Integration Manual

for S32K1_M24X UART Driver

Document Number: IM2UARTASRR21-11 Rev0000R2.0.0 Rev. 1.0

Chapter 1

Revision History

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

Chapter 2

Introduction

This integration manual describes the integration requirements for Uart Driver for S32K1_S32M24X microcontrollers.

2.1 Supported Derivatives

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

- s32k116_qfn32
- s32k116_lqfp48
- s32k118_lqfp48
- s32k118_lqfp64
- s32k142_lqfp48
- s32k142_lqfp64
- s32k142_lqfp100
- s32k142w_lqfp48
- s32k142w_lqfp64
- s32k144_lqfp48
- s32k144_lqfp64 / MWCT1014S_lqfp64
- s32k144_lqfp100 / MWCT1014S_lqfp100

- s32k144_mapbga100
- s32k144w_lqfp48
- s32k144w_lqfp64
- s32k146_lqfp64
- s32k146_lqfp100 / MWCT1015S_lqfp100
- s32k146_mapbga100 / MWCT1015S_mapbga100
- s32k146_lqfp144
- s32k148_lqfp100
- s32k148_mapbga100 / MWCT1016S_mapbga100
- s32k148_lqfp144
- s32k148_lqfp176
- s32m241_lqfp64
- s32m242_lqfp64
- s32m243_lqfp64
- s32m244_lqfp64

All of the above microcontroller devices are collectively named as S32K1_S32M24X. Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power

2.2 Overview

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

AUTOSAR:

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

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

- **Boldface** style: Used for important terms, notes and warnings.
- *Italic* style: Used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

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

2.5 Reference List

#	Title	Version
1	Reference Manual	S32K1xx Series Reference Manual, Rev. 14, 09/2021
		S32M24x Reference Manual, Rev. 2 Draft A, 05/2023
2	Errata	S32K116_0N96V Rev. 22/OCT/2021
		S32K118_0N97V Rev. 22/OCT/2021
		S32K142_0N33V Rev. 22/OCT/2021
		S32K144_0N57U Rev. 22/OCT/2021
		S32K144W_0P64A Rev. 22/OCT/2021
		S32K146_0N73V Rev. 22/OCT/2021
		S32K148_0N20V Rev. 22/OCT/2021
		S32M244_P64A+P73G, Rev. 0
		S32M242_N33V+P73G, Rev. 0, 6/2023

#	Title	Version
3	Datasheet	S32K1xx Data Sheet, Rev. 14, 08/2021
		S32M2xx Data Sheet, Rev. 3 DraftA, 05/2023

Chapter 3

Building the driver

- Build Options
- Files required for compilation
- Setting up the plugins

This section describes the source files and various compilers, linker options used for building the driver.
It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

- GCC Compiler/Assembler/Linker Options
- IAR Compiler/Assembler/Linker Options
- GHS Compiler/Assembler/Linker Options

The RTD driver files are compiled using:

- NXP GCC 10.2.0 20200723 (Build 1728 Revision g5963bc8)
- IAR ANSI C/C++ Compiler V8.40.3.228/W32 for ARM Functional Safety
- Green Hills Multi 7.1.6d / Compiler 2020.1.4

The compiler, assembler, and linker flags used for building the driver are explained below.
The TS_T40D2M20I0R0 part of the plugin name is composed as follows:

- T = Target_Id (e.g. T40 identifies Cortex-M architecture)
- D = Derivative_Id (e.g. D2 identifies S32K1 platform)
- M = SW_Version_Major and SW_Version_Minor
- I = SW_Version_Patch
- R = Reserved

3.1.1 GCC Compiler/Assembler/Linker Options

3.1.1.1 GCC Compiler Options

Compiler Option	Description
-mcpu=cortex-m4	Targeted ARM processor for which GCC should tune the performance of the code (for S32K14x or S32M24x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-mthumb	Generates code that executes in Thumb state
-mlittle-endian	Generate code for a processor running in little-endian mode
-mfpu=fpv4-sp-d16	Specifies the floating-point hardware available on the target (for S32K14x or S32M24x devices)
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions (for S32K14x or S32M24x devices)
-mfpu=auto	Specifies the floating-point hardware available on the target (for S32K11x devices)
-mfloat-abi=soft	Specifies the floating-point ABI to use. Specifying "soft" causes GCC to generate output containing library calls for floating-point operations (for S32K11x devices)
-std=c99	Specifies the ISO C99 base standard
-Os	Optimize for size. Enables all -O2 optimizations except those that often increase code size
-ggdb3	Produce debugging information for use by GDB using the most expressive format available, including GDB extensions if at all possible. Level 3 includes extra information, such as all the macro definitions present in the program
-Wall	Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros
-Wextra	This enables some extra warning flags that are not enabled by -Wall
-pedantic	Issue all the warnings demanded by strict ISO C. Reject all programs that use forbidden extensions. Follows the version of the ISO C standard specified by the aforementioend -std option
-Wstrict-prototypes	Warn if a function is declared or defined without specifying the argument types
-Wundef	Warn if an undefined identifier is evaluated in an #if directive. Such identifiers are replaced with zero
-Wunused	Warn whenever a function, variable, label, value, macro is unused
-Werror=implicit-function-declaration	Make the specified warning into an error. This option throws an error when a function is used before being declared
-Wsign-compare	Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.
-Wdouble-promotion	Give a warning when a value of type float is implicitly promoted to double
-fno-short-enums	Specifies that the size of an enumeration type is at least 32 bits regardless of the size of the enumerator values.

Compiler Option	Description
-funsigned-char	Let the type char be unsigned by default, when the declaration does not use either signed or unsigned
-funsigned-bitfields	Let a bit-field be unsigned by default, when the declaration does not use either signed or unsigned
-fomit-frame-pointer	Omit the frame pointer in functions that don't need one. This avoids the instructions to save, set up and restore the frame pointer; on many targets it also makes an extra register available.
-fno-common	Makes the compiler place uninitialized global variables in the BSS section of the object file. This inhibits the merging of tentative definitions by the linker so you get a multiple-definition error if the same variable is accidentally defined in more than one compilation unit
-fstack-usage	This option is only used to build test for generation Ram/↩ Stack size report. Makes the compiler output stack usage information for the program, on a per-function basis
-fdump-ipa-all	This option is only used to build test for generation Ram/↩ Stack size report. Enables all inter-procedural analysis dumps
-c	Stop after assembly and produce an object file for each source file
-DS32K1XX	Predefine S32K1XX as a macro, with definition 1
-DS32K148	Predefine S32K148 as a macro, with definition 1. S32↩ K148 can be replaced according to derivatives name S32K116,S32K118,S32K142,S32K142W,S32K144,S32↩ K144W,S32K146,S32K148,S32M244,S32M242.
-DGCC	Predefine GCC as a macro, with definition 1
-DUSE_SW_VECTOR_MODE	Predefine USE_SW_VECTOR_MODE as a macro, with definition 1. By default, the drivers are compiled to handle interrupts in Software Vector Mode
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initialization in source file system.c under the Platform driver (for S32K14x or S32↩ M24x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver (for S32K14x or S32M24x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPO↩ RT as a macro, with definition 1. Allows drivers to be configured in user mode.
–sysroot=	Specifies the path to the sysroot, for Cortex-M7 it is /arm-none-eabi/newlib
-specs=nano.specs	Use Newlib nano specs
-specs=nosys.specs	Do not use printf/scanf

3.1.1.2 GCC Assembler Options

Assembler Option	Description
-Xassembler-with-cpp	Specifies the language for the following input files (rather than letting the compiler choose a default based on the file name suffix)
-mcpu=cortex-m4	Targeted ARM processor for which GCC should tune the performance of the code (for S32K14x or S32M24x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-mfpu=fpv4-sp-d16	Specifies the floating-point hardware available on the target (for S32K14x devices)
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions (for S32K14x devices)
-mfpu=auto	Specifies the floating-point hardware available on the target (for S32K11x devices)
-mfloat-abi=soft	Specifies the floating-point ABI to use. Specifying "soft" causes GCC to generate output containing library calls for floating-point operations (for S32K11x devices)
-mthumb	Generates code that executes in Thumb state
-c	Stop after assembly and produce an object file for each source file

