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

Reference documents

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

- AURIX™ TC3xx MCAL User Manual General

Table of contents

About this document . . . 1
Table of contents . . . 2
1 I2C driver . . . 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 . . . 29
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 33
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 I2cInitDeInitApiMode 40
1.3.1.6.4 I2cSystemClock 40
1.3.2 Functions - Type definitions 40
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 41
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_DeInit 45
1.3.3.3 I2c_GetStatus 45
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 51
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 52
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 54
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
Revision history 56
Disclaimer 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.

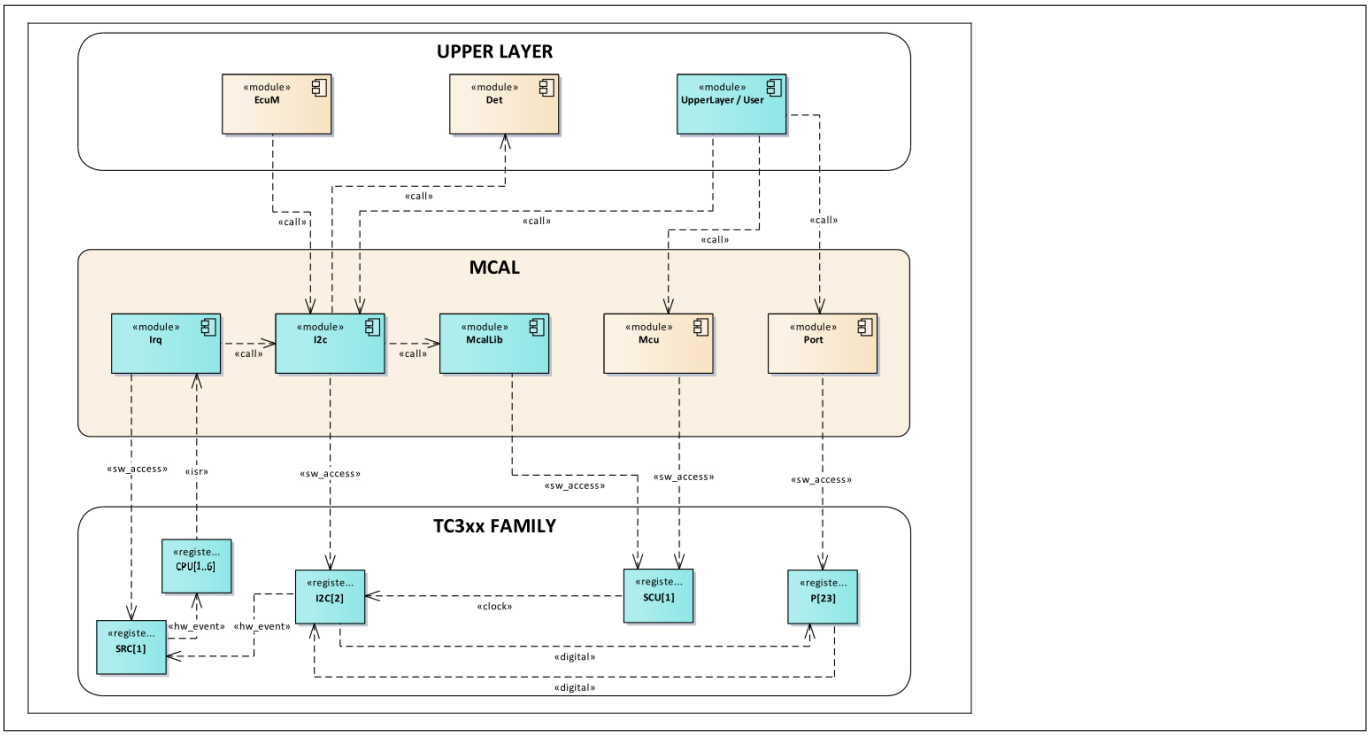


Figure 1 Mapping of hardware-software interfaces

1.1.2.1 I2C: primary hardware peripheral

Hardware functional features

The key I2C features used by the I2C driver are:

- Master mode

- 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.

