
User Manual

for S32K14X I2C Driver

Document Number: UM2I2CASR4.2 Rev0002R1.0.2
Rev. 1.0





Contents

Section number	Title	Page
Chapter 1		
Revision History		
Chapter 2		
Introduction		
2.1	Supported Derivatives.....	13
2.2	Overview.....	13
2.3	About this Manual.....	14
2.4	Acronyms and Definitions.....	14
2.5	Reference List.....	15
2.5.1	Reference for I2C Bus Specification.....	15
Chapter 3		
Driver		
3.1	Requirements.....	17
3.2	Driver Design Summary.....	17
3.3	Hardware Resources.....	17
3.4	Deviation from Requirements.....	19
3.5	Driver limitations.....	19
3.6	Driver usage and configuration tips.....	20
3.6.1	Generic Master Flow.....	20
3.6.1.1	Generic Master Configuration.....	20
3.6.1.2	Synchronous Master Transfer.....	21
3.6.1.3	Asynchronous Master Transfer With Polling.....	21
3.6.1.4	Asynchronous Master Transfer With Callbacks.....	21
3.6.2	Generic Slave Flow.....	22
3.6.2.1	Generic Slave Configuration.....	22
3.6.2.2	Asynchronous Slave Transfer With Polling.....	22
3.6.2.3	Asynchronous Slave Transfer With Callbacks.....	23
3.6.2.4	Controlling RX ACK/NACK Bit.....	24

Section number	Title	Page
3.7	Runtime Errors.....	24
3.8	Development Errors.....	25
3.9	Communication Errors.....	25
3.10	Software specification.....	26
3.10.1	Define Reference.....	26
3.10.1.1	Define I2C_E_ALREADY_INITIALIZED.....	26
3.10.1.2	Define I2C_E_INVALID_BUFFER_SIZE.....	27
3.10.1.3	Define I2C_E_INVALID_CHANNEL.....	27
3.10.1.4	Define I2C_E_INVALID_POINTER.....	27
3.10.1.5	Define I2C_E_UNINIT.....	28
3.10.1.6	Define I2C_INIT.....	28
3.10.1.7	Define I2C_MODULE_ID.....	29
3.10.1.8	Define I2C_UNINIT.....	29
3.10.1.9	Define I2C_DEV_ERROR_DETECT.....	29
3.10.1.10	Define I2C_DISABLE_DEM_REPORT_ERROR_STATUS.....	29
3.10.1.11	Define I2C_DMA_USED.....	30
3.10.1.12	Define I2C_E_ARBITRATION_LOST.....	30
3.10.1.13	Define I2C_E_FIFO_ERROR.....	30
3.10.1.14	Define I2C_E_PIN_LOW_TIMEOUT.....	30
3.10.1.15	Define I2C_E_RX_OVERFLOW.....	31
3.10.1.16	Define I2C_E_SLAVE_BIT_ERROR.....	31
3.10.1.17	Define I2C_E_SLAVE_FIFO_ERROR.....	31
3.10.1.18	Define I2C_E_TX_UNDERFLOW.....	31
3.10.1.19	Define I2C_E_UNEXPECTED_NACK.....	31
3.10.1.20	Define I2C_FLEXIO_FIRST_CHANNEL_U8.....	32
3.10.1.21	Define I2C_FLEXIO_MAX_CHANNELS.....	32
3.10.1.22	Define I2C_HW_MAX_MODULES.....	32
3.10.1.23	Define I2C_LPI2C_MAX_MODULES.....	32
3.10.1.24	Define I2C_ERROR_NOTIFICATION.....	32

Section number	Title	Page
3.10.1.25	Define I2C_MASTER_RECEIVE_COMPLETE_NOTIFICATION.....	33
3.10.1.26	Define I2C_MASTER_TRANSMIT_COMPLETE_NOTIFICATION.....	33
3.10.1.27	Define I2C_SLAVE_ADDR_MATCH_NOTIFICATION.....	34
3.10.1.28	Define I2C_SLAVE_RECEIVE_COMPLETE_NOTIFICATION.....	34
3.10.1.29	Define I2C_SLAVE_TRANSMIT_COMPLETE_NOTIFICATION.....	34
3.10.1.30	Define I2C_PRECOMPILE_SUPPORT.....	35
3.10.1.31	Define I2C_TIMEOUT_LOOPS.....	35
3.10.1.32	Define I2C_VERSION_INFO_API.....	36
3.10.2	Enum Reference.....	36
3.10.2.1	Enumeration I2c_ApiFunctionIdType.....	36
3.10.2.2	Enumeration I2c_DataDirectionType.....	37
3.10.2.3	Enumeration I2C_HwChannelType.....	37
3.10.2.4	Enumeration I2c_ApiFunctionIdType.....	36
3.10.2.5	Enumeration I2C_ASYNChronousMethodType.....	38
3.10.2.6	Enumeration I2C_MASTERSlaveModeType.....	38
3.10.2.7	Enumeration I2c_ModeType.....	39
3.10.2.8	Enumeration I2c_StatusType.....	39
3.10.3	Function Reference.....	39
3.10.3.1	Function I2C_ASYNCTransmit.....	39
3.10.3.2	Function I2c_DeInit.....	40
3.10.3.3	Function I2c_GetStatus.....	41
3.10.3.4	Function I2c_GetVersionInfo.....	41
3.10.3.5	Function I2c_Init.....	42
3.10.3.6	Function I2c_PrepareSlaveBuffer.....	43
3.10.3.7	Function I2c_StartListening.....	43
3.10.3.8	Function I2c_SyncTransmit.....	44
3.10.4	Structs Reference.....	45
3.10.4.1	Structure I2c_ConfigType.....	45
3.10.4.2	Structure I2c_DemConfigType.....	46

Section number	Title	Page
3.10.4.3	Structure I2C_HWUnitConfigType.....	47
3.10.4.4	Structure I2c_RequestType.....	48
3.10.5	Types Reference.....	49
3.10.5.1	Typedef I2c_AddressType.....	49
3.10.5.2	Typedef I2c_DataType.....	49
3.10.5.3	Typedef I2C_HWUnitType.....	49
3.10.6	Variables Reference.....	50
3.10.6.1	Variable I2c_aeChannelStatus.....	50
3.10.6.2	Variable I2c_pConfig.....	50
3.10.6.3	Variable I2c_pDemCfg.....	50
3.10.6.4	Variable I2c_u8DriverStatus.....	51
3.10.6.5	Variable I2c_DemConfig.....	51
3.11	Symbolic Names Disclaimer.....	51

Chapter 4

Tresos Configuration Plug-in

4.1	Configuration elements of I2C.....	53
4.2	Form IMPLEMENTATION_CONFIG_VARIANT.....	53
4.3	Form GeneralConfiguration.....	54
4.3.1	I2CDevErrorDetect (GeneralConfiguration).....	54
4.3.2	I2CDisableDemReportErrorStatus (GeneralConfiguration).....	55
4.3.3	I2CDmaUsed (GeneralConfiguration).....	55
4.3.4	I2CFlexIOUsed (GeneralConfiguration).....	56
4.3.5	I2CTimeoutDuration (GeneralConfiguration).....	56
4.3.6	I2CVersionInfoApi (GeneralConfiguration).....	57
4.3.7	I2CErrorNotification (GeneralConfiguration).....	57
4.3.8	I2CMasterTransmitCompleteNotification (GeneralConfiguration).....	58
4.3.9	I2CMasterReceiveCompleteNotification (GeneralConfiguration).....	58
4.3.10	I2CSlaveAddressMatchNotification (GeneralConfiguration).....	58
4.3.11	I2CSlaveTransmitCompleteNotification (GeneralConfiguration).....	59

Section number	Title	Page
4.3.12	I2CSlaveReceiveCompleteNotification (GeneralConfiguration).....	59
4.3.13	I2CEnableUserModeSupport (GeneralConfiguration).....	60
4.4	Form I2CGlobalConfig.....	60
4.4.1	Form I2CFlexIOModuleConfiguration.....	61
4.4.1.1	I2CFlexIOEnabledInDebug (I2CFlexIOModuleConfiguration).....	61
4.4.1.2	I2CFlexIOEnabledInDozeMode (I2CFlexIOModuleConfiguration).....	62
4.4.1.3	I2CFlexIOFastAccessMode (I2CFlexIOModuleConfiguration).....	62
4.4.1.4	I2CClockRef (I2CFlexIOModuleConfiguration).....	63
4.4.2	Form I2CDemEventParameterRefs.....	63
4.4.2.1	I2C_E_TIMEOUT_FAILURE (I2CDemEventParameterRefs).....	63
4.4.3	Form I2CChannel.....	63
4.4.3.1	I2CChannelId (I2CChannel).....	64
4.4.3.2	I2CHwChannel (I2CChannel).....	65
4.4.3.3	I2CMasterSlaveConfiguration (I2CChannel).....	65
4.4.3.4	I2CPinConfiguration (I2CChannel).....	65
4.4.3.5	Form I2cMasterConfiguration.....	66
4.4.3.5.1	I2cMasterEnabledInDebug (I2cMasterConfiguration).....	67
4.4.3.5.2	I2cMasterEnabledInDozeMode (I2cMasterConfiguration).....	67
4.4.3.5.3	I2cAsyncMethod (I2cMasterConfiguration).....	68
4.4.3.5.4	I2cPrescaler (I2cMasterConfiguration).....	68
4.4.3.5.5	I2cGlitchFilterSDA (I2cMasterConfiguration).....	69
4.4.3.5.6	I2cGlitchFilterSCL (I2cMasterConfiguration).....	69
4.4.3.5.7	I2cBusIdleTimeout (I2cMasterConfiguration).....	70
4.4.3.5.8	I2cPinLowTimeout (I2cMasterConfiguration).....	71
4.4.3.5.9	I2cDataValidDelay (I2cMasterConfiguration).....	71
4.4.3.5.10	I2cSetupHoldDelay (I2cMasterConfiguration).....	72
4.4.3.5.11	I2cClockHighPeriod (I2cMasterConfiguration).....	72
4.4.3.5.12	I2cClockLowPeriod (I2cMasterConfiguration).....	73
4.4.3.5.13	I2cBaudRate (I2cMasterConfiguration).....	74

Section number	Title	Page
4.4.3.5.14	I2cClockRef (I2cMasterConfiguration).....	74
4.4.3.5.15	I2cDmaTxChannelRef (I2cMasterConfiguration).....	74
4.4.3.5.16	I2cDmaRxChannelRef (I2cMasterConfiguration).....	75
4.4.3.5.17	Form I2CHighSpeedModeConfiguration.....	75
4.4.3.5.17.1	I2CDataValidDelay (I2CHighSpeedModeConfiguration).....	75
4.4.3.5.17.2	I2CSetupHoldDelay (I2CHighSpeedModeConfiguration).....	76
4.4.3.5.17.3	I2CClockHighPeriod (I2CHighSpeedModeConfiguration).....	76
4.4.3.5.17.4	I2CClockLowPeriod (I2CHighSpeedModeConfiguration).....	77
4.4.3.5.17.5	I2CHighSpeedBaudRate (I2CHighSpeedModeConfiguration).....	78
4.4.3.6	Form I2cSlaveConfiguration.....	78
4.4.3.6.1	I2cSlaveAddress (I2cSlaveConfiguration).....	79
4.4.3.6.2	I2cSlaveDisableFilterInDoze (I2cSlaveConfiguration).....	79
4.4.3.6.3	I2cSlaveFilterEnable (I2cSlaveConfiguration).....	80
4.4.3.6.4	I2cSlaveAckStall (I2cSlaveConfiguration).....	80
4.4.3.6.5	I2cSlaveTxStall (I2cSlaveConfiguration).....	81
4.4.3.6.6	I2cSlaveRxStall (I2cSlaveConfiguration).....	81
4.4.3.6.7	I2cSlaveAdrStall (I2cSlaveConfiguration).....	82
4.4.3.6.8	I2cGlitchFilterSDA (I2cSlaveConfiguration).....	82
4.4.3.6.9	I2cGlitchFilterSCL (I2cSlaveConfiguration).....	83
4.4.3.6.10	I2cDataValidDelay (I2cSlaveConfiguration).....	83
4.4.3.6.11	I2cClockHoldPeriod (I2cSlaveConfiguration).....	84
4.4.3.7	Form I2CFlexIOConfiguration.....	85
4.4.3.7.1	I2CAsyncMethod (I2CFlexIOConfiguration).....	85
4.4.3.7.2	I2CFlexIOSdaPin (I2CFlexIOConfiguration).....	86
4.4.3.7.3	I2CFlexIOSclPin (I2CFlexIOConfiguration).....	86
4.4.3.7.4	I2CFlexIOCompareValue (I2CFlexIOConfiguration).....	87
4.4.3.7.5	I2CBaudRate (I2CFlexIOConfiguration).....	87
4.4.3.7.6	I2CDmaTxChannelRef (I2CFlexIOConfiguration).....	88
4.4.3.7.7	I2CDmaRxChannelRef (I2CFlexIOConfiguration).....	88

Section number	Title	Page
4.5	Form CommonPublishedInformation.....	88
4.5.1	ArReleaseMajorVersion (CommonPublishedInformation).....	89
4.5.2	ArReleaseMinorVersion (CommonPublishedInformation).....	89
4.5.3	ArReleaseRevisionVersion (CommonPublishedInformation).....	90
4.5.4	ModuleId (CommonPublishedInformation).....	90
4.5.5	SwMajorVersion (CommonPublishedInformation).....	91
4.5.6	SwMinorVersion (CommonPublishedInformation).....	91
4.5.7	SwPatchVersion (CommonPublishedInformation).....	92
4.5.8	VendorApiInfix (CommonPublishedInformation).....	92
4.5.9	VendorId (CommonPublishedInformation).....	93



Chapter 1

Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	26/04/2019	NXP MCAL Team	Updated version for ASR 4.2.2S32K14XR1.0.2



Chapter 2

Introduction

This User Manual describes NXP Semiconductors AUTOSAR Inter-Integrated Circuit (I2c) for S32K14X .