3.1.1.3 GCC Linker Options

Linker Option	Description
-Wl,-Map,filename	Produces a map file
-T linkerfile	Use linkerfile as the linker script. This script replaces the default linker script (rather than adding to it)
–entry=Reset_Handler	Specifies that the program entry point is Reset_Handler
-nostartfiles	Do not use the standard system startup files when linking
-mcpu=cortex-m4	Targeted ARM processor for which GCC should tune the performance of the code (for S32K14x or S32M24x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-mthumb	Generates code that executes in Thumb state
-mfpu=fpv4-sp-d16	Specifies the floating-point hardware available on the target (for S32K14x or S32M24x devices)
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions (for S32K14x or S32M24x devices)
-mfpu=auto	Specifies the floating-point hardware available on the target (for S32K11x devices)
-mfloat-abi=soft	Specifies the floating-point ABI to use. Specifying "soft" causes GCC to generate output containing library calls for floating-point operations (for S32K11x devices)
-mlittle-endian	Generate code for a processor running in little-endian mode
-ggdb3	Produce debugging information for use by GDB using the most expressive format available, including GDB extensions if at all possible. Level 3 includes extra information, such as all the macro definitions present in the program
-lc	Link with the C library
-lm	Link with the Math library
-lgcc	Link with the GCC library
-n	Turn off page alignment of sections, and disable linking against shared libraries
–sysroot=	Specifies the path to the sysroot, for Cortex-M7 it is /arm-none-eabi/newlib

Linker Option	Description
-specs=nano.specs	Use Newlib nano specs
-specs=nosys.specs	Do not use printf/scanf

3.1.2 IAR Compiler/Assembler/Linker Options

3.1.2.1 IAR Compiler Options

Compiler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x or S32M24x devices)
–cpu=Cortex-M0+	Targeted ARM processor for which IAR should tune the performance of the code (for S32K11x devices)
–cpu_mode=thumb	Generates code that executes in Thumb state
–endian=little	Generate code for a processor running in little-endian mode
–fpu=FPv4-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant. (for S32K14x or S32M24x devices)
–fpu=none	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). No FPU. (for S32K11x devices)
-e	Enables all IAR C language extensions
-Ohz	Optimize for size. The compiler will emit AEABI attributes indicating the requested optimization goal. This information can be used by the linker to select smaller or faster variants of DLIB library functions
–debug	Makes the compiler include debugging information in the object modules. Including debug information will make the object files larger
–no_clustering	Disables static clustering optimizations. Static and global variables defined within the same module will not be arranged so that variables that are accessed in the same function are close to each other
–no_mem_idioms	Makes the compiler not optimize certain memory access patterns
–no_explicit_zero_opt	Do not treat explicit initializations to zero of static variables as zero initializations
–require_prototypes	Force the compiler to verify that all functions have proper prototypes. Generates an error otherwise
–no_wrap_diagnostics	Does not wrap long lines in diagnostic messages
–diag_suppress=Pa050	Suppresses diagnostic message Pa050
-DS32K1XX	Predefine S32K1XX as a macro, with definition 1
-DS32K148	Predefine S32K148 as a macro, with definition 1. S32↩ K148 can be replaced according to derivatives name S32K116,S32K118,S32K142,S32K142W,S32K144,S32↩ K144W,S32K146,S32K148,S32M244,S32M242.
-DIAR	Predefine IAR as a macro, with definition 1

Compiler Option	Description
-DUSE_SW_VECTOR_MODE	Predefine USE_SW_VECTOR_MODE as a macro, with definition 1. By default, the drivers are compiled to handle interrupts in Software Vector Mode.
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initialization in source file system.c under the Platform driver (for S32K14x or S32↩ M24x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver (for S32K14x or S32M24x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPO↩ RT as a macro, with definition 1. Allows drivers to be configured in user mode.

3.1.2.2 IAR Assembler Options

Assembler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x or S32M24x devices)
–cpu=Cortex-M0+	Targeted ARM processor for which IAR should tune the performance of the code (for S32K11x devices)
–fpu=FPv4-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant. (for S32K14x devices)
–fpu=none	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). No FPU. (for S32K11x devices)
–cpu_mode thumb	Selects the thumb mode for the assembler directive CODE
-g	Disables the automatic search for system include files
-r	Generates debug information

3.1.2.3 IAR Linker Options

Linker Option	Description
–map filename	Produces a map file
–config linkerfile	Use linkerfile as the linker script. This script replaces the default linker script (rather than adding to it)
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x or S32M24x devices)
–cpu=Cortex-M0+	Targeted ARM processor for which IAR should tune the performance of the code (for S32K11x devices)
–fpu=FPv4-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant. (for S32K14x or S32M24x devices)
–fpu=none	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). No FPU. (for S32K11x devices)

Linker Option	Description
–entry _start	Treats _start as a root symbol and start label
–enable_stack_usage	Enables stack usage analysis. If a linker map file is produced, a stack usage chapter is included in the map file
–skip_dynamic_initialization	Dynamic initialization (typically initialization of C++ objects with static storage duration) will not be performed automatically during application startup
–no_wrap_diagnostics	Does not wrap long lines in diagnostic messages

3.1.3 GHS Compiler/Assembler/Linker Options

3.1.3.1 GHS Compiler Options

Compiler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x or S32M24x devices)
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+ (for S32K11x devices)
-thumb	Selects generating code that executes in Thumb state
-fpu=vfpv4_d16	Specifies hardware floating-point using the v4 version of the VFP instruction set, with 16 double-precision floating-point registers (for S32K14x or S32M24x devices)
-fsingle	Use hardware single-precision, software double-precision FP instructions (for S32K14x or S32M24x devices)
-fsoft	Specifies software floating-point (SFP) mode. This setting causes your target to use integer registers to hold floating-point data and use library subroutine calls to emulate floating-point operations (for S32K11x devices)
-C99	Use (strict ISO) C99 standard (without extensions)
–ghstd=last	Use the most recent version of Green Hills Standard mode (which enables warnings and errors that enforce a stricter coding standard than regular C and C++)
-Osize	Optimize for size
–gnu_asm	Enables GNU extended asm syntax support
-dual_debug	Generate DWARF 2.0 debug information
-G	Generate debug information
-keeptempfiles	Prevents the deletion of temporary files after they are used. If an assembly language file is created by the compiler, this option will place it in the current directory instead of the temporary directory
-Wimplicit-int	Produce warnings if functions are assumed to return int
-Wshadow	Produce warnings if variables are shadowed
-Wtrigraphs	Produce warnings if trigraphs are detected
-Wundef	Produce a warning if undefined identifiers are used in #if preprocessor statements
–unsigned_chars	Let the type char be unsigned, like unsigned char

Compiler Option	Description
–unsigned_fields	Bitfields declared with an integer type are unsigned
–no_commons	Allocates uninitialized global variables to a section and initializes them to zero at program startup
–no_exceptions	Disables C++ support for exception handling
–no_slash_comment	C++ style // comments are not accepted and generate errors
–prototype_errors	Controls the treatment of functions referenced or called when no prototype has been provided
–incorrect_pragma_warnings	Controls the treatment of valid #pragma directives that use the wrong syntax
-c	Stop after assembly and produce an object file for each source file
-DS32K1XX	Predefine S32K1XX as a macro, with definition 1
-DS32K148	Predefine S32K148 as a macro, with definition 1. S32↩ K148 can be replaced according to derivatives name S32K116,S32K118,S32K142,S32K142W,S32K144,S32↩ K144W,S32K146,S32K148,S32M244,S32M242.
-DGHS	Predefine GHS as a macro, with definition 1
-DUSE_SW_VECTOR_MODE	Predefine USE_SW_VECTOR_MODE as a macro, with definition 1. By default, the drivers are compiled to handle interrupts in Software Vector Mode
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initialization in source file system.c under the Platform driver (for S32K14x or S32↩ M24x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver (for S32K14x or S32M24x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPO↩ RT as a macro, with definition 1. Allows drivers to be configured in user mode

