

# MCAL User Manual for I2c

## 32-bit TriCore™ AURIX™ TC3xx microcontroller

### About this document

#### Scope and purpose

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

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

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

#### Intended audience

This document is intended for anyone using the I2c module of the TC3xx MCAL software.

#### Document conventions

**Table 1** Conventions

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

#### Reference documents

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

- AURIX™ TC3xx MCAL User Manual General

## Table of contents

	<b>About this document</b> .....	1
	<b>Table of contents</b> .....	2
<b>1</b>	<b>I2C driver</b> .....	5
1.1	User information .....	5
1.1.1	Description .....	5
1.1.2	Hardware-software mapping .....	5
1.1.2.1	I2C: primary hardware peripheral .....	5
1.1.2.2	SCU: dependent hardware peripheral .....	6
1.1.2.3	Port: dependent hardware peripheral .....	6
1.1.2.4	SRC: dependent hardware peripheral .....	7
1.1.3	File structure .....	7
1.1.3.1	C file structure .....	7
1.1.3.2	Code generator plugin files .....	8
1.1.4	Integration hints .....	10
1.1.4.1	Integration with AUTOSAR stack .....	10
1.1.4.2	Multicore and Resource Manager .....	12
1.1.4.3	MCU support .....	12
1.1.4.4	Port support .....	13
1.1.4.5	DMA support .....	14
1.1.4.6	Interrupt connections .....	14
1.1.4.7	Example usage .....	17
1.1.5	Key architectural considerations .....	24
1.1.5.1	FIFO configuration .....	24
1.1.5.2	Peripheral configuration .....	25
1.2	Assumptions of Use (AoU) .....	25
1.3	Reference information .....	26
1.3.1	Configuration interfaces .....	26
1.3.1.1	Container: I2c .....	26
1.3.1.2	Container: I2cPublishedInformation .....	26
1.3.1.2.1	I2cMaxHwUnit .....	26
1.3.1.3	Container: I2cConfigSet .....	27
1.3.1.4	Container: I2cChannelConfiguration .....	27
1.3.1.4.1	I2cHwUnit .....	27
1.3.1.4.2	I2cSpeed .....	28
1.3.1.4.3	I2cAddressingMode .....	28
1.3.1.4.4	I2cFractionalDividerDec .....	28
1.3.1.4.5	I2cFractionalDividerInc .....	29
1.3.1.4.6	I2cRmc .....	29
1.3.1.4.7	I2cSclDelayStageHoldTimeStartBit .....	30

**Table of contents**

1.3.1.4.8	I2cSdaDelayStageDataHoldTime .....	30
1.3.1.4.9	I2cSetFastModeSclLowPerTime .....	31
1.3.1.4.10	I2cFastModeSclLowLength .....	31
1.3.1.4.11	I2cEnCfgFastModeSclLowLength .....	31
1.3.1.4.12	I2cAsyncNotification .....	32
1.3.1.4.13	I2cPacketEndNotification .....	32
1.3.1.4.14	I2cTxTimeOut .....	33
1.3.1.4.15	I2cRxTimeOut .....	33
1.3.1.4.16	I2cSDASelect .....	34
1.3.1.4.17	I2cSCLSelect .....	34
1.3.1.5	Container: CommonPublishedInformation .....	35
1.3.1.5.1	ArPatchVersion .....	35
1.3.1.5.2	ArMajorVersion .....	35
1.3.1.5.3	ArMinorVersion .....	36
1.3.1.5.4	SwMajorVersion .....	36
1.3.1.5.5	SwMinorVersion .....	37
1.3.1.5.6	SwPatchVersion .....	37
1.3.1.5.7	ModuleId .....	37
1.3.1.5.8	VendorId .....	38
1.3.1.5.9	Release .....	38
1.3.1.6	Container: I2cGeneral .....	39
1.3.1.6.1	I2cDevErrorDetect .....	39
1.3.1.6.2	I2cVersionInfoApi .....	39
1.3.1.6.3	I2cInitDelInitApiMode .....	40
1.3.1.6.4	I2cSystemClock .....	40
1.3.2	Functions - Type definitions .....	41
1.3.2.1	I2c_ConfigType .....	41
1.3.2.2	I2c_ChannelType .....	41
1.3.2.3	I2c_ChannelConfigType .....	41
1.3.2.4	I2c_AddressingModeType .....	42
1.3.2.5	I2c_NotifFunctionPtrType .....	42
1.3.2.6	I2c_ErrorType .....	42
1.3.2.7	I2c_SizeType .....	43
1.3.2.8	I2c_DataType .....	43
1.3.2.9	I2c_SlaveAddrType .....	43
1.3.2.10	I2c_ChannelStatusType .....	44
1.3.3	Functions - APIs .....	44
1.3.3.1	I2c_Init .....	44
1.3.3.2	I2c_DelInit .....	45
1.3.3.3	I2c_GetStatus .....	46
1.3.3.4	I2c_SyncWrite .....	46
1.3.3.5	I2c_SyncRead .....	47

---

**Table of contents**

1.3.3.6	I2c_AsyncWrite .....	48
1.3.3.7	I2c_AsyncRead .....	49
1.3.3.8	I2c_CancelOperation .....	50
1.3.3.9	I2c_GetVersionInfo .....	51
1.3.4	Notifications and callbacks .....	52
1.3.5	Scheduled functions .....	52
1.3.6	Interrupt service routines .....	52
1.3.6.1	I2c_IsrI2cDtr .....	52
1.3.6.2	I2c_IsrI2cProtocol .....	53
1.3.6.3	I2c_IsrI2cError .....	53
1.3.7	Callout .....	54
1.3.8	Error Handling .....	54
1.3.9	Deviations and limitations .....	54
1.3.9.1	Deviations .....	55
1.3.9.1.1	Software specification deviations .....	55
1.3.9.1.2	AMDC violations .....	55
1.3.9.1.3	VSMD violations .....	55
1.3.9.2	Limitations .....	55
	<b>Revision history</b> .....	56
	<b>Disclaimer</b> .....	57

## 1 I2C driver

### 1.1 User information

#### 1.1.1 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.

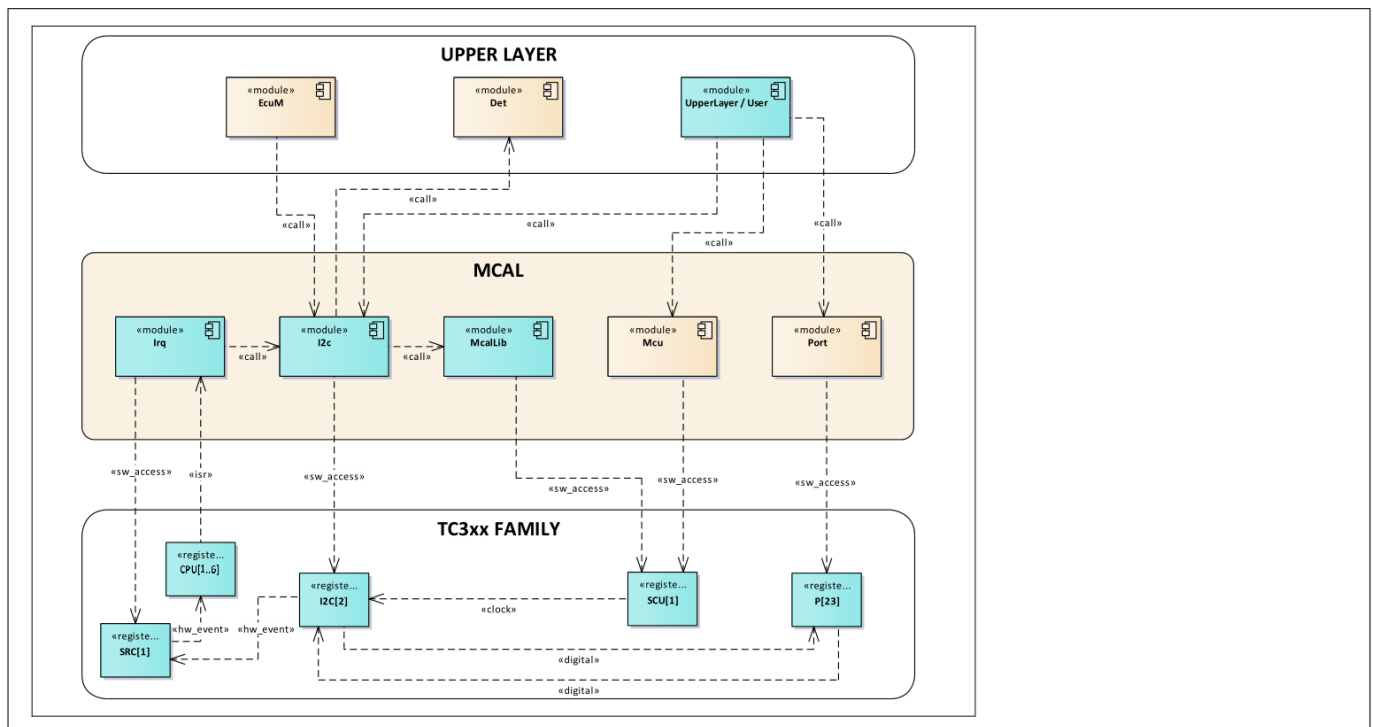
The I2C driver does not support the Slave mode.

