Integration Manual

for S32K3 CRYPTO Driver

Document Number: IM34CRYPTOASR4.4 Rev0000R3.0.0 Rev. 1.0

1 Revision History 2

2 Introduction 3

2.1 Supported Derivatives 3

2.2 Overview 4

2.3 About This Manual 5

2.4 Acronyms and Definitions 6

2.5 Reference List 6

3 Building the driver 8

3.1 Build Options 8

3.1.1 GCC Compiler/Assembler/Linker Options 9

3.1.2 DIAB Compiler/Assembler/Linker Options 11

3.1.3 GHS Compiler/Assembler/Linker Options 13

3.1.4 IAR Compiler/Assembler/Linker Options 15

3.2 Files required for compilation 17

3.3 Setting up the plugins 20

4 Function calls to module 22

4.1 Function Calls during Start-up 22

4.2 Function Calls during Shutdown 22

4.3 Function Calls during Wake-up 22

5 Module requirements 23

5.1 Exclusive areas to be defined in BSW scheduler 23

5.2 Exclusive areas not available on this platform 29

5.3 Peripheral Hardware Requirements 29

5.4 ISR to configure within AutosarOS - dependencies 29

5.5 ISR Macro 30

5.5.1 Without an Operating System 30

5.5.2 With an Operating System 30

5.6 Other AUTOSAR modules - dependencies 30

5.7 Data Cache Restrictions 31

5.8 User Mode support 31

5.8.1 User Mode configuration in the module 31

5.8.2 User Mode configuration in AutosarOS 31

5.9 Multicore support 32

6 Main API Requirements 34

6.1 Main function calls within BSW scheduler 34

6.2 API Requirements 34

6.3 Calls to Notification Functions, Callbacks, Callouts 34

7 Memory allocation **35**

7.1 Sections to be defined in Crypto_MemMap.h 35

7.2 Linker command file 37

8 Integration Steps **38**

9 External assumptions for driver **39**

Chapter 1

Revision History

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

Chapter 2

Introduction

- Supported Derivatives
- Overview
- About This Manual
- Acronyms and Definitions
- Reference List

This Integration Manual describes NXP Semiconductor AUTOSAR Crypto driver for S32K3 platform.

AUTOSAR CRYPTO driver configuration parameters and deviations from the specification are described in CRYPTO Driver chapter of this document. AUTOSAR CRYPTO driver requirements and APIs are described in the AUTOSAR CRYPTO driver software specification document.

2.1 Supported Derivatives

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

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100

- s32k322_mqfp172 / MWCT2D16S_mqfp172
- s32k324_mqfp172 / MWCT2D17S_mqfp172
- s32k324_mapbga257
- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

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

Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power.

2.2 Overview

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

AUTOSAR:

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

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

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

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
CAAM	Cryptographic Acceleration and Assurance Module
CMAC	Cipher-based Message Authentication Code
C/CPP	C and C++ Source Code
DET	Development Error Tracer
ECB	Electronic Code Book (refers to AES-ECB mode)
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECU	Electronic Control Unit
EdDSA	Edwards-curve Digital Signature Algorithm
FLS	Flash
GCM	Galois/Counter Mode (refers to AES-GCM mode)
GMAC	Galois Message Authentication Code
HSE	Hardware Security Engine
MAC	Message Authentication Code
MU	Messaging Unit
N/A	Not Applicable
NVM	Non-Volatile Memory
OID	Object Identifier an encoded identifier of a standardized object commonly used in public key certificates
RAM	Random Access Memory
RNG	Random number generator
ROM	Read-only Memory
RSA	A public-key cryptosystem named after the inventors Mr. Rivest, Mr. Shamir and Mr.Adleman
SHA	Secure Hash Algorithm
SHE	Secure Hardware Extension
TDES	Triple-DES operation

- The term "Application" is used for the software utilizing the Crypto Driver.

2.5 Reference List

#	Title	Version
1	Specification of Crypto Driver	AUTOSAR CP Release 4.↩ 4.0
2	S32K3xx Safety Manual	Rev. 3, Dec 2022
3	S32K39 and S32K37 Safety Manual	Rev. 1 Draft D, 16 Nov 2022
4	S32M276 Safety Manual	Rev. 1, Dec 2022

#	Title	Version
5	S32K3xx Reference Manual	Rev.6, Draft B, 01/2023
6	S32K39 and S32K37 Reference Manual	Rev. 2 Draft A, 11/2022
7	S32M27x Reference Manual	Rev.2, Draft A, — 02/2023
8	S32K3xx Data Sheet	Rev. 6, 11/2022
9	S32K396 Data Sheet	Rev. 1.1 — 08/2022
10	S32M2xx Data Sheet	Rev. 2 RC — 12/2022
11	S32K358_0P14E Mask Set Errata	Rev. 28, 9/2022
12	S32K396_0P40E Mask Set Errata	Rev. DEC2022, 12/2022
13	S32K311_0P98C Mask Set Errata	Rev. 6/March/2023, 3/2023
14	S32K312: Mask Set Errata for Mask 0P09C	Rev. 25/April/2022
15	S32K342: Mask Set Errata for Mask 0P97C	Rev. 10, 11/2022
16	S32K3x4: Mask Set Errata for Mask 0P55A/1P55A	Rev. 14/Oct/2022

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

The RTD driver files are compiled using:

- NXP GCC 10.2.0 20200723 (Build 1728 Revision g5963bc8)
- Wind River Diab Compiler 7.0.4
- Compiler Versions: Green Hills Multi 7.1.6d / Compiler 2021.1.4
- Compiler Versions: IAR ANSI C/C++ Compiler V8.50.10 (safety version)

The compiler, assembler, and linker flags used for building the driver are explained below.

