_0, &Sent_Stat);
  Sent_ReadData0 = Sent_ReadData(ChannelId_0);
```

4.1.5 Key architectural considerations

There are no key architectural considerations for the driver.

4.2 Assumptions of Use (AoU)

There are no AoUs for the driver.

4.3 Reference information

4.3.1 Configuration interfaces

The following diagram depicts the hierarchy along with the extensions provided for SENT module.

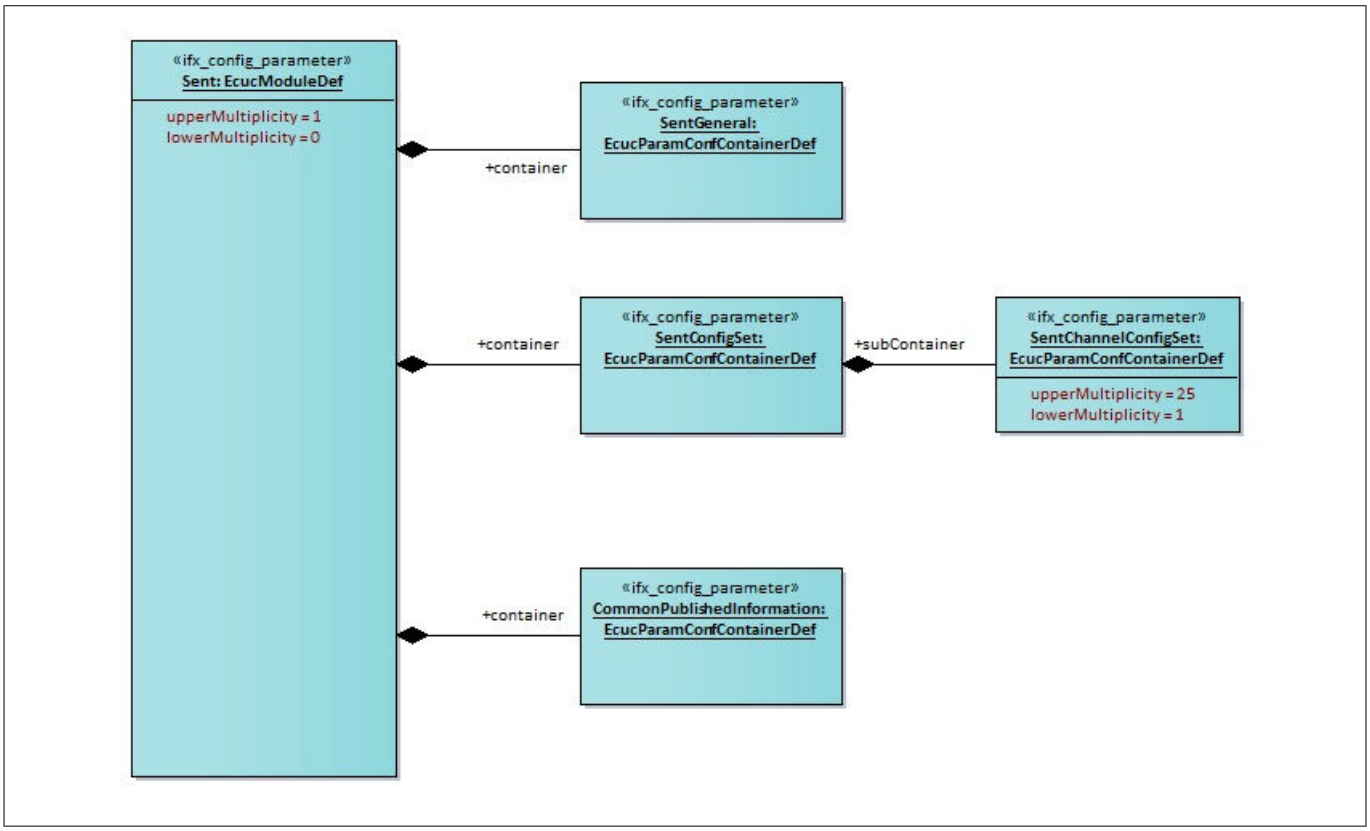


Figure 41 Container hierarchy along with their configuration parameters

4.3.1.1 Container: CommonPublishedInformation

This container contains published information about vendor and versions.

4.3.1.1.1 ArMajorVersion

Table 217 Specification for ArMajorVersion

Name	ArMajorVersion		
Description	Parameter provides the major version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.1.2 ArMinorVersion

Table 218 Specification for ArMinorVersion

Name	ArMinorVersion		
Description	Parameter provides the minor version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.1.3 ArPatchVersion

Table 219 Specification for ArPatchVersion

Name	ArPatchVersion		
Description	Parameter provides the patch version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.1.4 ModuleId

Table 220 Specification for ModuleId

Name	ModuleId		
Description	This parameter provides the module Id. The default value is set to 255 as this is the module ID of the SENT driver.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	0 - 255		
Default value	255		

Table 220 Specification for ModuleId (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.1.5 SwMajorVersion

Table 221 Specification for SwMajorVersion

Name	`SwMajorVersion`		
Description	Specifies the major version of the driver software.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	Scope	LOCAL
Dependency	-		

4.3.1.1.6 SwMinorVersion

Table 222 Specification for SwMinorVersion

Name	`SwMinorVersion`		
Description	Specifies the minor version of the driver software.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.1.7 SwPatchVersion

Table 223 Specification for SwPatchVersion

Name	SwPatchVersion		
Description	Specifies the patch version of the driver software.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.1.8 VendorId

Table 224 Specification for VendorId

Name	VendorId		
Description	Specifies the vendor ID for Infineon.		
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	-		

4.3.1.2 Container: Sent

This container contains the general configuration parameters of the SENT driver

4.3.1.2.1 Config Variant

Table 225 Specification for Config Variant

Name	Config Variant		
Description	Selects the config-variant for the SENT module.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef

Table 225 Specification for Config Variant (continued)

Range	VariantPostBuild: Post Build Support.		
Default value	VariantPostBuild		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.2.2 SentDeInitApi

Table 226 Specification for SentDeInitApi

Name	`SentDeInitApi`		
Description	Switches the DeInit Api ON or OFF. TRUE: enabled (ON). FALSE: disabled (OFF).		
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	-		

4.3.1.2.3 SentDevErrorDetect

Table 227 Specification for SentDevErrorDetect

Name	`SentDevErrorDetect`		
Description	Switches the Default Error Tracer (Det) detection and notification ON or OFF. TRUE: enabled (ON) FALSE: disabled (OFF)		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		

Table 227 Specification for SentDevErrorDetect (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.2.4 SentSpcFeatureSupport

Table 228 Specification for SentSpcFeatureSupport

Name	`SentSpcFeatureSupport`		
Description	Switches the SPC feature support ON or OFF. TRUE: enabled (ON) FALSE: disabled (OFF)		
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	-		

4.3.1.2.5 SentVersionInfoApi

Table 229 Specification for SentVersionInfoApi

Name	`SentVersionInfoApi`		
Description	Switches the Sent_GetVersionInfo function ON or OFF.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

Table 229 Specification for SentVersionInfoApi (continued)

Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.2.6 SentIndex

Table 230 Specification for SentIndex

Name	`SentIndex`		
Description	Specifies the Instance Id of this module instance. If only one instance is present it shall have the Id 0.		
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	-		

4.3.1.2.7 SentResetSfrAtInit

Table 231 Specification for SentResetSfrAtInit

Name	`SentResetSfrAtInit`		
Description	Switches the SFR reset at initialization ON or OFF. TRUE: enabled (ON) FALSE: disabled (OFF)		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
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	-		

4.3.1.2.8 SentInitDeInitApiMode

Table 232 Specification for SentInitDeInitApiMode

Name	SentInitDeInitApiMode		
Description	Defines the mode in which the Init and DeInit APIs will be used. The default value of this parameter is set to Supervisor to enable maximum access rights to the registers used by the SENT driver.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	SENT_MCAL_SUPERVISOR: operating mode used is Supervisory SENT_MCAL_USER1: operating mode used is USER-1		
Default value	SENT_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.2.9 SentMultiCoreErrorDetect

Table 233 Specification for SentMultiCoreErrorDetect

Name	SentMultiCoreErrorDetect		
Description	Switches the multi-core error detection and notification to ON or OFF. - TRUE: enabled (ON) - FALSE: disabled (OFF)		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
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	-		

4.3.1.3 Container: SentConfigSet

This container contains the module kernel specific configuration parameters.

4.3.1.3.1 SentSystemClock

Table 234 Specification for SentSystemClock

Name	SentSystemClock		
Description	This parameter refers to the system clock configured by MCU driver. This reference is used for BaudRate computation.		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: McuClockReferencePointConfig		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.3.2 SentSleepModeEnable

Table 235 Specification for SentSleepModeEnable

Name	SentSleepModeEnable		
Description	Switches the SentSleepModeEnable ON or OFF. TRUE: enabled (ON). FALSE: disabled (OFF).		
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	-		

4.3.1.3.3 SentModuleClkDiv

Table 236 Specification for SentModuleClkDiv

Name	SentModuleClkDiv
Description	This parameter refers to the 8-bit divider used to generate the SENT module clock.

Table 236 Specification for SentModuleClkDiv (continued)

	This value will be used to divide the MCU SPB clock f_{Spb} and derive the f_{Sent} SENT module clock.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	1 – 255		
Default value	1		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.3.4 SentBaudFracStep

Table 237 Specification for SentBaudFracStep

Name	`SentBaudFracStep`		
Description	This parameter value will generate the SENT fractional divider clock $f_{fracdiv}$ which is an input clock for all SENT channels. This parameter derives the clock as follows: $f_{fracdiv} = f_{SENT} / (1024 – SentBaudFracStep)$ where SentBaudFracStep = 0 - 1023.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	0 – 1023		
Default value	1023		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SentModuleClkDiv		

4.3.1.4 Container: SentChannelConfigSet

This container contains the channel specific configuration parameters.

4.3.1.4.1 SentChLogiIndex

Table 238 Specification for SentChLogiIndex

Name	`SentChLogiIndex`		
Description	This parameter refers to SENT logical channel number.		
Multiplicity	1	**Type**	EcucBooleanParamDef

Table 238 Specification for SentChLogiIndex (continued)

Range	0 – (x-1) where n is the Maximum No. of SENT channels available in a particular device.		
Default value	x where x is the index of the Sent Channel in the Config set.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SentModuleClkDiv		

4.3.1.4.2 SentChanPreDiv

Table 239 Specification for SentChanPreDiv

Name	`SentChanPreDiv`		
Description	This parameter refers to the setting of SENT channel pre-divider clock f_{pdiv_x} where x depends on the device variant. This parameter derives the clock as follows: $f_{pdiv\_x} = f_{fracdiv} / (SentChanPreDiv + 1)$		
Multiplicity	1	**Type**	EcucIntegerParamDef
Range	0 – 2047		
Default value	7		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SentBaudFracStep		

4.3.1.4.3 SentChanBaudDiv

Table 240 Specification for SentChanBaudDiv

Name	`SentChanBaudDiv`		
Description	This parameter value is used to derive the baud rate frequency for channel x (f_{tick_x}) where x depends on the device variant. This parameter derives the baud rate as follows: $f_{tick\_x} = f_{pdiv\_x} * 56 / SentChanBaudDiv$		
Multiplicity	1	**Type**	EcucIntegerParamDef
Range	2200 - 49100		
Default value	2200		

Table 240 Specification for SentChanBaudDiv (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SentBaudFracStep		

4.3.1.4.4 SentChanCRCMode

Table 241 Specification for SentChanCRCMode

Name	`SentChanCRCMode`		
Description	This parameter decides the CRC mode to be used for fast channel/slow channel data communication.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	SENT_STANDARD: Standard CRC Calculation as per standard SENT_IFX_ALTERNATE: Alternative CRC Calculation as used in IFX Hall Sensors		
Default value	SENT_STANDARD		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SentBaudFracStep		

4.3.1.4.5 SentChPhyIndex

Table 242 Specification for SentChPhyIndex

Name	`SentChPhyIndex`		
Description	This parameter refers to SENT physical channel number.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	SENT0: Signifies the physical channel 0. SENTx: This parameter signifies physical channel number, where x is varies from 0 to maximum number of units as per device variant. For example SENT0, SENT1,... SENTx, where x depends on device variant.		
Default value	SENT0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

Table 242 Specification for SentChPhyIndex (continued)

Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.6 SentRxInput

Table 243 Specification for SentRxInput

Name	`SentRxInput`		
Description	This parameter selects the alternate input for the RX signal for the given Sent channel.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	SENT_0_A: Signifies the receive input channel 0. SENT_0_x: This parameter signifies the receive input channel, where x is varies from 0 to maximum number of units as per device variant. For example SENT0, SENT1, ..., SENTx, where x depends on device variant.		
Default value	SENT_0_A		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.7 SentChanStatusNibbleCRCInc

Table 244 Specification for SentChanStatusNibbleCRCInc

Name	`SentChanStatusNibbleCRCInc`		
Description	This parameter defines whether status nibble should be used for CRC calculation.		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	FALSE: Status nibble not included for CRC calculation as per standard TRUE: Status nibble included for CRC calculation as used in IFX Hall Sensors		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL

Table 244	Specification for SentChanStatusNibbleCRCInc (continued)
Dependency	-

4.3.1.4.8 SentChanEnESF

Table 245 Specification for SentChanEnESF

Name	SentChanEnESF		
Description	This parameter decides whether standard serial mode or extended serial encoding mode should be used. If standard serial mode is used, processing will be done after 16 SENT frames (4-bit ID, 8-bit data, 4-bit CRC). If extended serial mode is used, processing will be done after 18 SENT frames (4 or 8-bit ID, 12 or 16-bit data, 6-bit CRC).		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	FALSE: Standard serial data encoding used. TRUE: Extended serial data encoding used.		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.9 SentChanSerialProcEn

Table 246 Specification for SentChanSerialProcEn

Name	SentChanSerialProcEn		