3.1.3.2 GHS Assembler Options

Assembler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x or S32M24x devices)
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+ (for S32K11x devices)
-fpu=vfpv4_d16	Specifies hardware floating-point using the v4 version of the VFP instruction set, with 16 double-precision floating-point registers (for S32K14x devices)
-fsingle	Use hardware single-precision, software double-precision FP instructions (for S32↩ K14x devices)
-fsoft	Specifies software floating-point (SFP) mode. This setting causes your target to use integer registers to hold floating-point data and use library subroutine calls to emulate floating-point operations (for S32K11x devices)
-preprocess_assembly_files	Controls whether assembly files with standard extensions such as .s and .asm are preprocessed
-list	Creates a listing by using the name and directory of the object file with the .lst extension

Assembler Option	Description
-c	Stop after assembly and produce an object file for each source file

3.1.3.3 GHS Linker Options

Linker Option	Description
-e Reset_Handler	Make the symbol Reset_Handler be treated as a root symbol and the start label of the application
-T linker_script_file.ld	Use linker_script_file.ld as the linker script. This script replaces the default linker script (rather than adding to it)
-map	Produce a map file
-keepmap	Controls the retention of the map file in the event of a link error
-Mn	Generates a listing of symbols sorted alphabetically/numerically by address
-delete	Instructs the linker to remove functions that are not referenced in the final executable. The linker iterates to find functions that do not have relocations pointing to them and eliminates them
-ignore_debug_references	Ignores relocations from DWARF debug sections when using -delete. DWARF debug information will contain references to deleted functions that may break some third-party debuggers
-Llibrary_path	Points to library_path (the libraries location) for thumb2 to be used for linking
-larch	Link architecture specific library
-lstartup	Link run-time environment startup routines. The source code for the modules in this library is provided in the src/libstartup directory
-lind_sd	Link language-independent library, containing support routines for features such as software floating point, run-time error checking, C99 complex numbers, and some general purpose routines of the ANSI C library (for S32K14x or S32M24x devices)
-lind_sf	Link language-independent library, containing support routines for features such as software floating point, run-time error checking, C99 complex numbers, and some general purpose routines of the ANSI C library (for S32K11x devices)
-v	Prints verbose information about the activities of the linker, including the libraries it searches to resolve undefined symbols
-keep=C40_Ip_AccessCode	Avoid linker remove function C40_Ip_AccessCode from Fls module because it is not referenced explicitly
-nostartfiles	Controls the start files to be linked into the executable

3.2 Files required for compilation

This section describes the include files required to compile, assemble and link the CDD Uart Driver for S32K1_S32M24X microcontrollers.

To avoid integration of incompatible files, all the include files from other modules shall have the same AR_MAJOR_VERSION and AR_MINOR_VERSION, i.e. only files with the same AUTOSAR major and minor versions can be compiled.

3.2.1 Uart Driver Files:

- Uart_TS_T40D2M20I0R0\src\CDD_Uart.c
- Uart_TS_T40D2M20I0R0\src\Uart_Ipw.c
- Uart_TS_T40D2M20I0R0\src\Lpuart_Uart_Ip.c
- Uart_TS_T40D2M20I0R0\src\Lpuart_Uart_Ip_Irq.c
- Uart_TS_T40D2M20I0R0\src\Flexio_Uart_Ip.c
- Uart_TS_T40D2M20I0R0\src\Flexio_Uart_Ip_Irq.c
- Uart_TS_T40D2M20I0R0\inc\CDD_Uart.h
- Uart_TS_T40D2M20I0R0\inc\Uart_Types.h
- Uart_TS_T40D2M20I0R0\inc\Uart_Ipw.h
- Uart_TS_T40D2M20I0R0\inc\Uart_Ipw_Types.h
- Uart_TS_T40D2M20I0R0\inc\Lpuart_Uart_Ip.h
- Uart_TS_T40D2M20I0R0\inc\Lpuart_Uart_Ip_HwAccess.h
- Uart_TS_T40D2M20I0R0\inc\Lpuart_Uart_Ip_Irq.h
- Uart_TS_T40D2M20I0R0\inc\Lpuart_Uart_Ip_Types.h
- Uart_TS_T40D2M20I0R0\inc\Flexio_Uart_Ip.h
- Uart_TS_T40D2M20I0R0\inc\Flexio_Uart_Ip_HwAccess.h
- Uart_TS_T40D2M20I0R0\inc\Flexio_Uart_Ip_Irq.h
- Uart_TS_T40D2M20I0R0\inc\Flexio_Uart_Ip_Types.h

3.2.2 Uart Driver Generated Files (must be generated by the user using a configuration tool):

- CDD_Uart_[VariantName]_PBcfg.c(For PB Variant)
- Uart_Ipw_[VariantName]_PBcfg.c
- Lpuart_Uart_Ip_[VariantName]_PBcfg.c
- Flexio_Uart_Ip_[VariantName]_PBcfg.c
- Uart_[VariantName]_PBcfg.h
- CDD_Uart_Cfg.h
- CDD_Uart_Defines.h
- Uart_Ipw_[VariantName]_PBcfg.h
- Uart_Ipw_Cfg.h
- Lpuart_Uart_Ip_[VariantName]_PBcfg.h

- Lpuart_Uart_Ip_Cfg.h
- Lpuart_Uart_Ip_Defines.h
- Flexio_Uart_Ip_[VariantName]_PBcfg.h
- Flexio_Uart_Ip_Cfg.h
- Flexio_Uart_Ip_CfgDefines.h

Note

As a deviation from standard:

- CDD_Uart_[VariantName]_PBcfg.c, Uart_Ipw_[VariantName]_PBcfg.c, Lpuart_Uart_Ip_[Variant↩ Name]_PBcfg.c, Flexio_Uart_Ip_[VariantName]_PBcfg.c - This file will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB).

3.2.3 Base Files:

- Base_TS_T40D2M20I0R0\include\Compiler.h
- Base_TS_T40D2M20I0R0\include\Compiler_Cfg.h
- Base_TS_T40D2M20I0R0\include\Mcal.h
- Base_TS_T40D2M20I0R0\include\Uart_MemMap.h
- Base_TS_T40D2M20I0R0\include\Platform_Types.h
- Base_TS_T40D2M20I0R0\include\Reg_eSys.h
- Base_TS_T40D2M20I0R0\include\RegLockMacros.h
- Base_TS_T40D2M20I0R0\include\Soc_Ips.h
- Base_TS_T40D2M20I0R0\include\StandardTypes.h
- Base_TS_T40D2M20I0R0\include\OsIf.h
- Base_TS_T40D2M20I0R0\include\Devassert.h
- Base_TS_T40D2M20I0R0\header\S32M24x_LPUART.h
- Base_TS_T40D2M20I0R0\header\S32M24x_FLEXIO.h

3.2.4 DET Files:

- Det_TS_T40D2M20I0R0\include\Det.h
- Det_TS_T40D2M20I0R0\src\Det.c

3.2.5 MCL Files(When DMA or Flexio is used):

- Mcl_TS_T40D2M20I0R0\include\CDD_Mcl.h
- Mcl_TS_T40D2M20I0R0\src\CDD_Mcl.c
- Mcl_TS_T40D2M20I0R0\include\CDD_Mcl_Cfg.h
- Mcl_TS_T40D2M20I0R0\src\CDD_Mcl_Cfg.c

Note: These are the files used for Flexio. In the application, all files in MCL Files must be added.