The TS_T40D34M30I0R0 part of the plugin name is composed as follows:

- T = Target_Id (e.g. T40 identifies Cortex-M architecture)
- D = Derivative_Id (e.g. D34 identifies S32K3 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-m7	Targeted ARM processor for which GCC should tune the performance of the code
-mthumb	Generates code that executes in Thumb state
-mlittle-endian	Generate code for a processor running in little-endian mode
-mfpu=fpv5-sp-d16	Specifies the floating-point hardware available on the target
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions
-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.
-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

Compiler Option	Description
-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
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1. For example: Predefine for S32K344 will be -DS32K344.
-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
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩ c under the Platform driver
-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
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT 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=cortexm7	Targeted ARM processor for which GCC should tune the performance of the code
-mfpu=fpv5-sp-d16	Specifies the floating-point hardware available on the target
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions
-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=cortexm7	Targeted ARM processor for which GCC should tune the performance of the code
-mthumb	Generates code that executes in Thumb state
-mfpu=fpv5-sp-d16	Specifies the floating-point hardware available on the target
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions
-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
-specs=nano.specs	Use Newlib nano specs
-specs=nosys.specs	Do not use printf/scanf

3.1.2 DIAB Compiler/Assembler/Linker Options

3.1.2.1 DIAB Compiler Options

Compiler Option	Description
-tARMCORTEXM7MG:simple	Selects target processor (hardware single-precision, software double-precision floating-point)
-mthumb	Selects generating code that executes in Thumb state
-std=c99	Follows the C99 standard for C
-Oz	Like -O2 with further optimizations to reduce code size
-g	Generates DWARF 4.0 debug information
-fstandalone-debug	Emits full debug info for all types used by the program
-Wstrict-prototypes	Warn if a function is declared or defined without specifying the argument types
-Wsign-compare	Produce warnings when comparing signed type with unsigned type
-Wdouble-promotion	Give a warning when a value of type float is implicitly promoted to double
-Wunknown-pragmas	Issues a warning for unknown pragmas
-Wundef	Warns if an undefined identifier is evaluated in an #if directive. Such identifiers are replaced with zero

Compiler Option	Description
-Wextra	Enables some extra warning flags that are not enabled by '-Wall'
-Wall	Enables all of the most useful warnings (for historical reasons this option does not literally enable all warnings)
-pedantic	Emits a warning whenever the standard specified by the -std option requires a diagnostic
-Werror=implicit-function-declaration	Generates an error whenever a function is used before being declared
-fno-common	Compile common globals like normal definitions
-fno-signed-char	Char is unsigned
-fno-trigraphs	Do not process trigraph sequences
-V	Displays the current version number of the tool suite
-c	Stop after assembly and produce an object file for each source file
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1
-DDIAB	Predefine DIAB 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
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩ c under the Platform driver
-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
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode

3.1.2.2 DIAB Assembler Options

Assembler Option	Description
-mthumb	Selects generating code that executes in Thumb state
-Xpreprocess-assembly	Invokes C preprocessor on assembly files before running the assembler
-Xassembly-listing	Produces an .lst assembly listing file
-c	Stop after assembly and produce an object file for each source file
-tARMCORTEXM7MG:simple	Selects target processor (hardware single-precision, software double-precision floating-point)

3.1.2.3 DIAB 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
linker_script_file.dld	Use linker_script_file.dld as the linker script. This script replaces the default linker script (rather than adding to it)
-m30	m2 + m4 + m8 + m16
-Xstack-usage	Gathers and display stack usage at link time
-Xpreprocess-lecl	Perform pre-processing on linker scripts
-Llibrary_path	Points to the libraries location for ARMV7EMMG to be used for linking
-lc	Links with the standard C library
-lm	Links with the math library
-tARMCORTEXM7MG:simple	Selects target processor (hardware single-precision, software double-precision floating-point)

3.1.3 GHS Compiler/Assembler/Linker Options

3.1.3.1 GHS Compiler Options

Compiler Option	Description
-cpu=cortexm7	Selects target processor: Arm Cortex M7
-thumb	Selects generating code that executes in Thumb state
-fpu=vfpv5_d16	Specifies hardware floating-point using the v5 version of the VFP instruction set, with 16 double-precision floating-point registers
-fsingle	Use hardware single-precision, software double-precision FP instructions
-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

Compiler Option	Description
–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
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1. For example: Predefine for S32K344 will be -DS32K344.
-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
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩ c under the Platform driver
-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
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT 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=cortexm7	Selects target processor: Arm Cortex M7
-fpu=vfpv5_d16	Specifies hardware floating-point using the v5 version of the VFP instruction set, with 16 double-precision floating-point registers
-fsingle	Use hardware single-precision, software double-precision FP instructions
-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

3.1.3.3 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 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
-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

3.1.4 IAR Compiler/Assembler/Linker Options

3.1.4.1 IAR Compiler Options

Compiler Option	Description
–cpu Cortex-M7	Targeted ARM processor for which IAR should tune the performance of the code
–cpu_mode thumb	Generates code that executes in Thumb state
–endian little	Generate code for a processor running in little-endian mode
–fpu VFPv5-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant.
-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

Compiler Option	Description
–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
–do_explicit_zero_opt_in_named_sections	Disable the exception for variables in user-named sections, and thus treat explicit initializations to zero as zero initializations, not copy 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
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1. For example: Predefine for S32K344 will be -DS32K344.
-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.
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩c under the Platform driver
-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
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode.

3.1.4.2 IAR Assembler Options

Assembler Option	Description
–cpu Cortex-M7	Targeted ARM processor for which IAR should generate the instruction set
–fpu VFPv5-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant.
–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

3.1.4.3 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-M7	Selects the ARM processor variant to link the application for
–fpu VFPv5-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant.
–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 and link the AUTOSAR Crypto Driver for S32K3 microcontrollers.