Description	This parameter decides whether automatic processing of serial data should be enabled or not. If enabled, serial data can be read through Sent_ReadSerialData once SDI interrupt has been activated. If not enabled, status nibble can be read manually through Sent_ReadChannelStatus for each SENT frame once RDI/RSI interrupt has been activated. The user should collate the serial data accordingly from the status nibbles of respective SENT frames as per standard		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	FALSE: Automatic serial data processing is disabled. TRUE: Automatic serial data processing is enabled.		
Default value	FALSE		

Table 246 Specification for SentChanSerialProcEn (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.10 SentChanSerialCrcDisable

Table 247 Specification for SentChanSerialCrcDisable

Name	`SentChanSerialProcEn`		
Description	This parameter decides whether the serial data's CRC should be verified internally by SENT hardware. If TRUE, then it is responsibility of the application to verify the CRC of the received serial data.		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	TRUE: Serial data CRC not verified by SENT hardware FALSE: Serial data CRC verified by SENT hardware		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.11 SentChanFrameCrcDisable

Table 248 Specification for SentChanFrameCrcDisable

Name	`SentChanFrameCrcDisable`		
Description	This parameter decides whether the serial data's CRC should be verified internally by SENT hardware. If TRUE, then it is responsibility of the application to verify the CRC of the received serial data.		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	TRUE: Serial data CRC not verified by SENT hardware FALSE: Serial data CRC verified by SENT hardware		
Default value	FALSE		

Table 248 Specification for SentChanFrameCrcDisable (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.12 SentChanFrameChk

Table 249 Specification for SentChanFrameChk

Name	`SentChanFrameChk`		
Description	This parameter decides whether the current SENT frame should be verified against preceding SENT frame/ last valid preceding SENT frame for successive sync pulse difference (> 1.5625 %). If SENT_PAST_SYNC_PULSE is selected, the sync pulse of the current frame is compared to sync pulse of the immediate preceded frame. This is the preferred option as per standard. If SENT_PAST_VALID_SYNC_PULSE is selected, the sync pulse of the current frame is compared to sync pulse of the last valid preceded frame.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	SENT_PAST_SYNC_PULSE: Check current SENT frame against past sync pulse SENT_PAST_VALID_SYNC_PULSE: Check current SENT frame against last valid sync pulse		
Default value	SENT_PAST_SYNC_PULSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.13 SentChanFrameDataLen

Table 250 Specification for SentChanFrameDataLen

Name	`SentChanFrameDataLen`
Description	This parameter determines the number of data nibbles per SENT frame. It does not include sync pulse, status nibble, CRC nibble, or the additional zero length nibble. If more than 8 nibbles are configured, RDI interrupt is issued each time 8 nibbles are written into RDR register of that channel. At the end of the last data frame also, RDI interrupt is issued. If no RDI interrupt occurs at the last data frame, an error has

Table 250 Specification for SentChanFrameDataLen (continued)

	occurred. RSI interrupt shall be issued at every successful receive of a single SENT frame.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	0 – 255		
Default value	6		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.14 SentChanDriftErrEn

Table 251 Specification for SentChanDriftErrEn

Name	`SentChanDriftErrEn`		
Description	This parameter determines whether drift errors should be enabled or not. Certain sensors triggered by SPC tend to have a long pause period and the accumulated drift could be more than 1.5625%, then it useful to disable this feature.		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	FALSE: Ignore drift errors TRUE: Drift errors enabled		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.15 SentChanCRZEn

Table 252 Specification for SentChanCRZEn

Name	`SentChanCRZEn`		
Description	If TRUE, augmentation is selected, (i.e. a ZERO NIBBLE is added at the end of CRC calculation (only in calculation)). E.g. as 7th nibble (in case of 6 data nibbles).		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	FALSE: Zero nibble is not augmented for CRC calculation		

Table 252 Specification for SentChanCRZEn (continued)

	TRUE: Zero nibble is augmented for CRC calculation		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.16 SentChanIgnoreEndPulse

Table 253 Specification for SentChanIgnoreEndPulse

Name	`SentChanIgnoreEndPulse`		
Description	This parameter determines whether end pulse should be ignored or not. For some systems with an end pulse, during synchronize or re-synchronize of reception, if calibration pulses are detected one immediately following the other, the first calibration pulse shall be ignored as it may be a pause pulse with duration matching the calibration pulse range.		
Multiplicity	1	**Type**	EcucBooleanParamDef
Range	FALSE: End pulse not ignored TRUE: End pulse ignored		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.17 SentChanInPulse

Table 254 Specification for SentChanInPulse

Name	`SentChanInPulse`		
Description	This parameter determines the pulse polarity of the respective input channel.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	SENT_ACTIVE_LOW: Pulse polarity is active low SENT_ACTIVE_HIGH: Pulse polarity is active high		
Default value	SENT_ACTIVE_LOW		

Table 254 Specification for SentChanInPulse (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.18 SentChanOutPulse

Table 255 Specification for SentChanOutPulse

Name	`SentChanOutPulse`		
Description	This parameter determines the pulse polarity of the respective input channel.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	SENT_ACTIVE_LOW: Pulse polarity is active low SENT_ACTIVE_HIGH: Pulse polarity is active high		
Default value	SENT_ACTIVE_LOW		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.19 SentChanGlitchFilterDepth

Table 256 Specification for SentChanGlitchFilterDepth

Name	`SentChanGlitchFilterDepth`		
Description	This parameter determines the number of input samples that should be taken into account for the digital glitch filter.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	0 – 15		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.20 SentChanDataView

Table 257 Specification for SentChanDataView

Name	SentChanDataView		
Description	This parameter determines the sequence in which the received data nibble shall be presented to the user for the respective channel. For example 0x76540123 means Received nibble 0 goes to bits 12-15 of RDR Received nibble 1 goes to bits 8-11 of RDR Received nibble 2 goes to bits 4-7 of RDR Received nibble 3 goes to bits 0-3 of RDR Received nibble 4 goes to bits 16-19 of RDR Received nibble 5 goes to bits 20-23 of RDR Received nibble 6 goes to bits 24-27 of RDR Received nibble 7 goes to bits 28-31 of RDR		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	0x01234567 – 0x76543210		
Default value	0x76543210		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.21 SentChanFreqDriftCheckLen

Table 258 Specification for SentChanFreqDriftCheckLen

Name	SentChanFreqDriftCheckLen		
Description	This parameter provides Frequency Drift Check based on Frame Length (FDFL). This is used for frames with pause pulse only. If set the drift error is not checked by HW. Pause Pulse expected and no HW check of drift error must always be set (=1) together with FDFL. Note: If FDFL is set, RCR.CFC is ignored and the checks described there are not executed.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	True: Enable Frequency Drift Check based on Frame Length False: Disable Frequency Drift Check based on Frame Length		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

Table 258 Specification for SentChanFreqDriftCheckLen (continued)

Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.1.4.22 SentChanCalloutFn

Table 259 Specification for SentChanCalloutFn

Name	`SentChanCalloutFn`		
Description	This parameter provides callout function to be invoked for events notification or error handling of the respective channel.		
Multiplicity	1	**Type**	EcucEnumerationParamDef
Range	Values: SENT Channel Callout notification pointer of type FUNCTION_NAME(Selectable)/ Address (Loadable) If IFX SENT is used, it should be configured as SentChanCalloutFn		
Default value	NULL_PTR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

4.3.2 Functions - Type definitions

This section describes all the type definitions used by APIs.

4.3.2.1 Sent_ChannelIdxType

Table 260 Specification for Sent_ChannelIdxType

Syntax	`Sent_ChannelIdxType`
Type	uint8
File	Sent_Types.h
Range	0 – (n-1) where n is the maximum number of SENT channels available in a particular device.
Description	Type definition to indicate the SENT channel number.
Source	IFX

4.3.2.2 Sent_NotifType

Table 261 Specification for Sent_NotifType

Syntax	`Sent_NotifType`
Type	uint32
File	Sent_Types.h
Range	Refer SENT Event Classification for more details.
Description	Type definition to indicate the interrupt events for a SENT channel.
Source	IFX

4.3.2.3 Sent_NotifFnPtrType

Table 262 Specification for Sent_NotifFnPtrType

Syntax	`Sent_NotifFnPtrType`
Type	typedef void (*Sent_NotifFnPtrType) (Sent_ChannelIdxType ChannelId, Sent_NotifType Stat)
File	Sent_Types.h
Range	User configurable function name
Description	Type definition for a callout function pointer; for a SENT channel. It provides two parameters of type Sent_ChannelIdxType – Logical channel number Sent_NotifType – interrupt status/error notification events
Source	IFX

4.3.2.4 Sent_ChannelCfgType

Table 263 Specification for Sent_ChannelCfgType

Syntax	`Sent_ChannelCfgType`	
Type	Structure	
File	Sent_Types.h	
Range	uint32 ChanRxCtrl	RCR value for the respective channel
	uint32 ChanIOCtrl	IOCR value for the respective channel
	uint32 ChanDataView	Receive Data View register (VIEWx) value for the respective channel
	Sent_NotifFnPtrType CallbackFn	Function pointer for user callback notification.
	uint16 ChanPreDiv	CPDR value for the respective channel
	uint16 ChanFracDiv	CFDR value for the respective channel
	uint8 ChanId	SENT physical channel identifier
	uint8 ChanFrameLen	Number of data nibbles per frame

Table 263	Specification for Sent_ChannelCfgType (continued)	
	uint8 Sent_InterruptNode	Interrupt node to channel ID
Description	Data structure containing the pointer to SENT channels configuration parameters required for initialization.	
Source	IFX	

4.3.2.5 Sent_CoreConfigType

Table 264	Specification for Sent_CoreConfigType	
Syntax	Sent_CoreConfigType	
Type	Structure	
File	Sent_Types.h	
Range	const Sent_ChannelCfgType *ChannelConfigPtr	Pointer to the base array of SENT channel configuration
	Sent_ChannelIdxType MaxChannels	Max number of channels
Description	Data structure containing the pointer to SENT module configuration parameters required for initialization. Pointer to object of this type is passed to API Sent_Init to initialize the SENT driver.	
Source	IFX	

4.3.2.6 Sent_RxSerialDataType

Table 265	Specification for Sent_RxSerialDataType	
Syntax	`Sent_RxSerialDataType`	
Type	Structure	
File	Sent_Types.h	
Range	uint16 Data	12/16 bit serial data.
	uint8 MsgId	4/8 bit message Id.
	uint8 CRC	6-bit CRC
	uint8 Configuration	0 – 12 bit data and 8 bit MsgId 1 – 16 bit data and 4 bit MsgId
Description	Data structure containing information of serial data (slow channel).	
Source	IFX	

4.3.2.7 Sent_ChanOpType

Table 266	Specification for Sent_ChanOpType
Syntax	`Sent_ChanOpType`

Table 266	Specification for Sent_ChanOpType (continued)
Type	Enumeration
File	Sent_Types.h
Range	SENT_ENABLE – Channel shall be enabled SENT_DISABLE – Channel shall be disabled
Description	Enumeration data structure to enable/disable a channel.
Source	IFX

4.3.2.8 Sent_ChanStateType

Table 267	Specification for Sent_ChanStateType
Syntax	`Sent_ChanStateType`
Type	Enumeration
File	Sent_Types.h
Range	SENT_STOP – Channel is disabled SENT_INITIALIZED – Channel enabled, waiting for sync/calibration pulse SENT_RUNNING – one or more sync pulse received, but frequency/Drift not in range SENT_SYNCHRONIZED – Frequency/Drift in range
Description	Enumeration data structure indicating the channel's state.
Source	IFX

4.3.2.9 Sent_ChanStatusType

