User Manual

for S32K3 UART Driver

Document Number: UM34UARTASRR21-11 Rev0000R 3.0.0 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	31.03.2023	NXP RTD Team	S32K3 Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 3.0.0

Chapter 2

Introduction

This User Manual describes NXP Semiconductor CDD UART for S32K3XX and S32M27X. 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:

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100
- s32k322_mqfp172 / MWCT2D16S_mqfp172
- s32k324_mqfp172 / MWCT2D17S_mqfp172

- s32k324_mapbga257
- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

All of the above microcontroller devices are collectively named as S32K3.

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 Signifigant 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	S32K3XX Reference Manual	Rev.6, Draft B, 01/2023
2	S32K3xx Data Sheet	Rev. 6, 11/2022
3	S32K39 and S32K37 Reference Manual	Rev.2, Draft A, — 11/2022
4	S32K396 Data Sheet	Rev. 1.1 — 08/2022
5	S32M27x Reference Manual	Rev.2, Draft A, — 02/2023
6	S32M2xx Data Sheet	Rev. 2 RC — 12/2022
7	S32K358_0P14E Mask Set Errata	Rev. 28, 9/2022
8	S32K396_0P40E Mask Set Errata	Rev. DEC2022, 12/2022
9	S32K311_0P98C Mask Set Errata	Rev. 6/March/2023, 3/2023
10	S32K312: Mask Set Errata for Mask 0P09C	Rev. 25/April/2022

#	Title	Version
11	S32K342: Mask Set Errata for Mask 0P97C	Rev. 10, 11/2022
12	S32K3x4: Mask Set Errata for Mask 0P55A/1P55A	Rev. 14/Oct/2022

Chapter 3

Driver

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 S32K3XX and S32M27X 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
- Usage in a multicore enviroment
- 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_3 with S32K311.
- From LPUART_0 to LPUART_15 with S32K358.
- From LPUART_0 to LPUART_3 and LPUART_MSC with S32K396.
- From LPUART_0 to LPUART_15 with S32K388.
- From LPUART_0 to LPUART_3 with S32M274.

The Flexio Channel configured by the Uart driver:

- From FLEXIO_0 to FLEXIO_7 with S32K311.
- From FLEXIO_0 to FLEXIO_7 with S32K358.
- From FLEXIO_0 to FLEXIO_7 with S32K396.
- From FLEXIO_0 to FLEXIO_7 with S32K388.
- From FLEXIO_0 to FLEXIO_7 with S32M274.

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

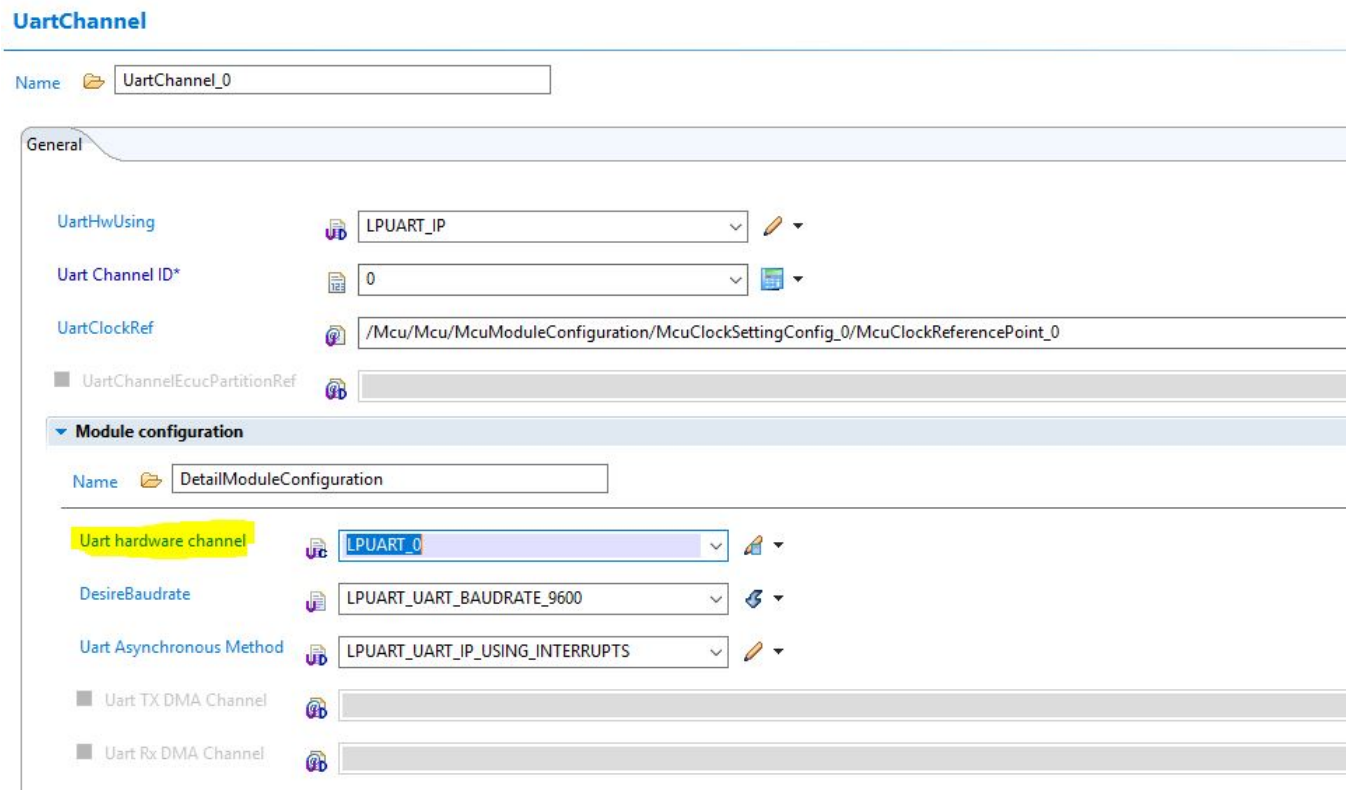

Figure 3.1 Uart Physical Unit has selected by UartHwChannel in EB Tresos

Flexio Uart Physical Channel has selected by MCL module at Mcl/MclConfig/FlexioCommon/FlexioCommon_↩ 0/FlexioMclLogicChannels path

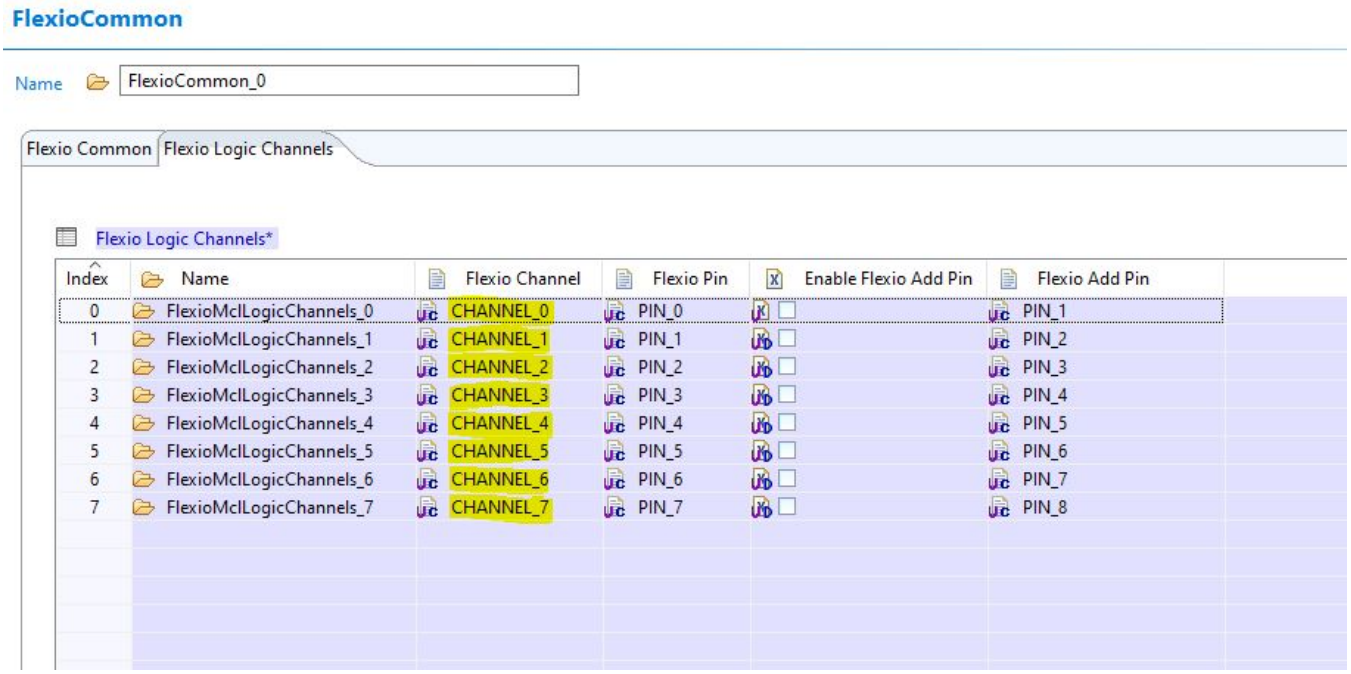

Figure 3.2 Flexio Uart Physical Channel has selected by MCL module in EB Tresos

3.3.2 Uart pins: The pins of S32K311 can be found out in the file "S32K311_S32K310_IOMUX.xlsx" from attached file of S32K3XX Reference Manual Reference List. The Lpuart_0,..., Lpuart_3 use the pins LPUART0,..., LPUART3 correspondingly. The pins of S32K358 can be found out in the file "S32K358_S32K48_S32K38_S32↩ K328_IOMUX.xlsx" from attached file of S32K3XX Reference Manual Reference List. The Lpuart_0,..., Lpuart_15 use the pins LPUART0,..., LPUART15 correspondingly. The pins of S32K396 can be found out in the file "S32↩ K396_S32K37_IOMUX.xlsx" from attached file of S32K39 and S32K37 Reference Manual List. The Lpuart_0,..., Lpuart_3 use the pins LPUART0,..., LPUART3 correspondingly. The pins of S32M276 can be found out in the file "S32M27x_IOMUX.xlsx" from attached file of S32M27x Reference Manual Reference List. The Lpuart_0,..., Lpuart_3 use the pins LPUART0,..., LPUART3 correspondingly.