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

3.2.0.0.1 CRYPTO Driver Files:

- Crypto_TS_T40D34M30I0R0\include\Crypto.h
- Crypto_TS_T40D34M30I0R0\include\Crypto_ASRExtension.h
- Crypto_TS_T40D34M30I0R0\include\Crypto_Hse.h
- Crypto_TS_T40D34M30I0R0\include\Crypto_Ipw.h
- Crypto_TS_T40D34M30I0R0\include\Crypto_KeyManagement.h
- Crypto_TS_T40D34M30I0R0\include\Crypto_Private.h
- Crypto_TS_T40D34M30I0R0\include\Crypto_Types.h
- Crypto_TS_T40D34M30I0R0\include\Crypto_Util.h
- Crypto_TS_T40D34M30I0R0\include\Hse_Ip.h
- Crypto_TS_T40D34M30I0R0\include\Mu_Ip.h
- Crypto_TS_T40D34M30I0R0\src\Crypto.c
- Crypto_TS_T40D34M30I0R0\src\Crypto_ASRExtension.c
- Crypto_TS_T40D34M30I0R0\src\Crypto_Hse.c
- Crypto_TS_T40D34M30I0R0\src\Crypto_KeyManagement.c
- Crypto_TS_T40D34M30I0R0\src\Crypto_Util.c
- Crypto_TS_T40D34M30I0R0\src\Hse_Ip.c
- Crypto_TS_T40D34M30I0R0\src\Mu_Ip_Irq.c

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

- Hse_Ip_Cfg.h
- Crypto_Cfg.h
- Crypto_Cfg.c

3.2.0.0.3 BASE Files:

- BaseNXP_TS_T40D34M30I0R0\include\Mcal.h
- BaseNXP_TS_T40D34M30I0R0\include\Crypto_MemMap.h
- BaseNXP_TS_T40D34M30I0R0\include\Platform_Types.h
- BaseNXP_TS_T40D34M30I0R0\include\Soc_Ips.h
- BaseNXP_TS_T40D34M30I0R0\include\Std_Types.h
- BaseNXP_TS_T40D34M30I0R0\include\OsIf.h
- BaseNXP_TS_T40D34M30I0R0\header\S32K358_MU.h
- BaseNXP_TS_T40D34M30I0R0\header\S32K39_MU.h
- BaseNXP_TS_T40D34M30I0R0\include\StandardTypes.h
- BaseNXP_TS_T40D34M30I0R0\include\Devassert.h
- BaseNXP_TS_T40D34M30I0R0\generate_PC\include\modules.h

3.2.0.0.4 DET Files:

- Det_TS_T40D34M30I0R0\include\Det.h
- Det_TS_T40D34M30I0R0\src\Det.c

3.2.0.0.5 RTE Files:

- Rte_TS_T40D34M30I0R0\include\SchM_Crypto.h
- Rte_TS_T40D34M30I0R0\src\SchM_Crypto.c

3.2.0.0.6 CRYIF Files:

- CryIf_TS_T40D34M30I0R0\include\CryIf.h
- CryIf_TS_T40D34M30I0R0\include\CryIf_Cbk.h
- CryIf_TS_T40D34M30I0R0\src\CryIf.c

3.2.0.0.7 CSM Files:

- Csm_TS_T40D34M30I0R0\include\Csm_Types.h

3.2.0.0.8 HSE Interface Files:

- hse_b_config.h
- hse_common_types.h
- hse_compiler_abs.h
- hse_compile_defs.h
- hse_defs.h
- hse_gpr_status.h
- hse_interface.h
- hse_keymgmt_common_types.h
- hse_platform.h
- hse_srv_aead.h
- hse_srv_attr.h
- hse_srv_bootdatasig.h
- hse_srv_cmac_with_counter.h
- hse_srv_combined_auth_enc.h
- hse_srv_crc32.h
- hse_srv_firmware_update.h
- hse_srv_hash.h
- hse_srv_ipsec.h
- hse_srv_key_derive.h
- hse_srv_key_generate.h
- hse_srv_key_import_export.h
- hse_srv_key_mgmt_utils.h
- hse_srv_mac.h
- hse_srv_monotonic_cnt.h
- hse_srv_otfad_install.h
- hse_srv_publish_sys_img.h
- hse_srv_random.h
- hse_srv_responses.h

- hse_srv_rsa_cipher.h
- hse_srv_sbaf_update.h
- hse_srv_self_test.h
- hse_srv_she_cmds.h
- hse_srv_sign.h
- hse_srv_siphash.h
- hse_srv_smr_install.h
- hse_srv_sym_cipher.h
- hse_srv_sys_authorization.h
- hse_srv_utils.h
- hse_status_and_errors.h
- hse_target.h
- std_typedefs.h
- hse_srv_msc_key_mgmt.h
- hse_srv_tmu_reg_config.h