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

#### 1.1.2 Hardware-software mapping

This section describes the system view of the I2C driver and peripherals administered by it.



**Figure 1** Mapping of hardware-software interfaces

##### 1.1.2.1 I2C: primary hardware peripheral

###### Hardware functional features

The key I2C features used by the I2C driver are:

- Master mode

---

**I2C driver**

- 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

The unsupported feature of the I2C is:

- Slave mode

**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.

### **1.1.2.2 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.

### **1.1.2.3 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.

#### 1.1.2.4 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.

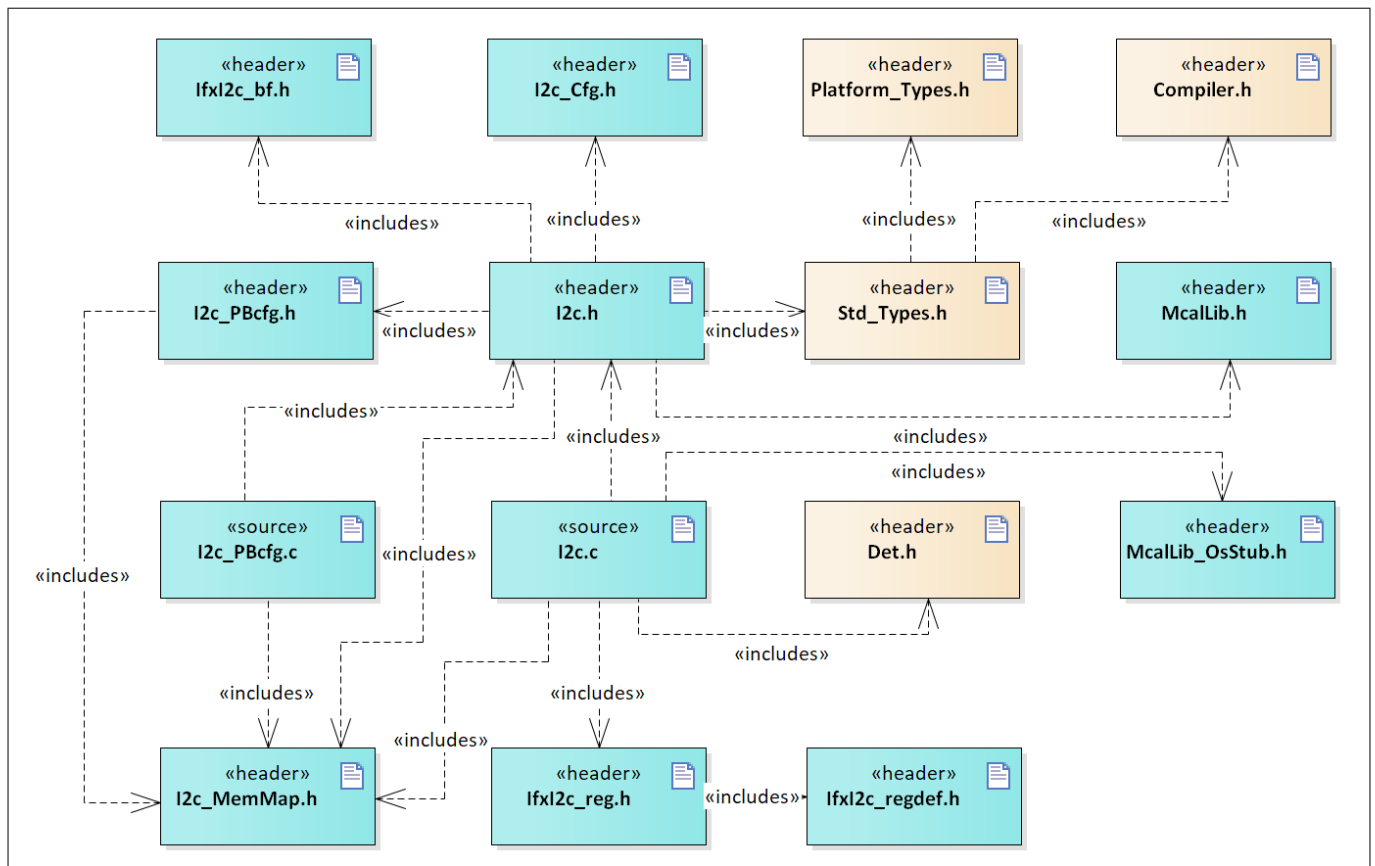
## Hardware events

None.

### 1.1.3 File structure

### 1.1.3.1 C file structure

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



**Figure 2** **C file structure**

**I2C driver**
**Table 2 C file structure**

<b>File name</b>	<b>Description</b>
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
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

### 1.1.3.2 Code generator plugin files

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



**I2C driver**

**Figure 3**      **Code generator plugin files**
**Table 3**      **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

## **1.1.4 Integration hints**

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

### **1.1.4.1 Integration with AUTOSAR stack**

This section lists the modules, which are not part of MCAL, but are required to integrate the I2C 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 `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

**I2C driver**

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
```

## I2C driver

```
#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.

