MCAL User Manual for Uart

32-bit TriCore™ AURIX™ TC3xx microcontroller

About this document

Scope and purpose

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

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

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

Intended audience

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

Document conventions

Table 1 Conventions

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

Reference documents

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

- AURIX™ TC3xx MCAL User Manual General

Table of contents

Table of contents

Table of contents

1 Uart driver

1.1 User information

1.1.1 Description

The UART driver is responsible for providing communication services as per the UART protocol. The ASCLIN module provides hardware support for asynchronous communication to realize the UART protocol. The UART driver provides functionality for configuration, initialization, data transmission, reception and also provides optional features such as abort transmission and abort reception.

1.1.2 Hardware-software mapping

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

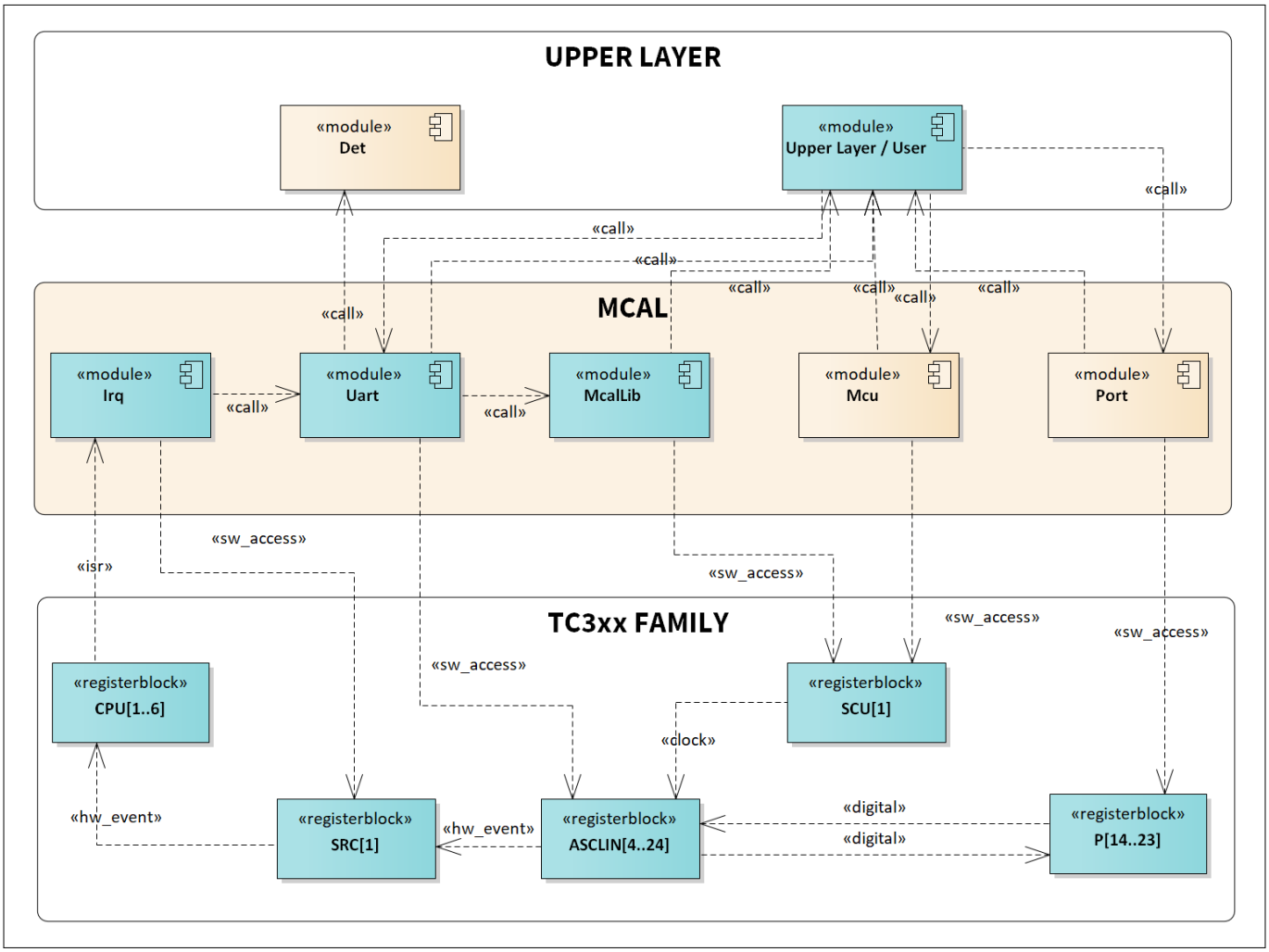


Figure 1 Mapping of hardware-software interfaces

1.1.2.1 SRC: dependent hardware peripheral

Hardware functional features

The UART driver depends on the interrupt router for raising an interrupt to the CPU based on the transmit and receive events, which indicates successful data transmission and reception respectively.

Users of the hardware

The interrupt router is configured either by the IRQ driver or the user software. No functional block of the interrupt router is administered by the UART driver.

Hardware diagnostic features

The SMU alarms configured for interrupt router are not monitored by the UART driver.

Hardware events

The interrupt events raised by the interrupt router are serviced by the CPU. The UART driver provides interrupt handlers as software interfaces, which must be invoked from the ISR.

1.1.2.2 PORT: dependent hardware peripheral

Hardware functional features

The ARX, ATX, CTS and RTS signals are routed to the ASCLIN through the port pads. These signals are configured and enabled through the PORT driver.

Users of the hardware

The port pads are configured by the PORT driver.

Hardware diagnostic features

Not applicable.

Hardware events

Hardware events from port pads are not used by the UART driver.

1.1.2.3 SCU: dependent hardware peripheral

Hardware functional features

The UART driver depends on the SCU IP for the clock, ENDINIT and reset functionalities. The driver requires the fSPB, fASCLINF and fASCLINS clock signals for functioning.

Users of the hardware

The SCU IP supplies clock for all the peripherals and the MCU driver is responsible for configuring the clock tree. To avoid conflicts due to simultaneous writes, update to all the ENDINIT protected registers is performed using the MCALLIB APIs.

Hardware diagnostic features

The SMU alarms configured for the SCU IP are not monitored by the UART driver.

Hardware events

Hardware events from the SCU are not used by the UART driver.

1.1.2.4 ASCLIN: primary hardware peripheral

Hardware functional features

The UART driver uses the ASCLIN for transmission and reception of data. The key hardware functional features used by the driver are:

- Full-duplex asynchronous operating modes
- Supports half duplex operating mode

- 16 bytes TXFIFO
- 16 bytes RXFIFO
- 2 to 16 bits data frames
- Parity-bit generation/checking
- One or two stop bits
- Baud rate configuration
- Optional RTS / CTS handshaking
- Programmable over-sampling 4 to 16 times per bit
- Programmable sample point position
- Interrupt generation
- Interrupt signals capable of triggering a CPU
- Programmable digital glitch filter and median filter for incoming bit stream
- Pack / unpack capabilities of the Tx and Rx FIFOs
- Shift direction LSB first for ASC

The unsupported feature of the ASCLIN is:

- Internal loop-back mode

Users of the hardware

The LIN and UART drivers utilizes the ASCLIN IP. The allocation of ASLIN channels to LIN/UART driver is done by the MCU driver. Both LIN and UART drivers utilize only the channels allocated to them.

Hardware diagnostic features

The SMU alarms configured for the ASCLIN are not monitored by the UART driver.

Hardware events

The UART driver uses the following hardware events from the ASCLIN IP:

- Transmit FIFO level interrupt: This interrupt is triggered when TXFIFO transmission is completed successfully.
- Receive FIFO level interrupt: This interrupt is triggered when RXFIFO filled up to configured level with received data.
- Error condition parity error, frame error, RXFIFO overflow error.

1.1.3 File structure

1.1.3.1 C file structure

This section provides details of the C files of the UART driver.

1 Uart driver

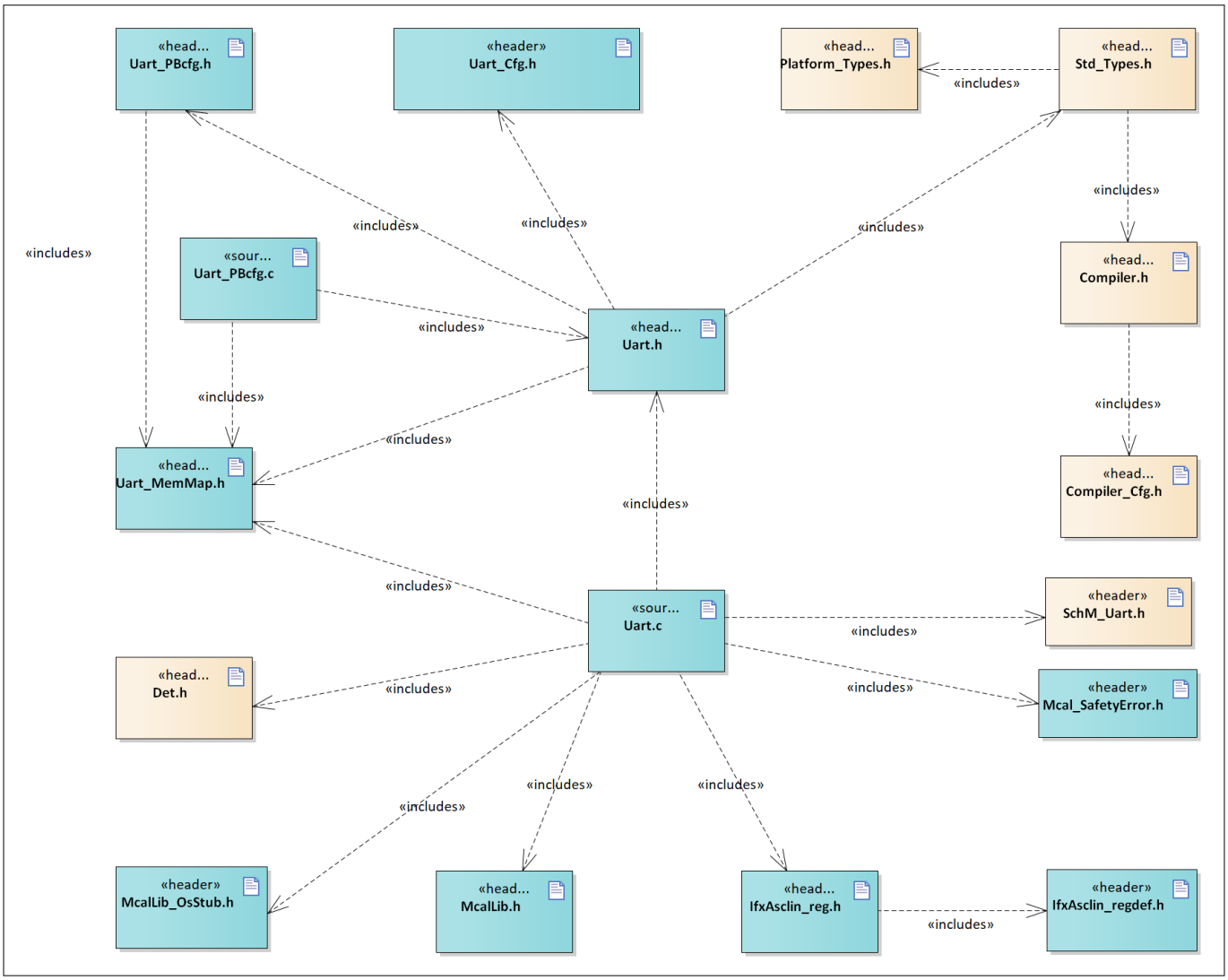


Figure 2 Uart_C_File_Structure-1.png

Table 2 C file structure

File name	Description
`Compiler.h`	Provides abstraction from compiler-specific keywords
`Compiler_Cfg.h`	Configuration header file for compiler abstraction
`Det.h`	Provides the exported interfaces of Development Error Tracer
`IfxAsclin_reg.h`	SFR header file for ASCLIN
`IfxAsclin_regdef.h`	SFR header file for ASCLIN
`McalLib.h`	Static header file defining prototypes of data structure and APIs exported by the MCALLIB.
`McalLib_OsStub.h`	McalLib_OsStub.h provides macros to support user mode of Tricore. This shall be included by other drivers to call OS APIs.
`Mcal_SafetyError.h`	Header file containing the prototype of the API for reporting safety-related errors
`Platform_Types.h`	Platform-specific type declaration file as defined by AUTOSAR

Table 2 C file structure (continued)

File name	Description
SchM_Uart.h	Header file containing prototype of the scheduled function of the Uart driver.
Std_Types.h	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.
Uart.c	File (static) containing implementation of APIs and ISRs
Uart.h	Header file (Static) defining prototypes of data structures, APIs and Interrupt handlers
Uart_Cfg.h	Contains UART driver pre-compile configuration parameters
Uart_MemMap.h	File containing the memory section definitions used by the UART driver
Uart_PBcfg.c	File (generated) containing objects to data structures
Uart_PBcfg.h	Post-build header file for the UART driver

1.1.3.2 Code generator plugin files

This section provides details of the code generator plugin files of the UART driver.

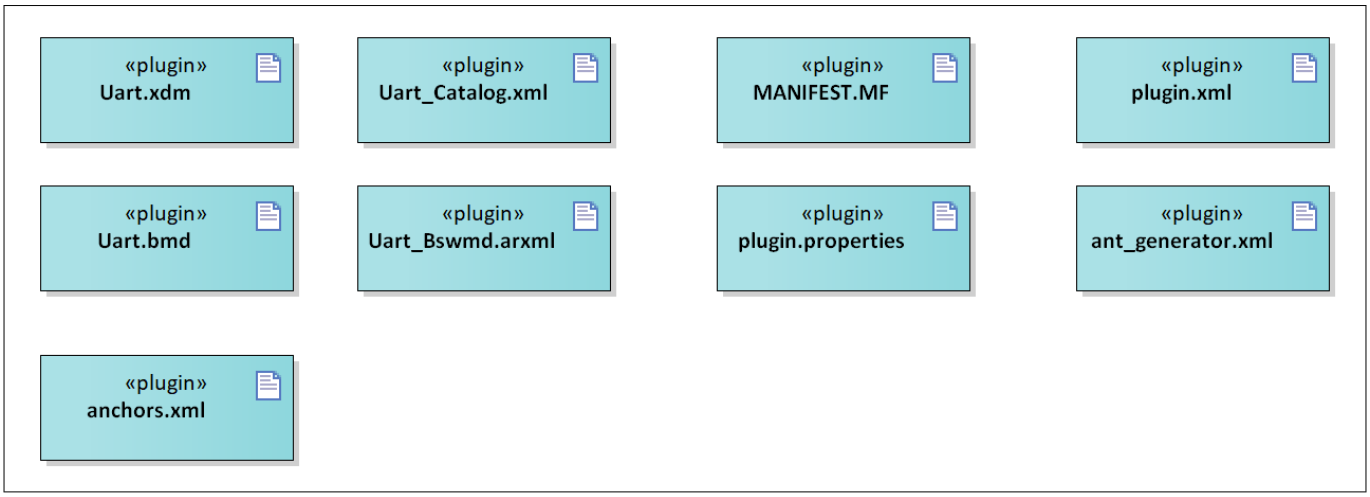


Figure 3 Uart_Code_Generator_Plugin_Files-1.png

Table 3 Code generator plugin files

File name	Description
MANIFEST.MF	Tresos plugin support file containing the metadata for UART driver
Uart.bmd	AUTOSAR format XML data model schema file (for each device)
Uart.xdm	Tresos format XML data model schema file
Uart_Bswmd.arxml	AUTOSAR format module description file
Uart_Catalog.xml	AUTOSAR format catalog file
anchors.xml	Tresos anchors support file for the UART driver
ant_generator.xml	Tresos support file to generate and rename multiple post-build configurations when using variation point

Table 3 Code generator plugin files (continued)

File name	Description
`plugin.properties`	Tresos plugin support file for the UART driver
`plugin.xml`	Tresos plugin support file for the UART driver

1.1.4 Integration hints

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

1.1.4.1 Integration with AUTOSAR stack

This section lists the modules, which are not part of MCAL, but required to integrate the UART driver.