1. Flexio MCL Driver Filess:
   - Mcl_TS_T40D2M20I0R0\include\Mcl.h
   - Mcl_TS_T40D2M20I0R0\include\Flexio_Mcl_Ip_HwAccess.h
   - Mcl_TS_T40D2M20I0R0\include\Flexio_Mcl_Ip_Types.h
   - Mcl_TS_T40D2M20I0R0\include\Flexio_Mcl_Ip.h
   - Mcl_TS_T40D2M20I0R0\src\Mcl.c
   - Mcl_TS_T40D2M20I0R0\src\Flexio_Mcl_Ip_HwAccess.c
   - Mcl_TS_T40D2M20I0R0\src\Flexio_Mcl_Ip_Types.c
   - Mcl_TS_T40D2M20I0R0\src\Flexio_Mcl_Ip.c
   - Mcl_TS_T40D2M20I0R0\src\Flexio_Mcl_Ip_Irq.c
2. Flexio MCL Driver Generated Files (must be generated by the user using a configuration tool):
   - Mcl_TS_T40D2M20I0R0\generate_PC\include\Flexio_Mcl_Ip_Cfg.h
   - Mcl_TS_T40D2M20I0R0\generate_PC\include\Flexio_Mcl_Ip_CfgDefines.h
   - Mcl_TS_T40D2M20I0R0\generate_PC\include\Flexio_Mcl_Ip_Definitions.h
   - Mcl_TS_T40D2M20I0R0\generate_PB\include\Flexio_Mcl_Ip_PBcfg.h
   - Mcl_TS_T40D2M20I0R0\generate_PB\include\Flexio_Mcl_Ip_PBcfg.c

3.2.6 RTE Files:

- Rte_TS_T40D2M20I0R0\include\SchM_Uart.h
- Rte_TS_T40D2M20I0R0\src\SchM_Uart.c

3.3 Setting up the plugins

The Uart Driver was designed to be configured by using the EB Tresos Studio (version 27.1.0 b200625-0900 or later)

3.3.1 Location of various files inside the Uart module folder:

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
  - Uart_TS_T40D2M20I0R0\config\Uart.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - Uart_TS_T40D2M20I0R0\autosar\Uart_<subderivative_name>.epd
- Code Generation Templates for variant aware parameters:
  - Uart_TS_T40D2M20I0R0\generate_PB\include\CDD_Uart_PBcfg.h
  - Uart_TS_T40D2M20I0R0\generate_PB\include\Lpuart_Uart_Ip_PBcfg.h
  - Uart_TS_T40D2M20I0R0\generate_PB\include\Flexio_Uart_Ip_PBcfg.h
  - Uart_TS_T40D2M20I0R0\generate_PB\include\Uart_Ipw_PBcfg.h
  - Uart_TS_T40D2M20I0R0\generate_PB\src\Uart_Ipw_PBcfg.c
  - Uart_TS_T40D2M20I0R0\generate_PB\src\Lpuart_Uart_Ip_PBcfg.c
  - Uart_TS_T40D2M20I0R0\generate_PB\src\Flexio_Uart_Ip_PBcfg.c
  - Uart_TS_T40D2M20I0R0\generate_PB\src\CDD_Uart_PBcfg.c
- Code Generation Templates for parameters without variation points:
  - Uart_TS_T40D2M20I0R0\generate_PC\include\Uart_Ipw_Cfg.h
  - Uart_TS_T40D2M20I0R0\generate_PC\include\CDD_Uart_Cfg.h
  - Uart_TS_T40D2M20I0R0\generate_PC\include\CDD_Uart_Defines.h
  - Uart_TS_T40D2M20I0R0\generate_PC\include\Lpuart_Uart_Ip_Cfg.h
  - Uart_TS_T40D2M20I0R0\generate_PC\include\Lpuart_Uart_Ip_Defines.h
  - Uart_TS_T40D2M20I0R0\generate_PC\include\Flexio_Uart_Ip_Cfg.h
  - Uart_TS_T40D2M20I0R0\generate_PC\include\Flexio_Uart_Ip_Defines.h

3.3.2 Steps to generate the configuration:

1. Copy the following module folders into the Tresos plugins folder:
   - BaseNXP_TS_T40D2M20I0R0
   - Det_TS_T40D2M20I0R0
   - EcuC_TS_T40D2M20I0R0
   - Rte_TS_T40D2M20I0R0
   - Resource_TS_T40D2M20I0R0
   - Mcu_TS_T40D2M20I0R0
   - Uart_TS_T40D2M20I0R0
   - Mcl_TS_T40D2M20I0R0
   - Os_TS_T40D2M20I0R0
2. Set the desired Tresos Output location folder for the generated sources and header files.
3. Use the EB Tresos Studio GUI to modify ECU configuration parameters values.
4. Generate the configuration files

dma_config

Chapter 4

Function calls to module

4.1 Function Calls during Start-up

If the Uart peripheral is used, the Uart driver must be initialized before.
The API to be called for the initialization of the driver during STARTUP is
Uart_Init(<&Uart_Configuration>).
After the Uart module is initialized, each Uart channel has to be initialized as well before using it.
This is also done by the **Uart_Init(<&Uart_Configuration>)** service.
The MCU module should be initialized before the Uart driver is initialized.
The PORT and MCL modules (if the DMA option and Flexio module are used) shall be initialized before Uart driver is initialized.

4.2 Function Calls during Shutdown

Uart module can be silenced by calling Uart_DeInit(). Note: Ensure that all the hw channels are not in transmission progress. If not, DeInit function will finish unsuccessfully, driver will report the runtime error for channel in the case user enables Det Runtime Report in configuration tools.

4.3 Function Calls during Wake-up

N/A

Chapter 5

Module requirements

5.1 Exclusive areas to be defined in BSW scheduler

In the current implementation, UART is using the services of Schedule Manager (SchM) for entering and exiting the critical regions, to preserve a resource. SchM implementation is done by the integrators of the MCAL using OS or non-OS services. For testing the UART, stubs are used for SchM. The following critical regions are used in the UART driver:

UART_EXCLUSIVE_AREA_00 is used in function Uart_SyncSend to protect the UartState->IsTxBusy variable from the read/modify/write operation in Lpuart_Uart_Ip_SyncSend.

UART_EXCLUSIVE_AREA_01 is used in function Uart_AsyncSend to protect the UartState->IsTxBusy variable from the read/modify/write operation in Lpuart_Uart_Ip_AsyncSend.

UART_EXCLUSIVE_AREA_02 is used in function Uart_SyncReceive to protect the UartState->IsRxBusy variable from the read/modify/write operation in Lpuart_Uart_Ip_SyncReceive.

UART_EXCLUSIVE_AREA_03 is used in function Uart_AsyncReceive to protect the UartState->IsRxBusy variable from the read/modify/write operation in Lpuart_Uart_Ip_AsyncReceive.

UART_EXCLUSIVE_AREA_04 is used in function Uart_AsyncSend to protect the UartState->DriverIdle variable from the read/modify/write operation in Flexio_Uart_Ip_AsyncSend.

UART_EXCLUSIVE_AREA_05 is used in function Uart_SyncSend to protect the UartState->DriverIdle variable from the read/modify/write operation in Flexio_Uart_Ip_SyncSend.

UART_EXCLUSIVE_AREA_06 is used in function Uart_AsyncReceive to protect the UartState->DriverIdle variable from the read/modify/write operation in Flexio_Uart_Ip_AsyncReceive.

UART_EXCLUSIVE_AREA_07 is used in function Uart_SyncReceive to protect the UartState->DriverIdle variable from the read/modify/write operation in Flexio_Uart_Ip_SyncReceive.

UART_EXCLUSIVE_AREA_08 is used in function Uart_AsyncSend to protect the UartState->DriverIdle variables from the read/write operation in Flexio_Uart_Ip_AsyncSend.

UART_EXCLUSIVE_AREA_09 is used in function Uart_SyncSend to protect the UartState->DriverIdle, UartState->Status, UartState->RemainingBytes variables from the read operation in Flexio_Uart_Ip_GetStatus.

UART_EXCLUSIVE_AREA_09 is used in function Uart_AsyncSend to protect the UartState->DriverIdle, UartState->Status, UartState->RemainingBytes variables from the read operation in Flexio_Uart_Ip_GetStatus.

UART_EXCLUSIVE_AREA_09 is used in function Uart_GetStatus to protect the UartState->DriverIdle, UartState->Status, UartState->RemainingBytes variables from the read operation in Flexio_Uart_Ip_GetStatus.

UART_EXCLUSIVE_AREA_09 is used in function Uart_Deinit to protect the UartState->DriverIdle, Uart↩ State->Status, UartState->RemainingBytes variables from the read operation in Flexio_Uart_Ip_GetStatus.

UART_EXCLUSIVE_AREA_09 is used in function Uart_SetBaudrate to protect the UartState->DriverIdle, UartState->Status, UartState->RemainingBytes variables from the read operation in Flexio_Uart_Ip_GetStatus.

UART_EXCLUSIVE_AREA_09 is used in function Uart_SyncReceive to protect the UartState->DriverIdle, UartState->Status, UartState->RemainingBytes variables from the read operation in Flexio_Uart_Ip_GetStatus.