2.1 Supported Derivatives

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

Table 2-1. S32K14X Derivatives

NXP Semiconductors	s32k148_lqfp144, s32k148_lqfp176, s32k148_mapbga100, s32k146_lqfp144, s32k146_lqfp100, s32k146_lqfp64, s32k146_mapbga100, s32k144_lqfp100, s32k144_lqfp64, s32k144_mapbga100, s32k142_lqfp100, s32k142_lqfp64, s32k118_lqfp48, s32k118_lqfp64, s32k142_lqfp48, s32k144_lqfp48, s32k148_lqfp100
--------------------	--

All of the above microcontroller devices are collectively named as S32K14X .

2.2 Overview

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

AUTOSAR

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.

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

2.3 About this Manual

This Technical Reference employs the following typographical conventions:

Boldface type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
ASM	Assembler
BSW	Basic Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
C/CPP	C and C++ Source Code
ECU	Electronic Control Unit
I2C	Inter-Integrated Circuit
ISR	Interrupt Service Routine
N/A	Not Applicable
VLE	Variable Length Encoding

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	Specification of I2c Driver	AUTOSAR Release 4.2.2
2	S32K14X Reference Manual	Reference Manual, Rev. 9, 9/2018
3	S32K142 Mask Set Errata for Mask 0N33V (0N33V)	30/11/2017
4	S32K144 Mask Set Errata for Mask 0N57U (0N57U)	30/11/2017
5	S32K146 Mask Set Errata for Mask 0N73V (0N73V)	30/11/2017
6	S32K148 Mask Set Errata for Mask 0N20V (0N20V)	25/10/2018
7	S32K118 Mask Set Errata for Mask 0N97V (0N97V)	07/01/2019

2.5.1 Reference for I2C Bus Specification

Table 2-4. I2C Bus Specification and user manual

#	Title	Version
1	I2C Bus Specification and user manual	http://www.nxp.com/documents/user_manual/UM10204.pdf

Chapter 3 Driver

3.1 Requirements

I2c is a Complex Device Driver (CDD), so there are no AUTOSAR requirements regarding this module. It has vendor-specific requirements and implementation.

3.2 Driver Design Summary

The I2c driver is implemented as an Autosar complex device driver. It uses the LPI2c and FlexIO hardware peripherals which provides support for implementing the I2c protocol. The I2c driver implements both master and slave mode for LPI2c channels and only master for FlexIO channels.

The driver offers a hardware independent API to the upper layer that can be used to configure the I2c and initiate synchronous and asynchronous data transfers. Asynchronous transfer for a master channel may use either interrupts or DMA. The slave operates only in asynchronous mode.

Hardware and software settings can be configured using an Autosar standard configuration tool. The information required for an I2c data transfer will be configured in a data structure that will be sent as parameter to the API of the driver.

The driver reports errors to the error manager as defined in AUTOSAR.

3.3 Hardware Resources

The hardware configured by the I2c driver is the same between derivatives.

Physical I2c Channels:LPI2C_0, LPI2C_1, FLEXIO_0_CH_0_1, FLEXIO_0_CH_2_3

Note: In EB tresos, Physical I2c Channel has selected by I2cHwChannel.

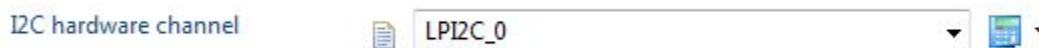


Figure 3-1. Physical I2c Channel has selected by I2cHwChannel in EB tresos

The LPI2C_0, LPI2C_1 use the pins correspondingly with naming is LPI2c0, LPI2c1 module.

The FLEXIO_0_CH_0_1, FLEXIO_0_CH_2_3 use the pins correspondingly with naming is FXIO module.

For example with the chip S32K144:

The pins can find in the file

"S32K144_IO_Signal_Description_Input_Multiplexing.xlsx" from attach files to Reference manual.

LPI2C_0 can be found in the xlsx file with naming is LPI2c0. And the Pin-Muxing is:

PTA2	PCR_PTA2	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTA2	PTA	Port A I/O	I/O
		0000_0010	FTM3_CH0	FTM3	FTM Channel	I/O
		0000_0011	LPI2C0_SDA	LPI2C0	LPI2C Data I/O	I/O
		0000_0100	EWM_OUT_b	EWM	External Watchdog Monitor Output	O
		0000_0101	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
		0000_0110	LPUART0_RX	LPUART0	Receive	I
	-	-	ADC1_SE0	ADC1	ADC Single Ended Input	I
PTA3	PCR_PTA3	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTA3	PTA	Port A I/O	I/O
		0000_0010	FTM3_CH1	FTM3	FTM Channel	I/O
		0000_0011	LPI2C0_SCL	LPI2C0	LPI2C Clock I/O	I/O
		0000_0100	EWM_IN	EWM	External Watchdog Monitor Input	I
		0000_0101	FXIO_D5	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
		0000_0110	LPUART0_TX	LPUART0	Transmit	I/O
	-	-	ADC1_SE1	ADC1	ADC Single Ended Input	I

Figure 3-2. IO Signal Description for LPI2C_0

FLEXIO can be found in the xlsx file with naming is FXIO_Dx, with 'x' is 0..7. In EB tresos, the pins of FLEXIO has selected by I2cFlexIOSdaPin and I2cFlexIOSclPin.



Figure 3-3. The pins of FLEXIO has selected by I2cFlexIOSdaPin and I2cFlexIOSclPin in EB tresos

And the Pin-Muxing is:

PTD0	PCR_PTD0	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTD0	PTD	Port D I/O	I/O
		0000_0010	FTM0_CH2	FTM0	FTM Channel	I/O
		0000_0011	LPSP1_SCK	LPSP1	LPSP1 Serial Clock I/O	I/O
		0000_0100	FTM2_CH0	FTM2	FTM Channel	I/O
		0000_0110	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
		0000_0111	TRGMUX_OUT1	TRGMUX	Trigger Mux Output	O
PTD1	PCR_PTD1	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTD1	PTD	Port D I/O	I/O
		0000_0010	FTM0_CH3	FTM0	FTM Channel	I/O
		0000_0011	LPSP1_SIN	LPSP1	LPSP1 Serial Data Input	I/O
		0000_0100	FTM2_CH1	FTM2	FTM Channel	I/O
		0000_0110	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
		0000_0111	TRGMUX_OUT2	TRGMUX	Trigger Mux Output	O
PTD2	PCR_PTD2	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0001	PTD2	PTD	Port D I/O	I/O
		0000_0010	FTM3_CH4	FTM3	FTM Channel	I/O
		0000_0011	LPSP1_SOUT	LPSP1	LPSP1 Serial Data Output	I/O
		0000_0100	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
		0000_0101	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
		0000_0110	TRGMUX_IN5	TRGMUX	Trigger Mux Input	I
	-	-	ADC1_SE2	ADC1	ADC Single Ended Input	I

Figure 3-4. IO Signal Description for FLEXIO

3.4 Deviation from Requirements

None

3.5 Driver limitations

- For Master mode, the driver does not support:

- Automatic generate STOP condition when the transmit FIFO is empty.
- Host request input can be used to control the start time of an I2c bus transfer.
- Flexible receive data match can generate interrupt on data match and/or discard unwanted data.

- For Slave mode, the driver does not support:

- SMBus alert and general call address.
- Transmit data with DMA requests.
- Receive data with DMA requests.
- Lengthy "I2C_SLAVE_TRANSMIT_COMPLETE_NOTIFICATION" or "I2C_SLAVE_RECEIVE_COMPLETE_NOTIFICATION" callbacks. These callbacks should be kept relatively short enough (where "short enough" is a quantity

that depends on the bitrate). Otherwise, the Slave will not generate the 9th bit (ACK/NACK).

3.6 Driver usage and configuration tips

This chapter is showcasing how certain features need to be configured/enabled.

3.6.1 Generic Master Flow

This section describes how the driver can be used in order to function as a Master.

3.6.1.1 Generic Master Configuration

```
/* Define the ID of the channel configured as a Master. */
/* Shall correspond to "I2cChannelId" */
uint8 masterChannel = 0x0;
/* Define an I2c request structure. */
I2c_RequestType request;
/* Define the buffer used for sending/receiving data. */
uint8 masterBuffer[MASTER_BUFFER_SIZE] = {1, 2, 3, ...};

/* Initialize the I2c Driver */

#if (I2C_PRECOMPILE_SUPPORT == STD_ON)
    I2c_Init(NULL_PTR);
#endif /* (I2C_PRECOMPILE_SUPPORT == STD_ON) */

#if (I2C_PRECOMPILE_SUPPORT == STD_OFF)
    /* The configuration structure has type "I2c_ConfigType". */
    I2c_Init(Identifier_Of_Generated_Configuration_Structure);
#endif /* (I2C_PRECOMPILE_SUPPORT == STD_OFF) */

/* Configure the I2c request structure with the desired values. */

/* The address of the slave in 8-bit format. */
/* (i.e. if 7-bit address is 0x50 => 8-bit address is (0x50 LSH 1) = 0xA0) */
request.SlaveAddress = 0xA0;

/* TRUE if the "SlaveAddress" field contains a 10-bit address. FALSE otherwise. */
request.b10BitsSlaveAddressSize = FALSE;

/* TRUE if the transfer shall use HighSpeed mode. FALSE otherwise. */
request.bHighSpeedMode = FALSE;

/* TRUE if the Slave is expected to send a NACK when addressed. FALSE otherwise. */
request.bExpectNack = FALSE;

/* The buffer used during the transfer. */
request.pDataBuffer = masterBuffer;

/* The number of bytes to be sent/received. Shall be less than or */
/* equal to MASTER_BUFFER_SIZE. */
```

```

request.u8BufferSize = 5;

/* The direction of the transfer (I2C_SEND_DATA for Master -> Slave and */
/* I2C_RECEIVE_DATA for Slave -> Master). */
request.eDataDirection = I2C_SEND_DATA;

```

3.6.1.2 Synchronous Master Transfer

```

/* Define the variable which will hold the return value of the */
/* "I2c_SyncTransmit" function. */
Std_ReturnType sendStatus;

/* Blocks until the transfer is completed or an error occurs. */
/* Returns I2C_CH_FINISHED if the transfer has successfully finished. */
/* Returns I2C_CH_ERROR_PRESENT if an error occurred. */
sendStatus = I2c_SyncTransmit(masterChannel, &request);

```

3.6.1.3 Asynchronous Master Transfer With Polling

```

/* Define the variable which will hold the return value of the */
/* "I2C_ASYNCTransmit" function. */
Std_ReturnType sendStatus;
/* Define the variable which will hold the polled status of the channel */
/* returned by the "I2c_GetStatus" function. */
I2c_StatusType masterStatus;

/* Returns immediately after sending the slave address. */
/* Returns E_NOT_OK if a timeout has occurred. Returns E_OK otherwise. */
sendStatus = I2C_ASYNCTransmit(masterChannel, &request);

/* Poll until the transfer has successfully finished or an error has occurred. */
do
{
    masterStatus = I2c_GetStatus(masterChannel);
} while (
    (I2C_CH_FINISHED != masterStatus) &&
    (I2C_CH_ERROR_PRESENT != masterStatus)
);

```

3.6.1.4 Asynchronous Master Transfer With Callbacks

```

/* The callback function whose identifier will be placed in the driver's */
/* "I2cMasterTransmitCompleteNotification" and/or */
/* "I2cMasterReceiveCompleteNotification" xdm/epc fields. */
/* This function will be called every time a Master transmission/reception */
/* successfully completes. */
void TransferCompleteNotification(uint8 u8Channel, uint8 u8NumberOfBytes)
{
    /* Here goes your logic at the end of a Master transfer. */
}

/* Define the variable which will hold the return value of the */
/* "I2C_ASYNCTransmit" function. */

```

```

Std_ReturnType sendStatus;

/* Returns immediately after sending the slave address. */
/* Returns E_NOT_OK if a timeout has occurred. Returns E_OK otherwise. */
sendStatus = I2C_ASYNCTransmit(masterChannel, &request);

/* Now there is no need to poll with "I2c_GetStatus" anymore. */

```

3.6.2 Generic Slave Flow

This section describes how the driver can be used in order to function as a Slave.