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **Memory mapping**

  Memory mapping is a concept from AUTOSAR that allows relocation of text, variables, constants and configuration data to user-specific memory regions. To achieve this, all the relocatable elements of the driver are encapsulated in different memory-section macros. These macros are defined in the `Uart_MemMap.h` file.

  The `Uart_MemMap.h` file is provided in the MCAL package as a stub code. The integrator must place appropriate compiler pragmas within the memory-section macros. The pragmas ensure that the elements are re-located to the correct memory region. A sample implementation listing the memory-section macros is shown as follows.

```
/*To be used for all global or static variables.*/
#if defined UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 /* User Pragma here */
 #undef UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 /* User Pragma here */
 #undef UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 #undef MEMMAP_ERROR
#elif defined UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 /* User Pragma here */
 #undef UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 #undef UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
#elif defined UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
 /*To be used for global or static constants.*/
#elif defined UART_START_SEC_CONST_ASIL_B_LOCAL_32
 /* User Pragma here */
 #undef UART_START_SEC_CONST_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_CONST_ASIL_B_LOCAL_32
 /* User Pragma here */
 #undef UART_STOP_SEC_CONST_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
 /* UART module configuration data */
#elif defined UART_START_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_START_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_STOP_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
 /* Code section */
#elif defined UART_START_SEC_CODE_ASIL_B_LOCAL
 /* User Pragma here */
 #undef UART_START_SEC_CODE_ASIL_B_LOCAL
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_CODE_ASIL_B_LOCAL
 /* User Pragma here */
 #undef UART_STOP_SEC_CODE_ASIL_B_LOCAL
 #undef MEMMAP_ERROR
#endif
```

```
#if defined MEMMAP_ERROR
#error "Uart_MemMap.h, wrong pragma command"
#endif
```

- **DET**

  The DET module is a part of the AUTOSAR stack that handles all the development and runtime errors reported by the BSW modules. The UART driver reports all the development errors to the DET module through the `Det_ReportError()` API . The user of the UART driver must process all the errors reported to the DET module through the API `Det_ReportError()`.

  The `Det.h` and `Det.c` files are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.
- **DEM**

  The DEM module is not required for integrating the UART driver.
- **SchM**

  The SchM is not required for integrating the UART driver.
- **Safety error**

  The UART driver reports all the detected safety errors through the `Mcal_ReportSafetyError()` API.

  The driver performs only detection and reporting of the safety errors. The handling of the reported errors shall be done by the user. The `Mcal_ReportSafetyError()` API is provided in the `Mcal_SafetyError.c` and `Mcal_SafetyError.h` files as a stub code, and must be updated by the integrator to handle the reported errors.

  Note: All DET errors are also reported as safety errors (error code used is same as DET).
- **Notifications and callbacks**

  The UART driver does not implement any notifications. However, the UART driver reports the completion of a transmit, receive, abort transmit and abort receive operation through notification functions. These notification functions can be configured by the user in EB tresos for each UART channel separately.
- **Operating system(OS)**

  The OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application.

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

1.1.4.2 Multicore and Resource Manager

The UART driver does not support execution on multiple cores simultaneously.

1.1.4.3 MCU support

The UART driver is dependent on the MCU driver for clock configuration and channel allocation services. The initialization of the UART driver must be started only after completing the MCU initialization. The following must be considered while configuring the MCU driver in tresos:

- The $f_{ASCLINF}$ or $f_{ASCLINS}$ defines the clock frequency for the ASCLIN kernel. To configure clock frequency for $f_{ASCLINF}$ or $f_{ASCLINS}$ refer to the `McuAscLinFastFrequency` and `McuAscLinSlowFrequency` parameters from the MCU driver configuration as follows:

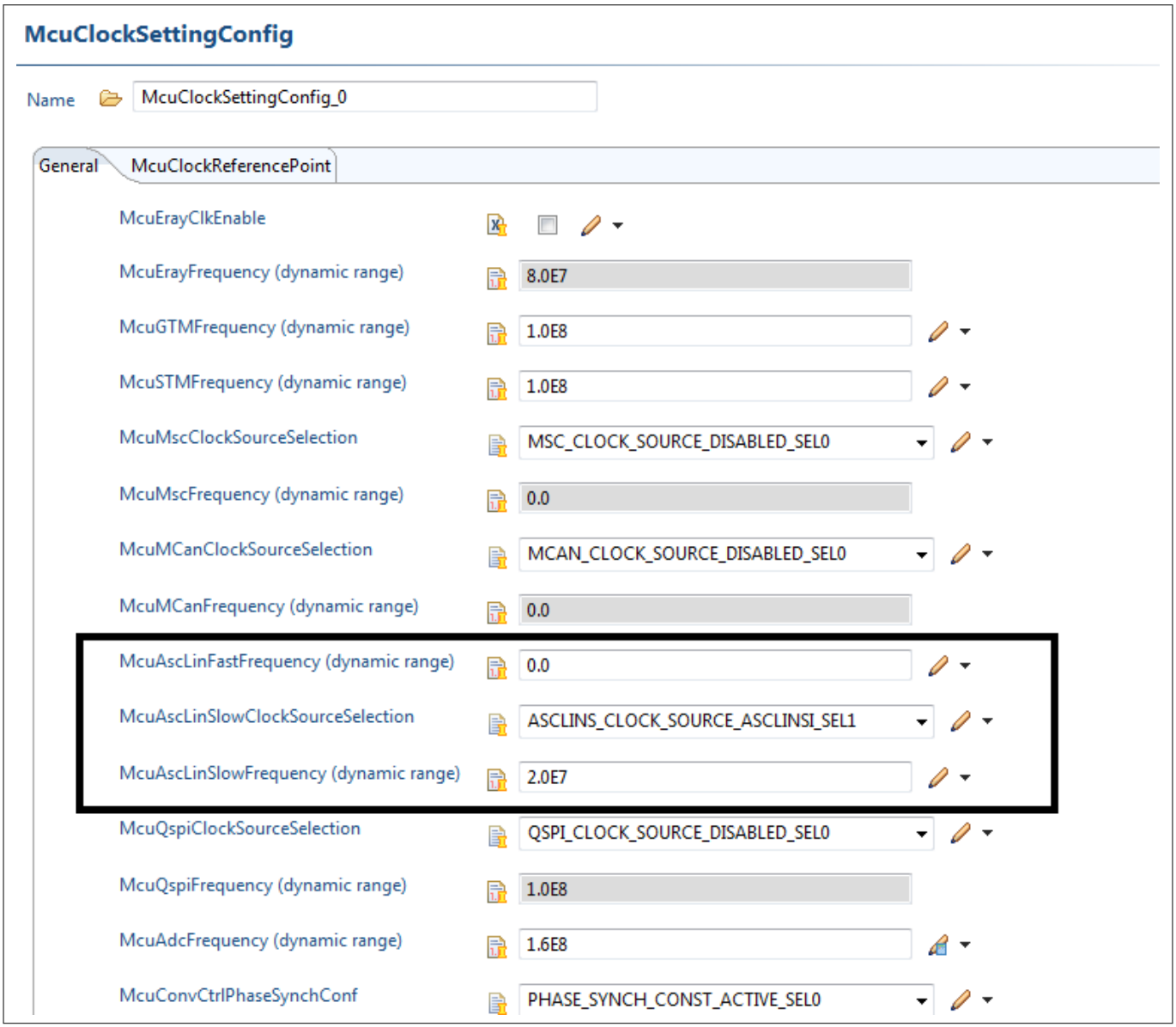


Figure 4 UART fast or slow clock configuration

- The ASCLIN hardware IP is shared between UART and LIN drivers. The resource allocation is configured in the MCU driver. Following is the example of allocation of the ASCLIN0 hardware resource to the UART driver.

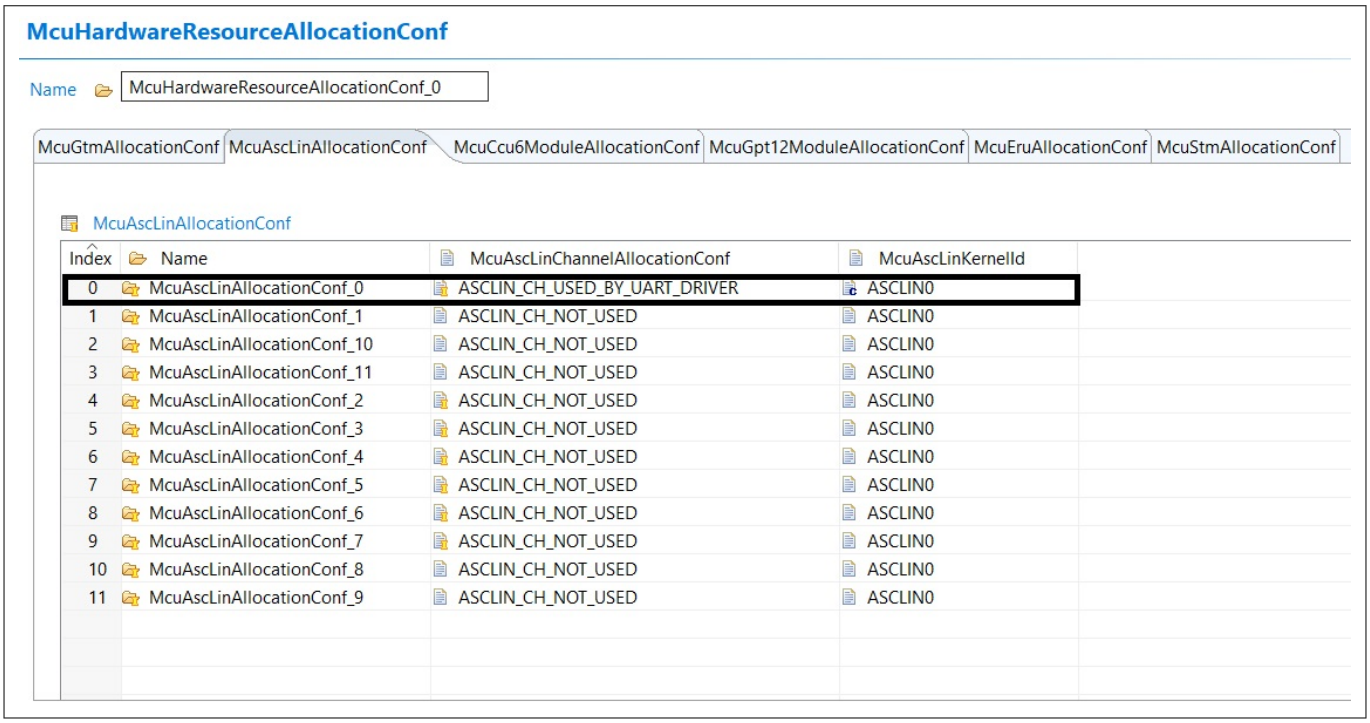


Figure 5 UART channel allocation in MCU driver

1.1.4.4 Port support

The PORT driver configures the port pins of the entire microcontroller. The user must configure port pins used by the UART driver through the PORT configuration and initialize the port pins prior to invoking of the UART initialization. The following must be considered while configuring PORT driver in the EB tresos tool:

- Configure all port pins that are used in the UART driver for RX, TX, CTS and RTS. That is, parameters such as `PortPinDirection` (input or output), `PortPinInitialMode` (as GPIO for input pin or corresponding ALT option for output pins) and so on.

  Refer to the following sample configurations for the PORT driver:

1 Uart driver

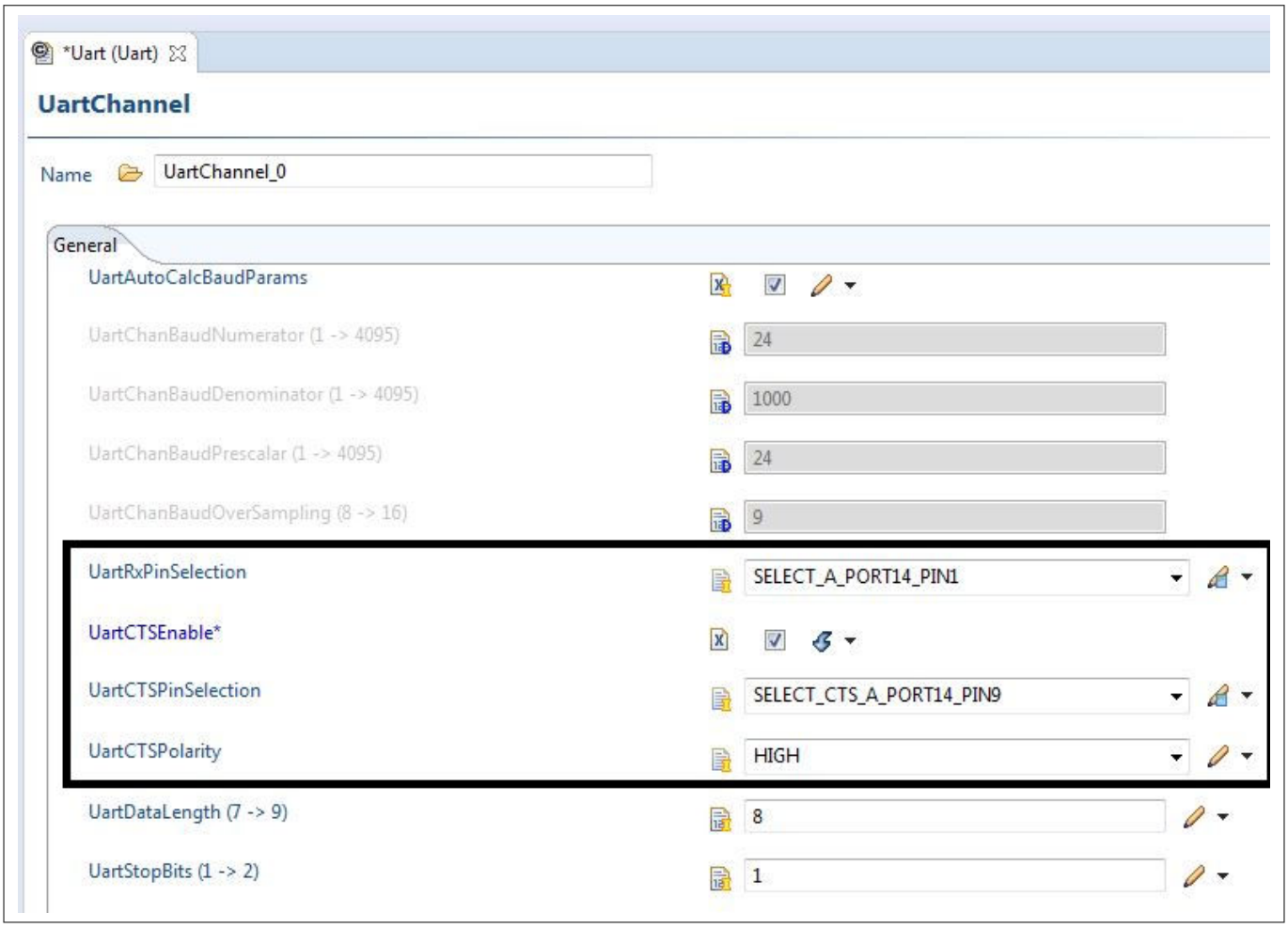
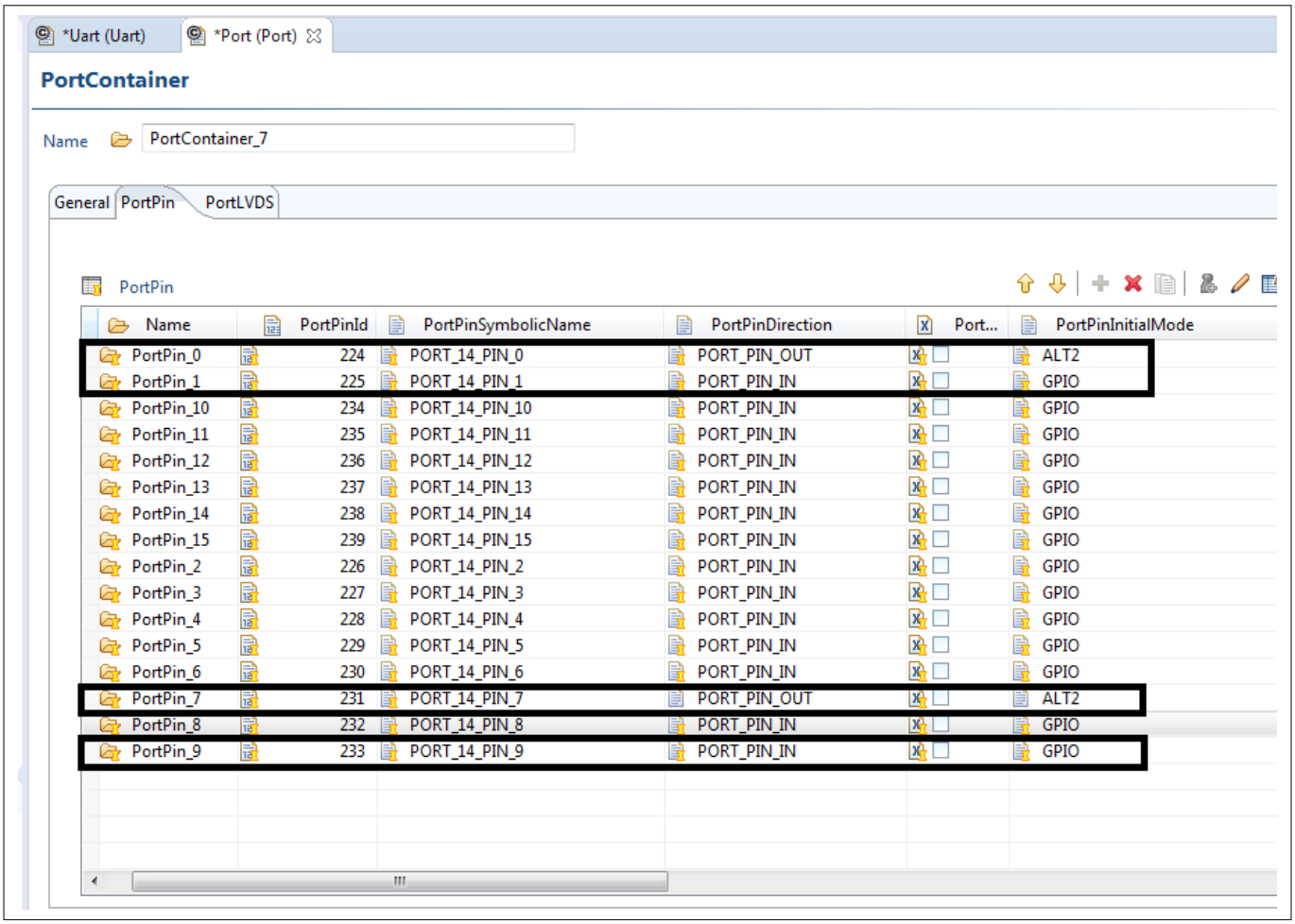