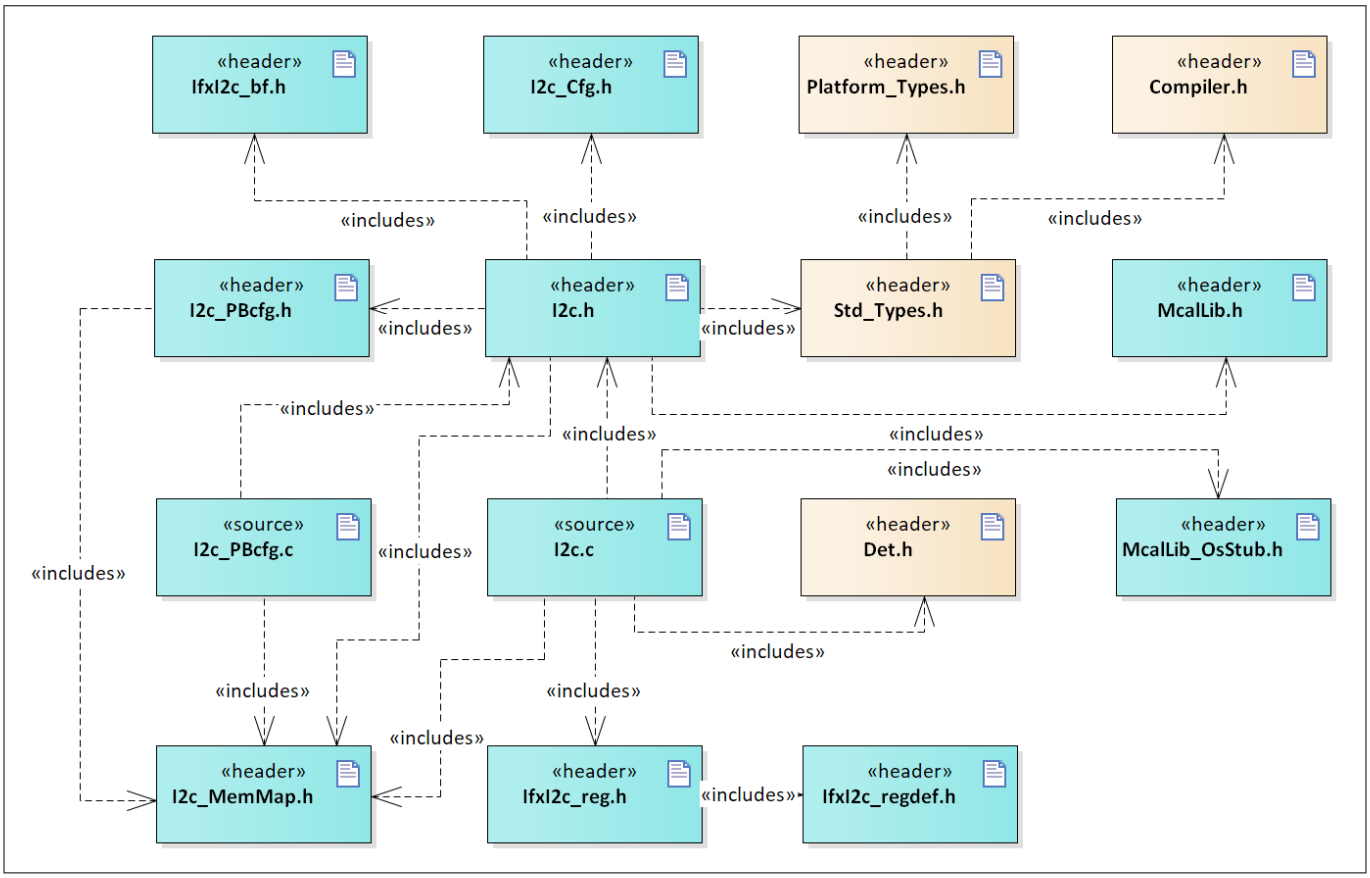


Figure 2 C file structure

Table 2 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
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.

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

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.

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 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:

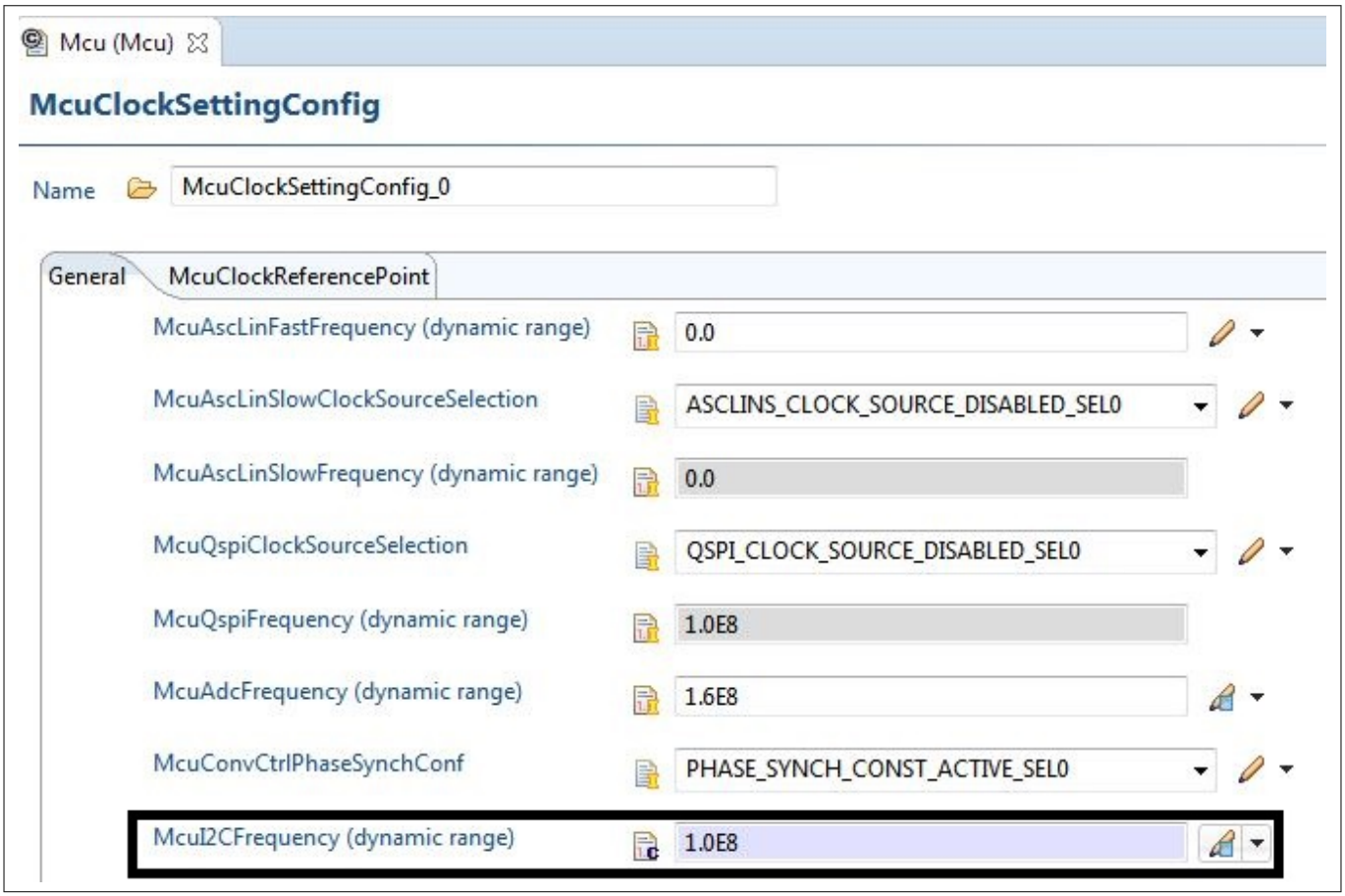
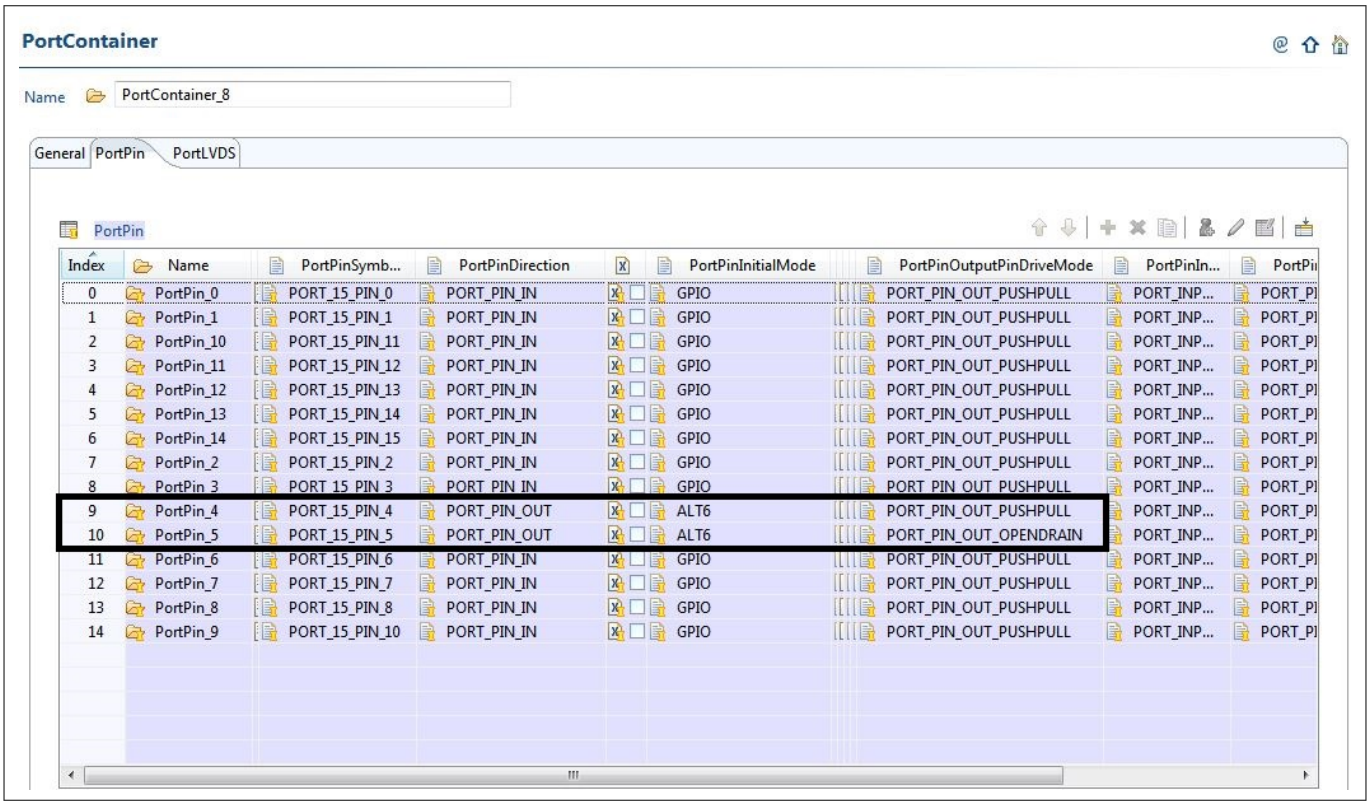


Figure 4 Mcu Configuration

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:



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 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_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
```

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.

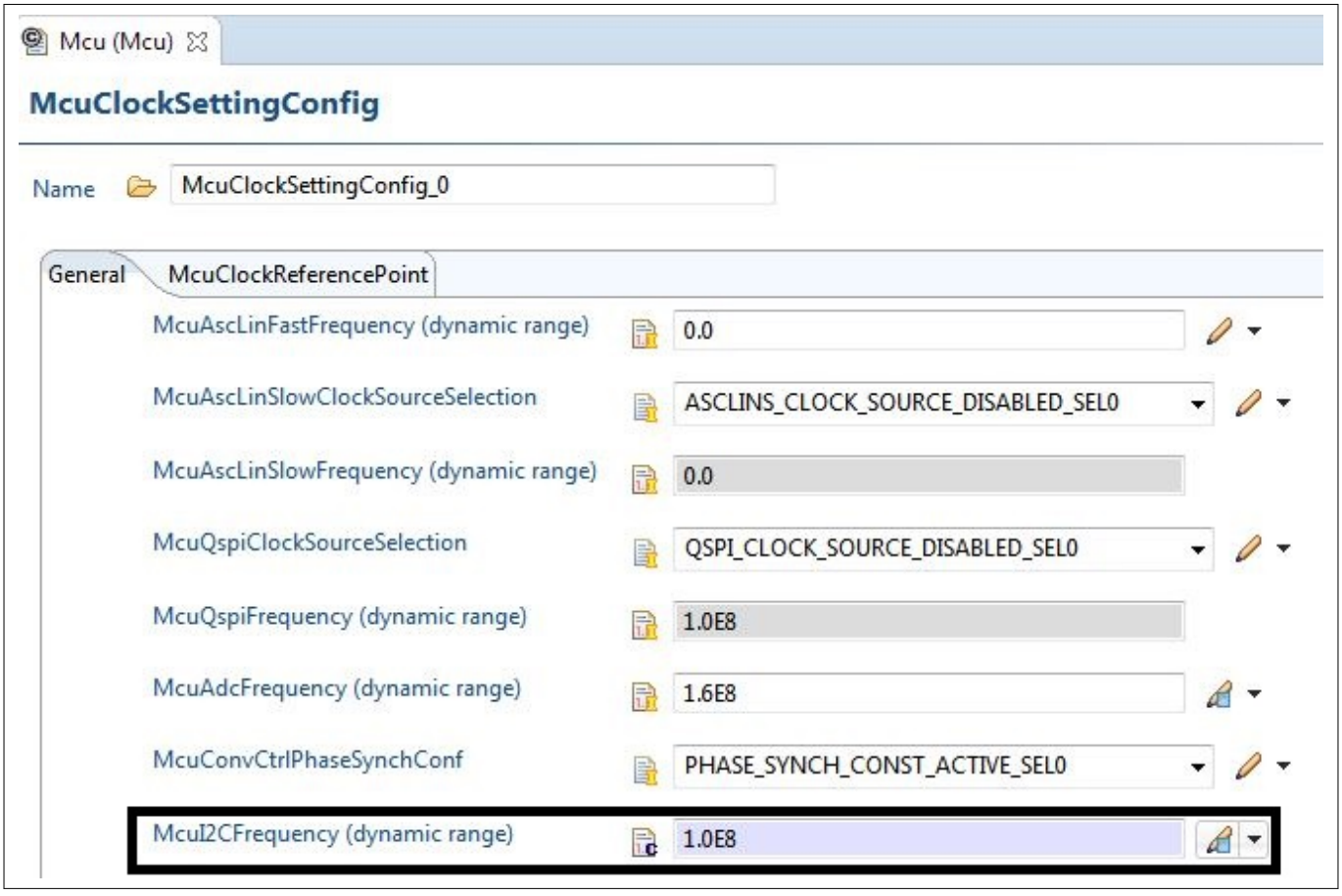


