Integration Manual

for S32K1 OCU Driver

Document Number: IM2OCUASR4.4 Rev0000R1.0.1 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	24.02.2022	NXP RTD Team	Prepared for release RTD S32K1 Version 1.0.1

Chapter 2

Introduction

This integration manual describes the integration requirements for Ocu Driver for S32K1XX 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
- s32k144_lqfp100

- s32k144_mapbga100
- s32k144w_lqfp48
- s32k144w_lqfp64
- s32k146_lqfp64
- s32k146_lqfp100
- s32k146_mapbga100
- s32k146_lqfp144
- s32k148_lqfp100
- s32k148_mapbga100
- s32k148_lqfp144
- s32k148_lqfp176

All of the above microcontroller devices are collectively named as S32K1.

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
AUTOSAR	Automotive Open System Architecture
API	Application Programming Interface
ARTD	Automotive Real Time Drivers
ASR	AUTOSAR
BSW	Basic Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DIO	Digital Input Output
DMA	Direct Memory Access
ECU	Electronic Control Unit
ECUC	ECU Configuration
EcuM	ECU state Manager
eMIOS	Enhanced Modular IO Subsystem
FTM	FlexTimer
GUI	Graphical User Interface
HLD	High Level Driver
HW	Hardware
ICU	Input Capture Unit
IP	Intellectual Property, referred as a hardware design block
IPL	IP Layer
IPW	IP Wrapper Layer
ISR	Interrupt Service Routine
MCAL	Microconroller Abstraction Layer
MCU	Micro Controller Unit
N/A	Not Applicable
OCU	Output Compare Unit
OS	Operating System
OSIF	OS Interface
PB Variant	Post Build Variant
PC Variant	Pre Compile Variant
PWM	Pulse Width Modulation
RAM	Random Access Memory
ROM	Read-only Memory
SCI	Serial Communication Interface
SoC	System on Chip
SPI	Serial Peripheral Interface
SWS	Software Specification
VLE	Variable Length Encoding
VSMD	Vendor Specific Module Definition
XML	Extensible Markup Language

2.5 Reference List

#	Title	Version
1	Specification of OCU Driver	AUTOSAR Release 4.↩ 4.0
2	S32K1xx Series Reference Manual	Rev. 14, 09/2021
3	S32K1xx Data Sheet	Rev. 14, 08/2021
4	Errata S32K116_0N96V	Rev. 22/OCT/2021
5	Errata S32K118_0N97V	Rev. 22/OCT/2021
6	Errata S32K142_0N33V	Rev. 22/OCT/2021
7	Errata S32K144_0N57U	Rev. 22/OCT/2021
8	Errata S32K144W_0P64A	Rev. 22/OCT/2021
9	Errata S32K146_0N73V	Rev. 22/OCT/2021
10	Errata S32K148_0N20V	Rev. 22/OCT/2021

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

The RTD driver files are compiled using:

- NXP GCC 9.2.0 20190812 (Build 1649 Revision gaf57174)
- 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_T40D2M10I1R0 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 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 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)
-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 dont 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	Makes the compiler output stack usage information for the program, on a per-function basis
-fdump-ipa-all	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
-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 initalization in source file system.c under the Platform driver (for S32K14x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver (for S32K14x 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.

3.1.1.2 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 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
-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 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 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)
-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

3.1.2 GHS Compiler/Assembler/Linker Options

3.1.2.1 GHS Compiler Options

Compiler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x 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 devices)
-fsingle	Use hardware single-precision, software double-precision FP instructions (for S32K14x devices)

Compiler Option	Description
-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
–unsigned_fields	Bitfelds 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 andgenerate 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
-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 initalization in source file system.c under the Platform driver (for S32K14x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver (for S32K14x devices)

Compiler Option	Description
-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 GHS Assembler Options

Assembler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x devices)
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+ (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
-c	Stop after assembly and produce an object file for each source file

3.1.2.3 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. DWA↩ RF 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 themodules 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 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

Linker Option	Description
-nostartfiles	Controls the start files to be linked into the executable

3.1.3 IAR Compiler/Assembler/Linker Options

3.1.3.1 IAR Compiler Options

Compiler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x 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 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
-DIAR	Predefine IAR 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.