Figure 6 Alternative RX and CTS pin selection for UART channel



Name	PortPinId	PortPinSymbolicName	PortPinDirection	Port...	PortPinInitialMode
PortPin_0	224	PORT_14_PIN_0	PORT_PIN_OUT		ALT2
PortPin_1	225	PORT_14_PIN_1	PORT_PIN_IN		GPIO
PortPin_10	234	PORT_14_PIN_10	PORT_PIN_IN		GPIO
PortPin_11	235	PORT_14_PIN_11	PORT_PIN_IN		GPIO
PortPin_12	236	PORT_14_PIN_12	PORT_PIN_IN		GPIO
PortPin_13	237	PORT_14_PIN_13	PORT_PIN_IN		GPIO
PortPin_14	238	PORT_14_PIN_14	PORT_PIN_IN		GPIO
PortPin_15	239	PORT_14_PIN_15	PORT_PIN_IN		GPIO
PortPin_2	226	PORT_14_PIN_2	PORT_PIN_IN		GPIO
PortPin_3	227	PORT_14_PIN_3	PORT_PIN_IN		GPIO
PortPin_4	228	PORT_14_PIN_4	PORT_PIN_IN		GPIO
PortPin_5	229	PORT_14_PIN_5	PORT_PIN_IN		GPIO
PortPin_6	230	PORT_14_PIN_6	PORT_PIN_IN		GPIO
PortPin_7	231	PORT_14_PIN_7	PORT_PIN_OUT		ALT2
PortPin_8	232	PORT_14_PIN_8	PORT_PIN_IN		GPIO
PortPin_9	233	PORT_14_PIN_9	PORT_PIN_IN		GPIO

Figure 7 Port pin direction and ALT configuration for UART

1.1.4.5 DMA support

The UART driver does not use any services provided by the DMA driver.

1.1.4.6 Interrupt connections

The interrupt connections of the UART driver are described in this section.

- TFL: TXFIFO level interrupt

This interrupt is triggered when TXFIFO transmission is completed successfully. A sample invocation for TFL interrupt handler is shown as follows:

```
#include "Uart.h"
#include "Irq.h"

/********TX Interrupt for ASCLIN0 *********/

IFX_INTERRUPT(ASCLIN0TX_ISR, 0, IRQ_ASCLIN0_TX_PRIO)
ISR(ASCLIN0TX_ISR)
{
 /* Enable Global Interrupts */
 ENABLE();
 /* Call Uart Interrupt function*/
 Uart_IsrTransmit(0U);
}
```

- RFL: RXFIFO level interrupt

This interrupt is triggered when RXFIFO filled up to configured level with received data. A sample invocation for RFL interrupt handler is shown as follows:

```
#include "Uart.h"
#include "Irq.h"

/********RX Interrupt for ASCLIN0 *********/
IFX_INTERRUPT(ASCLIN0RX_ISR, 0, IRQ_ASCLIN0_RX_PRIO)
ISR(ASCLIN0RX_ISR)
{
 /* Enable Global Interrupts */
 ENABLE();
 /* Call Uart Interrupt function*/
 Uart_IsrReceive(0U);
}
```

- ERR/Transmit complete: Receive error or transmit complete

This interrupt is triggered when receive error (Parity, Frame, RXFIFO overflow) event or transmit complete (last stop bit transmitted out from ASCLIN kernel) event occurs. A sample invocation for error interrupt handler is shown as follows:

```
#include "Uart.h"
#include "Irq.h"

/********Err Interrupt for ASCLIN0 *********/
IFX_INTERRUPT(ASCLIN0ERR_ISR, 0, IRQ_ASCLIN0_ERR_PRIO)
ISR(ASCLIN0ERR_ISR)
{
 /* Enable Global Interrupts */
 ENABLE();
 /* Call Uart Interrupt function*/
 Uart_IsrError(0U);
}
```

Configuration of interrupt category and priority shall be configured in the IRQ driver. The following examples show interrupt configurations for ASCLIN0:

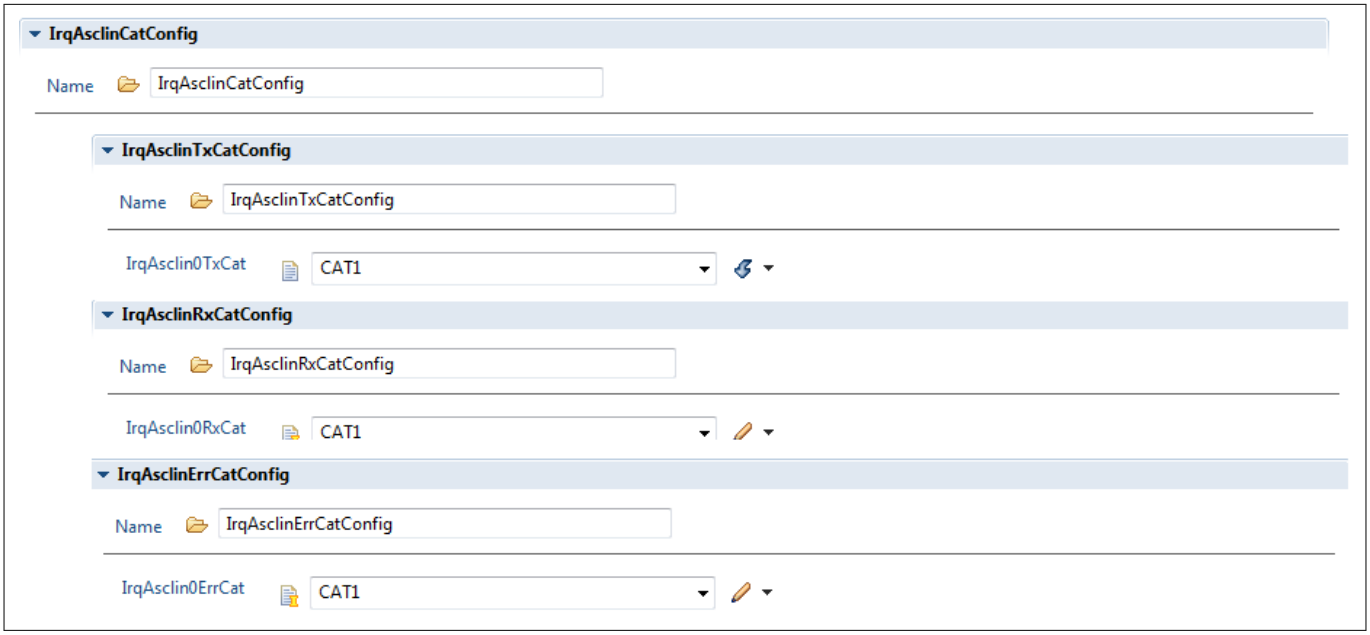


Figure 8 Interrupt category configuration for ASCLIN0

1 Uart driver

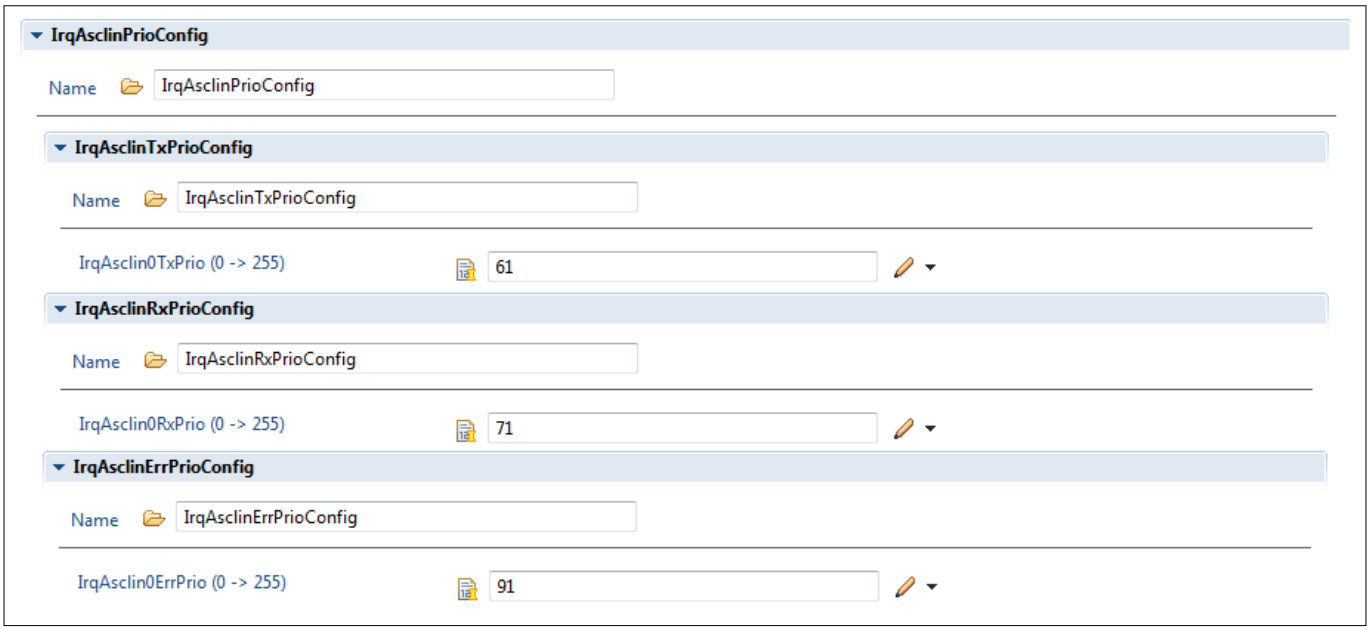


Figure 9 Interrupt priority configuration for ASCLIN0

1.1.4.7 Example usage

The following are some of the key use cases of the UART driver.

Configuration of UART and other modules

- Configuration of system clock: Before using the UART driver, the MCU driver should be configured and initialized so that the system clock is up and running at the required frequency. This configuration is done using the MCU driver.
- Configuration of the port pins: The TXD, RXD, optional CTS, RTS (for the relevant RX and CTS pin select) pins of the UART channels should be configured using the PORT driver.
- Configuration of the UART interrupts: For UART drivers with interrupt mode enabled, configure the interrupt priority, type of service and interrupt type in the IRQ driver.
- Configuration of the UART driver: Select the required API configuration and choose channel dependent parameters like Baud-rate, Tx and Rx mode (polling or interrupt) Stop bits, Parity, Optional CTS selection etc.

UART driver initialization

Refer to the Integration hints section and add all the dependent modules. Follow the sequence in the application code:

1. Initialize the MCU driver and the clock using the `Mcu_Init` API.
2. Initialize the PORT driver using the `Port_Init` API.
3. Initialize the IRQ to enable the interrupt generation.
4. Initialize the UART driver using the `Uart_Init` API.