Table 268	Specification for Sent_ChanStatusType	
Syntax	`Sent_ChanStatusType`	
Type	Structure	
File	Sent_Types.h	
Range	uint32 RxTimeStamp	Time the last frame for the respective channel was received. The time is captured during the falling edge of status/communication pulse.
	Sent_ChanStateType ChanStat	Status of the SENT channel
	uint32 IntStat	Snapshot of the INTSTAT register for that channel
	uint8 RxCrc	Last received frame's CRC
	uint8 StatCommNibble	Status and communication nibble value
Description	Data structure containing status information for a channel.	
Source	IFX	

4.3.2.10 Sent_SpcTrigSrcType

Table 269 Specification for Sent_SpcTrigSrcType

Syntax	Sent_SpcTrigSrcType	
Type	Enumeration	
File	Sent.h	
Range	PULSE_STOP	No pulse is generated
	PULSE_START_IMMED	Pulse generated immediately
	PULSE_START_SYNC	Pulse starts each time on the next falling edge after the sync/calibration pulse is received (Used for SPC Bi-Directional mode)
	PULSE_START_EXT_TRIGGER	Pulse starts after each external trigger event
Description	Enumeration data structure which pertains to the trigger type used for SPC transmission.	
Source	IFX	

4.3.2.11 Sent_SpcMode

Table 270 Specification for Sent_SpcMode

Syntax	`Sent_SpcMode`	
Type	Enumeration	
File	Sent.h	
Range	SYNC_MODE	This indicates SPC synchronous or range selection or id selection mode
	BIDIRECTIONAL_MODE	This indicates SPC Bi-directional mode
	RANGE_SELECTION	This indicates SPC range selection mode
	ID_SELECTION	This indicates SPC ID selection mode
Description	Enumeration data structure which pertains to the SPC transmission mode.	
Source	IFX	

4.3.2.12 Sent_SpcType

Table 271 Specification for Sent_SpcType

Syntax	Sent_SpcType	
Type	Structure	
File	Sent.h	
Range	Sent_SpcTimeBaseType TimeBase	Time base used for SPC transmission.
	Sent_SpcTrigSrcType TriggerSource	Trigger type used for SPC transmission.
	Sent_SpcMode Mode	SPC mode of operation

Table 271 Specification for Sent_SpcType (continued)

	uint8 Delay	This describes the delay time in ticks after which the SPC pulse will be sent.
	uint8 PulseLength	Length of the pulse in tick times.
Description	Data structure containing information required for a specific SPC transmission. It supports all the SPC modes.	
Source	IFX	

4.3.2.13 Sent_ChannelMapType

Table 272 Specification for Sent_ChannelMapType

Syntax	Sent_ChannelMapType	
Type	Structure	
File	Sent_Ttyps.h	
Range	uint8 Sent_ChannelCore	ID of core to which channel is mapped
	Sent_ChannelIdxType Sent_ChannelIndex	Channel index in core channel configuration
Description	Data structure containing core number and channel index of channel mapping.	
Source	IFX	

4.3.2.14 Sent_ConfigType

Table 273 Specification for Sent_ConfigType

Syntax	`Sent_ConfigType`	
Type	Structure	
File	Sent.ht	
Range	const Sent_CoreConfigType *SentCorePtr [MCAL_NO_OF_CORES]	Pointer to the base array of SENT channel core configuration
	const Sent_ChannelMapType* Sent_LogicalChanId	Pointer to Logical ID mapping
	const Sent_ChannelIdxType* Sent_IntrMapping	Pointer to Interrupt node to Channel Id
	const Sent_ChannelIdxType* Sent_PhysicalChanId	Pointer to Physical Id mapping
	uint32 ModuleClkDiv	SENT module clock divider
	uint16 ModuleFracDivStep	SENT module fractional divider clock
	uint8 NumChannelsConfigured	Number of SENT channels configured

Table 273	Specification for Sent_ConfigType (continued)
Description	Data structure containing the pointer to SENT module configuration parameters required for initialization. Pointer to object of this type is passed to API Sent_Init to initialize the SENT driver.
Source	IFX

4.3.2.15 Sent_SpcTimeBaseType

Table 274	Specification for Sent_SpcTimeBaseType	
Syntax	`Sent_SpcTimeBaseType`	
Type	Enumeration	
File	Sent.h	
Range	PULSE_NOMINAL_FREQ	Pulse based on the configured nominal frequency
	PULSE_LAST_SYNC_FREQ	Pulse based on last measured sync/calibration pulse frequency
Description	Enumeration data structure which pertains to the time base used for SPC transmission.	
Source	IFX	

4.3.2.16 Sent_GlitchStatusType

Table 275	Specification for Sent_GlitchStatusType	
Syntax	`Sent_GlitchStatusType`	
Type	Structure	
File	Sent_Types.h	
Range	uint8 RisingEdge	Snapshot of the Rising Edge Glitch Flag Status
	uint8 FallingEdge	Snapshot of the Falling Edge Glitch Flag Status
Description	Data structure containing glitch filter status information for a channel.	
Source	IFX	

4.3.3 Functions - APIs

This section lists the APIs provided along with a short description of the functionality.

4.3.3.1 Sent_Init

Table 276	Specification for Sent_Init API
Syntax	`void Sent_Init (const Sent_CfgType *ConfigPtr)`
Service ID	0x00
Sync/Async	Synchronous

Table 276 Specification for Sent_Init API (continued)

Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to SENT configuration structure
Parameters (out)	None	
Parameters (in-out)	None	
Return	void	
Description	Initializes the SENT module and the respective channels based on the configuration values passed in the pointer ConfigPtr.	
Source	IFX	
Error handling	DET: SENT_E_INIT_FAILED: This error is reported if init is called with an invalid configuration SENT_MASTER_CORE_UNINIT: This error is reported if the slave core initialization is requested when the master core is not initialized SENT_E_ALREADY_INITIALIZED: This error is reported if the initialization is requested for a core which is already initialized SENT_E_CORE_NOT_CONFIGURED: This error is reported if the requested core is not configured	
Configuration dependencies	-	

4.3.3.2 Sent_SetChannel

Table 277 Specification for Sent_SetChannel API

Syntax	`void Sent_SetChannel (const Sent_ChannelIdxType ChanId, Sent_ChanOpType Operation)`	
Service ID	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ChanId	SENT logical channel number
	Operation	Operation type SENT_ENABLE – Enable channel SENT_DISABLE – Disable channel
Parameters (out)	None	
Parameters (in-out)	None	
Return	void	
Description	Enable/Disable the SENT channel.	
Source	IFX	
Error handling	DET:	

Table 277 Specification for Sent_SetChannel API (continued)

	SENT_E_UNINIT: This error is reported if but the core is not initialized SENT_E_INVALID_CHANNEL: This error is reported if the channel id is invalid SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel Id is configured in different core SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if channel is not available in the hardware
Configuration dependencies	-

4.3.3.3 Sent_ReadData

Table 278 Specification for Sent_ReadData API

Syntax	`uint32 Sent_ReadData (const Sent_ChannelIdxType ChannelId)`
Service ID	0x02
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	ChannelId
Parameters (out)	None
Parameters (in-out)	None
Return	uint32
Description	Reads the current SENT frame received.
Source	IFX
Error handling	DET: SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core
Configuration dependencies	-

4.3.3.4 Sent_ReadSerialData

Table 279 Specification for Sent_ReadSerialData API

Syntax	`void Sent_ReadSerialData (const Sent_ChannelIdxType ChannelId, Sent_RxSerialDataType * DataPtr)`	
Service ID	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ChannelId	SENT channel number
Parameters (out)	DataPtr	Data pointer pointing to the serial data read from the SENT Channel
Parameters (in-out)	None	
Return	void	
Description	Reads the SENT slow channel frame received.	
Source	IFX	
Error handling	DET: SENT_E_PARAM_POINTER: This error is reported if the API is called with a NULL pointer as its parameter. SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core	
Configuration dependencies	-	

4.3.3.5 Sent_ReadChannelStatus

Table 280 Specification for Sent_ReadChannelStatus API

Syntax	`void Sent_ReadChannelStatus (const Sent_ChannelIdxType ChannelId, Sent_ChanStatusType * StatPtr)`	
Service ID	0x04	
Sync/Async	Synchronous	
Reentrancy	Not Reentrant	
Parameters (in)	ChannelId	SENT logical channel number

Table 280 Specification for Sent_ReadChannelStatus API (continued)

Parameters (out)	StatPtr	Pointer pointing to the status of the SENT Channel
Parameters (in-out)	None	
Return	void	
Description	Reads the SENT channel's current status.	
Source	IFX	
Error handling	DET: SENT_E_PARAM_POINTER: This error is reported if the API is called with a NULL pointer as its parameter. SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core	
Configuration dependencies	-	

4.3.3.6 Sent_SpcGenPulse

Table 281 Specification for Sent_SpcGenPulse API

Syntax	`void Sent_SpcGenPulse (const Sent_ChannelIdxType ChanId, const Sent_SpcType *SpcCfgPtr)`	
Service ID	0x05	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ChanId	SENT Channel's status has to be read
	SpcCfgPtr	Pointer to SPC configuration structure
Parameters (out)	None	
Return value	void	
Description	This function generates a Master pulse for SPC Sync transmission and it is also used for the bi-directional mode.	
Source	IFX	
Error handling	DET: SENT_E_PARAM_POINTER: This error is reported if the API is called with a NULL pointer as its parameter.	

Table 281 Specification for Sent_SpcGenPulse API (continued)

	SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core
Configuration dependencies	-

4.3.3.7 Sent_SetWdgTimer

Table 282 Specification for Sent_SetWdgTimer API

Syntax	`void Sent_SetWdgTimer (const Sent_ChannelIdxType ChanId, const uint16 WdgTimerReloadVal)`	
Service ID	0x06	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ChanId	SENT Channel's status has to be read
	WdgTimerReloadVal	Timer reload value
Parameters (out)	None	
Parameters (in-out)	None	
Return	void	
Description	This API allows enabling of internal watchdog timer for SENT channel ChanId with timer value WdgTimerReloadVal. To stop the watchdog timer, WdgTimerReloadVal should be set to 0.	
Source	IFX	
Error handling	DET: SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core	
Configuration dependencies	-	

4.3.3.8 Sent_GetVersionInfo

Table 283 Specification for Sent_GetVersionInfo API

Syntax	`void Sent_GetVersionInfo (Std_VersionInfoType * VersionInfoPtr)`	
Service ID	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Noe	-
Parameters (out)	versioninfo	Pointer to store the version information of this module.
Parameters (in-out)	None	
Return	void	
Description	This API retrieves the vendor-id, module-id along with major and minor version of the SENT driver.	
Source	IFX	
Error handling	DET: SENT_E_PARAM_POINTER: API service called with a NULL pointer.	
Configuration dependencies	-	

4.3.3.9 Sent_DeInit

Table 284 Specification for Sent_DeInit API

Syntax	`void Sent_DeInit (void)`
Service ID	0x0A
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (out)	None
Parameters (in-out)	None
Return	void
Description	This API provides service to de-initialize the SENT hardware and its channel's registers.
Source	IFX
Error handling	DET: SENT_E_UNINIT: This error is reported if core is uninitialized. SENT_E_SLAVE_CORE_INIT: This error is reported if the master core de-init is requested when all slave cores are not de-initialized

Table 284	Specification for Sent_DeInit API (continued)
Configuration dependencies	-

4.3.3.10 Sent_ReadGlitchFilterStatus

Table 285	Specification for Sent_ReadGlitchFilterStatus API	
Syntax	`Std_ReturnType Sent_ReadGlitchFilterStatus (const Sent_ChannelIdxType ChannelId)`	
Service ID	-	
Sync/Async	Synchronous	
Reentrancy	Non- Reentrant	
Parameters (in)	ChannelId	SENT logical channel number
Parameters (out)	None	
Parameters (in-out)	None	
Return	Std_ReturnType	E_OK: Power Mode changed E_NOT_OK: Service is rejected
Description	This function reads the status of the glitch filter	
Source	IFX	
Error handling	DET: SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core	
Configuration dependencies	-	

4.3.3.11 Sent_ResetGlitchFilterStatus

Table 286	Specification for Sent_ ResetGlitchFilterStatus API
Syntax	`Std_ReturnType Sent_ ResetGlitchFilterStatus (const Sent_ChannelIdxType ChannelId)`
Service ID	-
Sync/Async	Synchronous

Table 286 Specification for Sent_ ResetGlitchFilterStatus API (continued)

Reentrancy	Non- Reentrant	
Parameters (in)	ChannelId	SENT logical channel number
Parameters (out)	None	
Parameters (in-out)	None	
Return	Std_ReturnType	E_OK: Power Mode changed E_NOT_OK: Service is rejected
Description	This function reads the status of the glitch filter.	
Source	IFX	