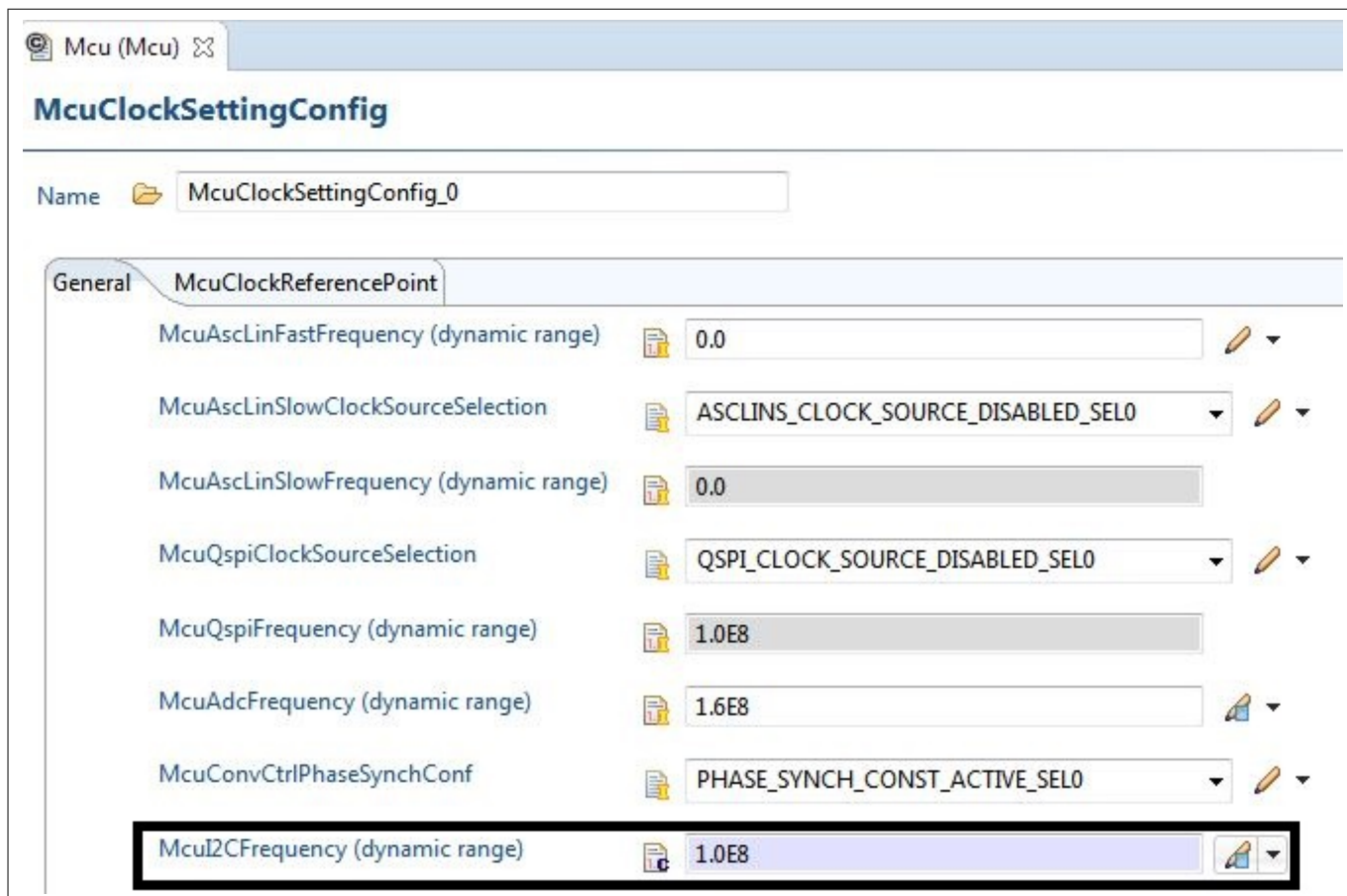
Operating system files provided by MCAL package is only an example code and must be updated by the integrator with the actual OS files for the desired function.

### 1.1.4.2 Multicore and Resource Manager

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

### 1.1.4.3 MCU support

The I2C driver is dependent on MCU driver for clock generation. The `fl2C` defines the application clock frequency for the I2C Kernel. The `fl2C` which is derived from PLL2 (200MHz) is independent on `fSPB` and allows the I2C to operate at a constant baud rate (frequency). The required `fl2C` 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:

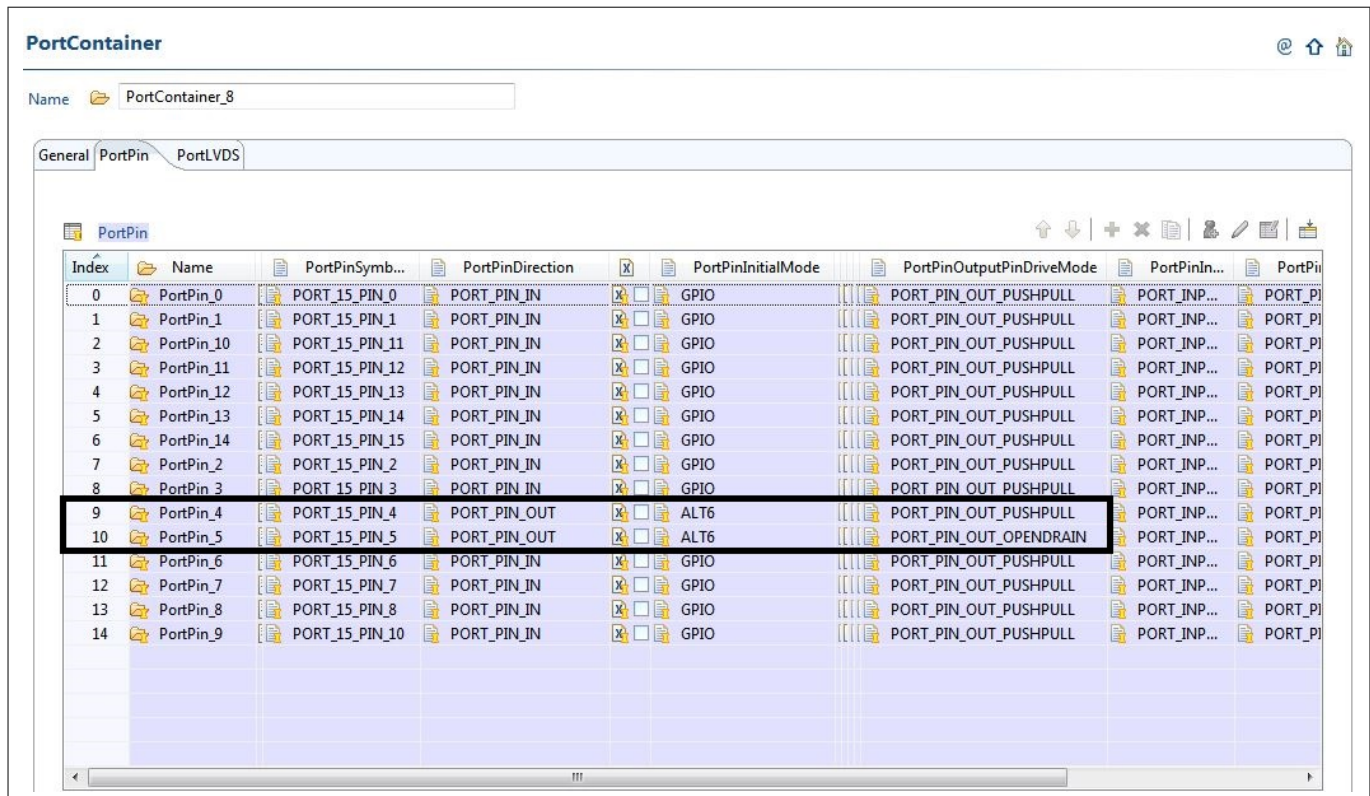


**Figure 4**      **Mcu Configuration**

#### 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 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:

**I2C driver**

**Figure 5 Port Configuration**

### 1.1.4.5 DMA support

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

### 1.1.4.6 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\_I2COP is used for protocol interrupt. User must ensure that the interrupt service routine provided by I2C

**I2C driver**

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 function */
    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 function */
    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.