3.6.2.1 Generic Slave Configuration

```

/* Define the ID of the channel configured as a Slave. */
/* Shall correspond to "I2cChannelId" */
uint8 slaveChannel = 0x0;
/* Define the buffer used for sending/receiving data. */
uint8 slaveBuffer[SLAVE_BUFFER_SIZE] = {0};

/* The callback function whose identifier will be placed in the driver's */
/* "I2cSlaveAddressMatchNotification" xdm/epc field. */
/* This function will be called every time a Slave will be addressed by a Master. */
void AddrMatchingNotification(uint8 u8Channel, I2c_DataDirectionType eDirection)
{
    /* Define the variable which will hold the return value of the */
    /* "I2c_PrepareSlaveBuffer" function. */
    /* Will be equal to E_NOT_OK when the driver is not initialized or the */
    /* channel is invalid (out of hardware id range or a Master channel). */
    /* Returns E_OK otherwise. */
    Std_ReturnType preparationStatus;

    /* Of course, NUM_BYTES_TO_SEND/RECEIVE must be less than or */
    /* equal to SLAVE_BUFFER_SIZE. */
    if (I2C_SEND_DATA == eDirection)
    {
        preparationStatus = I2c_PrepareSlaveBuffer(slaveChannel, NUM_BYTES_TO_SEND,
slaveBuffer);
    }
    else
    {
        preparationStatus = I2c_PrepareSlaveBuffer(slaveChannel,
NUM_BYTES_TO_RECEIVE, slaveBuffer);
    }

    /* Here goes your logic for when the Slave is addressed by a Master. */
}

```

3.6.2.2 Asynchronous Slave Transfer With Polling

```

/* Define the variable which will hold the return value of the */
/* "I2c_StartListening" function. */
Std_ReturnType listeningStatus;
/* Define the variable which will hold the polled status of the channel */

```

```

/* returned by the "I2c_GetStatus" function. */
I2c_StatusType slaveStatus;

/* Returns E_NOT_OK if the driver is not initialized or the channel is invalid */
/* (out of hardware id range or a Master channel). Returns E_OK otherwise. */
listeningStatus = I2c_StartListening(slaveChannel);

/* If (listeningStatus == E_OK), the Slave is available to be addressed by */
/* a Master anytime from now on. */

/* After the following Slave transfer completes, regardless of whether the */
/* Slave's channel status is I2C_CH_FINISHED or I2C_CH_ERROR_PRESENT, the */
/* Slave channel becomes again unavailable for addressing until */
/* "I2c_StartListening" is called. Therefore, "I2c_StartListening" must */
/* be called before every transfer. */

/* Poll until the transfer has successfully finished or an error has occurred. */
do
{
    slaveStatus = I2c_GetStatus(slaveChannel);
} while (
    (I2C_CH_FINISHED != slaveStatus) &&
    (I2C_CH_ERROR_PRESENT != slaveStatus)
);

```

3.6.2.3 Asynchronous Slave Transfer With Callbacks

```

/* The callback function whose identifier will be placed in the driver's */
/* "I2cSlaveTransmitCompleteNotification" and/or */
/* "I2cSlaveReceiveCompleteNotification" xdm/epc fields. */
/* This function will be called every time a Slave transmission/reception */
/* successfully completes. */
void TransferCompleteNotification(uint8 u8Channel, uint8 u8NumberOfBytes)
{
    /* Here goes your logic at the end of a Slave transfer. */
}

/* Define the variable which will hold the return value of the */
/* "I2c_StartListening" function. */
Std_ReturnType listeningStatus;

/* Returns E_NOT_OK if the driver is not initialized or the channel is invalid */
/* (out of hardware id range or a Master channel). Returns E_OK otherwise. */
listeningStatus = I2c_StartListening(slaveChannel);

/* If (listeningStatus == E_OK), the Slave is available to be addressed by */
/* a Master anytime from now on. */

/* After the following Slave transfer completes, regardless of whether the */
/* Slave's channel status is I2C_CH_FINISHED or I2C_CH_ERROR_PRESENT, the */
/* Slave channel becomes again unavailable for addressing until */
/* "I2c_StartListening" is called. Therefore, "I2c_StartListening" must */
/* be called before every transfer. */

/* If there is no external logic to be done between two consecutive Slave */
/* transfers, then the call to "I2c_StartListening" can be made in the */
/* "TransferCompleteNotification" callback. This way, the Slave will */
/* immediately become available for another transfer. */

/* Now there is no need to poll with "I2c_GetStatus" anymore. */

```

3.6.2.4 Controlling RX ACK/NACK Bit

```

/* Warning: It is mandatory to configure the "I2cPinConfiguration" field */
/* of the Slave channel as "PINCFG_2PIN_PUSH_PULL" so that it is able */
/* to perform clock stretching (i.e. delay the 9th bit of a frame). */

/* Warning: The below notification shall have a minimal duration. */
/* The application code within must be optimized as much as possible. */

/* The callout function whose identifier will be placed in the driver's */
/* "I2cSlaveByteReceiveNotification" xdm/epc field. */
/* This function will be called every time a Slave receives a new byte */
/* (i.e. after the 8th bit and before the 9th bit of a frame) in order */
/* to determine whether it shall send an ACK or a NACK on the bus. */
boolean ByteReceiveNotification(uint8 u8Channel, uint8 u8ByteIndex)
{
    /* Use slaveBuffer[u8ByteIndex] to retrieve the newly received byte */

    /* Here goes your logic that decides between ACK/NACK */

    /* Possible return values: */
    /* TRUE  = Send ACK */
    /* FALSE = Send NACK */
}

```

3.7 Runtime Errors

The driver generates the following DEM errors at runtime.

Table 3-1. Runtime Errors

Function	Error Code	Condition triggering the error
I2c_SyncTransmit	I2C_E_TIMEOUT_FAILURE	The error is reported if the I2c bus is busy when trying to send the slave address for a period of time larger than the configured timeout. The error is also reported if the transmit FIFO is full when trying to send the next byte and also after a stop signal is sent by the master if the stop detected flag is not set in the hardware for a period of time larger than the configured timeout. When receiving bytes blocking, the error is reported if no byte is received in a time period larger than the configured timeout.
I2C_ASYNCTransmit	I2C_E_TIMEOUT_FAILURE	The error is reported if the I2c bus is busy when trying to send the slave address for a period of time larger than the configured timeout. The error is also reported after a stop signal is sent by the master if the stop detected flag is not set in the hardware for a period of time larger than the configured timeout. If DMA is used, this error is reported if the transmit FIFO is full when the master tries to send the stop signal at the end of

Table 3-1. Runtime Errors

Function	Error Code	Condition triggering the error
		the transmission, for a period of time larger than the configured timeout.

3.8 Development Errors

If development errors are enabled (I2C_DEV_ERROR_DETECT is STD_ON) the following DET errors will be raised:

Table 3-2. Development Errors

Function	Error Code	Condition triggering the error
I2c_Init	I2C_E_ALREADY_INITIALIZED	API is called when the driver is already initialized
I2c_Init	I2C_E_INVALID_POINTER	API is called with a NULL pointer as parameter
I2c_DeInit	I2C_E_UNINIT	API is called when driver is not initialized
I2c_SyncTransmit	I2C_E_UNINIT	API is called when driver is not initialized
I2c_SyncTransmit	I2C_E_INVALID_CHANNEL	API is called with an invalid channel as parameter
I2c_SyncTransmit	I2C_E_INVALID_POINTER	API is called with a NULL pointer as parameter
I2c_SyncTransmit	I2C_E_INVALID_BUFFER_SIZE	API is called with an invalid number of bytes that exceeds the maximum number supported by the channel.
I2C_ASYNCTransmit	I2C_E_UNINIT	API is called when driver is not initialized
I2C_ASYNCTransmit	I2C_E_INVALID_CHANNEL	API is called with an invalid channel as parameter
I2C_ASYNCTransmit	I2C_E_INVALID_POINTER	API is called with a NULL pointer as parameter
I2C_ASYNCTransmit	I2C_E_INVALID_BUFFER_SIZE	API is called with an invalid number of bytes that exceeds the maximum number supported by the channel.
I2c_PrepareSlaveBuffer	I2C_E_UNINIT	API is called when driver is not initialized
I2c_PrepareSlaveBuffer	I2C_E_INVALID_CHANNEL	API is called with an invalid channel as parameter
I2c_PrepareSlaveBuffer	I2C_E_INVALID_POINTER	API is called with a NULL pointer as parameter
I2c_StartListening	I2C_E_UNINIT	API is called when driver is not initialized
I2c_StartListening	I2C_E_INVALID_CHANNEL	API is called with an invalid channel as parameter
I2c_GetStatus	I2C_E_UNINIT	API is called when driver is not initialized
I2c_GetStatus	I2C_E_INVALID_CHANNEL	API is called with an invalid channel as parameter
I2c_GetVersionInfo	I2C_E_PARAM_POINTER	API is called with a NULL pointer as parameter

3.9 Communication Errors

If the I2cErrorNotification is configured in the configuration, it can be called by the driver with the following error codes:

Table 3-3. Communication Errors

Error Code	Condition triggering the error
I2C_E_PIN_LOW_TIMEOUT	The error code is reported if the LPI2C_MSR[PLTF] flag is set.
I2C_E_FIFO_ERROR	The error code is reported if the LPI2C_MSR[FEF] flag is set.
I2C_E_ARBITRATION_LOST	The error code is reported if the LPI2C_MSR[ALF] flag is set or if SHIFTErr[RX_shifter_ID] is set and the TX shifter is not ready for new data.
I2C_E_UNEXPECTED_NACK	The error code is reported if the LPI2C_MSR[NDF] flag is set.
I2C_E_SLAVE_BIT_ERROR	The error code is reported if the LPI2C_SSR[BEF] flag is set.
I2C_E_SLAVE_FIFO_ERROR	The error code is reported if the LPI2C_SSR[FEF] flag is set.
I2C_E_TX_UNDERFLOW	The error code is reported if the SHIFTErr[TX_shifter_ID] flag is set.
I2C_E_RX_OVERFLOW	The error code is reported if the SHIFTErr[RX_shifter_ID] flag is set.

3.10 Software specification

The following sections contains driver software specifications.

3.10.1 Define Reference

Constants supported by the driver are as per AUTOSAR I2c Driver software specification Version 4.2 Rev0002 .

3.10.1.1 Define I2C_E_ALREADY_INITIALIZED

Initialization called when already initialized.

Details:

The I2C Driver module shall report the development error "I2C_E_ALREADY_INITIALIZED (0x04)", when initialization is called when the driver is already initialized.

Table 3-4. Define I2C_E_ALREADY_INITIALIZED Description

Name	I2C_E_ALREADY_INITIALIZED
Initializer	((uint8)0x04U)

3.10.1.2 Define I2C_E_INVALID_BUFFER_SIZE

Number of bytes is exceeded, if a limit exists for the channel.

Details:

The I2C Driver module shall report the development error "I2C_E_INVALID_BUFFER_SIZE (0x05)", when I2C_SyncTransmit or I2C_AsyncTransmit are called with a number of bytes that exceed the maximum number of bytes supported for that channel.

Table 3-5. Define I2C_E_INVALID_BUFFER_SIZE Description

Name	I2C_E_INVALID_BUFFER_SIZE
Initializer	((uint8)0x05U)

3.10.1.3 Define I2C_E_INVALID_CHANNEL

API service used with an invalid or inactive channel parameter.

Details:

The I2C Driver module shall report the development error "I2C_E_INVALID_CHANNEL (0x02)", when API Service used with an invalid or inactive channel parameter.

Table 3-6. Define I2C_E_INVALID_CHANNEL Description

Name	I2C_E_INVALID_CHANNEL
Initializer	((uint8)0x02U)

3.10.1.4 Define I2C_E_INVALID_POINTER

API service called with invalid configuration pointer.

Details:

The I2C Driver module shall report the development error "I2C_E_INVALID_POINTER (0x03)", when API Service is called with invalid configuration pointer.

Table 3-7. Define I2C_E_INVALID_POINTER Description

Name	I2C_E_INVALID_POINTER
Initializer	((uint8)0x03U)

3.10.1.5 Define I2C_E_UNINIT

API service used without module initialization.

Details:

The I2C Driver module shall report the development error "I2C_E_UNINIT (0x01)", when the API Service is used without module initialization.

Table 3-8. Define I2C_E_UNINIT Description

Name	I2C_E_UNINIT
Initializer	((uint8)0x01U)

3.10.1.6 Define I2C_INIT

I2C driver states.

Details:

The I2C_INIT state indicates that the I2C driver has been initialized, making each available channel ready for service.

Table 3-9. Define I2C_INIT Description

Name	I2C_INIT
Initializer	(0x02U)

3.10.1.7 Define I2C_MODULE_ID

Table 3-10. Define I2C_MODULE_ID Description

Name	I2C_MODULE_ID
Initializer	255

3.10.1.8 Define I2C_UNINIT

I2C driver states.

Details:

The state I2C_UNINIT means that the I2C module has not been initialized yet and cannot be used.

Table 3-11. Define I2C_UNINIT Description

Name	I2C_UNINIT
Initializer	(0x01U)

3.10.1.9 Define I2C_DEV_ERROR_DETECT

Switches the Development Error Detection and Notification ON or OFF.

Table 3-12. Define I2C_DEV_ERROR_DETECT Description

Name	I2C_DEV_ERROR_DETECT
Initializer	(STD_ON)

3.10.1.10 Define I2C_DISABLE_DEM_REPORT_ERROR_STATUS

Enable/Disable the API for reporting the Dem Error.

Table 3-13. Define I2C_DISABLE_DEM_REPORT_ERROR_STATUS Description

Name	I2C_DISABLE_DEM_REPORT_ERROR_STATUS
Initializer	(STD_OFF)

3.10.1.11 Define I2C_DMA_USED

DMA is used for at least one channel (STD_ON/STD_OFF).

Table 3-14. Define I2C_DMA_USED Description

Name	I2C_DMA_USED
Initializer	(STD_ON)

3.10.1.12 Define I2C_E_ARBITRATION_LOST

Table 3-15. Define I2C_E_ARBITRATION_LOST Description

Name	I2C_E_ARBITRATION_LOST
Initializer	((uint8)0x03U)

3.10.1.13 Define I2C_E_FIFO_ERROR

Table 3-16. Define I2C_E_FIFO_ERROR Description

Name	I2C_E_FIFO_ERROR
Initializer	((uint8)0x02U)

3.10.1.14 Define I2C_E_PIN_LOW_TIMEOUT

Table 3-17. Define I2C_E_PIN_LOW_TIMEOUT Description

Name	I2C_E_PIN_LOW_TIMEOUT
Initializer	((uint8)0x01U)

3.10.1.15 Define I2C_E_RX_OVERFLOW

Table 3-18. Define I2C_E_RX_OVERFLOW Description

Name	I2C_E_RX_OVERFLOW
Initializer	((uint8)0x06U)

3.10.1.16 Define I2C_E_SLAVE_BIT_ERROR

Table 3-19. Define I2C_E_SLAVE_BIT_ERROR Description

Name	I2C_E_SLAVE_BIT_ERROR
Initializer	((uint8)0x11U)

3.10.1.17 Define I2C_E_SLAVE_FIFO_ERROR

Table 3-20. Define I2C_E_SLAVE_FIFO_ERROR Description

Name	I2C_E_SLAVE_FIFO_ERROR
Initializer	((uint8)0x10U)

3.10.1.18 Define I2C_E_TX_UNDERFLOW

Table 3-21. Define I2C_E_TX_UNDERFLOW Description

Name	I2C_E_TX_UNDERFLOW
Initializer	((uint8)0x05U)

3.10.1.19 Define I2C_E_UNEXPECTED_NACK

Table 3-22. Define I2C_E_UNEXPECTED_NACK Description

Name	I2C_E_UNEXPECTED_NACK
Initializer	((uint8)0x04U)

3.10.1.20 Define I2C_FLEXIO_FIRST_CHANNEL_U8

This is the ID of the first FLEXIO channel.