Po	CR	SSS	Function	Module	Description	rection
PTA0	SIUL_IMCR872	0000_0001	LPUART0_CTS	LPUART0	Clear To Send (bar)	I
PTA1		0000_0011	LPUART0_RTS	LPUART0	Request To Send	O
PTA2	SIUL_IMCR699	0000_0001	LPUART0_RX	LPUART0	Receive	I
PTA3		0000_0110	LPUART0_TX	LPUART0	Transmit	O
PTA3	SIUL_IMCR875	0000_0001	LPUART0_TX	LPUART0	Transmit	I
PTA6	SIUL_IMCR702	0000_0010	LPUART3_RX	LPUART3	Receive	I
PTA6	SIUL_IMCR873	0000_0010	LPUART1_CTS	LPUART1	Clear To Send (bar)	I
PTA7		0000_0001	LPUART3_TX	LPUART3	Transmit	O
PTA7		0000_0101	LPUART1_RTS	LPUART1	Request To Send	O
PTA7	SIUL_IMCR878	0000_0001	LPUART3_TX	LPUART3	Transmit	I
PTA8	SIUL_IMCR701	0000_0011	LPUART2_RX	LPUART2	Receive	I
PTA9		0000_0010	LPUART2_TX	LPUART2	Transmit	O
PTA9	SIUL_IMCR877	0000_0001	LPUART2_TX	LPUART2	Transmit	I
PTA12	SIUL_IMCR710	0000_0010	LPUART11_RX	LPUART11	Receive	I
PTA13		0000_0111	LPUART11_TX	LPUART11	Transmit	O
PTA13	SIUL_IMCR886	0000_0001	LPUART11_TX	LPUART11	Transmit	I
PTA15	SIUL_IMCR705	0000_0010	LPUART6_RX	LPUART6	Receive	I
PTA16		0000_0101	LPUART6_TX	LPUART6	Transmit	O
PTA16	SIUL_IMCR881	0000_0001	LPUART6_TX	LPUART6	Transmit	I
PTA17		0000_0100	LPUART4_TX	LPUART4	Transmit	O
PTA17	SIUL_IMCR879	0000_0001	LPUART4_TX	LPUART4	Transmit	I
PTA18		0000_0011	LPUART1_TX	LPUART1	Transmit	O
PTA18	SIUL_IMCR876	0000_0100	LPUART1_TX	LPUART1	Transmit	I
PTA19	SIUL_IMCR700	0000_0101	LPUART1_RX	LPUART1	Receive	I
PTA27		0000_0100	LPUART0_TX	LPUART0	Transmit	O
PTA27	SIUL_IMCR875	0000_0100	LPUART0_TX	LPUART0	Transmit	I
PTA28	SIUL_IMCR699	0000_0100	LPUART0_RX	LPUART0	Receive	I
PTA29		0000_0100	LPUART2_TX	LPUART2	Transmit	O
PTA29	SIUL_IMCR877	0000_0101	LPUART2_TX	LPUART2	Transmit	I
PTA30	SIUL_IMCR701	0000_0100	LPUART2_RX	LPUART2	Receive	I
PTB0	SIUL_IMCR699	0000_0010	LPUART0_RX	LPUART0	Receive	I
PTB1		0000_0010	LPUART0_TX	LPUART0	Transmit	O
PTB1	SIUL_IMCR875	0000_0010	LPUART0_TX	LPUART0	Transmit	I
PTB2	SIUL_IMCR708	0000_0010	LPUART9_RX	LPUART9	Receive	I
PTB3		0000_0001	LPUART9_TX	LPUART9	Transmit	O

Figure 3.3 Pins selected for Lpuart

Each Flexio channel uses one of the pin in list FXIO_D0, FXIO_D1,..., FXIO_D31.

PTA0		0000_0100	FXIO_D2	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA0	SIUL_IMCR666	0000_0010	FXIO_D2	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA1		0000_0100	FXIO_D3	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA1	SIUL_IMCR667	0000_0001	FXIO_D3	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA2		0000_0101	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA2	SIUL_IMCR668	0000_0011	FXIO_D4	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA3		0000_0101	FXIO_D5	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA3	SIUL_IMCR669	0000_0011	FXIO_D5	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA4		0000_0011	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA4	SIUL_IMCR670	0000_1000	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA6		0000_0101	FXIO_D19	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA6	SIUL_IMCR683	0000_0100	FXIO_D19	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA7		0000_0110	FXIO_D9	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA7	SIUL_IMCR673	0000_0011	FXIO_D9	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA8		0000_0100	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA8	SIUL_IMCR670	0000_0010	FXIO_D6	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA9		0000_0100	FXIO_D7	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA9	SIUL_IMCR671	0000_0010	FXIO_D7	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA10		0000_0100	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA10	SIUL_IMCR664	0000_0010	FXIO_D0	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA11		0000_0100	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA11	SIUL_IMCR665	0000_0010	FXIO_D1	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA12		0000_0101	FXIO_D9	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA12	SIUL_IMCR673	0000_0100	FXIO_D9	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA13		0000_0101	FXIO_D8	FXIO	FlexIO Bi-directional Shift/timer I/O	O
PTA13	SIUL_IMCR672	0000_0100	FXIO_D8	FXIO	FlexIO Bi-directional Shift/timer I/O	I
PTA14		0000_0110	FXIO_D14	FXIO	FlexIO Bi-directional Shift/timer I/O	O

Figure 3.4 Pins selected for Flexio

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".

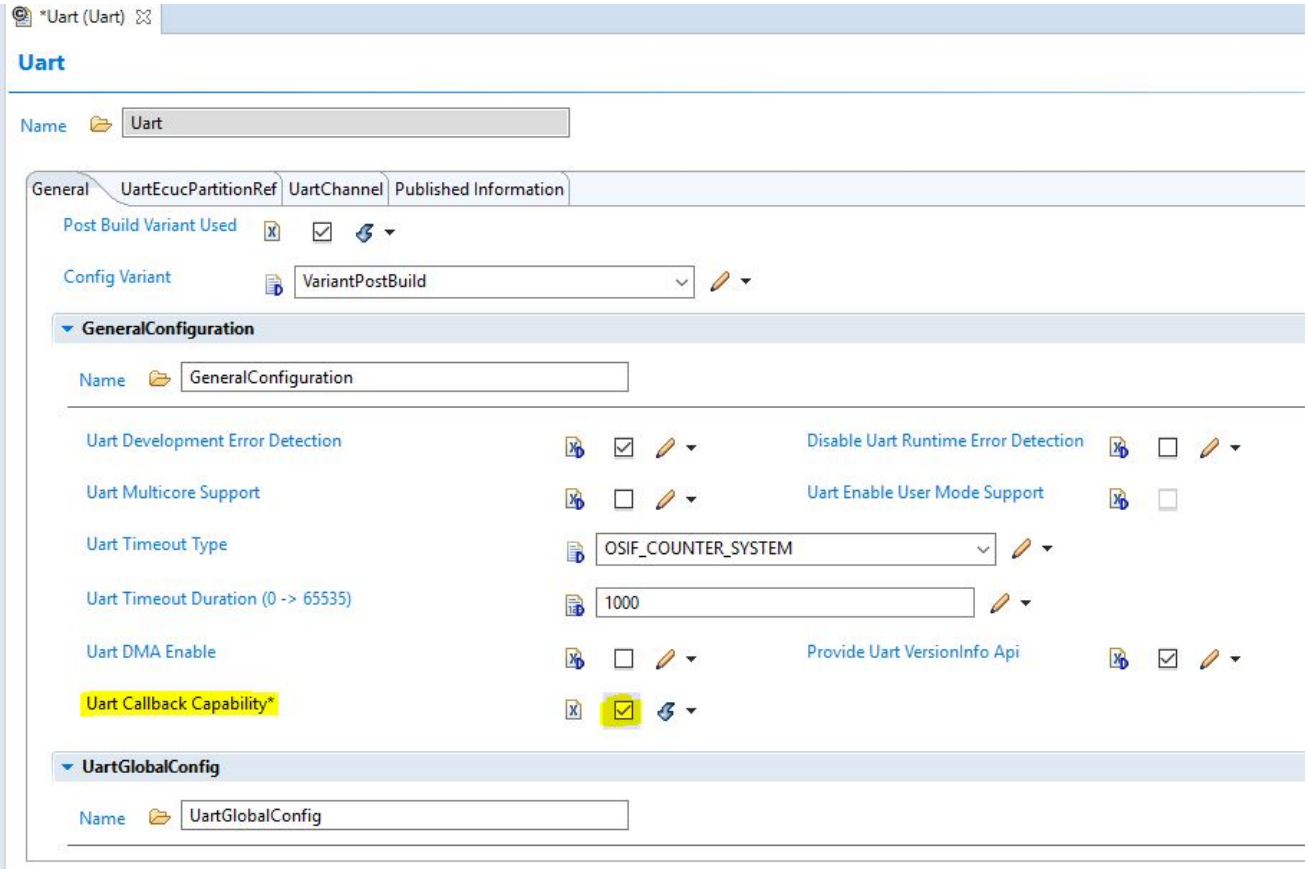

Figure 3.5 Enable Uart Callback Capability

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

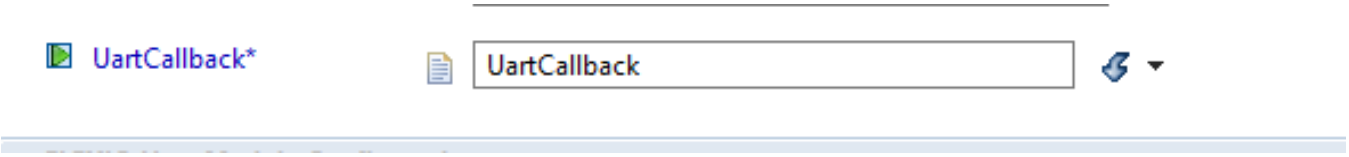

Figure 3.6 User Notifications

- 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 continous transfer.

3.6.2 How to configure multicore

Note: Multicore feature is only supported in S32K32X drivatives.

To enable Multicore Support, GeneralConfiguration/UartMulticoreSupport must be set to "True".