Compiler Option	Description
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver (for S32K14x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver (for S32K14x 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 IAR Assembler Options

Assembler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x devices)
–cpu=Cortex-M0+	Targeted ARM processor for which IAR should tune the performance of the code (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

3.1.3.3 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 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)
–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

3.2 Files required for compilation

This section describes the include files required to compile, assemble (if assembler code) and link the OCU driver for S32 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.

Ocu File:

- ../Ocu_TS_T40D2M10I1R0/include/Ocu_EnvCfg.h
- ../Ocu_TS_T40D2M10I1R0/include/Ocu_Types.h
- ../Ocu_TS_T40D2M10I1R0/include/Ocu_Irq.h
- ../Ocu_TS_T40D2M10I1R0/include/Ocu.h
- ../Ocu_TS_T40D2M10I1R0/src/Ocu.c
- ../Ocu_TS_T40D2M10I1R0/include/Ocu_Ipw_Types.h
- ../Ocu_TS_T40D2M10I1R0/include/Ocu_Ipw.h
- ../Ocu_TS_T40D2M10I1R0/src/Ocu_Ipw.c
- ../Ocu_TS_T40D2M10I1R0/include/Ftm_Ocu_Ip_Types.h
- ../Ocu_TS_T40D2M10I1R0/include/Ftm_Ocu_Ip_Irq.h
- ../Ocu_TS_T40D2M10I1R0/include/Ftm_Ocu_Ip_HwAccess.h
- ../Ocu_TS_T40D2M10I1R0/include/Ftm_Ocu_Ip.h
- ../Ocu_TS_T40D2M10I1R0/src/Ftm_Ocu_Ip_Irq.c
- ../Ocu_TS_T40D2M10I1R0/src/Ftm_Ocu_Ip.c

Ocu Generated Files:

- ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ftm_Ocu_Ip_CfgDefines.h
- ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ftm_Ocu_Ip_Cfg.h
- ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ocu_Ipw_CfgDefines.h
- ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ocu_Ipw_Cfg.h
- ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ocu_CfgDefines.h
- ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ocu_Cfg.h
- ../Ocu_TS_T40D2M10I1R0/generate_PB/include/Ftm_Ocu_Ip_PBcfg.h
- ../Ocu_TS_T40D2M10I1R0/generate_PB/src/Ftm_Ocu_Ip_PBcfg.c
- ../Ocu_TS_T40D2M10I1R0/generate_PB/include/Ocu_Ipw_PBcfg.h
- ../Ocu_TS_T40D2M10I1R0/generate_PB/src/Ocu_Ipw_PBcfg.c
- ../Ocu_TS_T40D2M10I1R0/generate_PB/include/Ocu_PBcfg.h

- ../Ocu_TS_T40D2M10I1R0/generate_PB/src/Ocu_PBcfg.c

Files from Base common folder:

- ../Base_TS_T40D2M10I1R0/include/Devassert.h
- ../Base_TS_T40D2M10I1R0/include/StandardTypes.h
- ../Base_TS_T40D2M10I1R0/include/Ocu_MemMap.h
- ../Base_TS_T40D2M10I1R0/include/Reg_eSys.h
- ../Base_TS_T40D2M10I1R0/header/S32K116_FTM.h
- ../Base_TS_T40D2M10I1R0/header/S32K118_FTM.h
- ../Base_TS_T40D2M10I1R0/header/S32K142_FTM.h
- ../Base_TS_T40D2M10I1R0/header/S32K142W_FTM.h
- ../Base_TS_T40D2M10I1R0/header/S32K144_FTM.h
- ../Base_TS_T40D2M10I1R0/header/S32K144W_FTM.h
- ../Base_TS_T40D2M10I1R0/header/S32K146_FTM.h
- ../Base_TS_T40D2M10I1R0/header/S32K148_FTM.h

Files from Det folder:

- ../Det_TS_T40D2M10I1R0/include/Det.h

Files from Rte folder:

- ../Rte_TS_T40D2M10I1R0/include/SchM_Ocu.h

3.3 Setting up the plugins

The Ocu driver was designed to be configured by using the EB Tresos Studio (version EB tresos Studio 27.1 or later.)