The sample code for the UART driver initialization is shown as follows:

```
#include "Mcu.h"
#include "Uart.h"
#include "Port.h"
#include "Irq.h"

/* MCU Initialization */
Mcu_Init(&Mcu_Config);
Mcu_InitClock(0U);
while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
Mcu_DistributePllClock();

/* Port initialization */
Port_Init(&Port_Config);

/* Irq initialization */
IrqAsclin_Init();

/* Uart driver initialization */
Uart_Init(&Uart_Config);

/* Check Uart initialization */
RetVal = Uart_InitCheck(&Uart_Config);
if(RetVal == E_NOT_OK)
{
 /* Uart initialization fail */
}

/* Uart driver de-initialization */
Uart_DeInit();
```

UART transmit operation in interrupt mode

The sequence diagram for the UART transmit operation in the interrupt mode is shown as follows:

1 Uart driver

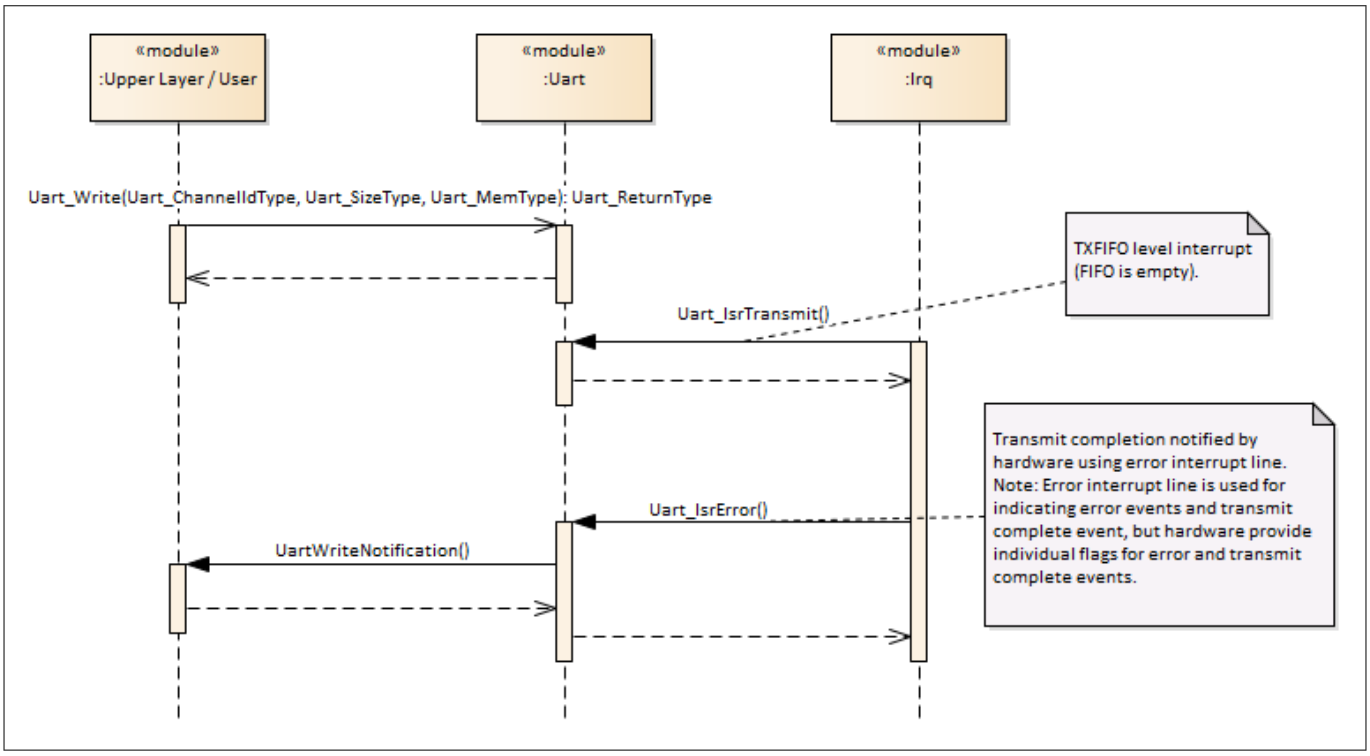


Figure 10 UART transmit operation in interrupt mode

The sample configuration for the UART transmit in the interrupt mode with 8-bit frame length is shown as follows:

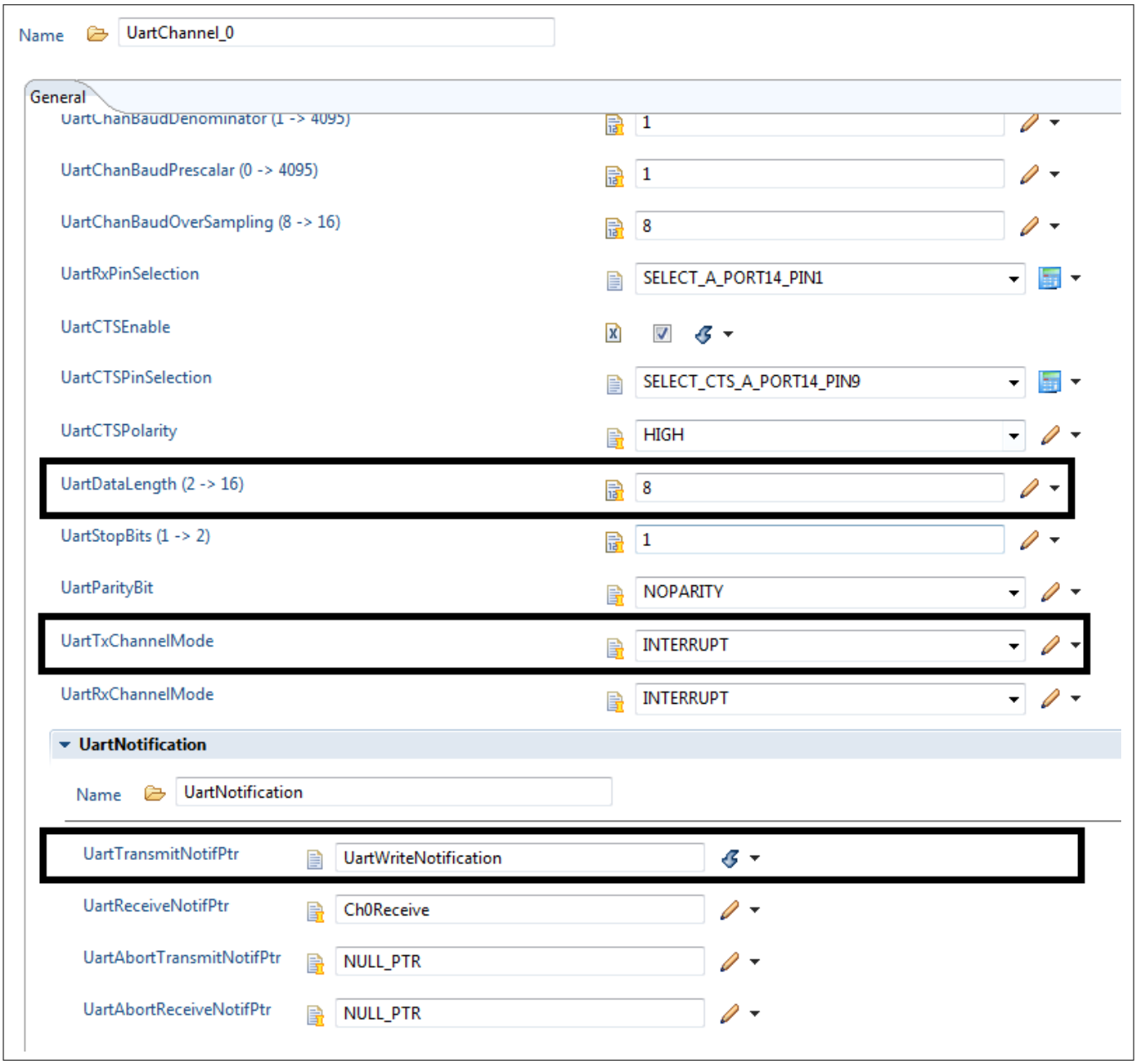


Figure 11 **Configuration: Frame length 8 bits, transmit in interrupt mode, transmit Notification function `UartWriteNotification`**

1 Uart driver

A sample code for transmitting 20 frames with 8-bit frame length in the interrupt mode is as follows:

```
/* Transmit buffer */
uint8 TxBuffer[20] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};

/* Write notification function */
void UartWriteNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* 20 frames transmited successfully */
 }
}

/* Uart write */
Uart_Write(0,&TxBuffer[0],20);
```

UART transmit operation in polling mode

The sequence diagram for the UART transmit operation in the polling mode is shown as follows:

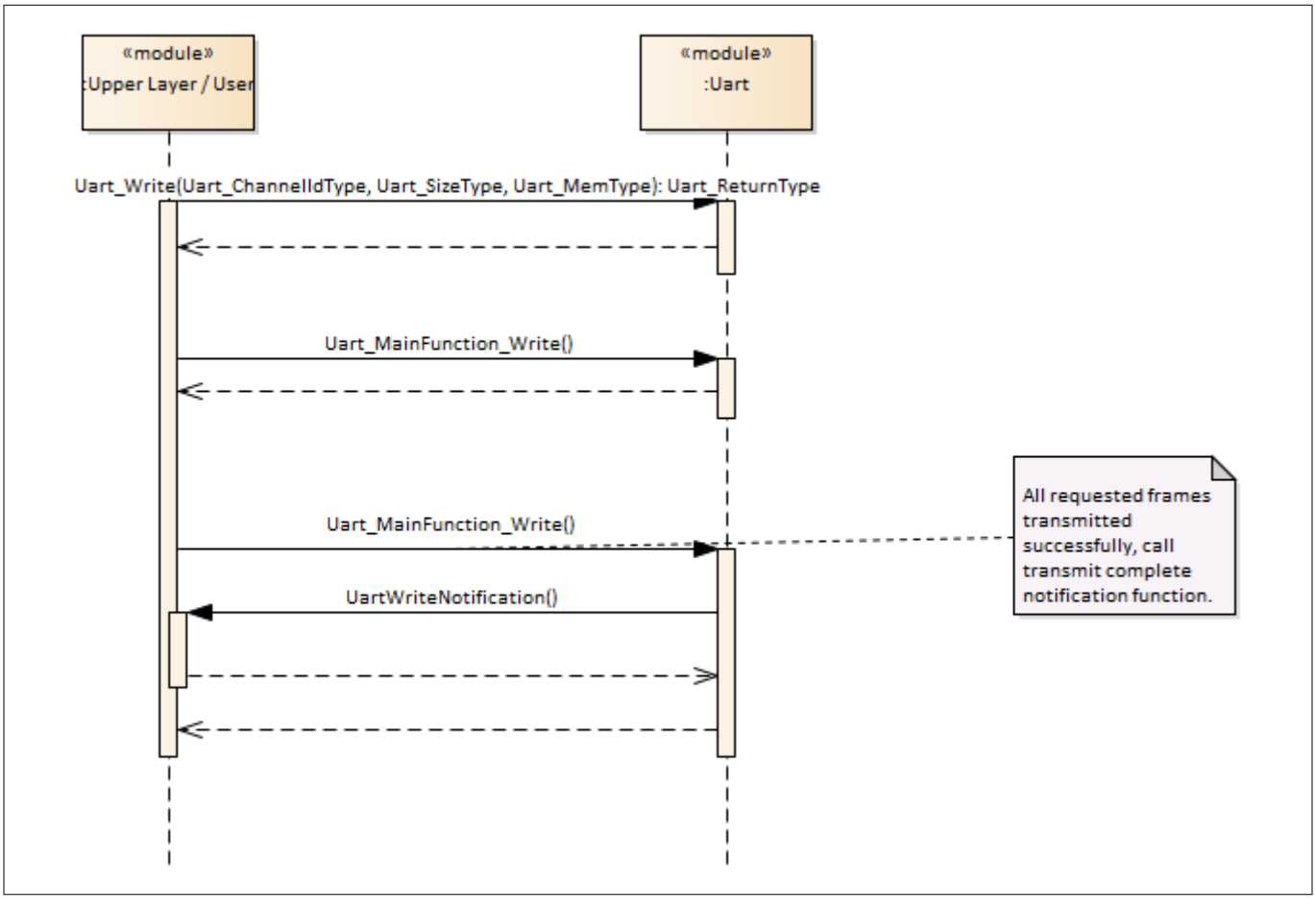


Figure 12 UART transmit operation in polling mode

The sample configuration for transmitting 8-bit frame in the polling mode is shown as follows:

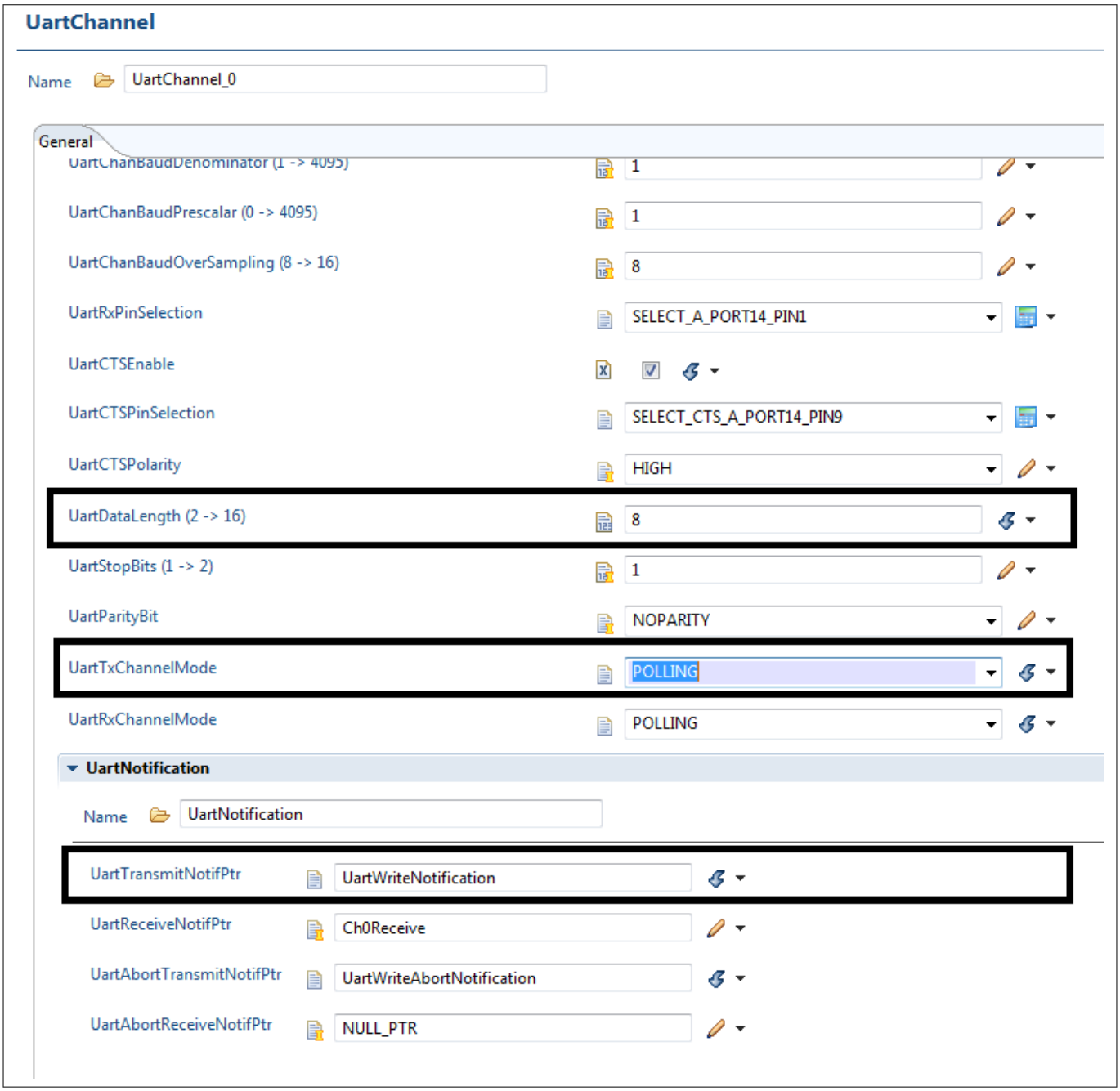


Figure 13 **Configuration: Frame length 8 bits, transmit in polling mode, transmit Notification function `UartWriteNotification`**

A sample code for transmitting 20 frames with 8-bit frame size in the polling mode is as follows:

```
/* Buffer use for transmission */
uint8 TxBuffer[20] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};

/* Write notification function */
void UartWriteNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* Write operation completed without error */
 }
}

/* Uart write */
Uart_Write(0,&TxBuffer[0],20);
/* Poll till transmission is completed */
while(RetVal == UART_BUSY_TRANSMIT)
{
 /* Function to poll data transmission and give notification once transmition
is finished */
 Uart_MainFunction_Write();
 /* Get channel 0 status */
 RetVal = Uart_GetStatus(0);
}
```

UART transmit abort operation

The sequence diagram of UART transmit abort operation is shown as follows:

1 Uart driver

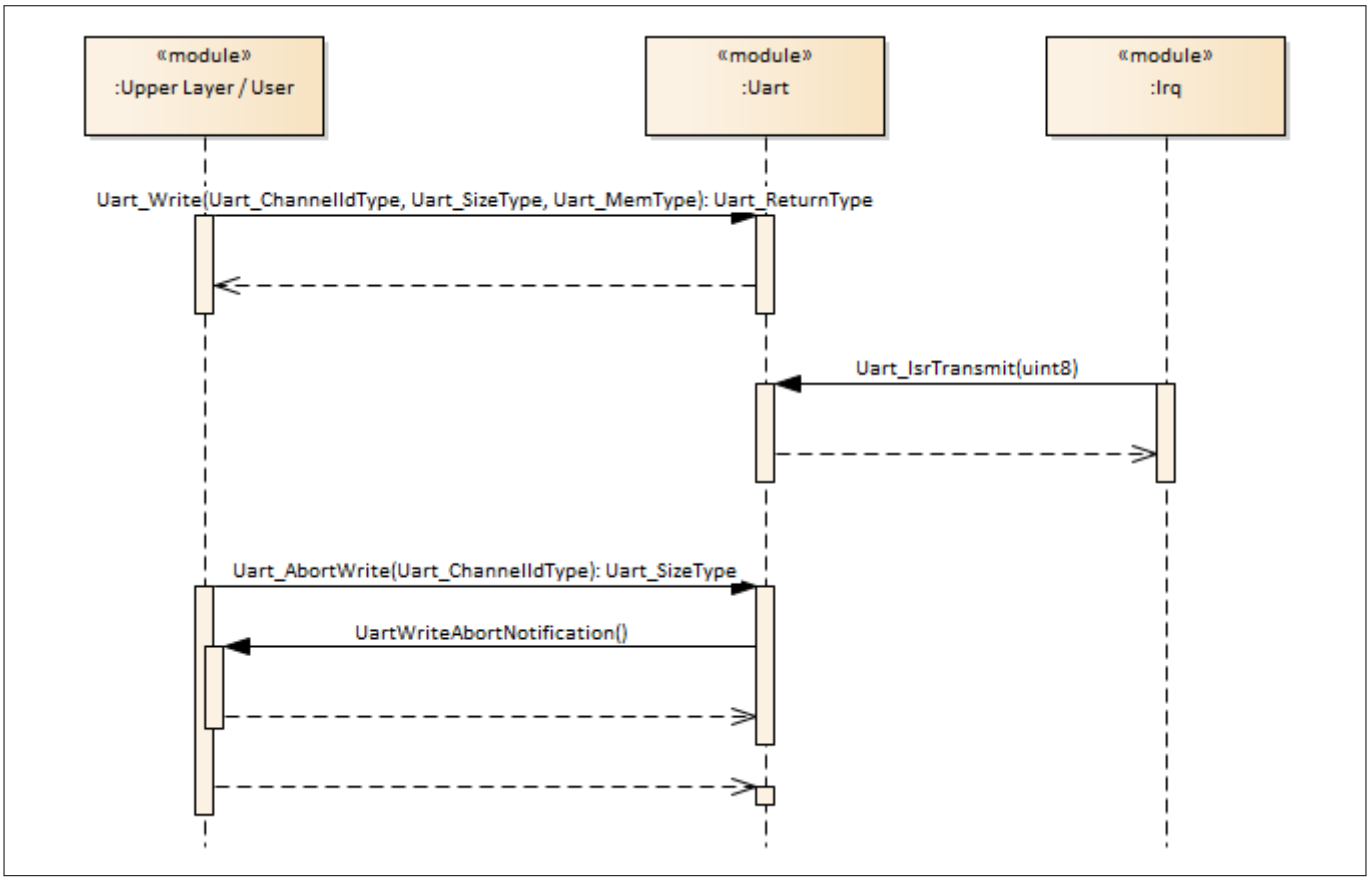


Figure 14 Transmit abort operation

A sample code for the abort transmit operation is as follows:

```
/* Buffer use for transmission */
uint8 TxBuffer[128];
uint16 NumberOfBytesTransmited;

/* Write notification function */
void UartWriteAbortNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* transmit operation aborted successfully */
 }
}

/* Initialize TxBuffer */
for(Counter = 0; Counter < 128; Counter++)
{
 TxBuffer[Counter] = Counter;
}

/* Uart write */
Uart_Write(0,&TxBuffer[0],128);

/* Abort write operation on channel 0 */
NumberOfBytesTransmited = Uart_AbortWrite(0);
```

UART receive operation in interrupt mode

The sequence diagram for the UART receive operation in the interrupt mode is shown as follows:

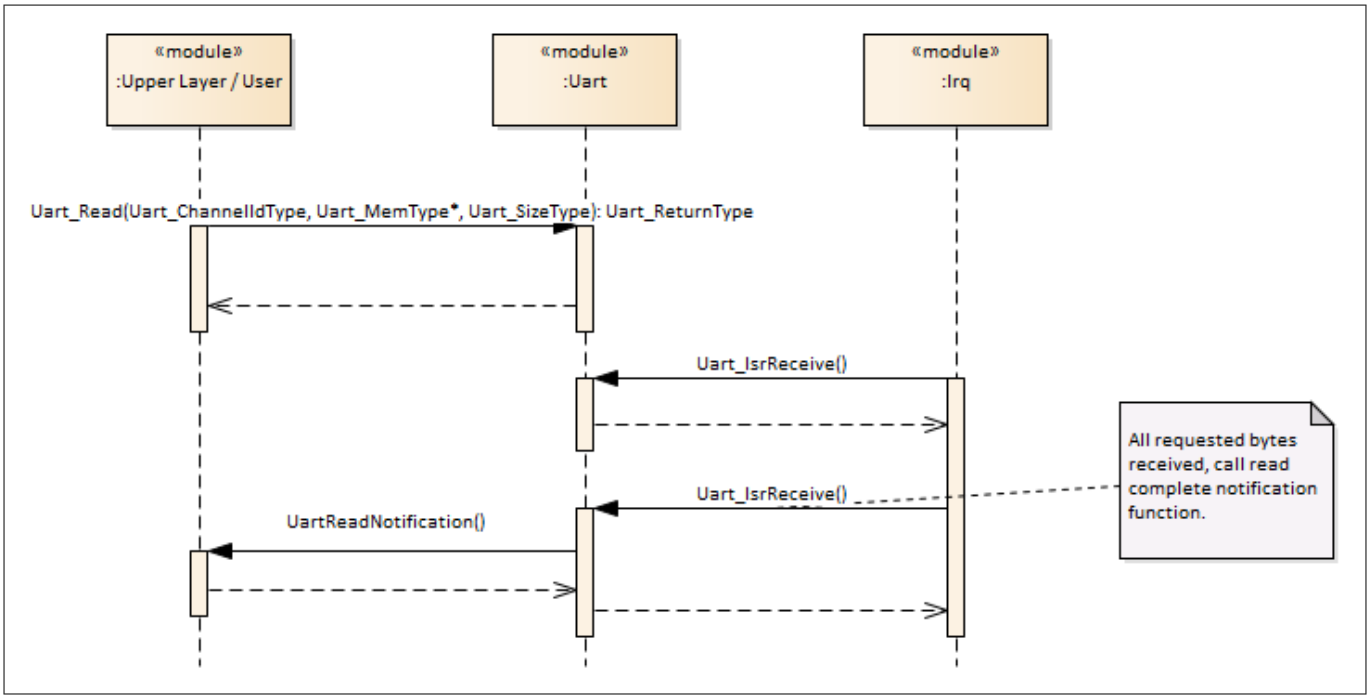


Figure 15 UART receive operation in interrupt mode

The sample configuration for receive 8-bit frame in the interrupt mode is as follows:

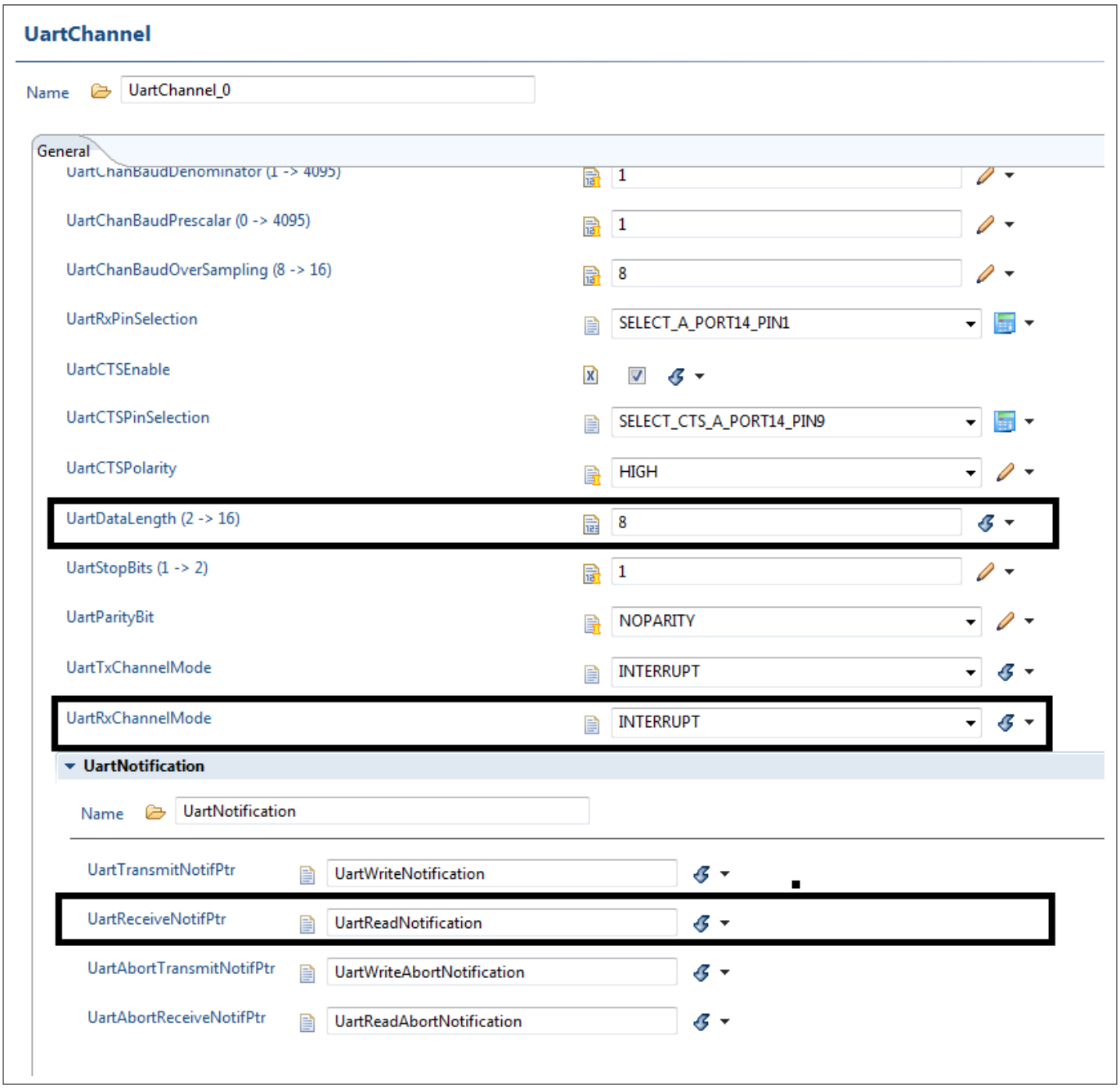


Figure 16 Configuration: Frame length 8 bits, receive in interrupt mode, receive notification function `UartReadNotification`

A sample code for receiving 20 frames with 8-bit frame size in the interrupt mode is as follows:

```
/* Receive buffer */
uint8 RxBuffer[20];

/* Read notification function */
void UartReadNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* 20 frames received without error start process received data */
 }
}


/* Uart read */
Uart_Read(0,&RxBuffer[0],20);
```

UART receive operation in polling mode

The sequence diagram for the UART receive operation in the polling mode is shown as follows:

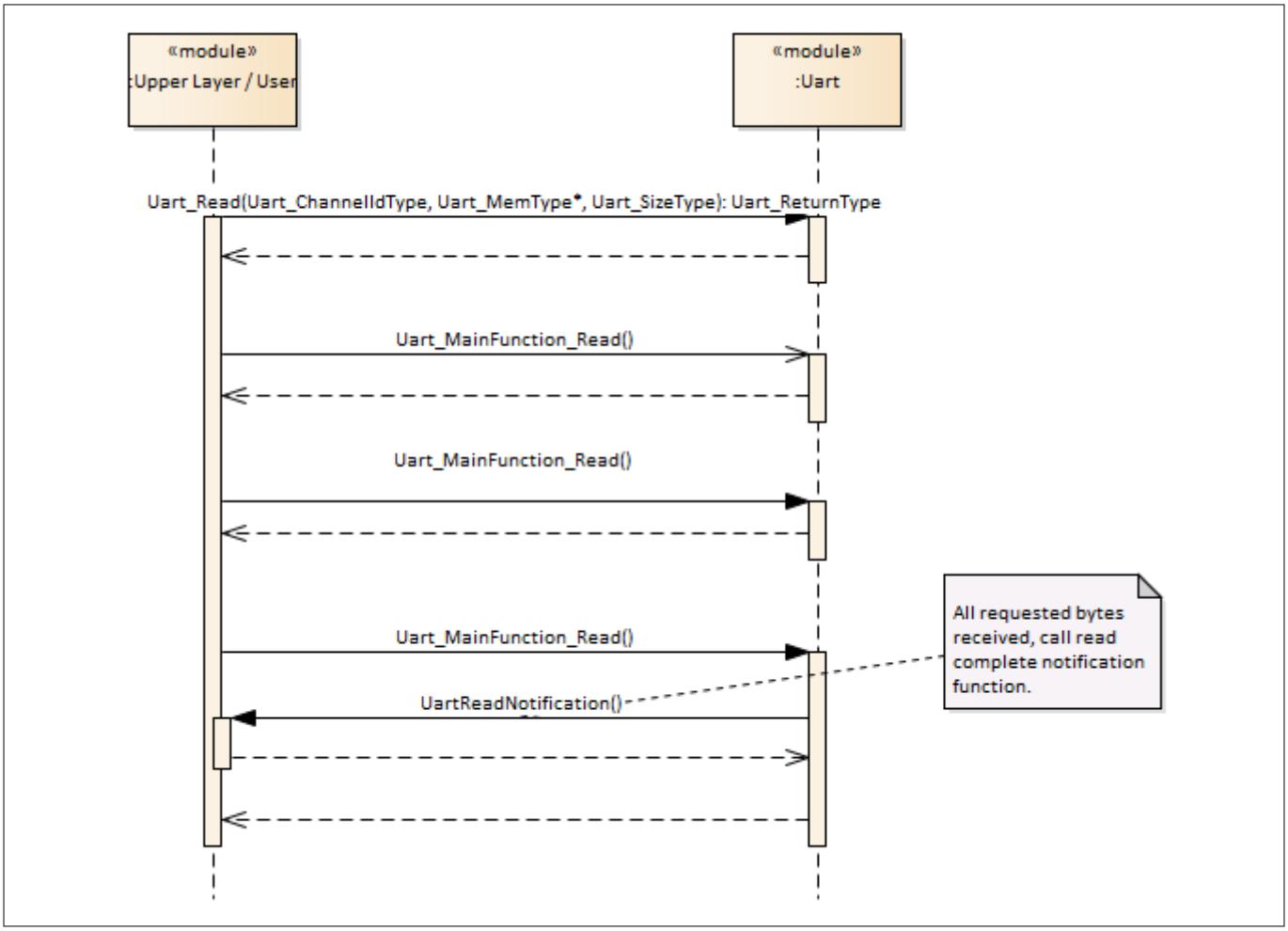


Figure 17 UART receive operation in polling mode

A sample code for receiving 20 frames with 8-bit frame size in the polling mode is as follows:

```
/* Receive buffer */
uint8 RxBuffer[20];

/* Read notification function */
void UartReadNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* 20 frames received without error start process received data */
 }
}


/* Uart read */
Uart_Read(0,&RxBuffer[0],20);
/* Poll till transmission is completed */
while(RetVal == UART_BUSY_RECEIVE)
{
 /* Function to poll data receive and give notification once receivce
operation is finished */
 Uart_MainFunction_Read();
 /* Get channel 0 status */
 RetVal = Uart_GetStatus(0);
}
```

