Integration Manual

for S32K1_S32M24X PWM Driver

Document Number: IM2PWMASRR21-11 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 PWM 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
ASM	Assembler
BSMI	Basic Software Make file Interface
CAN	Controller Area Network
C/CPP	C and C++ Source Code
CS	Chip Select
CTU	Cross Trigger Unit
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
VLE	Variable Length Encoding
XML	Extensible Markup Language

2.5 Reference List

#	Title	Version
1	Specification of PWM Driver	AUTOSAR Release R21-11
2	S32K1xx Series Reference Manual	Rev. 14, 09/2021
3	S32M24x Reference Manual	Rev. 2 Draft A, 05/2023
4	S32K116 Mask Set Errata for Mask 0N96V	Rev. 22/OCT/2021
5	S32K118 Mask Set Errata for Mask 0N97V	Rev. 22/OCT/2021
6	S32K142 Mask Set Errata for Mask 0N33V	Rev. 22/OCT/2021
7	S32K144 Mask Set Errata for Mask 0N57U	Rev. 22/OCT/2021
8	S32K144W Mask Set Errata for Mask 0P64A	Rev. 22/OCT/2021
9	S32K146 Mask Set Errata for Mask 0N73V	Rev. 22/OCT/2021
10	S32K148 Mask Set Errata for Mask 0N20V	Rev. 22/OCT/2021
11	S32M244 Mask Set Errata for Mask P64A+P73G	Rev. 0
12	S32M242 Mask Set Errata for Mask N33V+P73G	Rev. 0, 6/2023
13	S32K1xx Data Sheet	Rev. 14, 08/2021
14	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 (if assembler code) and link the PWM driver for S32K1XX and S32M24X micro-controllers. 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.

Pwm Files

- ..\Pwm_TS_T40D2M20I0R0\src\Pwm.c

- ..\Pwm_TS_T40D2M20I0R0\src\Pwm_Ipw.c
- ..\Pwm_TS_T40D2M20I0R0\src\Pwm_Ipw_Notif.c
- ..\Pwm_TS_T40D2M20I0R0\src\Ftm_Pwm_Ip.c
- ..\Pwm_TS_T40D2M20I0R0\src\Ftm_Pwm_Ip_Irq.c
- ..\Pwm_TS_T40D2M20I0R0\include\Pwm.h
- ..\Pwm_TS_T40D2M20I0R0\include\Pwm_EnvCfg.h
- ..\Pwm_TS_T40D2M20I0R0\include\Pwm_Ipw.h
- ..\Pwm_TS_T40D2M20I0R0\include\Pwm_Ipw_Types.h
- ..\Pwm_TS_T40D2M20I0R0\include\Pwm_Ipw_Notif.h
- ..\Pwm_TS_T40D2M20I0R0\include\Ftm_Pwm_Ip.h
- ..\Pwm_TS_T40D2M20I0R0\include\Ftm_Pwm_Ip_Irq.h
- ..\Pwm_TS_T40D2M20I0R0\include\Ftm_Pwm_Ip_Types.h
- ..\Pwm_TS_T40D2M20I0R0\include\Ftm_Pwm_Ip_TrustedFunctions.h
- ..\Pwm_TS_T40D2M20I0R0\include\Ftm_Pwm_Ip_HwAccess.h
- ..\Pwm_TS_T40D2M20I0R0\include\Flexio_Pwm_Ip.h
- ..\Pwm_TS_T40D2M20I0R0\include\Flexio_Pwm_Ip_Irq.h
- ..\Pwm_TS_T40D2M20I0R0\include\Flexio_Pwm_Ip_Types.h
- ..\Pwm_TS_T40D2M20I0R0\include\Flexio_Pwm_Ip_HwAccess.h

Pwm Generated Files

- Pwm_PBcfg.c - For driver compilation, this file should be generated by the user using a configuration tool.
- Ftm_Pwm_Ip_PBcfg.c - For driver compilation, this file should be generated by the user using a configuration tool.
- Ftm_PBcfg.h - For driver compilation, this file should be generated by the user using a configuration tool.
- Ftm_Pwm_Ip_PBcfg.h - For driver compilation, this file should be generated by the user using a configuration tool.
- Pwm_Cfg.h - For driver compilation, this file should be generated by the user using a configuration tool.
- Pwm_Ipw_Cfg.h - For driver compilation, this file should be generated by the user using a configuration tool.
- Ftm_Pwm_Ip_Cfg.h - For driver compilation, this file should be generated by the user using a configuration tool.
- Ftm_Pwm_Ip_CfgDefines.h - For driver compilation, this file should be generated by the user using a configuration tool.
- Flexio_Pwm_Ip_Cfg.h - For driver compilation, this file should be generated by the user using a configuration tool.