Figure 6 Mcu Configuration

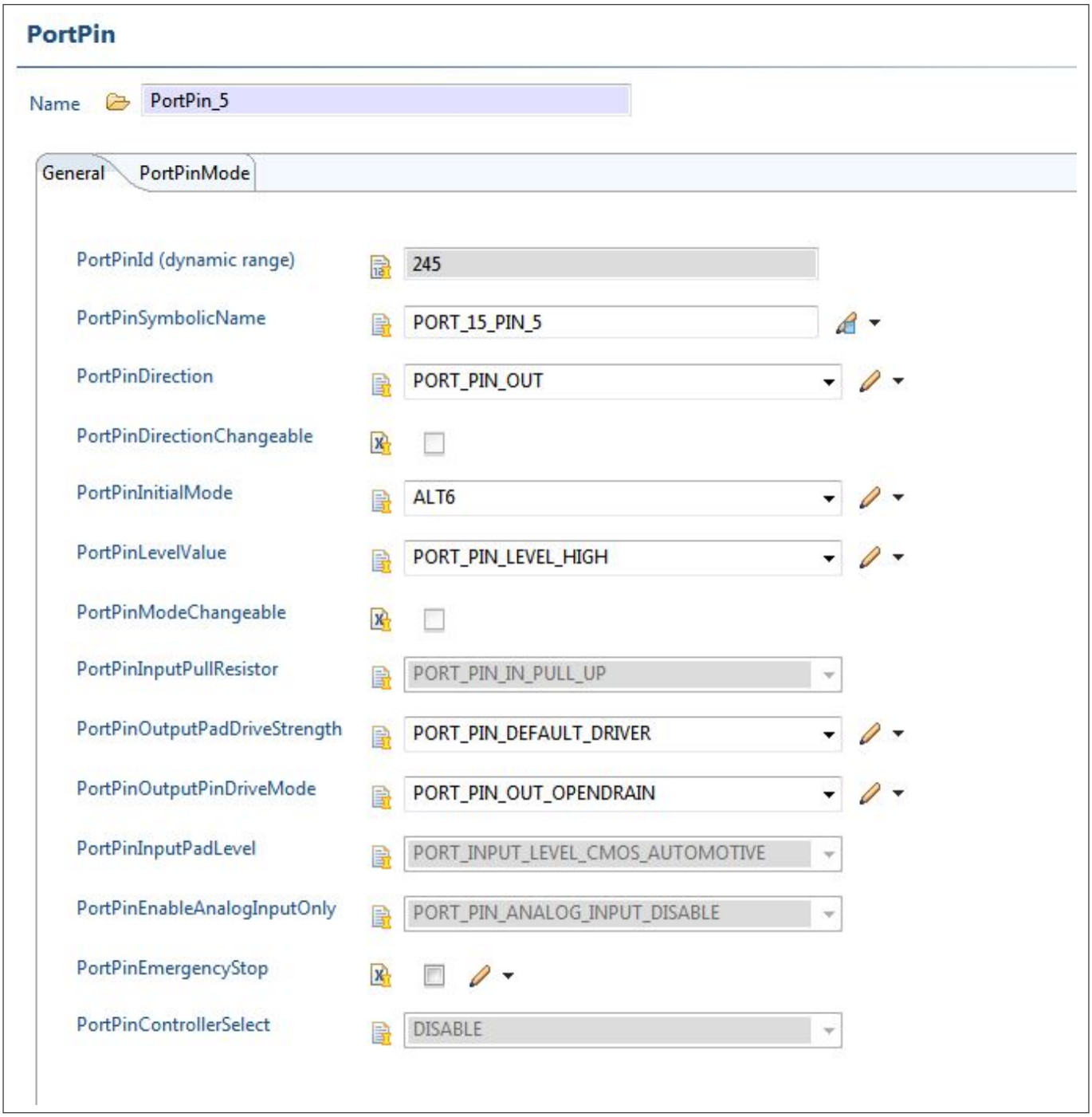


Figure 7 Port Pin Configuration

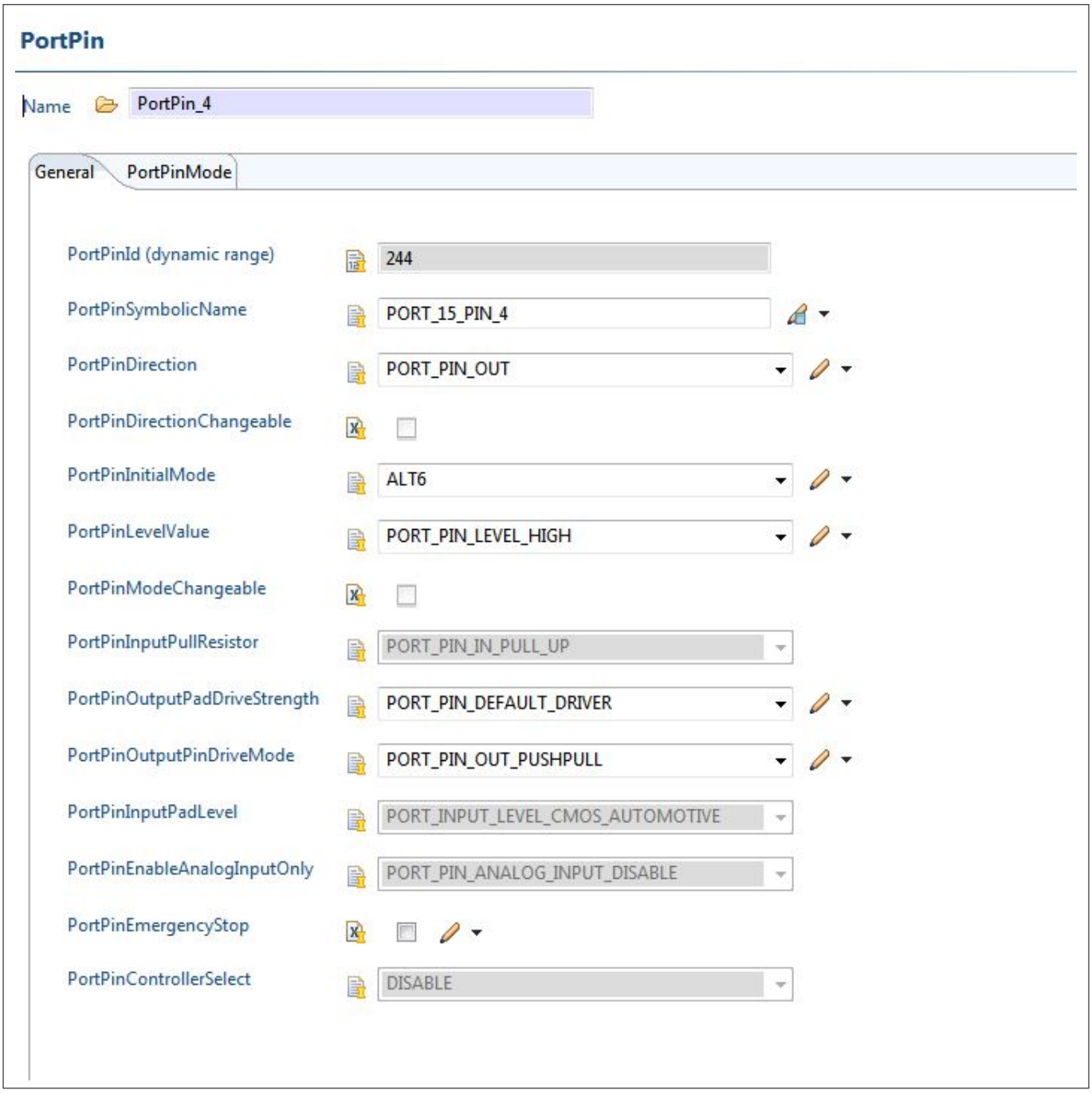


Figure 8 Port Pin Configuration

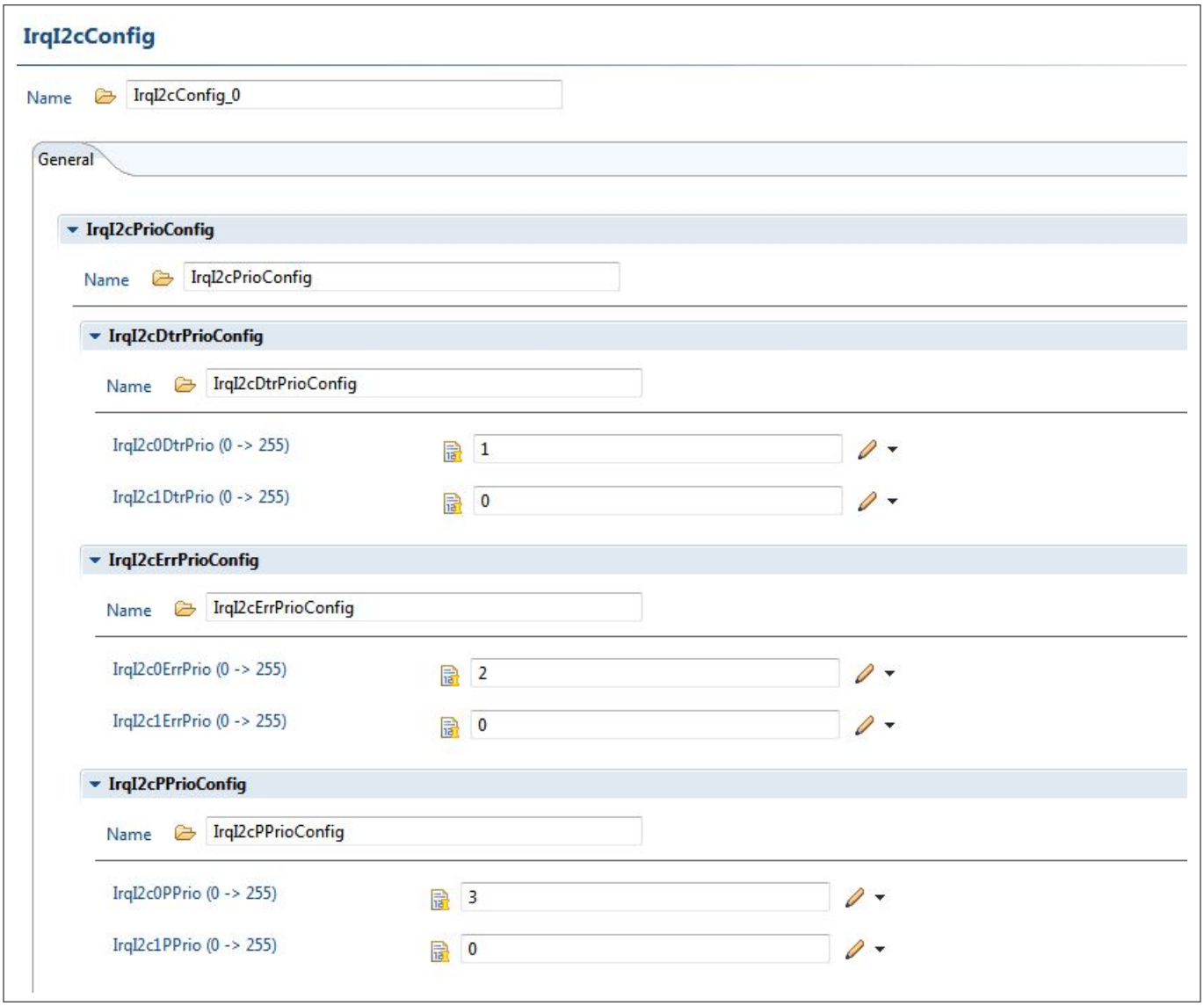


Figure 9 Irq Configuration