**I2C driver**

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
```



**I2C driver**
**1.1.4.7 Example usage**

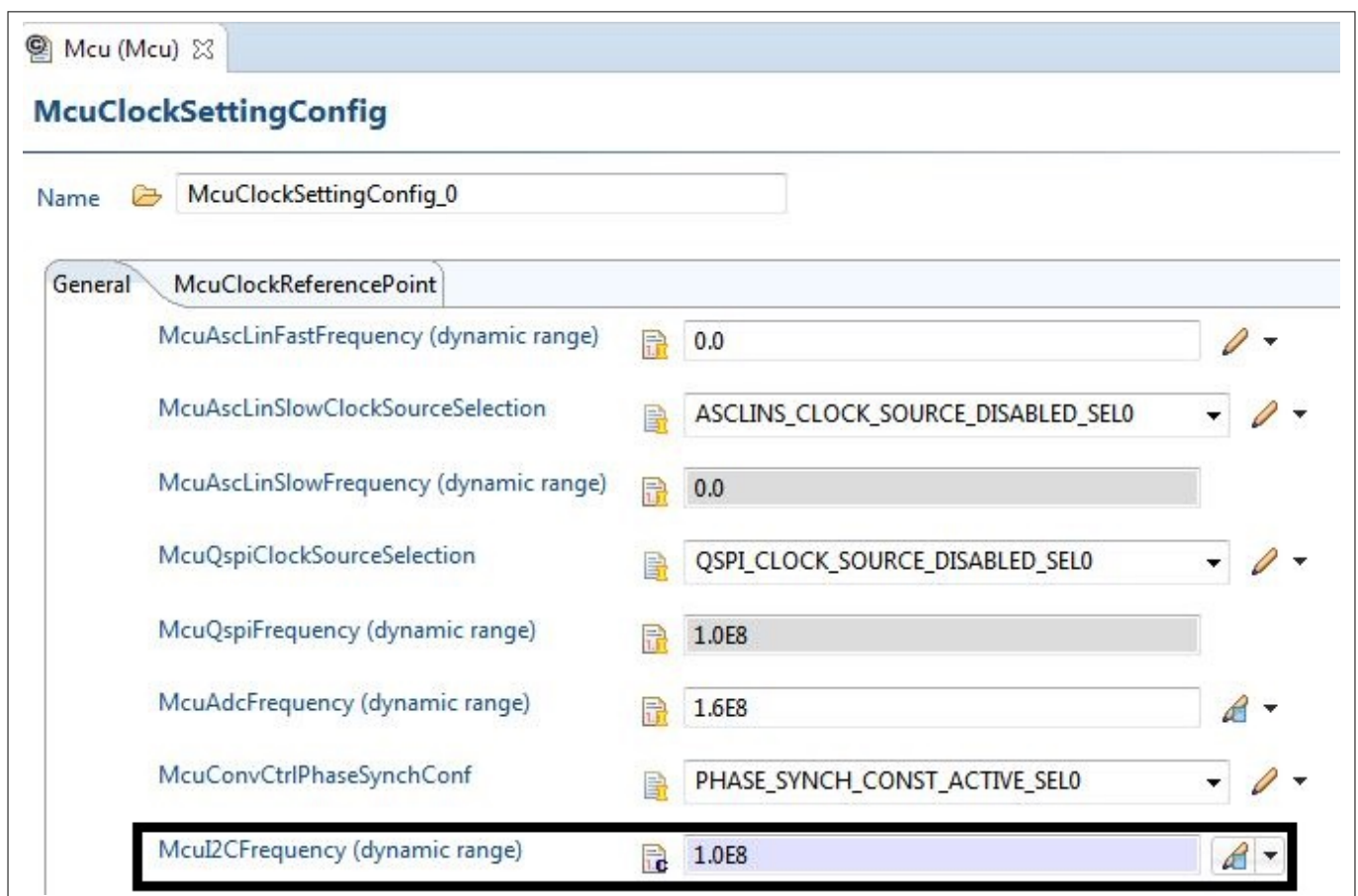
Examples of I2C driver API usage are as follows:

**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 6 Mcu Configuration**

## PortPin

Name

PortPin\_5

General
PortPinMode

PortPinId (dynamic range)

245

PortPinSymbolicName

PORT\_15\_PIN\_5

PortPinDirection

PORT\_PIN\_OUT

PortPinDirectionChangeable

☐

PortPinInitialMode

ALT6

PortPinLevelValue

PORT\_PIN\_LEVEL\_HIGH

PortPinModeChangeable

☐

PortPinInputPullResistor

PORT\_PIN\_IN\_PULL\_UP

PortPinOutputPadDriveStrength

PORT\_PIN\_DEFAULT\_DRIVER

PortPinOutputPinDriveMode

PORT\_PIN\_OUT\_OPENDRAIN

PortPinInputPadLevel

PORT\_INPUT\_LEVEL\_CMOS\_AUTOMOTIVE

PortPinEnableAnalogInputOnly

PORT\_PIN\_ANALOG\_INPUT\_DISABLE

PortPinEmergencyStop

☐

PortPinControllerSelect

DISABLE

**Figure 7**      **Port Pin Configuration**

## PortPin

Name


PortPin\_4

General
PortPinMode

PortPinId (dynamic range)	<div> <div>244</div> </div>
PortPinSymbolicName	<div> <div>PORT_15_PIN_4</div> <div></div> </div>
PortPinDirection	<div> <div>PORT_PIN_OUT</div> <div></div> </div>
PortPinDirectionChangeable	<div> <div></div> </div>
PortPinInitialMode	<div> <div>ALT6</div> <div></div> </div>
PortPinLevelValue	<div> <div>PORT_PIN_LEVEL_HIGH</div> <div></div> </div>
PortPinModeChangeable	<div> <div></div> </div>
PortPinInputPullResistor	<div> <div>PORT_PIN_IN_PULL_UP</div> </div>
PortPinOutputPadDriveStrength	<div> <div>PORT_PIN_DEFAULT_DRIVER</div> <div></div> </div>
PortPinOutputPinDriveMode	<div> <div>PORT_PIN_OUT_PUSH_PULL</div> <div></div> </div>
PortPinInputPadLevel	<div> <div>PORT_INPUT_LEVEL_CMOS_AUTOMOTIVE</div> </div>
PortPinEnableAnalogInputOnly	<div> <div>PORT_PIN_ANALOG_INPUT_DISABLE</div> </div>
PortPinEmergencyStop	<div> <div></div> <div></div> </div>
PortPinControllerSelect	<div> <div>DISABLE</div> </div>


**Figure 8** Port Pin Configuration

**IrqI2cConfig**


Name  IrqI2cConfig\_0



General



**IrqI2cPrioConfig**

Name  IrqI2cPrioConfig


**IrqI2cDtrPrioConfig**



Name  IrqI2cDtrPrioConfig



IrqI2c0DtrPrio (0 -> 255)  1 

IrqI2c1DtrPrio (0 -> 255)  0 


**IrqI2cErrPrioConfig**



Name  IrqI2cErrPrioConfig



IrqI2c0ErrPrio (0 -> 255)  2 

IrqI2c1ErrPrio (0 -> 255)  0 

**IrqI2cPPrioConfig**


Name  IrqI2cPPrioConfig

IrqI2c0PPrio (0 -> 255)  3 


IrqI2c1PPrio (0 -> 255)  0 

**Figure 9**      **Irq Configuration**















**I2C driver**


 I2c (I2c)

**I2cChannelConfiguration**

Name  I2cChannelConfiguration\_0

General

I2cHwUnit	 I2C_0
I2cSDASelect	 SDA0C_PORT15_PIN5
I2cSCLSelect	 SCL0C_PORT15_PIN4
I2cSpeed	 STANDARD_MODE
I2cAddressingMode	 I2C_7_BIT_ADDRESSING
I2cFractionalDividerDec (1 -> 2047)	 997
I2cFractionalDividerInc (1 -> 255)	 2
I2cRmc (1 -> 255)	 1
I2cSdaDelayStageDataHoldTime (0 -> 63)	 0
I2cScIDelayStageHoldTimeStartBit (0 -> 7)	 0
I2cSetFastModeScILowPerTime	 <input type="checkbox"/>
I2cFastModeScILowLength (0 -> 255)	 0
I2cAsyncNotification	 <input checked="" type="checkbox"/>
I2cPacketEndNotification	 I2c_NotificationPtrFun

I2cEnCfgFastModeScILowLength  ☐

**Figure 10 I2c Channel Configuration**
**Initializing the driver**

## I2C driver

The code sequence for initializing I2C driver is as follows.