UART_EXCLUSIVE_AREA_09 is used in function Uart_AsyncReceive to protect the UartState->DriverIdle, UartState->Status, UartState->RemainingBytes variables from the read operation in Flexio_Uart_Ip_GetStatus.

UART_EXCLUSIVE_AREA_10 is used in function Uart_SyncSend to protect the UartState->DriverIdle variables from the read/write operation in Flexio_Uart_Ip_SyncSend.

Exclusive Areas implemented in Low level driver layer (IPL)

UART_EXCLUSIVE_AREA_00 is used in function Lpuart_Uart_Ip_SyncSend to protect the UartState->IsTxBusy variable during the read/write action.

UART_EXCLUSIVE_AREA_01 is used in function Lpuart_Uart_Ip_AsyncSend to protect the UartState->IsTxBusy variable during the read/write action.

UART_EXCLUSIVE_AREA_02 is used in function Lpuart_Uart_Ip_SyncReceive to protect the UartState->IsRxBusy variable during the read/write action.

UART_EXCLUSIVE_AREA_03 is used in function Lpuart_Uart_Ip_AsyncReceive to protect the Uart↩ State->IsRxBusy variable during the read/write action.

UART_EXCLUSIVE_AREA_04 is used in function Lpuart_Uart_Ip_GetReceiveStatus to protect the Uart↩ State->IsRxBusy, UartState->ReceiveStatus, UartState->RxSize variables during the read action.

UART_EXCLUSIVE_AREA_05 is used in function Lpuart_Uart_Ip_GetTransmitStatus to protect the UartState->IsTxBusy, UartState->TransmitStatus, UartState->RxSize variables during the read action.

UART_EXCLUSIVE_AREA_06 is used in function Flexio_Uart_Ip_AsyncReceive to protect the UartState->DriverIdle variable during the read/write action.

UART_EXCLUSIVE_AREA_07 is used in function Flexio_Uart_Ip_SyncReceive to protect the UartState->DriverIdle variable during the read/write action.

UART_EXCLUSIVE_AREA_08 is used in function Flexio_Uart_Ip_AsyncSend to protect the UartState->DriverIdle variable during the read/write action.

UART_EXCLUSIVE_AREA_09 is used in function Flexio_Uart_Ip_GetStatus to protect the UartState->DriverIdle, UartState->Status, UartState->RemainingBytes variables during the read action.

UART_EXCLUSIVE_AREA_10 is used in function Flexio_Uart_Ip_SyncSend to protect the UartState->DriverIdle variable during the read/write action.

Exclusive Area ID	UART_EXCLUSIVE_AREA_00	UART_EXCLUSIVE_AREA_01	UART_EXCLUSIVE_AREA_05	UART_EXCLUSIVE_AREA_02	UART_EXCLUSIVE_AREA_03	UART_EXCLUSIVE_AREA_04	UART_EXCLUSIVE_AREA_08	UART_EXCLUSIVE_AREA_09	UART_EXCLUSIVE_AREA_10	UART_EXCLUSIVE_AREA_06	UART_EXCLUSIVE_AREA_07
UART_EXCLUSIVE_AREA_00	x	x									
UART_EXCLUSIVE_AREA_01	x	x									
UART_EXCLUSIVE_AREA_02				x	x						
UART_EXCLUSIVE_AREA_03				x	x						
UART_EXCLUSIVE_AREA_04			x			x					
UART_EXCLUSIVE_AREA_05			x			x					
UART_EXCLUSIVE_AREA_06							x	x	x	x	x
UART_EXCLUSIVE_AREA_07							x	x	x	x	x
UART_EXCLUSIVE_AREA_08							x		x	x	x
UART_EXCLUSIVE_AREA_09								x		x	x
UART_EXCLUSIVE_AREA_10							x		x	x	x

(Extracted table from RTD_LIN_EXCLUSIVE_AREAS.xlsx)

5.2 Exclusive areas not available on this platform

None.

5.3 Peripheral Hardware Requirements

N/A

5.4 ISR to configure within AutosarOS - dependencies

S32K11X derivatives:

ISR Name	IRQ Number
LPUART_UART_IP_0_IRQHandler	31
LPUART_UART_IP_1_IRQHandler	30
MCL_FLEXIO_ISR	25

S32K14X derivatives(except S32K142):

ISR Name	IRQ Number
LPUART_UART_IP_0_IRQHandler	31
LPUART_UART_IP_1_IRQHandler	33
LPUART_UART_IP_2_IRQHandler	35
MCL_FLEXIO_ISR	69

S32K142 derivative:

ISR Name	IRQ Number
LPUART_UART_IP_0_IRQHandler	31
LPUART_UART_IP_1_IRQHandler	33
MCL_FLEXIO_ISR	69

S32M24X derivatives:

ISR Name	IRQ Number
LPUART_UART_IP_0_IRQHandler	31
LPUART_UART_IP_1_IRQHandler	33
MCL_FLEXIO_ISR	69

5.5 ISR Macro

RTD drivers use the ISR macro to define the functions that will process hardware interrupts. Depending on whether the OS is used or not, this macro can have different definitions.

5.5.1 Without an Operating System

The macro *USING_OS_AUTOSAROS* must not be defined.

5.5.1.1 Using Software Vector Mode

The macro *USE_SW_VECTOR_MODE* must be defined and the ISR macro is defined as:

#define ISR(IsrName) void IsrName(void)

In this case, the drivers' interrupt handlers are normal C functions and their prologue/epilogue will handle the context save and restore.

5.5.1.2 Using Hardware Vector Mode

The macro *USE_SW_VECTOR_MODE* must not defined and the ISR macro is defined as:

#define ISR(IsrName) INTERRUPT_FUNC void IsrName(void)

In this case, the drivers' interrupt handlers must also handle the context save and restore.

5.5.2 With an Operating System

Please refer to your OS documentation for description of the ISR macro.

5.6 Other AUTOSAR modules - dependencies

- **BASE**: The BASE module contains the common files/definitions needed by all MCAL modules.
- **RESOURCE** :Resource module is used to select microcontroller's derivatives.
- **RTE** : The RTE module is needed for implementing data consistency of exclusive areas that are used by UART module.
- **DET**: The DET module is used for enabling Default Error Tracer detection. The API functions used are Det_ReportError() and Det_ReportRuntimeError(). The activation / deactivation of Default Error Tracer detection is configurable using the "UARTDevErrorDetect" configuration parameter. **DEM**: The DEM module is used for enabling reporting of production relevant error status. The API function used is Dem_Report↩ ErrorStatus().
- **ECUC**: The ECUC module is used for ECU configuration. MCAL modules need ECUC to retrieve the variant information.
- **PORT** : The PORT module is used to configure the port pins with the needed modes, before they are used by the UART module. For each LINFlex, the TXD and RXD signals need to be configured.
- **MCU**: The MCU driver provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required by other MCAL software modules. The clocks need to be initialized prior to using the UART driver. The UART reference clock is provided by MCU plugin. The clock frequency may affect the Baudrate, Timing between clock and chip select, Timing between chip select and clock, Timing between chip select assertions. The reference is specified by the parameter UARTGeneral\U↩ ARTClockRef.
- **MCL**: For each UART in use, a transmit and a receive DMA channel need to be defined and routed through the DMA Multiplexer using MCL plugin. switch to DMA mode.
  For using Flexio Uart. Flexio module need to be configured Flexio Channel and Flexio Pin in MCL.
  In two cases using Flexio or Dma. MCL should be initialized before UART.

5.7 Data Cache Restrictions

In the DMA transfer mode, DMA transfers may have cache coherency problems. To avoid possible coherency issues when **D-CACHE** is enabled, the user shall ensure that the buffers used as TCD source and destination are allocated in the **NON-CACHEABLE** area (by means of Uart_Memmap). Otherwise, the Uart driver has some dependencies. User must to put all variables, which were used for transmitter and receiver, in the **NON CACHEABLE** memory section in the RAM zone by the definition **UART_START_SEC_VAR_<INIT↩ _POLICY>_<ALIGNMENT>_NO_CACHEABLE** and **UART_STOP_SEC_VAR_<INIT_PO↩ LICY>_<ALIGNMENT>_NO_CACHEABLE**.