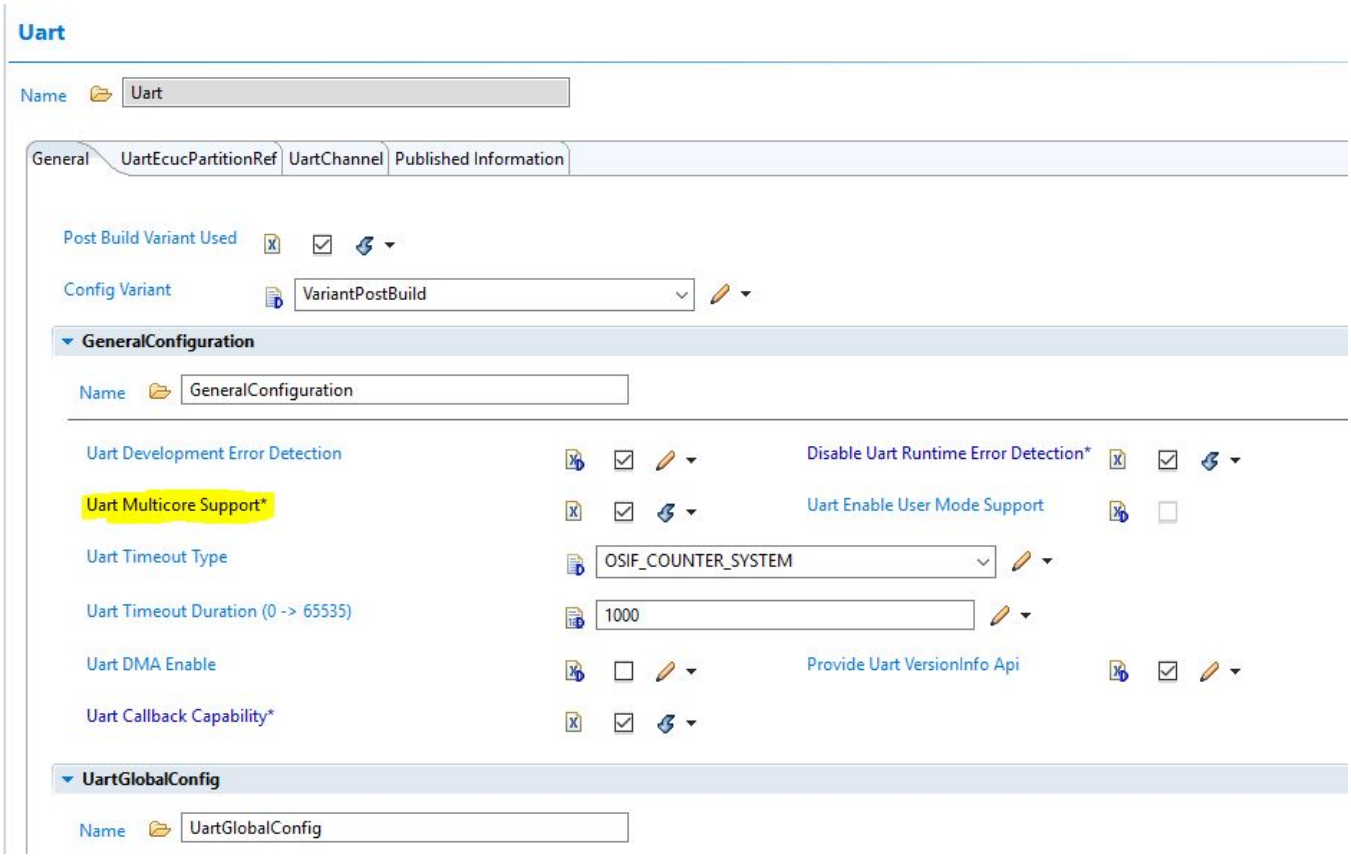

Figure 3.7 Enable Multicore Support

The core for each Uart Channel will be select by UartChannelEcucPartitionRef.

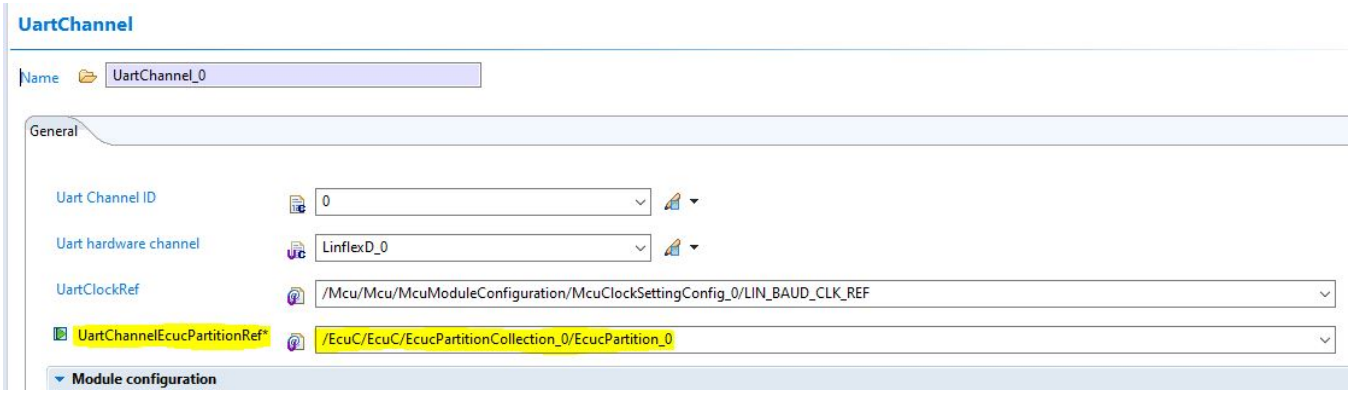

Figure 3.8 Select Uart Channel Ecuc Partition Reference

Each Uart Channel select only one partition (reference from ECUC). Note: All Uart Channels which select UartChannelEcucPartitionRef need to assigned to map with partition in the table below:

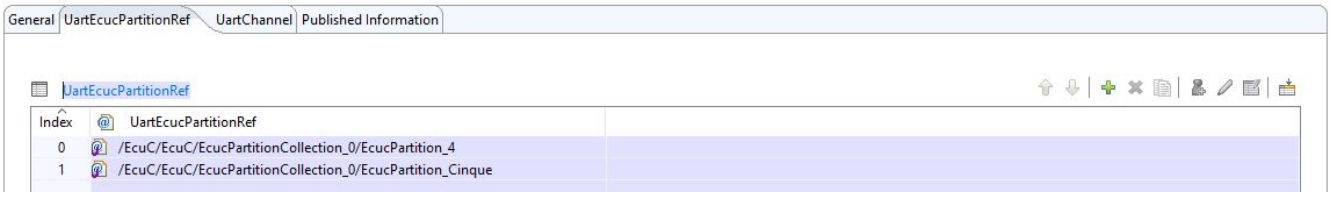

Figure 3.9 Partition for each ExternalDevice

Each partition will refer to a core by OS. A OsApplication will setup for 1 partition mapping with 1 core.

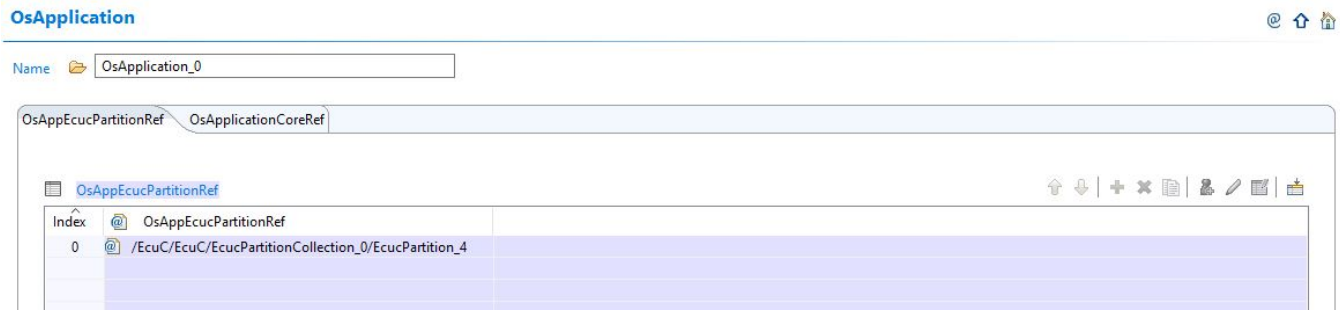

Figure 3.10 Partition is selected by OsApplication_0

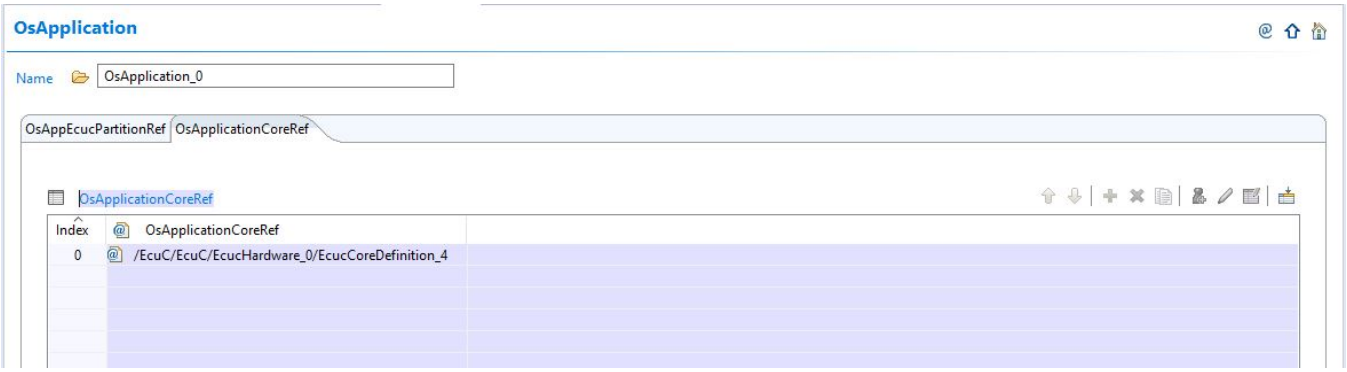

Figure 3.11 Core is also selected by OsApplication_0

In ECUC, for each EcucCoreDefinition will select a Core.

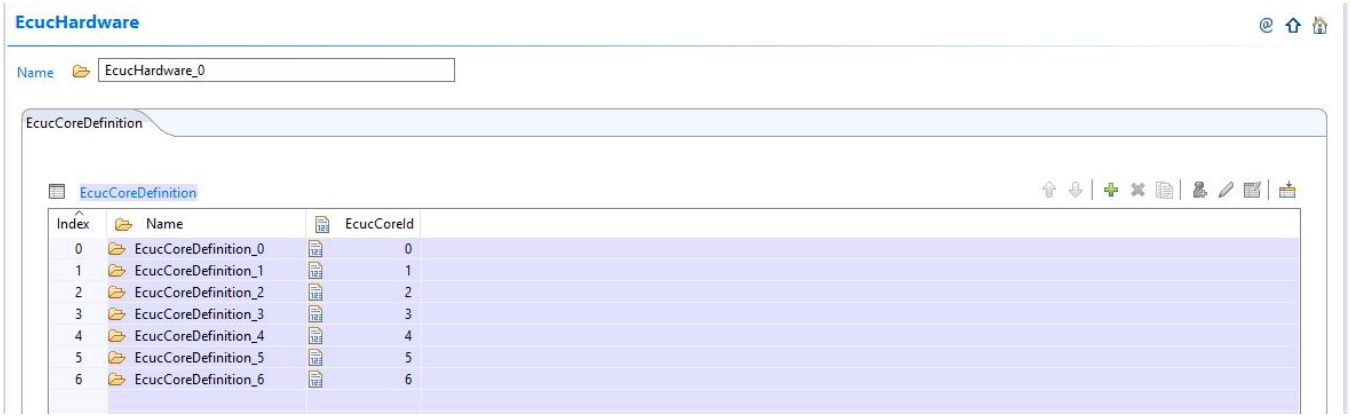

Figure 3.12 Each EcucCoreDefinition will select a Core

3.6.3 How to configure Uart Callback

To enable Uart Callback, GeneralConfiguration/UartCallbackCapability must be set to "True".