Table 3-23. Define I2C_FLEXIO_FIRST_CHANNEL_U8 Description

Name	I2C_FLEXIO_FIRST_CHANNEL_U8
Initializer	(I2C_LPI2C_MAX_MODULES)

3.10.1.21 Define I2C_FLEXIO_MAX_CHANNELS

Total number of available hardware FLEXIO sub channels. Each FLEXIO module can support 2 I2C channels.

Table 3-24. Define I2C_FLEXIO_MAX_CHANNELS Description

Name	I2C_FLEXIO_MAX_CHANNELS
Initializer	(2U)

3.10.1.22 Define I2C_HW_MAX_MODULES

Table 3-25. Define I2C_HW_MAX_MODULES Description

Name	I2C_HW_MAX_MODULES
Initializer	The number of LPI2C controllers

3.10.1.23 Define I2C_LPI2C_MAX_MODULES

Total number of available hardware LPI2C channels.

Table 3-26. Define I2C_LPI2C_MAX_MODULES Description

Name	I2C_LPI2C_MAX_MODULES
Initializer	(1U)

3.10.1.24 Define I2C_ERROR_NOTIFICATION

The call out configured by the user for error notifications.

Violates: MISRA 2004 Advisory Rule 19.7 This is used to define user configurable callouts with parameters .

Violates: This is used to define user configurable callouts with parameters

Table 3-27. Define I2C_ERROR_NOTIFICATION Description

Name	I2C_ERROR_NOTIFICATION
Initializer	((Plugin_I2CErrorNotification)(u8Channel, u8ErrorCode))

3.10.1.25 Define I2C_MASTER_RECEIVE_COMPLETE_NOTIFICATION

The call out configured by the user for master reception completion.

Violates: MISRA 2004 Advisory Rule 19.7 This is used to define user configurable callouts with parameters .

Violates: This is used to define user configurable callouts with parameters

Table 3-28. Define I2C_MASTER_RECEIVE_COMPLETE_NOTIFICATION Description

Name	I2C_MASTER_RECEIVE_COMPLETE_NOTIFICATION
Initializer	((Plugin_I2CMasterReceiveCompleteNotification)(u8Channel, u8NumberOfBytes))

3.10.1.26 Define I2C_MASTER_TRANSMIT_COMPLETE_NOTIFICATION

The call out configured by the user for master transmission completion.

Violates: MISRA 2004 Advisory Rule 19.7 This is used to define user configurable callouts with parameters .

Violates: This is used to define user configurable callouts with parameters

Table 3-29. Define I2C_MASTER_TRANSMIT_COMPLETE_NOTIFICATION
Description

Name	I2C_MASTER_TRANSMIT_COMPLETE_NOTIFICATION
Initializer	((Plugin_I2CMasterTransmitCompleteNotification)(u8Channel, u8NumberOfBytes))

3.10.1.27 Define I2C_SLAVE_ADDR_MATCH_NOTIFICATION

The call out configured by the user for address matching notifications.

Violates: MISRA 2004 Advisory Rule 19.7 This is used to define user configurable callouts with parameters .

Violates: This is used to define user configurable callouts with parameters

Table 3-30. Define I2C_SLAVE_ADDR_MATCH_NOTIFICATION Description

Name	I2C_SLAVE_ADDR_MATCH_NOTIFICATION
Initializer	((Plugin_I2CSlaveAddressMatchNotification)(u8Channel, eDirection))

3.10.1.28 Define I2C_SLAVE_RECEIVE_COMPLETE_NOTIFICATION

The call out configured by the user for slave reception completion.

Violates: MISRA 2004 Advisory Rule 19.7 This is used to define user configurable callouts with parameters .

Violates: This is used to define user configurable callouts with parameters

Table 3-31. Define I2C_SLAVE_RECEIVE_COMPLETE_NOTIFICATION
Description

Name	I2C_SLAVE_RECEIVE_COMPLETE_NOTIFICATION
Initializer	((Plugin_I2CSlaveReceiveCompleteNotification)(u8Channel, u8NumberOfBytes))

3.10.1.29 Define I2C_SLAVE_TRANSMIT_COMPLETE_NOTIFICATION

The call out configured by the user for slave transmission completion.

Violates: MISRA 2004 Advisory Rule 19.7 This is used to define user configurable callouts with parameters .

Violates: This is used to define user configurable callouts with parameters

Table 3-32. Define I2C_SLAVE_TRANSMIT_COMPLETE_NOTIFICATION Description

Name	I2C_SLAVE_TRANSMIT_COMPLETE_NOTIFICATION
Initializer	((Plugin_I2CSlaveTransmitCompleteNotification)(u8Channel, u8NumberOfBytes))

3.10.1.30 Define I2C_PRECOMPILE_SUPPORT

Precompile Support On.

Details:

VARIANT-PRE-COMPILE: Only parameters with "Pre-compile time" configuration are allowed in this variant.

Table 3-33. Define I2C_PRECOMPILE_SUPPORT Description

Name	I2C_PRECOMPILE_SUPPORT
Initializer	(STD_ON)

3.10.1.31 Define I2C_TIMEOUT_LOOPS

Number of loops before returning I2C_E_TIMEOUT.

Table 3-34. Define I2C_TIMEOUT_LOOPS Description

Name	I2C_TIMEOUT_LOOPS
Initializer	The value configured in the plugin

3.10.1.32 Define I2C_VERSION_INFO_API

Support for version info API.

Details:

Switches the I2C_GetVersionInfo() API ON or OFF.

Table 3-35. Define I2C_VERSION_INFO_API Description

Name	I2C_VERSION_INFO_API
Initializer	(STD_ON)

3.10.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR I2c Driver software specification Version 4.2 Rev0002 .

3.10.2.1 Enumeration I2c_ApiFunctionIdType

API functions service IDs.

Details:

Service IDs of the I2c API.

Table 3-36. Enumeration I2c_ApiFunctionIdType Values

Name	Initializer	Description
I2C_INIT_ID	(uint8)0x00U	I2c_Init() ID.
I2C_DEINIT_ID	(uint8)0x01U	I2c_DeInit() ID.
I2C_SYNCTRANSMIT_ID	(uint8)0x02U	I2c_SyncTransmit() ID.
I2C_ASYNCTRANSMIT_ID	(uint8)0x03U	I2C_ASYNCTransmit() ID.
I2C_GETSTATUS_ID	(uint8)0x04U	I2c_GetStatus() ID.
I2C_PREPARESLAVEBUFFER_ID	(uint8)0x05U	I2c_PrepareSlaveBuffer() ID.
I2C_STARTLISTENING_ID	(uint8)0x06U	I2c_StartListening() ID.

Table continues on the next page...

Table 3-36. Enumeration I2c_ApiFunctionIdType Values (continued)

Name	Initializer	Description
I2C_GETVERSIONINFO_ID	(uint8)0x0AU	I2c_GetVersionInfo() ID.

3.10.2.2 Enumeration I2c_DataDirectionType

Definition of the type of activation or procession mechanism of an I2c hw unit.

Implements: I2c_DataDirectionType_enumeration

Table 3-37. Enumeration I2c_DataDirectionType Values

Name	Initializer	Description
I2C_SEND_DATA	0x0U	Used to send data to a slave.
I2C_RECEIVE_DATA	0x1U	Used to receive data from a slave.

3.10.2.3 Enumeration I2C_HwChannelType

Definition of the hardware channel type.

Table 3-38. Enumeration I2C_HwChannelType Values

Name	Initializer	Description
LPI2C_CHANNEL	0x0U	This is used for LPI2C channels.
FLEXIO_CHANNEL	0x1U	This is used for FlexIO channels.

3.10.2.4 Enumeration I2c_ApiFunctionIdType

API functions service IDs.

Details:

Service IDs of the I2c API.

Table 3-39. Enumeration I2c_ApiFunctionIdType Values

Name	Initializer	Description
I2C_INIT_ID	(uint8)0x00U	I2c_Init() ID.
I2C_DEINIT_ID	(uint8)0x01U	I2c_DeInit() ID.
I2C_SYNCTRANSMIT_ID	(uint8)0x02U	I2c_SyncTransmit() ID.
I2C_ASYNCTRANSMIT_ID	(uint8)0x03U	I2C_ASYNCTransmit() ID.
I2C_GETSTATUS_ID	(uint8)0x04U	I2c_GetStatus() ID.
I2C_PREPARESLAVEBUFFER_ID	(uint8)0x05U	I2c_PrepareSlaveBuffer() ID.
I2C_STARTLISTENING_ID	(uint8)0x06U	I2c_StartListening() ID.
I2C_GETVERSIONINFO_ID	(uint8)0x0AU	I2c_GetVersionInfo() ID.

3.10.2.5 Enumeration I2C_ASYNCChronousMethodType

Asynchronous method used.

Implements: I2C_ASYNCChronousMethodType_enumeration

Table 3-40. Enumeration I2C_ASYNCChronousMethodType Values

Name	Initializer	Description
I2C_INTERRUPT_MODE	0x0U	Asynchronous Mechanism using interrupts.
I2C_DMA_MODE	0x1U	Asynchronous Mechanism using DMA.

3.10.2.6 Enumeration I2C_MASTERSlaveModeType

Definition of the master/slave mode of an I2c hw unit.

Table 3-41. Enumeration I2C_MASTERSlaveModeType Values

Name	Initializer	Description
I2C_MASTER_MODE	0x0U	The channel is an I2c master.
I2C_SLAVE_MODE	0x1U	The channel is an I2c master.
I2C_MASTER_SLAVE_MODE	0x2U	The channel is both I2c master and I2c slave.

3.10.2.7 Enumeration I2c_ModeType

Definition of the type of a transmission.

Table 3-42. Enumeration I2c_ModeType Values

Name	Initializer	Description
I2C_ASYNC_MODE	0x0U	Asynchronous Mode.
I2c_SYNC_MODE	0x1U	Synchronous Mode.

3.10.2.8 Enumeration I2c_StatusType

Definition for different state and errors of Operation Status.

Implements: I2c_StatusType_enumeration

Table 3-43. Enumeration I2c_StatusType Values

Name	Initializer	Description
I2C_CH_IDLE	0U	Status Indication I2c channel is idle.
I2C_CH_SEND		Status Indication send operation is ongoing.
I2C_CH_RECEIVE		Status Indication receiving operation is ongoing.
I2C_CH_FINISHED		Status Indication operation is finished.
I2C_CH_ERROR_PRESENT		Status Indication an error is present.

3.10.3 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR I2c Driver software specification Version 4.2 Rev0002 .

3.10.3.1 Function I2C_ASYNCTransmit

Starts an asynchronous transmission on the I2c bus.

Details:

Sends the slave address and enables the interrupts that will send or receive data depending on the direction of the message if the asynchronous method selected for the channel is INTERRUPT or configures a DMA channel and starts a DMA transfer if DMA is selected as the asynchronous method.

Return: Std_ReturnType

Violates: Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x03. Synchronous, non re-entrant function.

Prototype: Std_ReturnType I2C_ASYNCTransmit(uint8 u8Channel, I2c_RequestType *pRequestPtr);

Table 3-44. I2C_ASYNCTransmit Arguments

Type	Name	Direction	Description
uint8	u8Channel	input	I2c channel to be addressed.
I2c_RequestType*	pRequestPtr	input	Pointer to data information to be used.

Table 3-45. I2C_ASYNCTransmit Return Values

Name	Description
E_NOT_OK	If the I2c Channel is not valid or I2c driver is not initialized or pRequestPtr is NULL or I2c Channel is in busy state.
E_OK	Otherwise.

3.10.3.2 Function I2c_DeInit

DeInitializes the I2c module.

Details:

This function performs software de initialization of I2c modules to reset values.

The service influences only the peripherals, which are allocated by static configuration and the runtime configuration set passed by the previous call of I2c_Init()

The driver needs to be initialized before calling I2c_DeInit(). Otherwise, the function I2c_DeInit shall raise the development error I2C_E_UNINIT and leave the desired de initialization functionality without any action.

Return: void

Violates: Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x01. Synchronous, non re-entrant function.

Prototype: void I2c_DeInit(void);

3.10.3.3 Function I2c_GetStatus

Gets the status of an I2c channel.

Details:

Gets the status of an I2c channel and checks for errors.

Return: I2c_StatusType

Violates: Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x04. Synchronous, non re-entrant function.

Prototype: I2c_StatusType I2c_GetStatus(uint8 u8Channel);

Table 3-46. I2c_GetStatus Arguments

Type	Name	Direction	Description
uint8	u8Channel	input	I2c channel to be addressed.

Table 3-47. I2c_GetStatus Return Values

Name	Description
I2C_CH_IDLE	If the I2c Channel is in default state.
I2C_CH_SEND	If the I2c Channel is busy sending data.
I2C_CH_RECEIVE	If the I2c Channel is busy receiving data.
I2C_CH_FINISHED	If the I2c Channel finished the last transmission (sending or receiving data) successfully with no errors.
I2C_CH_ERROR_PRESENT	If the I2c Channel encountered an error during the last transmission.

3.10.3.4 Function I2c_GetVersionInfo

Returns the version information of this module.

Details:

The version information includes:

- Two bytes for the Vendor ID
- Two bytes for the Module ID
- One byte for the Instance ID
- Three bytes version number. The numbering shall be vendor specific: it consists of:
 - The major, the minor and the patch version number of the module;
 - The AUTOSAR specification version number shall not be included. The AUTOSAR specification version number is checked during compile time and therefore not required in this API.

Return: void.

Implements: I2c_GetVersionInfo_Activity

Violates: Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x0A. Synchronous, non re-entrant function.

Prototype: void I2c_GetVersionInfo(Std_VersionInfoType *pVersionInfo);

Table 3-48. I2c_GetVersionInfo Arguments

Type	Name	Direction	Description
Std_VersionInfoType *	pVersionInfo	input, output	A pointer to a variable to store version info.

3.10.3.5 Function I2c_Init

Initializes the I2c module.

Details:

This function performs software initialization of I2c driver

- Maps logical channels to hardware channels
- Initializes all channels
- Sets driver state machine to I2c_INIT.

Return: void

Violates: Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x00. Synchronous, non re-entrant function.

