

# User Manual

for S32K1\_M24X UART Driver

Document Number: UM2UARTASRR21-11 Rev0000R2.0.0 Rev. 1.0

<b>1 Revision History</b>	<b>2</b>
<b>2 Introduction</b>	<b>3</b>
2.1 Supported Derivatives	3
2.2 Overview	4
2.3 About This Manual	5
2.4 Acronyms and Definitions	6
2.5 Reference List	6
<b>3 Driver</b>	<b>8</b>
3.1 Requirements	8
3.2 Driver Design Summary	8
3.3 Hardware Resources	9
3.3.1 Physical Uart Channels:	9
3.3.2 Uart pins:	10
3.4 Deviations from Requirements	12
3.5 Driver Limitations	12
3.5.1 DMA limitation	12
3.5.2 Feature not supporting	13
3.6 Driver usage and configuration tips	13
3.6.1 How to use User notifications	13
3.6.2 How to configure Flexio Uart module	14
3.6.3 How to choose the appropriate baudrate	17
3.6.4 How to choose the appropriate timeout in general	17
3.6.5 How to enable Internal Loopback for self test.(Available only on LPUART channels)	18
3.6.6 How to set customized baudrate	18
3.6.7 How to configure DMA	20
3.6.8 How to configure Idle Interrupt	26
3.7 Runtime errors	27
3.8 Symbolic Names Disclaimer	28
<b>4 Tressos Configuration Plug-in</b>	<b>29</b>
4.1 Module Uart	30
4.2 Container GeneralConfiguration	31
4.3 Parameter UartDevErrorDetect	31
4.4 Parameter DisableUartRuntimeErrorDetect	32
4.5 Parameter UartMulticoreSupport	32
4.6 Parameter UartEnableUserModeSupport	33
4.7 Parameter UartTimeoutMethod	33
4.8 Parameter UartTimeoutDuration	34
4.9 Parameter UartDmaEnable	34

4.10 Parameter UartVersionInfoApi . . . . .	35
4.11 Parameter UartCallbackCapability . . . . .	35
4.12 Parameter UartCallback . . . . .	36
4.13 Reference UartEcucPartitionRef . . . . .	36
4.14 Container UartGlobalConfig . . . . .	37
4.15 Container UartChannel . . . . .	37
4.16 Parameter UartHwUsing . . . . .	38
4.17 Parameter UartChannelId . . . . .	38
4.18 Reference UartClockRef . . . . .	39
4.19 Reference UartChannelEcucPartitionRef . . . . .	39
4.20 Container DetailModuleConfiguration . . . . .	40
4.21 Parameter UartHwChannel . . . . .	40
4.22 Parameter DesireBaudrate . . . . .	41
4.23 Parameter CustomBaudrateMantissa . . . . .	41
4.24 Parameter CustomBaudrateDivisor . . . . .	42
4.25 Parameter CustomBaudrateValue . . . . .	42
4.26 Parameter UartInterruptDmaMethod . . . . .	43
4.27 Parameter UartParityType . . . . .	43
4.28 Parameter UartStopBitNumber . . . . .	44
4.29 Parameter UartWordLength . . . . .	44
4.30 Parameter UartInternalLoopbackEnable . . . . .	45
4.31 Parameter UartTimeoutEnable . . . . .	45
4.32 Reference UartDmaTxChannelRef . . . . .	46
4.33 Reference UartDmaRxChannelRef . . . . .	46
4.34 Container FlexioModuleConfiguration . . . . .	47
4.35 Parameter FlexioUartInterruptDmaMethod . . . . .	47
4.36 Parameter DesireBaudrate . . . . .	48
4.37 Parameter CustomTimerDecrement . . . . .	48
4.38 Parameter CustomBaudrateDivider . . . . .	49
4.39 Parameter CustomBaudrateValue . . . . .	49
4.40 Parameter bitCount . . . . .	50
4.41 Parameter driverDirection . . . . .	50
4.42 Reference UartHwChannelRef . . . . .	51
4.43 Reference FlexioDmaChannelRef . . . . .	51
4.44 Container CommonPublishedInformation . . . . .	52
4.45 Parameter ArReleaseMajorVersion . . . . .	52
4.46 Parameter ArReleaseMinorVersion . . . . .	53
4.47 Parameter ArReleaseRevisionVersion . . . . .	53
4.48 Parameter ModuleId . . . . .	54
4.49 Parameter SwMajorVersion . . . . .	54

4.50 Parameter SwMinorVersion . . . . .	55
4.51 Parameter SwPatchVersion . . . . .	55
4.52 Parameter VendorApiInfix . . . . .	56
4.53 Parameter VendorId . . . . .	57
<b>5 Module Index . . . . .</b>	<b>58</b>
5.1 Software Specification . . . . .	58
<b>6 Module Documentation . . . . .</b>	<b>59</b>
6.1 UART Driver . . . . .	59
6.1.1 Detailed Description . . . . .	59
6.1.2 Data Structure Documentation . . . . .	60
6.1.3 Enum Reference . . . . .	62
6.1.4 Function Reference . . . . .	63
6.2 Flexio UART IPL . . . . .	72
6.2.1 Detailed Description . . . . .	72
6.2.2 Data Structure Documentation . . . . .	72
6.2.3 Macro Definition Documentation . . . . .	73
6.2.4 Enum Reference . . . . .	73
6.2.5 Variable Documentation . . . . .	76
6.3 Lpuart UART IPL . . . . .	77
6.3.1 Detailed Description . . . . .	77
6.3.2 Data Structure Documentation . . . . .	78
6.3.3 Types Reference . . . . .	79
6.3.4 Enum Reference . . . . .	80
6.3.5 Function Reference . . . . .	81

## Chapter 1

### Revision History

Revision	Date	Author	Description
1.0	04.08.2023	NXP RTD Team	S32K1_S32M24X Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 2.0.0

## Chapter 2

### Introduction

- [Supported Derivatives](#)
- [Overview](#)
- [About This Manual](#)
- [Acronyms and Definitions](#)
- [Reference List](#)

This User Manual describes NXP Semiconductor CDD UART for S32K1\_S32M24X. CDD UART driver configuration parameters and deviations from the specification are described in Driver chapter of this document.

### 2.1 Supported Derivatives

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

- s32k116\_qfn32
- s32k116\_lqfp48
- s32k118\_lqfp48
- s32k118\_lqfp64
- s32k142\_lqfp48
- s32k142\_lqfp64
- s32k142\_lqfp100
- s32k142w\_lqfp48
- s32k142w\_lqfp64
- s32k144\_lqfp48
- s32k144\_lqfp64 / MWCT1014S\_lqfp64

- s32k144\_lqfp100 / MWCT1014S\_lqfp100
- s32k144\_mapbga100
- s32k144w\_lqfp48
- s32k144w\_lqfp64
- s32k146\_lqfp64
- s32k146\_lqfp100 / MWCT1015S\_lqfp100
- s32k146\_mapbga100 / MWCT1015S\_mapbga100
- s32k146\_lqfp144
- s32k148\_lqfp100
- s32k148\_mapbga100 / MWCT1016S\_mapbga100
- s32k148\_lqfp144
- s32k148\_lqfp176
- s32m241\_lqfp64
- s32m242\_lqfp64
- s32m243\_lqfp64
- s32m244\_lqfp64

All of the above microcontroller devices are collectively named as S32K1\_S32M24X. Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power

## 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** style: Used for important terms, notes and warnings.
- *Italic* style: 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.

Warning

This is a warning



## 2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSMI	Basic Software Make file Interface
CS	Chip Select
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DMA	Direct Memory Access
ECU	Electronic Control Unit
FIFO	First In First Out
LSB	Least Significant Bit
MCU	Micro Controller Unit
MIDE	Multi Integrated Development Environment
MSB	Most Significant Bit
N/A	Not Applicable
RAM	Random Access Memory
SIU	Systems Integration Unit
SWS	Software Specification
UART	Universal asynchronous receiver-transmitter
XML	Extensible Markup Language
BSW	Basic Software
ISR	Interrupt Service Routine
OS	Operating System
GUI	Graphical User Interface
PB Variant	Post Build Variant
PC Variant	Pre Compile Variant
LT Variant	Link Time Variant

## 2.5 Reference List

#	Title	Version
1	Reference Manual	S32K1xx Series Reference Manual, Rev. 14, 09/2021
		S32M24x Reference Manual, Rev. 2 Draft A, 05/2023
2	Errata	S32K116_0N96V Rev. 22/OCT/2021
		S32K118_0N97V Rev. 22/OCT/2021
		S32K142_0N33V Rev. 22/OCT/2021
		S32K144_0N57U Rev. 22/OCT/2021
		S32K144W_0P64A Rev. 22/OCT/2021
		S32K146_0N73V Rev. 22/OCT/2021
		S32K148_0N20V Rev. 22/OCT/2021
		S32M244_P64A+P73G, Rev. 0

#	Title	Version
		S32M242_N33V+P73G, Rev. 0, 6/2023
3	Datasheet	S32K1xx Data Sheet, Rev. 14, 08/2021
		S32M2xx Data Sheet, Rev. 3 DraftA, 05/2023

## Chapter 3

### Driver

- [Requirements](#)
- [Driver Design Summary](#)
- [Hardware Resources](#)
- [Deviations from Requirements](#)
- [Driver Limitations](#)
- [Driver usage and configuration tips](#)
- [Runtime errors](#)
- [Symbolic Names Disclaimer](#)

### 3.1 Requirements

UART 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 Uart driver is part of the Real Time Drivers, performs the hardware access and offers a hardware independent API to the application. The Uart driver is implemented as an Autosar complex device driver. It uses the Uart hardware peripheral which provides support for implementing the Uart protocol. Hardware and software settings can be configured using an Autosar standard configuration tool. The driver reports errors to the error manager as defined in AUTOSAR. The following features are implemented in the Uart Module for S32K1\_S32M24X device:

- Full-duplex communication
- Configurable baud-rate. It supports the standard UART baudrates. The values can be checked in the Uart↔\_BaudrateType enum.
- Configurable parity, stop bits
- DMA transfers
- Callback notifications
- Continuous transfers in an asynchronous manner

### 3.3 Hardware Resources

**3.3.1 Physical Uart Channels:** The hardware instances configured by the Uart driver:

- From Lpuart\_0 to Lpuart\_2 with S32K144, S32K142W, S32K144W, S32K146, S32K148.
- From Lpuart\_0 to Lpuart\_1 with S32K116, S32K118, S32K142.
- From Lpuart\_0 to Lpuart\_1 with S32M241, S32M242, S32M243, S32M244.

The Flexio Channel configured by the Uart driver:

- From Flexio\_0 to Flexio\_3 with S32K116, S32K118, S32K142, S32K144, S32K142W, S32K144W, S32K146, S32K148.
- From Flexio\_0 to Flexio\_3 with S32M241, S32M242, S32M243, S32M244.

**Note:** In EB tresos, Lpuart Uart Physical Unit has selected by UartHwChannel ("General" tab).

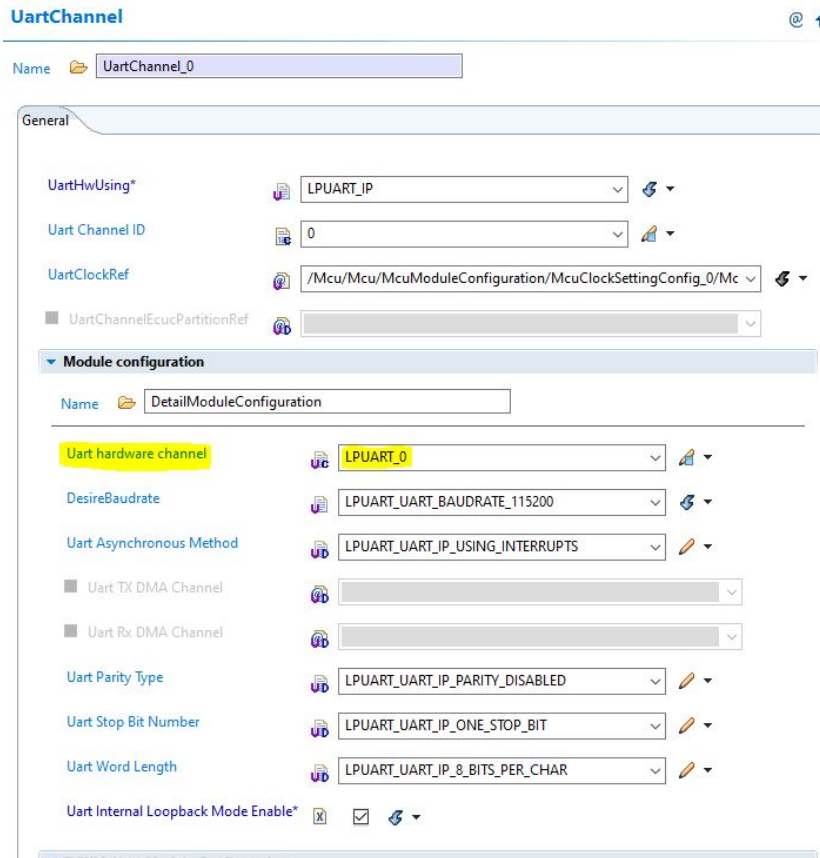


Figure 3.1 Uart Physical Unit has selected by UartHwChannel in EB Tresos

Flexio Uart Physical Channel has selected by MCL module at Mcl/MclConfig/FlexioCommon/FlexioCommon\_0/FlexioMclLogicChannels path

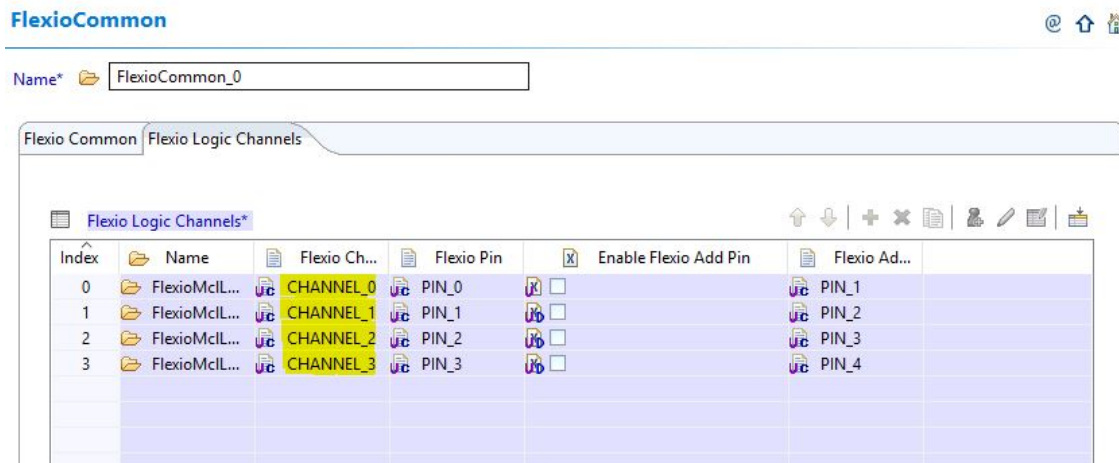


Figure 3.2 Flexio Uart Physical Channel has selected by MCL module in EB Tresos

### 3.3.2 Uart pins:

- The pins of S32K1xx can be found out in the file "S32K1xx\_IO\_Signal\_Description\_Input\_Multiplexing.xlsx" from attached file of S32K1XX Reference Manual Reference List. The Lpuart\_0, Lpuart\_1 and Lpuart\_2 use the pins LPUART0, LPUART1 and LPUART2 correspondingly.
- The pins of S32M24x can be found out in the file "S32M24x\_IOMUX.xlsx" from attached file of S32M24x Reference Manual Reference List. The Lpuart\_0 and Lpuart\_1 use the pins LPUART0 and LPUART1 correspondingly.
- Example for S32K148 the pins can be found in "S32K148\_IO\_Signal\_Description\_Input\_Multiplexing.xlsx" as blow:
  - For Lpuart channel:

Port	CR	SSS	Function	Module	Description	Direction
PTA0	PCR_PTA0	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART0_CTS	LPUART0	Clear To Send (bar)	I
PTA1	PCR_PTA1	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART0_RTS	LPUART0	Request To Send	O
PTA2	PCR_PTA2	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART0_RX	LPUART0	Receive	I
PTA3	PCR_PTA3	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART0_TX	LPUART0	Transmit	I/O
PTA27	PCR_PTA27	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0100	LPUART0_TX	LPUART0	Transmit	I/O
PTA28	PCR_PTA28	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0100	LPUART0_RX	LPUART0	Receive	I
PTB0	PCR_PTB0	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART0_RX	LPUART0	Receive	I
PTB1	PCR_PTB1	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART0_TX	LPUART0	Transmit	I/O
PTC2	PCR_PTC2	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0100	LPUART0_RX	LPUART0	Receive	I
PTC3	PCR_PTC3	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0100	LPUART0_TX	LPUART0	Transmit	I/O
PTC8	PCR_PTC8	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART0_CTS	LPUART0	Clear To Send (bar)	I
PTC9	PCR_PTC9	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART0_RTS	LPUART0	Request To Send	O

Figure 3.3 Pins selected for LPUART0 of S32K148

Port	CR	SSS	Function	Module	Description	Direction
PTA6	PCR_PTA6	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART1_CTS	LPUART1	Clear To Send (bar)	I
PTA7	PCR_PTA7	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART1_RTS	LPUART1	Request To Send	O
PTA18	PCR_PTA18	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART1_TX	LPUART1	Transmit	I/O
PTA19	PCR_PTA19	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART1_RX	LPUART1	Receive	I
PTB22	PCR_PTB22	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0101	LPUART1_TX	LPUART1	Transmit	I/O
PTB23	PCR_PTB23	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART1_RX	LPUART1	Receive	I
PTC6	PCR_PTC6	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART1_RX	LPUART1	Receive	I
PTC7	PCR_PTC7	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART1_TX	LPUART1	Transmit	I/O
PTC8	PCR_PTC8	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART1_RX	LPUART1	Receive	I
PTC9	PCR_PTC9	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART1_TX	LPUART1	Transmit	I/O
PTD13	PCR_PTD13	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART1_RX	LPUART1	Receive	I
PTD14	PCR_PTD14	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART1_TX	LPUART1	Transmit	I/O
PTE2	PCR_PTE2	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART1_CTS	LPUART1	Clear To Send (bar)	I
PTE6	PCR_PTE6	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART1_RTS	LPUART1	Request To Send	O
PTE15	PCR_PTE15	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART1_CTS	LPUART1	Clear To Send (bar)	I
PTE16	PCR_PTE16	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART1_RTS	LPUART1	Request To Send	O

Figure 3.4 Pins selected for LPUART1 of S32K148

Port	CR	SSS	Function	Module	Description	Direction
PTA8	PCR_PTA8	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART2_RX	LPUART2	Receive	I
PTA9	PCR_PTA9	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART2_TX	LPUART2	Transmit	I/O
PTA29	PCR_PTA29	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0100	LPUART2_TX	LPUART2	Transmit	I/O
PTA30	PCR_PTA30	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART2_RX	LPUART2	Receive	I
PTC12	PCR_PTC12	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0100	LPUART2_CTS	LPUART2	Clear To Send (bar)	I
PTC13	PCR_PTC13	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0100	LPUART2_RTS	LPUART2	Request To Send	O
PTD6	PCR_PTD6	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART2_RX	LPUART2	Receive	I
PTD7	PCR_PTD7	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0010	LPUART2_TX	LPUART2	Transmit	I/O
PTD11	PCR_PTD11	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART2_CTS	LPUART2	Clear To Send (bar)	I
PTD12	PCR_PTD12	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0110	LPUART2_RTS	LPUART2	Request To Send	O
PTD17	PCR_PTD17	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART2_RX	LPUART2	Receive	I
PTE3	PCR_PTE3	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART2_RTS	LPUART2	Request To Send	O
PTE9	PCR_PTE9	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART2_CTS	LPUART2	Clear To Send (bar)	I
PTE12	PCR_PTE12	0000_0000	DISABLED		Signal Path Disabled	-
		0000_0011	LPUART2_TX	LPUART2	Transmit	I/O

Figure 3.5 Pins selected for LPUART2 of S32K148

- For Flexio channel: Each Flexio channel uses one of the pin in list FXIO\_D0, FXIO\_D1,..., FXIO\_D3.

Po	CR	SSS	Function	Module	Description	Section
PTA0		0000_0100	FXIO_D2	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA1		0000_0100	FXIO_D3	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA2		0000_0101	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA3		0000_0101	FXIO_D5	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA8		0000_0100	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA9		0000_0100	FXIO_D7	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA10		0000_0100	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA11		0000_0100	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA21		0000_0011	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA22		0000_0011	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA23		0000_0011	FXIO_D2	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTA24		0000_0011	FXIO_D3	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTC30		0000_0011	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTC31		0000_0011	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD0		0000_0110	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD1		0000_0110	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD2		0000_0100	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD2		0000_0101	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD3		0000_0100	FXIO_D5	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD3		0000_0101	FXIO_D7	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD8		0000_0101	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD9		0000_0011	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD18		0000_0011	FXIO_D2	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD19		0000_0011	FXIO_D3	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD26		0000_0011	FXIO_D7	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTD31		0000_0011	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE4		0000_0110	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE5		0000_0110	FXIO_D7	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE10		0000_0110	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE11		0000_0110	FXIO_D5	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE15		0000_0110	FXIO_D2	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE16		0000_0110	FXIO_D3	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE17		0000_0011	FXIO_D5	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O
PTE18		0000_0011	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	I/O

Figure 3.6 Pins selected for Flexio of S32K148

## 3.4 Deviations from Requirements

The driver deviates from the AUTOSAR UART Driver software specification in some places. The table identifies the AUTOSAR requirements that are out of scope, not implemented or not fully implemented for the Uart Driver.

Term	Definition
N/S	Out of scope
N/I	Not implemented
N/F	Not fully implemented

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

## 3.5 Driver Limitations

### 3.5.1 DMA limitation



- Driver does not support DMA feature in 9-bits, 10-bits mode.

### 3.5.2 Feature not supporting

- FlexIO Uart does not support Loopback mode for self testing.

## 3.6 Driver usage and configuration tips

### 3.6.1 How to use User notifications

- The Uart driver can use a function callback in order to notify the application on some of the following events in a transmission: *end transfer*, *rx buffer full*, *tx buffer empty* and *error*. This notification is configured via the configurators supported on the driver: EBT and Design Studio. To enable Uart Callback, on both configurators, GeneralConfiguration/UartCallbackCapability must be set to "True".

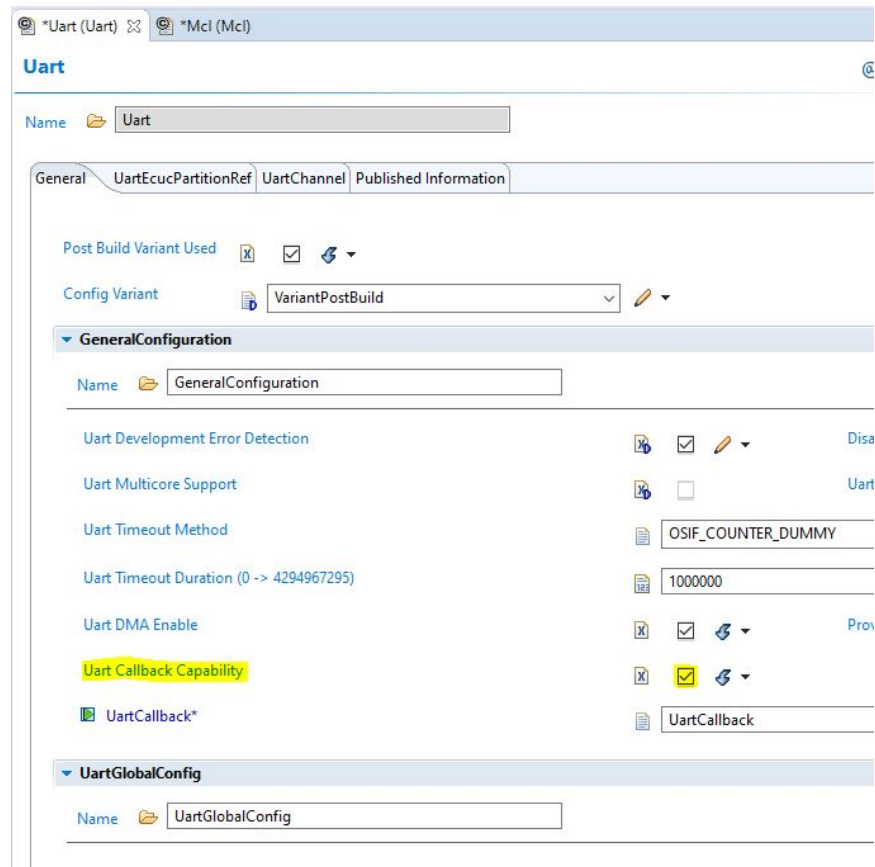
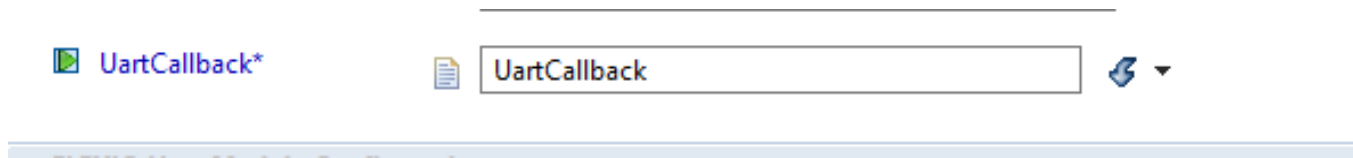


Figure 3.7 Enable Uart Callback Capability

If the HLD driver is used, then the name of the functions must be configured in the following nodes:





**Figure 3.8 User Notifications**

- The signature of the functions shall follow the `Uart_CallbackType` type.
- If the IP driver is used, the function name can be configured in the same path, but the function signature should follow the callback type corresponding to the Hardware Instance used (Lpuart or Flexio).
- For IP driver, an extra feature can be used: a callback parameter. This variable is a parameter of the notification function and has the `void*` type in order to be a flexible solution for the user. This parameter can be casted to any data type and can be used to pass information from callback call to another. In order to use it in the application, define it in the application as a global variable `void* callback_parameter_name`. Add the variable name (`callback_parameter_name`) in the GeneralConfiguration container of the configurator.
- Another important callback parameter passed is the event that triggered the notification call. Its type depends on the instances of driver in use: `Uart_EventType` or `Ip_name_Uart_Ip_EventType`. Both enums have the following macros: `_EVENT_RX_FULL = 0x00U`, `_EVENT_TX_EMPTY = 0x01U`, `_EVENT_END_TRANSFER = 0x02U`, `_EVENT_ERROR = 0x03U`,

#### Note

The first two can be used with `Uart_SetBuffer` or `Lpuart/Flexio_Uart_Ip_Setbuffer` API to replace the current buffer when it is full/empty. This approach ensures a continuous transfer.

### 3.6.2 How to configure Flexio Uart module

FlexIO module has three types of resources: SHIFTERS, TIMERS and PINS. These resources must be split between RTD drivers without any overlap. In order to define the solution for a correct resource allocation, we defined a channel as following: `FLEXIO_CHANNEL_x` - Has allocated Shifter `x` and Timer `x`.

For example: If a driver chooses `FLEXIO_CHANNEL_0`, Shifter 0 and Timer 0 are allocated to it. Separately, a channel has to configure a PIN.

First, `MclGeneral/MclFlexioCommon/MclEnableFlexioCommon` must be set to "True"

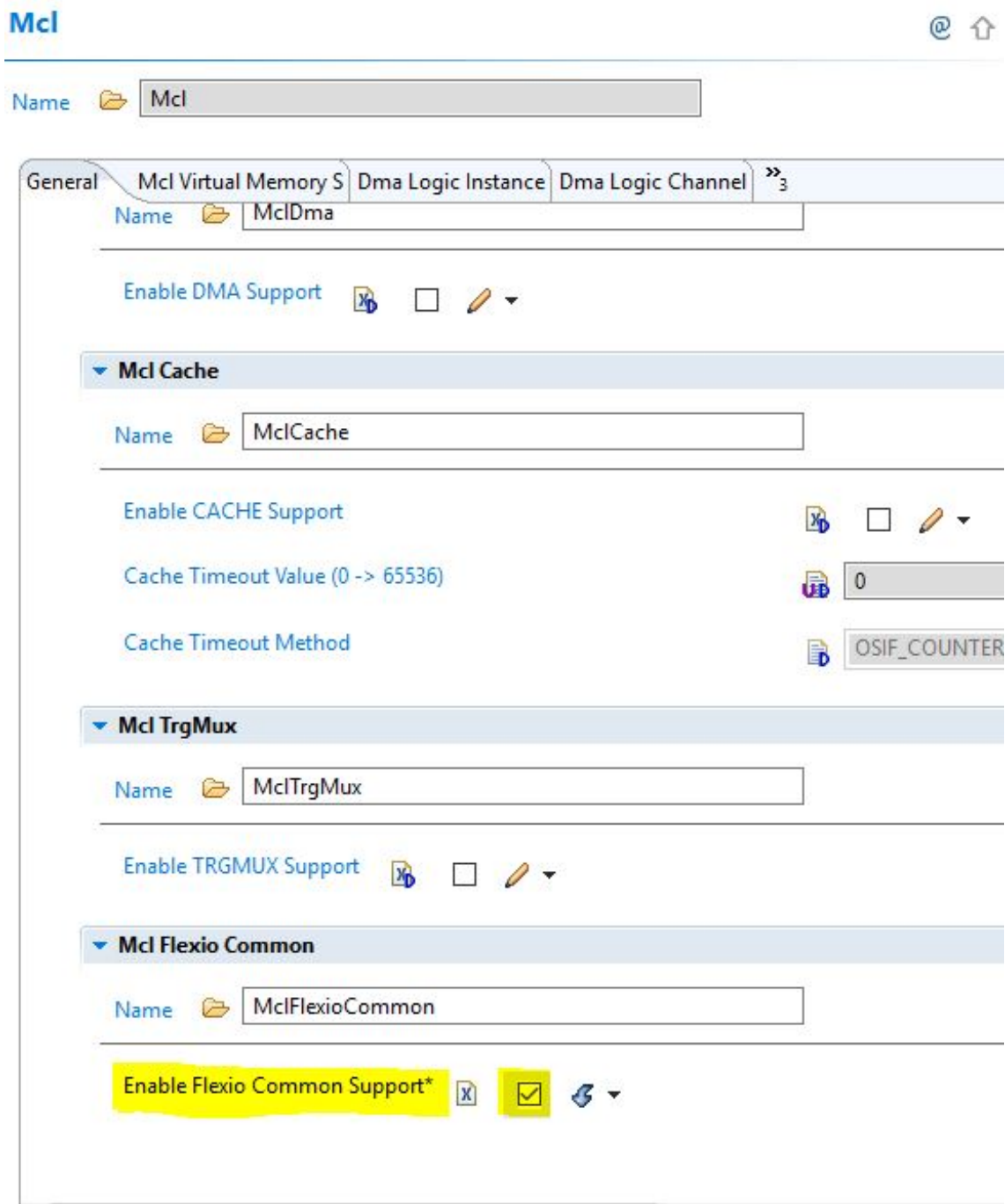
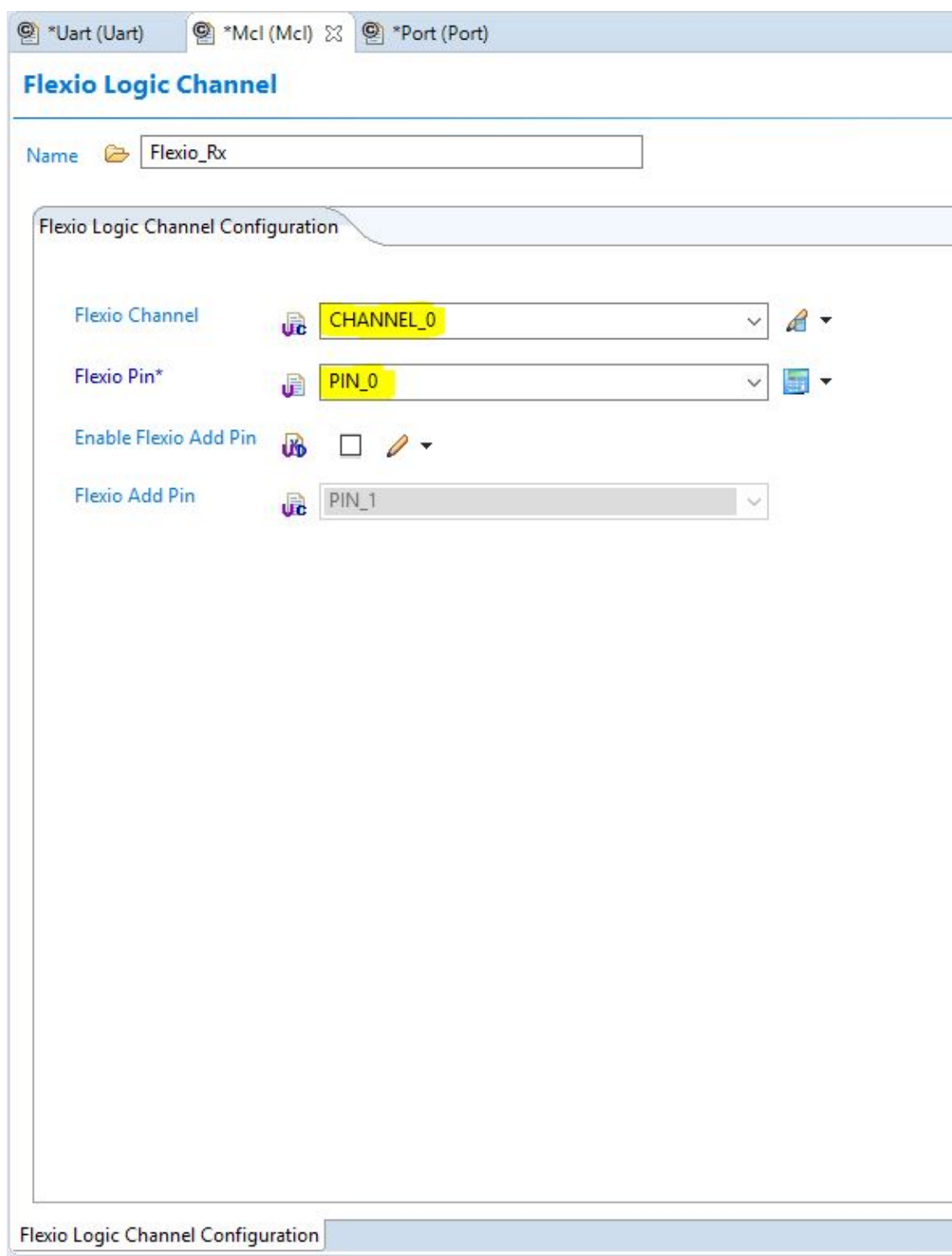