5.8 User Mode support

- User Mode configuration in the module
- User Mode configuration in AutosarOS

5.8.1 User Mode configuration in the module

No special measures need to be taken to run UART module from user mode. The UART driver code can be executed at any time from both supervisor and user mode.

5.8.2 User Mode configuration in AutosarOS

When User mode is enabled, the driver may have the functions that need to be called as trusted functions in AutosarOS context. Those functions are already defined in driver and declared in the header `<IpName>_Ip↩ _TrustedFunctions.h`. This header also included all headers files that contains all types definition used by parameters or return types of those functions. Refer the chapter User Mode configuration in the module for more detail about those functions and the name of header files they are declared inside. Those functions will be called indirectly with the naming convention below in order to AutosarOS can call them as trusted functions.

`Call_<Function_Name>_TRUSTED(parameter1,parameter2,...)`

That is the result of macro expansion `OsIf_Trusted_Call` in driver code:

`#define` OsIf_Trusted_Call[1-6params](name,param1,...,param6) Call_##name##_TRUSTED(param1,...,param6)

So, the following steps need to be done in AutosarOS:

- Ensure `MCAL_ENABLE_USER_MODE_SUPPORT` macro is defined in the build system or somewhere global.
- Define and declare all functions that need to call as trusted functions follow the naming convention above in Integration/User code. They need to visible in `Os.h` for the driver to call them. They will do the marshalling of the parameters and call `CallTrustedFunction()` in OS specific manner.
- `CallTrustedFunction()` will switch to privileged mode and call `TRUSTED_<Function_Name>()`.
- `TRUSTED_<Function_Name>()` function is also defined and declared in Integration/User code. It will unmarshalling of the parameters to call `<Function_Name>`() of driver. The `<Function_Name>`() functions are already defined in driver and declared in `<IpName>_Ip_TrustedFunctions.h`. This header should be included in OS for OS call and indexing these functions.

See the sequence chart below for an example calling `Linflexd_Uart_Ip_Init_Privileged()` as a trusted function.

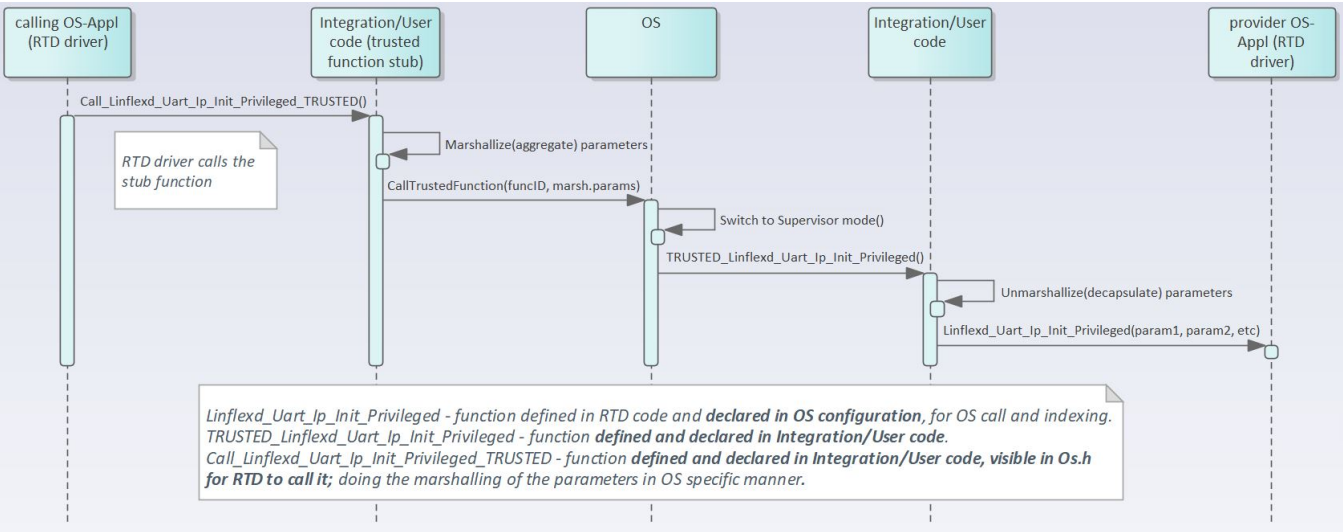

Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

S32K1_S32M24X product family is composed single core MCUs, so this feature is not available on this kind of platform.

Chapter 6

Main API Requirements

- Main function calls within BSW scheduler
- API Requirements
- Calls to Notification Functions, Callbacks, Callouts

6.1 Main function calls within BSW scheduler

N/A

6.2 API Requirements

N/A

6.3 Calls to Notification Functions, Callbacks, Callouts

Call-back Notifications: The driver provides callback notifications for asynchronous transfers. The driver gives possibility to configure callbacks for each channel to be called at the end of a transmission.

AUTOSAR Mode:

Callback for Receive mode: In interrupt mode, the callback is called twice. First time is when the Rx buffer is full, giving the opportunity for user to call Uart_SetBuffer() in order to provide a new buffer for a continuous reception. If a new buffer is not provided in the first call, a second call is performed for ending a reception. The event parameter in the callback signature is used to distinguish between the two calls. In case of the first one, the parameter is UART_EVENT_RX_FULL, while for the second one, the callback parameter is UART_EVENT↩ _END_TRANSFER.

Callback for Transmit mode: In case of the Tx callback, the same approach is used. There are also two calls of it during a transmission. In the first call, the event parameter is UART_EVENT_TX_EMPTY and indicates that all the bytes from the provided buffer have been sent. In this call, Uart_SetBuffer() can be called in order to

change the buffer and transfer more data. In case of the end of transmission (all data has been transfered), a second call is executed with UART_EVENT_END_TRANSFER parameter.

Callback for Error occurred: The driver treats the following errors: rx overrun, noise error and framing errors. In case of an error detected during an ongoing reception, the transfer is aborted and the callback is called with UART_EVENT_ERROR parameter.

-How to config Uart Callback Function:
LPUART: The Uart Callback Function can be configured in the UI in the UartCallback field.
FLEXIO: The Uart Callback Function can be configured in the UI in the Flexio Callback Function field.

Non-Autosar(Ip) Mode:
LPUART:

_Lpuart Callback Function: Lpuart Callback can be configured in the UartCallback field in the Design Studio configurator for the callback functions below:

Callback for Receive mode: In interrupt mode, the callback is called twice. First time is when the Rx buffer is full, giving the opportunity for user to call Lpuart_Uart_Ip_SetRxBuffer() in order to provide a new buffer for a continuous reception. If a new buffer is not provided in the first call, a second call is performed for ending a reception. The event parameter in the callback signature is used to distinguish between the two calls. In case of the first one, the parameter is LPUART_UART_IP_EVENT_RX_FULL, while for the second one, the callback parameter is LPUART_UART_IP_EVENT_END_TRANSFER.

Callback for Transmit mode: In case of the Tx callback, the same approach is used. There are also two calls of it during a transmission. In the first call, the event parameter is LPUART_UART_IP_EVENT_TX_EMPTY and indicates that all the bytes from the provided buffer have been sent. In this call, Lpuart_Uart_Ip_SetTxBuffer() can be called in order to change the buffer and transfer more data. In case of the end of transmission (all data has been transfered), a second call is executed with LPUART_UART_IP_EVENT_END_TRANSFER parameter.

Callback for Error occurred: The driver treats the following errors: rx overrun, noise error and framing errors. In case of an error detected during an ongoing reception, the transfer is aborted and the callback is called with LPUART_UART_IP_EVENT_ERROR parameter.

Parameter for Uart Callback:
Callback for all peripherals which support UART features
typedef void (∗Lpuart_Uart_Ip_CallbackType)(const uint8 HwInstance, const Lpuart_Uart_Ip_EventType Event, void ∗UserData);
HwChannel: Lpuart Hardware Channel.
Event: Lpuart Uart event which can trigger UART callback.
UserData: User data passing onto callback function. It is a variable which can be used by the application. If UserData is not desired, the appropriate Param for Uart Callback field shall be left blank.