```
#include "McalLib.h"
#include "I2c.h"
#include "Mcu.h"
#include "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);
```

## I2C driver

### 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);
```

## I2C driver

### 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*/
}
```

## 1.1.5 Key architectural considerations

The key architectural considerations are as follows:

### 1.1.5.1 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



---

**I2C driver**

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.

### **1.1.5.2 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.

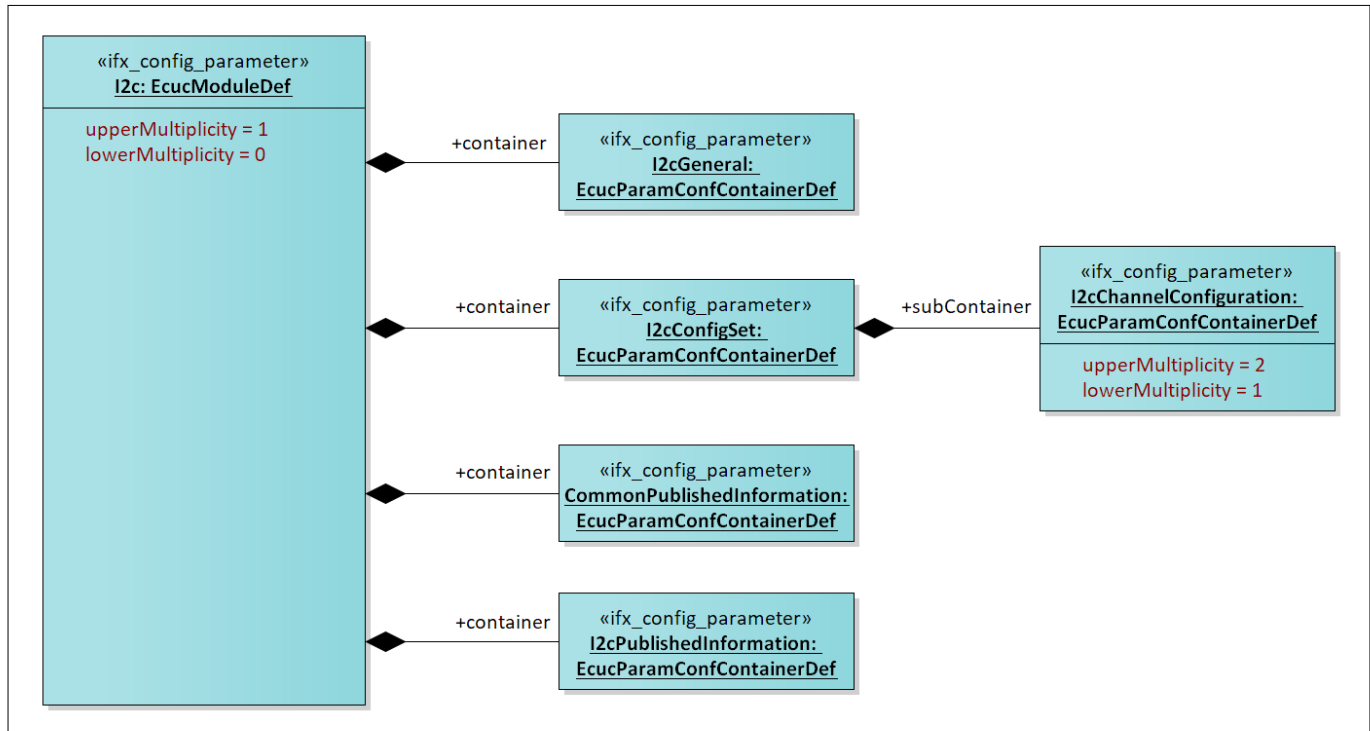
## **1.2 Assumptions of Use (AoU)**

There are no AoU for the I2C driver.

## 1.3 Reference information

### 1.3.1 Configuration interfaces

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



**Figure 11** Container hierarchy along with their configuration parameters

#### 1.3.1.1 Container: I2c

Configuration of the I2C (I2c driver) module

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

#### 1.3.1.2 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

Multiplication Class: Pre-Compile

##### 1.3.1.2.1 I2cMaxHwUnit

**Table 4** Specification for I2cMaxHwUnit

<b>Name</b>	I2cMaxHwUnit		
<b>Description</b>	The parameter represents maximum supported I2c hardware units.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucIntegerParamDef
(table continues...)			

**Table 4 (continued) Specification for I2cMaxHwUnit**

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

### 1.3.1.3 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: -

### 1.3.1.4 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

Multiplication Class: Pre-Compile

#### 1.3.1.4.1 I2cHwUnit

**Table 5 Specification for I2cHwUnit**

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

## I2C driver

### 1.3.1.4.2 I2cSpeed

**Table 6 Specification for I2cSpeed**

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

### 1.3.1.4.3 I2cAddressingMode

**Table 7 Specification for I2cAddressingMode**

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

### 1.3.1.4.4 I2cFractionalDividerDec

**Table 8 Specification for I2cFractionalDividerDec**

<b>Name</b>	I2cFractionalDividerDec
-------------	-------------------------

(table continues...)

**I2C driver**
**Table 8 (continued) Specification for I2cFractionalDividerDec**

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

**1.3.1.4.5 I2cFractionalDividerInc**
**Table 9 Specification for I2cFractionalDividerInc**

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

**1.3.1.4.6 I2cRmc**
**Table 10 Specification for I2cRmc**

<b>Name</b>	I2cRmc		
<b>Description</b>	This parameter contains Rmc value of the CLC1 register.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucIntegerParamDef
<b>Range</b>	1-255		
<b>Default value</b>	1		
<b>Post-build variant value</b>	TRUE	<b>Post-build variant multiplicity</b>	-

(table continues...)

## I2C driver

**Table 10** (continued) **Specification for I2cRmc**

<b>Value configuration class</b>	Post-Build	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

**1.3.1.4.7 I2cSclDelayStageHoldTimeStartBit**
**Table 11** **Specification for I2cSclDelayStageHoldTimeStartBit**

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

**1.3.1.4.8 I2cSdaDelayStageDataHoldTime**
**Table 12** **Specification for I2cSdaDelayStageDataHoldTime**

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

### 1.3.1.4.9 I2cSetFastModeSclLowPerTime

**Table 13 Specification for I2cSetFastModeSclLowPerTime**

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

### 1.3.1.4.10 I2cFastModeSclLowLength

**Table 14 Specification for I2cFastModeSclLowLength**

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

### 1.3.1.4.11 I2cEnCfgFastModeSclLowLength

**Table 15 Specification for I2cEnCfgFastModeSclLowLength**

<b>Name</b>	I2cEnCfgFastModeSclLowLength		
<b>Description</b>	This parameter enables Direct Configuration of SCL Low Period Length in Fast Mode.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucBooleanParamDef

(table continues...)

**I2C driver**
**Table 15 (continued) Specification for I2cEnCfgFastModeSclLowLength**

<b>Range</b>	TRUE FALSE		
<b>Default value</b>	FALSE		
<b>Post-build variant value</b>	TRUE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Post-Build	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	I2cSpeed		

**1.3.1.4.12 I2cAsyncNotification**
**Table 16 Specification for I2cAsyncNotification**

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

**1.3.1.4.13 I2cPacketEndNotification**
**Table 17 Specification for I2cPacketEndNotification**

<b>Name</b>	I2cPacketEndNotification		
<b>Description</b>	This parameter is a reference to a notification function.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucFunctionNameDef
<b>Range</b>	String		
<b>Default value</b>	NULL		
<b>Post-build variant value</b>	TRUE	<b>Post-build variant multiplicity</b>	-

(table continues...)



**I2C driver**
**Table 17 (continued) Specification for I2cPacketEndNotification**

<b>Value configuration class</b>	Post-Build	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

**1.3.1.4.14 I2cTxTimeOut**
**Table 18 Specification for I2cTxTimeOut**

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

**1.3.1.4.15 I2cRxTimeOut**
**Table 19 Specification for I2cRxTimeOut**

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

### 1.3.1.4.16 I2cSDASelect

**Table 20 Specification for I2cSDASelect**

<b>Name</b>	I2cSDASelect		
<b>Description</b>	<p>This parameter selects the port pin for SDA line.</p> <p>Refer DS for the list of pins applicable for specific I2C, format of pin description is as below:</p> <p>SDAxy_PORTz_PINk</p> <p>x - represents 0 to 1 based on the AURIX variant</p> <p>y - represents A, B, C, DN, DP, CN</p> <p>z - represents the port number</p> <p>k - represents the pin number</p> <p>Respective Alt-x function to be selected from the configuration.</p> <p>This parameter is IFX specific to make use of Hardware provided capability for selecting the right SDA pins.</p>		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucStringParamDef
<b>Range</b>	String		
<b>Default value</b>	Depends on Micro variant		
<b>Post-build variant value</b>	TRUE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Post-Build	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

### 1.3.1.4.17 I2cSCLSelect

**Table 21 Specification for I2cSCLSelect**

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

(table continues...)

**Table 21 (continued) Specification for I2cSCLSelect**

<b>Range</b>	String		
<b>Default value</b>	Depends on Micro variant		
<b>Post-build variant value</b>	TRUE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Post-Build	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

### 1.3.1.5 Container: CommonPublishedInformation

This section describes the parameters published by the I2C driver.