Figure 3.9 Enable Flexio Common Support

Flexio Channels are configured in the Mcl component at the Mcl/MclConfig/FlexioCommon/FlexioMclLogicChannels path. Each driver shall use as many channels references from Mcl as it needs. For example Flexio Uart Transmit, requires 1 shifter, 1 timer and 1 pin, therefore this channel references will be used in the configuration.



**Figure 3.10 Use Flexio Channel 0 (TIMER0, SHIFTER0) and Pin\_0(FXIO\_D0)**

Then Uart/UartGlobalConfig/UartChannel/FlexioModuleConfiguration/UartHwChannelRef must be select Flexio Channel respectively

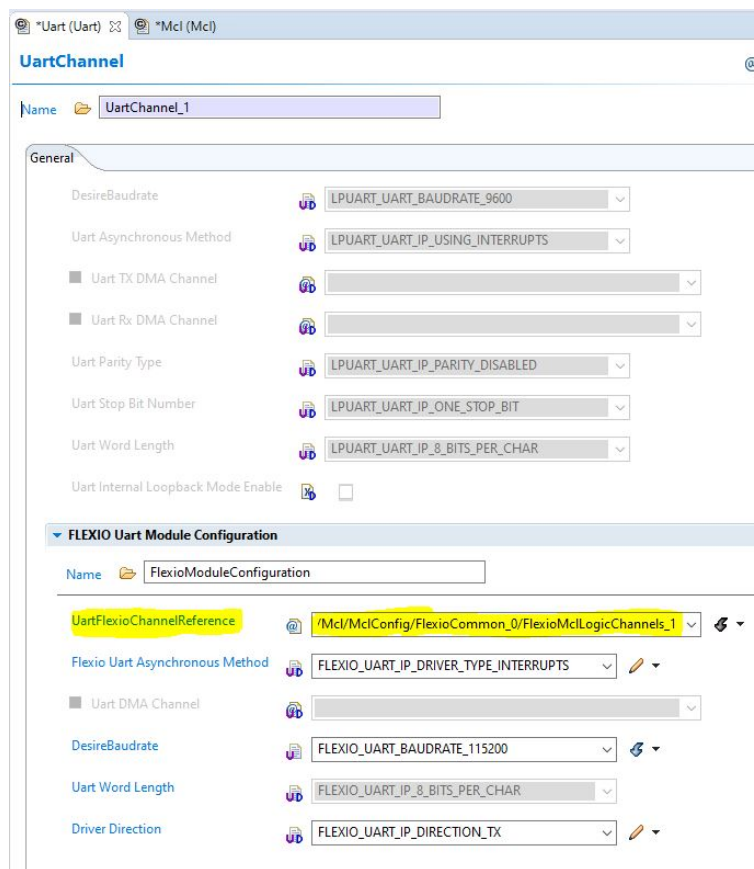


Figure 3.11 Select Flexio Channel configured in MCL module

Note: Remove any access to common registers, described in the Code guideline section. In the application/testcases, before using FlexIO, call `Mcl_Init()` function, located in `CDD_Mcl.c` source file. This API will set the module enable bit (`CTRL[FLEXEN]`). At the end of the application/testcase, call `Mcl_DeInit` function. This API will disable the FlexIO module by set to 0 `CTRL[FLEXEN]` bit. For accessing the common registers, include `Flexio_Mcl_Ip_HwAccess.h` header and use the API from it.

**3.6.3 How to choose the appropriate baudrate** Check the generated user struct configuration for the actual calculated baudrate.

- The baudrate deviation should be less than 5% with the normal baudrate value (lower than 921600) for properly operation.
- For higher baudrate (921600 or 1843200), The baudrate deviation should be less than 1%.
- Flexio Baudrate divider greater than 255 it will raise an error, user need to choose a reasonable Clock or Baudrate.

**3.6.4 How to choose the appropriate timeout in general** The timeout need to config in GeneralConfiguration tab of EB Tresos or S32DS tool. For normal operation, this value need to config larger than 1 Uart frame period ( $1s * (\text{number of bit per frame}) / \text{baudrate}$ )

**3.6.5 How to enable Internal Loopback for self test.(Available only on LPUART channels)** Loop mode is sometimes used to check software, independent of connections in the external system, to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input and the RXD pin is not used by the LPUART

For enabling this mode, you can check the `UartInternalLoopbackEnable` for each channel that will be configured in this mode.

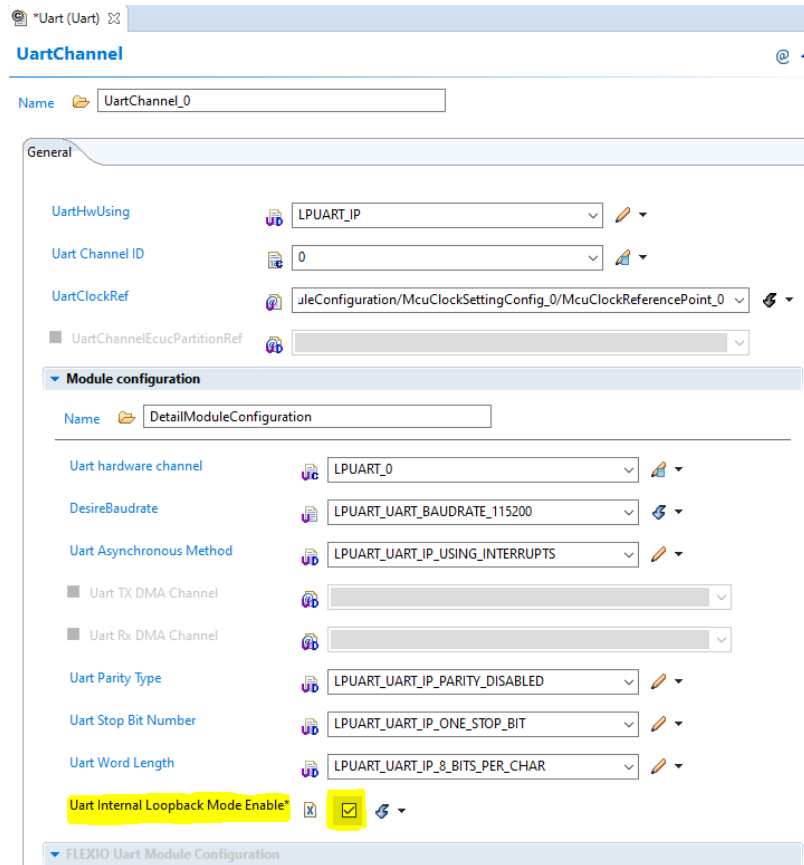


Figure 3.12 Internal Loopback mode configuration

Note: User should call Async Receive function before Async Send function to ensure that the receiver is ready to receive data before the data is transmitted.

### 3.6.6 How to set customized baudrate

Baudrate customization allows the user to set the any baudrate value supported by hardware for the Uart channel.

For LPUART IP:

- Select `LPUART_UART_BAUDRATE_CUSTOM` in `DesireBaudrate` field.

- Fill in the value of Custom Baudrate Mantissa (SBR) and Custom Baudrate Divisor (OSR). It will be evaluated depending on Clock Source value and user desired baudrate.

Desired baudrate = Clock Source value / (Custom Baudrate Mantissa \* Custom Baudrate Divisor)

For example:

**Figure 3.13** With LPUART1\_CLK is 2.4E7 Hz and user desire baudrate is 4800000 Bps

Note: Press the automatic calculate value button to show the baudrate value in Custom baudrate value field.

For FLEXIO UART IP:

- Select FLEXIO\_UART\_BAUDRATE\_CUSTOM in DesireBaurate field.
- Select the Custom Timer Decrement:  
 FLEXIO\_TIMER\_DECREMENT\_FXIO\_CLK\_SHIFT\_TMR is Decrement counter on FlexIO clock. Shift clock equals timer output.  
 FLEXIO\_TIMER\_DECREMENT\_FXIO\_CLK\_DIV\_16 is Decrement counter on FlexIO clock divided by 16. Shift clock equals timer output.(Not support on S32M24x platform)  
 FLEXIO\_TIMER\_DECREMENT\_FXIO\_CLK\_DIV\_256 is Decrement counter on FlexIO clock divided by 256. Shift clock equals timer output.(Not support on S32M24x platform)

- Fill in the value of Custom Baudrate Divider. It will be evaluated depending on Clock Source value and user desire baudrate.

Desire baudrate = Clock Source value / (Custom Timer Decrement \* 2 \* (Custom Baudrate Divider + 1))

For example:

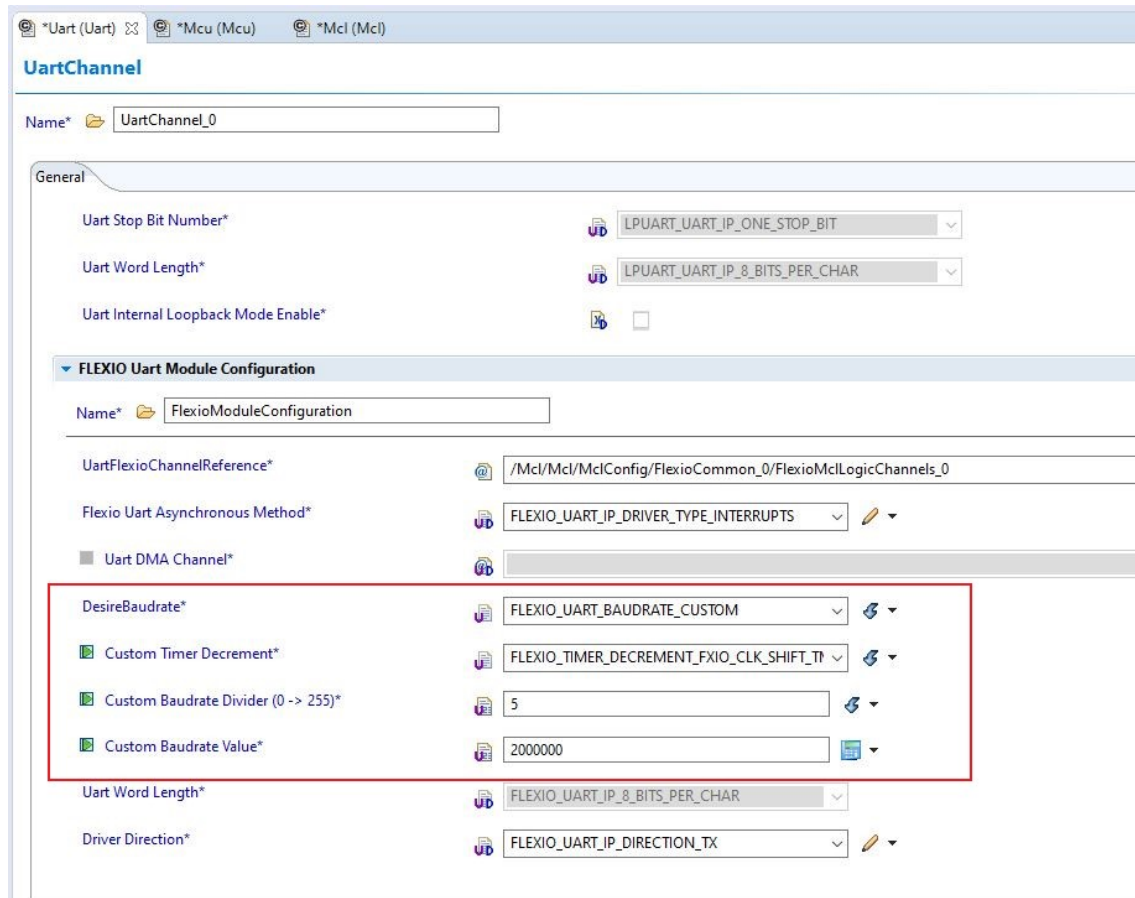
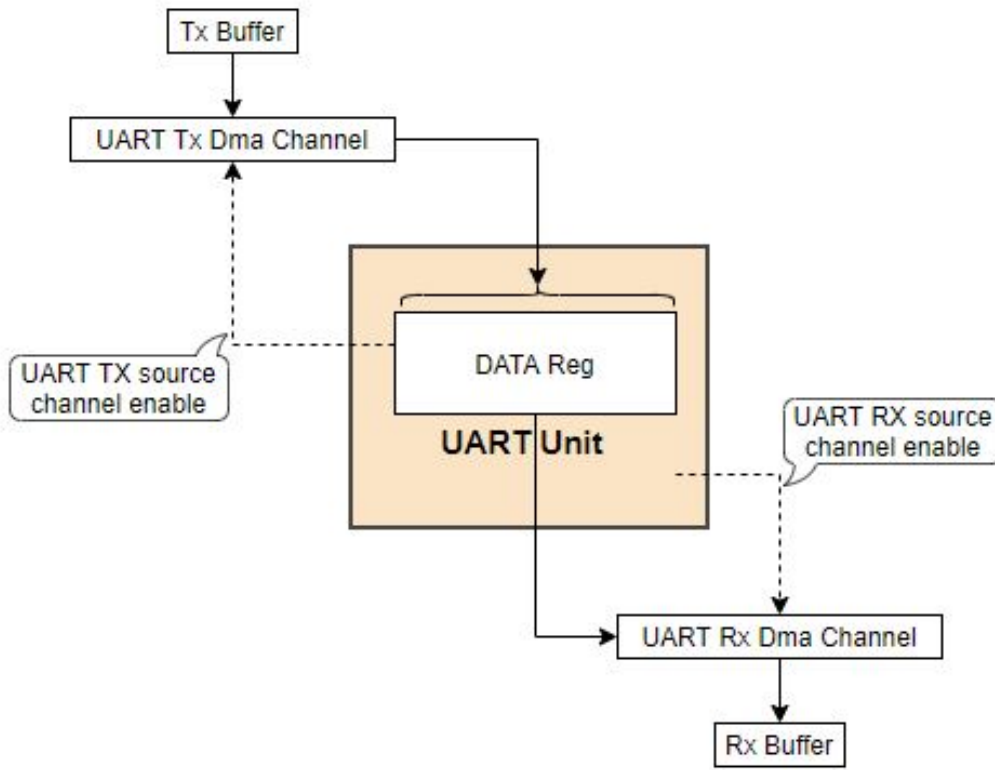


Figure 3.14 With FLEXIO\_CLK is 2.4E7 Hz and user desire baudrate is 2000000 Bps

Note: Press the automatic calculate value button to show the baudrate value in Custom baudrate value field.