Error handling	DET: SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core	
Configuration dependencies	-	

4.3.3.12 Sent_FDFLParameters

Table 287 Specification for Sent_FDFLParameters API

Syntax	`Std_ReturnType Sent_FDFLParameters (const Sent_ChannelIdxType ChannelId)`	
Service ID	-	
Sync/Async	Synchronous	
Reentrancy	Non- Reentrant	
Parameters (in)	ChannelId	SENT logical channel number
Parameters (out)	FDFLParam	
Parameters (in-out)	None	
Return	Std_ReturnType	E_OK: Power Mode changed E_NOT_OK: Service is rejected
Description	This function checks for the frequency drift in the received channel.	
Source	IFX	
Error handling	DET: SENT_E_UNINIT: This error is reported if core is uninitialized SENT_E_INVALID_CHANNEL: This error reported if the channel Id is invalid	

Table 287 Specification for Sent_FDFLParameters API (continued)

	SENT_E_CHANNEL_NOT_CONFIGURED: This error is reported if the channel is not configured SENT_E_CHANNEL_NOT_ENABLED: This error is reported if the channel is not enabled SENT_E_CORE_CHANNEL_MISMATCH: This error is reported if the channel is configured in different core
Configuration dependencies	-

4.3.4 Notifications and callbacks

This section lists all the notifications and callbacks of the driver.

A callout function is linked uniquely with a SENT channel to be notified with the channel's interrupt events or any error/status events. The callout function prototype is defined by Sent_NotifFnPtrType. The callout functions fall under the MCAL layer and are allowed to access the SENT registers if required. The application can determine the necessary action based on the event notifications. It is responsibility of the user to define the SENT callout functions.

4.3.4.1 SENT event classification

The following table provides the events that will be raised by the SENT driver through the callout function per channel.

Event	Event description	Value (Hex)
SENT_INT_RSI_EVENT	Successful reception of SENT frame after verification of CRC	0x1
SENT_INT_RDI_EVENT	Successful reception of SENT frame and data has been moved to the RDR register but CRC may not been verified by HW (depends on SentChanFrameCrcDisable configuration parameter)	0x2
SENT_INT_RBI_EVENT	Receive buffer overflow occurred	0x4
SENT_INT_TDI_EVENT	Successful transmission of SPC master pulse	0x8
SENT_INT_TBI_EVENT	Transmit buffer underflow occurred	0x10
SENT_INT_FRI_EVENT	Synchronization/Calibration pulse deviation occurred from nominal value (more than +/- 25%)	0x20
SENT_INT_FDI_EVENT	Subsequent Synchronization/Calibration pulse deviation from its predecessor (more than 1.5625%)	0x40
SENT_INT_NNI_EVENT	More nibbles received than expected or next Synchronization/Calibration pulse indicating less nibbles received	0x80
SENT_INT_NVI_EVENT	Too long or too short nibble pulse received	0x100

Event	Event description	Value (Hex)
SENT_INT_CRCI_EVENT	CRC verification failed for the last received SENT frame	0x200
SENT_INT_WSI_EVENT	This occurs only in standard serial mode; where Status/Communication nibble shows a start bit in a frame other than first SENT frame	0x400
SENT_INT_SDI_EVENT	Successful reception of all serial data bits	0x800
SENT_INT_SCRI_EVENT	CRC verification failed for the serial data received	0x1000
SENT_INT_WDI_EVENT	Watchdog timer for the channel expired; since it didn't receive the SENT frame within the desired time	0x2000
SENT_TRANS_INPROGRESS_EVENT	Timeout error indicating a transfer is still ongoing	0x4000

4.3.5 Scheduled functions

The driver does not support any scheduled functions.

4.3.6 Interrupt service routines

This section lists all the interrupt handlers of the driver.

4.3.6.1 Sent_Isr

Table 288 Specification for Sent_Isr API

Syntax	void Sent_Isr (uint8 IntrNode)	
Service ID	-	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	IntrNode	Interrupt node for the channel
Parameters (out)	-	
Parameters (in-out)	-	
Return	void	
Description	This function is the interrupt handler and collects the status of the relevant channels and inform the user.	
Source	IFX	
Error handling	None	

Table 288 Specification for Sent_Isr API (continued)

Configuration dependencies	-

4.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

4.3.7.1 Development errors

The following table lists all the development errors reported by the driver.

Table 289 Description of development errors reported

Description	Source	Error code and value	Applicable APIs
Synchronous transmission service called at invalid channel.	IFX	SENT_E_INVALID_CHANNEL= 0x02	Sent_SetChannel Sent_ReadData Sent_ReadSerialData Sent_ReadChannelStatus Sent_SpcGenPulse Sent_SetWdgTimer Sent_ReadGlitchFilterStatus Sent_ResetGlitchFilterStatus Sent_FDFLParameters
API service is called with a NULL pointer as its parameter.	IFX	. SENT_E_PARAM_POINTER = 0x03	Sent_GetVersionInfo Sent_ReadSerialData Sent_ReadChannelStatus Sent_SpcGenPulse
Service is called before Init.	IFX	SENT_E_UNINIT = 0x05	Sent_DeInit Sent_SetChannel Sent_ReadData Sent_ReadSerialData Sent_ReadChannelStatus Sent_SpcGenPulse Sent_SetWdgTimer Sent_ReadGlitchFilterStatus Sent_ResetGlitchFilterStatus Sent_FDFLParameters
Service is called when initialization is failed.	IFX	SENT_E_INIT_FAILED = 0x10	Sent_Init
Service is called when Sent driver is already initialized.	IFX	SENT_E_ALREADY_INITIALIZED = 0x14	Sent_Init

Table 289 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
Report DET SENT channel is not configured for this Core.	IFX	SENT_E_CORE_NOT_CONFIGURED = 0x64	Sent_Init
SENT channel is not allocated to this core.	IFX	SENT_E_CORE_CHANNEL_MISMATCH = 0x65	Sent_SetChannel Sent_ReadData Sent_ReadSerialData Sent_ReadChannelStatus Sent_SpcGenPulse Sent_SetWdgTimer Sent_ReadGlitchFilterStatus Sent_ResetGlitchFilterStatus Sent_FDFLParameters
Core Initialization called when master initialization is not done.	IFX	SENT_MASTER_CORE_UNINIT = 0x66	Sent_Init
Master core de-initialization called before de-initialization of slave cores.	IFX	SENT_E_SLAVE_CORE_INIT=0x67	Sent_DeInit
Sent channel is not configured.	IFX	SENT_E_CHANNEL_NOT_CONFIGURED=0x68	Sent_SetChannel Sent_ReadData Sent_ReadSerialData Sent_ReadChannelStatus Sent_SpcGenPulse Sent_SetWdgTimer Sent_ReadGlitchFilterStatus Sent_ResetGlitchFilterStatus Sent_FDFLParameters
Sent channel is not enabled.	IFX	SENT_E_CHANNEL_NOT_ENABLED=0x69	Sent_ReadData Sent_ReadSerialData Sent_ReadChannelStatus Sent_SpcGenPulse Sent_SetWdgTimer Sent_ReadGlitchFilterStatus Sent_ResetGlitchFilterStatus Sent_FDFLParameters

4.3.7.2 Production errors

The driver does not report any production errors.

4.3.7.3 Safety errors

The driver does not report any safety errors.

4.3.7.4 Runtime errors

The driver does not report any runtime errors.

4.3.8 Deviations and limitations

4.3.8.1 Deviations

No deviations are reported from the requirement.

4.3.8.2 Limitations

Table 290 Known limitations

Reference	Limitation
Interrupt handling	The SENT channels report 14 interrupt per channel. But number of interrupt nodes available are 10. Hence interrupts of each channel are limited to a single interrupt node only.

4.3.8.3 Unsupported hardware features

None.

5 STM driver

5.1 User information

5.1.1 Description

The STM provides free running 64 bit timer for real-time operating systems and other system purposes (high precision and long range). The STM complex driver is responsible for enabling STM interrupts and providing alarms at regular intervals.

5.1.2 Hardware-software mapping

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

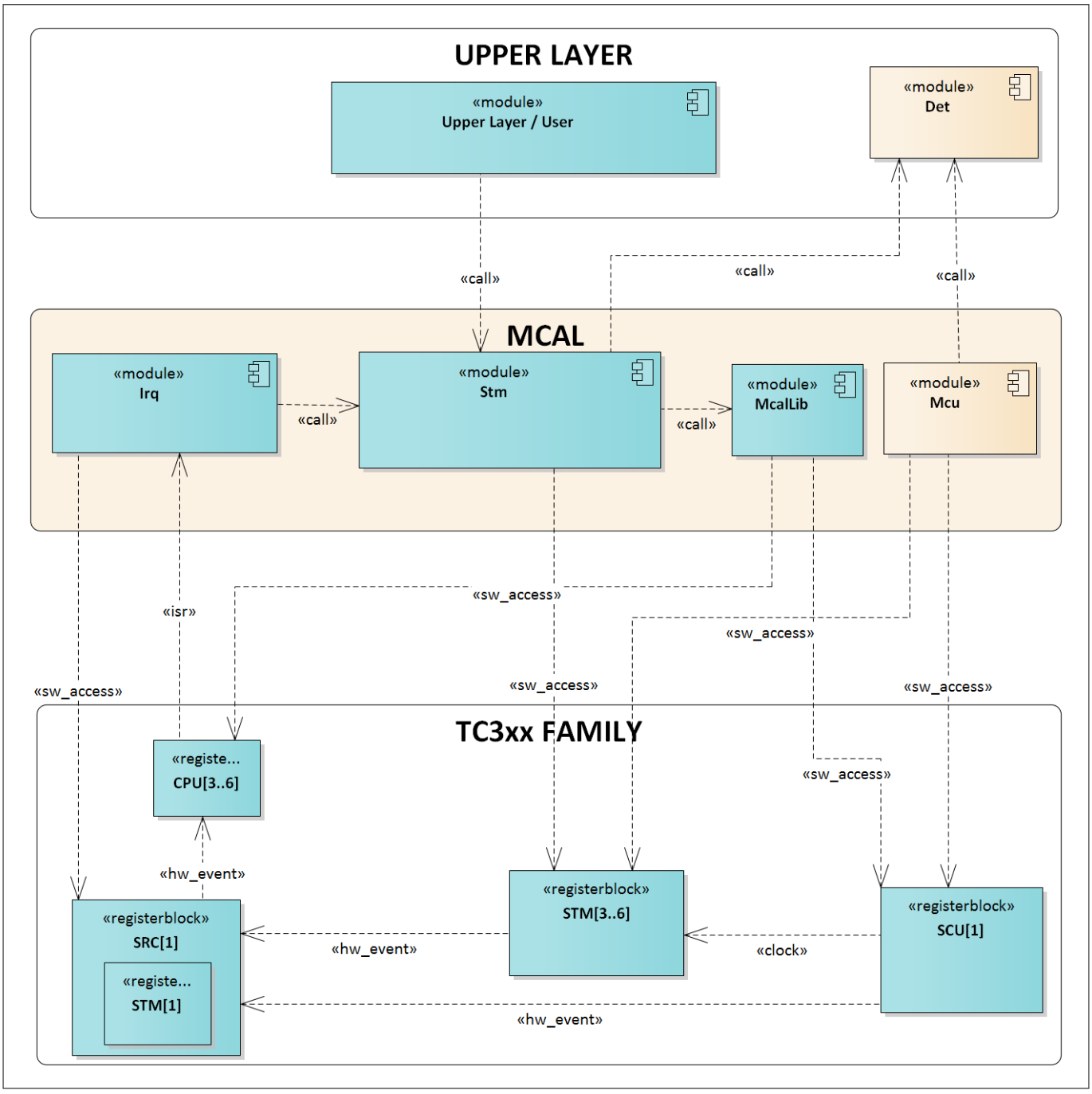


Figure 42 Mapping of hardware-software interfaces

5.1.2.1 STM: primary hardware peripheral

Hardware functional features

- The STM provides free running 64 bit timer for real-time operating systems and other system purposes (high precision and long range)
- STM complex driver is responsible for enabling STM interrupts and providing alarms at regular intervals
- Counting starts automatically after an application reset

Users of the hardware

STM(0..x)(x:number of cores available in the device) is exclusively by STM driver. User can directly access STM registers to get the call back notification.

Hardware diagnostic features

Not applicable.

Hardware events

The content of the 64-bit STM can be compared against the content of compare values stored in the CMP0 or CMP1 register. Service requests can be generated on a compare match of the STM with CMP0 or CMP1 registers.

5.1.2.2 SCU: dependent hardware peripheral

Hardware functional features

The STM driver depends upon the SCU for clock and reset functionality.

Users of the hardware

Several peripherals have their clocks supplied by SCU. However it is only the MCU driver that is responsible for configuration of the clock tree.

Hardware diagnostic features

Not applicable.

Hardware events

Not applicable.

5.1.2.3 SRC-IR: dependent hardware peripheral

Hardware functional features

CMP0 and CMP1 register has its compare register flag that is set by hardware on a compare match event.

Users of the hardware

Several peripherals have their interrupt nodes mapped to Interrupt router. STM driver uses two interrupt nodes to trigger STM timer compare match event.

Hardware diagnostic features

Safety mechanism implemented in the Interrupt router module to trigger SMU alarm incase of hardware failure.

Hardware events

Interrupt event generation based on the configured priority and core.

5.1.3 File structure

5.1.3.1 C File Structure

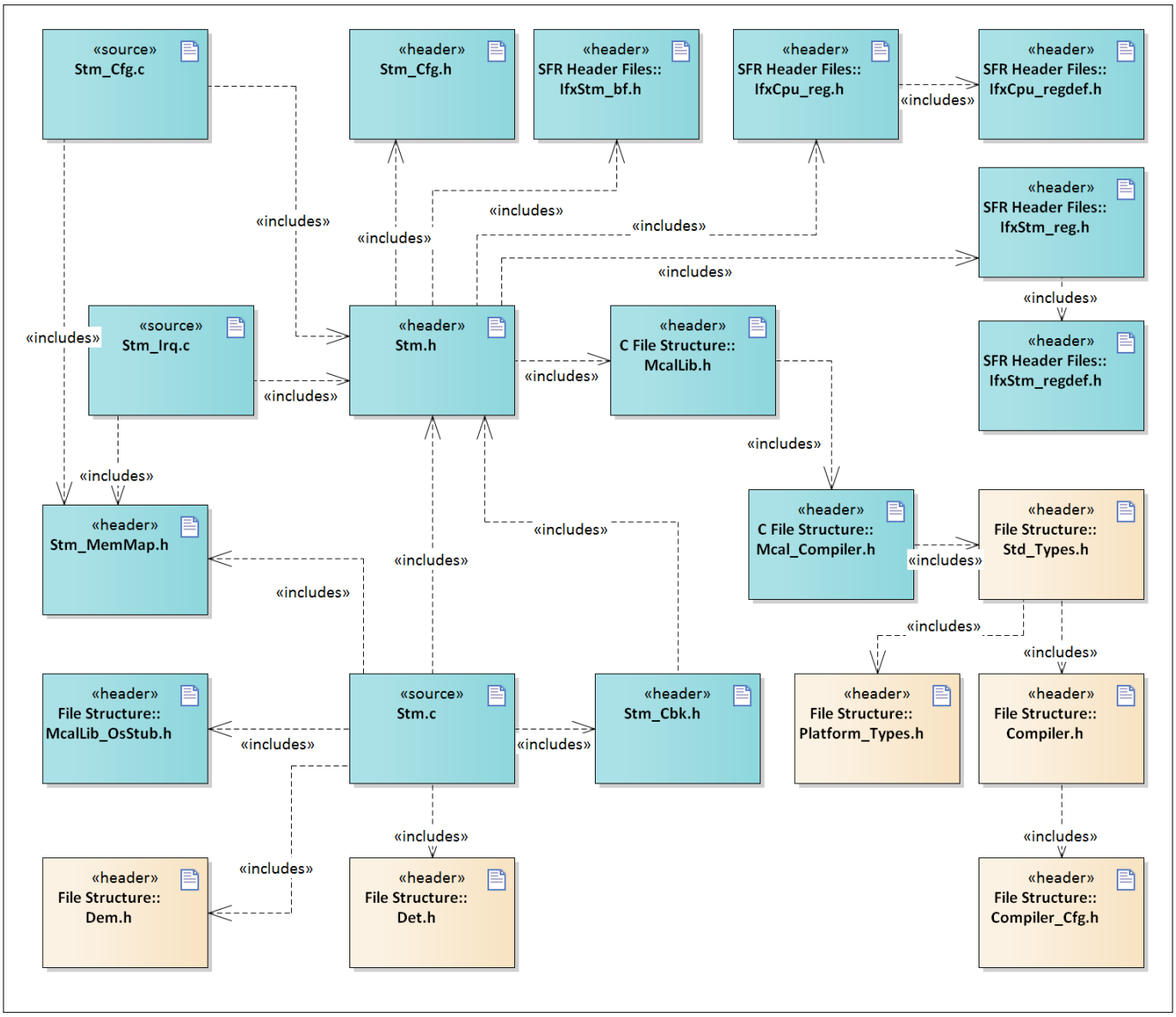


Figure 43 C File Structure

Table 291 C File Structure

File name	Description
`Compiler.h`	Provides abstraction from compiler-specific keywords
`Compiler_Cfg.h`	Configuration header file for compiler abstraction
`Dem.h`	Provides the exported interfaces of Diagnostic Event Manager
`Det.h`	Provides the exported interfaces of Development Error Tracer
`IfxCpu_reg.h`	SFR header file for CPU
`IfxCpu_regdef.h`	SFR header file for CPU
`IfxStm_bf.h`	SFR header file for STM
`IfxStm_reg.h`	SFR header file for STM
`IfxStm_regdef.h`	SFR header file for STM

Table 291 C File Structure (continued)

File name	Description
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	Provides abstraction for TriCore-intrinsic instruction
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.
Stm.c	This file contains functionality of STM driver.
Stm.h	This file contains function declaration and interrupt routine declaration of STM driver.
Stm_Cbk.h	Stm Driver Callback header definition file
Stm_Cfg.c	STM Module configuration file to store STM timer usage information across the cores.
Stm_Cfg.h	STM Module Configuration header file. it contains macro definition for all the pre compile configuration parameters.
Stm_Irq.c	This file contains the interrupt frames for the STM Module.
Stm_MemMap.h	File (Static) containing the memory section definitions used by the STM driver.

5.1.3.2 Code Generator Plugin Files

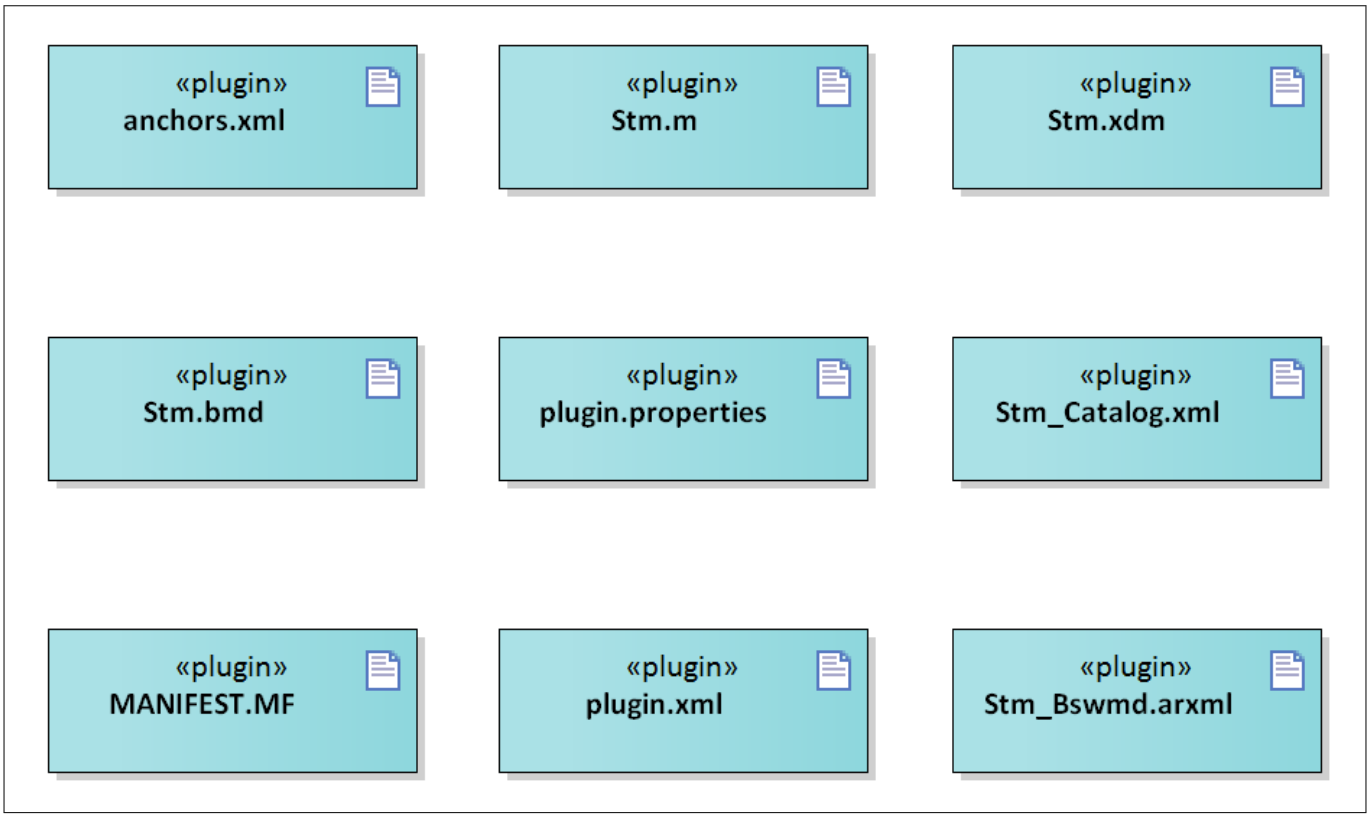


Figure 44 Code Generator Plugin Files

Table 292 Code Generator Plugin Files

File name	Description
`MANIFEST.MF`	Tresos plugin support file containing the meta-data for Stm driver.
`Stm.bmd`	AUTOSAR format XML data model schema file (for each device).
`Stm.m`	Code template macro file for Stm driver
`Stm.xdm`	Tresos format XML data model schema file.
`Stm_Bswmd.arxml`	AUTOSAR format module description file.
`Stm_Catalog.xml`	AUTOSAR format catalog file.
`anchors.xml`	Tresos anchors support file for the Stm driver
`plugin.properties`	Tresos plugin support file for the Stm driver.
`plugin.xml`	Tresos plugin support file for the Stm driver.

5.1.4 Integration hints

5.1.4.1 Integration with AUTOSAR stack

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software

drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **Memory mapping**

  Memory mapping is a concept from AUTOSAR that allows re-location of text, variables, constants and configuration data to user specific memory regions. In order to achieve this, all the re-locatable elements of the driver are encapsulated in different memory-section macros. These macros are defined in the `Stm_MemMap.h`.

  The file `Stm_MemMap.h` 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 re-located to the correct memory region. A sample implementation listing the memory-section macros is depicted below.