Post-Build Variant Multiplicity: -

Configuration Class: -

#### 1.3.1.5.1 ArPatchVersion

**Table 22 Specification for ArPatchVersion**

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

#### 1.3.1.5.2 ArMajorVersion

**Table 23 Specification for ArMajorVersion**

<b>Name</b>	ArMajorVersion		
<b>Description</b>	Major version number of AUTOSAR specification on which the appropriate implementation is based upon.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucIntegerParamDef

(table continues...)

**Table 23 (continued) Specification for ArMajorVersion**

<b>Range</b>	0-255		
<b>Default value</b>	4		
<b>Post-build variant value</b>	FALSE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Published-Information	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

### 1.3.1.5.3 ArMinorVersion

**Table 24 Specification for ArMinorVersion**

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

### 1.3.1.5.4 SwMajorVersion

**Table 25 Specification for SwMajorVersion**

<b>Name</b>	SwMajorVersion		
<b>Description</b>	Major version number of the vendor specific implementation of the module.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucIntegerParamDef
<b>Range</b>	0-255		
<b>Default value</b>	As per driver		
<b>Post-build variant value</b>	FALSE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Published-Information	<b>Multiplicity configuration class</b>	-

(table continues...)

**I2C driver**
**Table 25 (continued) Specification for SwMajorVersion**

<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

**1.3.1.5.5 SwMinorVersion**
**Table 26 Specification for SwMinorVersion**

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

**1.3.1.5.6 SwPatchVersion**
**Table 27 Specification for SwPatchVersion**

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

**1.3.1.5.7 ModuleId**
**Table 28 Specification for ModuleId**

<b>Name</b>	ModuleId
-------------	----------

**(table continues...)**

**I2C driver**
**Table 28 (continued) Specification for ModuleId**

<b>Description</b>	ModId ID of this module from Module List.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucIntegerParamDef
<b>Range</b>	0-65535		
<b>Default value</b>	255		
<b>Post-build variant value</b>	FALSE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Published-Information	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

**1.3.1.5.8 VendorId**
**Table 29 Specification for VendorId**

<b>Name</b>	VendorId		
<b>Description</b>	Vendor ID of the dedicated implementation of this mode according to the AUTOSAR vendor list.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucIntegerParamDef
<b>Range</b>	0-65535		
<b>Default value</b>	17		
<b>Post-build variant value</b>	FALSE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Published-Information	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

**1.3.1.5.9 Release**
**Table 30 Specification for Release**

<b>Name</b>	Release		
<b>Description</b>	This parameter indicates the TC3xx device derivative used for the implementation.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucStringParamDef
<b>Range</b>	String		
<b>Default value</b>	As per hardware derivative		
<b>Post-build variant value</b>	FALSE	<b>Post-build variant multiplicity</b>	-

(table continues...)

**I2C driver**
**Table 30 (continued) Specification for Release**

<b>Value configuration class</b>	Published-Information	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

**1.3.1.6 Container: I2cGeneral**

General configuration of I2C driver module.

Post-Build Variant Multiplicity: -

Configuration Class: -

**1.3.1.6.1 I2cDevErrorDetect**
**Table 31 Specification for I2cDevErrorDetect**

<b>Name</b>	I2cDevErrorDetect		
<b>Description</b>	Switches the Development Error Detection and Notification ON or OFF true: enabled (ON). false: disabled (OFF).		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucEnumerationParamDef
<b>Range</b>	TRUE FALSE		
<b>Default value</b>	FALSE		
<b>Post-build variant value</b>	FALSE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Pre-Compile	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

**1.3.1.6.2 I2cVersionInfoApi**
**Table 32 Specification for I2cVersionInfoApi**

<b>Name</b>	I2cVersionInfoApi		
<b>Description</b>	Switches the I2c_GetVersionInfo function ON or OFF		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucBooleanParamDef
<b>Range</b>	TRUE FALSE		

(table continues...)

**Table 32 (continued) Specification for I2cVersionInfoApi**

<b>Default value</b>	FALSE		
<b>Post-build variant value</b>	FALSE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Pre-Compile	<b>Multiplicity configuration class</b>	-
<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

### 1.3.1.6.3 I2cInitDeInitApiMode

**Table 33 Specification for I2cInitDeInitApiMode**

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

### 1.3.1.6.4 I2cSystemClock

**Table 34 Specification for I2cSystemClock**

<b>Name</b>	I2cSystemClock		
<b>Description</b>	This parameter refers to the System clock configured in MCU module.		
<b>Multiplicity</b>	1..1	<b>Type</b>	EcucReferenceDef
<b>Range</b>	Reference to Node: McuClockReferencePointConfig		
<b>Default value</b>	NULL		
<b>Post-build variant value</b>	TRUE	<b>Post-build variant multiplicity</b>	-
<b>Value configuration class</b>	Pre-Compile	<b>Multiplicity configuration class</b>	-

(table continues...)



**I2C driver**
**Table 34 (continued) Specification for I2cSystemClock**

<b>Origin</b>	IFX	<b>Scope</b>	LOCAL
<b>Dependency</b>	-		

### 1.3.2 Functions - Type definitions

This section describes all the type definitions used by APIs.

#### 1.3.2.1 I2c\_ConfigType

**Table 35 Specification for I2c\_ConfigType**

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

#### 1.3.2.2 I2c\_ChannelType

**Table 36 Specification for I2c\_ChannelType**

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

#### 1.3.2.3 I2c\_ChannelConfigType

**Table 37 Specification for I2c\_ChannelConfigType**

<b>Syntax</b>	I2c_ChannelConfigType		
<b>Type</b>	Structure		
<b>File</b>	I2c.h		
<b>Range</b>	-	The elements of the data structure are specific to the microcontroller.	

(table continues...)

## I2C driver

**Table 37** (continued) **Specification for I2c\_ChannelConfigType**

<b>Description</b>	This type contains the implementation-specific to Channel configuration structure of the I2C driver.
<b>Source</b>	IFX

### 1.3.2.4 I2c\_AddressingModeType

**Table 38** **Specification for I2c\_AddressingModeType**

<b>Syntax</b>	I2c_AddressingModeType	
<b>Type</b>	Enumeration	
<b>File</b>	I2c.h	
<b>Range</b>	I2C_7_BIT_ADDRESSING	Represents 7-bit addressing mode
	I2C_10_BIT_ADDRESSING	Represents 10-bit addressing mode
<b>Description</b>	This type contains the return type information which is used in various functions.	
<b>Source</b>	IFX	