### 3.6.7 How to configure DMA

This section applies only to Uart units configured for asynchronous transmission and which use DMA for serializing/deserializing data between the hardware unit and the TX/RX buffers (UartInterruptDmaMethod = DMA).



**Figure 3.15 DMA transferring mode internal architecture**

Each Uart unit configured in DMA mode requires 2 distinct DMA channels from the same DMA Mux:

- **Uart TX DMA Channel:** The TX DMA channel used for filling TX Data register with the data in TX Buffer. This channel is triggered by TX Uart unit event and must be wired to Uart TX source (configured inside the MCL module "MclConfig/Dma Logic Channel" container).
- **Uart RX DMA Channel:** The RX DMA channel used for filling RX buffer with the deserialized data. This channel is triggered by RX Uart unit event and must be wired to Uart RX source (configured inside the MCL module "MclConfig/Dma Logic Channel" container).

#### Note

- If DMA uses fixed priority arbitration, then the priority must be Uart RX DMA Channel > Uart TX DMA Channel.
- If DMA uses round robin arbitration, no priority constraints are applied on Uart TX DMA Channel, Uart RX DMA Channel priority.
- DMA asynchronous transmission is just used in 7-bits data mode and 8-bits data mode.

Next figures show an example of DMA configuration for Uart unit.



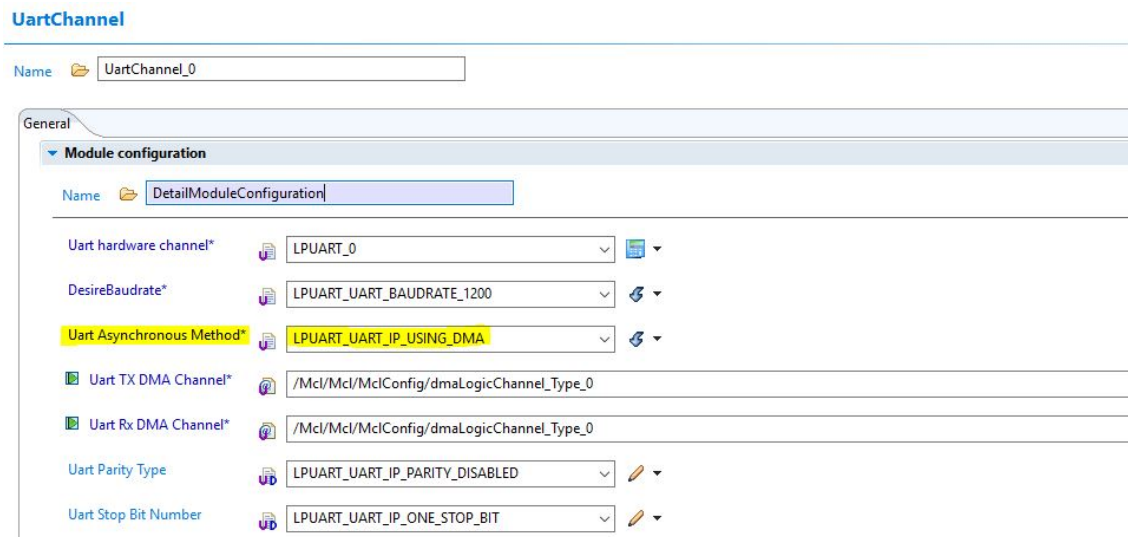


Figure 3.16 DMA Configuration sample for Uart Physical Unit - Uart module in EB Tresos

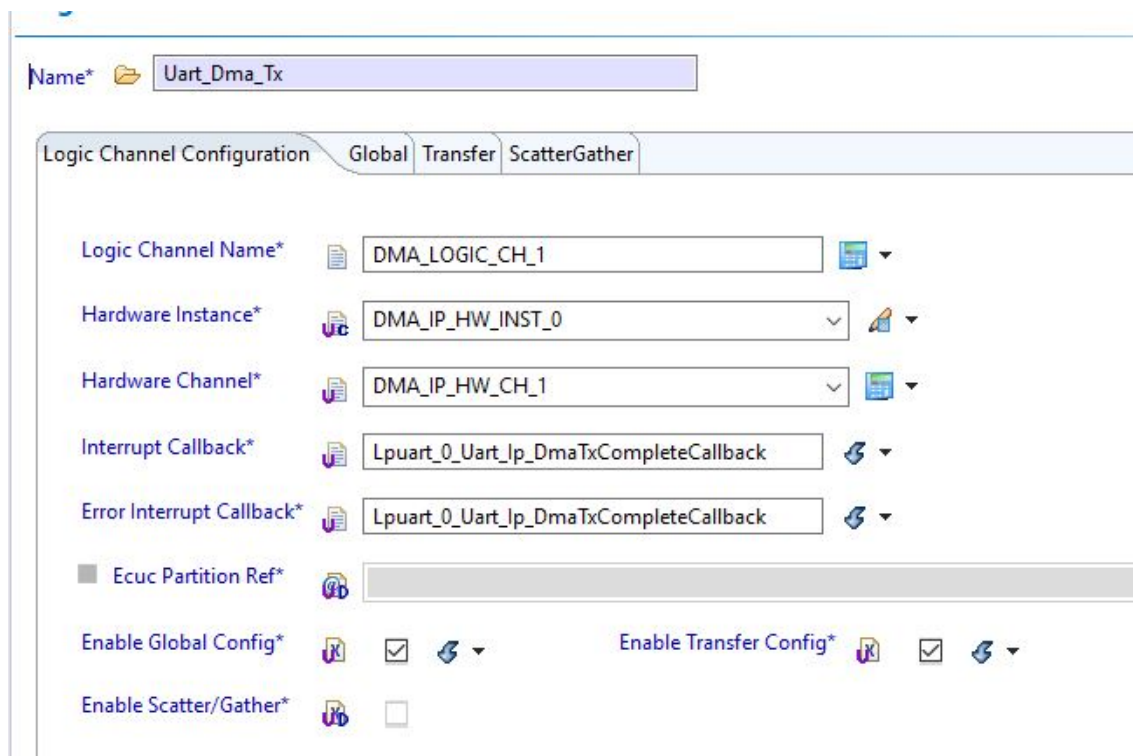




Figure 3.17 DMA TX General configuration - MCL module in EB Tresos

**Logic Channel** @ ↑


Name\*  Uart\_Dma\_Tx





Logic Channel Configuration **Global** Transfer ScatterGather



**▼ dmaLogicChannel\_GlobalConfigType**



Name\*  dmaLogicChannel\_GlobalConfigType

**▼ Request**


Name\*  dmaLogicChannelConfig\_GlobalRequestType



Enable DMAMUX Trigger\*  ☐  Enable DMAMUX Source\*  ☒ 

DMAMUX0 Source\*  DMA\_IP\_REQ\_MUX0\_LPUART0\_TX 


Enable DMA Request\*  ☒ 




**▼ Interrupt**

Name\*  dmaLogicChannelConfig\_GlobalInterruptType

Enable Error Interrupt\*  ☐ 

**▼ Priority**

Name\*  dmaLogicChannelConfig\_GlobalPriorityType

Level Priority\*  DMA\_IP\_LEVEL\_PRIO0  










Enable Preemption\*  ☐  Disable Preempt\*  ☐ 



Figure 3.18 DMA TX Global Configuration - MCL module in EB Tresos



Name\*  Uart\_Dma\_Rx



Logic Channel Configuration   Global   Transfer   ScatterGather



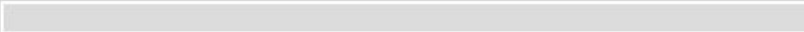
Logic Channel Name\*  DMA\_LOGIC\_CH\_0 





Hardware Instance\*  DMA\_IP\_HW\_INST\_0 

Hardware Channel\*  DMA\_IP\_HW\_CH\_0 

Interrupt Callback\*  Lpuart\_0\_Uart\_Ip\_DmaRxCompleteCallback 

Error Interrupt Callback\*  Lpuart\_0\_Uart\_Ip\_DmaRxCompleteCallback 

 Ecuc Partition Ref\*  

Enable Global Config\*  ☒    Enable Transfer Config\*  ☒ 


Enable Scatter/Gather\*  ☐

Figure 3.19 DMA Rx General configuration - MCL module in EB Tresos

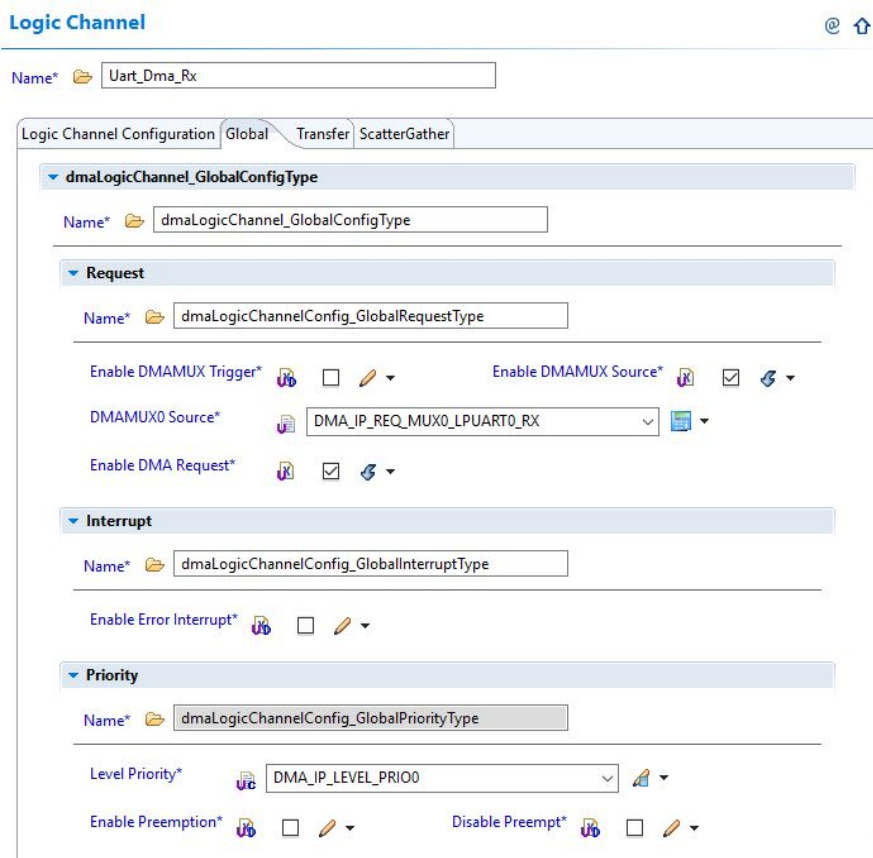


Figure 3.20 DMA Rx Global Configuration - MCL module in EB Tresos

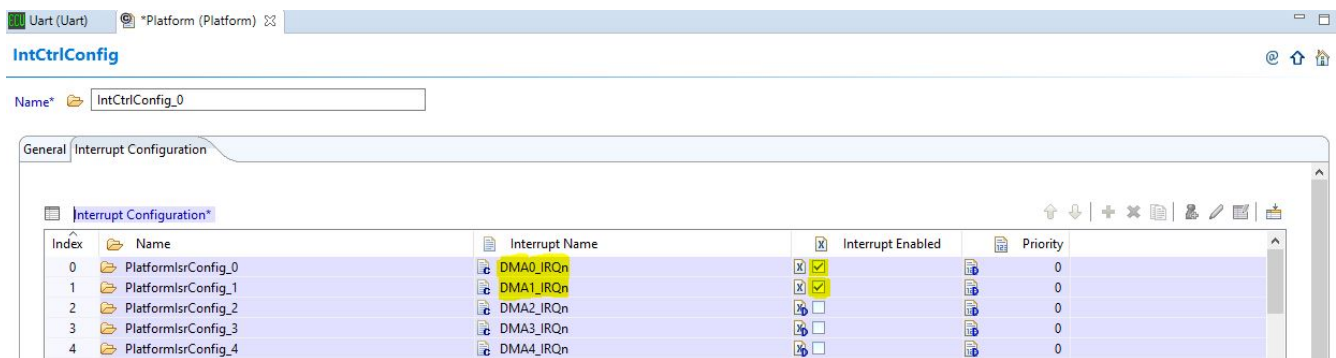


Figure 3.21 Enable DMA IRQ Configuration - Platform module in EB Tresos

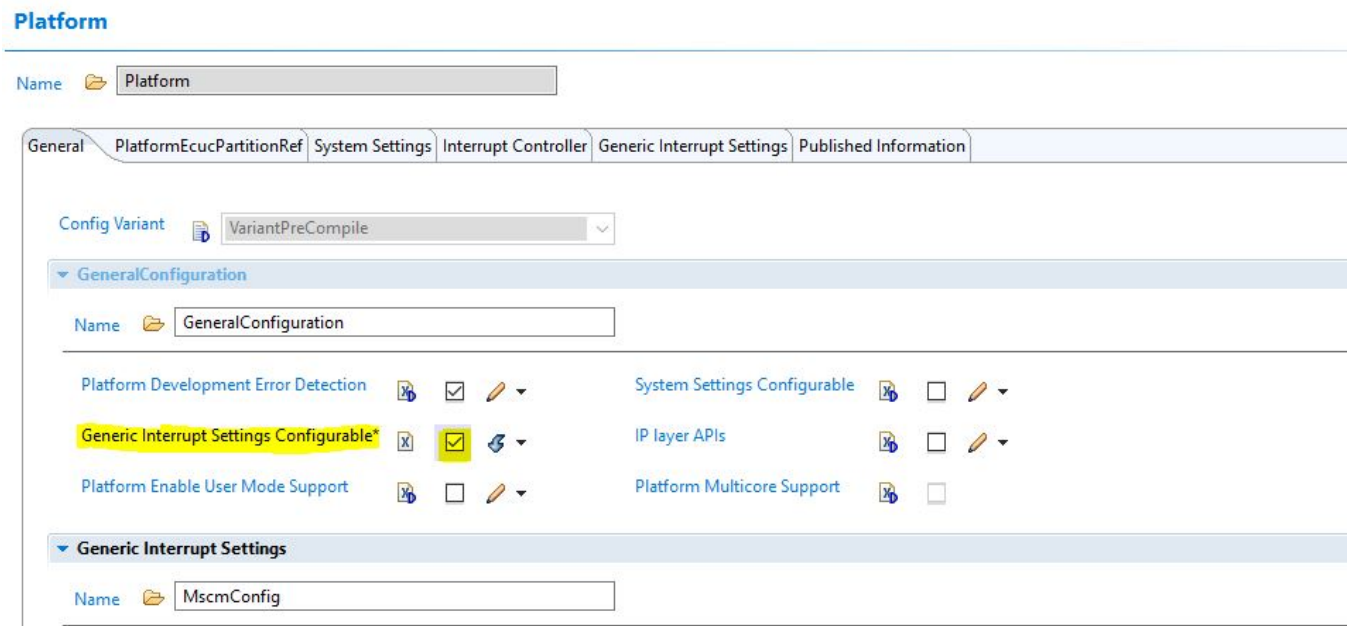


Figure 3.22 Enable Generic Interrupt Setting Configuration - Platform module in EB Tresos

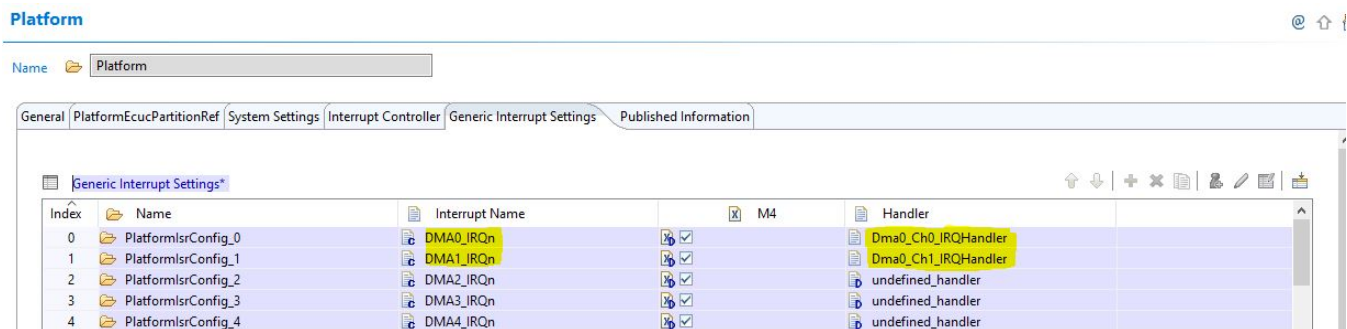


Figure 3.23 Fill In DMA IRQ Function Name - Platform module in EB Tresos

Note:

- About configuration of DMA Callback notification please refer chapter 6.3 Calls to Notification Functions, Callbacks, Callouts in Integration Manual document.
- Address of Buffer is used for Transmit and Receive data with DMA should be placed in UNSPECIFIED\_N←O\_CACHEABLE area please refer chapter 5.6 Data Cache Restrictions in Integration Manual document.