When compiling for a S32K311 derivative, the HSE files above should be retrieved from the release s32k3x1_hse↩_fw_0.12.0_2.14.0.
When compiling for a S32K312 derivative, the HSE files above should be retrieved from the release s32k3x2_hse↩_fw_0.13.0_2.6.0.
When compiling for a S32K342 derivative, the HSE files above should be retrieved from the release s32k3x2_hse↩_fw_0.13.0_2.8.0.
When compiling for a S32K344 derivative, the HSE files above should be retrieved from the release s32k3x4_hse↩_fw_0.5.0_2.1.0.
When compiling for a S32K358 derivative, the HSE files above should be retrieved from the release s32k3x8_hse↩_fw_0.14.0_2.12.0.
When compiling for a S32K396 derivative, the HSE files above should be retrieved from the release s32k3x6_hse↩_fw_0.15.0_2.7.0.
The release s32k3x8_hse_fw_0.14.0_2.12.0 contains the HSE files (firmware image + interface) that were used for testing the Crypto driver in this RTD release, on the S32K3x8 derivatives.
The release s32k3x6_hse_fw_0.15.0_2.7.0 contains the HSE files (firmware image + interface) that were used for testing the Crypto driver in this RTD release, on the S32K3x6 derivatives.
The release s32k3x4_hse_fw_0.5.0_2.1.0 contains the HSE files (firmware image + interface) that were used for testing the Crypto driver in this RTD release, on the S32K3x4 derivatives.
The releases s32k3x2_hse_fw_0.13.0_2.8.0 and s32k3x2_hse_fw_0.13.0_2.6.0 contains the HSE files (firmware image + interface) that were used for testing the Crypto driver in this RTD release, on the S32K3x2 derivatives.
The release s32k3x1_hse_fw_0.12.0_2.14.0 contains the HSE files (firmware image + interface) that were used for testing the Crypto driver in this RTD release, on the S32K3x1 derivatives.

3.3 Setting up the plugins

The Crypto Driver was designed to be configured by using the EB Tresos Studio (version 29.0.0 b220329-0119 or later)

3.3.0.0.1 Location of various files inside the CRYPTO module folder:

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
  - Crypto_TS_T40D34M30I0R0\config\Crypto.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - Crypto_TS_T40D34M30I0R0\autosar\Crypto_<subderivative_name>.epd
- Code Generation Templates :
  - Crypto_TS_T40D34M30I0R0\generate_PC\src\Crypto_Cfg.c
  - Crypto_TS_T40D34M30I0R0\generate_PC\include\Crypto_Cfg.h
  - Crypto_TS_T40D34M30I0R0\generate_PC\include\Hse_Ip_Cfg.h

3.3.0.0.2 Steps to generate the configuration:

1. Copy the following module folders into the Tresos plugins folder:
   - Crypto_TS_T40D34M30I0R0
   - BaseNXP_TS_T40D34M30I0R0
   - Det_TS_T40D34M30I0R0
   - EcuC_TS_T40D34M30I0R0
   - Rte_TS_T40D34M30I0R0
   - Resource_TS_T40D34M30I0R0
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

Chapter 4

Function calls to module

- Function Calls during Start-up
- Function Calls during Shutdown
- Function Calls during Wake-up

4.1 Function Calls during Start-up

CRYPTO driver shall be initialized during STARTUP phase of EcuM initialization. The API member to be called to accomplish this is Crypto_Init.

The MCU module should be initialized before CRYPTO module is initialized.

4.2 Function Calls during Shutdown

None.

4.3 Function Calls during Wake-up

None.

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
- Multicore support

5.1 Exclusive areas to be defined in BSW scheduler

In the current implementation, CRYPTO driver is using the services of Run-TimeEnvironment (RTE) for entering and exiting the critical regions. RTE implementation is done by the integrators of the MCAL using OS or non-OS services. For testing the CRYPTO driver, stubs are used for RTE. The following critical regions are used in the CRYPTO driver:

Exclusive Areas implemented in High level driver layer (HLD)

CRYPTO_EXCLUSIVE_AREA_00 is used in function Crypto_ProcessJob to protect the Crypto_aObjectQueueList[ObjIndex].u32HeadOfQueuedJobs, Crypto_aObjectQueueList[ObjIndex].u32HeadOfFreeJobs, Crypto_aDriverObjectList[ObjectIdx].pQueuedJobs[IdxQueueElementJob] global variables from read/write operation.

CRYPTO_EXCLUSIVE_AREA_01 is used in function ISR(Mu_Ip_Mu0_OredRx_Isr) to protect the Crypto_aObjectQueueList[ObjIndex].u32HeadOfQueuedJobs, Crypto_aObjectQueueList[ObjIndex].u32HeadOfFreeJobs, Crypto_aDriverObjectList[ObjectIdx].pQueuedJobs[IdxQueueElementJob] global variables from read/write operation.

CRYPTO_EXCLUSIVE_AREA_01 is used in function ISR(Mu_Ip_Mu1_OredRx_Isr) to protect the Crypto_aObjectQueueList[ObjIndex].u32HeadOfQueuedJobs, Crypto_aObjectQueueList[ObjIndex].u32HeadOfFreeJobs, Crypto_aDriverObjectList[ObjectIdx].pQueuedJobs[IdxQueueElementJob] global variables from read/write operation.

CRYPTO_EXCLUSIVE_AREA_01 is used in function ISR(Mu_Ip_Mu2_OredRx_Isr) to protect the Crypto_aObjectQueueList[ObjIndex].u32HeadOfQueuedJobs, Crypto_aObjectQueueList[ObjIndex].u32HeadOfFreeJobs, Crypto_aDriverObjectList[ObjectIdx].pQueuedJobs[IdxQueueElementJob] global variables from read/write operation.

CRYPTO_EXCLUSIVE_AREA_01 is used in function ISR(Mu_Ip_Mu3_OredRx_Isr) to protect the Crypto_aObjectQueueList[ObjIndex].u32HeadOfQueuedJobs, Crypto_aObjectQueueList[ObjIndex].u32HeadOfFreeJobs, Crypto_aDriverObjectList[ObjectIdx].pQueuedJobs[IdxQueueElementJob] global variables from read/write operation.