UART abort read operation

The sequence diagram for the UART receive abort operation in the interrupt mode is shown as follows:

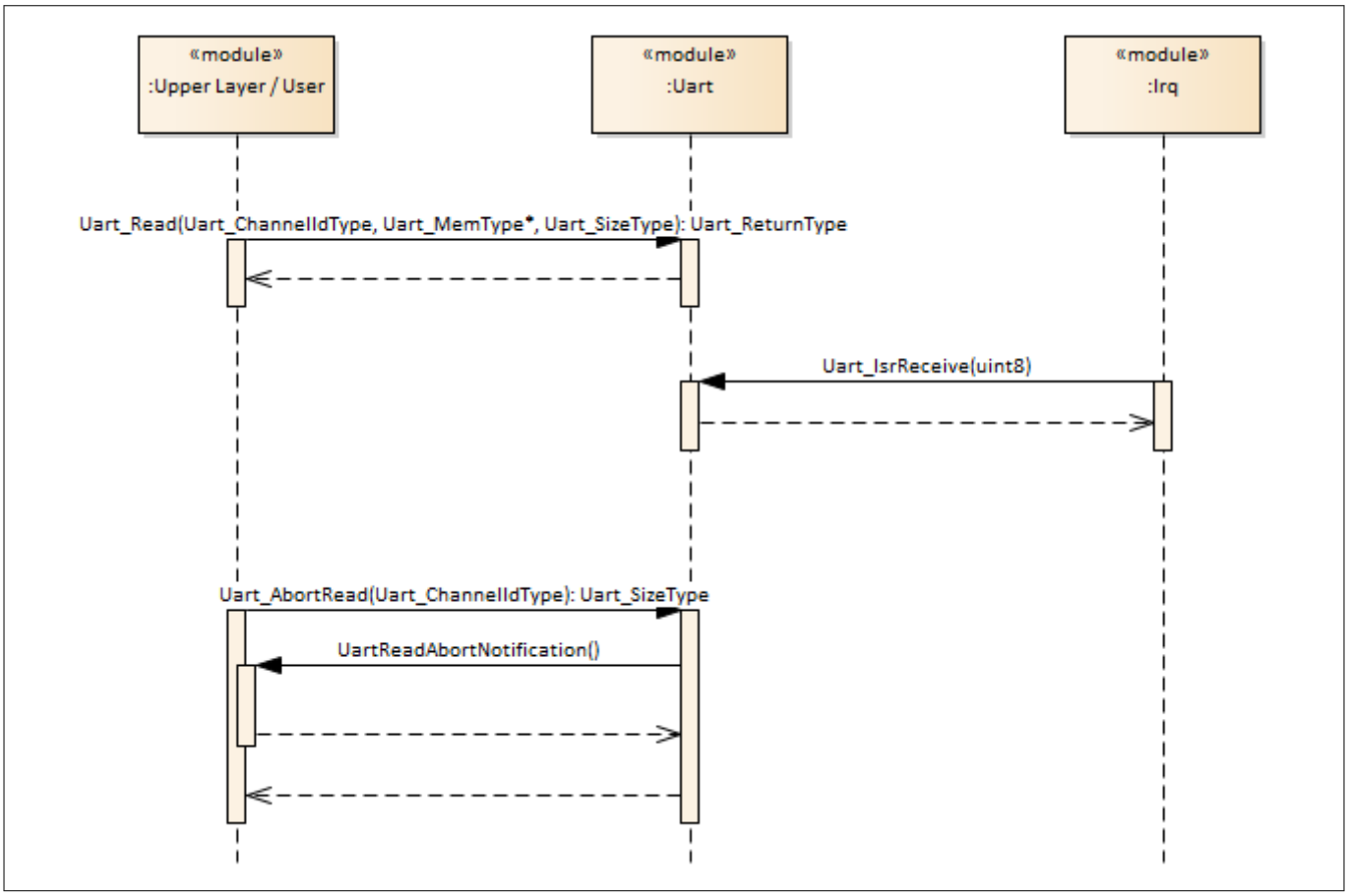


Figure 18 UART abort receive operation

A sample code for the abort receive operation is as follows:

```
/* Receive buffer */
uint8 RxBuffer[128];
uint16 NumberOfBytesReceived;

/* Read abort notification function */
void UartReadAbortNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* Read operation aborted successfully */
 }
}

/* Uart read */
Uart_Read(0,&RxBuffer[0],128);

/* Abort read operation on channel 0 */
NumberOfBytesReceived = Uart_AbortRead(0);
```

1.1.5 Key architectural considerations

1.2 Assumptions of Use (AoU)

The AoU for the UART driver are as follows.

- **Configuration check**

Integrator shall check that configuration code generated is correct for all the configured UART channels.
[cover parentID UART={B3D0ECC3-553D-4743-AC7A-8C6A81DEF4C0}]

- **Address check**

Integrator shall pass valid buffer pointer to transmit/receive data.
[cover parentID UART={E601C70E-216F-42eb-A2E4-DCDC329C91F1}]

- **Freedom from interference for MCAL data**

Integrator shall ensure that there is no interference to the MCAL from other modules.

Rationale: Variables/SFRs can be corrupted by the QM software.
[cover parentID UART={ADFDA904-F0CE-431e-AC65-E1D42A402D37}]

- **Frequency check**

Baudrate parameters of ASCLIN are calculated using configured frequency. Therefore, user shall ensure that the UART driver is invoked only when the operating frequency of ASCLIN is same as the configured frequency. In case of a mismatch, the ASCLIN operates with a different baudrate than the configured value (UartBaudRate).
[cover parentID UART={4C54A9DC-9200-4126-B41F-C8E733371CEF}]

- **Receiver check**

ASCLIN cannot detect errors when data is being shifted from the shift register to the UART pins. Therefore the receiver device shall ensure to have an error detection mechanisms in place.
[cover parentID UART={85AB9E29-DE6F-49fd-B058-BE1A307BC8E5}]

- **Transmission complete notification**

Hardware triggers the interrupt when the last frame is shifted from TXFIFO to the shift register. Hence last frame transmission is not completed when the interrupt is triggered which will call transmit complete notification.

The next frame transmission can be initiated by the user using the Uart_Write API, which will fill the TXFIFO without disturbing the current frame transmission.

However if the peripheral has to be de-initialized the user shall wait for 1 frame duration else the last frame transmission may be stopped.
[cover parentID UART={51BE3A04-5FA3-420d-AC83-D5378ABB7003}]

- **UART driver usage mode**

It is assumed that the user of the UART driver is aware of the number of bytes to be received from the external device as the Uart_Read() API has Size as a parameter. Also it is assumed that the user knows the instance of time when the data is expected to be received from external device and accordingly the Uart_Read() API is invoked. That is driver shall be operated in master configuration and not as a slave or peer device.

[cover parentID UART={D6E91D13-C314-435a-AAB5-716892AA30EF}]

1.3 Reference information

1.3.1 Configuration interfaces

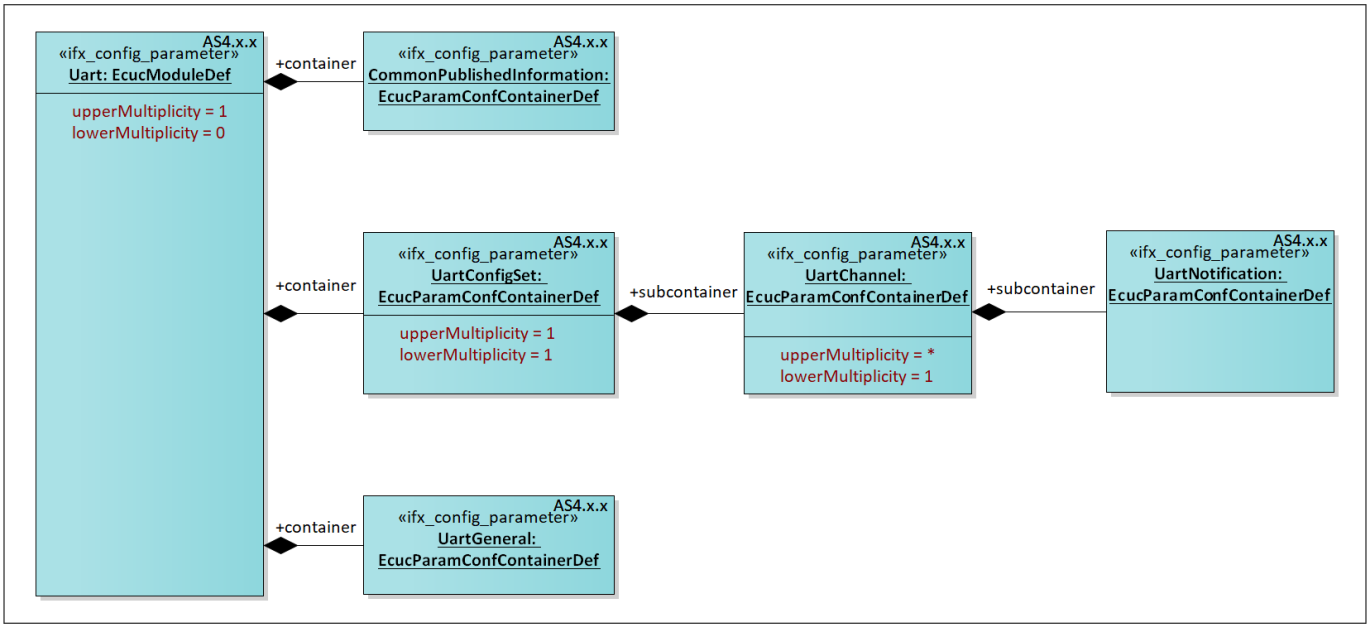


Figure 19 Container hierarchy along with their configuration parameters

1.3.1.1 Container: CommonPublishedInformation

Publish information about module.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.1.1 ArMajorVersion

Table 4 Specification for ArMajorVersion

Name	`ArMajorVersion`		
Description	Major version number of AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

Table 4	Specification for ArMajorVersion (continued)
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.1.2 ArMinorVersion

Table 5	Specification for ArMinorVersion		
Name	`ArMinorVersion`		
Description	Minor version of AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per selected Autosar version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.3 ArPatchVersion

Table 6	Specification for ArPatchVersion		
Name	`ArPatchVersion`		
Description	Patch level version number of AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per selected Autosar version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.4 ModuleId

Table 7 Specification for ModuleId

Name	`ModuleId`		
Description	Module identifier of UART driver from module list.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	255		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.5 Release

Table 8 Specification for Release

Name	`Release`		
Description	Specifies the derivate for which the configuration project is created.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	As per UART driver.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.6 SWMajorVersion

Table 9 Specification for SWMajorVersion

Name	`SWMajorVersion`
Description	Major version number of the implementation of the module.

Table 9 Specification for SWMajorVersion (continued)

Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.7 SWMinorVersion

Table 10 Specification for SWMinorVersion

Name	`SWMinorVersion`		
Description	Minor version number of implementation of the module.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.8 SWPatchVersion

Table 11 Specification for SWPatchVersion

Name	`SWPatchVersion`		
Description	Patch level version number of implementation of the module.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		

Table 11 Specification for SWPatchVersion (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.9 VendorId

Table 12 Specification for VendorId

Name	`VendorId`		
Description	Vendor identifier of dedicated implementation of UART driver according to the AUTOSAR vendor list.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2 Container: UartChannel

This container contains the configuration parameters of UART channel. Maximum number of UART channels varies as per device variant.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.2.1 UartAutoCalcBaudParams

Table 13 Specification for UartAutoCalcBaudParams

Name	`UartAutoCalcBaudParams`
Description	Enable or disable automatic calculation of baud rate parameters (Numerator, Denominator, Pre-scalar and Over sampling) based on the configuration of parameter UartBaudRate.

Table 13 Specification for UartAutoCalcBaudParams (continued)

	User can disable the feature and manually enter the values for baud rate parameters. TRUE: Automatic calculation of baudrate parameters are enabled. FALSE: Automatic calculation of baudrate parameters are disabled.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.2 UartBaudRate

Table 14 Specification for UartBaudRate

Name	`UartBaudRate`		
Description	UART channel transmit and receive baud rate in bits per second. Parameter is applicable if UartAutoCalcBaudParams is enabled. *Note: Default value set to 9600 bits per second as UART standard baud rate.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1000 - 6250000		
Default value	9600		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.3 UartCTSEnable

Table 15 Specification for UartCTSEnable

Name	`UartCTSEnable`		
Description	Enable or disable CTS for UART channel. CTS (clear to transmit) used to notify sender that receiver is ready to receive data. TRUE: CTS is enabled. FALSE: CTS is disabled. *Note: Default CTS is disabled to save hardware resource (port pin) for basic communication.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.4 UartCTSPinSelection

Table 16 Specification for UartCTSPinSelection

Name	`UartCTSPinSelection`		
Description	This parameter selects the alternate input for the CTS select line for the given Uart channel. *Note: The first available data line for configured ASCLIN HW unit is selected as default value.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	SELECT_CTS _X_PORTY_PINZ: SELECT_CTS _X_PORTY_PINZ: This parameter varies in availability as per configured Uart channel, and device variant, where x signifies data line, Y signifies port number and Z signifies pin number. Values of X, Y, Z will be extracted from property file. For example: SELECT_CTS_A_PORT14_PIN9. SELECT_CTS _X_PORTY_PINZ: NONE: This option is chosen to indicate no CTS pin is selected for Uart driver.		
Default value	SELECT_CTS_A_PORT14_PIN9		

Table 16 Specification for UartCTSPinSelection (continued)

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartCTSEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.5 UartCTSPolarity

Table 17 Specification for UartCTSPolarity

Name	`UartCTSPolarity`		
Description	Parameter decides active polarity of CTS pin. Parameter applicable if UartCTSEnable is enabled. *Note: Default polarity set with HIGH.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	HIGH: CTS is considered to be active when the signal is HIGH. LOW: CTS is considered to be active when the signal is LOW.		
Default value	HIGH		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartCTSEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.6 UartChanBaudDenominator

Table 18 Specification for UartChanBaudDenominator

Name	`UartChanBaudDenominator`
Description	Specifies the BRG register denominator value used for Baudrate calculation. The value configured in this parameter will be written to the BRG.DENOMINATOR register field. Baud rate is derived based on the below formula. fPD = fA / (BITCON.PRESCALER +1) fOVS = fPD * BRG.NUMERATOR / BRG.DENOMINATOR

Table 18 Specification for UartChanBaudDenominator (continued)

	fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1) fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams is enabled then value of this parameter calculated internally. Default value set 1000 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 4095		
Default value	1000		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.7 UartChanBaudNumerator

Table 19 Specification for UartChanBaudNumerator

Name	`UartChanBaudNumerator`		
Description	Specifies the BRG register numerator value used for Baudrate calculation. The value configured in this parameter will be written to the BRG.NUMERATOR register field. Baud rate is derived based on the below formula. fPD = fA / (BITCON.PRESCALER + 1) fOVS = fPD* BRG.NUMERATOR / BRG.DENOMINATOR fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1) fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams is enabled then value of this parameter calculated internally. Default value set 24 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 4095		
Default value	24		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-