- Flexio_Pwm_Ip_CfgDefines.h - For driver compilation, this file should be generated by the user using a configuration tool.
- Flexio_Pwm_Ip_PBcfg.c - For driver compilation, this file should be generated by the user using a configuration tool.
- Flexio_Pwm_Ip_PBcfg.h - For driver compilation, this file should be generated by the user using a configuration tool.

Files from Base common folder

- ..\BaseNXP_TS_T40D2M20I0R0\include\Compiler.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\Compiler_Cfg.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\ComStack_Types.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\MemMap.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\Mcal.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\Platform_Types.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\Std_Types.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\Reg_eSys.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\Soc_Ips.h
- ..\BaseNXP_TS_T40D2M20I0R0\include\SilRegMacros.h

Files from Rte folder:

- ..\Rte_TS_T40D2M20I0R0\include\SchM_Pwm.h

Files from Mcl folder:

- ..\Mcl_TS_T40D2M20I0R0\include\Flexio_Mcl_Ip.h
- ..\Mcl_TS_T40D2M20I0R0\include\Flexio_Mcl_Ip_HwAccess.h
- ..\Mcl_TS_T40D2M20I0R0\include\Flexio_Mcl_Ip_Types.h
- ..\Mcl_TS_T40D2M20I0R0\src\Flexio_Mcl_Ip.c
- ..\Mcl_TS_T40D2M20I0R0\src\Flexio_Mcl_Ip_HwAccess.c
- ..\Mcl_TS_T40D2M20I0R0\src\Flexio_Mcl_Ip_Irq.c

Mcl Generated Files

- Flexio_Mcl_Ip_Cfg.h (For PC Variant) - For driver compilation, this file should be generated by the user using a configuration tool
- Flexio_Mcl_Ip_CfgDefines.h (For PC Variant) - For driver compilation, this file should be generated by the user using a configuration tool

- Flexio_Mcl_Ip_Definitions.h (For PC Variant) - For driver compilation, this file should be generated by the user using a configuration tool
- Flexio_Mcl_Ip_PBcfg.h (For PB Variant) - For driver compilation, this file should be generated by the user using a configuration tool
- Flexio_Mcl_Ip_PBcfg.c (For PB Variant) - For driver compilation, this file should be generated by the user using a configuration tool.

Files from Det folder:

- ..\Det_TS_T40D2M20I0R0\include\Det.h

3.3 Setting up the plugins

The Pwm driver was designed to be configured by using the EB Tresos Studio (version EB tresos Studio 29.0.0 or later.)

All the AUTOSAR RTD drivers for S32K1 were designed to be configured using Tresos Studio configuration and code generation tool from EB tresos Studio 29.0.0.

Location of various files inside the plugin folder is explained below.

- VSMD (Vendor Specific Module Definition) file in EB tresos Studio XDM format:
  - ..\Pwm_TS_T40D2M20I0R0\config\Pwm.xdm
  - ..\Mcu_TS_T40D2M20I0R0\config\Mcu.xdm
  - ..\Mcl_TS_T40D2M20I0R0\config\Mcl.xdm
  - ..\Resource_TS_T40D2M20I0R0\config\Resource.xdm

- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - ..\Pwm_TS_T40D2M20I0R0\autosar\Pwm_<subderivative_name>.epd
  - ..\Mcu_TS_T40D2M20I0R0\autosar\Mcu_<subderivative_name>.epd
  - ..\Mcl_TS_T40D2M20I0R0\autosar\Mcl_<subderivative_name>.epd
  - ..\Resource_TS_T40D2M20I0R0\autosar\Resource_<subderivative_name>.epd