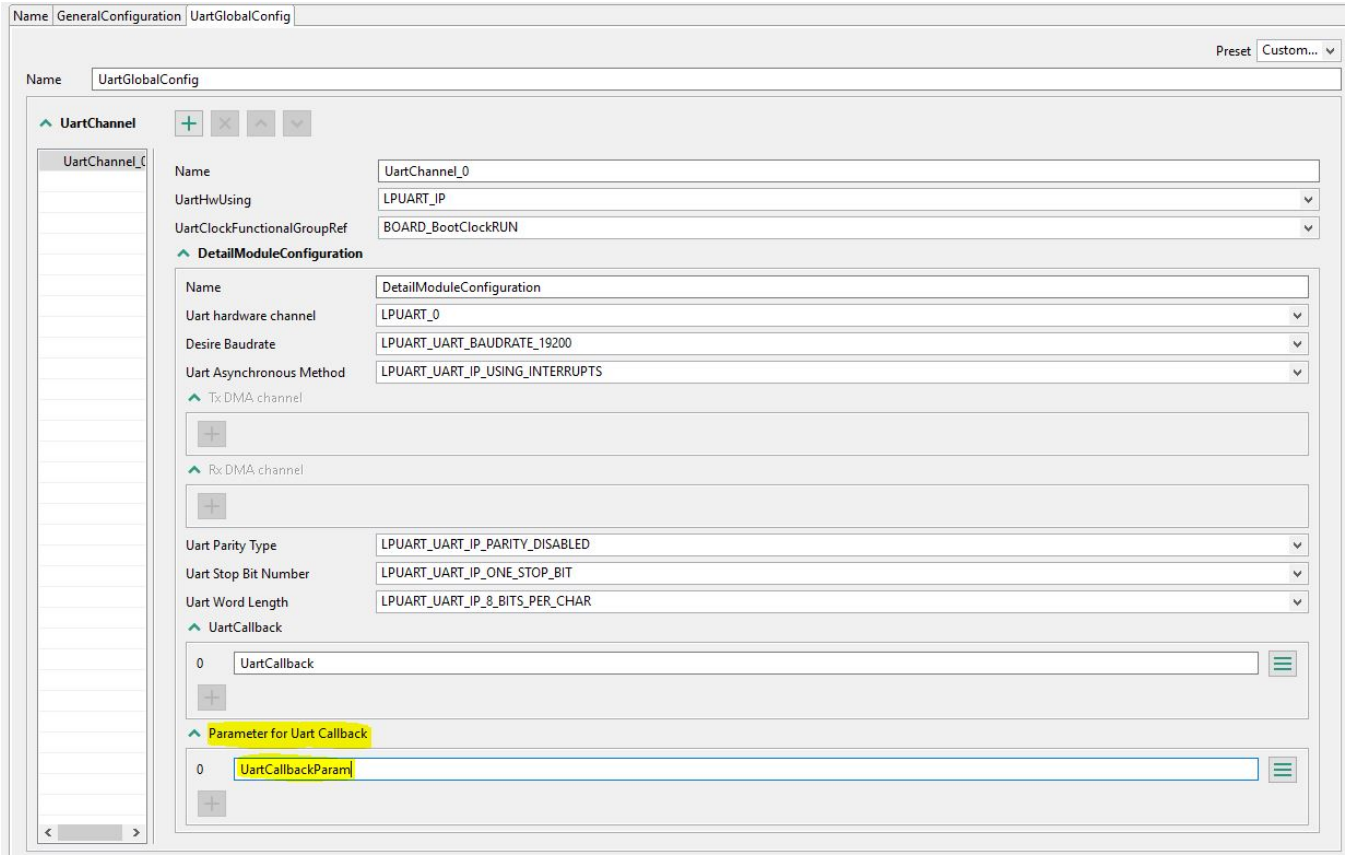

Figure 6.1 Lpuart Callback Param

FLEXIO UART:

- Flexio Callback Function: Flexio Callback can be configured in the Flexio Callback Function field in the Design Studio configurator for the callback functions below:

  Callback for Receive mode: In interrupt mode, the callback is called twice. First time is when the Rx buffer is full, giving the opportunity for user to call Flexio_Uart_Ip_SetRxBuffer() in order to provide a new buffer for a continuous reception. If a new buffer is not provided in the first call, a second call is performed for ending a reception. The event parameter in the callback signature is used to distinguish between the two calls. In case of the first one, the parameter is FLEXIO_UART_IP_EVENT_RX_F↩ ULL, while for the second one, the callback parameter is FLEXIO_UART_IP_EVENT_END_TRANSFER.

  Callback for Transmit mode: In case of the Tx callback, the same approach is used. There are also two calls of it during a transmission. In the first call, the event parameter is FLEXIO_UART_IP_EVEN↩ T_TX_EMPTY and indicates that all the bytes from the provided buffer have been sent. In this call, Flexio_Uart_Ip_SetTxBuffer() can be called in order to change the buffer and transfer more data. In case of the end of transmission (all data has been transfered), a second call is executed with FLEXIO_UART_IP↩ _EVENT_END_TRANSFER parameter.

Callback for Error occurred: The driver treats the following errors: rx overrun, noise error and framing errors. In case of an error detected during an ongoing reception, the transfer is aborted and the callback is called with FLEXIO_UART_IP_EVENT_ERROR parameter.

Parameter for Callback Function:
Callback for all peripherals which support UART features
typedef void (∗Flexio_Uart_Ip_CallbackType)(const uint32 HwChannel, const Flexio_Uart_Ip_EventType Event, void ∗UserData);
HwChannel: Flexio Hardware Channel.
Event: Flexio Uart event which can trigger UART callback.
UserData: User data passing onto callback function. It is a variable which can be used by the application. If UserData is not desired, the appropriate Parameter for Transfer Callback field shall be left blank.