CRYPTO_EXCLUSIVE_AREA_01 is used in function Crypto_MainFunction to protect the Crypto_aObjectQueueList[ObjIndex].u32HeadOfQueuedJobs, Crypto_aObjectQueueList[ObjIndex].u32HeadOfFreeJobs, Crypto_aDriverObjectList[ObjectIdx].pQueuedJobs[IdxQueueElementJob] global variables from read/write operation.

CRYPTO_EXCLUSIVE_AREA_02 is used in function Crypto_CancelJob to protect the Crypto_aObjectQueueList[ObjIndex].u32HeadOfQueuedJobs, Crypto_aObjectQueueList[ObjIndex].u32HeadOfFreeJobs, Crypto_aDriverObjectList[ObjectIdx].pQueuedJobs[IdxQueueElementJob] global variables from read/write operation.

CRYPTO_EXCLUSIVE_AREA_03 is used in function Crypto_MainFunction to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_03 is used in function ISR(Mu_Ip_Mu0_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_03 is used in function ISR(Mu_Ip_Mu1_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_03 is used in function ISR(Mu_Ip_Mu2_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_03 is used in function ISR(Mu_Ip_Mu3_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_03 is used in function Crypto_ProcessJob to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_04 is used in function Crypto_MainFunction to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_04 is used in function ISR(Mu_Ip_Mu0_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_04 is used in function ISR(Mu_Ip_Mu1_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_04 is used in function ISR(Mu_Ip_Mu2_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_04 is used in function ISR(Mu_Ip_Mu3_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_04 is used in function Crypto_ProcessJob to protect the Crypto_aCrypto↩HseMuState[MuInstance].u8StreamBusyBitMap global variable from read/modify/write operation.

CRYPTO_EXCLUSIVE_AREA_05 is used in function Crypto_CancelJob to ensure the send message function should complete the request to HSE and not be interrupted by TMU.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_MainFunction to protect the Crypto_a↩CryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_CopyKeyElements to protect the Crypto↩_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Exts_FormatKeyCatalogs to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Exts_MPCompression to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Exts_SHE_BootFailure to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Exts_SHE_BootOk to protect the Crypto↩_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Exts_SHE_DebugAuth to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Exts_SHE_DebugChal to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Exts_SHE_GetId to protect the Crypto↩_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_Init to protect the Crypto_aCryptoHse↩MuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyCopy to protect the Crypto_aCrypto↩HseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyDerive to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyElementCopy to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyElementCopyPartial to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyElementGet to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyElementSet to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyExchangeCalcPubVal to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyExchangeCalcSecret to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeyGenerate to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_KeySetValid to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Crypto_ProcessJob to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function ISR(Mu_Ip_Mu0_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function ISR(Mu_Ip_Mu1_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function ISR(Mu_Ip_Mu2_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function ISR(Mu_Ip_Mu3_OredRx_Isr) to protect the Crypto_aCryptoHseMuState[MuInstance].Hse_Ip_MuState.abChannelAllocated[Channel] global variable from read/write operation in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_ProcessJob to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyElementGet to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_MainFunction to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyGenerate to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyDerive to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyExchangeCalcSecret to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyElementSet to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyElementCopy to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_CancelJob to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyExchangeCalcPubVal to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_CopyKeyElements to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyElementCopyPartial to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeyCopy to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_KeySetValid to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Init to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Exts_FormatKeyCatalogs to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Exts_SHE_BootFailure to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Exts_SHE_BootOk to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Exts_SHE_GetId to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Exts_SHE_DebugChal to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Exts_SHE_DebugAuth to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Crypto_Exts_MPCompression to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function ISR(Mu_Ip_Mu0_OredRx_Isr) to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function ISR(Mu_Ip_Mu1_OredRx_Isr) to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function ISR(Mu_Ip_Mu2_OredRx_Isr) to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function ISR(Mu_Ip_Mu3_OredRx_Isr) to protect the Receive Control Register (RCR) from read/modify/write operation in Hse_Ip_ServiceRequest.

Exclusive Areas implemented in Low level driver layer (IPL)

CRYPTO_EXCLUSIVE_AREA_10 is used in function Hse_Ip_GetFreeChannel to protect the updates for Hse_Ip_apMuState[MuInstance]->abChannelAllocated[MuChannel] global variable.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Hse_Ip_MainFunction to protect the updates for Hse_Ip_apMuState[MuInstance]->abChannelAllocated[MuChannel] global variable in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_10 is used in function Hse_Ip_RxIrqHandler to protect the updates for Hse_Ip_apMuState[MuInstance]->abChannelAllocated[MuChannel] global variable in Hse_Ip_GetFreeChannel.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Hse_Ip_ServiceRequest to protect the updates for Receive Control Register (RCR).

CRYPTO_EXCLUSIVE_AREA_11 is used in function Hse_Ip_MainFunction to protect the updates for Receive Control Register (RCR) in Hse_Ip_ServiceRequest.

CRYPTO_EXCLUSIVE_AREA_11 is used in function Hse_Ip_RxIrqHandler to protect the updates for Receive Control Register (RCR) in Hse_Ip_ServiceRequest.

Critical Region Exclusive Matrix
Below is the table depicting the exclusivity between different critical region IDs from the CRYPTO driver. If there is an “X” in the table, it means that those 2 critical regions cannot interrupt each other.

#	CRYPTO_EA_00	CRYPTO_EA_01	CRYPTO_EA_02	CRYPTO_EA_03	CRYPTO_EA_04	CRYPTO_EA_05	CRYPTO_EA_10	CRYPTO_EA_11
CRYPTO_EA_00	x	x	x					
CRYPTO_EA_01	x	x	x					
CRYPTO_EA_02	x	x	x					
CRYPTO_EA_03				x	x			
CRYPTO_EA_04				x	x			
CRYPTO_EA_05						x		
CRYPTO_EA_10							x	
CRYPTO_EA_11								x

Note
CRYPTO_EA_xx means CRYPTO_EXCLUSIVE_AREA_xx

5.2 Exclusive areas not available on this platform

CRYPTO_EXCLUSIVE_AREA_12 is not available on this platform.