- Code Generation Templates for variant aware parameters:
  - ..\Pwm_TS_T40D2M20I0R0\output\src\Pwm_PBCfg.c
  - ..\Pwm_TS_T40D2M20I0R0\output.h
  - ..\Mcu_TS_T40D2M20I0R0\output.h
  - ..\Mcl_TS_T40D2M20I0R0\output.h

Steps to generate the configuration:

1. Copy the module folders Pwm_TS_T40D2M20I0R0 , Dem_TS_T40D2M20I0R0 , BaseNXP_TS_T40D2↩ M20I0R0 , Resource_TS_T40D2M20I0R0 ,Mcu_TS_T40D2M20I0R0 , Mcl_TS_T40D2M20I0R0, EcuC_↩ TS_T40D2M20I0R0 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

- **RESOURCE** : Resource module is used to select micro-controller's derivatives .
- **MCU** : The MCU driver provides services for basic micro-controller initialization, power down functionality, reset and microcontroller specific functions required by other RTD software modules. The clocks need to be initialized prior to using the MDL driver.
- **MCL** : The MCL module is required for support for PWM
- **RTE** : The RTE module is required for critical sections
- **DET** : The DET module is used for enabling Development error detection. The API function used is Det_↩ ReportError(). The activation / deactivation of Development error detection is configurable using the 'Mdl↩ DevErrorDetect' configuration parameter

Chapter 4

Function calls to module

4.1 Function Calls during Start-up

This driver does not need OS Support except for ISRs. Hence can be initialized either in STARTUP1 or START↩ UP2 phase of EcuM initialization. This depends on the implementation, desired duration for STARTUP1 & Target hardware design. The API to be called is Pwm_Init(ConfigPtr).

For proper driver usage, prior MCL, MCU and PORT modules initialization should be done.

4.2 Function Calls during Shutdown

During shutdown phase, Pwm_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, Pwm_Init() function may be called but only if during a previous phase Pwm_DeInit() was called. Calling this function depends on the initialization-deinitialization strategy deployed by user.

Chapter 5

Module requirements

- Exclusive areas to be defined in BSW scheduler
- Exclusive areas not available on this platform
- Peripheral Hardware Requirements
- ISR to configure within AutosarOS - dependencies
- ISR Macro
- Other AUTOSAR modules - dependencies
- Data Cache Restrictions
- User Mode support
- Multi-core support

5.1 Exclusive areas to be defined in BSW scheduler

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

Exclusive Areas are used in High level driver layer (HLD)

PWM_EXCLUSIVE_AREA_00 is used in function Pwm_SetDutyCycle to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_33 is used in function Pwm_SetDutyCycle to protect the updates for:

- FLEXIO_TIMCTL_ADDR32 register
- FLEXIO_PINOUTE_ADDR32 register

PWM_EXCLUSIVE_AREA_00 is used in function Pwm_SetDutyCycle_NoUpdate to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_01 is used in function Pwm_SetPeriodAndDuty to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_CSC_ADDR32
- FTM_SC_ADDR32
- FTM_MOD_ADDR32

PWM_EXCLUSIVE_AREA_33 is used in function Pwm_SetPeriodAndDuty to protect the updates for:

- FLEXIO_TIMCTL_ADDR32 register
- FLEXIO_PINOUTE_ADDR32 register

PWM_EXCLUSIVE_AREA_01 is used in function Pwm_SetPeriodAndDuty_NoUpdate to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_CSC_ADDR32
- FTM_SC_ADDR32
- FTM_MOD_ADDR32

PWM_EXCLUSIVE_AREA_03 is used in function Pwm_GetOutputState to protect the updates for:

- FTM_CSC_ADDR32

PWM_EXCLUSIVE_AREA_04 is used in function Pwm_DisableNotification to protect the updates for:

- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_34 is used in function Pwm_DisableNotification to protect the updates for:

- FLEXIO_PINREN_ADDR32 register
- FLEXIO_PINIEN_ADDR32 register

PWM_EXCLUSIVE_AREA_05 is used in function Pwm_EnableNotification to protect the updates for:

- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_34 is used in function Pwm_EnableNotification to protect the updates for:

- FLEXIO_PINREN_ADDR32 register
- FLEXIO_PINIEN_ADDR32 register

PWM_EXCLUSIVE_AREA_07 is used in function Pwm_MaskOutputs to protect the updates for:

- FTM_SYNC_ADDR32
- FTM_OUTMASK_ADDR32

PWM_EXCLUSIVE_AREA_08 is used in function Pwm_UnMaskOutputs to protect the updates for:

- FTM_SYNC_ADDR32
- FTM_OUTMASK_ADDR32

PWM_EXCLUSIVE_AREA_09 is used in function Pwm_SetPhaseShift to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_MOD_ADDR32

PWM_EXCLUSIVE_AREA_09 is used in function Pwm_SetPhaseShift_NoUpdate to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_MOD_ADDR32

PWM_EXCLUSIVE_AREA_10 is used in function Pwm_DisableTrigger to protect the updates for:

- FTM_EXTTRIG_ADDR32

PWM_EXCLUSIVE_AREA_11 is used in function Pwm_EnableTrigger to protect the updates for:

- FTM_EXTTRIG_ADDR32

PWM_EXCLUSIVE_AREA_12 is used in function Pwm_SyncUpdate to protect the updates for:

- FTM_SYNC_ADDR32

PWM_EXCLUSIVE_AREA_13 is used in function Pwm_SetChannelDeadTime to protect the updates for:

- FTM_DEADTIME_ADDR32

Exclusive Areas are implemented in Low level driver layer (IPL)

PWM_EXCLUSIVE_AREA_00 is used in function Ftm_Pwm_Ip_Init to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_00 is used in function Ftm_Pwm_Ip_UpdatePwmPeriodAndDuty to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_01 is used in function Ftm_Pwm_Ip_UpdatePwmPeriodAndDuty to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_CSC_ADDR32
- FTM_SC_ADDR32

- FTM_MOD_ADDR32 **PWM_EXCLUSIVE_AREA_03** is used in function Ftm_Pwm_Ip_Get↩ OutputState to protect the updates for:
- FTM_CSC_ADDR32

PWM_EXCLUSIVE_AREA_04 is used in function Ftm_Pwm_Ip_DisableNotification to protect the updates for:

- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_05 is used in function Ftm_Pwm_Ip_EnableNotification to protect the updates for:

- FTM_CSC_ADDR32
- FTM_SC_ADDR32

PWM_EXCLUSIVE_AREA_07 is used in function Ftm_Pwm_Ip_MaskOutputChannels to protect the updates for:

- FTM_SYNC_ADDR32
- FTM_OUTMASK_ADDR32

PWM_EXCLUSIVE_AREA_08 is used in function Ftm_Pwm_Ip_UnMaskOutputChannels to protect the updates for:

- FTM_SYNC_ADDR32
- FTM_OUTMASK_ADDR32

PWM_EXCLUSIVE_AREA_09 is used in function Ftm_Pwm_Ip_SetPhaseShift to protect the updates for:

- FTM_SWOCTRL_ADDR32
- FTM_SYNC_ADDR32
- FTM_MOD_ADDR32

PWM_EXCLUSIVE_AREA_10 is used in function Ftm_Pwm_Ip_DisableTrigger to protect the updates for:

- FTM_EXTTRIG_ADDR32

PWM_EXCLUSIVE_AREA_11 is used in function Ftm_Pwm_Ip_EnableTrigger to protect the updates for:

- FTM_EXTTRIG_ADDR32

PWM_EXCLUSIVE_AREA_12 is used in function Ftm_Pwm_Ip_SyncUpdate to protect the updates for:

- FTM_SYNC_ADDR32

PWM_EXCLUSIVE_AREA_13 is used in function Ftm_Pwm_Ip_SetChannelDeadTime to protect the updates for:

- FTM_DEADTIME_ADDR32

PWM_EXCLUSIVE_AREA_31 is used in function Flexio_Pwm_Ip_UpdateClockPrescaler to protect the updates for:

- FLEXIO_TIMCFG_ADDR32 register

PWM_EXCLUSIVE_AREA_32 is used in function Flexio_Pwm_Ip_ForceOuputLevel to protect the updates for:

- FLEXIO_TIMCTL_ADDR32 register
- FLEXIO_PINOUTE_ADDR32 register

PWM_EXCLUSIVE_AREA_32 is used in function Flexio_Pwm_Ip_InitChannel to protect the updates for:

- FLEXIO_TIMCTL_ADDR32 register
- FLEXIO_PINOUTE_ADDR32 register

PWM_EXCLUSIVE_AREA_33 is used in function Flexio_Pwm_Ip_UpdatePeriodDuty to protect the updates for:

- FLEXIO_TIMCTL_ADDR32 register
- FLEXIO_PINOUTE_ADDR32 register

PWM_EXCLUSIVE_AREA_34 is used in function Flexio_Pwm_Ip_UpdateInterruptMode to protect the updates for:

- FLEXIO_PINREN_ADDR32 register
- FLEXIO_PINIEN_ADDR32 register

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.

Exclusive Area Matrix

Exclusive Area ID	PWM_EXCLUSIVE_AREA_00	PWM_EXCLUSIVE_AREA_01	PWM_EXCLUSIVE_AREA_03	PWM_EXCLUSIVE_AREA_04	PWM_EXCLUSIVE_AREA_05	PWM_EXCLUSIVE_AREA_07	PWM_EXCLUSIVE_AREA_08	PWM_EXCLUSIVE_AREA_09	PWM_EXCLUSIVE_AREA_10	PWM_EXCLUSIVE_AREA_11	PWM_EXCLUSIVE_AREA_12	PWM_EXCLUSIVE_AREA_13	PWM_EXCLUSIVE_AREA_33	PWM_EXCLUSIVE_AREA_34	PWM_EXCLUSIVE_AREA_32	PWM_EXCLUSIVE_AREA_31
PWM_EXCLUSIVE_AREA_00	X	X	X	X	X	X	X	X			X					
PWM_EXCLUSIVE_AREA_01	X	X	X	X	X	X	X	X			X					
PWM_EXCLUSIVE_AREA_03	X	X	X	X	X											
PWM_EXCLUSIVE_AREA_04	X	X	X	X	X											
PWM_EXCLUSIVE_AREA_05	X	X	X	X	X											
PWM_EXCLUSIVE_AREA_07	X	X				X	X	X			X					
PWM_EXCLUSIVE_AREA_08	X	X				X	X	X			X					
PWM_EXCLUSIVE_AREA_09	X	X				X	X	X			X					
PWM_EXCLUSIVE_AREA_10									X	X						
PWM_EXCLUSIVE_AREA_11									X	X						
PWM_EXCLUSIVE_AREA_12	X	X				X	X	X			X					
PWM_EXCLUSIVE_AREA_13												X				
PWM_EXCLUSIVE_AREA_31																X
PWM_EXCLUSIVE_AREA_32													X		X	
PWM_EXCLUSIVE_AREA_33													X		X	
PWM_EXCLUSIVE_AREA_34														X		

Figure 5.1 Exclusive Areas form.

5.2 Exclusive areas not available on this platform

List of exclusive areas which are not available on this platform (or blank if they're all available).

-___PWM_EXCLUSIVE_AREA_31___ is not available -___PWM_EXCLUSIVE_AREA_32___ is not available

5.3 Peripheral Hardware Requirements

Refer section **Hardware Resources** in User Manual for more details.

5.4 ISR to configure within AutosarOS - dependencies

The following ISR's are used by the Pwm driver:

PWM ISR's S32K14X and S32M244

ISR Name	CM4 Hardware interrupt vector
ISR(MCL_FLEXIO_ISR)	69
ISR(FTM_0_CH_0_CH_1_ISR)	115
ISR(FTM_0_CH_2_CH_3_ISR)	116
ISR(FTM_0_CH_4_CH_5_ISR)	117
ISR(FTM_0_CH_6_CH_7_ISR)	118
ISR(FTM_0_FAULT_ISR)	119
ISR(FTM_0_OVF_RELOAD_ISR)	120
ISR(FTM_1_CH_0_CH_1_ISR)	121
ISR(FTM_1_CH_2_CH_3_ISR)	122
ISR(FTM_1_CH_4_CH_5_ISR)	123
ISR(FTM_1_CH_6_CH_7_ISR)	124
ISR(FTM_1_FAULT_ISR)	125
ISR(FTM_1_OVF_RELOAD_ISR)	126
ISR(FTM_2_CH_0_CH_1_ISR)	127
ISR(FTM_2_CH_2_CH_3_ISR)	128
ISR(FTM_2_CH_4_CH_5_ISR)	129
ISR(FTM_2_CH_6_CH_7_ISR)	130
ISR(FTM_2_FAULT_ISR)	131
ISR(FTM_2_OVF_RELOAD_ISR)	132
ISR(FTM_3_CH_0_CH_1_ISR)	133
ISR(FTM_3_CH_2_CH_3_ISR)	134
ISR(FTM_3_CH_4_CH_5_ISR)	135
ISR(FTM_3_CH_6_CH_7_ISR)	136
ISR(FTM_3_FAULT_ISR)	137