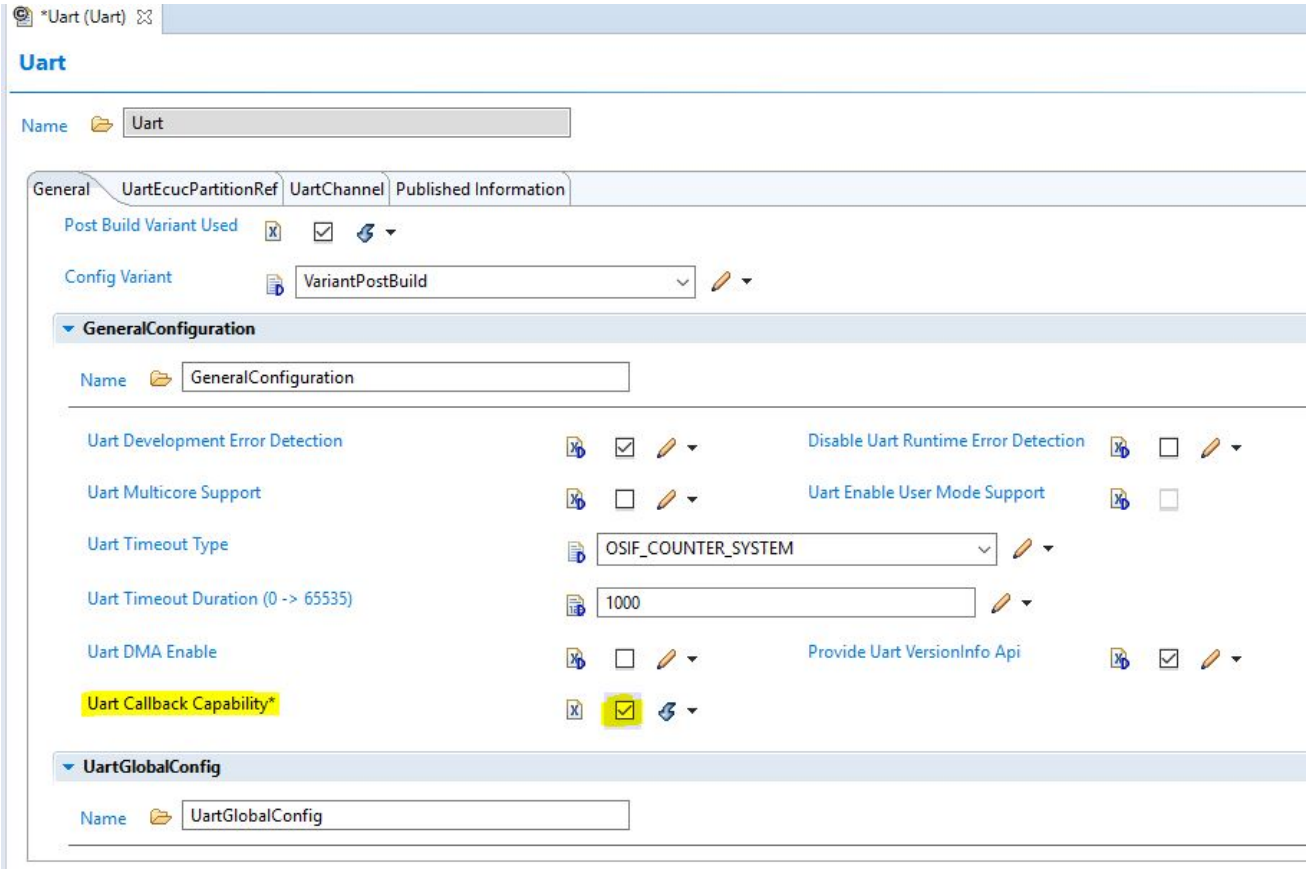

Figure 3.13 Enable Uart Callback Capability

3.6.4 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"

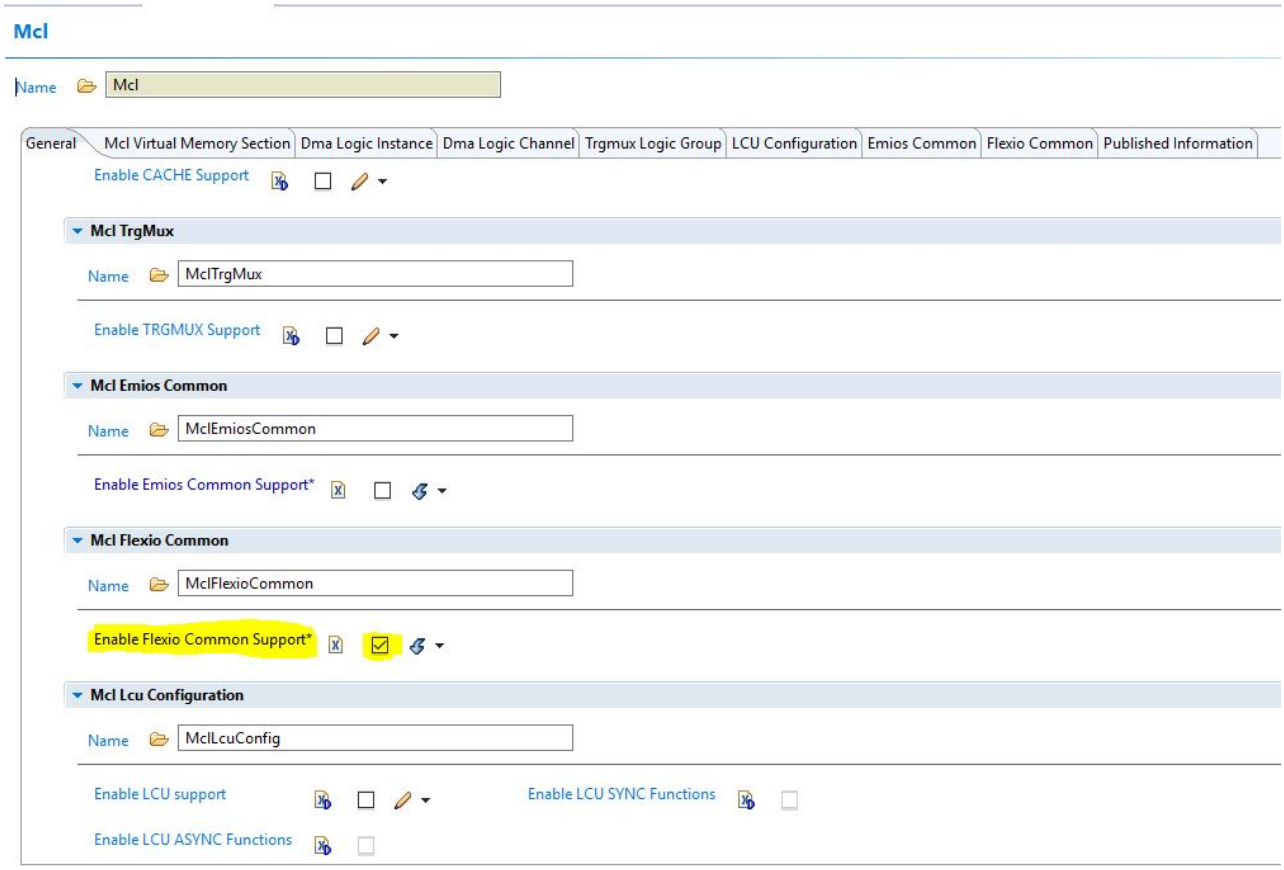

Figure 3.14 Enable Flexio Common Support

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.

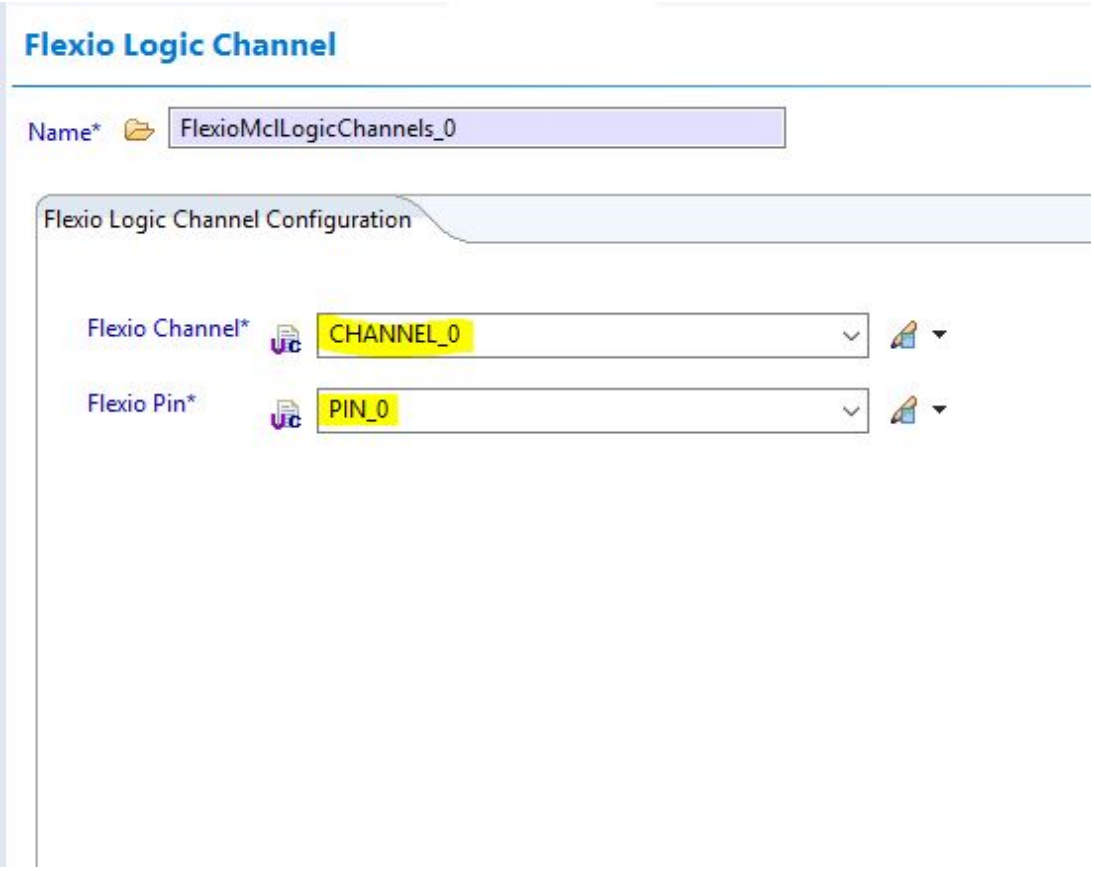

Figure 3.15 Use Flexio Channel 0 (TIMER0, SHIFTER0) and Pin_0(FXIO_D0)