5.3 Peripheral Hardware Requirements

For S32K3 controllers, the CRYPTO driver functionality is provided with the help of the MU module, which enables communication with HSE Firmware. The MU is a NXP IP which is present on this platform in 2 instances, each instance having a number of 4 channels.

5.4 ISR to configure within AutosarOS - dependencies

The following ISRs are used by the Crypto Driver when interrupts are switched on (the driver can also be run in polling mode):

ISR Name	NVIC Interrupt ID
Mu_Ip_Mu0_OredRx_Isr	193
Mu_Ip_Mu0_OredGP_Isr	194
Mu_Ip_Mu1_OredRx_Isr	196
Mu_Ip_Mu1_OredGP_Isr	197

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

- **BASENXP**: Contains the common files/definitions needed by all RTD modules.
- **CRYIF**: Is the interface to the services of the Crypto Driver(s) for the upper service layer.
- **CSM**: Provides synchronous or asynchronous services to enable a unique access to basic cryptographic functionalities for all software modules.
- **DET**: Is required for implementing the development error detection (parameters out of range, null pointers, etc). The activation / deactivation of development error detection is configurable using the `_CryptoDevErrorDetect_` configuration parameter.
- **RTE**: Is needed for implementing data consistency of exclusive areas that are used by Crypto module.
- **ECUC**: The ECUC module is used for ECU configuration. RTD modules need ECUC to retrieve the variant information.
- **OS**: The OS module is used for OS configuration. RTD modules need OS to define a mapping between EcuC partitions and EcuC core ids when multicore support is enabled.
- **RESOURCE**: The RESOURCE module is used to select microcontroller's derivatives.

5.7 Data Cache Restrictions

To avoid possible coherency issues when D-CACHE is enabled, the user shall ensure that the buffers used as input and output parameters to driver's APIs are allocated in the NON_CACHEABLE area (by means of Crypto_MemMap).

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 Crypto driver can be run in user mode if the following steps are performed:

- Enable CryptoEnableUserModeSupport from the configuration.
- The Crypto driver can be run in user mode, no special measures needed.