Prototype: `void I2c_Init(const I2c_ConfigType *pConfig);`

Table 3-49. I2c_Init Arguments

Type	Name	Direction	Description
constI2c_ConfigType*	pConfig	input	Pointer to I2c driver configuration set.

3.10.3.6 Function I2c_PrepareSlaveBuffer

Prepare the RX or TX buffer for a slave channel.

Details:

Prepares a RX or TX buffer that will be used to receive data or send data when requested by the master.

Return: Std_ReturnType

Violates : Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x05. Synchronous, non re-entrant function.

Prototype: `Std_ReturnType I2c_PrepareSlaveBuffer(uint8 u8Channel, uint8 u8NumberOfBytes, I2c_DataType *pDataBuffer);`

Table 3-50. I2c_PrepareSlaveBuffer Arguments

Type	Name	Direction	Description
uint8	u8Channel	input	I2c channel to be addressed.
uint8	u8NumberOfBytes	input	Maximum number of bytes to be sent or received.
I2c_DataType*	pDataBuffer	input	Pointer to data buffer to be used.

Table 3-51. I2c_PrepareSlaveBuffer Return Values

Name	Description
E_NOT_OK	If the I2c Channel is not valid or I2c driver is not initialized or pDataBuffer is NULL or I2c Channel is a master channel or I2c Channel is in busy state.
E_OK	Otherwise.

3.10.3.7 Function I2c_StartListening

Makes a slave channel available for processing requests (addressings).

Details:

When called, the slave channel becomes available for starting incoming or outgoing transfers.

Return: Std_ReturnType

Violates: Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x06. Synchronous, non re-entrant function.

Prototype: Std_ReturnType I2c_StartListening(uint8 u8Channel);

Table 3-52. I2c_StartListening Arguments

Type	Name	Direction	Description
uint8	u8Channel	input	I2c channel to be addressed.

Table 3-53. I2c_StartListening Return Values

Name	Description
E_NOT_OK	If the I2c Channel is not valid or I2c driver is not initialized or I2c Channel is a master channel.
E_OK	Otherwise.

3.10.3.8 Function I2c_SyncTransmit

Sends or receives an I2c message blocking.

Details:

Sends the slave address and based on the direction of the message it sends or receives data by using a blocking mechanism.

Return: Std_ReturnType

Violates: Violates MISRA 2004 Required Rule 8.10, global declaration of function

Note: Service ID: 0x02. Synchronous, non re-entrant function.

Prototype: Std_ReturnType I2c_SyncTransmit(uint8 u8Channel, I2c_RequestType *pRequestPtr);

Table 3-54. I2c_SyncTransmit Arguments

Type	Name	Direction	Description
uint8	u8Channel	input	I2c channel to be addressed.
I2c_RequestType*	pRequestPtr	input	Pointer to data information to be used.

Table 3-55. I2c_SyncTransmit Return Values

Name	Description
E_NOT_OK	If the I2c Channel is not valid or I2c driver is not initialized or pRequestPtr is NULL or I2c Channel is in busy state.
E_OK	Otherwise.

3.10.4 Structs Reference

Data structures supported by the driver are as per AUTOSAR I2c Driver software specification Version 4.2 Rev0002 .

3.10.4.1 Structure I2c_ConfigType

This type contains initialization data.

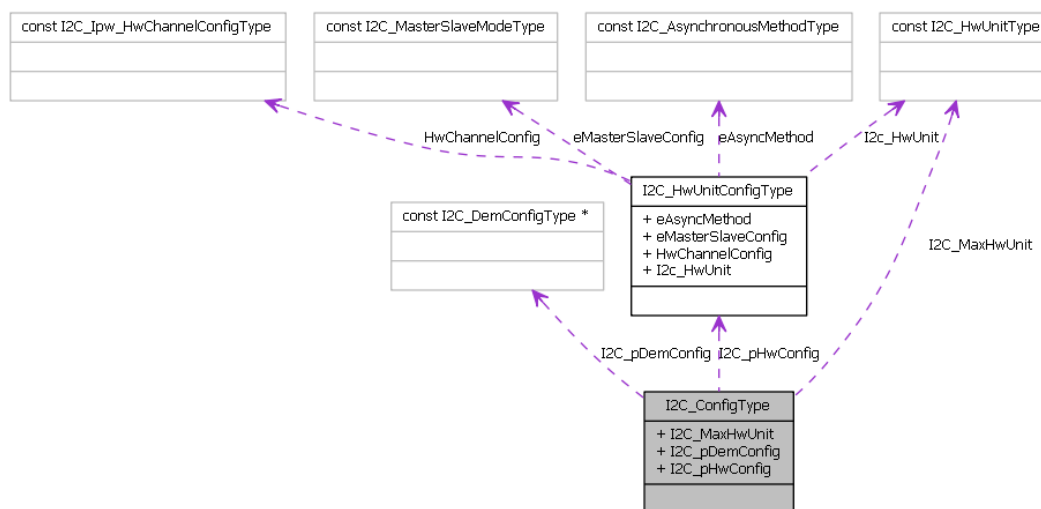


Figure 3-5. Struct I2c_ConfigType

Details:

This contains initialization data for the I2c driver. It shall contain:

- The number of I2c modules to be configured
- Dem error reporting configuration
- I2c dependent properties for used HW units

Declaration:

```
typedef struct
{
    constI2C_HWUnitType I2c_MaxHwUnit,
    constI2c_DemConfigType* I2c_pDemConfig,
    constI2C_HWUnitConfigType(*I2c_pHwConfig) []
} I2c_ConfigType;
```

Table 3-56. Structure I2c_ConfigType member description

Member	Description
I2c_MaxHwUnit	The maximum number of configured I2c Hw unit in the current configuration structure.
I2c_pDemConfig	DEM error reporting configuration.
I2c_pHwConfig	Pointer to I2c hardware unit configuration.

3.10.4.2 Structure I2c_DemConfigType

DEM error reporting configuration.

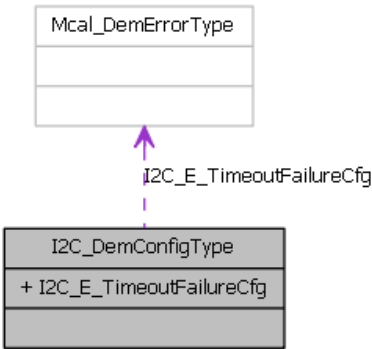


Figure 3-6. Struct I2c_DemConfigType

Details:

This structure contains information DEM error reporting

Declaration:

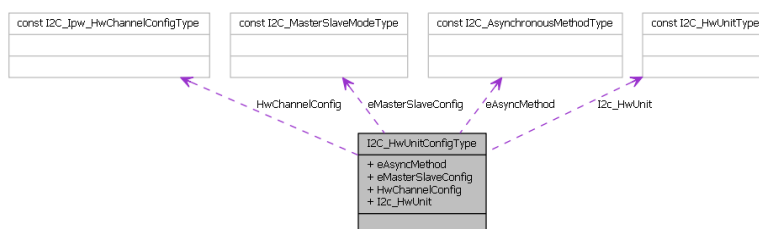
```
typedef struct
{
    Mcal_DemErrorType I2C_E_TimeoutFailureCfg
} I2c_DemConfigType;
```

Table 3-57. Structure I2c_DemConfigType member description

Member	Description
I2C_E_TimeoutFailureCfg	

3.10.4.3 Structure I2C_HWUnitConfigType

Structure that contains I2c Hw configuration.

**Figure 3-7. Struct I2C_HWUnitConfigType**

Details:

It contains the information/characteristics specific to one I2c Hw unit

Declaration:

```
typedef struct
{
    const I2C_ASYNCMethodType eAsyncMethod,
    const I2C_MASTERSlaveModeType eMasterSlaveConfig,
    const I2c_Ipw_HwChannelConfigType HwChannelConfig,
    const I2C_HWUnitType I2C_HWUnit
} I2C_HWUnitConfigType;
```

Table 3-58. Structure I2C_HWUnitConfigType member description

Member	Description
eAsyncMethod	Asynchronous method.
eMasterSlaveConfig	Master/slave mode configuration of the I2c Hw Unit.
HwChannelConfig	Structure containing the hardware specific configuration for the channel.
I2C_HWUnit	Numeric instance value of I2c Hw Unit.

3.10.4.4 Structure I2c_RequestType

Definition for Request Buffer. This is the structure which is passed to I2c_SyncTransmit or I2C_ASYNCTransmit function. This holds the necessary information required for the communication of I2c Hw with the Slave device.

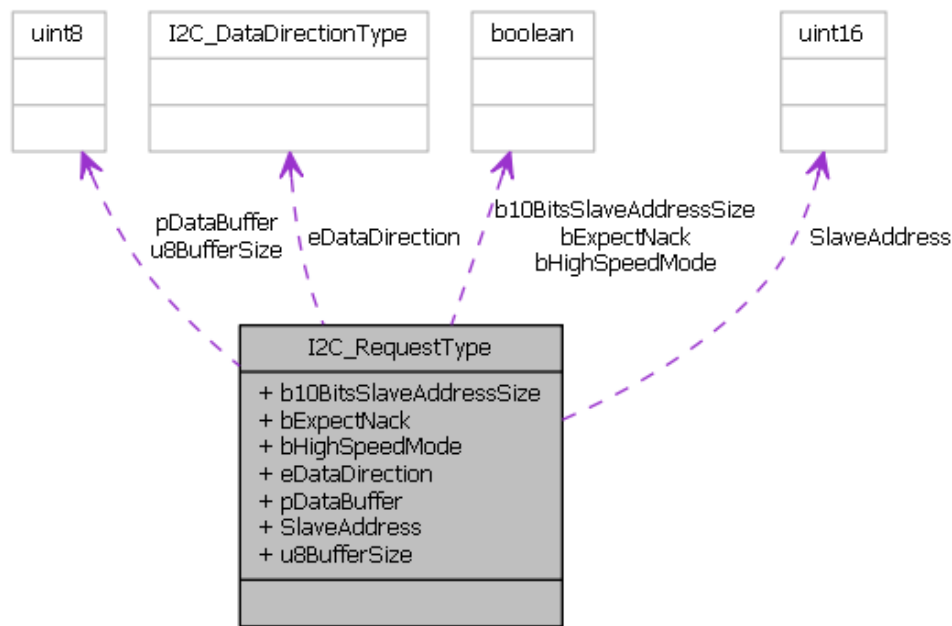


Figure 3-8. Struct I2c_RequestType

Declaration:

```
typedef struct
{
    boolean b10BitsSlaveAddressSize,
    boolean bExpectNack,
    boolean bHighSpeedMode,
    I2c_DataDirectionType eDataDirection,
    I2c_DataType* pDataBuffer,
    I2c_AddressType SlaveAddress,
    uint8 u8BufferSize
} I2c_RequestType;
```

Table 3-59. Structure I2c_RequestType member description

Member	Description
b10BitsSlaveAddressSize	This is true when the slave address is 10 bits, when false the address is on 7 bits.
bExpectNack	When this is true, NACK will be ignored during the address cycle.
bHighSpeedMode	If this is true, the data will be sent with high speed enabled (if hardware support exists).
eDataDirection	Direction of the data. Can be either Send or Receive.

Table continues on the next page...

Table 3-59. Structure I2c_RequestType member description (continued)

Member	Description
pDataBuffer	Buffer to Store or to transmit Serial data
u8BufferSize	Buffer Size : The number of bytes for reading or writing.
SlaveAddress	Slave Device Address.

3.10.5 Types Reference

Types supported by the driver are as per AUTOSAR I2c Driver software specification Version 4.2 Rev0002 .

3.10.5.1 Typedef I2c_AddressType

Type Address Value of Device and its register value.

Implements: I2c_AddressType_typedef

Type: uint16

3.10.5.2 Typedef I2c_DataType

Type Data to be sent or received.

Implements: I2c_DataType_typedef

Type: uint8

3.10.5.3 Typedef I2C_HWUnitType

This gives the numeric ID (hardware number) of an I2c hw Unit.

Implements: I2C_HWUnitType_typedef

Type: uint8

3.10.6 Variables Reference

Variables supported by the driver are as per AUTOSAR I2c Driver software specification Version 4.2 Rev0002 .

3.10.6.1 Variable I2c_aeChannelStatus

I2c channel status array.

Details:

The status of the I2c channels.

Declaration:

```
I2c_StatusType I2c_aeChannelStatus[I2C_HW_MAX_MODULES]
```

3.10.6.2 Variable I2c_pConfig

Global configuration pointer.

Details:

Pointer to the configuration structure.

Violates: include statements in a file should only be preceded by other preprocessor directives or comments.

Violates: Precautions shall be taken in order to prevent the contents of a header file being included twice.

Declaration:

```
const I2c_ConfigType* I2c_pConfig
```

3.10.6.3 Variable I2c_pDemCfg

DEM errors configuration pointer.

Details:

This is used to report DEM errors in the I2c driver.

Declaration:

```
const I2c_DemConfigType* I2c_pDemCfg
```

3.10.6.4 Variable I2c_u8DriverStatus

I2c driver status variable.

Details:

I2c driver state machine.

Violates: include statements in a file should only be preceded by other preprocessor directives or comments.

Violates: Precautions shall be taken in order to prevent the contents of a header file being included twice.

Declaration:

```
uint8 I2c_u8DriverStatus
```

3.10.6.5 Variable I2c_DemConfig

Violates: I2c_Cfg_c_REF_1 MISRA 2004 Required Rule 19.15, Repeated include file

Violates: I2c_Cfg_c_REF_2 MISRA 2004 Advisory Rule 19.1, only preprocessor statements and comments before 'include'

Declaration:

```
const I2c_DemConfigType I2c_DemConfig
```

3.11 Symbolic Names Disclaimer

All containers having the symbolic name tag set as true in the Autosar schema will generate defines like:

```
#define <Container_Short_Name> <Container_ID>
```

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

Chapter 4

Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the I2c Driver. The most of the parameters are described below.

4.1 Configuration elements of I2C

Included forms :

- IMPLEMENTATION_CONFIG_VARIANT
- GeneralConfiguration
- CommonPublishedInformation
- I2CGlobalConfig

4.2 Form IMPLEMENTATION_CONFIG_VARIANT

VariantPreCompile: Only precompile time configuration parameters. Only one set of parameters.