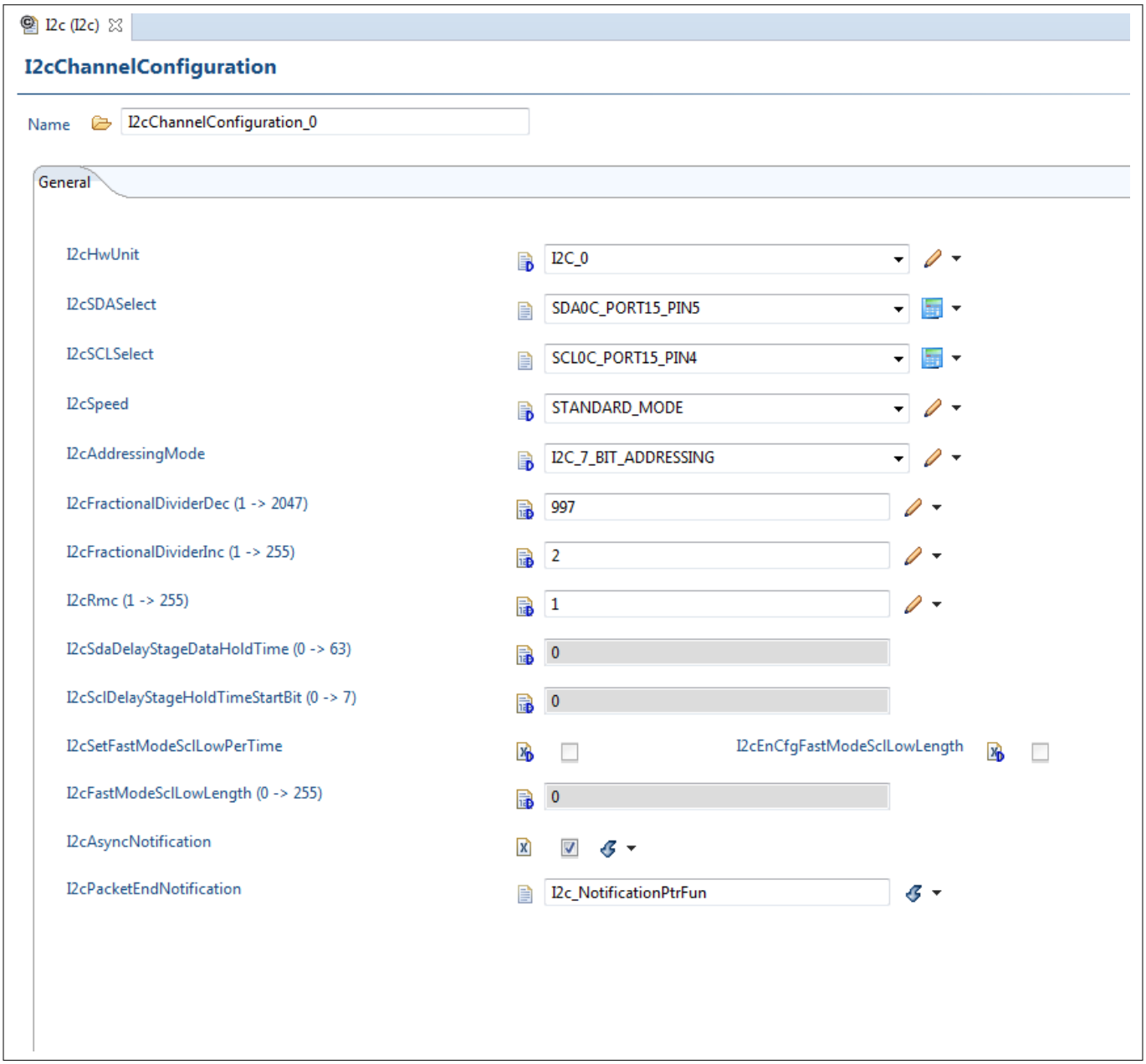


Figure 10 I2c Channel Configuration

Initializing the driver

The code sequence for initializing I2C driver is as follows.

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

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*/
}
```