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

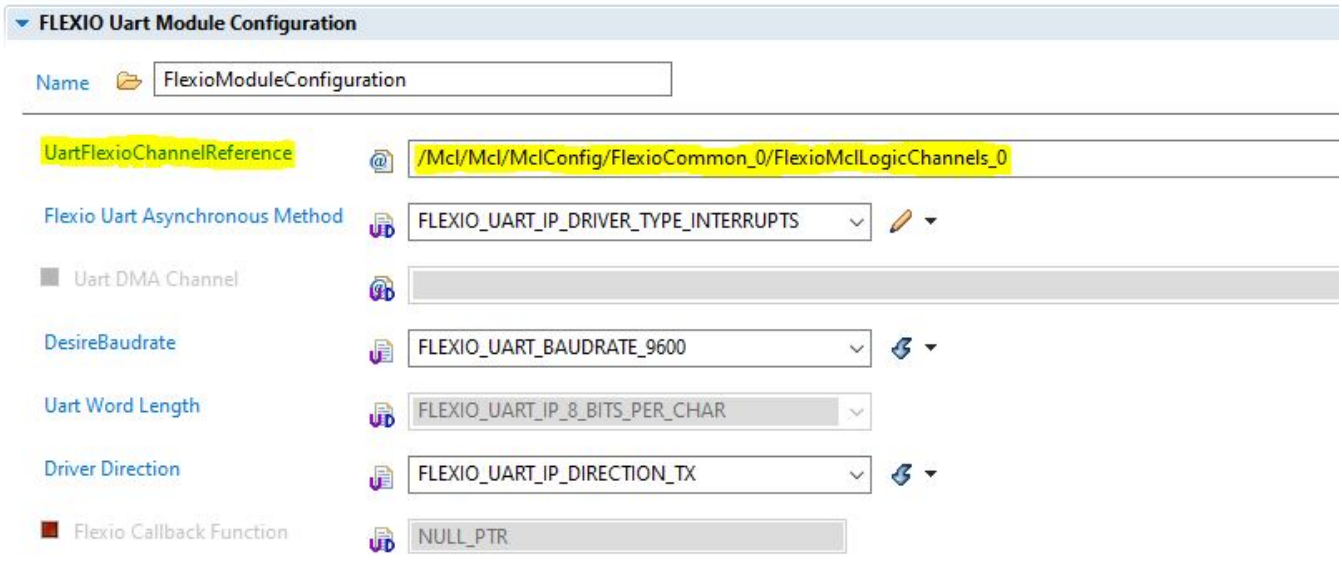

Figure 3.16 Select Flexio Channel configured in MCL module

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]). When performing the abort function in case of transmission using FlexIO, the user needs to use the while function after it to ensure the transmission frame ends. User need to setup a Callback function in Platform module when using DMA Transmission with FlexIO. The multi-core feature is only available for S32K324 device When using multi-core for Flexio, all Flexio channels must be assigned to the same core. 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.

3.6.5 How to choose the appropriate baudrate

Check the generated user struct configuration for the actual calculated baudrate.

- The baudrate deviation should be less than 3% with the normal baudrate value for properly operation.
- For higher baudrate (921600 or 1843200), The baudrate deviation should be less than 1%. Some recommend clock value for baudrate of 1843200:
- Lpuart: LPUART_0 and LPUART_8 can work with AIPS_PLAT_CLK clock value of 59MHz; other instances can work with AIPS_SLOW_CLK clock value of 37MHz or 39MHz.
- Flexio: Can work with clock value with 59MHz or 118MHz.

3.6.6 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∗(number of bit per frame)/baudrate)

3.6.7 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.

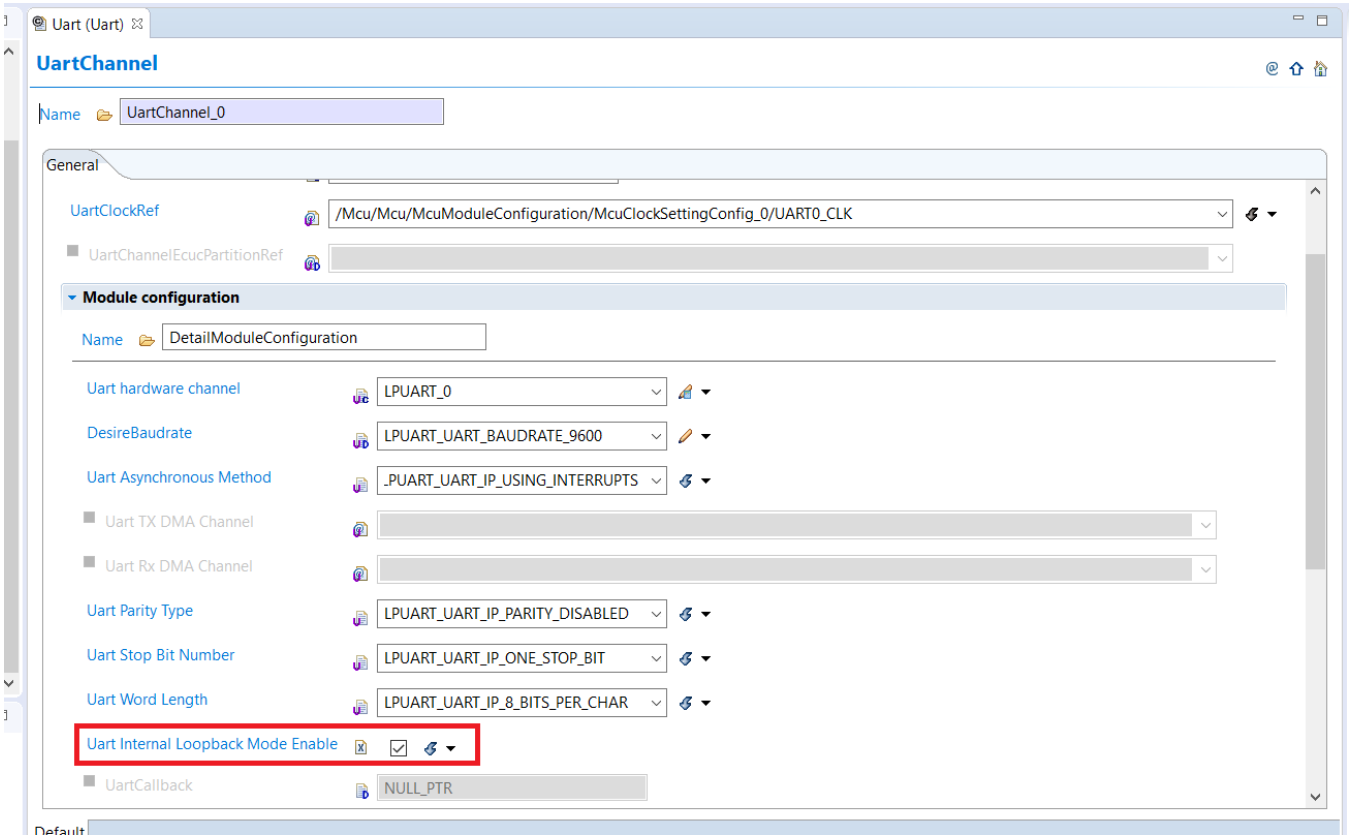

Figure 3.17 Internal Loopback mode configuration

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.

3.6.8 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 (UartInteruptDmaMethod = DMA).

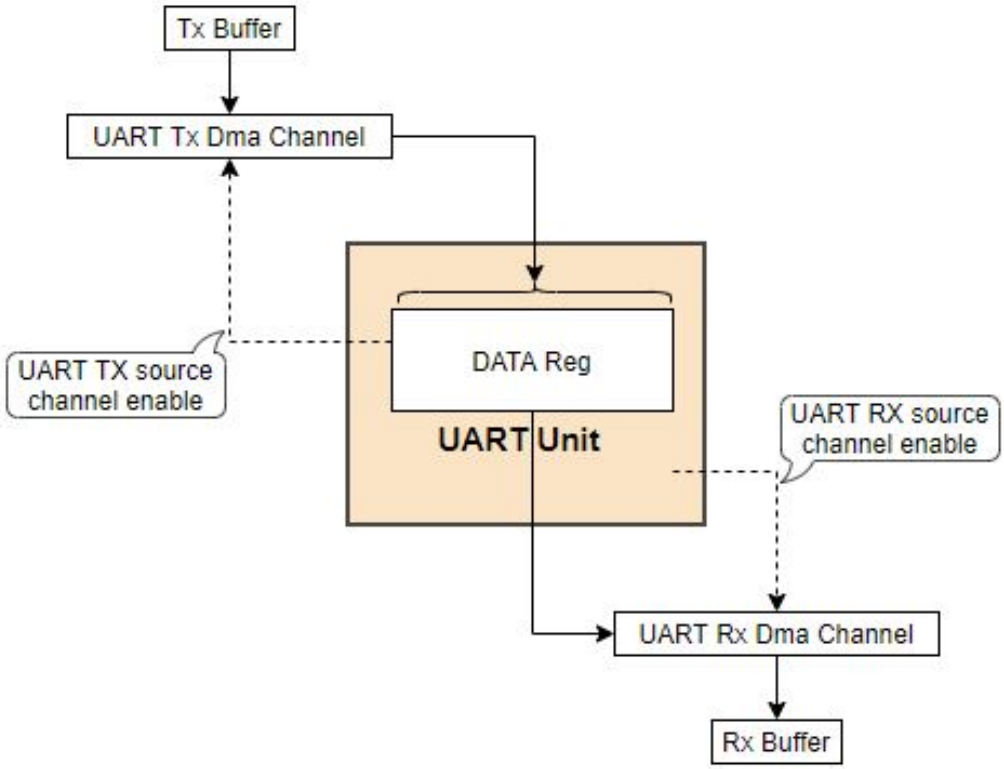

Figure 3.18 DMA transferring mode internal architecture

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.