VariantPostBuild: Mix of precompile and postbuild time configuration parameters. Only one set of parameters.

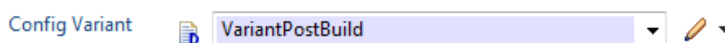


Figure 4-1. Tresos Plugin snapshot for IMPLEMENTATION_CONFIG_VARIANT form.

Table 4-1. Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description

Property	Value
Label	Config Variant
Type	ENUMERATION
Default	VariantPostBuild

Table continues on the next page...

Table 4-1. Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description (continued)

Property	Value
Range	VariantPostBuild VariantPreCompile

4.3 Form GeneralConfiguration

GeneralConfiguration

This container contains the global configuration parameters of the Non-Autosar I2C driver.

Note: Implementation Specific Parameter.

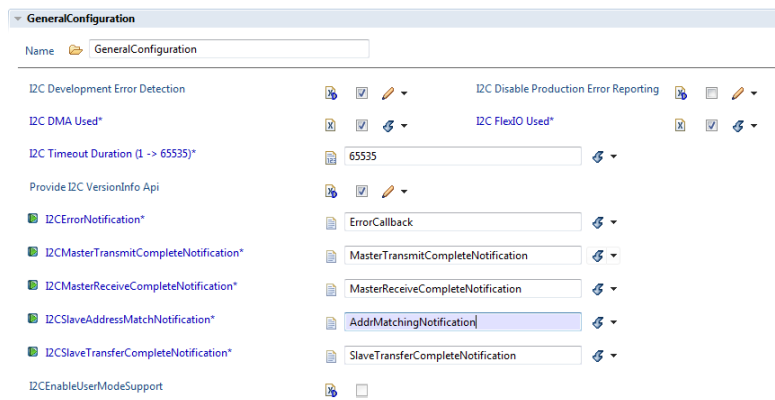


Figure 4-2. TRESOS Plugin snapshot for GeneralConfiguration form.

4.3.1 I2CDevErrorDetect (GeneralConfiguration)

I2CDevErrorDetect

Switches the Development Error Detection and Notification ON or OFF.

Note: Implementation Specific Parameter.

Table 4-2. Attribute I2CDevErrorDetect (GeneralConfiguration) detailed description

Property	Value
Label	I2C Development Error Detection

Table continues on the next page...

Table 4-2. Attribute I2CDevErrorDetect (GeneralConfiguration) detailed description (continued)

Property	Value
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.3.2 I2CDisableDemReportErrorStatus (GeneralConfiguration)

I2CDisableDemReportErrorStatus

Switches the Diagnostic Error Reporting and Notification OFF.

Note: Implementation Specific Parameter.

Table 4-3. Attribute I2CDisableDemReportErrorStatus (GeneralConfiguration) detailed description

Property	Value
Label	I2C Disable Production Error Reporting
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.3.3 I2CDmaUsed (GeneralConfiguration)

I2CDmaUsed

Check this in order to be able to use DMA in the I2C driver.

Leaving this unchecked will allow the I2C driver to compile with no dependencies from the Mcl driver.

Note: Implementation Specific Parameter.

Table 4-4. Attribute I2CDmaUsed (GeneralConfiguration) detailed description

Property	Value
Label	I2C DMA Used
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.3.4 I2CFlexIOUsed (GeneralConfiguration)

I2CFlexIOUsed

Check this in order to be able to use FlexIO channels.

Leaving this unchecked will allow the I2C driver to generate code without configuring the FlexIO module.

Note: Implementation Specific Parameter.

Table 4-5. Attribute I2CFlexIOUsed (GeneralConfiguration) detailed description

Property	Value
Label	I2C FlexIO Used
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.3.5 I2CTimeoutDuration (GeneralConfiguration)

Specifies the maximum number of loops for blocking function until a timeout is raised in short term wait loops

Note: Implementation Specific Parameter.

Table 4-6. Attribute I2CTimeoutDuration (GeneralConfiguration) detailed description

Property	Value
Label	I2C Timeout Duration

Table continues on the next page...

Table 4-6. Attribute I2CTimeoutDuration (GeneralConfiguration) detailed description (continued)

Property	Value
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	1000
Invalid	Range <=65535 >=1

4.3.6 I2CVersionInfoApi (GeneralConfiguration)

I2CVersionInfoApi

Switches the I2C_GetVersionInfo function ON or OFF.

Note: Implementation Specific Parameter.

Table 4-7. Attribute I2CVersionInfoApi (GeneralConfiguration) detailed description

Property	Value
Label	Provide I2C VersionInfo Api
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.3.7 I2CErrorNotification (GeneralConfiguration)

Function name of callout. This function will be called if an error is detected.

Note: Implementation Specific Parameter.

Table 4-8. Attribute I2CErrorNotification (GeneralConfiguration) detailed description

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.3.8 I2CMasterTransmitCompleteNotification (GeneralConfiguration)

Function name of callout. This function will be called when a data transmission completes.

(Only applicable in master mode!)

Note: Implementation Specific Parameter.

Table 4-9. Attribute I2CMasterTransmitCompleteNotification (GeneralConfiguration) detailed description

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.3.9 I2CMasterReceiveCompleteNotification (GeneralConfiguration)

Function name of callout. This function will be called when a data reception completes.

(Only applicable in master mode!)

Note: Implementation Specific Parameter.

Table 4-10. Attribute I2CMasterReceiveCompleteNotification (GeneralConfiguration) detailed description

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.3.10 I2CSlaveAddressMatchNotification (GeneralConfiguration)

Function name of callout. This function will be called if an address matching is detected.

Note: Implementation Specific Parameter.

Table 4-11. Attribute I2CSlaveAddressMatchNotification (GeneralConfiguration) detailed description

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.3.11 I2CSlaveTransmitCompleteNotification (GeneralConfiguration)

Function name of callout. This function will be called when a data transmission completes.

(Only applicable in slave mode!)

Note: Implementation Specific Parameter.

Table 4-12. Attribute I2CSlaveTransmitCompleteNotification (GeneralConfiguration) detailed description

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.3.12 I2CSlaveReceiveCompleteNotification (GeneralConfiguration)

Function name of callout. This function will be called when a data reception completes.

(Only applicable in slave mode!)

Note: Implementation Specific Parameter.

Table 4-13. Attribute I2CSlaveReceiveCompleteNotification (GeneralConfiguration) detailed description

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.3.13 I2CEnableUserModeSupport (GeneralConfiguration)

When this parameter is enabled, the I2C module will adapt to run from User Mode.

Note

Note: I2C module does not include registers protection. So, It is accessible to all registered in any public mode. I2C is not affected by this field.

Table 4-14. Attribute I2CEnableUserModeSupport (GeneralConfiguration) detailed description

Property	Value
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4 Form I2CGlobalConfig

This container contains the global configuration parameter of the I2C driver. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.

Included forms :

- [Form I2CFlexIOModuleConfiguration](#)
- [Form I2CDemEventParameterRefs](#)
- [Form I2CChannel](#)

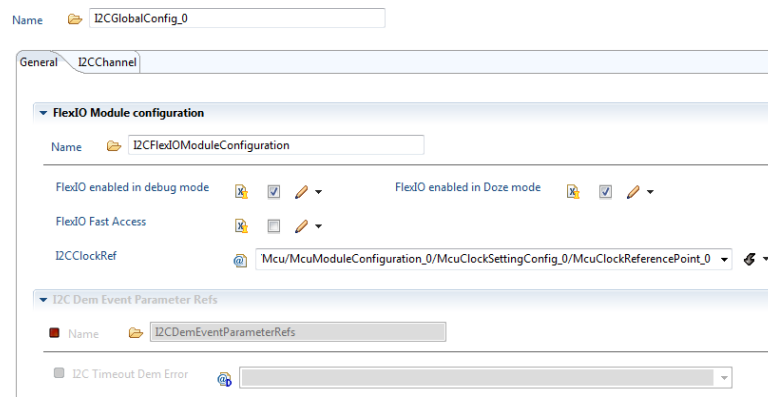


Figure 4-3. TRESOS Plugin snapshot for I2CGlobalConfig form.

4.4.1 Form I2CFlexIOModuleConfiguration

This container contains the configuration (parameters) of the Master Configuration.

Is included by form : [Form I2CGlobalConfig](#)

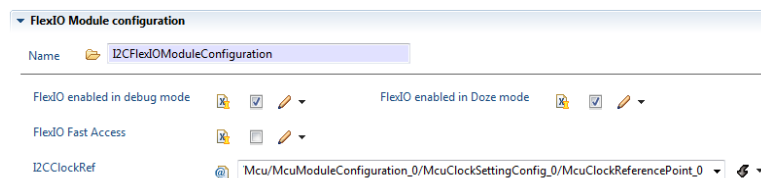


Figure 4-4. TRESOS Plugin snapshot for I2CFlexIOModuleConfiguration form.

4.4.1.1 I2CFlexIOEnabledInDebug (I2CFlexIOModuleConfiguration)

This configures FLEXIO_CTRL[DBGEN]

Check to enable the FlexIO in debug mode. Uncheck to disable it.

Note: Implementation Specific Parameter.

Table 4-15. Attribute I2CFlexIOEnabledInDebug (I2CFlexIOModuleConfiguration) detailed description

Property	Value
Label	FlexIO enabled in debug mode
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.1.2 I2CFlexIOEnabledInDozeMode (I2CFlexIOModuleConfiguration)

This configures FLEXIO_CTRL[DOZEN]

Check to enable the master in doze mode. Uncheck to disable it.

Note: Implementation Specific Parameter.

Table 4-16. Attribute I2CFlexIOEnabledInDozeMode (I2CFlexIOModuleConfiguration) detailed description

Property	Value
Label	FlexIO enabled in Doze mode
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.1.3 I2CFlexIOFastAccessMode (I2CFlexIOModuleConfiguration)

This configures FLEXIO_CTRL[FASTACC]

Enables fast register accesses to FlexIO registers, but requires the FlexIO clock to be at least twice the frequency of the bus clock.

Check to configure fast register accesses to FlexIO.

Uncheck to configure normal register accesses to FlexIO.

Note: Implementation Specific Parameter.

Table 4-17. Attribute I2CFlexIOFastAccessMode (I2CFlexIOModuleConfiguration) detailed description

Property	Value
Label	FlexIO Fast Access
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.1.4 I2CClockRef (I2CFlexIOModuleConfiguration)

Reference to the FlexIO clock source configuration, which is set in the MCU driver configuration.

Note: Implementation Specific Parameter.

Table 4-18. Attribute I2CClockRef (I2CFlexIOModuleConfiguration) detailed description

Property	Value
Type	REFERENCE
Origin	Custom

4.4.2 Form I2CDemEventParameterRefs

Is included by form : [Form I2CGlobalConfig](#)

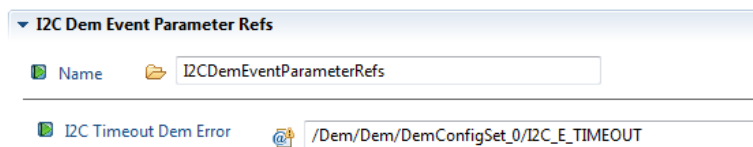


Figure 4-5. Tresos Plugin snapshot for I2CDemEventParameterRefs form.

4.4.2.1 I2C_E_TIMEOUT_FAILURE (I2CDemEventParameterRefs)

Table 4-19. Attribute I2C_E_TIMEOUT_FAILURE (I2CDemEventParameterRefs) detailed description

Property	Value
Label	I2C Timeout Dem Error
Type	SYMBOLIC-NAME-REFERENCE
Origin	Custom

4.4.3 Form I2CChannel

Note

This container contains the configuration (parameters) of the I2C Controller(s).

"User should use unique names for naming the I2C channels across different I2CGlobalConfig Sets."

Is included by form : [Form I2CGlobalConfig](#)

Included forms :

-
-
- [Form I2CFlexIOConfiguration](#)

I2CChannel

Name I2CChannel_0

General

I2C Channel ID (0 -> 4) 0

I2C hardware channel FLEXIO0_2_3

I2C master/slave configuration MASTER_MODE

I2C pin configuration PINCFG_2PIN_PUSH_PULL

▶ LPI2C Master configuration

▶ LPI2C Slave configuration

▶ FlexIO Master configuration

Figure 4-6. Tresos Plugin snapshot for I2CChannel form.

4.4.3.1 I2CChannelId (I2CChannel)

Identifies the I2C channel.

Note: Implementation Specific Parameter.

Table 4-20. Attribute I2CChannelId (I2CChannel) detailed description

Property	Value
Label	I2C Channel ID
Type	INTEGER
Origin	Custom
Symbolic Name	true
Invalid	Range >=0 <=4

4.4.3.2 I2CHwChannel (I2CChannel)

Selects the physical I2C Channel.

Note: Implementation Specific Parameter.

Table 4-21. Attribute I2CHwChannel (I2CChannel) detailed description

Property	Value
Label	I2C Hardware Channel
Type	ENUMERATION
Origin	Custom
Symbolic Name	false

4.4.3.3 I2CMasterSlaveConfiguration (I2CChannel)

Selects the master slave configuration.

Select whether the selected channel will be used as Master, Slave or both.

Note: Implementation Specific Parameter.

Table 4-22. Attribute I2CMasterSlaveConfiguration (I2CChannel) detailed description

Property	Value
Label	I2C Master/Slave Configuration
Type	ENUMERATION
Origin	Custom
Symbolic Name	false
Default	MASTER_MODE
Range	MASTER_MODE SLAVE_MODE

4.4.3.4 I2CPinConfiguration (I2CChannel)

Selects the pin configuration.

Configures the pin mode.

000 LPI2C configured for 2-pin open drain mode.

001 LPI2C configured for 2-pin output only mode (ultra-fast mode).

010 LPI2C configured for 2-pin push-pull mode.

Note: Implementation Specific Parameter.