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 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.

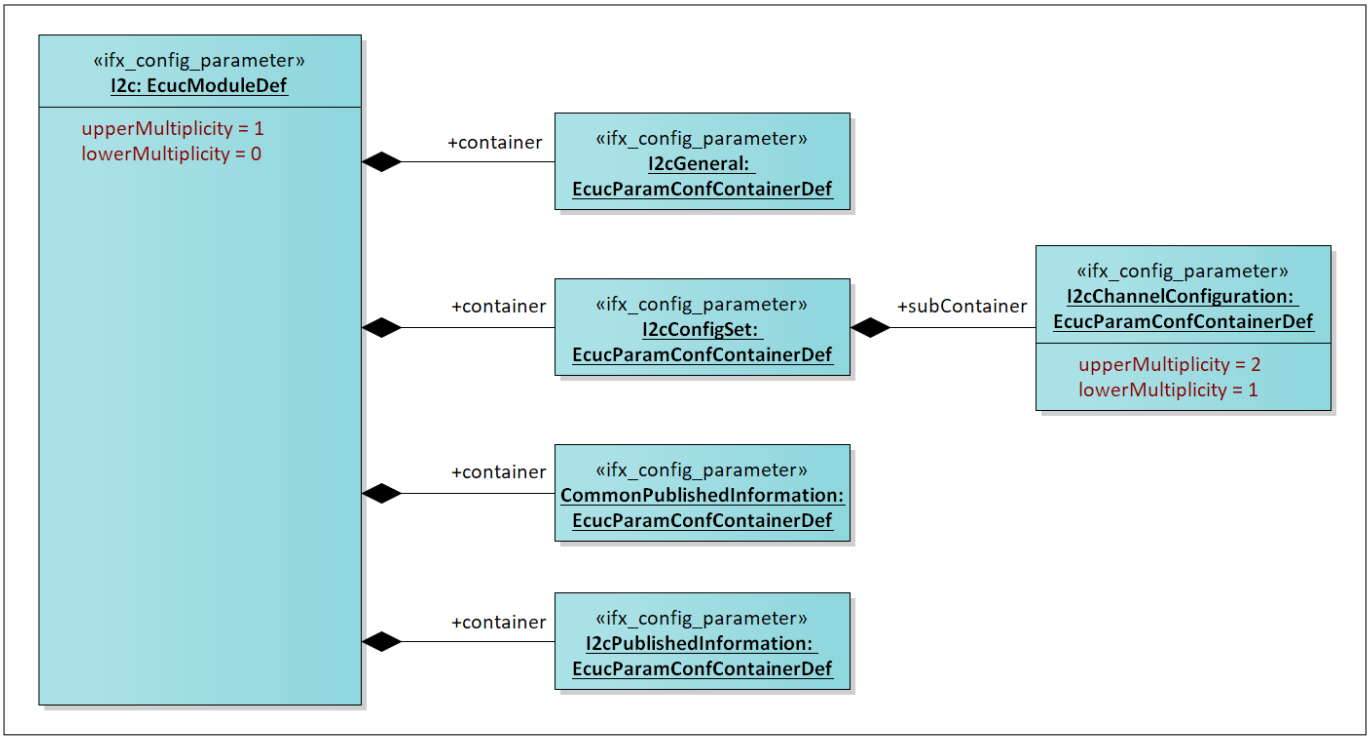


Figure 11 Container hierarchy along with their configuration parameters

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

Multiplicuration Class: Pre-Compile

1.3.1.2.1 I2cMaxHwUnit

Table 4 Specification for I2cMaxHwUnit

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

Table 4 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	-		

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

Multipliciration Class: Pre-Compile

1.3.1.4.1 I2cHwUnit

Table 5 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	-		

1.3.1.4.2 I2cSpeed

Table 6 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	-		

1.3.1.4.3 I2cAddressingMode

Table 7 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	-		

1.3.1.4.4 I2cFractionalDividerDec

Table 8 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	-		

1.3.1.4.5 I2cFractionalDividerInc

Table 9 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	-		

1.3.1.4.6 I2cRmc

Table 10 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 10 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	-		

1.3.1.4.7 I2cSclDelayStageHoldTimeStartBit

Table 11 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	-		

1.3.1.4.8 I2cSdaDelayStageDataHoldTime

Table 12 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	-		

1.3.1.4.9 I2cSetFastModeSclLowPerTime

Table 13 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	-		

1.3.1.4.10 I2cFastModeSclLowLength

Table 14 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		

1.3.1.4.11 I2cEnCfgFastModeSclLowLength

Table 15 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 15 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		

1.3.1.4.12 I2cAsyncNotification

Table 16 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	-		

1.3.1.4.13 I2cPacketEndNotification

Table 17 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 17 Specification for I2cPacketEndNotification (continued)

Origin	IFX	**Scope**	LOCAL
Dependency	-		

1.3.1.4.14 I2cTxTimeOut

Table 18 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	-		

1.3.1.4.15 I2cRxTimeOut

Table 19 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	-		

1.3.1.4.16 I2cSDASelect

Table 20 Specification for I2cSDASelect

Name	`I2cSDASelect`

Table 20 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	-		

1.3.1.4.17 I2cSCLSelect

Table 21 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 21 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	-		

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

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	As per AUTOSAR patch version.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

1.3.1.5.2 ArMajorVersion

Table 23 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 23 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	-		