Location of various files inside the FR module folder:

- VSMD (Vendor Specific Module Definition) file in EB tresos Studio XDM format:
  - ../Dem_TS_T40D2M10I1R0/config/Dem.xdm
  - ../Base_TS_T40D2M10I1R0/config/Base.xdm
  - ../Ocu_TS_T40D2M10I1R0/config/Ocu.xdm
  - ../Mcu_TS_T40D2M10I1R0/config/Mcu.xdm
  - ../Resource_TS_T40D2M10I1R0/config/Resource.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:

  - ../Dem_TS_T40D2M10I1R0/autosar/Dem.epd
  - ../Base_TS_T40D2M10I1R0/autosar/Base.epd
  - ../Ocu_TS_T40D2M10I1R0/autosar/Ocu<subderivative_name>.epd
  - ../Mcu_TS_T40D2M10I1R0/autosar/Mcu<subderivative_name>.epd
  - ../Resource_TS_T40D2M10I1R0/autosar/Resource.epd

- Code Generation Templates for parameters:
  - ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ocu_CfgDefines.h
  - ../Ocu_TS_T40D2M10I1R0/generate_PC/include/Ocu_Cfg.h
  - ../Ocu_TS_T40D2M10I1R0/generate_PB/src/Ocu_PBcfg.c
  - ../Mcu_TS_T40D2M10I1R0/generate_PC/include/Clock_Ip_Cfg_Defines.h
  - ../Mcu_TS_T40D2M10I1R0/generate_PC/include/Clock_Ip_Cfg.h
  - ../Mcu_TS_T40D2M10I1R0/generate_PC/include/Mcu_Cfg.h
  - ../Mcu_TS_T40D2M10I1R0/generate_PB/src/Clock_Ip_PBcfg.c
  - ../Mcu_TS_T40D2M10I1R0/generate_PB/src/Mcu_PBcfg.c

Steps to generate the configuration:

1. Copy the module folders Ocu_TS_T40D2M10I1R0, Dem_TS_T40D2M10I1R0, Base_TS_T40D2M10I1R0, Resource_TS_T40D2M10I1R0, Mcu_TS_T40D2M10I1R0, into the Tresos plugins folder.
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.

Dependencies:

- MCU is required to use System Clock when clock source is used as Peripheral clock source to generate Ocu Segment values.
- MCL is required to provide some common files used by FTM peripherals, configuration Masterbus for eMios peripherals.
- RESOURCE is required to select processor derivative. Current Ocu driver has support for the following derivatives, everyone having attached a Resource file
- ECUC is needed to allows users to configure multiple configuration.
- DET is required for signaling the development error detection (parameters out of range, null pointers, etc).
- PORT is required to configure port I/O.

Chapter 4

Function calls to module

4.1 Function Calls during Start-up

Ocu shall be initialized during STARTUP phase of EcuM initialization. The API to be called for this is Ocu _Init(). The MCU module should be initialized before the Ocu is initialized.

4.2 Function Calls during Shutdown

During shutdown phase, Ocu_DeInit() function can be called. Calling this function depends on the initialization-deinitialization strategy deployed by user.

4.3 Function Calls during Wake-up

During Wake-up phase, Ocu_Init() function may be called but only if during a previous phase Ocu_DeInit() was called. Calling this function depends on the initialization deinitialization strategy deployed by user.

Chapter 5

Module requirements

5.1 Exclusive areas to be defined in BSW scheduler

In the current implementation, OCU is using the services of Schedule Manager (SchM) for entering and exiting the exclusive areas. The following critical regions are used in the OCU driver:

OCU_EXCLUSIVE_AREA_10 is used in functions
ISR(FTM_0_CH_0_CH_1_ISR), ISR(FTM_0_CH_2_CH_3_ISR), ISR(FTM_0_CH_4_CH_5_ISR), ISR(FTM_0_CH_6_CH_7_ISR), ISR(FTM_1_CH_0_CH_1_ISR), ISR(FTM_1_CH_2_CH_3_ISR), ISR(FTM_1_CH_4_CH_5_ISR), ISR(FTM_1_CH_6_CH_7_ISR), ISR(FTM_2_CH_0_CH_1_ISR), ISR(FTM_2_CH_2_CH_3_ISR), ISR(FTM_2_CH_4_CH_5_ISR), ISR(FTM_2_CH_6_CH_7_ISR), ISR(FTM_3_CH_0_CH_1_ISR), ISR(FTM_3_CH_2_CH_3_ISR), ISR(FTM_3_CH_4_CH_5_ISR), ISR(FTM_3_CH_6_CH_7_ISR), ISR(FTM_4_CH_0_CH_1_ISR), ISR(FTM_4_CH_2_CH_3_ISR), ISR(FTM_4_CH_4_CH_5_ISR), ISR(FTM_4_CH_6_CH_7_ISR), ISR(FTM_5_CH_0_CH_1_ISR), ISR(FTM_5_CH_2_CH_3_ISR), ISR(FTM_5_CH_4_CH_5_ISR), ISR(FTM_5_CH_6_CH_7_ISR), ISR(FTM_6_CH_0_CH_1_ISR), ISR(FTM_6_CH_2_CH_3_ISR), ISR(FTM_6_CH_4_CH_5_ISR), ISR(FTM_6_CH_6_CH_7_ISR), ISR(FTM_7_CH_0_CH_1_ISR), ISR(FTM_7_CH_2_CH_3_ISR), ISR(FTM_7_CH_4_CH_5_ISR), ISR(FTM_7_CH_6_CH_7_ISR)
to protect the updates for:

- FTM_CSC

OCU_EXCLUSIVE_AREA_11 is used in function Ocu_StartChannel to protect the updates for:

- FTM_CSC
- FTM_SWOCTRL

OCU_EXCLUSIVE_AREA_12 is used in function Ocu_StopChannel to protect the updates for:

- FTM_SWOCTRL

OCU_EXCLUSIVE_AREA_13 is used in function Ocu_SetPinAction to protect the updates for:

- FTM_CSC

OCU_EXCLUSIVE_AREA_14 is used in function Ocu_SetPinState to protect the updates for:

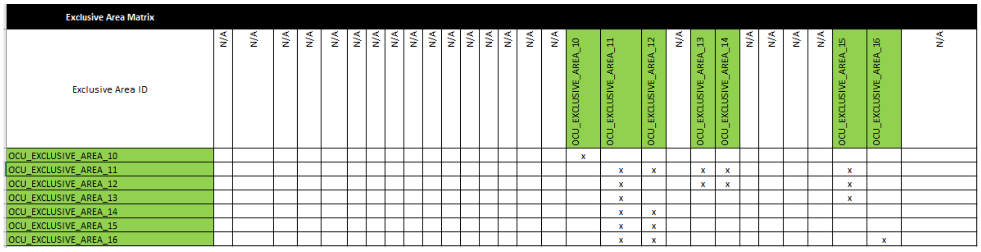
- FTM_CSC

OCU_EXCLUSIVE_AREA_15 is used in functions
ISR(FTM_0_CH_0_CH_1_ISR), ISR(FTM_0_CH_2_CH_3_ISR), ISR(FTM_0_CH_4_CH_5_ISR), ISR(FTM_0_CH_6_CH_7_ISR), ISR(FTM_1_CH_0_CH_1_ISR), ISR(FTM_1_CH_2_CH_3_ISR), ISR(FTM_1_CH_4_CH_5_ISR), ISR(FTM_1_CH_6_CH_7_ISR), ISR(FTM_2_CH_0_CH_1_ISR), ISR(FTM_2_CH_2_CH_3_ISR), ISR(FTM_2_CH_4_CH_5_ISR), ISR(FTM_2_CH_6_CH_7_ISR), ISR(FTM_3_CH_0_CH_1_ISR), ISR(FTM_3_CH_2_CH_3_ISR), ISR(FTM_3_CH_4_CH_5_ISR), ISR(FTM_3_CH_6_CH_7_ISR), ISR(FTM_4_CH_0_CH_1_ISR), ISR(FTM_4_CH_2_CH_3_ISR), ISR(FTM_4_CH_4_CH_5_ISR), ISR(FTM_4_CH_6_CH_7_ISR), ISR(FTM_5_CH_0_CH_1_ISR), ISR(FTM_5_CH_2_CH_3_ISR), ISR(FTM_5_CH_4_CH_5_ISR), ISR(FTM_5_CH_6_CH_7_ISR), ISR(FTM_6_CH_0_CH_1_ISR), ISR(FTM_6_CH_2_CH_3_ISR), ISR(FTM_6_CH_4_CH_5_ISR), ISR(FTM_6_CH_6_CH_7_ISR), ISR(FTM_7_CH_0_CH_1_ISR), ISR(FTM_7_CH_2_CH_3_ISR), ISR(FTM_7_CH_4_CH_5_ISR), ISR(FTM_7_CH_6_CH_7_ISR)
to protect the updates for:

- FTM_CSC

OCU_EXCLUSIVE_AREA_16 is used in functions
ISR(FTM_0_CH_0_CH_1_ISR), ISR(FTM_0_CH_2_CH_3_ISR), ISR(FTM_0_CH_4_CH_5_ISR), ISR(FTM_0_CH_6_CH_7_ISR), ISR(FTM_1_CH_0_CH_1_ISR), ISR(FTM_1_CH_2_CH_3_ISR), ISR(FTM_1_CH_4_CH_5_ISR), ISR(FTM_1_CH_6_CH_7_ISR), ISR(FTM_2_CH_0_CH_1_ISR), ISR(FTM_2_CH_2_CH_3_ISR), ISR(FTM_2_CH_4_CH_5_ISR), ISR(FTM_2_CH_6_CH_7_ISR), ISR(FTM_3_CH_0_CH_1_ISR), ISR(FTM_3_CH_2_CH_3_ISR), ISR(FTM_3_CH_4_CH_5_ISR), ISR(FTM_3_CH_6_CH_7_ISR), ISR(FTM_4_CH_0_CH_1_ISR), ISR(FTM_4_CH_2_CH_3_ISR), ISR(FTM_4_CH_4_CH_5_ISR), ISR(FTM_4_CH_6_CH_7_ISR), ISR(FTM_5_CH_0_CH_1_ISR), ISR(FTM_5_CH_2_CH_3_ISR), ISR(FTM_5_CH_4_CH_5_ISR), ISR(FTM_5_CH_6_CH_7_ISR), ISR(FTM_6_CH_0_CH_1_ISR), ISR(FTM_6_CH_2_CH_3_ISR), ISR(FTM_6_CH_4_CH_5_ISR), ISR(FTM_6_CH_6_CH_7_ISR), ISR(FTM_7_CH_0_CH_1_ISR), ISR(FTM_7_CH_2_CH_3_ISR), ISR(FTM_7_CH_4_CH_5_ISR), ISR(FTM_7_CH_6_CH_7_ISR)
to protect the updates for:

- FTM_SC



Exclusive Area Matrix

Exclusive Area ID	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	OCU_EXCLUSIVE_AREA_10	OCU_EXCLUSIVE_AREA_11	OCU_EXCLUSIVE_AREA_12	N/A	OCU_EXCLUSIVE_AREA_13	OCU_EXCLUSIVE_AREA_14	N/A	N/A	N/A	N/A	OCU_EXCLUSIVE_AREA_15	OCU_EXCLUSIVE_AREA_16	N/A
OCU_EXCLUSIVE_AREA_10																	x												
OCU_EXCLUSIVE_AREA_11																		x	x		x	x					x		
OCU_EXCLUSIVE_AREA_12																		x			x	x					x		
OCU_EXCLUSIVE_AREA_13																		x									x		
OCU_EXCLUSIVE_AREA_14																		x	x										
OCU_EXCLUSIVE_AREA_15																		x	x										
OCU_EXCLUSIVE_AREA_16																		x	x									x	

Figure 5.1 Exclusive Areas

The critical regions from interrupts are grouped in “Interrupt Service Routines Critical Regions (composed diagram)”. If an exclusive area is “exclusive” with the composed “Interrupt Service Routines Critical Regions (composed diagram)” group, it means that it is exclusive with each one of the ISR critical regions.