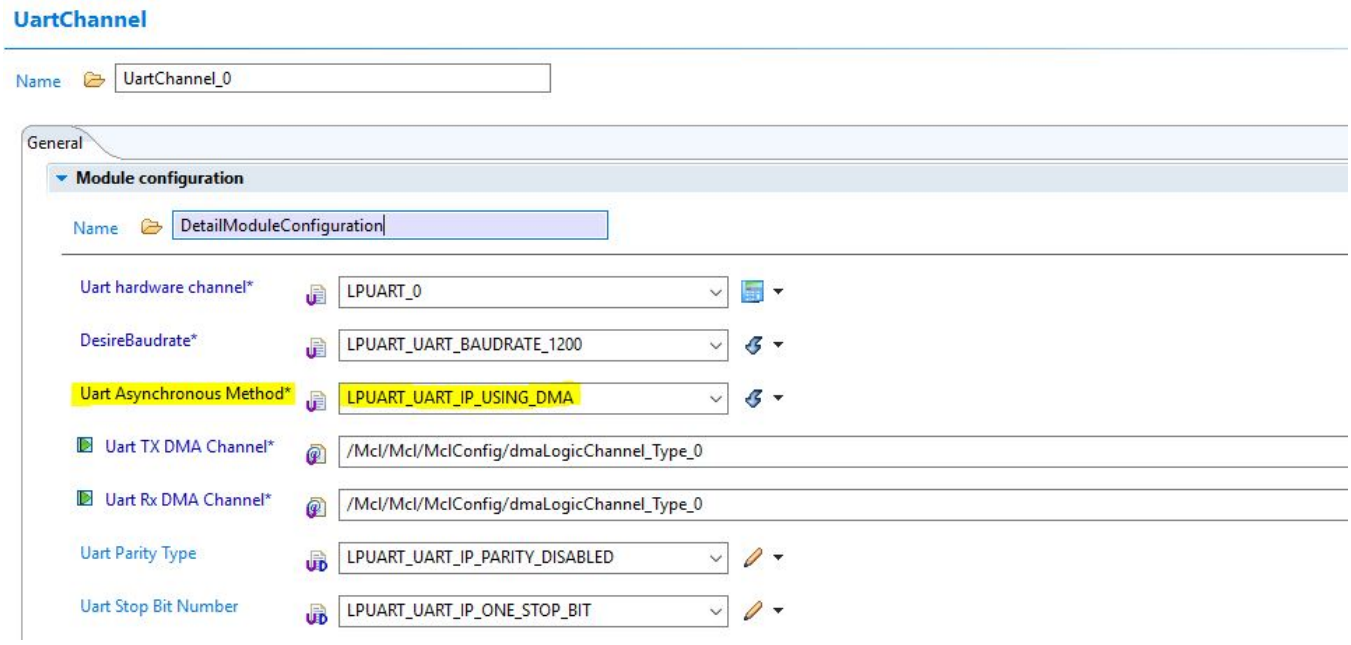

Figure 3.19 DMA Configuration sample for Uart Physical Unit - Uart module in EB Tresos