Table 19 Specification for UartChanBaudNumerator (continued)

Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.8 UartChanBaudOverSampling

Table 20 Specification for UartChanBaudOverSampling

Name	`UartChanBaudOverSampling`		
Description	Specifies the BITCON register over sampling value used for Baudrate calculation. The value configured in this parameter will be written to the BITCON.OVERSAMPLING register field. Baud rate is derived based on the below formula. fPD = fA/ (BITCON.PRESCALER + 1) fOVS = fPD * BRG.NUMERATOR / BRG.DENOMINATOR fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1). fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams enabled then value of parameter calculated internally. Default value set 9 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	3 - 15		
Default value	9		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.9 UartChanBaudPrescalar

Table 21 Specification for UartChanBaudPrescalar

Name	`UartChanBaudPrescalar`
Description	Specifies the BITCON register prescalar value used for Baudrate calculation. The value configured in this parameter will be written to the BITCON.PRESCALAR register field. Baud rate is derived based on the below formula. fPD = fA / (BITCON.PRESCALER + 1) fOVS = fPD * BRG.NUMERATOR / BRG.DENOMINATOR

Table 21 Specification for UartChanBaudPrescalar (continued)

	fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1). fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams is enabled then value of this parameter calculated internally. Default value set 4 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4095		
Default value	4		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.10 UartChannelId

Table 22 Specification for UartChannelId

Name	`UartChannelId`		
Description	UART channel logical identifier. Upper limit of the channel identifier varies as per device variant. *Note: Minimum value of the parameter set as default value.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - *		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.11 UartDataLength

Table 23 Specification for UartDataLength

Name	`UartDataLength`		
Description	Parameter decides the frame length of UART channel. *Note: Default frame size set as 8 because commonly used.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	2 - 16		
Default value	8		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.12 UartHwUnit

Table 24 Specification for UartHwUnit

Name	`UartHwUnit`		
Description	Parameter specify ASCLIN hardware channel configured for logical channel. Maximum number of ASCLIN channel depends on device variant. *Note: Default value is set with parameter minimum value.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	ASCLIN0: Hardware channel ASCLIN0. ASCLINx: Hardware channel varies as per device variant from ASCLIN1 to ASCLINx where x is maximum number of ASCLIN channel supported by the device.		
Default value	ASCLIN0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.13 UartParityBit

Table 25 Specification for UartParityBit

Name	`UartParityBit`		
Description	Parameter decides type of parity in data frame. *Note: Default type set with no parity to reduce frame size.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	EVENPARITY: Parity bit set with 1 when even number of 1's in data frame. NOPARITY: Parity bit not present in data frame. ODDPARITY: Parity bit set with 1 when odd number of 1's present in data frame.		
Default value	NOPARITY		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.14 UartRxChannelMode

Table 26 Specification for UartRxChannelMode

Name	`UartRxChannelMode`		
Description	UART channel receive operation configuration mode. *Note: Default set in interrupt mode to disable optional interface (schedule function will enable in case any channel configured in polling mode).*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: UART channel receive operation in interrupt mode. POLLING: UART channel receive operation in polling mode.		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		

Table 26	Specification for UartRxChannelMode (continued)
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.2.15 UartRxPinSelection

Table 27 Specification for UartRxPinSelection

Name	`UartRxPinSelection`		
Description	This parameter selects the alternate input for the receive signal for the given Uart channel. *Note: The first available data line for configured ASCLIN HW unit is selected as default value.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	SELECT_X_PORTY_PINZ: SELECT_X_PORTY_PINZ: This parameter varies in availability as per configured Uart channel, and device variant, where x signifies data line, Y signifies port number and Z signifies pin number. Values of X, Y, Z will be extracted from property file. For example SELECT_A_PORT14_PIN1.		
Default value	SELECT_A_PORT14_PIN1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.16 UartStopBits

Table 28 Specification for UartStopBits

Name	`UartStopBits`		
Description	This parameter is used for selecting the number of stop bits configuration in data frame. *Note: Default value set with 1 bit to reduce frame size.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 2		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-

Table 28 Specification for UartStopBits (continued)

Origin	IFX	**Scope**	None
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.17 UartTxChannelMode

Table 29 Specification for UartTxChannelMode

Name	`UartTxChannelMode`		
Description	UART channel transmit operation mode. *Note: Default set in interrupt mode to disable optional interface (schedule function will enable in case any channel configured in polling mode).*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: UART channel transmit operation in interrupt mode. POLLING: UART channel transmit operation in polling mode.		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3 Container: UartConfigSet

This container contains the channel configuration of the UART driver. This container is a multiple configuration container.This container and it's sub-containers exist once per configuration set.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.4 Container: UartGeneral

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.4.1 UartAbortReadApi

Table 30 Specification for UartAbortReadApi

Name	`UartAbortReadApi`		
Description	Switch to enable or disable abort read feature. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.2 UartAbortWriteApi

Table 31 Specification for UartAbortWriteApi

Name	`UartAbortWriteApi`		
Description	Switch to enable or disable abort write feature. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.3 UartClockRef

Table 32 Specification for UartClockRef

Name	`UartClockRef`		
Description	This parameter refers to the system clock configured by MCU driver. This reference is used for BaudRate computation. *Note: Since the name of the dependent container is user configurable, the default value is kept as NULL.*		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: McuAscLinChannelAllocationConf, McuClockReferencePointConfig		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.4 UartCsrClksel

Table 33 Specification for UartCsrClksel

Name	`UartCsrClksel`		
Description	This parameter selects the baud rate logic clock for the UART driver. *Note: Default value set with fast mode.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	ASCLINF: McuAscLinFastFrequency is selected as input frequency of ASCLIN. ASCLINS: McuAscLinSlowFrequencyis selected as input frequency of ASCLIN.		
Default value	ASCLINF		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.5 UartDeInitApi

Table 34 Specification for UartDeInitApi

Name	`UartDeInitApi`		
Description	Switch to enable or disable UART driver de-init feature. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.6 UartDevErrorDetect

Table 35 Specification for UartDevErrorDetect

Name	`UartDevErrorDetect`		
Description	Switches the Default Error Tracer (Det) detection and notification ON or OFF. TRUE: enabled (ON). FALSE: disabled (OFF).		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.7 UartIndex

Table 36 Specification for UartIndex

Name	`UartIndex`		
Description	Specifies the instance identifier of this module instance. In case single instance is present value should be 0. *Note: Default value set minimum because single instance is present.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.8 UartInitCheckApi

Table 37 Specification for UartInitCheckApi

Name	`UartInitCheckApi`		
Description	Parameter adds or removes the Uart_InitCheck() API from the code. *Note: The default value of this parameter is set to false to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.9 UartInitDeInitApiMode

Table 38 Specification for UartInitDeInitApiMode

Name	`UartInitDeInitApiMode`		
Description	Configuration parameter defines the privilege mode in which the initialization and de-initialization API's operate. *Note: Since UART driver accesses the SFRs, it is more efficient to operate the UART driver in supervisor mode. Hence, the default mode of operation is supervisor.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	UART_MCAL_SUPERVISOR: Init and De-init APIs operate in supervisory mode. UART_MCAL_USER1: Init and De-init APIs operate in USER1 mode.		
Default value	UART_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.10 UartMainFunctionReadPeriod

Table 39 Specification for UartMainFunctionReadPeriod

Name	`UartMainFunctionReadPeriod`		
Description	Specifies the period of main function Uart_MainFunction_Read in seconds. UART driver does not require this information but the BSW schedule will use this information.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0 - 10.0		
Default value	0.005		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.11 UartMainFunctionWritePeriod

Table 40 Specification for UartMainFunctionWritePeriod

Name	`UartMainFunctionWritePeriod`		
Description	Specifies the period of main function Uart_MainFunction_Write in seconds. UART driver does not require this information but the BSW schedule will use this information.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0 - 10.0		
Default value	0.005		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.12 UartRunTimeErrorDetect

Table 41 Specification for UartRunTimeErrorDetect

Name	`UartRunTimeErrorDetect`		
Description	The activation of runtime errors is configurable (ON / OFF) at pre-compile time. *Note: The detection of runtime related errors is enabled by default to ensure that runtime issues are addressed during the product lifecycle.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.13 UartSafetyEnable

Table 42 Specification for UartSafetyEnable

Name	`UartSafetyEnable`		
Description	Switch to enable or disable the safety check. TRUE: Enable safety check FALSE: Disable safety check. *Note: The detection of safety related errors is enabled by default to ensure that safety issues are addressed during the product lifecycle.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.14 UartSleepEnable

Table 43 Specification for UartSleepEnable

Name	`UartSleepEnable`		
Description	Switch enable/disable the ASCLIN module sleep request handling by setting EDIS bit in CLC register. MCU API can request for sleep mode. Refer MCU design specification for more details. TRUE: EDIS bit is set to 1 in CLC register, sleep mode request can be recognized by ASCLIN module and enter in sleep mode. FALSE: EDIS is set to 0, a sleep mode request is ignore and module continues its operation. *Note: The optional feature is disabled by default.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

Table 43 Specification for UartSleepEnable (continued)

Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.15 UartTimeoutCount

Table 44 Specification for UartTimeoutCount

Name	`UartTimeoutCount`		
Description	Specifies the maximum time in nanoseconds to wait for hardware timeout errors. *Note: UartTimeoutCount uses the STM timer current resolution and calculate maximum number of ticks to wait before expected hardware behaviour is occurred during initialization and deinitialization of Uart driver.* *Maximum value is kept as default value for this parameter.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	100 - 4294967295		
Default value	4294967295		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.16 UartVersionInfoApi

Table 45 Specification for UartVersionInfoApi

Name	`UartVersionInfoApi`		
Description	Switch to enable or disable get version information API. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		

Table 45	Specification for UartVersionInfoApi (continued)		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5 Container: UartNotification

This section lists all the notification and callbacks of the Uart driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.5.1 UartAbortReceiveNotifPtr

Table 46	Specification for UartAbortReceiveNotifPtr		
Name	`UartAbortReceiveNotifPtr`		
Description	Parameter which holds receive abort notification function address. Definition of function present in application. If the user does not require notification then parameter shall be configured to NULL_PTR. *Note: Optional interface so default value set NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtrType
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAbortReadApi		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.2 UartAbortTransmitNotifPtr

Table 47 Specification for UartAbortTransmitNotifPtr

Name	`UartAbortTransmitNotifPtr`		
Description	Parameter holds transmit abort notification function address. Definition of function present in application. If the user does not require notification then parameter shall be configured to NULL_PTR. *Note: Optional interface so default value set NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtrType
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAbortWriteApi		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.3 UartReceiveNotifPtr

Table 48 Specification for UartReceiveNotifPtr

Name	`UartReceiveNotifPtr`		
Description	Parameter holds receive complete notification function address. Definition of function present in application. If the user does not require notification then parameter shall be configured with NULL_PTR. *Note: Optional interface so default value set with NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtrType
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL

Table 48	Specification for UartReceiveNotifPtr (continued)
Dependency	-
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.5.4 UartTransmitNotifPtr

Table 49	Specification for UartTransmitNotifPtr		
Name	`UartTransmitNotifPtr`		
Description	Parameter holds transmit complete notification function address. Definition of function present in application. If the user does not require notification then parameter should be configured with NULL_PTR. *Note: Optional interface so default value set NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtrType
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.6 Container: Uart

Post-Build Variant Multiplicity: FALSE
Multiplicity Configuration Class: -

1.3.2 Functions - Type definitions

This section lists all the data types of the Uart driver.

1.3.2.1 Uart_ChannelIdType

Table 50	Specification for Uart_ChannelIdType
Syntax	`Uart_ChannelIdType`
Type	uint8
File	`Uart.h`

Table 50 Specification for Uart_ChannelIdType (continued)

Range	0-255	
Description	Data type used to specifies logical channel identifier of UART.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.2 Uart_ConfigType

Table 51 Specification for Uart_ConfigType

Syntax	`Uart_ConfigType`	
Type	Structure	
File	`Uart.h`	
Range	--	The elements of the data structure are specific to the microcontroller.
Description	Data type used to specify UART driver configuration.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.3 Uart_ErrorIdType

Table 52 Specification for Uart_ErrorIdType

Syntax	`Uart_ErrorIdType`	
Type	Enumeration	
File	`Uart.h`	
Range	0 - UART_E_NO_ERR	No error.
	1 - UART_E_PARITY_ERR	Parity error.
	2 - UART_E_FRAME_ERR	Frame error.
	3 - UART_E_RXOVERFLOW_ERR	RXFIFO overflow error.
Description	Data type specifies the error occurred during the data transmission or reception.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.4 Uart_MemType

Table 53 Specification for Uart_MemType

Syntax	`Uart_MemType`
Type	uint8

Table 53 Specification for Uart_MemType (continued)

File	Uart.h	
Range	0-255	
Description	Data type of the buffer used in read and writes operation.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.5 Uart_NotificationPtrType

Table 54 Specification for Uart_NotificationPtrType

Syntax	Uart_NotificationPtrType
Type	Pointer to a function of type void Function_Name (const Uart_ErrorIdType ErrorId)
File	Uart.h
Description	Data type to specify function pointer declaration of UART call back.
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.6 Uart_ReturnType

Table 55 Specification for Uart_ReturnType

Syntax	Uart_ReturnType	
Type	Enumeration	
File	Uart.h	
Range	0 - UART_E_OK	API successful completed.
	1 - UART_E_NOT_OK	API reported development error.
	2 - UART_E_BUSY	UART channel is busy in same operation which is requested by API.
Description	Data type used to specify the return value of Uart driver API.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.7 Uart_SizeType

Table 56 Specification for Uart_SizeType

Syntax	Uart_SizeType
Type	uint16
File	Uart.h

Table 56 Specification for Uart_SizeType (continued)

Range	0-65535	
Description	Data type used to specify the number of bytes to be transmit or receive.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.8 Uart_StatusType

Table 57 Specification for Uart_StatusType

Syntax	`Uart_StatusType`	
Type	Enumeration	
File	`Uart.h`	
Range	0 - UART_IDLE	Idle state (no transmits or receives operation in progress).
	1 - UART_BUSY_TRANSMIT	UART channel busy in transmit operation.
	2 - UART_BUSY_RECEIVE	UART channel busy in receive operation.
	3 - UART_BUSY_TRANSMIT_RECEIVE	UART channel busy in receive and transmit operation.
Description	Data type used to specify UART channel status.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3 Functions - APIs

This section lists all the APIs of the UART driver.

1.3.3.1 Uart_InitCheck

Table 58 Specification for `Uart_InitCheck` API

Syntax	`Std_ReturnType Uart_InitCheck ( const Uart_ConfigType * const ConfigPtr )`	
Service ID	0xD8	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Address of UART driver configuration set.