```
/**** CORE[x] GLOBAL RAM DATA -- NON-CACHED LMU ****/
#if defined STM_START_SEC_VAR_CLEARED_QM_CORE[x]_UNSPECIFIED
/***** User pragmas here for Non-cached LMU*****/
#undef STM_START_SEC_VAR_CLEARED_QM_CORE[x]_UNSPECIFIED
#undef MEMMAP_ERROR
#elif defined STM_STOP_SEC_VAR_CLEARED_QM_CORE[x]_UNSPECIFIED
/***** User pragmas here for Non-cached LMU*****/
#undef STM_STOP_SEC_VAR_CLEARED_QM_CORE[x]_UNSPECIFIED
#undef MEMMAP_ERROR
/**** CODE -- PF[x] ****/
#elif defined STM_START_SEC_CODE_QM_GLOBAL
/*****User pragmas here for PF[x]*****/
#undef STM_START_SEC_CODE_QM_GLOBAL
#undef MEMMAP_ERROR
#elif defined STM_STOP_SEC_CODE_QM_GLOBAL
/*****User pragmas here for PF[x]*****/
#undef STM_STOP_SEC_CODE_QM_GLOBAL
#undef MEMMAP_ERROR

/**** CORE CONFIG DATA -- PF****/
#elif defined STM_START_SEC_CONFIG_DATA_QM_GLOBAL_UNSPECIFIED
/*****User pragmas here for PF[x]*****/
#undef STM_START_SEC_CONFIG_DATA_QM_GLOBAL_UNSPECIFIED
#undef MEMMAP_ERROR
#elif defined STM_STOP_SEC_CONFIG_DATA_QM_GLOBAL_UNSPECIFIED
/*****User pragmas here for PF[x]*****/
#undef STM_STOP_SEC_CONFIG_DATA_QM_GLOBAL_UNSPECIFIED
#undef MEMMAP_ERROR