### 3.6.8 How to configure Idle Interrupt

In Idle Interrupt mode, for reception operation, the LPUART will be configured to start detecting an idle character from the previous stop bit(any data bits and stop bits count towards the idle character detection). And an idle characters that must be received before an idle line condition is detected. It only support for the LPUART.

For enabling this mode, you can check the UartTimeoutEnable for each channel that will be configured in this mode.

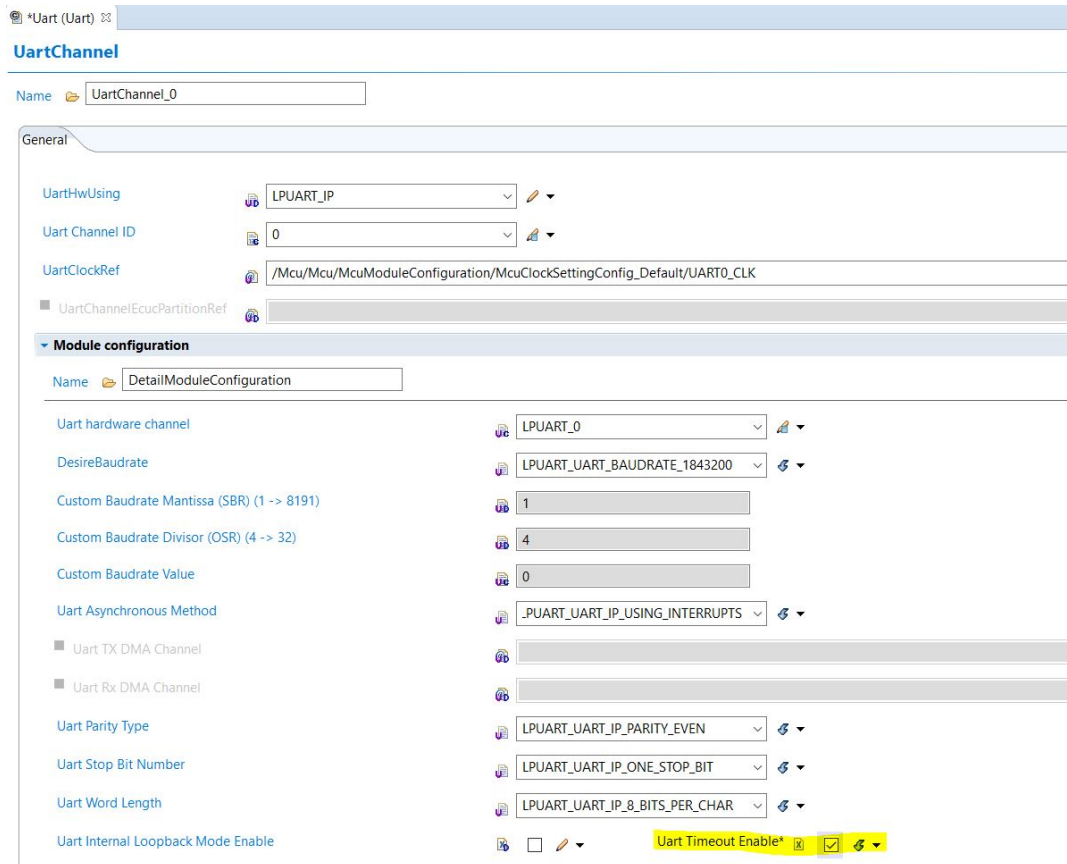


Figure 3.24 Idle interrupt mode internal architecture

Note:

- When the mode is enable for DMA, please add configuration for the interrupt handle to use.

### 3.7 Runtime errors

The driver generates the following DEM errors at runtime.

Function	Error Code	Condition triggering the error
Uart_SyncSend	UART_E_TIMEOUT	If an error occurs in transmit process, then transmitter can not send entire the message for a period of time longer than the configured timeout
Uart_SyncReceive	UART_E_TIMEOUT	If an error occurs in receive process or the receiver waits the message for a period of time longer than the configured timeout
Uart_Deinit	UART_E_DEINIT_FAILED	If one or more specific channels are in transmission progress, then Deinit channel will finish unsuccessfully

### 3.8 Symbolic Names Disclaimer

All containers having `symbolicNameValue` set to `TRUE` in the AUTOSAR schema will generate defines like:

```
#define <Mip>Conf_<Container_ShortName>_<Container_ID>
```

For this reason it is forbidden to duplicate the names of such containers across the RTD configurations 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 driver. All the parameters are described below.

- Module [Uart](#)
  - Container [GeneralConfiguration](#)
    - \* Parameter [UartDevErrorDetect](#)
    - \* Parameter [DisableUartRuntimeErrorDetect](#)
    - \* Parameter [UartMulticoreSupport](#)
    - \* Parameter [UartEnableUserModeSupport](#)
    - \* Parameter [UartTimeoutMethod](#)
    - \* Parameter [UartTimeoutDuration](#)
    - \* Parameter [UartDmaEnable](#)
    - \* Parameter [UartVersionInfoApi](#)
    - \* Parameter [UartCallbackCapability](#)
    - \* Parameter [UartCallback](#)
    - \* Reference [UartEcucPartitionRef](#)
  - Container [UartGlobalConfig](#)
    - \* Container [UartChannel](#)
      - Parameter [UartHwUsing](#)
      - Parameter [UartChannelId](#)
      - Reference [UartClockRef](#)
      - Reference [UartChannelEcucPartitionRef](#)
      - Container [DetailModuleConfiguration](#)
      - Parameter [UartHwChannel](#)
      - Parameter [DesireBaudrate](#)
      - Parameter [CustomBaudrateMantissa](#)
      - Parameter [CustomBaudrateDivisor](#)
      - Parameter [CustomBaudrateValue](#)
      - Parameter [UartInterruptDmaMethod](#)
      - Parameter [UartParityType](#)
      - Parameter [UartStopBitNumber](#)
      - Parameter [UartWordLength](#)
      - Parameter [UartInternalLoopbackEnable](#)



- Parameter [UartTimeoutEnable](#)
- Reference [UartDmaTxChannelRef](#)
- Reference [UartDmaRxChannelRef](#)
- Container [FlexioModuleConfiguration](#)
- Parameter [FlexioUartInterruptDmaMethod](#)
- Parameter [DesireBaudrate](#)
- Parameter [CustomTimerDecrement](#)
- Parameter [CustomBaudrateDivider](#)
- Parameter [CustomBaudrateValue](#)
- Parameter [bitCount](#)
- Parameter [driverDirection](#)
- Reference [UartHwChannelRef](#)
- Reference [FlexioDmaChannelRef](#)
- Container [CommonPublishedInformation](#)
  - \* Parameter [ArReleaseMajorVersion](#)
  - \* Parameter [ArReleaseMinorVersion](#)
  - \* Parameter [ArReleaseRevisionVersion](#)
  - \* Parameter [ModuleId](#)
  - \* Parameter [SwMajorVersion](#)
  - \* Parameter [SwMinorVersion](#)
  - \* Parameter [SwPatchVersion](#)
  - \* Parameter [VendorApiInfix](#)
  - \* Parameter [VendorId](#)

## 4.1 Module Uart

Configuration of the Universal asynchronous receiver-transmitter (Uart) module.

Included containers:

- [GeneralConfiguration](#)
- [UartGlobalConfig](#)
- [CommonPublishedInformation](#)

Property	Value
type	ECUC-MODULE-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantSupport	true
supportedConfigVariants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

## 4.2 Container GeneralConfiguration

### GeneralConfiguration

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

Note: Implementation Specific Parameter.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

## 4.3 Parameter UartDevErrorDetect

### UartDevErrorDetect

Switches the Development Error Detection and Notification ON or OFF.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	true

## 4.4 Parameter DisableUartRuntimeErrorDetect

DisableUartRuntimeErrorDetect

Switches the Runtime Error Detection and Notification ON or OFF.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
default Value	false

## 4.5 Parameter UartMulticoreSupport

UartMulticoreEnable

When this parameter is enabled, multi-core feature will be used in Uart driver.

That means mapping the Uart driver to multiple ECUC partitions to make the module API available in this partition.

The Uart driver will operate as an independent instance in each of the partitions.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
default Value	false

## 4.6 Parameter UartEnableUserModeSupport

When this parameter is enabled, the MDL module will adapt to run from User Mode, with the following measures:

- a) configuring REG\_PROT for ABC1, ABC2 IPs so that the registers under protection can be accessed from user mode by setting UAA bit in REG\_PROT\_GCR to 1
- b) using 'call trusted function' stubs for all internal function calls that access registers requiring supervisor mode.
- c) other module specific measures

for more information, please see chapter 5.7 User Mode Support in IM

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

## 4.7 Parameter UartTimeoutMethod

This parameter is used to select between different OsIf counter implementations. For additional details, please refer to the OsIf documentation.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE

Property	Value
defaultValue	OSIF_COUNTER_DUMMY
literals	['OSIF_COUNTER_DUMMY', 'OSIF_COUNTER_SYSTEM', 'OSIF_COUNTER_CUSTOM']

## 4.8 Parameter UartTimeoutDuration

Specifies the maximum number of loops for blocking function until a timeout is raised in short term wait loops. If LinTimeoutMethod is OSIF\_COUNTER\_SYSTEM or OSIF\_COUNTER\_CUSTOM, UartTimeoutDuration is in microsecond value. If LinTimeoutMethod is OSIF\_COUNTER\_DUMMY, the UartTimeoutDuration is number of wait loop.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1000
max	4294967295
min	0

## 4.9 Parameter UartDmaEnable

UartDmaEnable

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

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

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

## 4.10 Parameter UartVersionInfoApi

UartVersionInfoApi

Switches the Uart\_GetVersionInfo function ON or OFF.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	true

## 4.11 Parameter UartCallbackCapability

When this parameter is enabled, Callback feature will be used in Uart driver, with the following functions:

- a) completion of a reception or sending operation.
- b) reporting communication errors.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP

Property	Value
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
default Value	false

## 4.12 Parameter UartCallback

Callback for the rx full buffer event.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
default Value	NULL_PTR

## 4.13 Reference UartEcucPartitionRef

ECUC\_Uart\_00244.Maps the Uart driver to zero or multiple ECUC partitions to make the driver

API available in the according partition.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	0

Property	Value
upperMultiplicity	Infinite
postBuildVariantMultiplicity	true
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcuDefs/EcuC/EcuPartitionCollection/EcuPartition

## 4.14 Container UartGlobalConfig

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

Included subcontainers:

- [UartChannel](#)

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

## 4.15 Container UartChannel

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

Note: "User should use unique names for naming the Uart channels across different UartGlobalConfig Sets."

Included subcontainers:

- [DetailModuleConfiguration](#)
- [FlexioModuleConfiguration](#)



Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	6
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE

## 4.16 Parameter UartHwUsing

This parameter is user's desire Uart Hardware.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
defaultValue	LPUART_IP
literals	['LPUART_IP', 'FLEXIO_IP']

## 4.17 Parameter UartChannelId

Identifies the Uart channel.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	true
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A

Property	Value
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	6
min	0

## 4.18 Reference UartClockRef

Reference to the Uart clock source configuration, which is set into the MCU driver configuration.

This clock source is used for configure Uart baudrate.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Mcu/McuModuleConfiguration/McuClockSetting↔ Config/McuClockReferencePoint

## 4.19 Reference UartChannelEcucPartitionRef

Maps one single Uart channel to zero or one ECUC partitions.

The ECUC partition referenced is a subset of the ECUC partitions where the Uart driver is mapped to.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	true

Property	Value
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcuDefs/EcuC/EcuPartitionCollection/EcuPartition

## 4.20 Container DetailModuleConfiguration

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

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE

## 4.21 Parameter UartHwChannel

Selects the physical Lpuart Uart Channel.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true

Property	Value
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
defaultValue	LPUART_0
literals	['LPUART_0', 'LPUART_1']

## 4.22 Parameter DesireBaudrate

This parameter is user's desire baudrate.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
defaultValue	LPUART_UART_BAUDRATE_9600
literals	['LPUART_UART_BAUDRATE_CUSTOM', 'LPUART_UART_BAUDRATE_1200', 'LPUART_UART_BAUDRATE_2400', 'LPUART_UART_BAUDRATE_4800', 'LPUART_UART_BAUDRATE_7200', 'LPUART_UART_BAUDRATE_9600', 'LPUART_UART_BAUDRATE_14400', 'LPUART_UART_BAUDRATE_19200', 'LPUART_UART_BAUDRATE_28800', 'LPUART_UART_BAUDRATE_38400', 'LPUART_UART_BAUDRATE_57600', 'LPUART_UART_BAUDRATE_115200', 'LPUART_UART_BAUDRATE_230400', 'LPUART_UART_BAUDRATE_460800', 'LPUART_UART_BAUDRATE_921600', 'LPUART_UART_BAUDRATE_1843200']

## 4.23 Parameter CustomBaudrateMantissa

Identifies the custom baudrate value.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-INTEGER-PARAM-DEF

Property	Value
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE VARIANT-POST-BUILD: POST-BUILD
defaultValue	1
max	8191
min	1

## 4.24 Parameter CustomBaudrateDivisor

Identifies the custom baudrate value.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE VARIANT-POST-BUILD: POST-BUILD
defaultValue	4
max	32
min	4

## 4.25 Parameter CustomBaudrateValue

Identifies the custom baudrate value.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
default Value	0
max	9223372036854775807
min	-9223372036854775808

## 4.26 Parameter UartInterruptDmaMethod

Configures the mechanism used by the 'AsyncSend' or 'AsyncReceive' function (interrupts or DMA).

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
default Value	LPUART_UART_IP_USING_INTERRUPTS
literals	['LPUART_UART_IP_USING_INTERRUPTS', 'LPUART_UART_IP_USING_DMA']

## 4.27 Parameter UartParityType

Configures the type of parity to be used for UART bytes.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	LPUART_UART_IP_PARITY_DISABLED
literals	['LPUART_UART_IP_PARITY_DISABLED', 'LPUART_UART_IP_PARITY_EVEN', 'LPUART_UART_IP_PARITY_ODD']

## 4.28 Parameter UartStopBitNumber

Configures the number of stop bit to be used for UART frame.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	LPUART_UART_IP_ONE_STOP_BIT
literals	['LPUART_UART_IP_ONE_STOP_BIT', 'LPUART_UART_IP_TWO_STOP_BIT']

## 4.29 Parameter UartWordLength

Configures the word length in UART mode.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	LPUART_UART_IP_8_BITS_PER_CHAR
literals	['LPUART_UART_IP_7_BITS_PER_CHAR', 'LPUART_UART_IP_8_↵ BITS_PER_CHAR', 'LPUART_UART_IP_9_BITS_PER_CHAR', 'LPUA↵ RT_UART_IP_10_BITS_PER_CHAR']

## 4.30 Parameter UartInternalLoopbackEnable

UartInternalLoopbackEnable

This checks enables the internal loopback for the current channel.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

## 4.31 Parameter UartTimeoutEnable

UartTimeoutEnable

This checks enables the Uart Timeout Interrupt for the current channel.



Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: PRE-COMPILE
default Value	false

## 4.32 Reference UartDmaTxChannelRef

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

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Mcl/MclConfig/dmaLogicChannel_Type

## 4.33 Reference UartDmaRxChannelRef

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

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-REFERENCE-DEF

Property	Value
origin	NXP
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcuDefs/Mcl/MclConfig/dmaLogicChannel_Type

## 4.34 Container FlexioModuleConfiguration

This container contains the configuration (parameters) of the Flexio Uart Configuration.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: PRE-COMPILE
	VARIANT-PRE-COMPILE: PRE-COMPILE

## 4.35 Parameter FlexioUartInterruptDmaMethod

Configures the mechanism used by the 'AsyncSend' or 'AsyncReceive' function (interrupts or DMA).

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	FLEXIO_UART_IP_DRIVER_TYPE_INTERRUPTS
literals	['FLEXIO_UART_IP_DRIVER_TYPE_INTERRUPTS', 'FLEXIO_UART_IP_DRIVER_TYPE_DMA']

## 4.36 Parameter DesireBaudrate

This parameter is user's desire baudrate.

Note: Implementation Specific Parameter. In the case of baudrate is 1200, should use clocks less than 155MHz

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
defaultValue	FLEXIO_UART_BAUDRATE_9600
literals	['FLEXIO_UART_BAUDRATE_CUSTOM', 'FLEXIO_UART_BAUDRATE_1200', 'FLEXIO_UART_BAUDRATE_2400', 'FLEXIO_UART_BAUDRATE_4800', 'FLEXIO_UART_BAUDRATE_7200', 'FLEXIO_UART_BAUDRATE_9600', 'FLEXIO_UART_BAUDRATE_14400', 'FLEXIO_UART_BAUDRATE_19200', 'FLEXIO_UART_BAUDRATE_28800', 'FLEXIO_UART_BAUDRATE_38400', 'FLEXIO_UART_BAUDRATE_57600', 'FLEXIO_UART_BAUDRATE_115200', 'FLEXIO_UART_BAUDRATE_230400', 'FLEXIO_UART_BAUDRATE_460800', 'FLEXIO_UART_BAUDRATE_921600', 'FLEXIO_UART_BAUDRATE_1843200']

## 4.37 Parameter CustomTimerDecrement

This parameter is user's custom baudrate.

Note: Implementation Specific Parameter. In the case of baudrate is 1200, should use clocks less than 155MHz

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
defaultValue	FLEXIO_TIMER_DECREMENT_FXIO_CLK_SHIFT_TMR
literals	['FLEXIO_TIMER_DECREMENT_FXIO_CLK_SHIFT_TMR', 'FLEXIO_TIMER_DECREMENT_FXIO_CLK_DIV_16', 'FLEXIO_TIMER_DECREMENT_FXIO_CLK_DIV_256']

## 4.38 Parameter CustomBaudrateDivider

Identifies the custom baudrate value.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
defaultValue	32
max	255
min	0

## 4.39 Parameter CustomBaudrateValue

Identifies the custom baudrate value.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
defaultValue	0
max	9223372036854775807
min	-9223372036854775808

## 4.40 Parameter bitCount

Configures the word length in UART mode.

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	FLEXIO_UART_IP_8_BITS_PER_CHAR
literals	['FLEXIO_UART_IP_8_BITS_PER_CHAR']

## 4.41 Parameter driverDirection

Configures the mechanism used by the Send or Receive function (TX or RX).

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	FLEXIO_UART_IP_DIRECTION_TX
literals	['FLEXIO_UART_IP_DIRECTION_TX', 'FLEXIO_UART_IP_DIRECTION_RX']

## 4.42 Reference UartHwChannelRef

Uart Flexio Channel Reference

Reference to the Flexio Channel configure for the Request

Property	Value
type	ECUC-CHOICE-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
	VARIANT-POST-BUILD: POST-BUILD
requiresSymbolicNameValue	False
destinations	['/AUTOSAR/EcuDefs/Mcl/MclConfig/FlexioCommon/FlexioMclLogicChannels']

## 4.43 Reference FlexioDmaChannelRef

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

Note: Implementation Specific Parameter.

Property	Value
type	ECUC-REFERENCE-DEF
origin	NXP
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	true
valueConfigClasses	VARIANT-POST-BUILD: POST-BUILD
	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Mcl/MclConfig/dmaLogicChannel_Type

## 4.44 Container CommonPublishedInformation

Common container, aggregated by all modules.

It contains published information about vendor and versions.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

## 4.45 Parameter ArReleaseMajorVersion

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

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	4
max	4
min	4

#### 4.46 Parameter ArReleaseMinorVersion

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

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	7
max	7
min	7

#### 4.47 Parameter ArReleaseRevisionVersion

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

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A



Property	Value
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

## 4.48 Parameter ModuleId

Module ID of this module from Module List.

Note: Implementation Specific Parameter

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	255
max	255
min	255

## 4.49 Parameter SwMajorVersion

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

Note: Implementation Specific Parameter

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	2
max	2
min	2

## 4.50 Parameter SwMinorVersion

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

Note: Implementation Specific Parameter

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

## 4.51 Parameter SwPatchVersion

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

Note: Implementation Specific Parameter

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP

Property	Value
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

## 4.52 Parameter VendorApiInfix

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>.

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.

Note: Implementation Specific Parameter

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	

## 4.53 Parameter VendorId

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

Note: Implementation Specific Parameter

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	43
max	43
min	43



# Chapter 5

## Module Index

### 5.1 Software Specification

Here is a list of all modules:

UART Driver . . . . .	59
Flexio UART IPL . . . . .	72
Lpuart UART IPL . . . . .	77

## Chapter 6

### Module Documentation

#### 6.1 UART Driver

##### 6.1.1 Detailed Description

##### Data Structures

- struct [Uart\\_ChannelConfigType](#)  
*Uart channel configuration type structure. [More...](#)*
- struct [Uart\\_ConfigType](#)  
*Uart driver configuration type structure. [More...](#)*

##### Enum Reference

- enum [Uart\\_DrvStatusType](#)  
*Driver initialization status.*
- enum [Uart\\_DataDirectionType](#)  
*The type operation of an Uart channel.*
- enum [Uart\\_StatusType](#)  
*Uart operation status type.*

##### Function Reference

- void [Uart\\_Init](#) (const [Uart\\_ConfigType](#) \*Config)  
*Initializes the UART module.*
- void [Uart\\_Deinit](#) (void)  
*De-initializes the UART module.*
- Std\_ReturnType [Uart\\_SetBaudrate](#) (uint8 Channel, Uart\_BaudrateType Baudrate)  
*Configures the baud rate for the serial communication.*
- Std\_ReturnType [Uart\\_GetBaudrate](#) (uint8 Channel, uint32 \*Baudrate)

*Retrieves the baud rate which is currently set for the serial communication.*

- void [Uart\\_SetBuffer](#) (uint8 Channel, uint8 \*Buffer, uint32 BufferSize, [Uart\\_DataDirectionType](#) Direction)

*Configures a new buffer for continuous transfers.*

- Std\_ReturnType [Uart\\_SyncSend](#) (uint8 Channel, const uint8 \*Buffer, uint32 BufferSize, uint32 Timeout)

*Starts a synchronous transfer of bytes.*

- Std\_ReturnType [Uart\\_SyncReceive](#) (uint8 Channel, uint8 \*Buffer, uint32 BufferSize, uint32 Timeout)

*Starts a synchronous reception of bytes.*

- Std\_ReturnType [Uart\\_Abort](#) (uint8 Channel, [Uart\\_DataDirectionType](#) TransmissionType)

*Aborts an on-going transfer.*

- Std\_ReturnType [Uart\\_AsyncSend](#) (uint8 Channel, const uint8 \*Buffer, uint32 BufferSize)

*Starts an asynchronous transfer(send) of bytes.*

- Std\_ReturnType [Uart\\_AsyncReceive](#) (uint8 Channel, uint8 \*Buffer, uint32 BufferSize)

*Starts an asynchronous transfer(receive) of bytes.*

- [Uart\\_StatusType](#) [Uart\\_GetStatus](#) (uint8 Channel, uint32 \*BytesTransferred, [Uart\\_DataDirectionType](#) TransferType)

*Returns the status of the previous transfer.*

## 6.1.2 Data Structure Documentation

### 6.1.2.1 struct Uart\_ChannelConfigType

Uart channel configuration type structure.

This is the type of the external data structure containing the overall initialization data for one Uart Channel. A pointer to such a structure is provided to the Uart channel initialization routine for configuration of the Uart hardware channel.

Definition at line 316 of file Uart\_Types.h.

Data Fields

Type	Name	Description
uint8	UartChannelId	Uart channel configured
uint32	ChannelClockFrequency	The clock frequency configured on the given channel
const Uart_Ipw_HwConfigType *	UartChannelConfig	Pointer to a lower level channel configuration

### 6.1.2.2 struct Uart\_ConfigType

Uart driver configuration type structure.

This is the type of the pointer to the external data Uart Channels. A pointer of such a structure is provided to the Uart driver initialization routine for configuration of the Uart hardware channel.

Definition at line 338 of file Uart\_Types.h.



### Data Fields

Type	Name	Description
const <a href="#">Uart_ChannelConfigType</a> *	Configs[UART_CH_MAX_CONFIG]	Hardware channel. Constant pointer of the constant external data structure containing the overall initialization data for all the configured Uart Channels.

## 6.1.3 Enum Reference

### 6.1.3.1 Uart\_DrvStatusType

enum [Uart\\_DrvStatusType](#)

Driver initialization status.

This enum contains the values for the driver initialization status.

Enumerator

UART_DRV_UNINIT	Driver not initialized.
UART_DRV_INIT	Driver ready.

Definition at line 265 of file Uart\_Types.h.

### 6.1.3.2 Uart\_DataDirectionType

enum [Uart\\_DataDirectionType](#)

The type operation of an Uart channel.

Enumerator

UART_SEND	The sending operation.
UART_RECEIVE	The receiving operation.

Definition at line 277 of file Uart\_Types.h.

### 6.1.3.3 Uart\_StatusType

enum `Uart_StatusType`

Uart operation status type.

Enumerator

<code>UART_STATUS_NO_ERROR</code>	Uart operation is successfull
<code>UART_STATUS_OPERATION_ONGOING</code>	Uart operation on going
<code>UART_STATUS_ABORTED</code>	Uart operation aborted
<code>UART_STATUS_FRAMING_ERROR</code>	Uart framing error
<code>UART_STATUS_RX_OVERRUN_ERROR</code>	Uart overrun error
<code>UART_STATUS_PARITY_ERROR</code>	Uart parity error
<code>UART_STATUS_TIMEOUT</code>	Uart operation has timeout
<code>UART_STATUS_NOISE_ERROR</code>	Uart noise error
<code>UART_STATUS_DMA_ERROR</code>	Uart Dma Error error

Definition at line 286 of file `Uart_Types.h`.

## 6.1.4 Function Reference

### 6.1.4.1 Uart\_Init()

```
void Uart_Init (
    const Uart_ConfigType * Config )
```

Initializes the UART module.

This function performs software initialization of UART driver. It shall configure the Uart hardware peripheral for each channel.

Parameters

in	<i>Config</i>	- Pointer to UART driver configuration set.
----	---------------	---

Returns

void

Precondition

-

### 6.1.4.2 Uart\_Deinit()

```
void Uart_Deinit (
    void )
```

De-initializes the UART module.

This function performs software de-initialization of UART driver.

Parameters

-	
---	--

Returns

void

Precondition

-

### 6.1.4.3 Uart\_SetBaudrate()

```
Std_ReturnType Uart_SetBaudrate (
    uint8 Channel,
    Uart_BaudrateType Baudrate )
```

Configures the baud rate for the serial communication.

This function performs the setting of the communication baudrate provided in the parameter.

Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
in	<i>Baudrate</i>	- Baudrate value to be set.

Returns

Std\_ReturnType.

Return values

<i>E_NOT_OK</i>	If the Uart Channel is not valid or Uart driver is not initialized or a transfer is on-going or wrong core is addressed.
<i>E_OK</i>	Successfull baudrate configuration.

Precondition

Uart\_Init function must be called before this API.

#### 6.1.4.4 Uart\_GetBaudrate()

```
Std_ReturnType Uart_GetBaudrate (
    uint8 Channel,
    uint32 * Baudrate )
```

Retrieves the baud rate which is currently set for the serial communication.

This function returns via the second parameter the current serial baudrate.

Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
out	<i>Baudrate</i>	- Pointer to a memory location where the baudrate value is returned.

Returns

Std\_ReturnType.

Return values

<i>E_NOT_OK</i>	If the Uart Channel is not valid or Uart driver is not initialized or a transfer is on-going or wrong core is addressed or a NULL_PTR pointer has been provided
<i>E_OK</i>	Successfull baudrate retrieval.

Precondition

Uart\_Init function must be called before this API. Otherwise a random value can be returned.

6.1.4.5 Uart\_SetBuffer()

```
void Uart_SetBuffer (
    uint8 Channel,
    uint8 * Buffer,
    uint32 BufferSize,
    Uart_DataDirectionType Direction )
```

Configures a new buffer for continuous transfers.

This function can be called inside a notification callback and offers the possibility to change the buffer in order to assure a continuous asynchronous transfer.

Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
in	<i>Buffer</i>	- The new buffer provided.
in	<i>BufferSize</i>	- The size of the new buffer.
in	<i>Direction</i>	- This parameter indicates for which type of transmission is the buffer set. Its values are UART_SEND for setting a buffer when the previous buffer is empty and there are more bytes to send and UART_RECEIVE to set a new buffer when the previous buffer is full with received buffer and there is more data to be received.

Returns

void.

Precondition

Uart\_Init function must be called before this API.

6.1.4.6 Uart\_SyncSend()

```
Std_ReturnType Uart_SyncSend (
    uint8 Channel,
    const uint8 * Buffer,
    uint32 BufferSize,
    uint32 Timeout )
```

Starts a synchronous transfer of bytes.

This function starts sending a number of bytes in a synchronous manner.

Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
in	<i>Buffer</i>	- The buffer which contains the bytes to be sent.
in	<i>BufferSize</i>	- The Buffer size.
in	<i>Timeout</i>	- The timeout in us.

Returns

Std\_ReturnType.

Return values

<i>E_NOT_OK</i>	If the Uart Channel is not valid or Uart driver is not initialized or Buffer is a NULL_PTR or BufferSize is 0, meaning no space has been allocated for the buffer or a wrong core has been accessed or a transfer is already on going on the requested channel or timeout occurred.
<i>E_OK</i>	Successful transfer.

Precondition

Uart\_Init function must be called before this API.

#### 6.1.4.7 Uart\_SyncReceive()

```
Std_ReturnType Uart_SyncReceive (
    uint8 Channel,
    uint8 * Buffer,
    uint32 BufferSize,
    uint32 Timeout )
```

Starts a synchronous reception of bytes.

This function starts receiving a number of bytes in a synchronous manner.

Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
in	<i>Buffer</i>	- The buffer where the bytes will be located.
in	<i>BufferSize</i>	- The Buffer size.
in	<i>Timeout</i>	- The timeout in us.

Returns

Std\_ReturnType.

Return values

<i>E_NOT_OK</i>	If the Uart Channel is not valid or Uart driver is not initialized or Buffer is a NULL_PTR or BufferSize is 0, meaning no space has been allocated for the buffer or a wrong core has been accessed or a reception is already on going on the requested channel or timeout occurred.
<i>E_OK</i>	Successful reception.

### Precondition

Uart\_Init function must be called before this API.

#### 6.1.4.8 Uart\_Abort()

```
Std_ReturnType Uart_Abort (
    uint8 Channel,
    Uart_DataDirectionType TransmissionType )
```

Aborts an on-going transfer.

This function aborts either a reception or a transmission depending on the last parameter.

### Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
in	<i>TransmissionType</i>	- Type of the transfer to be aborted. It can be either UART_SEND or UART_RECEIVE.

### Returns

Std\_ReturnType.

### Return values

<i>E_NOT_OK</i>	If the Uart Channel is not valid or Uart driver is not initialized or a wrong core has been accessed
<i>E_OK</i>	Successful transfer aborted or in case no transfer was on going.

### Precondition

Uart\_Init function must be called before this API.

#### 6.1.4.9 Uart\_AsyncSend()

```
Std_ReturnType Uart_AsyncSend (
    uint8 Channel,
    const uint8 * Buffer,
    uint32 BufferSize )
```

Starts an asynchronous transfer(send) of bytes.