ISR Name	CM4 Hardware interrupt vector
ISR(FTM_3_OVF_RELOAD_ISR)	138
ISR(FTM_4_CH_0_CH_1_ISR)	139
ISR(FTM_4_CH_2_CH_3_ISR)	140
ISR(FTM_4_CH_4_CH_5_ISR)	141
ISR(FTM_4_CH_6_CH_7_ISR)	142
ISR(FTM_4_FAULT_ISR)	143
ISR(FTM_4_OVF_RELOAD_ISR)	144
ISR(FTM_5_CH_0_CH_1_ISR)	145
ISR(FTM_5_CH_2_CH_3_ISR)	146
ISR(FTM_5_CH_4_CH_5_ISR)	147
ISR(FTM_5_CH_6_CH_7_ISR)	148
ISR(FTM_5_FAULT_ISR)	149
ISR(FTM_5_OVF_RELOAD_ISR)	150
ISR(FTM_6_CH_0_CH_1_ISR)	151
ISR(FTM_6_CH_2_CH_3_ISR)	152
ISR(FTM_6_CH_4_CH_5_ISR)	153
ISR(FTM_6_CH_6_CH_7_ISR)	154
ISR(FTM_6_FAULT_ISR)	155
ISR(FTM_6_OVF_RELOAD_ISR)	156
ISR(FTM_7_CH_0_CH_1_ISR)	157
ISR(FTM_7_CH_2_CH_3_ISR)	158
ISR(FTM_7_CH_4_CH_5_ISR)	159
ISR(FTM_7_CH_6_CH_7_ISR)	160
ISR(FTM_7_FAULT_ISR)	161
ISR(FTM_7_OVF_RELOAD_ISR)	162

PWM ISR's S32K11X

ISR Name	CM4 Hardware interrupt vector
ISR(MCL_FLEXIO_ISR)	25
ISR(FTM_0_ISR)	12
ISR(FTM_0_FAULT_ISR)	13
ISR(FTM_0_OVF_RELOAD_ISR)	14
ISR(FTM_1_ISR)	15
ISR(FTM_1_FAULT_ISR)	16
ISR(FTM_1_OVF_RELOAD_ISR)	17

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

'PwmDevErrorDetect' configuration parameter.

Mcl:

MCL module shall be initialized before using Pwm driver.

Mcu:

MCU module shall be initialized before using Pwm driver.

Port:

PORT module shall configure the Ftm channels which are used by the Pwm driver.

ECUC:

ECUC is required for configuring the variant handling in Tresos.

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

The PWM driver can be run in 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.