#endif

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

- **DET**

  The DET module is a part of the AUTOSAR stack that handles all the development and runtime errors reported by the Basic Software modules. The <Mod> driver reports all the development errors to the DET

module through the API `Det_ReportError()`. The user of the <Mod> driver must process all the errors reported to the DET module through the API `Det_ReportError()`.

The files `Det.h` and `Det.c` are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.

- **DEM**

  The DEM module is a part of the AUTOSAR stack that handles all the production errors reported by the Basic Software modules. The <Mod> driver reports all the production errors to the DEM modules through the API `Dem_ReportErrorStatus()`. The user of the <Mod> driver must process all the production errors (fail / pass) reported to the DEM module through the API `Dem_ReportErrorStatus()`.

  The files `Dem.h` and `Dem.c` are provided in the MCAL package as a stub code and needs to be replaced with a complete DEM module during the integration phase.

- **SchM**

  STM module does not have any critical section

- **Safety error**

  STM module does not report any safety error.

- **Notifications and callbacks**

  The STM driver gives callback to user when the compare match is detected based on the ticks configured by user.

- **Operating system (OS)**

  OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application. Operating system files provided by MCAL package is only an example code and must be updated by the integrator with actual OS files for desired function.

5.1.4.2 Multicore and Resource Manager

The STM driver supports execution of its APIs in parallel from all the CPU cores. The user has to allocate STM timers to CPU cores at pre-compile time using the Resource Manager and MCU module. The following are the key points to be considered with respect to multicore in the driver:

- For each core one STM timer is allowed to configure in the resource manager module.

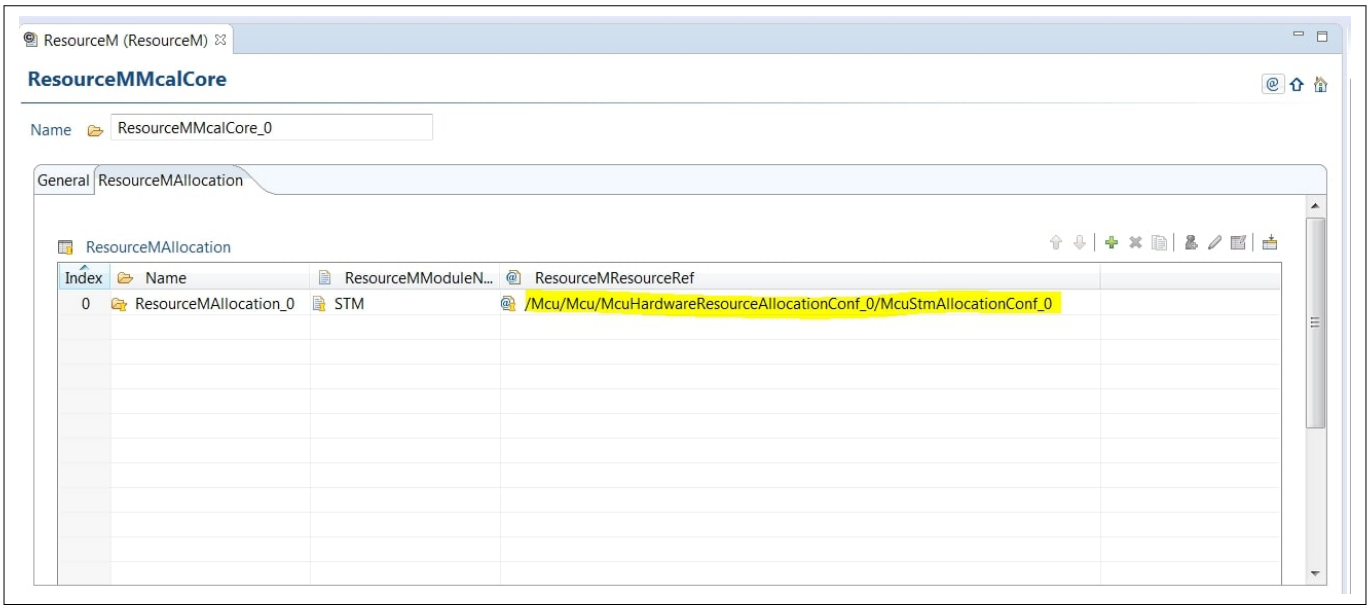


Figure 45 STM timer allocation in Resource Manager

- The STM timer can be allocated to any of the core available on the silicon. The user must ensure that only the allocated STM peripheral resource is accessed in the core.
- It must be ensured that the STM timer number passed as a parameter while invoking a STM API, belong to the same core on which the API is invoked.
- DETs will be raised in case APIs are invoked with mismatch of core and STM timer.
- The locating of constants, variables and configuration data to correct memory space should be done by the user. Memory sections are marked GLOBAL (common to all cores) and CORE[x] (specific to a CPU core). The following should be considered by the user to ensure better performance of the driver:

  Code Section:

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

  Data Section:

  The RAM variable memory sections marked as specific to core, should be re-located to the DSPR of the same core. Those marked as global should be relocated to the non-cached LMU region.

  Configuration data and constants:

  The configuration data section sections marked as specific to core, should be re-located to the PFLASH of the same core. Those marked as global should be relocated to the PFlash of the master core.

  Note1: Relocating of code, data, constants to a distant memory space would impact execution timings.

  Note2: If the driver operates from a single (master) core, all the sections may be relocated to the PFlash or DSPR of the same CPU core.

5.1.4.3 MCU support

The STM driver is dependent on MCU driver for clock configuration. The initialization of STM driver must be started only after completion of MCU initialization. The following must be considered while configuring the MCU driver in tresos:

- The CMP register used by STM driver must be reserved in the MCU configuration for exclusive use by STM.
- The CMP register is shared between STM and WDG driver, user must ensure CMP is allocated to STM before using it in STM API's.

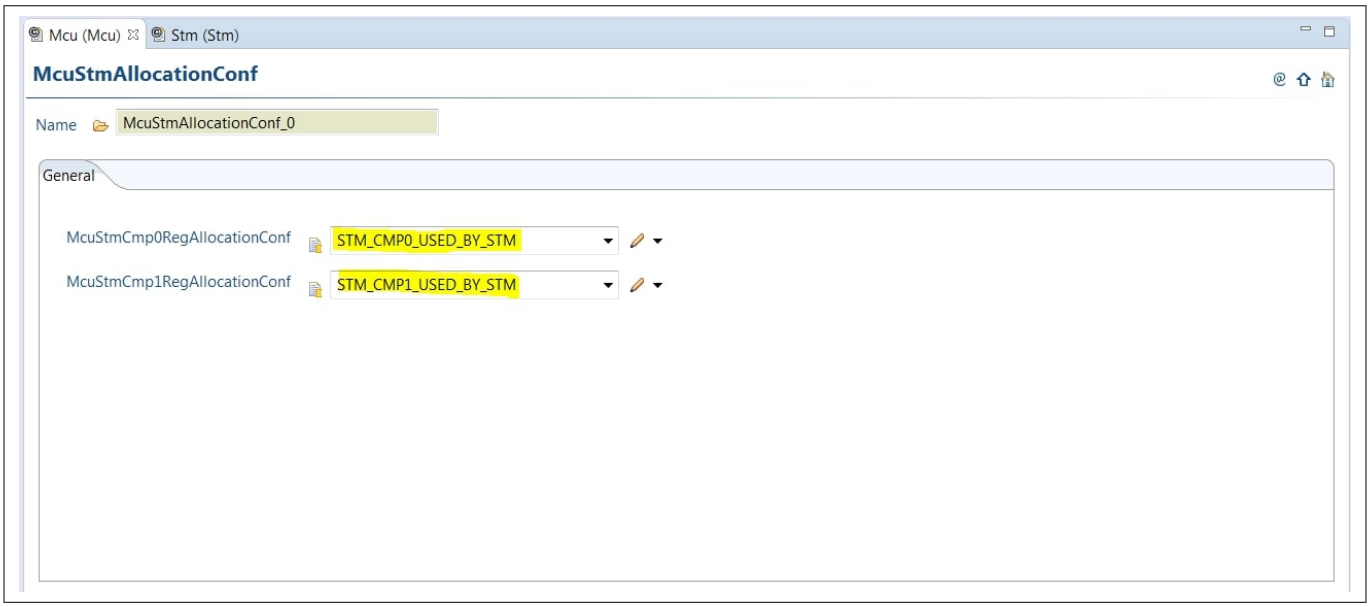


Figure 46 STM compare register allocation for STM in MCU

5.1.4.4 Port support

PORT support is not needed for STM.

5.1.4.5 DMA support

DMA support is not needed for STM.

5.1.4.6 Interrupt connections

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

Compare Match Interrupts

The compare match interrupt is generated based on the ticks configured by the user. The interrupts for CMP registers are enabled based on the allocation of CMP registers for STM. The interrupts from CMP0 is routed to SR0 and CMP1 is routed to SR1.

```
/* include MCU timer header file */
#include "Mcu_17_TimerIp.h"
/*********SRC_ STM0SR0********/
ISR(STM0SR0_ISR)
{
/* Enable Global Interrupts */
ENABLE();
/* Call Interrupt function */
Mcu_17_Stm_CompareMatchIsr(STM_NUMBER, STM_INTERRUPT_NODE);
/* STM_NUMBER = 0, STM_INTERRUPT_NODE = 0 */
}
/********* SRC_ STM0SR1********/
ISR(STM0SR1_ISR)
{
/* Enable Global Interrupts */
ENABLE();
/* Call Interrupt function */
 Mcu_17_Stm_CompareMatchIsr(STM_NUMBER, STM_INTERRUPT_NODE);
}
```

5.1.4.7 Example usage

Enable STM interrupts

The code Listing depicts the steps involved to enable the STM driver interrupts.

```
Mcu_Init(&Mcu_Config);
Mcu_InitClock(0);
while(Mcu_GetPllStatus() == MCU_PLL_LOCKED);
Mcu_DistributePllClock();
/* Configure Interrupt priority */
IrqStm_Init();
/*Enable Stm interrupt*/
Stm_EnableModule(StmModuleNumber);
```

After the call of `Stm_EnableModule()` it enables the STM interrupts and interrupts node mapping.