Table 4-23. Attribute I2CPinConfiguration (I2CChannel) detailed description

Property	Value
Label	I2C pin configuration
Type	ENUMERATION
Origin	Custom
Symbolic Name	false
Default	PINCFG_2PIN_PUSH_PULL
Range	PINCFG_2PIN_OPEN_DRAIN PINCFG_2PIN_PUSH_PULL PINCFG_2PIN_OUTPUT_ONLY

4.4.3.5 Form I2cMasterConfiguration

This container contains the configuration (parameters) of the Master Configuration.

Is included by form : [Form I2CChannel](#)

Included forms :

- [Form I2CHighSpeedModeConfiguration](#)

Figure 4-7. Tressos Plugin snapshot for I2cMasterConfiguration form.

4.4.3.5.1 I2cMasterEnabledInDebug (I2cMasterConfiguration)

I2cMasterEnabledInDebug configures LPI2C_MCR[DBGEN]

Check to enable the master in debug mode. Uncheck to disable it.

Note: Implementation Specific Parameter.

Table 4-24. Attribute I2cMasterEnabledInDebug (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Master enabled in debug mode
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.3.5.2 I2cMasterEnabledInDozeMode (I2cMasterConfiguration)

I2cMasterEnabledInDozeMode configures LPI2C_MCR[DOZEN]

Check to enable the master in doze mode. Uncheck to disable it.

Note: Implementation Specific Parameter.

Table 4-25. Attribute I2cMasterEnabledInDozeMode (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Master enabled in Doze mode
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.3.5.3 I2cAsyncMethod (I2cMasterConfiguration)

Configures the asynchronous mechanism used by the 'AsyncTransmit' function (interrupts or DMA).

Note: Implementation Specific Parameter.

Table 4-26. Attribute I2cAsyncMethod (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Asynchronous Method
Type	ENUMERATION
Origin	Custom
Symbolic Name	false
Default	INTERRUPT
Range	INTERRUPT DMA

4.4.3.5.4 I2cPrescaler (I2cMasterConfiguration)

PRESALE: Configures LPI2C_MCFGR1[PRESALE]

Configures the clock prescaler used for all LPI2c master logic, except the digital glitch filters.

Note: Implementation Specific Parameter.

Table 4-27. Attribute I2cPrescaler (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Prescaler
Type	ENUMERATION
Origin	Custom

Table continues on the next page...

Table 4-27. Attribute I2cPrescaler (I2cMasterConfiguration) detailed description (continued)

Property	Value
Symbolic Name	false
Default	DIVIDE_BY_1
Range	DIVIDE_BY_1 DIVIDE_BY_2 DIVIDE_BY_4 DIVIDE_BY_8 DIVIDE_BY_16 DIVIDE_BY_32 DIVIDE_BY_64 DIVIDE_BY_128

4.4.3.5.5 I2cGlitchFilterSDA (I2cMasterConfiguration)

Glitch Filter SDA: Configures LPI2C_MCFGR2[FILTSDA]

Configures the I2c master digital glitch filters for SDA input, a configuration of 0 will disable the glitch filter.

Glitches equal to or less than FILTSDA cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSDA cycles and must be configured less than the minimum SCL low or high period.

The glitch filter cycle count is not affected by the PRESCALE configuration and is automatically bypassed in High Speed mode.

Note: Implementation Specific Parameter.

Table 4-28. Attribute I2cGlitchFilterSDA (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Glitch Filter SDA (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <=15 >=0

4.4.3.5.6 I2cGlitchFilterSCL (I2cMasterConfiguration)

Glitch Filter SCL: Configures LPI2C_MCFGR2[FILTSCL]

Configures the I2c master digital glitch filters for SCL input, a configuration of 0 will disable the glitch filter.

Glitches equal to or less than FILTSCL cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSCL cycles and must be configured less than the minimum SCL low or high period.

The glitch filter cycle count is not affected by the PRESCALE configuration and is automatically bypassed in High Speed mode.

Note: Implementation Specific Parameter.

Table 4-29. Attribute I2cGlitchFilterSCL (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Glitch Filter SCL (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range ≤15 ≥0

4.4.3.5.7 I2cBusIdleTimeout (I2cMasterConfiguration)

Bus Idle Timeout: Configures LPI2C_MCFGR2[BUSIDLE]

Configures the bus idle timeout period in clock cycles. If both SCL and SDA are high for longer than BUSIDLE cycles, then the I2c bus is assumed to be idle and the master can generate a START condition. When set to zero, this feature is disabled.

Note: Implementation Specific Parameter.

Table 4-30. Attribute I2cBusIdleTimeout (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Bus Idle Timeout (cycles)
Type	INTEGER

Table continues on the next page...

Table 4-30. Attribute I2cBusIdleTimeout (I2cMasterConfiguration) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <div><=4095</div> <div>>=0</div>

4.4.3.5.8 I2cPinLowTimeout (I2cMasterConfiguration)

Pin Low Timeout: Configures LPI2C_MCFGR3[PINLOW]

Configures the pin low timeout flag in clock cycles. If SCL and/or SDA is low for longer than (PINLOW * 256) cycles then PLTF is set. When set to zero, this feature is disabled.

Note: Implementation Specific Parameter.

Table 4-31. Attribute I2cPinLowTimeout (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Pin Low Timeout (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <div><=4095</div> <div>>=0</div>

4.4.3.5.9 I2cDataValidDelay (I2cMasterConfiguration)

Data Valid Delay: Configures LPI2C_MCCR0[DATAVD]

Minimum number of cycles (minus one) that is used as the data hold time for SDA. Must be configured less than the minimum SCL low period.

Note: Implementation Specific Parameter.

Table 4-32. Attribute I2cDataValidDelay (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Data Valid Delay (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range <=63 >=1

4.4.3.5.10 I2cSetupHoldDelay (I2cMasterConfiguration)

Setup Hold Delay: Configures LPI2C_MCCR0[SETHOLD]

Minimum number of cycles (minus one) that is used by the master as the setup and hold time for a (repeated) START condition and setup time for a STOP condition.

The setup time is extended by the time it takes to detect a rising edge on the external SCL pin. Ignoring any additional board delay due to external loading, this is equal to $(2 + \text{FILTSCL}) / 2^{\text{PRESCALE}}$ cycles.

Note: Implementation Specific Parameter.

Table 4-33. Attribute I2cSetupHoldDelay (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Setup Hold Delay (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range <=63 >=2

4.4.3.5.11 I2cClockHighPeriod (I2cMasterConfiguration)

Clock High Period: Configures LPI2C_MCCR0[CLKHI]

Minimum number of cycles (minus one) that the SCL clock is driven high by the master.

The SCL high time is extended by the time it takes to detect a rising edge on the external SCL pin.

Ignoring any additional board delay due to external loading, this is equal to $(2 + \text{FILTSCCL}) / 2^{\text{PRESCALE}}$ cycles.

Note: Implementation Specific Parameter.

Table 4-34. Attribute I2cClockHighPeriod (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Clock High Period (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range ≤ 63 ≥ 1

4.4.3.5.12 I2cClockLowPeriod (I2cMasterConfiguration)

Clock Low Period: Configures LPI2C_MCCR0[CLKLO]

Minimum number of cycles (minus one) that the SCL clock is driven low by the master.

This value is also used for the minimum bus free time between a STOP and a START condition.

Note: Implementation Specific Parameter.

Table 4-35. Attribute I2cClockLowPeriod (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Clock Low Period (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	3
Invalid	Range ≤ 63 ≥ 3

4.4.3.5.13 I2cBaudRate (I2cMasterConfiguration)

Calculated as $\text{Frequency} / (((\text{CLKLO} + \text{CLKHI} + 2) * 2^{\text{PRESCALER}}) + \text{ROUNDDOWN}((2 + \text{FILTSCL}) / 2^{\text{PRESCALER}}))$

The functional clock must be at least 8 times faster than the I2c bus bandwidth.

Note: Implementation Specific Parameter.

Table 4-36. Attribute I2cBaudRate (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c Baud Rate
Type	FLOAT
Origin	Custom
Symbolic Name	false
Invalid	Range ≥ 0.0 ≤ 6000000.0

4.4.3.5.14 I2cClockRef (I2cMasterConfiguration)

Reference to the I2c clock source configuration, which is set in the MCU driver configuration.

Note: Implementation Specific Parameter.

Table 4-37. Attribute I2cClockRef (I2cMasterConfiguration) detailed description

Property	Value
Type	REFERENCE
Origin	Custom

4.4.3.5.15 I2cDmaTxChannelRef (I2cMasterConfiguration)

Reference to the DMA TX channel, which is set in the Mcl driver configuration.

Note: Implementation Specific Parameter.

Table 4-38. Attribute I2cDmaTxChannelRef (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c TX DMA Channel
Type	REFERENCE
Origin	Custom

4.4.3.5.16 I2cDmaRxChannelRef (I2cMasterConfiguration)

Reference to the DMA RX channel, which is set in the Mcl driver configuration.

Note: Implementation Specific Parameter.

Table 4-39. Attribute I2cDmaRxChannelRef (I2cMasterConfiguration) detailed description

Property	Value
Label	I2c RX DMA Channel
Type	REFERENCE
Origin	Custom

4.4.3.5.17 Form I2CHighSpeedModeConfiguration

This container contains the configuration (parameters) of the Master Clock Configuration Register 1.

Is included by form :

Figure 4-8. Tresos Plugin snapshot for I2CHighSpeedModeConfiguration form.

4.4.3.5.17.1 I2CDataValidDelay (I2CHighSpeedModeConfiguration)

Data Valid Delay: Configures LPI2C_MCCR1[DATAVD]

Minimum number of cycles (minus one) that is used as the data hold time for SDA. Must be configured less than the minimum SCL low period.

Note: Implementation Specific Parameter.

Table 4-40. Attribute I2CDataValidDelay (I2CHighSpeedModeConfiguration) detailed description

Property	Value
Label	I2C Data Valid Delay (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range ≤ 63 ≥ 1

4.4.3.5.17.2 I2CSetupHoldDelay (I2CHighSpeedModeConfiguration)

Setup Hold Delay: Configures LPI2C_MCCR1[SETHOLD]

Minimum number of cycles (minus one) that is used by the master as the setup and hold time for a (repeated) START condition and setup time for a STOP condition.

The setup time is extended by the time it takes to detect a rising edge on the external SCL pin.

Ignoring any additional board delay due to external loading, this is equal to $(2 + \text{FILTSCl}) / 2^{\text{PRESCALE}}$ cycles.

Note: Implementation Specific Parameter.

Table 4-41. Attribute I2CSetupHoldDelay (I2CHighSpeedModeConfiguration) detailed description

Property	Value
Label	I2C Setup Hold Delay (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range ≤ 63 ≥ 2

4.4.3.5.17.3 I2CClockHighPeriod (I2CHighSpeedModeConfiguration)

Clock High Period: Configures LPI2C_MCCR1[CLKHI]

Minimum number of cycles (minus one) that the SCL clock is driven high by the master.

The SCL high time is extended by the time it takes to detect a rising edge on the external SCL pin.

Ignoring any additional board delay due to external loading, this is equal to $(2 + \text{FILTSCLE}) / 2^{\text{PRESCALE}}$ cycles.

Note: Implementation Specific Parameter.

Table 4-42. Attribute I2CClockHighPeriod (I2CHighSpeedModeConfiguration) detailed description

Property	Value
Label	I2C Clock High Period (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range ≤ 63 ≥ 1

4.4.3.5.17.4 I2CClockLowPeriod (I2CHighSpeedModeConfiguration)

Clock Low Period: Configures LPI2C_MCCR1[CLKLO]

Minimum number of cycles (minus one) that the SCL clock is driven low by the master.

This value is also used for the minimum bus free time between a STOP and a START condition.

Note: Implementation Specific Parameter.

Table 4-43. Attribute I2CClockLowPeriod (I2CHighSpeedModeConfiguration) detailed description

Property	Value
Label	I2C Clock Low Period (cycles)
Type	INTEGER

Table continues on the next page...

Table 4-43. Attribute I2CClockLowPeriod (I2CHighSpeedModeConfiguration) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	3
Invalid	Range <div><=63</div> <div>>=3</div>

4.4.3.5.17.5 I2CHighSpeedBaudRate (I2CHighSpeedModeConfiguration)

Calculated as $\text{Frequency} / (((\text{CLKLO} + \text{CLKHI} + 2) * 2^{\text{PRESCALER}}) + \text{ROUNDDOWN}((2 + \text{FILTSCL}) / 2^{\text{PRESCALER}}))$

The functional clock must be at least 8 times faster than the I2C bus bandwidth.

Note: Implementation Specific Parameter.

Table 4-44. Attribute I2CHighSpeedBaudRate (I2CHighSpeedModeConfiguration) detailed description

Property	Value
Label	I2C Baud Rate
Type	FLOAT
Origin	Custom
Symbolic Name	false
Invalid	Range <div>>=0.0</div> <div><=6000000.0</div>

4.4.3.6 Form I2cSlaveConfiguration

This container contains the configuration (parameters) of the Slave Channel.

Is included by form : [Form I2CChannel](#)

The screenshot shows the 'I2cSlaveConfiguration' form. At the top, there's a 'Name' field with a folder icon and the text 'I2cSlaveConfiguration'. Below this, the form is organized into several rows of configuration options:

- I2C Slave Address (0 -> 1023):** A text input field containing the value '40'.
- I2C Disable Slave Filter in Doze mode:** A checkbox that is currently unchecked.
- I2C Enable Slave Filter:** A checkbox that is currently checked.
- I2C Enable ACK SCL Stall:** A checkbox that is currently unchecked.
- I2C Enable TX Data SCL Stall:** A checkbox that is currently unchecked.
- I2C Enable RX Data SCL Stall:** A checkbox that is currently unchecked.
- I2C Enable Address SCL Stall:** A checkbox that is currently unchecked.
- I2C Glitch Filter SDA (cycles) (0 -> 15):** A text input field containing the value '0'.
- I2C Glitch Filter SCL (cycles) (0 -> 15):** A text input field containing the value '0'.
- I2C Data Valid Delay (cycles) (0 -> 63):** A text input field containing the value '0'.
- I2C Clock Hold Period (cycles) (0 -> 15):** A text input field containing the value '0'.