1.3.1.5.3 ArMinorVersion

Table 24 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	As per AUTOSAR minor version.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

1.3.1.5.4 SwMajorVersion

Table 25 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	-		

1.3.1.5.5 SwMinorVersion

Table 26 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	-		

1.3.1.5.6 SwPatchVersion

Table 27 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	-		

1.3.1.5.7 ModuleId

Table 28 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 28 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	-		

1.3.1.5.8 VendorId

Table 29 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	-		

1.3.1.5.9 Release

Table 30 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	-		

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

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

1.3.1.6.2 I2cVersionInfoApi

Table 32 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	-		

1.3.1.6.3 I2cInitDeInitApiMode

Table 33 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	-		

1.3.1.6.4 I2cSystemClock

Table 34 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	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

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

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	

1.3.2.2 I2c_ChannelType

Table 36 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	

1.3.2.3 I2c_ChannelConfigType

Table 37 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	

1.3.2.4 I2c_AddressingModeType

Table 38 Specification for I2c_AddressingModeType

Syntax	`I2c_AddressingModeType`

Table 38 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	

1.3.2.5 I2c_NotifFunctionPtrType

Table 39 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	

1.3.2.6 I2c_ErrorType

Table 40 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 40 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	

1.3.2.7 I2c_SizeType

Table 41 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	

1.3.2.8 I2c_DataType

Table 42 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	

1.3.2.9 I2c_SlaveAddrType

Table 43 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	

1.3.2.10 I2c_ChannelStatusType

Table 44 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	

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

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	I2C_E_ALREADY_INITIALIZED, I2C_E_INIT_FAILED	

Table 45	Specification for `I2c_Init` API (continued)
Configuration dependencies	-
User hints	None

1.3.3.2 I2c_DeInit

Table 46	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	I2C_E_UNINIT	
Configuration dependencies	-	
User hints	None	

1.3.3.3 I2c_GetStatus

Table 47	Specification for `I2c_GetStatus` API
Syntax	I2c_ChannelStatusType I2c_GetStatus (const I2c_ChannelType ChannelId)

Table 47 Specification for `I2c_GetStatus` API (continued)

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	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL	
Configuration dependencies	-	
User hints	None	

1.3.3.4 I2c_SyncWrite

Table 48 Specification for `I2c_SyncWrite` API

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

Table 48 Specification for `I2c_SyncWrite` API (continued)

	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	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	
Configuration dependencies	-	
User hints	None	

1.3.3.5 I2c_SyncRead

Table 49 Specification for `I2c_SyncRead` API

Syntax	I2c_ErrorType I2c_SyncRead (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.

Table 49 Specification for `I2c_SyncRead` API (continued)

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	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	
Configuration dependencies	-	
User hints	None	

1.3.3.6 I2c_AsyncWrite

Table 50 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

Table 50 Specification for `I2c_AsyncWrite` API (continued)

	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	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	
Configuration dependencies	-	
User hints	None	

1.3.3.7 I2c_AsyncRead

Table 51 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)	-	

Table 51 Specification for `I2c_AsyncRead` API (continued)

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_AsyncRead() is called to perform Read operation.	
Source	IFX	
Error handling	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	
Configuration dependencies	-	
User hints	None	

1.3.3.8 I2c_CancelOperation

Table 52 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 52 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	I2C_E_UNINIT, I2C_E_INVALID_CHANNEL, I2C_E_PARAM_POINTER
Configuration dependencies	I2cAsyncReadWriteEnable
User hints	None

1.3.3.9 I2c_GetVersionInfo

Table 53 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	I2C_E_PARAM_POINTER	
Configuration dependencies	I2cVersionInfoApi	
User hints	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`

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	

1.3.6.2 I2c_IsrI2cProtocol

Table 55 Specification for `I2c_IsrI2cProtocol`

Syntax	void I2c_IsrI2cProtocol (const uint8 HwUnit)

Table 55 Specification for `I2c_IsrI2cProtocol` (continued)

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	

1.3.6.3 I2c_IsrI2cError

Table 56 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
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Handles the error interrupts passed from I2C kernel.	

Table 56 Specification for `I2c_IsrI2cError` (continued)

Source	IFX
Error handling	DET: None
Configuration dependencies	I2cAsyncReadWriteEnable
User hints	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.

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
I2C_E_PARAM_POINTER: This error is reported if API Service called with NULL pointer.	IFX	0x00	DET	0x00	DET
I2C_E _ UNINIT: This error is reported if API Service used without initialization.	IFX	0x01	DET	0x01	DET
I2C_E_INVALID_ CHANNEL : This error is reported if transmission service called at invalid channel.	IFX	0x02	DET	0x02	DET
I2C_E_ALREADY_INITIALIZED : This error is reported if I2C driver is already initialized.	IFX	0x03	DET	0x03	DET
I2C_E_HW_UNIT_BUSY : This error is reported if I2C peripheral is busy.	IFX	0x04	DET	0x04	DET
I2C_E_INVALID_SLAVE_ADDRESS: This error is reported if I2C driver is provided with invalid slave address.	IFX	0x05	DET	0x05	DET
I2C_E_INVALID_SIZE : This error is reported if I2C driver is provided with invalid data size.	IFX	0x06	DET	0x06	DET
I2C_E_INIT_FAILED: 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

Major changes since the last revision

Date	Version	Description
2020-11-27	2.0	Document is released
2020-11-26	1.1	• Updated default value of I2cDevErrorDetect • Error handling format of all the APIs updated in Functions - APIs section • Error handling section format updated
2020-08-13	1.0	Document is released
2020-08-10	0.1	• Initial version • I2C driver chapter moved from TC3xx_SW_MCAL_UM_DEMO to this document • Updated post-build variant value of I2cSystemClock.

Trademarks

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

Edition 2020-11-27

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-gpf1596787328709

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.