Enable Alarm

The code Listing depicts the steps involved to invoke a Call-back function after the elapse of configured amount of time.

```
Stm_EnableModule(StmModuleNumber);
/*Enable interrupt and call-back */
Stm_EnableAlarm(StmModuleNumber,CompareRegisterId,TimerMode,Ticks,Stm_Applicati
onfunction);
```

Read timer value

The following code snippet shows call to `Stm_ReadTotalTimerValue()` and `Stm_ReadTimerValue()` API.

```
/* Read total timer value */
Timer = Stm_ReadTotalTimerValue(StmModuleNumber);
/*Read timer value of STM_TIM */
TimerValue= Stm_ReadTimerValue(StmModuleNumber,TimerNumber);
```

Compare Match Control

The following code snippet shows call to `Stm_SetCompareMatchControl()`

```
Stm_EnableModule(StmModuleNumber);
/* Set Compare Match Control Register*/
Stm_SetCompareMatchControl(StmModuleNumber,CompareRegisterId,Mstart,MSize);
/*Enable interrupt and call-back */
Stm_EnableAlarm(StmModuleNumber,CompareRegisterId,TimerMode,Ticks,Stm_Applicati
onfunction);
```

Note:

If Stm_EnableAlarm() is called without calling Stm_SetCompareMatchControl(), STM timer bit range [0-31] is used for comparison.

In Stm_SetCompareMatchControl(), Mstart and Msize should be used appropriately for other STM timer bit range.

Eg. For Stm timer bit range[4-35], MStart should be set to 0x4.

For CMP register bit range[0-30], Msize should be set to 0x1E.

Disable Alarm

After the call of `Stm_DisableAlarm()` it disables the STM interrupts and stops invoking a call-back function.

```
Stm_EnableAlarm(StmModuleNumber,CompareRegisterId,TimerMode,Ticks,Stm_Applicati
onfunction);
/*Stop invoking call-back function */
Stm_DisableAlarm(StmModuleNumber,CompareRegisterId);
```

5.1.5 Key architectural considerations

There are no key architectural considerations for the driver.

5.2 Assumptions of Use (AoUs)

There are no AoUs for the driver.

5.3 Reference information

5.3.1 Configuration interfaces

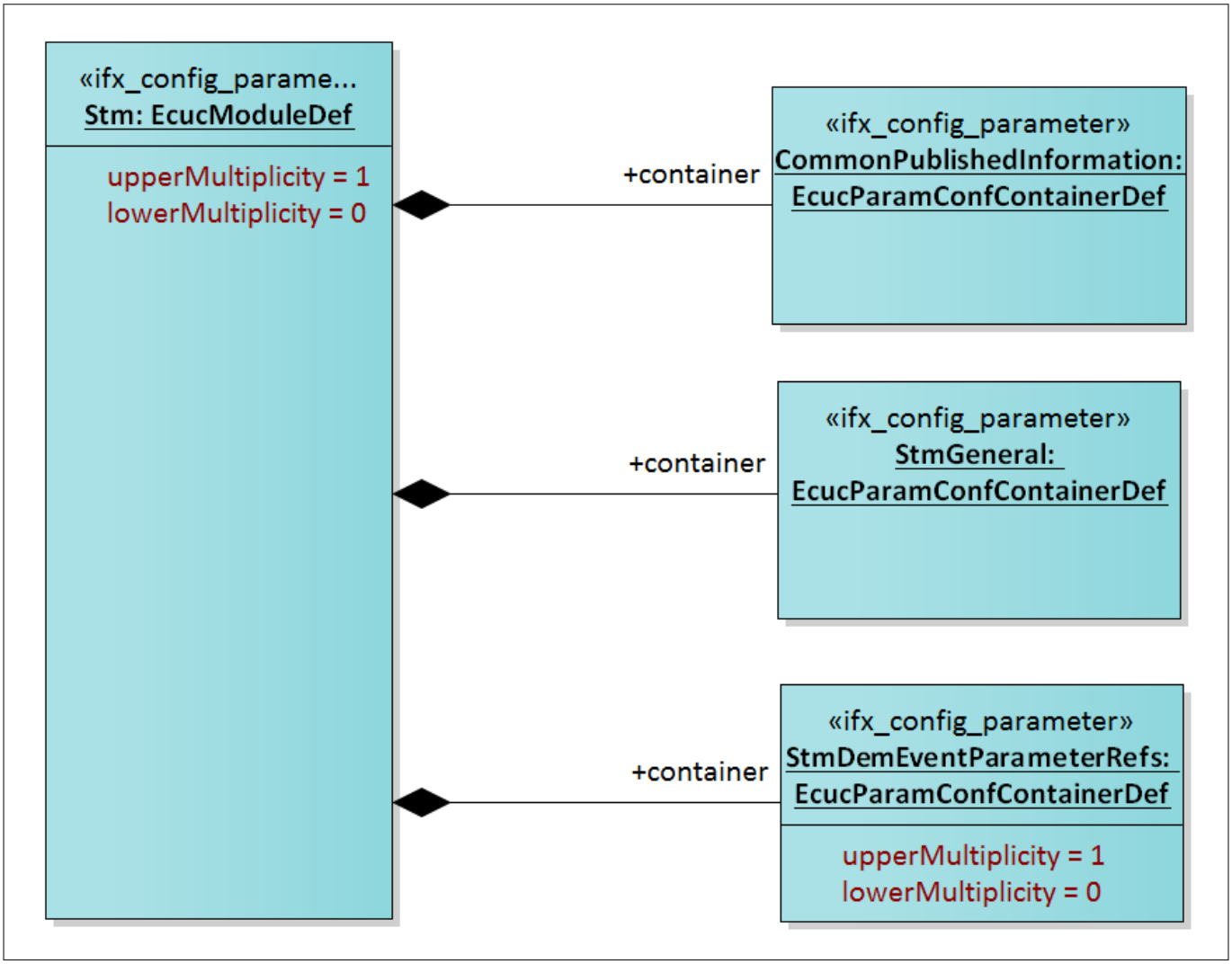


Figure 47 Container hierarchy along with their configuration parameters

5.3.1.1 Container: StmDemEventParameterRefs

Post-Build Variant Multiplicity: FALSE
Multiplicity Configuration Class: Pre-Compile

5.3.1.1.1 STM_E_CLC_ENABLE_ERR

Table 293 Specification for STM_E_CLC_ENABLE_ERR

Name	STM_E_CLC_ENABLE_ERR		
Description	DEM enable/disable for STM module enable failure.		
Multiplicity	0..1	**Type**	EcucSymbolicNameReferenceDef

Table 293 Specification for STM_E_CLC_ENABLE_ERR (continued)

Range	Reference to Node:		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2 Container: CommonPublishedInformation

Common container, aggregated by all modules. It contains published information about vendor and versions.
Post-Build Variant Multiplicity: FALSE
Multiplicity Configuration Class: -

5.3.1.2.1 ArMajorVersion

Table 294 Specification for ArMajorVersion

Name	`ArMajorVersion`		
Description	STM Autosar major version		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.2 ArMinorVersion

Table 295 Specification for ArMinorVersion

Name	`ArMinorVersion`		
Description	STM Autosar minor version		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		

Table 295 Specification for ArMinorVersion (continued)

Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.3 ArPatchVersion

Table 296 Specification for ArPatchVersion

Name	`ArPatchVersion`		
Description	STM Autosar patch version		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.4 ModuleId

Table 297 Specification for ModuleId

Name	`ModuleId`		
Description	STM Module ID		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	255		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-

Table 297 Specification for ModuleId (continued)

Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.5 Release

Table 298 Specification for Release

Name	`Release`		
Description	Aurix derivative used for the implementation,		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	As per hardware derivative		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.6 SwMajorVersion

Table 299 Specification for SwMajorVersion

Name	`SwMajorVersion`		
Description	Major version number of the vendor specific implementation of the module. The numbering is vendor specific		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.7 SwMinorVersion

Table 300 Specification for SwMinorVersion

Name	SwMinorVersion		
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.8 SwPatchVersion

Table 301 Specification for SwPatchVersion

Name	SwPatchVersion		
Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.2.9 VendorId

Table 302 Specification for VendorId

Name	VendorId		
Description	STM Vendor ID		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

Table 302	Specification for VendorId (continued)		
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.3 Container: StmGeneral

Stm General configuration container
Post-Build Variant Multiplicity: FALSE
Multiplicity Configuration Class: -

5.3.1.3.1 StmDevErrorDetect

Table 303	Specification for StmDevErrorDetect		
Name	`StmDevErrorDetect`		
Description	STM Development error detection enable/disable		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.3.2 StmRuntimeApiMode

Table 304	Specification for StmRuntimeApiMode
Name	`StmRuntimeApiMode`
Description	Stm Runtime Api Mode Configuration

Table 304 Specification for StmRuntimeApiMode (continued)

Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	STM_MCAL_SUPERVISOR: STM Mcal Supervisor Mode Selection STM_MCAL_USER1: STM Mcal User1 Mode Selection		
Default value	STM_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.1.3.3 StmVersionInfoApi

Table 305 Specification for StmVersionInfoApi

Name	`StmVersionInfoApi`		
Description	Enable/Disable Stm_GetVersionInfo() API.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

5.3.2 Functions - Type definitions

5.3.2.1 Stm_CallbackFnPtrType

Table 306 Specification for Stm_CallbackFnPtrType

Syntax	`Stm_CallbackFnPtrType`
Type	Pointer to a function of type void Function_Name (void)

Table 306 Specification for Stm_CallbackFnPtrType (continued)

File	`Stm.h`
Description	Function pointer used for Call-back function.
Source	IFX

5.3.2.2 Stm_TotalTimerCaptureType

Table 307 Specification for Stm_TotalTimerCaptureType

Syntax	`Stm_TotalTimerCaptureType`	
Type	Structure	
File	`Stm.h`	
Range	uint32 LowerPart	Lower 32 bit value of 64 bit timer.
	uint32 UpperPart	Upper 32 bit value of 64 bit timer.
Description	Holds the complete 64 bit STM timer value.	
Source	IFX	

5.3.3 Functions - APIs

5.3.3.1 Stm_EnableModule

Table 308 Specification for `Stm_EnableModule` API

Syntax	`void Stm_EnableModule ( const uint8 ModuleNumber )`	
Service ID	0x0	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	ModuleNumber	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	none
Description	By default, all the STM timers will run after power on reset. The API does:	

Table 308 Specification for `Stm_EnableModule` API (continued)

	i) Enable the STM in such a way that all the 32 bits of CMP1 register values are compared with first 32 bit values (TIM0) of 64 bit STM. ii) Enable the compare match interrupt. By default, tie the interrupt node with compare register - - CMP0 compare match interrupt directed to interrupt node 0 - CMP1 compare match interrupt directed to interrupt node 1 v) This service resets the RAM corresponding to the STM module number. In multi core context, user shall ensure that only the allocated STM peripheral resource is accessed in the core.
Source	IFX
Error handling	DET: STM_E_CORE_TIMER_MISMATCH: Wrong STM Module number is passed to the function. Runtime Errors: None DEM: STM_E_CLC_ENABLE_ERR: The hardware does not react in the expected time (hardware malfunction) Safety Errors: None *Note: All DET IDs are also reported as safety errors.*
Configuration dependencies	-
User hints	None

5.3.3.2 Stm_EnableAlarm

Table 309 Specification for `Stm_EnableAlarm` API

Syntax	`void Stm_EnableAlarm ( const uint8 ModuleNumber, const uint8 CompareRegisterId, const uint8 TimerMode, const uint32 Ticks, const Stm_CallbackFnPtrType Stm_Applicationfunction )`	
Service ID	0x1	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant for different STM timers	
Parameters (in)	ModuleNumber CompareRegisterId TimerMode	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon.