This function starts sending a number of bytes in an asynchronous manner. The transfer can be performed using either DMA or interrupts depending on the transfer type configured on the addressed channel.

## Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
in	<i>Buffer</i>	- The buffer where the data to be sent is located.
in	<i>BufferSize</i>	- The Buffer size.

## Returns

Std\_ReturnType.

## Return values

<i>E_NOT_OK</i>	If the Uart Channel is not valid or Uart driver is not initialized or Buffer is a NULL_PTR or BufferSize is 0, meaning no space has been allocated for the buffer or a wrong core has been accessed or a transfer(send) is already on going on the requested channel.
<i>E_OK</i>	The transfer(send) started successfully.

## Precondition

Uart\_Init function must be called before this API.

**6.1.4.10 Uart\_AsyncReceive()**

```
Std_ReturnType Uart_AsyncReceive (
    uint8 Channel,
    uint8 * Buffer,
    uint32 BufferSize )
```

Starts an asynchronous transfer(receive) of bytes.

This function starts receiving a number of bytes in an asynchronous manner. The transfer can be performed using either DMA or interrupts depending on the transfer type configured on the addressed channel.

## Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
in	<i>Buffer</i>	- The buffer where the data to be received will located.
in	<i>BufferSize</i>	- The Buffer size.

## Returns

Std\_ReturnType.



Return values

<i>E_NOT_OK</i>	If the Uart Channel is not valid or Uart driver is not initialized or Buffer is a NULL_PTR or BufferSize is 0, meaning no space has been allocated for the buffer or a wrong core has been accessed or a transfer(receive) is already on going on the requested channel.
<i>E_OK</i>	The transfer(receive) started successfully.

Precondition

Uart\_Init function must be called before this API.

### 6.1.4.11 Uart\_GetStatus()

```
Uart_StatusType Uart_GetStatus (
    uint8 Channel,
    uint32 * BytesTransferred,
    Uart_DataDirectionType TransferType )
```

Returns the status of the previous transfer.

This function returns the status of the previous transfer. If there is a transfer in progress, this function will also get the number of remaining bytes at the time the function was called.

Parameters

in	<i>Channel</i>	- Uart channel to be addressed.
out	<i>BytesTransferred</i>	- A pointer where the number of remaining bytes will be written.
in	<i>TransferType</i>	- The type of trasfer in discussion (UART_SEND or UART_RECEIVE).

Returns

Uart\_StatusType.

Return values

<i>UART_STATUS_NO_ERROR</i>	- Operation has ended successfully.
<i>UART_STATUS_FRAMING_ERROR</i>	- Operation has had a framing error. This status is returned only if the TransferType parameter is RECEIVE.
<i>UART_STATUS_RX_OVERRUN_ERROR</i>	- Operation has had an overrun error. This status is returned only if the TransferType parameter is RECEIVE.
<i>UART_STATUS_PARITY_ERROR</i>	- Operation has had a parity error. This status is returned only if the TransferType parameter is RECEIVE.
<i>UART_STATUS_OPERATION_ONGOING</i>	- Operation has not finished at the moment of function call.
<i>UART_STATUS_ABORTED</i>	- Operation has been aborted.

## Return values

<i>UART_STATUS_TIMEOUT</i>	- Operation has had timeout in synchronous transfer functions with timeout value pass in Timeout parameter or functions that use the loop function whose execution time exceeds the timeout value configured through the UI
----------------------------	---

## Precondition

Uart\_Init function must be called before this API.

## 6.2 Flexio UART IPL

### 6.2.1 Detailed Description

#### Data Structures

- struct [Flexio\\_Uart\\_Ip\\_UserConfigType](#)  
*Driver configuration structure. [More...](#)*

#### Macros

- `#define FLEXIO_INSTANCE_COUNT`
- `#define FLEXIO_HW_INSTANCE`

#### Enum Reference

- enum [Flexio\\_Uart\\_IP\\_DriverType](#)  
*Driver type: Interrupts/DMA Implements : Flexio\_Uart\_IP\_DriverType\_Class.*
- enum [Flexio\\_Uart\\_Ip\\_DirectionType](#)  
*flexio\_uart driver direction (tx or rx)*
- enum [Flexio\\_Uart\\_Ip\\_EventType](#)  
*Define the enum of the Events which can trigger UART callback.*
- enum [Flexio\\_Uart\\_Ip\\_StatusType](#)
- enum [Flexio\\_Uart\\_Ip\\_BaudrateType](#)  
*Define the enum of the baudrate values that should be set for Uart.*
- enum [Flexio\\_Uart\\_Ip\\_TimerDecrementType](#)  
*FLEXIO Timer decrement options.*
- enum [Flexio\\_Uart\\_Ip\\_BitCountPerCharType](#)  
*FLEXIO number of bits in a character.*

#### Variables

- [Flexio\\_Uart\\_Ip\\_StateStructureType](#)  
*Driver internal context structure.*

### 6.2.2 Data Structure Documentation

#### 6.2.2.1 struct Flexio\_Uart\_Ip\_UserConfigType

Driver configuration structure.

This structure is used to provide configuration parameters for the flexio\_uart driver at initialization time. Implements : `Flexio_Uart_Ip_UserConfigType_Class`

Definition at line 245 of file `Flexio_Uart_Ip_Types.h`.

## Data Fields

Type	Name	Description
uint32	Channel	Flexio Uart Channel has been configured. Note that Make sure the Channel is used in all API corresponds to this parameter.
<a href="#">Flexio_Uart_IP_DriverType</a>	DriverType	Driver type: Interrupts/DMA.
uint8	Divider	Baudrate divider.
<a href="#">Flexio_Uart_Ip_TimerDecrementType</a>	TimerDec	The source of the Timer decrement and the source of the Shift clock.
uint32	BaudRate	Baud rate in hertz.
<a href="#">Flexio_Uart_Ip_BitCountPerCharType</a>	BitCount	Number of bits per word.
<a href="#">Flexio_Uart_Ip_DirectionType</a>	Direction	Driver direction: Tx or Rx.
uint8	DataPin	Flexio pin to use as Tx or Rx pin.
<a href="#">Flexio_Uart_Ip_CallbackType</a>	Callback	User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL_PTR if it is not needed.
void *	CallbackParam	Parameter for the callback function.
<a href="#">Flexio_Uart_Ip_StateStructureType</a> *	StateStruct	

## 6.2.3 Macro Definition Documentation

### 6.2.3.1 FLEXIO\_INSTANCE\_COUNT

```
#define FLEXIO_INSTANCE_COUNT
```

Number of instances of the FLEXIO module.

Definition at line 91 of file `Flexio_Uart_Ip_Types.h`.

### 6.2.3.2 FLEXIO\_HW\_INSTANCE

```
#define FLEXIO_HW_INSTANCE
```

FLEXIO Instance support .

Definition at line 94 of file `Flexio_Uart_Ip_Types.h`.

## 6.2.4 Enum Reference

### 6.2.4.1 Flexio\_Uart\_IP\_DriverType

```
enum Flexio_Uart_IP_DriverType
```

Driver type: Interrupts/DMA Implements : `Flexio_Uart_IP_DriverType_Class`.

Enumerator

FLEXIO_UART_IP_DRIVER_TYPE_INTERRUPTS	Driver uses interrupts for data transfers
FLEXIO_UART_IP_DRIVER_TYPE_DMA	Driver uses DMA for data transfers

Definition at line 102 of file Flexio\_Uart\_Ip\_Types.h.

### 6.2.4.2 Flexio\_Uart\_Ip\_DirectionType

enum `Flexio_Uart_Ip_DirectionType`

flexio\_uart driver direction (tx or rx)

This structure describes the direction configuration options for the flexio\_uart driver. Implements : flexio\_uart\_↔ driver\_direction\_t\_Class

Enumerator

FLEXIO_UART_IP_DIRECTION_TX	Tx UART driver
FLEXIO_UART_IP_DIRECTION_RX	Rx UART driver

Definition at line 114 of file Flexio\_Uart\_Ip\_Types.h.

### 6.2.4.3 Flexio\_Uart\_Ip\_EventType

enum `Flexio_Uart_Ip_EventType`

Define the enum of the Events which can trigger UART callback.

This enum should include the Events for all platforms

Enumerator

FLEXIO_UART_IP_EVENT_RX_FULL	Rx buffer is full.
FLEXIO_UART_IP_EVENT_TX_EMPTY	Tx buffer is empty.
FLEXIO_UART_IP_EVENT_END_TRANSFER	The current transfer is ending.
FLEXIO_UART_IP_EVENT_ERROR	An error occurred during transfer.

Definition at line 127 of file Flexio\_Uart\_Ip\_Types.h.

#### 6.2.4.4 Flexio\_Uart\_Ip\_StatusType

enum `Flexio_Uart_Ip_StatusType`

Implements : Driver status type.

Enumerator

<code>FLEXIO_UART_IP_STATUS_SUCCESS</code>	Operation has been successfully
<code>FLEXIO_UART_IP_STATUS_ERROR</code>	Operation has had error
<code>FLEXIO_UART_IP_STATUS_BUSY</code>	Function is called during an on-going transfer
<code>FLEXIO_UART_IP_STATUS_DMA_ERROR</code>	Operation has had DMA error
<code>FLEXIO_UART_IP_STATUS_TX_UNDERRUN</code>	TX underrun error
<code>FLEXIO_UART_IP_STATUS_RX_OVERRUN</code>	RX overrun error
<code>FLEXIO_UART_IP_STATUS_ABORTED</code>	A transfer was aborted

Definition at line 138 of file `Flexio_Uart_Ip_Types.h`.

#### 6.2.4.5 Flexio\_Uart\_Ip\_BaudrateType

enum `Flexio_Uart_Ip_BaudrateType`

Define the enum of the baudrate values that should be set for Uart.

Definition at line 156 of file `Flexio_Uart_Ip_Types.h`.

#### 6.2.4.6 Flexio\_Uart\_Ip\_TimerDecrementType

enum `Flexio_Uart_Ip_TimerDecrementType`

FLEXIO Timer decrement options.

Enumerator

<code>FLEXIO_TIMER_DECREMENT_FXIO_CLK_↔ SHIFT_TMR</code>	Decrement counter on FlexIO clock, Shift clock equals Timer output.
<code>FLEXIO_TIMER_DECREMENT_FXIO_CLK_↔ DIV_16</code>	Decrement counter on FlexIO clock divided by 16, Shift clock equals Timer output.
<code>FLEXIO_TIMER_DECREMENT_FXIO_CLK_↔ DIV_256</code>	Decrement counter on FlexIO clock divided by 256, Shift clock equals Timer output.

Definition at line 180 of file Flexio\_Uart\_Ip\_Types.h.

6.2.4.7 Flexio\_Uart\_Ip\_BitCountPerCharType

enum Flexio\_Uart\_Ip\_BitCountPerCharType

FLEXIO number of bits in a character.

Enumerator

FLEXIO_UART_IP_8_BITS_PER_CHAR	8-bit data characters
--------------------------------	-----------------------

Definition at line 191 of file Flexio\_Uart\_Ip\_Types.h.

6.2.5 Variable Documentation

6.2.5.1 Flexio\_Uart\_Ip\_StateStructureType

Flexio\_Uart\_Ip\_StateStructureType

Driver internal context structure.

This structure is used by the flexio\_uart driver for its internal logic. It must be provided by the application through the Flexio\_Uart\_Ip\_Init() function, then it cannot be freed until the driver is de-initialized using Flexio\_Uart\_Ip\_Deinit(). The application should make no assumptions about the content of this structure.

Definition at line 237 of file Flexio\_Uart\_Ip\_Types.h.

## 6.3 Lpuart UART IPL

### 6.3.1 Detailed Description

#### Data Structures

- struct [Lpuart\\_Uart\\_Ip\\_StateStructureType](#)  
*Runtime of the LPUART driver. [More...](#)*
- struct [Lpuart\\_Uart\\_Ip\\_UserConfigType](#)  
*LPUART configuration structure. [More...](#)*

#### Types Reference

- typedef void(\* [Lpuart\\_Uart\\_Ip\\_CallbackType](#)) (const uint8 HwInstance, const [Lpuart\\_Uart\\_Ip\\_EventType](#) Event, void \*UserData)  
*Callback for all peripherals which support UART features.*

#### Enum Reference

- enum [Lpuart\\_Uart\\_Ip\\_TransferType](#)  
*Type of UART transfer (based on interrupts or DMA).*
- enum [Lpuart\\_Uart\\_Ip\\_StatusType](#)  
*Driver status type.*
- enum [Lpuart\\_Uart\\_Ip\\_EventType](#)  
*Define the enum of the Events which can trigger UART callback.*
- enum [Lpuart\\_Uart\\_Ip\\_BaudrateType](#)  
*Define the enum of the baudrate values that should be set for Uart.*

#### Function Reference

- void [Lpuart\\_Uart\\_Ip\\_Init](#) (const uint8 Instance, const [Lpuart\\_Uart\\_Ip\\_UserConfigType](#) \*UserConfig)  
*Initializes an LPUART operation instance.*
- [Lpuart\\_Uart\\_Ip\\_StatusType](#) [Lpuart\\_Uart\\_Ip\\_Deinit](#) (const uint8 Instance)  
*Shuts down the LPUART by disabling interrupts and transmitter/receiver.*
- [Lpuart\\_Uart\\_Ip\\_StatusType](#) [Lpuart\\_Uart\\_Ip\\_SyncSend](#) (const uint8 Instance, const uint8 \*TxBuff, const uint32 TxSize, const uint32 Timeout)  
*Send out multiple bytes of data using polling method.*
- [Lpuart\\_Uart\\_Ip\\_StatusType](#) [Lpuart\\_Uart\\_Ip\\_AsyncSend](#) (const uint8 Instance, const uint8 \*TxBuff, const uint32 TxSize)  
*Sends data out through the LPUART module using a non-blocking method. This enables an a-sync method for transmitting data. When used with a non-blocking receive, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete.*



- [Lpuart\\_Uart\\_Ip\\_StatusType Lpuart\\_Uart\\_Ip\\_GetTransmitStatus](#) (const uint8 Instance, uint32 \*BytesRemaining)  
*Returns whether the previous transmit is complete.*
- [Lpuart\\_Uart\\_Ip\\_StatusType Lpuart\\_Uart\\_Ip\\_AbortSendingData](#) (const uint8 Instance)  
*Terminates a non-blocking transmission early.*
- [Lpuart\\_Uart\\_Ip\\_StatusType Lpuart\\_Uart\\_Ip\\_SyncReceive](#) (const uint8 Instance, uint8 \*RxBuff, const uint32 RxSize, const uint32 Timeout)  
*Receive multiple bytes of data using polling method.*
- [Lpuart\\_Uart\\_Ip\\_StatusType Lpuart\\_Uart\\_Ip\\_AsyncReceive](#) (const uint8 Instance, uint8 \*RxBuff, const uint32 RxSize)  
*Gets data from the LPUART module by using a non-blocking method. This enables an a-sync method for receiving data. When used with a non-blocking transmission, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the receive is complete.*
- [Lpuart\\_Uart\\_Ip\\_StatusType Lpuart\\_Uart\\_Ip\\_GetReceiveStatus](#) (const uint8 Instance, uint32 \*BytesRemaining)  
*Returns whether the previous receive is complete.*
- [Lpuart\\_Uart\\_Ip\\_StatusType Lpuart\\_Uart\\_Ip\\_AbortReceivingData](#) (const uint8 Instance)  
*Terminates a non-blocking receive early.*
- [Lpuart\\_Uart\\_Ip\\_StatusType Lpuart\\_Uart\\_Ip\\_SetBaudRate](#) (const uint8 Instance, const [Lpuart\\_Uart\\_Ip\\_BaudrateType](#) DesiredBaudrate, const uint32 ClockFrequency)  
*Configures the LPUART baud rate.*
- void [Lpuart\\_Uart\\_Ip\\_GetBaudRate](#) (const uint8 Instance, uint32 \*ConfiguredBaudRate)  
*Returns the LPUART baud rate.*
- void [Lpuart\\_Uart\\_Ip\\_SetTxBuffer](#) (const uint8 Instance, const uint8 \*TxBuff, const uint32 TxSize)  
*Sets the internal driver reference to the tx buffer.*
- void [Lpuart\\_Uart\\_Ip\\_SetRxBuffer](#) (const uint8 Instance, uint8 \*RxBuff, const uint32 RxSize)  
*Sets the internal driver reference to the rx buffer.*

### 6.3.2 Data Structure Documentation

#### 6.3.2.1 struct Lpuart\_Uart\_Ip\_StateStructureType

Runtime of the LPUART driver.