### 1.3.2.5 I2c\_NotifFunctionPtrType

**Table 39** **Specification for I2c\_NotifFunctionPtrType**

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

### 1.3.2.6 I2c\_ErrorType

**Table 40** **Specification for I2c\_OperationType**

<b>Syntax</b>	I2c_ErrorType	
<b>Type</b>	Enumeration	
<b>File</b>	I2c.h	
<b>Range</b>	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

(table continues...)

**I2C driver**
**Table 40 (continued) Specification for I2c\_OperationType**

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

**1.3.2.7 I2c\_SizeType**
**Table 41 Specification for I2c\_SizeType**

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

**1.3.2.8 I2c\_DataType**
**Table 42 Specification for I2c\_DataType**

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

**1.3.2.9 I2c\_SlaveAddrType**
**Table 43 Specification for I2c\_SlaveAddrType**

<b>Syntax</b>	I2c_SlaveAddrType
<b>(table continues...)</b>	

**I2C driver**
**Table 43 (continued) Specification for I2c\_SlaveAddrType**

<b>Type</b>	uint16	
<b>File</b>	I2c.h	
<b>Range</b>	0-1023	-
<b>Description</b>	This type contains the possible slave address.	
<b>Source</b>	IFX	

### 1.3.2.10 I2c\_ChannelStatusType

**Table 44 Specification for I2c\_ChannelStatusType**

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

## 1.3.3 Functions - APIs

This section lists all the APIs of the I2C driver.

### 1.3.3.1 I2c\_Init

**Table 45 Specification for I2c\_Init API**

<b>Syntax</b>	void I2c_Init ( const I2c_ConfigType* const ConfigPtr )	
<b>Service ID</b>	0x4F	
<b>Sync/Async</b>	Synchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant	
<b>Parameters (in)</b>	ConfigPtr	Pointer to configuration set.
<b>Parameters (out)</b>	-	-

(table continues...)

## I2C driver

**Table 45 (continued) Specification for I2c\_Init API**

<b>Parameters (in - out)</b>	-	-
<b>Return</b>	void	-
<b>Description</b>	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.	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_ALREADY_INITIALIZED, I2C_E_INIT_FAILED	
<b>Configuration dependencies</b>	-	
<b>User hints</b>	None	

### 1.3.3.2 I2c\_DeInit

**Table 46 Specification for I2c\_DeInit API**

<b>Syntax</b>	Std_ReturnType I2c_DeInit ( void )	
<b>Service ID</b>	0x50	
<b>Sync/Async</b>	Synchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant	
<b>Parameters (in)</b>	-	-
<b>Parameters (out)</b>	-	-
<b>Parameters (in - out)</b>	-	-
<b>Return</b>	Std_ReturnType	E_OK: de-initialization command has been accepted. E_NOT_OK: de-initialization command has not been accepted.
<b>Description</b>	This function will de-initialize the driver. It will reset all the I2C SFRs that were configured during the initialization of the driver	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_UNINIT	
<b>Configuration dependencies</b>	-	
<b>User hints</b>	None	

### 1.3.3.3 I2c\_GetStatus

**Table 47 Specification for I2c\_GetStatus API**

<b>Syntax</b>	I2c_ChannelStatusType I2c_GetStatus ( const I2c_ChannelType ChannelId )	
<b>Service ID</b>	0x55	
<b>Sync/Async</b>	Synchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant	
<b>Parameters (in)</b>	ChannelId	I2C channel identifier
<b>Parameters (out)</b>	-	-
<b>Parameters (in - out)</b>	-	-
<b>Return</b>	I2c_ChannelStatusType	I2C_UNINIT : I2C module is uninitialized I2C_IDLE: I2C module is idle I2C_BUSY: I2C module is busy
<b>Description</b>	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.	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL	
<b>Configuration dependencies</b>	-	
<b>User hints</b>	None	

### 1.3.3.4 I2c\_SyncWrite

**Table 48 Specification for I2c\_SyncWrite API**

<b>Syntax</b>	I2c_ErrorType I2c_SyncWrite ( const I2c_ChannelType ChannelId, I2c_DataType *const DataPtr, const I2c_SizeType Size, const I2c_SlaveAddrType SlaveAddress )	
<b>Service ID</b>	0x51	

(table continues...)

**I2C driver**
**Table 48** (continued) **Specification for I2c\_SyncWrite API**

<b>Sync/Async</b>	Synchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant (for same channel)	
<b>Parameters (in)</b>	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
<b>Parameters (out)</b>	-	-
<b>Parameters (in - out)</b>	-	-
<b>Return</b>	I2c_ReturnType	I2C_OK: Operation success I2C_NOT_OK: Operation not success I2C_IS_BUSY: Bus busy
<b>Description</b>	The service I2c_Write() is called to perform Write operation.	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL, I2C_E_PARAM_POINTER, I2C_E_INVALID_SIZE, I2C_E_INVALID_SLAVE_ADDRESS, I2C_E_HW_UNIT_BUSY	
<b>Configuration dependencies</b>	-	
<b>User hints</b>	None	

**1.3.3.5 I2c\_SyncRead**
**Table 49** **Specification for I2c\_SyncRead API**

<b>Syntax</b>	I2c_ErrorType I2c_SyncRead ( const I2c_ChannelType ChannelId, I2c_DataType *const DataPtr, const I2c_SizeType Size, const I2c_SlaveAddrType SlaveAddress ) 	
<b>Service ID</b>	0x52	
<b>Sync/Async</b>	Synchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant (for same channel)	

**(table continues...)**

**I2C driver**
**Table 49 (continued) Specification for I2c\_SyncRead API**

<b>Parameters (in)</b>	ChannelId	I2C channel identifier
	Size	Size of data to be received in Bytes
	SlaveAddress	Address of slave
<b>Parameters (out)</b>	DataPtr	Pointer to data that is received.
<b>Parameters (in - out)</b>	-	-
<b>Return</b>	I2c_ReturnType	I2C_OK : Operation Success I2C_NOT_OK: Operation not success I2C_IS_BUSY: Bus busy
<b>Description</b>	The service I2c_Read() is called to perform Read operation.	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL, I2C_E_PARAM_POINTER, I2C_E_INVALID_SIZE, I2C_E_INVALID_SLAVE_ADDRESS, I2C_E_HW_UNIT_BUSY	
<b>Configuration dependencies</b>	-	
<b>User hints</b>	None	

**1.3.3.6 I2c\_AsyncWrite**
**Table 50 Specification for I2c\_AsyncWrite API**

<b>Syntax</b>	I2c_ErrorType I2c_AsyncWrite ( const I2c_ChannelType ChannelId, I2c_DataType *const DataPtr, const I2c_SizeType Size, const I2c_SlaveAddrType SlaveAddress )	
<b>Service ID</b>	0x53	
<b>Sync/Async</b>	Asynchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant (for same channel)	
<b>Parameters (in)</b>	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

**(table continues...)**



## I2C driver

**Table 50** (continued) **Specification for I2c\_AsyncWrite API**

<b>Parameters (out)</b>	-	-
<b>Parameters (in - out)</b>	-	-
<b>Return</b>	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
<b>Description</b>	The service I2c_AsyncWrite() is called to perform Write operation.	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL, I2C_E_PARAM_POINTER, I2C_E_INVALID_SIZE, I2C_E_INVALID_SLAVE_ADDRESS, I2C_E_HW_UNIT_BUSY	
<b>Configuration dependencies</b>	-	
<b>User hints</b>	None	

### 1.3.3.7 I2c\_AsyncRead

**Table 51** **Specification for I2c\_AsyncRead API**

<b>Syntax</b>	<pre> I2c_ErrorType I2c_AsyncRead (     const I2c_ChannelType ChannelId,     I2c_DataType *const DataPtr,     const I2c_SizeType Size,     const I2c_SlaveAddrType SlaveAddress ) </pre>	
<b>Service ID</b>	0x54	