5.2 Exclusive areas unavailable on this platform

None

5.3 Peripheral Hardware Requirements

For S32K1 controllers, Ocu functionality is provided by the and FTM modules

5.4 ISR to configure within AutosarOS - dependencies

The following ISR’s are used by the OCU driver: The ISR table is presented below. Depending on the derivative used, some of the ISRs may not be available. For complete details please consult the Reference Manual:

ISR Name	CM4 Hardware interrupt vector	Observations
For FTM0	-	-
For S32K11X derivatives	-	-
ISR(FTM_0_ISR)	12	FTM0 shared interrupt for 0 & 7 channels
For S32K14X derivatives	-	-
ISR(FTM_0_CH_0_CH_1_ISR)	99	FTM0 shared interrupt for 0 & 1 channels
ISR(FTM_0_CH_2_CH_3_ISR)	100	FTM0 shared interrupt for 2 & 3 channels
ISR(FTM_0_CH_4_CH_5_ISR)	101	FTM0 shared interrupt for 4 & 5 channels
ISR(FTM_0_CH_6_CH_7_ISR)	102	FTM0 shared interrupt for 6 & 7 channels
For FTM1	-	-
For S32K11X derivatives	-	-
ISR(FTM_1_ISR)	15	FTM1 shared interrupt for 0 & 7 channels
For S32K14X derivatives	-	-
ISR(FTM_1_CH_0_CH_1_ISR)	105	FTM1 shared interrupt for 0 & 1 channels
ISR(FTM_1_CH_2_CH_3_ISR)	106	FTM1 shared interrupt for 2 & 3 channels
ISR(FTM_1_CH_4_CH_5_ISR)	107	FTM1 shared interrupt for 4 & 5 channels
ISR(FTM_1_CH_6_CH_7_ISR)	108	FTM1 shared interrupt for 6 & 7 channels
For S32K14X derivatives	-	-
For FTM2	-	-
ISR(FTM_2_CH_0_CH_1_ISR)	111	FTM2 shared interrupt for 0 & 1 channels
ISR(FTM_2_CH_2_CH_3_ISR)	112	FTM2 shared interrupt for 2 & 3 channels
ISR(FTM_2_CH_4_CH_5_ISR)	113	FTM2 shared interrupt for 4 & 5 channels
ISR(FTM_2_CH_6_CH_7_ISR)	114	FTM2 shared interrupt for 6 & 7 channels
For FTM3	-	-
ISR(FTM_3_CH_0_CH_1_ISR)	117	FTM3 shared interrupt for 0 & 1 channels
ISR(FTM_3_CH_2_CH_3_ISR)	118	FTM3 shared interrupt for 2 & 3 channels
ISR(FTM_3_CH_4_CH_5_ISR)	119	FTM3 shared interrupt for 4 & 5 channels
ISR(FTM_3_CH_6_CH_7_ISR)	120	FTM3 shared interrupt for 6 & 7 channels
For FTM4	-	-
ISR(FTM_4_CH_0_CH_1_ISR)	123	FTM4 shared interrupt for 0 & 1 channels

ISR Name	CM4 Hardware interrupt vector	Observations
ISR(FTM_4_CH_2_CH_3_ISR)	124	FTM4 shared interrupt for 2 & 3 channels
ISR(FTM_4_CH_4_CH_5_ISR)	125	FTM4 shared interrupt for 4 & 5 channels
ISR(FTM_4_CH_6_CH_7_ISR)	126	FTM4 shared interrupt for 6 & 7 channels
For FTM5	-	-
ISR(FTM_5_CH_0_CH_1_ISR)	129	FTM5 shared interrupt for 0 & 1 channels
ISR(FTM_5_CH_2_CH_3_ISR)	130	FTM5 shared interrupt for 2 & 3 channels
ISR(FTM_5_CH_4_CH_5_ISR)	131	FTM5 shared interrupt for 4 & 5 channels
ISR(FTM_5_CH_6_CH_7_ISR)	132	FTM5 shared interrupt for 6 & 7 channels
For FTM6	-	-
ISR(FTM_6_CH_0_CH_1_ISR)	135	FTM6 shared interrupt for 0 & 1 channels
ISR(FTM_6_CH_2_CH_3_ISR)	136	FTM6 shared interrupt for 2 & 3 channels
ISR(FTM_6_CH_4_CH_5_ISR)	137	FTM6 shared interrupt for 4 & 5 channels
ISR(FTM_6_CH_6_CH_7_ISR)	138	FTM6 shared interrupt for 6 & 7 channels
For FTM7	-	-
ISR(FTM_7_CH_0_CH_1_ISR)	141	FTM7 shared interrupt for 0 & 1 channels
ISR(FTM_7_CH_2_CH_3_ISR)	142	FTM7 shared interrupt for 2 & 3 channels
ISR(FTM_7_CH_4_CH_5_ISR)	143	FTM7 shared interrupt for 4 & 5 channels
ISR(FTM_7_CH_6_CH_7_ISR)	144	FTM7 shared interrupt for 6 & 7 channels

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

Development Error Tracer:

This module is necessary for enabling Development error detection. The API function used is Det_ReportError(). The activation / deactivation of Development error detection is configurable using the 'OcuDevErrorDetect' configuration parameter.