Table 58 Specification for `Uart_InitCheck` API (continued)

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Initialization check passed. E_NOT_OK: Initialization check failed.
Description	API returns the status of the modules initialization. API (optional API) is available only when the parameter UartInitCheckApi is enabled. *Note: Init check should be performed in the following sequence:* *1. Call Uart_Init.* *2. Call Uart_InitCheck.*	
Source	IFX	
Error handling	UART_E_PARAM_POINTER, UART_E_UNINIT	
Configuration dependencies	UartInitCheckApi	
User hints	-	
SFR accessed	ASCLIN_BITCON(r), ASCLIN_BRG(r), ASCLIN_CLC(r), ASCLIN_DATCON(r), ASCLIN_FRAMECON(r), ASCLIN_IOCR(r), ASCLIN_RXFIFOCON(r), ASCLIN_TXFIFOCON(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.2 Uart_Init

Table 59 Specification for `Uart_Init` API

Syntax	`void Uart_Init`
	`(`
	`  const Uart_ConfigType * const ConfigPtr`
	`)`	

Service ID	0xD7	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Address of UART driver configuration set.
Parameters (out)	-	-

Table 59	Specification for `Uart_Init` API (continued)	
Parameters (in - out)	-	-
Return	void	-
Description	API to initialize all configured ASCLIN hardware units with the values referenced by the parameter ConfigPtr.	
Source	IFX	
Error handling	UART_E_ALREADY_INITIALIZED, UART_E_INIT_FAILED	
Configuration dependencies	-	
User hints	-	
SFR accessed	ASCLIN_BITCON(w), ASCLIN_BRG(w), ASCLIN_CLC(rw), ASCLIN_CSR(rw), ASCLIN_DATCON(w), ASCLIN_FRAMECON(w), ASCLIN_IOCR(w), ASCLIN_KRST0(rw), ASCLIN_KRST1(rw), ASCLIN_KRSTCLR(rw), ASCLIN_RXFIFOCON(w), ASCLIN_TXFIFOCON(w), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.3 Uart_Read

Table 60	Specification for `Uart_Read` API	

Syntax	`Uart_ReturnType  Uart_Read`
	`(`
	`  const Uart_ChannelIdType Channel,`
	`  Uart_MemType * const MemPtr,`
	`  const Uart_SizeType Size`
	`)`	

Service ID	0xD9	
Sync/Async	Asynchronous	
ASIL Level	B	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel Size	UART channel id. Number of bytes to be read. Note: If channel frame length configured with greater than 8 bit then number of bytes should be multiple of 2.
Parameters (out)	MemPtr	Application buffer address.
Parameters (in - out)	-	-

Table 60 Specification for `Uart_Read` API (continued)

Return	Uart_ReturnType	UART_E_OK - Receive operation initiated successfully. UART_E_NOT_OK - Receive operation couldn't be initiated due to development errors. UART_E_BUSY - UART channel is busy in receive operation. If DET and Safety is disabled API will return UART_E_OK and UART_E_BUSY.
Description	API to read data from an UART channel, with specified size and the memory location. Timeout for this API need to be handled by application. Driver does not handle any timeout for this API.	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_SIZE, UART_E_STATE_BUSY, UART_E_INVALID_CHANNEL, UART_E_PARAM_POINTER	
Configuration dependencies	-	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.4 Uart_Write

Table 61 Specification for `Uart_Write` API

Syntax	`Uart_ReturnType  Uart_Write`
	`(`
	`  const Uart_ChannelIdType Channel,`
	`  const Uart_MemType * const MemPtr,`
	`  const Uart_SizeType Size`
	`)`	

Service ID	0xDA	
Sync/Async	Asynchronous	
ASIL Level	B	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel MemPtr Size	UART channel id. Application memory address from where data to be transmit. Number of data bytes to be transmitted. Note: If channel frame length configured with greater than 8 bits then number of bytes should be multiple of 2.

Table 61 Specification for `Uart_Write` API (continued)

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_ReturnType	UART_E_OK - Transmit operation initiated successfully. UART_E_NOT_OK - Transmit operation couldn't be initiated due to development errors. UART_E_BUSY - UART channel is busy in transmit operation. If DET and Safety is disabled API will return UART_E_OK and UART_E_BUSY.
Description	API to write data to an Uart channel, with specified size and the memory location. API returning success indicates that data accepted for transmission, API will update the data to be transmitted in FIFO and enable interrupts for successive writes to FIFO.	
Source	IFX	
Error handling	UART_E_STATE_BUSY, UART_E_INVALID_SIZE, UART_E_UNINIT, UART_E_INVALID_CHANNEL, UART_E_PARAM_POINTER, UART_E_TXFIFO_FILL_ERR	
Configuration dependencies	-	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_TXDATA(w), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.5 Uart_AbortRead

Table 62 Specification for `Uart_AbortRead` API

Syntax	`Uart_SizeType  Uart_AbortRead`
	`(`
	`  const Uart_ChannelIdType Channel`
	`)`	

Service ID	0xDC	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.

Table 62 Specification for `Uart_AbortRead` API (continued)

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_SizeType	Number of bytes successfully received and stored to the application memory location before the read operation was aborted.
Description	API to abort read operation on given channel. *Note: API (optional API) is available only when the parameter UartAbortReadApi is enabled. Abort read notification will be called at the end of successful abort.*	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL	
Configuration dependencies	UartAbortReadApi	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.6 Uart_AbortWrite

Table 63 Specification for `Uart_AbortWrite` API

Syntax	`Uart_SizeType  Uart_AbortWrite`
	`(`
	`  const Uart_ChannelIdType Channel`
	`)`	

Service ID	0xDB	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.
Parameters (out)	-	-
Parameters (in - out)	-	-

Table 63 Specification for `Uart_AbortWrite` API (continued)

Return	Uart_SizeType	Number of bytes that have been successfully transmitted before the write operation was aborted.
Description	API to abort data transmission on given channel. *Note: API(optional API) is available only when the parameter UartAbortWriteApi is enabled Notification will be called at the end of successful abort.*	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL	
Configuration dependencies	UartAbortWriteApi	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.7 Uart_GetStatus

Table 64 Specification for `Uart_GetStatus` API

Syntax	`Uart_StatusType  Uart_GetStatus`
	`(`
	`  const Uart_ChannelIdType Channel`
	`)`	

Service ID	0xDD	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_StatusType	UART_IDLE: Idle state (no transmit or receive operation in progress). UART_BUSY_TRANSMIT: UART channel busy in transmit operation. UART_BUSY_RECEIVE: UART channel busy in receive operation.

Table 64 Specification for `Uart_GetStatus` API (continued)

		UART_BUSY_TRANSMIT_RECEIVE: UART channel busy in transmit and receive operation.
Description	API to read an UART channels status.	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL	
Configuration dependencies	-	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.8 Uart_DeInit

Table 65 Specification for `Uart_DeInit` API

Syntax	`void Uart_DeInit ( void )`	
Service ID	0xDE	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	UART driver de-initialization function. *Note: API (optional API) is available only if parameter UartDeInitApi is enabled.* *Upper layer need to ensure that all configured channels are in IDLE sate and no communication on the channel before driver de-initializing Uart driver.*	
Source	IFX	
Error handling	UART_E_UNINIT	
Configuration dependencies	UartDeInitApi	

Table 65 Specification for `Uart_DeInit` API (continued)

User hints	-
SFR accessed	ASCLIN_CLC(rw), ASCLIN_CSR(rw), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_FRAMECON(w), ASCLIN_KRST0(rw), ASCLIN_KRST1(rw), ASCLIN_KRSTCLR(rw), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.9 Uart_GetVersionInfo

Table 66 Specification for `Uart_GetVersionInfo` API

Syntax	`void Uart_GetVersionInfo ( Std_VersionInfoType * const VersionInfoPtr )`	
Service ID	0xDF	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	VersionInfoPtr	Address on which version information to be stored.
Parameters (in - out)	-	-
Return	void	-
Description	API to get the version information of UART driver. *Note: API (optional API) is available only if parameter UartVersionInfoApi is enabled.*	
Source	IFX	
Error handling	UART_E_PARAM_POINTER	
Configuration dependencies	UartVersionInfoApi	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.4 Notifications and Callbacks

The UART driver does not provide any notification or callbacks.

1.3.5 Scheduled functions

This section lists all the scheduled functions of the UART driver.

1.3.5.1 Uart_MainFunction_Read

Table 67 Specification for `Uart_MainFunction_Read` API

Syntax	`void Uart_MainFunction_Read ( void )`	
Service ID	0xE0	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Schedule function to handle receives operation in polling mode. *Note: Function will be available if any of channels receive operation configured in polling mode.*	
Source	IFX	
Error handling	UART_E_FRAME_ERROR, UART_E_RXFIFO_OVERFLOW, UART_E_PARITY_ERROR	
Configuration dependencies	UartRxChannelMode	
User hints	-	
SFR accessed	ASCLIN_DATCON(r), ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXDATA(r), ASCLIN_RXFIFOCON(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.5.2 Uart_MainFunction_Write

Table 68 Specification for `Uart_MainFunction_Write` API

Syntax	`void Uart_MainFunction_Write ( void )`	
Service ID	0xE1	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Schedule function to handle transmits operation in polling mode. *Note: Function will be available if any of channels transmit operation configured in polling mode.*	
Source	IFX	
Error handling	UART_E_TXFIFO_FILL_ERR	
Configuration dependencies	UartTxChannelMode	
User hints	-	
SFR accessed	ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_TXDATA(w), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.6 Interrupt service routines

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

1.3.6.1 Uart_IsrError

Table 69 Specification for `Uart_IsrError` API

Syntax	`void Uart_IsrError ( const uint8 HwUnit )`	
Service ID	0xE2	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant (Not for the same HW Unit).	
Parameters (in)	HwUnit	ASCLIN channel number.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	IsrErrorHandler is invoked when any error during UART reception is triggered or when transmit complete event is triggered. Note that these events shares the same interrupt signal line due to which events cannot be handled separately in driver. *Note: UartReceiveNotifPtr triggers when receive error occurred and UartTransmitNotifPtr triggers after successful transmission of data.*	
Source	IFX	
Error handling	UART_E_RXFIFO_OVERFLOW, UART_E_PARITY_ERROR, UART_E_FRAME_ERROR, UART_E_INVALID_HW_UNIT, UART_E_SPURIOUS_INTERRUPT	
Configuration dependencies	UartRxChannelMode,UartTxChannelMode	
User hints	-	
SFR accessed	ASCLIN_DATCON(r), ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(rw), ASCLIN_RXDATA(r), ASCLIN_RXFIFOCON(rw), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.6.2 Uart_IsrReceive

Table 70 Specification for `Uart_IsrReceive` API

Syntax	`void Uart_IsrReceive (`

Table 70 Specification for `Uart_IsrReceive` API (continued)

	`const uint8 HwUnit` `)`	
Service ID	0xE3	
Sync/Async	Synchronous	
ASIL Level	B	
Re-entrancy	Reentrant (Not for the same HW Unit).	
Parameters (in)	HwUnit	ASCLIN channel number.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interrupt is triggered when RXFIFO filled up to configured level with received data. This will be notified with UartReceiveNotifPtr. *Note: RX FIFO level configured by the Uart driver and it varies from 1 to 16 bytes.*	
Source	IFX	
Error handling	UART_E_INVALID_HW_UNIT, UART_E_SPURIOUS_INTERRUPT	
Configuration dependencies	UartRxChannelMode	
User hints	-	
SFR accessed	ASCLIN_DATCON(r), ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(rw), ASCLIN_RXDATA(r), ASCLIN_RXFIFOCON(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.6.3 Uart_IsrTransmit

Table 71 Specification for `Uart_IsrTransmit` API

Syntax	`void Uart_IsrTransmit`
	`(`
	`const uint8 HwUnit`
	`)`

Service ID	0xE4
Sync/Async	Synchronous
ASIL Level	B

Table 71 Specification for `Uart_IsrTransmit` API (continued)

Re-entrancy	Reentrant (Not for the same HW Unit).	
Parameters (in)	HwUnit	ASCLIN channel number.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Uart driver sets TXFIFO level with No of bytes to be transmitted and it varies from 1 to 16 bytes. Transmit interrupt is generated when the TXFIFO becomes empty. *Note: UartTransmitNotifPtr will trigger after successful transmission of data.*	
Source	IFX	
Error handling	UART_E_SPURIOUS_INTERRUPT, UART_E_INVALID_HW_UNIT, UART_E_TXFIFO_FILL_ERR	
Configuration dependencies	UartTxChannelMode	
User hints	-	
SFR accessed	ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(rw), ASCLIN_TXDATA(w), ASCLIN_TXFIFOCON(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.7 Callout

The Uart driver does not provide any callout.

1.3.8 Errors Handling

This section describes the various error types reported by the UART driver.

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
UART_E_FRAME_ERROR: This runtime error is reported when the frame check fail.	IFX	0x02	RUNTIME	0x02	RUNTIME
UART_E_PARITY_ERROR: This runtime error is reported when the parity check fail.	IFX	0x01	RUNTIME	0x01	RUNTIME
UART_E_RXFIFO_OVERFLOW: This runtime error is reported	IFX	0x03	RUNTIME	0x03	RUNTIME

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
when the RXFIFO overflow error set.					
UART_E_UNINIT: API service used without UART driver initialization.	IFX	0x00	DET_SAFETY	0x00	DET_SAFETY
UART_E_INVALID_CHANNEL: API service used with an invalid channel identifier.	IFX	0x01	DET_SAFETY	0x01	DET_SAFETY
UART_E_PARAM_POINTER: API service used with NULL pointer.	IFX	0x02	DET_SAFETY	0x02	DET_SAFETY
UART_E_STATE_BUSY: API service called when channel is in busy state.	IFX	0x03	DET_SAFETY	0x03	DET_SAFETY
UART_E_INIT_FAILED: UART driver initialization fails.	IFX	0x04	DET_SAFETY	0x04	DET_SAFETY
UART_E_INVALID_SIZE: API Service called with invalid data length parameter.	IFX	0x05	DET_SAFETY	0x05	DET_SAFETY
UART_E_ALREADY_INITIALIZED: UART driver is already initialized.	IFX	0x06	DET_SAFETY	0x06	DET_SAFETY
UART_E_SPURIOUS_INTERRUPT: Spurious interrupt detected.	IFX	0x07	SAFETY	0x07	SAFETY
UART_E_INVALID_HW_UNIT: IRQ handler called with invalid hardware unit identifier.	IFX	0x08	SAFETY	0x08	SAFETY
UART_E_TXFIFO_FILL_ERR: TXFIFO fill error, Fill level not matched with number of bytes filled in TXFIFO.	IFX	0x09	SAFETY	0x09	SAFETY

1.3.9 Deviations and limitations

This section describes the deviations and limitations of the UART driver.

1.3.9.1 Deviations

This section describes the deviations of the UART driver.

1.3.9.1.1 Software specification deviations

The UART driver does not have any deviations.

1.3.9.1.2 AMDC Violations

The UART driver does not have any AMDC violation.

1.3.9.1.3 VSMD Violations

The UART driver does not have any VSMD violation.

1.3.9.2 Limitations

This section describes the limitations of the UART deriver.

Table 72 Known limitations

Reference	Limitation
Uart transmit complete notification.	It is observed that the UART channel transmit complete notification is triggered before transmission of the last frame from the ASCLIN kernel. If application has any use case of calling `Uart_DeInit` API immediately after receiving the transmit completion notification, it is being observed that the receiver is unable to receive the last frame. This behavior is observed in both interrupt and polling mode. Workaround: Provide one frame delay (based on the baudrate used) before calling the `Uart_DeInit` API. Example: In 9600 kbps baud rate configuration, if the application software is waiting for transmit complete notification to invoke `Uart_DeInit` API, there should be a delay of 1.04167 milliseconds between the application received transmit complete notification and `Uart_DeInit` API invocation.

Revision history

Table 73 Major changes since last version

Date	Version	Description
2020-11-25	2.0	Document is released.
2020-11-19	1.1	• Port support section updated. • Default value updated for UartRunTimeErrorDetect. • Range updated for UartParitybit parameter.
2020-08-13	1.0	Document is released.
2020-08-07	0.1	• Initial version. • UART chapter moved from MCISAR_TC3xx_UM_CD to thisdocument. • Updated description for Uart_IsrError interrupt handler. • Figure 5 updated in MCU support.

Trademarks

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

Edition 2020-11-25

Published by

Infineon Technologies AG

81726 Munich, Germany

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

Email: erratum@infineon.com

Document reference
IFX-ocr1484806431059

IMPORTANT NOTICE

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

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.