<b>Sync/Async</b>	Asynchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant (for same channel)	
<b>Parameters (in)</b>	ChannelId	I2C channel identifier
	DataPtr	Pointer to data that needs to be transmitted

(table continues...)

## I2C driver

**Table 51** (continued) **Specification for I2c\_AsyncRead API**

	Size	Size of data to be transmitted in bytes
	SlaveAddress	Address of slave
<b>Parameters (out)</b>	DataPtr	
<b>Parameters (in - out)</b>	-	
<b>Return</b>	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
<b>Description</b>	The service I2c_AsyncRead() is called to perform Read operation.	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL, I2C_E_PARAM_POINTER, I2C_E_INVALID_SIZE, I2C_E_INVALID_SLAVE_ADDRESS, I2C_E_HW_UNIT_BUSY	
<b>Configuration dependencies</b>	-	
<b>User hints</b>	None	

### 1.3.3.8 I2c\_CancelOperation

**Table 52** **Specification for I2c\_CancelOperation API**

<b>Syntax</b>	Std_ReturnType I2c_CancelOperation ( const I2c_ChannelType ChannelId, I2c_SizeType *const TransmittedDataSize ) 	
<b>Service ID</b>	0x56	
<b>Sync/Async</b>	Synchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Non-Reentrant	
<b>Parameters (in)</b>	ChannelId	I2C channel id

(table continues...)

**I2C driver**
**Table 52 (continued) Specification for I2c\_CancelOperation API**

<b>Parameters (out)</b>	TransmittedDataSize	Size transmitted before cancel (in bytes)
<b>Parameters (in - out)</b>	-	-
<b>Return</b>	Std_ReturnType	E_OK: Operation successful E_NOT_OK: Operation unsuccessful
<b>Description</b>	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.	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL, I2C_E_PARAM_POINTER	
<b>Configuration dependencies</b>	I2cAsyncReadWriteEnable	
<b>User hints</b>	None	

**1.3.3.9 I2c\_GetVersionInfo**
**Table 53 Specification for I2c\_GetVersionInfo API**

<b>Syntax</b>	<pre>void I2c_GetVersionInfo (   Std_VersionInfoType * const VersionInfoPtr )</pre>	
<b>Service ID</b>	0x57	
<b>Sync/Async</b>	Synchronous	
<b>Safety Level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Reentrant	
<b>Parameters (in)</b>	-	-
<b>Parameters (out)</b>	VersionInfoPtr	Address where the version information of the I2C module must be stored.
<b>Parameters (in - out)</b>	-	-
<b>Return</b>	void	-
<b>Description</b>	This API returns the version information of this module.  <i>Note: This API is available only when I2cVersionInfoApi is configured as true.</i>	
<b>Source</b>	IFX	
<b>Error handling</b>	I2C_E_PARAM_POINTER	

(table continues...)

**I2C driver**
**Table 53 (continued) Specification for I2c\_GetVersionInfo API**

<b>Configuration dependencies</b>	I2cVersionInfoApi
<b>User hints</b>	None

### 1.3.4 Notifications and callbacks

The I2C driver does not support any notification and callbacks.

### 1.3.5 Scheduled functions

The I2C driver does not support any scheduled functions.

### 1.3.6 Interrupt service routines

This section lists all the interrupt handlers of the I2C driver.

#### 1.3.6.1 I2c\_IsrI2cDtr

**Table 54 Specification for I2c\_IsrI2cDtr**

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

### 1.3.6.2 I2c\_IsrI2cProtocol

**Table 55 Specification for I2c\_IsrI2cProtocol**

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

### 1.3.6.3 I2c\_IsrI2cError

**Table 56 Specification for I2c\_IsrI2cError**

<b>Syntax</b>	void I2c_IsrI2cError ( const uint8 HwUnit )	
<b>Service ID</b>	NA	
<b>Sync/Async</b>	Asynchronous	
<b>Safety level</b>	Refer to the release notes for the safety related info	
<b>Re-entrancy</b>	Reentrant	
<b>Parameters (in)</b>	HwUnit	HW unit index

(table continues...)

**I2C driver**
**Table 56 (continued) Specification for I2c\_IsrI2cError**

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

### 1.3.7 Callout

The I2C driver does not provide any callout.

### 1.3.8 Error Handling

This section describes the various errors reported by the I2C driver.

<b>Error Name: Description</b>	<b>Source</b>	<b>Error ID (AS422)</b>	<b>Type (AS422)</b>	<b>Error ID (AS440)</b>	<b>Type (AS440)</b>
<b>I2C_E_PARAM_POINTER:</b> This error is reported if API Service called with NULL pointer.	IFX	0x00	DET	0x00	DET
<b>I2C_E_UNINIT:</b> This error is reported if API Service used without initialization.	IFX	0x01	DET	0x01	DET
<b>I2C_E_INVALID_CHANNEL :</b> This error is reported if transmission service called at invalid channel.	IFX	0x02	DET	0x02	DET
<b>I2C_E_ALREADY_INITIALIZED :</b> This error is reported if I2C driver is already initialized.	IFX	0x03	DET	0x03	DET
<b>I2C_E_HW_UNIT_BUSY :</b> This error is reported if I2C peripheral is busy.	IFX	0x04	DET	0x04	DET
<b>I2C_E_INVALID_SLAVE_ADDRESS:</b> This error is reported if I2C driver is provided with invalid slave address.	IFX	0x05	DET	0x05	DET
<b>I2C_E_INVALID_SIZE :</b> This error is reported if I2C driver is provided with invalid data size.	IFX	0x06	DET	0x06	DET
<b>I2C_E_INIT_FAILED:</b> This error is reported if I2C driver is provided with NULL config pointer.	IFX	0x07	DET	0x07	DET

### 1.3.9 Deviations and limitations

The section describes the deviations and limitations of the I2C driver.

### **1.3.9.1          Deviations**

This section describes the deviations of the I2C driver.

#### **1.3.9.1.1          Software specification deviations**

The I2C driver does not have any deviations.

#### **1.3.9.1.2          AMDC violations**

The I2C driver does not have any AMDC violations.

#### **1.3.9.1.3          VSMD violations**

The I2C driver does not have any VSMD violations.

### **1.3.9.2          Limitations**

The I2C driver does not have any limitations.

---

**Revision history**

## Revision history

Major changes since the last revision

Date	Version	Description
2023-06-20	3.0	Document is released
2023-05-25	2.1	Safety Level Tagged value added for all API's and ASIL Level tagged value removed since module specific safety level captured in release notes.
2020-11-27	2.0	Document is released
2020-11-26	1.1	<ul style="list-style-type: none"><li>Updated default value of I2cDevErrorDetect</li><li>Error handling format of all the APIs updated in Functions - APIs section</li><li>Error handling section format updated</li></ul>
2020-08-13	1.0	Document is released
2020-08-10	0.1	<ul style="list-style-type: none"><li>Initial version</li><li>I2C driver chapter moved from TC3xx_SW_MCAL_UM_DEMO to this document</li><li>Updated post-build variant value of I2cSystemClock.</li></ul>



## Trademarks

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

**Edition 2023-06-20**

**Published by**

**Infineon Technologies AG**  
**81726 Munich, Germany**

**© 2023 Infineon Technologies AG**  
**All Rights Reserved.**

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

**Email:** [erratum@infineon.com](mailto:erratum@infineon.com)

**Document reference**  
**IFX-gpf1596787328709**

## Important notice

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

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.