Diagnostic Event Manager:

This module is necessary for enabling Production error detection. The API function used is Dem_ReportErrorStatus ().

Mcu:

MCU module shall be initialized before using Ocu. This module is required for setting the eMios global Pre-scalar value and clock.

Port:

PORT module shall configure the eMios channels which are used by the Ocu driver.

EcuC:

This module is necessary for handling Postbuild Variant. This module allows users to configure multiple configuration

Configuration dependency to other module: Care must be used not to allocate the same FTM channels to other MCAL drivers (ICU/ GPT/PWM).

5.7 Data Cache Restrictions

None

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

There is no restriction when running from user mode for all OCU IPs. Therefore no further actions are needed in OCU driver.

5.8.2 User Mode configuration in AutosarOS

When User mode is enabled, the driver may has 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,...)
```

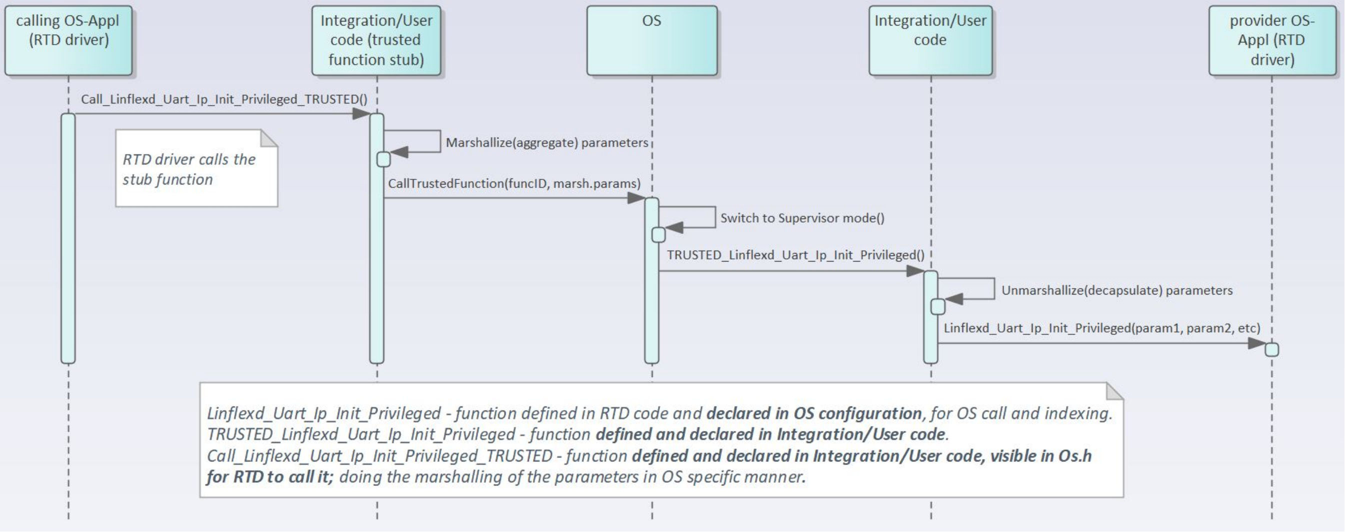
That is the result of macro expansion `OsIf_Trusted_Call` in driver code:

`#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.


calling OS-Appl (RTD driver)
Integration/User code (trusted function stub)
OS
Integration/User code
provider OS-Appl (RTD driver)
Call_Linflexd_Uart_Ip_Init_Privileged_TRUSTED()
Marshallize(aggregate) parameters
CallTrustedFunction(funcID, marsh.params)
Switch to Supervisor mode()
TRUSTED_Linflexd_Uart_Ip_Init_Privileged()
Unmarshallize(decapsulate) parameters
Linflexd_Uart_Ip_Init_Privileged(param1, param2, etc)

RTD driver calls the stub function

Linflexd_Uart_Ip_Init_Privileged - function defined in RTD code and ***declared in OS configuration****, for OS call and indexing.*
TRUSTED_Linflexd_Uart_Ip_Init_Privileged - function ***defined and declared in Integration/User code****.*
Call_Linflexd_Uart_Ip_Init_Privileged_TRUSTED - function ***defined and declared in Integration/User code, visible in Os.h for RTD to call it;*** *doing the marshalling of the parameters in OS specific manner.*

Figure 5.2 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

All S32K1 derivatives will be treated as a single-core device, so this platform does not support multicore feature!

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

None.

6.2 API Requirements

None.

6.3 Calls to Notification Functions, Callbacks, Callouts

The Ocu Driver provides a notification per channel that is called whenever the selected edges are generated. The notifications can be configured as pointers to user defined functions. If notification is not desired for a specific channel then 'NULL_PTR' or 'NULL' shall be configured. The syntax of this function is as follows: void Ocu_Notification↩ _::channel(void) An extern declaration of the notification functions is available in Ocu_PBCfg.c. The notification functions have to be implemented by the user.

Chapter 7

Memory allocation

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

7.1 Sections to be defined in Ocu_MemMap.h