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,...)
```

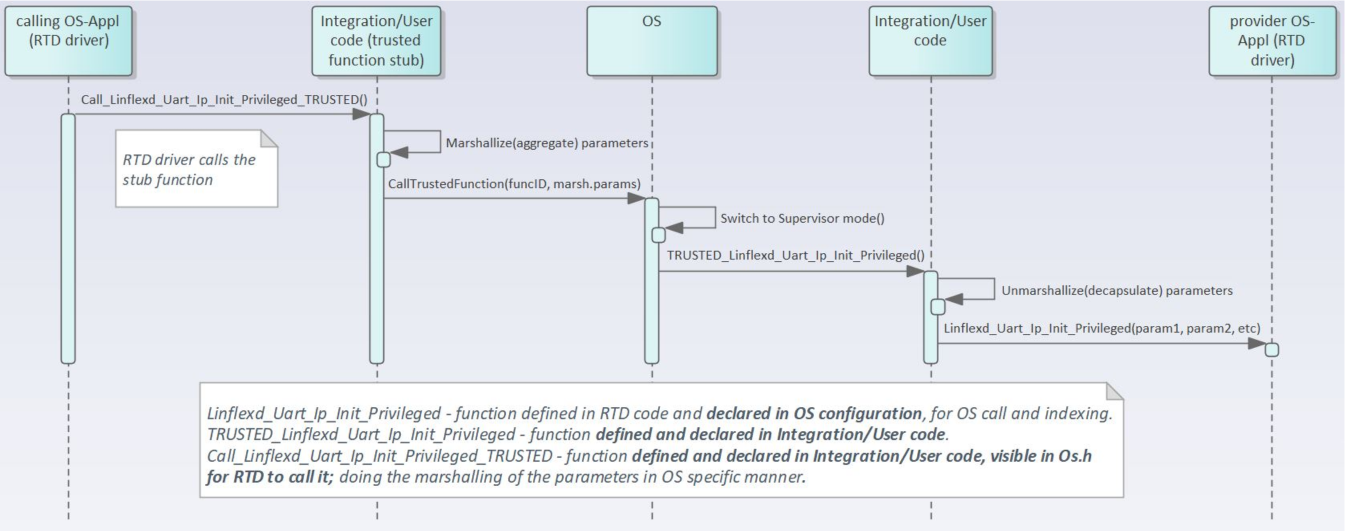
That is the result of macro expansion `OsIf_Trusted_Call` in driver code:

`#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()
RTD driver calls the stub function
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)

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

5.9 Multicore support

The **Crypto** driver implements the **Autosar 4.4 MCAL Multicore Distribution** according to type II, in which the mappable element is set to Crypto Driver Object. For additional details, please refer to **AUTOSAR_EXP↩ _BSWDistributionGuide**.

The **Crypto** driver and the mappable elements can be allocated to zero, one or several ECUC partitions, by means of **CryptoEcucPartionRef**. If the **Crypto** is mapped to zero ECUC partitions, the **Crypto** behavior reverts to single-core implementation, similar to previous Autosar versions. If the **Crypto** is mapped to one or more ECUC partitions, the **Crypto** enforces the following multi-core assumptions:

1. The **Crypto** driver assumes there is a single EcucPartition allocated per core. Internally, the module will use the Core ID returned by GetCoreID API to reference the appropriate global data and configuration elements.

2. The **Crypto** driver assumes the EcucCoreIDs are defined in a compact/consecutive order, starting from zero. The rationale is that the number of EcucPartitions is used for dimensioning the **Crypto** internal variables and the EcucCoreIDs are used for indexing those variables. (AR-86601 Zero based and dense IDs for OS-Cores and OSApplications)

3. The **Crypto** driver assumes that initialization is performed on each core, Crypto_Init() is called separately for each core.

4. The **Crypto** driver will check upon each API call if the requested resource is configured to be available on the current core, if DET error reporting is enabled.

5. The **Crypto** driver requires that all variables in NonCacheable MemMap sections be allocated accordingly, to avoid data corruption in multicore context.

6. The **Crypto** driver assumes that RTE module implements the EXCLUSIVE AREAS to be core-aware only. The rationale is that the module implementation ensures data integrity by separating the mappable elements for different cores already, thus implementing the EXCLUSIVE AREAS in a blocking manner (ex: spin-lock) on a multicore scope, might affect the performance of the drivers on the two cores, although they might access separate HW elements. For single-core scope, the EXCLUSIVE AREAS keep the same purpose as on previous AUTOSAR implementations. (to be updated per **Crypto** usecase, to be detailed/removed if some modules require such kind of functionality for critical features which cannot be atomically shared among cores).

7. The **Crypto** driver assumes that each interrupt is routed by the system only to the core on which is supposed to be serviced.

8. The **HSE IP** driver has no configuration to enable multicore support but it supports multicore in standalone operation.

9. The **HSE IP** driver assumes that initialization is performed on each core, Hse_Ip_Init() is called separately for each core.

10. The **HSE IP** driver must be intialized only on MU instances different from the MU instances used by the Crypto driver. If no Crypto driver is used and only the **HSE IP** driver is used any MU instance can be used as long as that MU instance is not used by other application.

11. The **HSE IP** driver assumes that RTE module implements the EXCLUSIVE AREAS to be core-aware only. The rationale is that the module implementation ensures data integrity by separating the mappable elements for different cores already, thus implementing the EXCLUSIVE AREAS in a blocking manner (ex: spin-lock) on a multicore scope, might affect the performance of the drivers on the two cores, although they might access separate HW elements. For single-core scope, the EXCLUSIVE AREAS keep the same purpose as on previous AUTOSAR implementations. (to be updated per **HSE IP** use case, to be detailed/removed if some modules require such kind of functionality for critical features which cannot be atomically shared among cores).

12. The **HSE IP** driver assumes that each interrupt is routed by the system only to the core on which is supposed to be serviced.

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

The **Crypto** driver supports one main function that can be configured to be scheduled by BSW scheduler: void Crypto_MainFunction (void). The period is configured by the following parameter: #define CRYPTO_MAIN_↩ FUNCTION_PERIOD 1U

6.2 API Requirements

The function Crypto_KeySetValid() must be called after the function Crypto_KeyElementSet() in order to validate the key.

6.3 Calls to Notification Functions, Callbacks, Callouts

For each asynchronous request the **Crypto** Driver shall notify CRYIF about the completion of the job by calling the CryIf_CallbackNotification function passing on the job information and the result of cryptographic operation. The CryIf_CallbackNotification should be defined within the CryIf module, which is provided as stub.

Chapter 7

Memory allocation

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

7.1 Sections to be defined in Crypto_MemMap.h

Section name	Type of section	Description
CRYPTO_START_SEC_CODE	Code	Start of memory section for code.
CRYPTO_STOP_SEC_CODE	Code	Stop of memory section for code.
CRYPTO_START_SEC_CONFIG↩_DATA_8_NO_CACHEABLE	Constant configuration data	Used for configuration constant data which have to be aligned to 8 bit and be placed in a non-cacheable memory area.
CRYPTO_STOP_SEC_CONFIG_↩DATA_8_NO_CACHEABLE	Constant configuration data	End of above section.
CRYPTO_START_SEC_CONST_8	Constant Data	Used for constants that have to be aligned to 8 bit.
CRYPTO_STOP_SEC_CONST_8	Constant Data	End of above section.
CRYPTO_START_SEC_CONST_32	Constant Data	Used for constants that have to be aligned to 32 bit.
CRYPTO_STOP_SEC_CONST_32	Constant Data	End of above section.
CRYPTO_START_SEC_CONST_↩UNSPECIFIED	Constant Data	Used for constants, does not fit the criteria of 8, 16 or 32 bit.
CRYPTO_STOP_SEC_CONST_↩UNSPECIFIED	Constant Data	End of above section.
CRYPTO_START_SEC_VAR_↩INIT_8	Variables	Used for variables 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 containing elements of maximum 8 bits. These variables are initialized with values after every reset.
CRYPTO_STOP_SEC_VAR_INIT↩_8	Variables	End of above section.

Section name	Type of section	Description