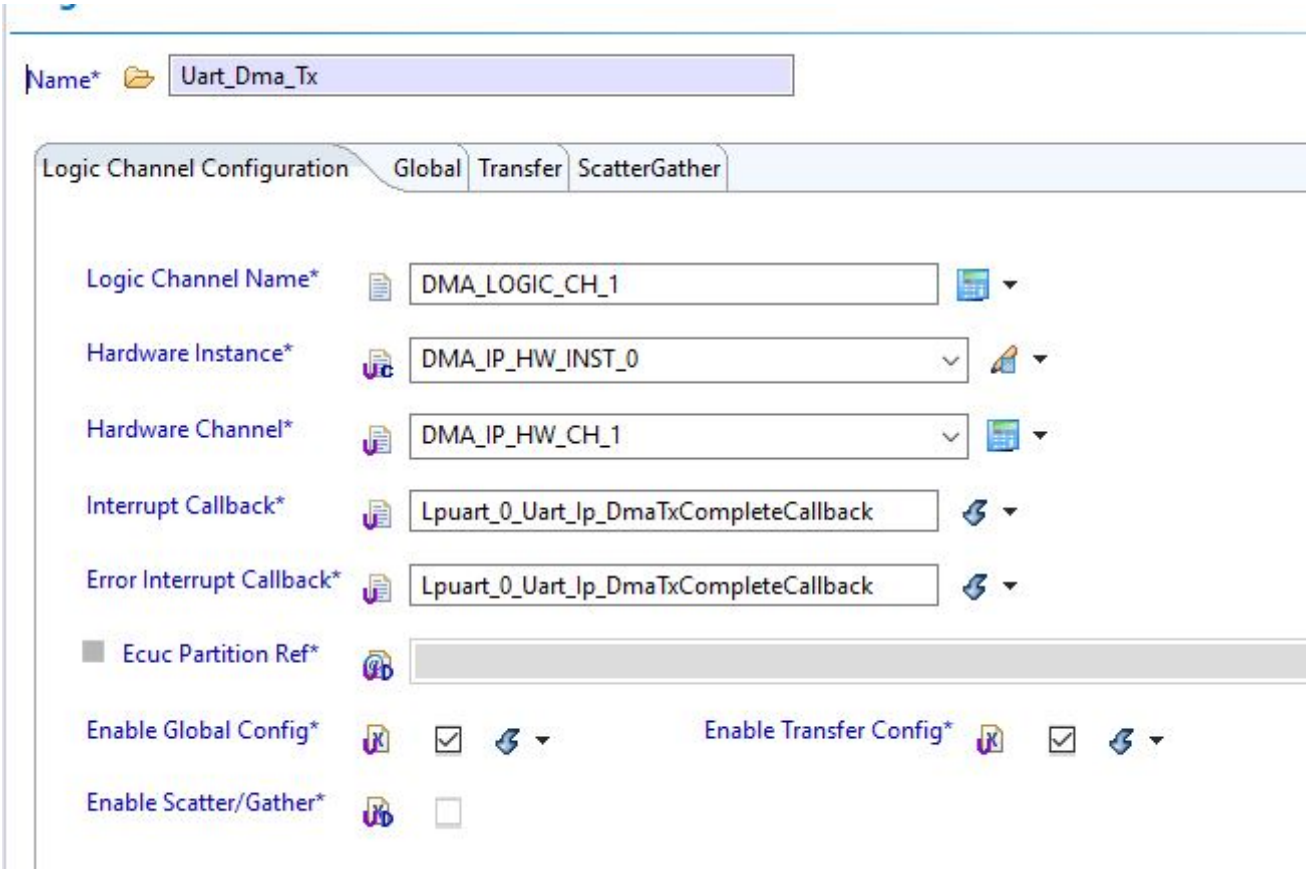

Figure 3.20 DMA TX General configuration - MCL module in EB Tresos

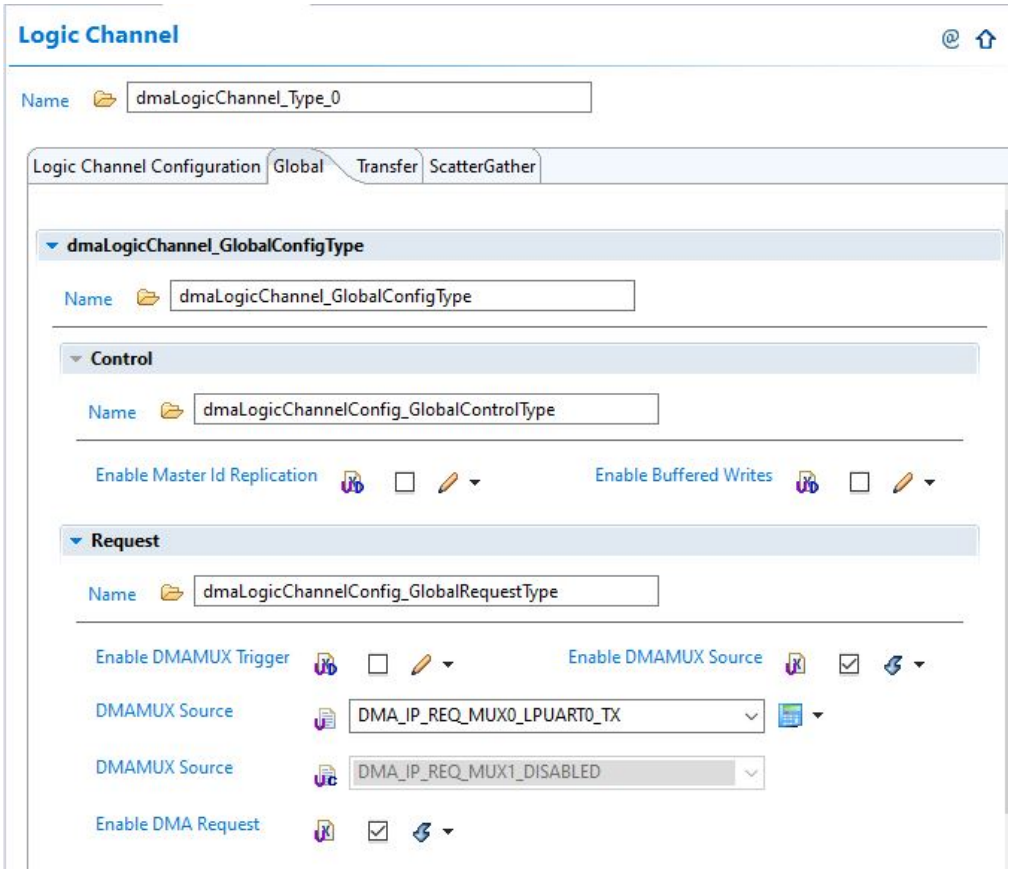

Figure 3.21 DMA TX Global Configuration - MCL module in EB Tresos

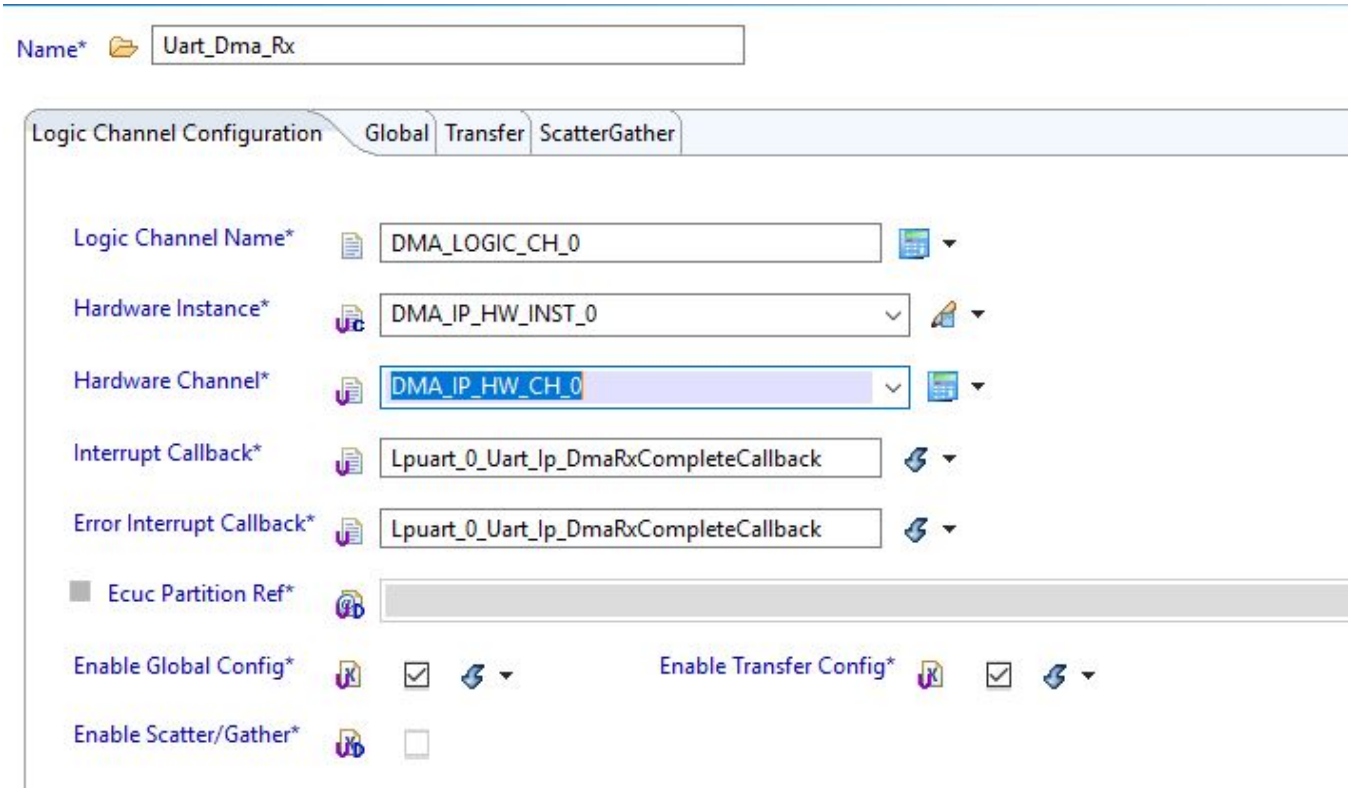

Figure 3.22 DMA Rx General configuration - MCL module in EB Tresos

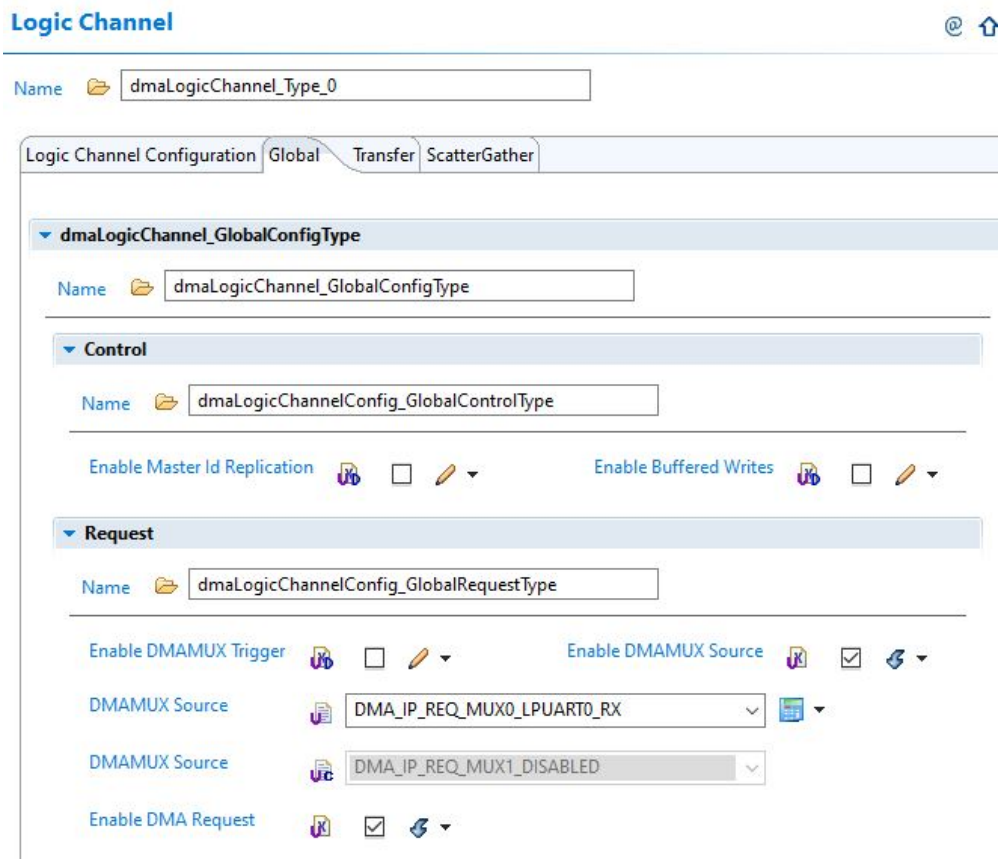

Figure 3.23 DMA Rx Global Configuration - MCL module in EB Tresos

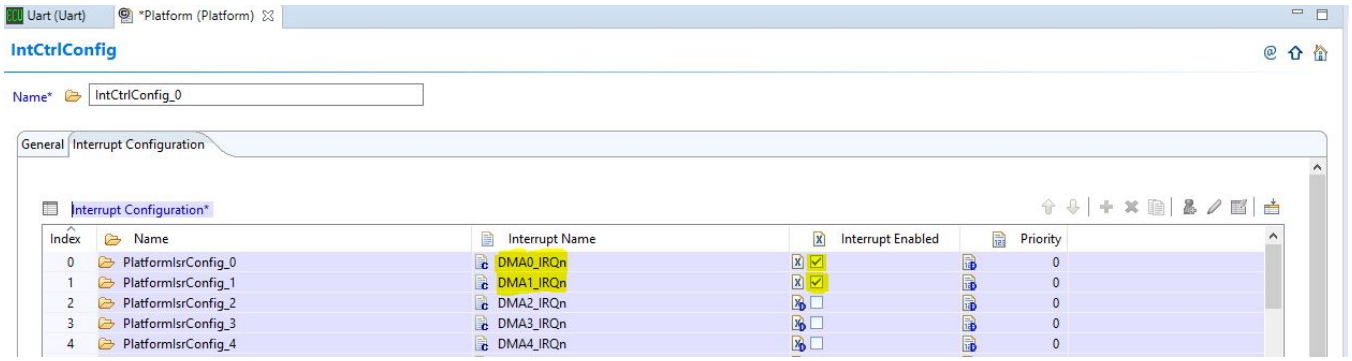

Figure 3.24 Enable DMA IRQ Cofiguration - Platform module in EB Tresos

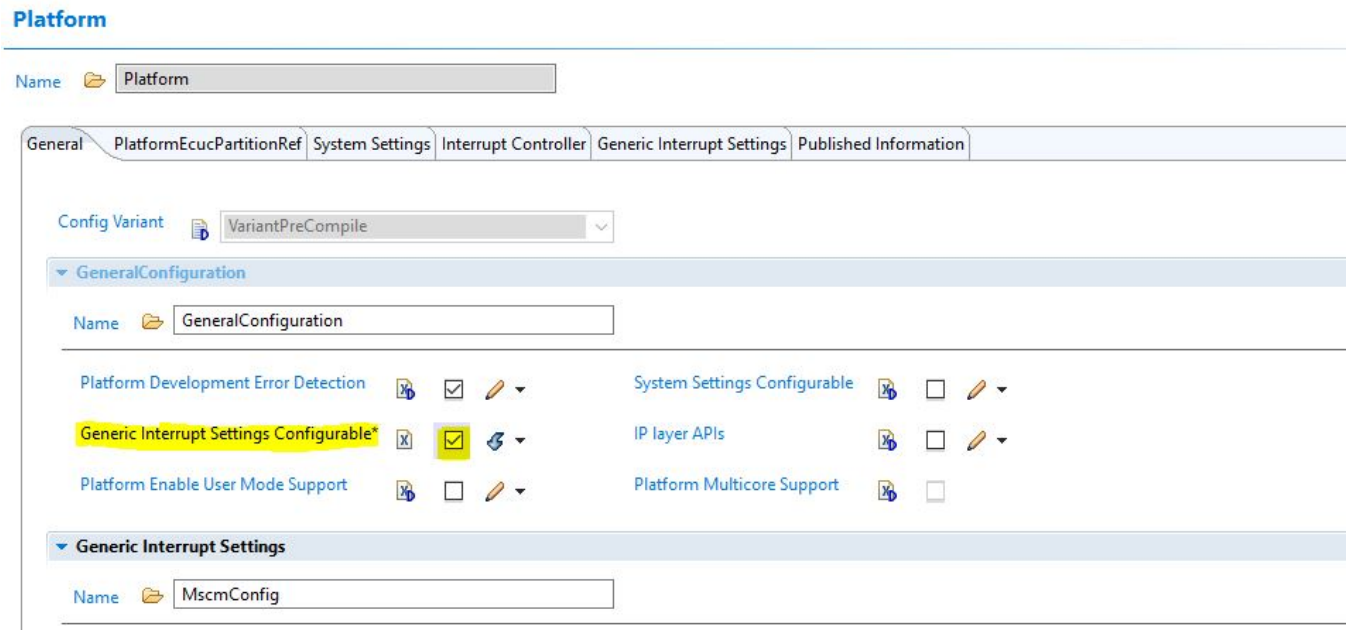

Figure 3.25 Enable Generic Interrupt Seting Configuration - Platform module in EB Tresos

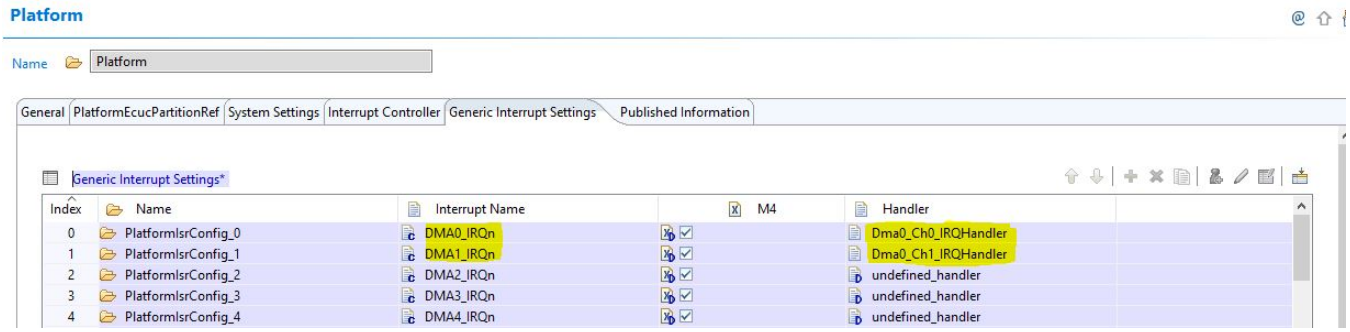

Figure 3.26 Fill In DMA IRQ Function Name - Platform module in EB Tresos

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.