Note that the caller provides memory for the driver structures during initialization because the driver does not statically allocate memory.

Implements : [Lpuart\\_Uart\\_Ip\\_StateStructureType](#)

Definition at line 194 of file Lpuart\_Uart\_Ip\_Types.h.

Data Fields

	Type	Name	Description
	uint32	BaudRate	Variable that indicates if structure belongs to an instance already initialized.
	const uint8 *	TxBuff	The buffer of data being sent.

## Data Fields

Type	Name	Description
uint8 *	RxBuff	The buffer of received data.
volatile uint32	TxSize	The remaining number of bytes to be transmitted.
volatile uint32	RxSize	The remaining number of bytes to be received.
volatile boolean	IsTxBusy	True if there is an active transmit.
volatile boolean	IsRxBusy	True if there is an active receive.
volatile <a href="#">Lpuart_Uart_Ip_StatusType</a>	TransmitStatus	Status of last driver transmit operation.
volatile <a href="#">Lpuart_Uart_Ip_StatusType</a>	ReceiveStatus	Status of last driver receive operation.

**6.3.2.2 struct Lpuart\_Uart\_Ip\_UserConfigType**

LPUART configuration structure.

Implements : [Lpuart\\_Uart\\_Ip\\_UserConfigType](#)

Definition at line 212 of file [Lpuart\\_Uart\\_Ip\\_Types.h](#).

## Data Fields

Type	Name	Description
uint32	BaudRate	Baudrate value.
uint32	BaudRateDivisor	Baud clock divisor.
uint8	BaudOverSamplingRatio	Over sampling ratio.
<a href="#">Lpuart_Uart_Ip_ParityModeType</a>	ParityMode	Parity mode, disabled (default), even, odd.
<a href="#">Lpuart_Uart_Ip_StopBitCountType</a>	StopBitsCount	Number of stop bits, 1 stop bit (default) or 2 stop bits.
<a href="#">Lpuart_Uart_Ip_BitCountPerCharType</a>	BitCountPerChar	Number of bits in a character (8-default, 9 or 10); for 9/10 bits chars, users must provide appropriate buffers to the send/receive functions (bits 8/9 in subsequent bytes); for DMA transmission only 8-bit char is supported.
<a href="#">Lpuart_Uart_Ip_TransferType</a>	TransferType	@briefType of LPUART transfer (interrupt/dma based)
<a href="#">Lpuart_Uart_Ip_CallbackType</a>	Callback	Callback to invoke for handle uart event.
void *	CallbackParam	User callback parameter pointer.
<a href="#">Lpuart_Uart_Ip_StateStructureType</a> *	StateStruct	

**6.3.3 Types Reference**

### 6.3.3.1 Lpuart\_Uart\_Ip\_CallbackType

```
typedef void(* Lpuart_Uart_Ip_CallbackType) (const uint8 HwInstance, const Lpuart_Uart_Ip_EventType Event, void *UserData)
```

Callback for all peripherals which support UART features.

Definition at line 181 of file Lpuart\_Uart\_Ip\_Types.h.

### 6.3.4 Enum Reference

#### 6.3.4.1 Lpuart\_Uart\_Ip\_TransferType

```
enum Lpuart_Uart_Ip_TransferType
```

Type of UART transfer (based on interrupts or DMA).

Enumerator

LPUART_UART_IP_USING_DMA	The driver will use DMA to perform UART transfer.
LPUART_UART_IP_USING_INTERRUPTS	The driver will use interrupts to perform UART transfer.

Definition at line 99 of file Lpuart\_Uart\_Ip\_Types.h.

#### 6.3.4.2 Lpuart\_Uart\_Ip\_StatusType

```
enum Lpuart_Uart_Ip_StatusType
```

Driver status type.

Enumerator

LPUART_UART_IP_STATUS_SUCCESS	Generic operation success status.
LPUART_UART_IP_STATUS_ERROR	Generic operation failure status.
LPUART_UART_IP_STATUS_BUSY	Generic operation busy status.
LPUART_UART_IP_STATUS_TIMEOUT	Generic operation timeout status.
LPUART_UART_IP_STATUS_TX_UNDERRUN	TX underrun error.
LPUART_UART_IP_STATUS_RX_OVERRUN	RX overrun error.
LPUART_UART_IP_STATUS_ABORTED	<ul style="list-style-type: none"> <li>A transfer was aborted</li> </ul>
LPUART_UART_IP_STATUS_FRAMING_ERROR	Framing error.
LPUART_UART_IP_STATUS_PARITY_ERROR	Parity error.
LPUART_UART_IP_STATUS_NOISE_ERROR	Noise error.
LPUART_UART_IP_STATUS_DMA_ERROR	DMA error.

Definition at line 111 of file Lpuart\_Uart\_Ip\_Types.h.

#### 6.3.4.3 Lpuart\_Uart\_Ip\_EventType

```
enum Lpuart_Uart_Ip_EventType
```

Define the enum of the Events which can trigger UART callback.

This enum should include the Events for all platforms

Enumerator

LPUART_UART_IP_EVENT_RX_FULL	Rx buffer is full.
LPUART_UART_IP_EVENT_TX_EMPTY	Tx buffer is empty.
LPUART_UART_IP_EVENT_END_TRANSFER	The current transfer is ending.
LPUART_UART_IP_EVENT_ERROR	An error occurred during transfer.

Definition at line 137 of file Lpuart\_Uart\_Ip\_Types.h.

#### 6.3.4.4 Lpuart\_Uart\_Ip\_BaudrateType

```
enum Lpuart_Uart_Ip_BaudrateType
```

Define the enum of the baudrate values that should be set for Uart.

Definition at line 153 of file Lpuart\_Uart\_Ip\_Types.h.

### 6.3.5 Function Reference

#### 6.3.5.1 Lpuart\_Uart\_Ip\_Init()

```
void Lpuart_Uart_Ip_Init (
    const uint8 Instance,
    const Lpuart_Uart_Ip_UserConfigType * UserConfig )
```

Initializes an LPUART operation instance.

The caller provides memory for the driver state structures during initialization. The user must select the LPUART clock source in the application to initialize the LPUART.

### Parameters

<i>Instance</i>	LPUART instance number
<i>UserConfig</i>	user configuration structure of type <a href="#">Lpuart_Uart_Ip_UserConfigType</a>

### Returns

void

#### 6.3.5.2 Lpuart\_Uart\_Ip\_Deinit()

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_Deinit (
    const uint8 Instance )
```

Shuts down the LPUART by disabling interrupts and transmitter/receiver.

### Parameters

<i>Instance</i>	LPUART instance number
-----------------	------------------------

### Returns

LPUART\_UART\_IP\_STATUS\_SUCCESS if successful; LPUART\_UART\_IP\_STATUS\_ERROR if the progress has not fully completed;

#### 6.3.5.3 Lpuart\_Uart\_Ip\_SyncSend()

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_SyncSend (
    const uint8 Instance,
    const uint8 * TxBuff,
    const uint32 TxSize,
    const uint32 Timeout )
```

Send out multiple bytes of data using polling method.

### Parameters

<i>Instance</i>	LPUART instance number.
<i>TxBuff</i>	The buffer pointer which saves the data to be sent.
<i>TxSize</i>	Size of data to be sent in unit of byte.
<i>Timeout</i>	value in microseconds.

## Returns

LPUART\_UART\_IP\_STATUS\_SUCCESS if successful; LPUART\_UART\_IP\_STATUS\_BUSY if the resource is busy; LPUART\_UART\_IP\_STATUS\_TIMEOUT if timeout occur

#### 6.3.5.4 Lpuart\_Uart\_Ip\_AsyncSend()

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_AsyncSend (
    const uint8 Instance,
    const uint8 * TxBuff,
    const uint32 TxSize )
```

Sends data out through the LPUART module using a non-blocking method. This enables an a-sync method for transmitting data. When used with a non-blocking receive, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete.

## Parameters

<i>Instance</i>	LPUART instance number.
<i>TxBuff</i>	The buffer pointer which saves the data to be sent.
<i>TxSize</i>	Size of data to be sent in unit of byte.

## Returns

LPUART\_UART\_IP\_STATUS\_SUCCESS if successful; LPUART\_UART\_IP\_STATUS\_BUSY if the resource is busy;

#### 6.3.5.5 Lpuart\_Uart\_Ip\_GetTransmitStatus()

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_GetTransmitStatus (
    const uint8 Instance,
    uint32 * BytesRemaining )
```

Returns whether the previous transmit is complete.

## Parameters

<i>Instance</i>	LPUART instance number
<i>BytesRemaining</i>	Pointer to value that is populated with the number of bytes that have been sent in the active transfer

Note

In DMA mode, this parameter may not be accurate, in case the transfer completes right after calling this function; in this edge-case, the parameter will reflect the initial transfer size, due to automatic reloading of the major loop count in the DMA transfer descriptor.

Returns

The transmit status.

Return values

<i>LPUART_UART_IP_STATUS_SUCCESS</i>	The transmit has completed successfully.
<i>LPUART_UART_IP_STATUS_BUSY</i>	The transmit is still in progress. <i>bytesRemaining</i> will be filled with the number of bytes that are yet to be transmitted.
<i>LPUART_UART_IP_STATUS_ABORTED</i>	The transmit was aborted.
<i>LPUART_UART_IP_STATUS_TIMEOUT</i>	A timeout was reached.
<i>LPUART_UART_IP_STATUS_ERROR</i>	An error occurred.

6.3.5.6 Lpuart\_Uart\_Ip\_AbortSendingData()

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_AbortSendingData (
    const uint8 Instance )
```

Terminates a non-blocking transmission early.

Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

Returns

LPUART\_UART\_IP\_STATUS\_ERROR if the transmit process has not fully completed, LPUART\_UART\_IP\_STATUS\_SUCCESS if the transmit process has successfully completed

6.3.5.7 Lpuart\_Uart\_Ip\_SyncReceive()

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_SyncReceive (
    const uint8 Instance,
    uint8 * RxBuff,
    const uint32 RxSize,
    const uint32 Timeout )
```

Receive multiple bytes of data using polling method.

## Parameters

<i>Instance</i>	LPUART instance number.
<i>RxBuff</i>	The buffer pointer which saves the data to be received.
<i>RxSize</i>	Size of data need to be received in unit of byte.
<i>Timeout</i>	value in microseconds.

## Returns

LPUART\_UART\_IP\_STATUS\_SUCCESS if the transaction is successful; LPUART\_UART\_IP\_STATUS\_BUSY if the resource is busy; LPUART\_UART\_IP\_STATUS\_RX\_OVERRUN if an overrun error occurred LPUART\_UART\_IP\_STATUS\_FRAMING\_ERROR if a framing error occurred LPUART\_UART\_IP\_STATUS\_PARITY\_ERROR if a parity error occurred LPUART\_UART\_IP\_STATUS\_NOISE\_ERROR if a noise error occurred LPUART\_UART\_IP\_STATUS\_TIMEOUT if timeout occur

**6.3.5.8 Lpuart\_Uart\_Ip\_AsyncReceive()**

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_AsyncReceive (
    const uint8 Instance,
    uint8 * RxBuff,
    const uint32 RxSize )
```

Gets data from the LPUART module by using a non-blocking method. This enables an a-sync method for receiving data. When used with a non-blocking transmission, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the receive is complete.

## Parameters

<i>Instance</i>	LPUART instance number
<i>RxBuff</i>	buffer containing 8-bit read data chars received
<i>RxSize</i>	the number of bytes to receive


## Returns

LPUART\_UART\_IP\_STATUS\_SUCCESS if successful; LPUART\_UART\_IP\_STATUS\_BUSY if the resource is busy

**6.3.5.9 Lpuart\_Uart\_Ip\_GetReceiveStatus()**

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_GetReceiveStatus (
    const uint8 Instance,
    uint32 * BytesRemaining )
```





## Module Documentation

Returns whether the previous receive is complete.

## Parameters

<i>Instance</i>	LPUART instance number
<i>BytesRemaining</i>	pointer to value that is filled with the number of bytes that still need to be received in the active transfer.

## Note

In DMA mode, this parameter may not be accurate, in case the transfer completes right after calling this function; in this edge-case, the parameter will reflect the initial transfer size, due to automatic reloading of the major loop count in the DMA transfer descriptor.

## Returns

The receive status.

## Return values

<i>LPUART_UART_IP_STATUS_SUCCESS</i>	the receive has completed successfully.
<i>LPUART_UART_IP_STATUS_BUSY</i>	the receive is still in progress. <i>bytesReceived</i> will be filled with the number of bytes that have been received so far.
<i>LPUART_UART_IP_STATUS_ABORTED</i>	The receive was aborted.
<i>LPUART_UART_IP_STATUS_TIMEOUT</i>	A timeout was reached.
<i>LPUART_UART_IP_STATUS_RX_OVERRUN, LPUART_UART_IP_STATUS_FRAMING_ERROR, LPUART_UART_IP_STATUS_PARITY_ERROR, or</i>	<i>LPUART_UART_IP_STATUS_NOISE_ERROR, LPUART_UART_IP_STATUS_ERROR</i> An error occurred during reception.

**6.3.5.10 Lpuart\_Uart\_Ip\_AbortReceivingData()**

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_AbortReceivingData (
    const uint8 Instance )
```

Terminates a non-blocking receive early.

## Parameters

<i>Instance</i>	LPUART instance number
-----------------	------------------------

## Returns

*LPUART\_UART\_IP\_STATUS\_ERROR* if the receive process has not fully completed, *LPUART\_UART\_IP\_STATUS\_SUCCESS* if the receive process has successfully completed

6.3.5.11 Lpuart\_Uart\_Ip\_SetBaudRate()

```
Lpuart_Uart_Ip_StatusType Lpuart_Uart_Ip_SetBaudRate (
    const uint8 Instance,
    const Lpuart_Uart_Ip_BaudrateType DesiredBaudrate,
    const uint32 ClockFrequency )
```

Configures the LPUART baud rate.

This function configures the LPUART baud rate. In some LPUART instances the user must disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>Instance</i>	LPUART instance number.
<i>DesiredBaudrate</i>	LPUART desired baud rate.
<i>ClockFrequency</i>	Clock Frequency of LPUART instance.

Returns

LPUART\_UART\_IP\_STATUS\_BUSY if called during an on-going transfer, LPUART\_UART\_IP\_STATUS\_SUCCESS otherwise

6.3.5.12 Lpuart\_Uart\_Ip\_GetBaudRate()

```
void Lpuart_Uart_Ip_GetBaudRate (
    const uint8 Instance,
    uint32 * ConfiguredBaudRate )
```

Returns the LPUART baud rate.

This function returns the LPUART configured baud rate.

Parameters

	<i>Instance</i>	LPUART instance number.
out	<i>ConfiguredBaudRate</i>	LPUART configured baud rate.

**6.3.5.13 Lpuart\_Uart\_Ip\_SetTxBuffer()**

```
void Lpuart_Uart_Ip_SetTxBuffer (
    const uint8 Instance,
    const uint8 * TxBuff,
    const uint32 TxSize )
```

Sets the internal driver reference to the tx buffer.

This function can be called from the tx callback to provide the driver with a new buffer, for continuous transmission.

Parameters

<i>Instance</i>	LPUART instance number
<i>TxBuff</i>	source buffer containing 8-bit data chars to send
<i>TxSize</i>	the number of bytes to send

Returns

void

**6.3.5.14 Lpuart\_Uart\_Ip\_SetRxBuffer()**

```
void Lpuart_Uart_Ip_SetRxBuffer (
    const uint8 Instance,
    uint8 * RxBuff,
    const uint32 RxSize )
```

Sets the internal driver reference to the rx buffer.

This function can be called from the rx callback to provide the driver with a new buffer, for continuous reception.

Parameters

<i>instance</i>	LPUART instance number
<i>RxBuff</i>	destination buffer containing 8-bit data chars to receive
<i>RxSize</i>	the number of bytes to receive

Returns

void

**How to Reach Us:**

**Home Page:**

[nxp.com](http://nxp.com)

**Web Support:**

[nxp.com/support](http://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](http://nxp.com/SalesTermsandConditions).

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. ARM, AMBA, ARM Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and Vision are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7, ARM9, ARM11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. 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.

© 2023 NXP B.V.