Tables descibe Sections to be defined in Ocu_MemMap.h:

[**Section to be define**]

Section name	Type of section	Description
OCU_START_SEC_CONFIG_DATA_<ALIGNMENT>	Configuration Data	Start of Memory Section for Config Data
OCU_STOP_SEC_CONFIG_DATA_<ALIGNMENT>	Code	Start of memory Section for Code
OCU_START_SEC_CODE	Code	Start of memory Section for Code in Flash.
OCU_STOP_SEC_CODE	Code	Stop of memory Section for Code in Flash.
OCU_START_SEC_RAMCODE	Code	Start of memory Section for Code in Ram.
OCU_STOP_SEC_RAMCODE	Code	Stop of memory Section for Code in Ram.
OCU_START_SEC_VAR_<INIT_POLICY>_<ALIGNMENT>	Variables	Start of memory Section for Variables.
OCU_STOP_SEC_VAR_<INIT_POLICY>_<ALIGNMENT>	Variables	Stop of memory Section for Variables.
OCU_START_SEC_CONST_<ALIGNMENT>	Constant data	Start of memory Section for Constant.
OCU_STOP_SEC_CONST_<ALIGNMENT>	Constant data	Stop of memory Section for Constant.

Which the shortcut '<ALIGNMENT >' means the variable alignment. In order to avoid memory gaps in the allocation variables are allocated according their size. Possible ALIGNMENT postfixes are described in the table at the end of this section. The shortcut '<INIT_POLICY>' means the initialization policy of variables. Possible '<INIT_POLICY>' postfixes are described in the table at the end of this section.

Tables descibe value range of shortcut ALIGMENT, INIT_POLICY:

[**Range of <ALIGNMENT>**]

<ALIGNMENT>	Description
BOOLEAN	Used for variables and constants of size 1 bit
8	Used for variables and constants which have to be aligned to 8 bit. For instance used for variables of size 8 bit or used for composite data types: arrays, structs and unions containing elements of maximum 8 bits
16	Used for variables and constants which have to be aligned to 16 bit. For instance used for variables of size 16 bit or used for composite data types: arrays, structs and unions containing elements of maximum 16 bits
32	Used for variables and constants which have to be aligned to 32 bit. For instance used for variables of size 32 bit or used for composite data types: arrays, structs and unions containing elements of maximum 32 bits
UNSPECIFIED	Used for variables, constants, structure, array and unions when SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. For instance used for variables of unknown size

[**Range of <INIT_POLICY>**]

<INIT_POLICY>	Description
NO-INIT	Used for variables that are never cleared and never initialized by start up code (BSS)
INIT	Used for variables that are initialized with values after every reset

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 OCU 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

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.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C–5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C–Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. ARM, AMBA, ARM Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and Vision are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7, ARM9, ARM11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2022 NXP B.V.