3.7 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 transmiter 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 UartInteruptDmaMethod
      - Parameter UartParityType
      - Parameter UartStopBitNumber
      - Parameter UartWordLength
      - Parameter UartInternalLoopbackEnable

- Reference UartDmaTxChannelRef
- Reference UartDmaRxChannelRef
- Container FlexioModuleConfiguration
- Parameter FlexioUartInteruptDmaMethod
- 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
defaultValue	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
defaultValue	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_COU↩ NTER_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
defaultValue	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
defaultValue	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/EcucDefs/EcuC/EcucPartitionCollection/EcucPartition

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 exit 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	24
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	24
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/EcucDefs/EcuC/EcucPartitionCollection/EcucPartition

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', 'LPUART_2', 'LPUART_3', 'LPUART_4', 'L↩ PUART_5', 'LPUART_6', 'LPUART_7', 'LPUART_8', 'LPUART_9', 'LPU↩ ART_10', 'LPUART_11', 'LPUART_12', 'LPUART_13', 'LPUART_14', 'L↩ PUART_15']

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_BAUDR↩ ATE_1200', 'LPUART_UART_BAUDRATE_2400', 'LPUART_UART_B↩ AUDRATE_4800', 'LPUART_UART_BAUDRATE_7200', 'LPUART_U↩ ART_BAUDRATE_9600', 'LPUART_UART_BAUDRATE_14400', 'LPU↩ ART_UART_BAUDRATE_19200', 'LPUART_UART_BAUDRATE_28800', 'LPUART_UART_BAUDRATE_38400', 'LPUART_UART_BAUDRATE_↩ 57600', 'LPUART_UART_BAUDRATE_115200', 'LPUART_UART_BAU↩ DRATE_230400', 'LPUART_UART_BAUDRATE_460800', 'LPUART_UA↩ RT_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
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
defaultValue	0
max	9223372036854775807
min	-9223372036854775808

4.26 Parameter UartInteruptDmaMethod

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
defaultValue	LPUART_UART_IP_USING_INTERRUPTS
literals	['LPUART_UART_IP_USING_INTERRUPTS', 'LPUART_UART_IP_U↩ SING_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_PA↩ RITY_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', 'LPUART_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 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

4.32 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
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 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

4.34 Parameter FlexioUartInteruptDmaMethod

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
defaultValue	FLEXIO_UART_IP_DRIVER_TYPE_INTERRUPTS
literals	['FLEXIO_UART_IP_DRIVER_TYPE_INTERRUPTS', 'FLEXIO_UART↩ _IP_DRIVER_TYPE_DMA']

4.35 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

Property	Value
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_BAUDRAT↩E_1200', 'FLEXIO_UART_BAUDRATE_2400', 'FLEXIO_UART_BAUDR↩ATE_4800', 'FLEXIO_UART_BAUDRATE_7200', 'FLEXIO_UART_BAU↩DRATE_9600', 'FLEXIO_UART_BAUDRATE_14400', 'FLEXIO_UART_↩BAUDRATE_19200', 'FLEXIO_UART_BAUDRATE_28800', 'FLEXIO_U↩ART_BAUDRATE_38400', 'FLEXIO_UART_BAUDRATE_57600', 'FLEX↩IO_UART_BAUDRATE_115200', 'FLEXIO_UART_BAUDRATE_230400', 'FLEXIO_UART_BAUDRATE_460800', 'FLEXIO_UART_BAUDRATE_↩921600', 'FLEXIO_UART_BAUDRATE_1843200']

4.36 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_DEC↩REMENT_FXIO_CLK_DIV_256']

4.37 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

4.38 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

4.39 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']

4.40 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_DIRECTI↩ ON_RX']

4.41 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/EcucDefs/Mcl/MclConfig/FlexioCommon/FlexioMclLogicChannels']

4.42 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

4.43 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

4.44 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
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

4.45 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

Property	Value
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	7
max	7
min	7

4.46 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
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.47 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

Property	Value
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	255
max	255
min	255

4.48 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
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-POST-BUILD: PUBLISHED-INFORMATION
	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	3
max	3
min	3

4.49 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

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

4.50 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
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 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	

4.52 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:

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 *BytesTransfered, 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 Uart_ChannelConfigType *	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

UART_STATUS_NO_ERROR	Uart operation is successfull
UART_STATUS_OPERATION_ONGOING	Uart operation on going
UART_STATUS_ABORTED	Uart operation aborted
UART_STATUS_FRAMING_ERROR	Uart framing error
UART_STATUS_RX_OVERRUN_ERROR	Uart overrun error
UART_STATUS_PARITY_ERROR	Uart parity error
UART_STATUS_TIMEOUT	Uart operation has timeout
UART_STATUS_NOISE_ERROR	Uart noise error
UART_STATUS_DMA_ERROR	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	*Config*	- 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	*Channel*	- Uart channel to be addressed.
in	*Baudrate*	- Baudrate value to be set.

Returns

Std_ReturnType.

Return values

E_NOT_OK	If the Uart Channel is not valid or Uart driver is not initialized or a transfer is on-going or wrong core is addressed.
E_OK	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	*Channel*	- Uart channel to be addressed.
out	*Baudrate*	- Pointer to a memory location where the baudrate value is returned.

Returns

Std_ReturnType.

Return values

E_NOT_OK	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
E_OK	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	*Channel*	- Uart channel to be addressed.
in	*Buffer*	- The new buffer provided.
in	*BufferSize*	- The size of the new buffer.
in	*Direction*	- 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	*Channel*	- Uart channel to be addressed.
in	*Buffer*	- The buffer which contains the bytes to be sent.
in	*BufferSize*	- The Buffer size.
in	*Timeout*	- The timeout in us.

Returns

Std_ReturnType.

Return values

E_NOT_OK	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 occured.
E_OK	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	*Channel*	- Uart channel to be addressed.
in	*Buffer*	- The buffer where the bytes will be located.
in	*BufferSize*	- The Buffer size.
in	*Timeout*	- The timeout in us.

Returns

Std_ReturnType.

Return values

E_NOT_OK	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 occured.
E_OK	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	*Channel*	- Uart channel to be addressed.
in	*TransmissionType*	- Type of the transfer to be aborted. It can be either UART_SEND or UART_RECEIVE.

Returns

Std_ReturnType.

Return values

E_NOT_OK	If the Uart Channel is not valid or Uart driver is not initialized or a wrong core has been accessed
E_OK	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	*Channel*	- Uart channel to be addressed.
in	*Buffer*	- The buffer where the data to be sent is located.
in	*BufferSize*	- The Buffer size.

Returns

Std_ReturnType.

Return values

E_NOT_OK	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.
E_OK	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	*Channel*	- Uart channel to be addressed.
in	*Buffer*	- The buffer where the data to be received will located.
in	*BufferSize*	- The Buffer size.

Returns

Std_ReturnType.

Return values

E_NOT_OK	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.
E_OK	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 * BytesTransfered,
            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	*Channel*	- Uart channel to be addressed.
out	*BytesTransfered*	- A pointer where the number of remaining bytes will be written.
in	*TransferType*	- The type of trasfer in discussion (UART_SEND or UART_RECEIVE).

Returns

Uart_StatusType.

Return values

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

Return values

UART_STATUS_TIMEOUT	- 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.
Flexio_Uart_IP_DriverType	DriverType	Driver type: Interrupts/DMA.
uint8	Divider	Baudrate divider.
Flexio_Uart_Ip_TimerDecrementType	TimerDec	The source of the Timer decrement and the source of the Shift clock.
uint32	BaudRate	Baud rate in hertz.
Flexio_Uart_Ip_BitCountPerCharType	BitCount	Number of bits per word.
Flexio_Uart_Ip_DirectionType	Direction	Driver direction: Tx or Rx.
uint8	DataPin	Flexio pin to use as Tx or Rx pin.
Flexio_Uart_Ip_CallbackType	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.
Flexio_Uart_Ip_StateStructureType *	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 occured 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

FLEXIO_UART_IP_STATUS_SUCCESS	Operation has been successfully
FLEXIO_UART_IP_STATUS_ERROR	Operation has had error
FLEXIO_UART_IP_STATUS_BUSY	Function is called during an on-going transfer
FLEXIO_UART_IP_STATUS_DMA_ERROR	Operation has had DMA error
FLEXIO_UART_IP_STATUS_TX_UNDERRUN	TX underrun error
FLEXIO_UART_IP_STATUS_RX_OVERRUN	RX overrun error
FLEXIO_UART_IP_STATUS_ABORTED	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

FLEXIO_TIMER_DECREMENT_FXIO_CLK_↩ SHIFT_TMR	Decrement counter on FlexIO clock, Shift clock equals Timer output.
FLEXIO_TIMER_DECREMENT_FXIO_CLK_↩ DIV_16	Decrement counter on FlexIO clock divided by 16, Shift clock equals Timer output.
FLEXIO_TIMER_DECREMENT_FXIO_CLK_↩ DIV_256	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 *Bytes↩ Remaining)

  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 *Bytes↩ Remaining)

  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_BaudrateTyp 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 189 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 Lpuart_Uart_Ip_StatusType	TransmitStatus	Status of last driver transmit operation.
volatile Lpuart_Uart_Ip_StatusType	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 207 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.
Lpuart_Uart_Ip_ParityModeType	ParityMode	Parity mode, disabled (default), even, odd.
Lpuart_Uart_Ip_StopBitCountType	StopBitsCount	Number of stop bits, 1 stop bit (default) or 2 stop bits.
Lpuart_Uart_Ip_BitCountPerCharType	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.
Lpuart_Uart_Ip_TransferType	TransferType	@briefType of LPUART transfer (interrupt/dma based)
Lpuart_Uart_Ip_CallbackType	Callback	Callback to invoke for handle uart event.
void *	CallbackParam	User callback parameter pointer.
Lpuart_Uart_Ip_StateStructureType *	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 176 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	• A transfer was aborted
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 occured during transfer.

Definition at line 135 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 148 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

Instance	LPUART instance number
UserConfig	user configuration structure of type Lpuart_Uart_Ip_UserConfigType

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

Instance	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

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

Instance	LPUART instance number.
TxBuff	The buffer pointer which saves the data to be sent.
TxSize	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

Instance	LPUART instance number
BytesRemaining	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

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

instance	LPUART instance number

Returns

LPUART_UART_IP_STATUS_ERROR if the transmit process has not fully completed, LPUART_UA↩ RT_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

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

Returns

LPUART_UART_IP_STATUS_SUCCESS if the transaction is successful; LPUART_UART_IP_STA↩ TUS_BUSY if the resource is busy; LPUART_UART_IP_STATUS_RX_OVERRUN if an overrun error occured LPUART_UART_IP_STATUS_FRAMING_ERROR if a framing error occured LPUART_UA↩ RT_IP_STATUS_PARITY_ERROR if a parity error occured LPUART_UART_IP_STATUS_NOISE↩ _ERROR if a noise error occured 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

Instance	LPUART instance number
RxBuff	buffer containing 8-bit read data chars received
RxSize	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 )
```

Returns whether the previous receive is complete.

Parameters

Instance	LPUART instance number
BytesRemaining	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

LPUART_UART_IP_STATUS_SUCCESS	the receive has completed successfully.
LPUART_UART_IP_STATUS_BUSY	the receive is still in progress. *bytesReceived* will be filled with the number of bytes that have been received so far.
LPUART_UART_IP_STATUS_ABORTED	The receive was aborted.
LPUART_UART_IP_STATUS_TIMEOUT	A timeout was reached.
LPUART_UART_IP_STATUS_RX_OVERR↩ UN,LPUART_UART_IP_STATUS_FRAMING_↩ ERROR,LPUART_UART_IP_STATUS_PARIT↩ Y_ERROR,or	LPUART_UART_IP_STATUS_NOISE_ERROR, LPUART_UART_IP_STATUS_ERROR 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

Instance	LPUART instance number

Returns

LPUART_UART_IP_STATUS_ERROR if the receive process has not fully completed, LPUART_UAR↩ T_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

Instance	LPUART instance number.
DesiredBaudrate	LPUART desired baud rate.
ClockFrequency	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

	Instance	LPUART instance number.
out	*ConfiguredBaudRate*	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

Instance	LPUART instance number
TxBuff	source buffer containing 8-bit data chars to send
TxSize	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

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

Returns

void

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

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

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