CRYPTO_START_SEC_VAR_↩INIT_8_NO_CACHEABLE	Variables	Used for variables which have to be aligned to 8 bit and be placed in a non cacheable memory area. For instance used for variables of size 8 bit or used for composite data types: arrays, structs containing elements of maximum 8 bits. These variables are initialized with values after every reset.
CRYPTO_STOP_SEC_VAR_INIT↩_8_NO_CACHEABLE	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩INIT_BOOLEAN	Variables	Used for boolean variables. These variables are initialized with values after every reset.
CRYPTO_STOP_SEC_VAR_INIT↩_BOOLEAN	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩INIT_UNSPECIFIED	Variables	Used for variables, structures, arrays, when the SIZE (alignment) does not fit the criteria of 8, 16 or 32 bit. These variables are initialized with values after every reset.
CRYPTO_STOP_SEC_VAR_INIT↩_UNSPECIFIED	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩CLEARED_8_NO_CACHEABLE	Variables	Used for variables which have to be aligned to 8 bit and be placed in a non-cacheable memory area. These variables are cleared to zero by start-up code.
CRYPTO_STOP_SEC_VAR_↩CLEARED_8_NO_CACHEABLE	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩CLEARED_32	Variables	Used for variables which have to be aligned to 32 bit. These variables are cleared to zero by start-up code.
CRYPTO_STOP_SEC_VAR_↩CLEARED_32	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩CLEARED_32_NO_CACHEABLE	Variables	Used for variables which have to be aligned to 32 bit and be placed in a non-cacheable memory area. These variables are cleared to zero by start-up code.
CRYPTO_STOP_SEC_VAR_↩CLEARED_32_NO_CACHEABLE	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩CLEARED_BOOLEAN	Variables	Used for boolean variables which have to be aligned to 8 bit. These variables are cleared to zero by start-up code.
CRYPTO_STOP_SEC_VAR_↩CLEARED_BOOLEAN	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩CLEARED_UNSPECIFIED	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8, 16 or 32 bit. These variables are cleared to zero by start-up code.

Section name	Type of section	Description
CRYPTO_STOP_SEC_VAR_↩CLEARED_UNSPECIFIED	Variables	End of above section.
CRYPTO_START_SEC_VAR_↩CLEARED_UNSPECIFIED_NO_↩CACHEABLE	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8, 16 or 32 bit and the variables must be placed in a non-cacheable memory area. These variables are cleared to zero by start-up code.
CRYPTO_STOP_SEC_VAR_↩CLEARED_UNSPECIFIED_NO_↩CACHEABLE	Variables	End of above section.
CRYPTO_START_SEC_↩VAR_SHARED_CLEARED_↩UNSPECIFIED_NO_CACHEABLE	Variables	Used for descriptors structures that are storing the requests sent from Crypto driver to HSE Firmware. Must be placed in the RAM memory shared between the host and the HSE Firmware. These variables are cleared to zero by start-up code.
CRYPTO_STOP_SEC_VAR↩_SHARED_CLEARED_↩UNSPECIFIED_NO_CACHEABLE	Variables	End of above section.

7.2 Linker command file

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

Chapter 8

Integration Steps

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

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

Chapter 9

External assumptions for driver

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

External Assumption Req ID	External Assumption Text
SWS_Crypto_00043	Range: - - 0x02 - The service request failed because the service is still busy - CRYPTO_E_SMALL_BUFFER - 0x03 - The service request failed because the provided buffer is too small to store the result - CRYPTO_↩E_ENTROPY_EXHAUSTION - 0x04 - The service request failed because the entropy of the random number generator is exhausted - CRYPTO_↩E_QUEUE_FULL - 0x05 - The service request failed because the queue is full - CRYPTO_E_KEY_READ_FAIL - 0x06 - The service request failed, because key element extraction is not allowed - CRYPTO_E_KEY↩_WRITE_FAIL - 0x07 - The service request failed because the writing access failed - CRYPTO_E_KEY_NOT_AVAILABLE - 0x08 - The service request failed because the key is not available - CRYPTO_E_KEY_↩NOT_VALID - 0x09 - The service request failed because the key is invalid. - CRYPTO_E_KEY_SIZE_MISMATCH - 0x0A - The service request failed because the key size does not match. - CRYPTO_E_JOB_CANCELED - 0x0C - The service request failed because the Job has been canceled. - CRYPTO_E_KEY_EMPTY - 0x0D - The service request failed because of uninitialized source key element. - Description: - – - Available via: - CryIf.h -
SWS_Crypto_00215	The Configuration pointer configPtr shall always have a null pointer value.
HSE_IP_006_005	For asynchronous polling requestests sent to HSE with the help of Hse_↩Ip_ServiceRequest API, the application shall call periodically the function Hse_Ip_MainFunction() in order to be notified when the service request is completed through the associated callback.
HSE_IP_006_007	The minimum value for the request timeout field of the pReqType parameter for a Hse_Ip_ServiceRequest API call shall be 1 ms.
EA_RTD_00071	If interrupts are locked, a centralized function pair to lock and unlock interrupts shall be used.
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.

External Assumption Req ID	External Assumption Text
EA_RTD_00092	The integrator shall allocate a single EcucPartition per core or the partition in which the Crypto is allocated shall be exclusively mapped to a core. Note: Internally, the Crypto will use the Core ID returned by GetCoreID API to reference the appropriate global data and configuration elements, that is why a core should reference only one configured partition.
EA_RTD_00093	The application shall define EcucCoreIDs in a compact/consecutive order, starting from zero.
EA_RTD_00094	When multicore support is enabled, the application shall call Crypto_Init() for each core, using the dedicated configuration pointer for that core.
EA_RTD_00096	The application shall pass the correct initialization pointer, specific to the partition in which the driver is to be used.
EA_RTD_00102	The application should not call Hse_Ip_ReleaseChannel() function while the channel is used for processing a HSE request. Releasing a channel shall be performed only in the following situations: 1. After the channel is reserved with Hse_Ip_GetFreeChannel() but before initiating a request over it using Hse_Ip_ServiceRequest(). 2. After reserving the channel and intitiating a request over it using Hse_Ip_ServiceRequest() only when the response from HSE is received. 3. After reserving the channel and intitiating a request over it using Hse_Ip_ServiceRequest(), in case that no response is received in the timeout window and Hse_Ip layer reports HSE_IP_↩ SRV_RSP_NO_RESPONSE, only after canceling the request by sending a HSE_SRV_ID_CANCEL service to HSE for that particular channel.
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.