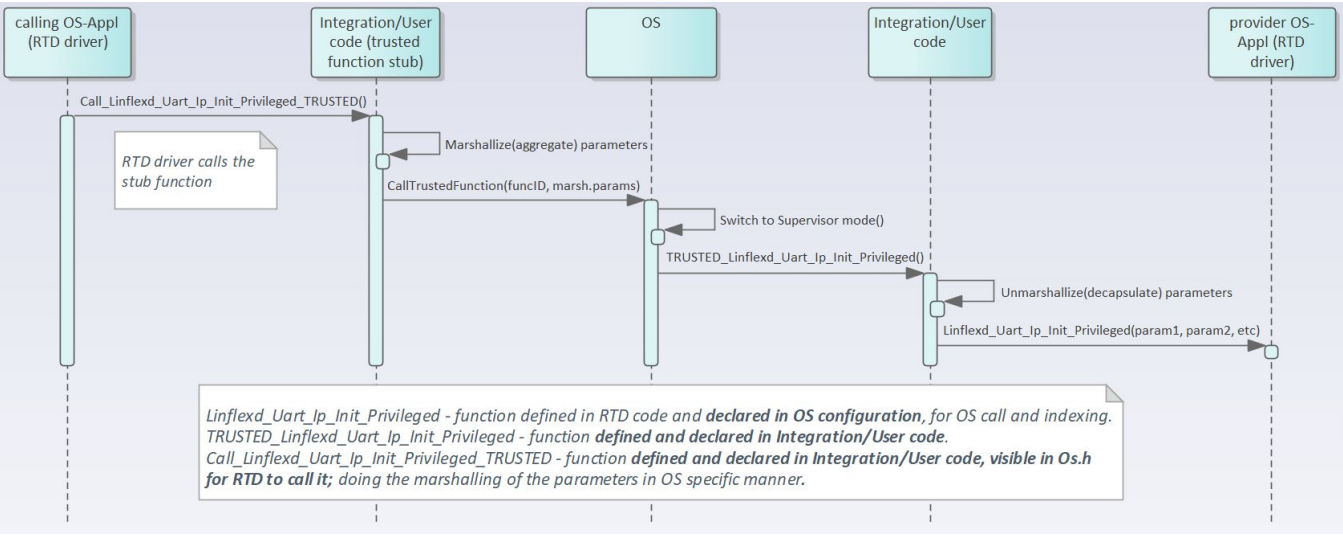

Figure 5.2 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multi-core support

All S32K1 and S32M24 derivatives will be treated as a single-core device, so this platform does not support multi-core feature!

Chapter 6

Main API Requirements

6.1 Main function calls within BSW scheduler

None.

6.2 API Requirements

None.

6.3 Calls to Notification Functions, Callbacks, Callouts

Call-back Notifications:

None.

User Notification:

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.

An extern declaration of the notification functions is generated in Pwm_PBCfg.c file. The notification functions have to be implemented by the user.

Chapter 7

Memory allocation

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

7.1 Sections to be defined in Pwm_MemMap.h

Section name	Type of section	Description
PWM_START_SEC_CONFIG_DATA↩ _<ALIGNMENT>	Configuration Data	Start of Memory Section for Config Data
PWM_STOP_SEC_CONFIG_DATA_↩ <ALIGNMENT>	Configuration Data	End of Memory Section for Config Data
PWM_START_SEC_CODE	Code	Start of memory Section for Code in Flash.
PWM_STOP_SEC_CODE	Code	End of memory Section for Code in Flash.
PWM_START_SEC_VAR_<INIT_P↩ OLICY>_<ALIGNMENT>	Variables	Start of memory Section for Variables.
PWM_STOP_SEC_VAR_<INIT_PO↩ LICY>_<ALIGNMENT>	Variables	End of memory Section for Variables.

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.

<ALIGMENT>	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

<ALIGMENT>	Description
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

<INIT_POLICY>	Description
CLEARED	Used for variables that are cleared to zero by start-up code
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 PWM driver into the application.

External Assumption Req ID	External Assumption Text
SWS_Pwm_00001	The Pwm SWS does not cover PWM emulation on general purpose I/O.
SWS_Pwm_00089	The Pwm module's user shall ensure the integrity if several function calls are made during run time in different tasks or ISRs for the same PWM channel. Note: Out of scope sMcal
SWS_Pwm_00093	The users of the Pwm module shall not call the function Pwm_Init during a running operation. Note: Out of scope sMcal
SWS_Pwm_00116	The Pwm module's environment shall not call any function of the Pwm module before having called Pwm_Init. . Note: Out of scope sMcal
EA_RTD_00071	If interrupts are locked, a centralized function pair to lock and unlock interrupts shall be used.
EA_RTD_00073	When PWM read-back is deployed using ICU, in order to perform the PWM readback, the integrator shall use the following configuration: a given PWM channel on one PWM Hw module shall be only readback on an ICU channel implemented using another Hw module. Note: **Implementation hints**: the configuration can be realized using the safety PORT module
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

External Assumption Req ID	External Assumption Text
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.