Table 309 Specification for `Stm_EnableAlarm` API (continued)

	Ticks Stm_Applicationfunction	Compare register number. CompareRegisterId value can be 0 or 1. Defines the STM running mode: 0: One Shot, 1: Continuous Timer ticks w.r.t selected timer. By default TIM0 is used as Timernumber for reference to invoke a Call-back function based on values of Ticks. If user wants to change the TimerNumber then user should call the API Stm_SetCompareMatchControl () with appropriate Mstart number. Function pointer used for Call-back function. Stm_CallbackFnPtrType is defined as void (*Stm_CallbackFnPtrType) (void). NULL_PTR is a invalid function
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	A service shall be provided to invoke a call-back function after the elapse of configured amount of time (scheduling a function). This callback function shall be called either once or continuously after the elapse of the time. The timeout shall be ticks. In multi core context, user shall ensure that only the allocated STM peripheral resource is accessed in the core.	
Source	IFX	
Error handling	DET: STM_E_CORE_TIMER_MISMATCH: Wrong STM Module number is passed to the function. STM_E_CMPREG_FAILED: Invalid STM compare register number is passed. STM_E_TIMER_MODE_FAILED: Invalid timer mode is passed to the function. STM_E_PARAM_POINTER: Invalid Call back function is passed to the function. Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	None	

5.3.3.3 Stm_DisableAlarm

Table 310 Specification for `Stm_DisableAlarm` API

Syntax	`void Stm_DisableAlarm (`

Table 310 Specification for `Stm_DisableAlarm` API (continued)

	`const uint8 ModuleNumber,`
	`const uint8 CompareRegisterId`
	`)`	

Service ID	0x2	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant for different STM timers	
Parameters (in)	ModuleNumber CompareRegisterId	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon. Compare register number. CompareRegisterId value can be 0 or 1.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	none
Description	This function stops invoking call-back function. In multi core context, user shall ensure that only the allocated STM peripheral resource is accessed in the core.	
Source	IFX	
Error handling	DET: STM_E_CMPREG_FAILED: Invalid STM compare register number is passed. STM_E_CORE_TIMER_MISMATCH: Wrong STM Module number is passed to the function. Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	None	

5.3.3.4 Stm_SetCompareMatchControl

Table 311 Specification for `Stm_SetCompareMatchControl` API

Syntax	`void Stm_SetCompareMatchControl`
	`(`
	`const uint8 ModuleNumber,`
	`const uint8 CompareRegisterId,`
	`const uint8 Mstart,`

Table 311 Specification for `Stm_SetCompareMatchControl` API (continued)

	`const uint8 MSize` `)`	
Service ID	0x03	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant for different STM timers	
Parameters (in)	ModuleNumber CompareRegisterId Mstart MSize	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon. Compare register number. CompareRegisterId value can be 0 or 1. The lowest bit number of the 64-bit STM that is compared with the content of Compare register. Mstart values can be 0 to 31. MSize values can be 0 to 31. The number of bits in register CMP0 or CMP1 (starting from bit 0) that are used for the compare operation with the System Timer.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	API Service shall be provided to write the CMCON register: Match Start and Match Size for a particular STM module. In multi core context, user shall ensure that only the allocated STM peripheral resource is accessed in the core.	
Source	IFX	
Error handling	DET: STM_E_MSTART_FAILED: Invalid lowest bit number of the 64-bit STM is passed STM_E_CMPREG_FAILED: Invalid STM compare register number is passed. STM_E_CORE_TIMER_MISMATCH: Wrong STM Module number is passed to the function. STM_E_MSIZE_FAILED: Invalid msize is passed to the API Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	None	

5.3.3.5 Stm_ReadTimerValue

Table 312 Specification for `Stm_ReadTimerValue` API

Syntax	`uint32 Stm_ReadTimerValue ( const uint8 ModuleNumber, const uint8 TimerNumber )`	
Service ID	0x04	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant for different STM timers	
Parameters (in)	ModuleNumber TimerNumber	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon. The TimerNumber values shall be 0 (TIM0), 1(TIM1), 2(TIM2), 3(TIM3), 4 (TIM4), 5(TIM5) and 6(TIM6).
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	uint32	Current running timer value in ticks.
Description	Service to read individual TIM values of particular STM modules shall be provided. In multi core context, user shall ensure that only the allocated STM peripheral resource is accessed in the core.	
Source	IFX	
Error handling	DET: STM_E_CORE_TIMER_MISMATCH: Wrong STM Module number is passed to the function. STM_E_INV_TIMER_NUMBER: Invalid STM timer number is passed Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	None	

5.3.3.6 Stm_ReadTotalTimerValue

Table 313 Specification for `Stm_ReadTotalTimerValue` API

Syntax	`Stm_TotalTimerCaptureType Stm_ReadTotalTimerValue ( const uint8 ModuleNumber )`	
Service ID	0x05	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant for different STM timers	
Parameters (in)	ModuleNumber	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Stm_TotalTimerCaptureType	LowerPart : Lower 32 bit value of 64 bit timer. UpperPart : Upper 32 bit value of 64 bit timer
Description	Service to read complete 64 bit STM Timer value of a particular STM module shall be provided. In multi core context, user shall ensure that only the allocated STM peripheral resource is accessed in the core.	
Source	IFX	
Error handling	DET: STM_E_CORE_TIMER_MISMATCH: Wrong STM Module number is passed to the function. Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	None	

5.3.3.7 Stm_SleepModeHandle

Table 314 Specification for `Stm_SleepModeHandle` API

Syntax	`void Stm_SleepModeHandle ( const uint8 ModuleNumber,`

Table 314 Specification for `Stm_SleepModeHandle` API (continued)

	`const uint8 SleepmodeControl` `)`	
Service ID	0x06	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	ModuleNumber SleepmodeControl	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon. The values can be 0 : Enable STM during Controller sleep mode. 1 : Disable STM during Controller sleep mode.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Service to enable/disable of STM during controller sleep mode. In multi core context, user shall ensure that only the allocated STM peripheral resource is accessed in the core.	
Source	IFX	
Error handling	DET: STM_E_CORE_TIMER_MISMATCH: Wrong STM Module number is passed to the function. STM_E_SLEEP_MODE_FAILED: Invalid sleep mode control is passed to the function. Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	None	

5.3.3.8 Stm_GetVersionInfo

Table 315 Specification for `Stm_GetVersionInfo` API

Syntax	`void Stm_GetVersionInfo`
	`(`
	`Std_VersionInfoType * const versioninfo`
	`)`

Service ID	0x7

Table 315 Specification for `Stm_GetVersionInfo` API (continued)

Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	versioninfo	Pointer to where to store the version information of this module.
Parameters (in - out)	-	-
Return	void	-
Description	This function returns the version information of this module. The version information include: Module ID, Vendor ID and Vendor specific version numbers.	
Source	IFX	
Error handling	DET: STM_E_PARAM_POINTER: Invalid Call back function is passed to the function. Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	StmVersionInfoApi	
User hints	None	

5.3.4 Notifications and Callbacks

5.3.4.1 Stm_CallbackFunction

Table 316 Specification for `Stm_CallbackFunction` API

Syntax	`void Stm_CallbackFunction ( void )`
Service ID	
Sync/Async	Synchronous
ASIL Level	QM
Re-entrancy	Non Reentrant
Parameters (in)	- -

Table 316 Specification for `Stm_CallbackFunction` API (continued)

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Stm call back notification: After configured elapsed amount of time, user will get the call back notification.	
Source	IFX	
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	-	

5.3.5 Scheduled functions

There are no scheduled functions.

5.3.6 Interrupt service routines

5.3.6.1 Stm_Isr

Table 317 Specification for `Stm_Isr` API

Syntax	`void Stm_Isr`
	`(`
	`  const uint8 ModuleNumber,`
	`  const uint8 InterruptNode`
	`)`	

Service ID	0x8	
Sync/Async	Synchronous	
ASIL Level	QM	
Re-entrancy	Non Reentrant	
Parameters (in)	ModuleNumber InterruptNode	System timer peripheral number. The values shall be 0, 1, 2,3,4 and 5 based on the number of STM peripheral available on the silicon.

Table 317 Specification for `Stm_Isr` API (continued)

		0 or 1
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	none
Description	This function is the interrupt handler and collects the interrupt node and invoke the call-back function	
Source	IFX	
Error handling	DET: None Runtime Errors: None DEM: None Safety Errors: None *Note: All DET IDs are also reported as safety errors.*	
Configuration dependencies	-	
User hints	None	

5.3.7 Error codes classification

This section explains various error types and their corresponding source APIs.

5.3.7.1 Development errors

The following table lists all the development errors reported by the driver.

Table 318 Description of development errors reported

Description	Source	Error code and value	Applicable APIs
Wrong STM Module number is passed to the function.	IFX	STM_E_CORE_TIMER_MISMATCH=0x65	Stm_SleepModeHandle, Stm_ReadTotalTimerValue, Stm_ReadTimerValue, Stm_SetCompareMatchControl, Stm_DisableAlarm, Stm_EnableAlarm, Stm_EnableModule
Invalid STM compare register number is passed.	IFX	STM_E_CMPREG_FAILED=0x1	Stm_SetCompareMatchControl, Stm_DisableAlarm, Stm_EnableAlarm
Invalid timer mode is passed to the function.	IFX	STM_E_TIMER_MODE_FAILED=0x2	Stm_EnableAlarm
Invalid Call back function is passed to the function.	IFX	STM_E_PARAM_POINTER=0x3	Stm_GetVersionInfo, Stm_EnableAlarm

Table 318 Description of development errors reported (continued)

Description	Source	Error code and value	Applicable APIs
Invalid lowest bit number of the 64-bit STM is passed	IFX	STM_E_MSTART_FAILED=0x4	Stm_SetCompareMatchControl
Invalid sleep mode control is passed to the function.	IFX	STM_E_SLEEP_MODE_FAILED=0x5	Stm_SleepModeHandle
Invalid STM timer number is passed.	IFX	STM_E_INV_TIMER_NUMBER=0x6	Stm_ReadTimerValue
Invalid msize is passed to the API.	IFX	STM_E_MSIZE_FAILED=0x7	Stm_SetCompareMatchControl

5.3.7.2 Production errors

The following table lists all the production errors reported by the driver.

Table 319 Description of production errors reported

Description	Source	Error code and value	Applicable APIs
The hardware does not react in the expected time (hardware malfunction).	IFX	STM_E_CLC_ENABLE_ERR=0	Stm_EnableModule

5.3.7.3 Safety errors

The driver does not report any safety errors.

5.3.7.4 Runtime errors

The driver does not report any runtime errors.

5.3.8 Deviations and limitations

The section describes the deviations and limitations from software specification.

5.3.8.1 Deviations

No deviations are reported from the requirement.

5.3.8.2 Limitations

There are no limitations from software specification.

5.3.9 Unsupported hardware features

All the features of STM are supported by the STM driver.

Revision history

Major changes since the last revision

Date	Version	Description
2019-10-10	1.30.0_9.0	• Iom - Memory mapping in Integration hints section is updated • I2c - Memory mapping in Integration hints section is updated - Service ID of APIs is updated - Example usage section is updated - Notifications and callbacks section is updated • Hssl - Memory mapping in Integration hints section is updated - Error handling description for APIs is updated
2019-08-05	8.0	Reference to the BASIC User Manual is updated.
2019-07-26	7.0	Reference to the BASIC User Manual is updated.
2019-07-23	6.0	• Hssl - Integration hints section is updated. - API descriptions are updated. Hssl_GetChannelError API is added. • Sent - API descriptions is updated. - Error table is updated. - Limitations and deviations section is updated. • Stm - Stm_SetCompareMatchControl API is updated.
2019-04-22	5.0	Added support for the TC37xA and TC37xA_ED devices.
2019-04-11	4.0	• Added support for the TC35xA device. • Added the IOM, SENT and HSSL drivers.
2019-02-05	3.0	Updated the *Functions - Type definitions* section for the I2C driver.
2019-02-04	2.0	Updated the *Integration hints* and *Reference information* for all modules.
2018-10-12	1.0	Initial version.

Trademarks

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

Edition 2019-10-10

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-cwx1559810041751

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