Each text input field has a small icon to its left (representing a document or settings) and a pencil icon to its right (representing edit).

Figure 4-9. Tresos Plugin snapshot for I2cSlaveConfiguration form.

4.4.3.6.1 I2cSlaveAddress (I2cSlaveConfiguration)

The address of the slave: Configures LPI2C_SAMR[ADDR0]

Configures the I2c slave address.

Note: Implementation Specific Parameter.

Table 4-45. Attribute I2cSlaveAddress (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Slave Address
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <div> <div><=1023</div> <div>>=0</div> </div>

4.4.3.6.2 I2cSlaveDisableFilterInDoze (I2cSlaveConfiguration)

I2cSlaveDisableFilterInDoze configures LPI2C_SCR[FILTDZ]

Check to disable the filter in Doze mode. Uncheck to have the filters enabled in Doze mode.

Note: Implementation Specific Parameter.

Table 4-46. Attribute I2cSlaveDisableFilterInDoze (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Disable Slave Filter in Doze mode
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.3.6.3 I2cSlaveFilterEnable (I2cSlaveConfiguration)

I2cSlaveFilterEnable configures LPI2C_SCR[FILTEN]

Check to enable the digital filter and output delay counter for slave mode.

Note: Implementation Specific Parameter.

Table 4-47. Attribute I2cSlaveFilterEnable (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Enable Slave Filter
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.3.6.4 I2cSlaveAckStall (I2cSlaveConfiguration)

I2cSlaveAckStall configures LPI2C_SCFGR1[ACKSTALL]

Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted.

Clock stretching occurs when transmitting the 9th bit and is therefore not compatible with high speed mode.

When ACKSTALL is enabled, there is no need to set either RXSTALL or ADRSTALL.

Note: Implementation Specific Parameter.

Table 4-48. Attribute I2cSlaveAckStall (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Enable ACK SCL Stall
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.3.6.5 I2cSlaveTxStall (I2cSlaveConfiguration)

I2cSlaveAckStall configures LPI2C_SCFGR1[TXSTALL]

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

Clock stretching occurs following the 9th bit and is therefore compatible with high speed mode.

Note: Implementation Specific Parameter.

Table 4-49. Attribute I2cSlaveTxStall (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Enable TX Data SCL Stall
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.3.6.6 I2cSlaveRxStall (I2cSlaveConfiguration)

I2cSlaveAckStall configures LPI2C_SCFGR1[RXSTALL]

Enables SCL clock stretching when the receive data flag is set during a slave-receive transfer.

Clock stretching occurs following the 9th bit and is therefore compatible with high speed mode.

Note: Implementation Specific Parameter.

Table 4-50. Attribute I2cSlaveRxStall (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Enable RX Data SCL Stall
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.3.6.7 I2cSlaveAdrStall (I2cSlaveConfiguration)

I2cSlaveAdrStall configures LPI2C_SCFGR1[ADRSTALL]

Enables SCL clock stretching when the address valid flag is asserted. Clock stretching only occurs following the 9th bit and is therefore compatible with high speed mode.

Note: Implementation Specific Parameter.

Table 4-51. Attribute I2cSlaveAdrStall (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Enable Address SCL Stall
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.3.6.8 I2cGlitchFilterSDA (I2cSlaveConfiguration)

Glitch Filter SDA: Configures LPI2C_MCFGR2[FILTSDA]

Configures the I2c master digital glitch filters for SDA input, a configuration of 0 will disable the glitch filter.

Glitches equal to or less than FILTSDA cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSDA cycles and must be configured less than the minimum SCL low or high period.

The glitch filter cycle count is not affected by the PRESCALE configuration and is automatically bypassed in High Speed mode.

Note: Implementation Specific Parameter.

Table 4-52. Attribute I2cGlitchFilterSDA (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Glitch Filter SDA (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <=15 >=0

4.4.3.6.9 I2cGlitchFilterSCL (I2cSlaveConfiguration)

Glitch Filter SCL: Configures LPI2C_MCFGR2[FILTSCCL]

Configures the I2c master digital glitch filters for SCL input, a configuration of 0 will disable the glitch filter.

Glitches equal to or less than FILTSCCL cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSCCL cycles and must be configured less than the minimum SCL low or high period.

The glitch filter cycle count is not affected by the PRESCALE configuration and is automatically bypassed in High Speed mode.

Note: Implementation Specific Parameter.

Table 4-53. Attribute I2cGlitchFilterSCL (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Glitch Filter SCL (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <=15 >=0

4.4.3.6.10 I2cDataValidDelay (I2cSlaveConfiguration)

Data Valid Delay: Configures LPI2C_SCFGR2[DATAVD]

Configures the SDA data valid delay time for the I2c slave equal to $FILTSCL + DATAVD + 3$ cycles.

This data valid delay must be configured to less than the minimum SCL low period.

The I2c slave data valid delay time is not affected by the PRESCALE configuration, and is disabled in high speed mode.

Note: Implementation Specific Parameter.

Table 4-54. Attribute I2cDataValidDelay (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Data Valid Delay (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range ≤ 63 ≥ 0

4.4.3.6.11 I2cClockHoldPeriod (I2cSlaveConfiguration)

Clock Hold Period: Configures LPI2C_SCFGR2[CLKHOLD]

Minimum number of cycles (minus one) that the SCL clock is driven high by the master.

Configures the minimum clock hold time for the I2c slave, when clock stretching is enabled.

The minimum hold time is equal to $CLKHOLD + 3$ cycles.

The I2c slave clock hold time is not affected by the PRESCALE configuration, and is disabled in high speed mode.

Note: Implementation Specific Parameter.

Table 4-55. Attribute I2cClockHoldPeriod (I2cSlaveConfiguration) detailed description

Property	Value
Label	I2c Clock Hold Period (cycles)
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <div> <div><=15</div> <div>>=0</div> </div>

4.4.3.7 Form I2CFlexIOConfiguration

This container contains the configuration (parameters) of the Master Configuration.

Is included by form : [Form I2CChannel](#)

Figure 4-10. Tresos Plugin snapshot for I2CFlexIOConfiguration form.

4.4.3.7.1 I2CAsyncMethod (I2CFlexIOConfiguration)

Configures if the AsyncTransmit function will use interrupts or DMA.

Note: Implementation Specific Parameter.

Table 4-56. Attribute I2CAsyncMethod (I2CFlexIOConfiguration) detailed description

Property	Value
Label	I2C Asynchronous Method
Type	ENUMERATION
Origin	Custom
Symbolic Name	false
Default	INTERRUPT

Table continues on the next page...

Table 4-56. Attribute I2CAsyncMethod (I2CFlexIOConfiguration) detailed description (continued)

Property	Value
Range	INTERRUPT DMA

4.4.3.7.2 I2CFlexIOSdaPin (I2CFlexIOConfiguration)

This configures FLEXIO_SHIFTCTL_a[PINSEL]

Selects which pin is used by the Shifter input or output. PINSEL=*i* will select the FXIO__{Di} pin.

Note: Implementation Specific Parameter.

Table 4-57. Attribute I2CFlexIOSdaPin (I2CFlexIOConfiguration) detailed description

Property	Value
Label	SDA Pin Select
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range <=7 >=0

4.4.3.7.3 I2CFlexIOSclPin (I2CFlexIOConfiguration)

This configures FLEXIO_TIMCTL_a[PINSEL]

Selects which pin is used by the Timer input or output. PINSEL=*i* will select the FXIO__{Di} pin.

Note: Implementation Specific Parameter.

Table 4-58. Attribute I2CFlexIOSclPin (I2CFlexIOConfiguration) detailed description

Property	Value
Label	SCL Pin Select
Type	INTEGER

Table continues on the next page...

Table 4-58. Attribute I2CFlexIOScIPin (I2CFlexIOConfiguration) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range ≤ 7 ≥ 0

4.4.3.7.4 I2CFlexIOCompareValue (I2CFlexIOConfiguration)

This configures FLEXIO_TIMCMPa[$\text{CMP}[7:0]$]

This is used to calculate the Baud rate of the I2C. The baud rate divider is equal to $(\text{CMP}[7:0] + 1) * 2$.

The divider will be $(\text{input_clock} + \text{desired_baud_rate}) \text{ div } (2 * \text{desired_baud_rate}) - 2$. The extra -1 is from the timer reset setting used for clock stretching. Round to nearest integer.

This must be manually inserted, and the baud rate will be calculated based on it.

Note: Implementation Specific Parameter.

Table 4-59. Attribute I2CFlexIOCompareValue (I2CFlexIOConfiguration) detailed description

Property	Value
Label	Timer Compare Value
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	8
Invalid	Range ≤ 255 ≥ 0

4.4.3.7.5 I2CBaudRate (I2CFlexIOConfiguration)

Considering Timer Compare Value = cmp and Frequency the input frequency of the FlexIO the baud rate is calculated as:

Frequency div ($2 * \text{cmp} + 3$)

Note: Implementation Specific Parameter.

Table 4-60. Attribute I2CBaudRate (I2CFlexIOConfiguration) detailed description

Property	Value
Label	I2C FlexIO Baud Rate
Type	FLOAT
Origin	Custom
Symbolic Name	false
Invalid	Range ≥ 0.0 ≤ 6000000.0

4.4.3.7.6 I2CDmaTxChannelRef (I2CFlexIOConfiguration)

Reference to the DMA TX channel, which is set in the Mcl driver configuration.

Note: Implementation Specific Parameter.

Table 4-61. Attribute I2CDmaTxChannelRef (I2CFlexIOConfiguration) detailed description

Property	Value
Label	FlexIO TX DMA Channel
Type	REFERENCE
Origin	Custom

4.4.3.7.7 I2CDmaRxChannelRef (I2CFlexIOConfiguration)

Reference to the DMA RX channel, which is set in the Mcl driver configuration.

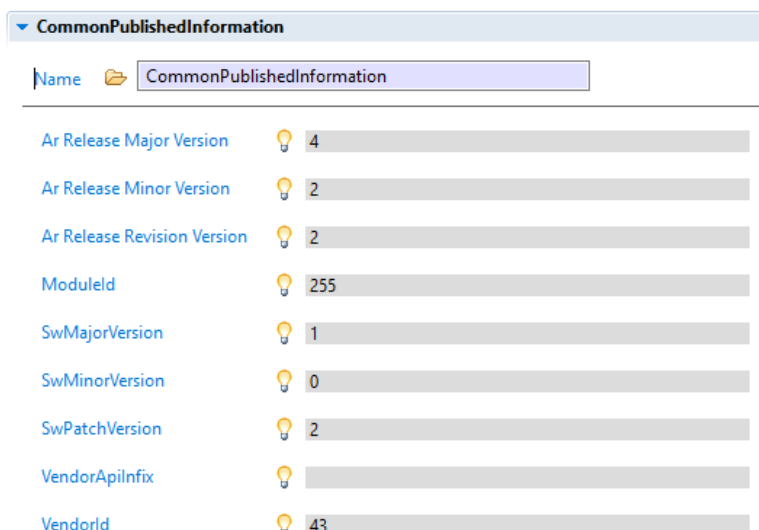
Note: Implementation Specific Parameter.

Table 4-62. Attribute I2CDmaRxChannelRef (I2CFlexIOConfiguration) detailed description

Property	Value
Label	FlexIO RX DMA Channel
Type	REFERENCE
Origin	Custom

4.5 Form CommonPublishedInformation

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



CommonPublishedInformation	
Name	Value
Ar Release Major Version	4
Ar Release Minor Version	2
Ar Release Revision Version	2
ModuleId	255
SwMajorVersion	1
SwMinorVersion	0
SwPatchVersion	2
VendorApiInfix	
VendorId	43

Figure 4-11. Tresos Plugin snapshot for CommonPublishedInformation form.

4.5.1 ArReleaseMajorVersion (CommonPublishedInformation)

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

Table 4-63. Attribute ArReleaseMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	4
Invalid	Range <div>>=4</div> <div><=4</div>

4.5.2 ArReleaseMinorVersion (CommonPublishedInformation)

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

Table 4-64. Attribute ArReleaseMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range <div>>=2</div> <div><=2</div>

4.5.3 ArReleaseRevisionVersion (CommonPublishedInformation)

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

Table 4-65. Attribute ArReleaseRevisionVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Release Revision Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range <div>>=2</div> <div><=2</div>

4.5.4 ModuleId (CommonPublishedInformation)

Module ID of this module from Module List.

Table 4-66. Attribute ModuleId (CommonPublishedInformation) detailed description

Property	Value
Label	Module Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	255
Invalid	Range <div> <div>>=255</div> <div><=255</div> </div>

4.5.5 SwMajorVersion (CommonPublishedInformation)

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

Table 4-67. Attribute SwMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range <div> <div>>=1</div> <div><=1</div> </div>

4.5.6 SwMinorVersion (CommonPublishedInformation)

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

Table 4-68. Attribute SwMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false

Table continues on the next page...

Table 4-68. Attribute SwMinorVersion (CommonPublishedInformation) detailed description (continued)

Property	Value
Default	0
Invalid	Range >=0 <=0

4.5.7 SwPatchVersion (CommonPublishedInformation)

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

Table 4-69. Attribute SwPatchVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Patch Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.5.8 VendorApiInfix (CommonPublishedInformation)

In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the name of APIs is extended by the VendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific name is generated as follows:

<ModuleName>_>VendorId>_<VendorApiInfix><Api name from SWS>. E.g. assuming that the VendorId of the implementor is 123 and the implementer chose a VendorApiInfix of "v11r456" a api name Can_Write defined in the SWS will translate to Can_123_v11r456Write. This parameter is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

Table 4-70. Attribute VendorApiInfix (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Api Infix
Type	STRING_LABEL
Origin	Custom
Symbolic Name	false
Default	
Enable	false

4.5.9 VendorId (CommonPublishedInformation)

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

Table 4-71. Attribute VendorId (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	43
Invalid	Range >=43 <=43

How to Reach Us:**Home Page:**nxp.com**Web Support:**nxp.com/support

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

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2019 NXP B.V.

Document Number UM212CASR4.2 Rev0002R1.0.2
Revision 1.0