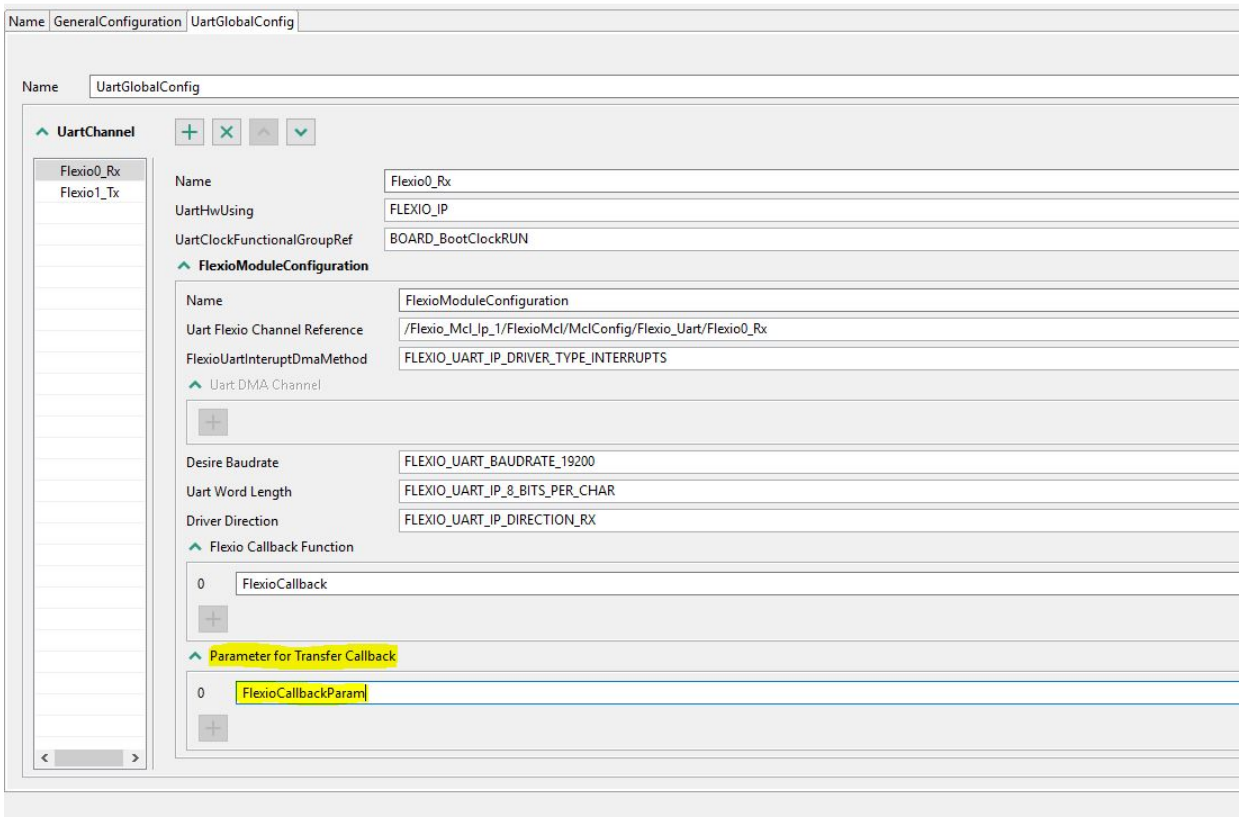

Figure 6.2 Flexio Callback Param

DMA Call-back Notifications: The DMA Tx/Rx notification must be enabled for the specified Tx/Rx D↩ MA channel. The name of the function to be used as a notification is Lpuart_X_Uart_Ip_DmaTxComplete↩ Callback for Tx or Lpuart_X_Uart_Ip_DmaRxCompleteCallback for Rx in LPUART module, where X is the number of Uart unit used and Flexio_Y_Uart_Ip_DmaTxCompleteCallback for Tx or Flexio_Y_Uart_Ip↩ _DmaRxCompleteCallback for Rx in FLEXIO module, where Y is number of FLEXIO channel.

This notification function is filled on both "Interrupt Callback" and "Error Interrupt Callback" field in dma config, refer the Figure 3.17 and Figure 3.19 in User Manual document. The user can perform error checking of the DMA module in the function which is filled in "Error Interrupt Callback" field.

Table 6.1 UART DMA Notification (S32K14X derivatives except S32K142)

Physical Unit	UART TX DMA Notification Name	UART RX DMA Notification Name
LPUART_0	Lpuart_0_Uart_Ip_DmaTxComplete↩ Callback	Lpuart_0_Uart_Ip_DmaRxComplete↩ Callback
LPUART_1	Lpuart_1_Uart_Ip_DmaTxComplete↩ Callback	Lpuart_1_Uart_Ip_DmaRxComplete↩ Callback
LPUART_2	Lpuart_2_Uart_Ip_DmaTxComplete↩ Callback	Lpuart_2_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_0	Flexio_0_Uart_Ip_DmaTxComplete↩ Callback	Flexio_0_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_1	Flexio_1_Uart_Ip_DmaTxComplete↩ Callback	Flexio_1_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_2	Flexio_2_Uart_Ip_DmaTxComplete↩ Callback	Flexio_2_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_3	Flexio_3_Uart_Ip_DmaTxComplete↩ Callback	Flexio_3_Uart_Ip_DmaRxComplete↩ Callback

Table 6.2 UART DMA Notification (S32K11X, S32M24X derivatives and S32K142)

Physical Unit	UART TX DMA Notification Name	UART RX DMA Notification Name
LPUART_0	Lpuart_0_Uart_Ip_DmaTxComplete↩ Callback	Lpuart_0_Uart_Ip_DmaRxComplete↩ Callback
LPUART_1	Lpuart_1_Uart_Ip_DmaTxComplete↩ Callback	Lpuart_1_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_0	Flexio_0_Uart_Ip_DmaTxComplete↩ Callback	Flexio_0_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_1	Flexio_1_Uart_Ip_DmaTxComplete↩ Callback	Flexio_1_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_2	Flexio_2_Uart_Ip_DmaTxComplete↩ Callback	Flexio_2_Uart_Ip_DmaRxComplete↩ Callback
FLEXIO_3	Flexio_3_Uart_Ip_DmaTxComplete↩ Callback	Flexio_3_Uart_Ip_DmaRxComplete↩ Callback

Chapter 7

Memory allocation

- Sections to be defined in Uart_MemMap.h
- Linker command file

7.1 Sections to be defined in Uart_MemMap.h

Section name	Type of section	Description
UART_START_SEC_CONFIG_↩ DATA_UNSPECIFIED	Configuration Data	Start of the Memory Section for Config Data when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit.
UART_STOP_SEC_CONFIG_D↩ ATA_UNSPECIFIED	Configuration Data	End of the Memory Section for Config Data
UART_START_SEC_CODE	Code	Start of the memory Section for Code
UART_STOP_SEC_CODE	Code	End of the memory Section for Code
UART_START_SEC_VAR_INI↩ T_8	Variables	Used for variables, structures, arrays when the SIZE (alignment) fit the criteria of 8 bit.
UART_STOP_SEC_VAR_INIT↩ _8	Variables	End of the above section.
UART_START_SEC_VAR_CL↩ EARED_UNSPECIFIED	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. These variables are cleared to zero by start-up code.
UART_STOP_SEC_VAR_CLE↩ ARED_UNSPECIFIED	Variables	End of the above section.
UART_START_SEC_CONST_B↩ OOLEAN	Configuration Data	Start of the Memory Section for constant data which have to be aligned to boolean type (1 bit).
UART_STOP_SEC_CONST_B↩ OOLEAN	Configuration Data	End of the Memory Section for constant data.
UART_START_SEC_CONST_↩ UNSPECIFIED	Configuration Data	Start of Memory Section for constant data when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit.

Section name	Type of section	Description
UART_STOP_SEC_CONST_U↩ NSPECIFIED	Configuration Data	End of Memory Section for constant data.
UART_START_SEC_VAR_CL↩ EARED_32	Variables	Used for variables which have to be aligned to 32 bit. For instance used for variables of size 32 bit or used for composite data types:arrays, structures containing elements of maximum 32 bits. These variables are cleared to zero by start-up code.
UART_STOP_SEC_VAR_CLE↩ ARED_32	Variables End of above section.	End of above section.
UART_START_SEC_VAR_CL↩ EARED_UNSPECIFIED_NO_C↩ ACHEABLE	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit, and that have to be stored in a non-cacheable memory section. These variables are cleared to zero by start-up code.
UART_STOP_SEC_VAR_CLE↩ ARED_UNSPECIFIED_NO_CA↩ CHEABLE	Variables	End of above section.

7.2 Linker command file

Memory shall be allocated for every section defined in the driver's "<Module>"_MemMap.h.

Chapter 8

Integration Steps

This section gives a brief overview of the steps needed for integrating this module:

1. Generate the required module configuration(s). For more details refer to section Files Required for Compilation
2. Allocate the proper memory sections in the driver's memory map header file ("<Module>"_MemMap.h) and linker command file. For more details refer to section Sections to be defined in <Module>_MemMap.h
3. Compile & build the module with all the dependent modules. For more details refer to section Building the Driver

Chapter 9

External assumptions for driver

The section presents requirements that must be complied with when integrating the UART driver into the application.

External Assumption Req ID	External Assumption Text
EA_RTD_00071	If interrupts are locked, a centralized function pair to lock and unlock interrupts shall be used.
EA_RTD_00081	The integrator shall assure that <MSN>_Init() and <MSN>_DeInit() functions do not interrupt each other.
EA_RTD_00082	When caches are enabled and data buffers are allocated in cacheable memory regions the buffers involved in DMA transfer shall be aligned with both start and end to cache line size. Note: **Rationale**: This ensures that no other buffers/variables compete for the same cache lines.
EA_RTD_00106	Standalone IP configuration and HL configuration of the same driver shall be done in the same project
EA_RTD_00107	The integrator shall use the IP interface only for hardware resources that were configured for standalone IP usage. Note: The integrator shall not directly use the IP interface for hardware resources that were allocated to be used in HL context.
EA_RTD_00108	The integrator shall use the IP interface to a build a CDD, therefore the BSWMD will not contain reference to the IP interface
EA_RTD_00113	When RTD drivers are integrated with AutosarOS and User mode support is enabled, the integrator shall assure that the definition and declaration of all RTD functions needed to be called as trusted functions follow the naming convention Call<Function_Name>TRUSTED(parameter1,parameter2,...) in Integration/User code. They need to visible in Os.h for the driver to call them. They will call RTD <Function_Name>() as trusted functions in OS specific manner.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

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

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