Integration Manual

for S32K14X CRCU Driver

Document Number: IM2CRCUASR4.2 Rev0002R1.0.1
Rev. 1.0

Contents

Section number **Title** **Page**

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	13/07/2018	NXP MCAL Team	Updated version for ASR 4.2.2S32K14X1.0.1 Release

Chapter 2
Introduction

This integration manual describes the integration requirements for CRCU Driver for S32K14X microcontrollers.

2.1 Supported Derivatives

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

Table 2-1. S32K14X Derivatives

NXP Semiconductors	s32k148_lqfp144, s32k148_lqfp176, s32k148_mapbga100, s32k146_lqfp144, s32k146_lqfp100, s32k146_lqfp64, s32k146_mapbga100, s32k144_lqfp100, s32k144_lqfp64, s32k144_mapbga100, s32k142_lqfp100, s32k142_lqfp64, s32k118_lqfp48, s32k118_lqfp64

All of the above microcontroller devices are collectively named as S32K14X .

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 type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is 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.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
ASM	Assembler
AUTOSAR	Automotive Open System Architecture
CDD	Complex Device Driver
CRC	Cyclic Redundancy Check
CRCU	CRC Unit
DEM	Diagnostic Event Manager
DET	Development Error Tracer
N/A	Not Applicable
MCU	Micro Controller Unit

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	S32K14X Reference Manual	Reference Manual, Rev. 7, 4/2018
2	S32K142 Mask Set Errata for Mask 0N33V (0N33V)	30/11/2017
3	S32K144 Mask Set Errata for Mask 0N57U (0N57U)	30/11/2017
4	S32K146 Mask Set Errata for Mask 0N73V (0N73V)	30/11/2017
5	S32K148 Mask Set Errata for Mask 0N20V (0N20V)	30/11/2017
6	S32K118 Mask Set Errata for Mask 0N97V (0N97V)	26/02/2018

Chapter 3
Building the Driver

This section describes the source files and various compilers, linker options used for building the Autosar CRCU driver for NXP SemiconductorsS32K14X . It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

The CRCU driver files are compiled using

- Green Hills Multi 7.1.4 / Compiler 2017.1.4
- (Linaro GCC 6.3-2017.06~dev) 6.3.1 20170509 (Thu Dec 7 13:28:42 CST 2017 build.sh rev=g7fea41d s=L631 Earmv7 -V release_g7fea41d_build_Fed_Earmv7) (from S32-DS-ARM_v2018)
- IAR: V8.11.2

The compiler, linker flags used for building the driver are explained below:

Note

The TS_T40D2M10I1R0 plugin name is composed as follow:

TS_T = Target_Id

D = Derivative_Id

M = SW_Version_Major

I = SW_Version_Minor

R = Revision

(i.e. Target_Id = 40 identifies CORTEXM architecture and Derivative_Id = 2 identifies the S32K14X)

3.1.1 GHS Compiler/Linker/Assembler Options

Table 3-1. Compiler Options

Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+
-ansi	Specifies ANSI C with extensions. This mode extends the ANSI X3.159-1989 standard with certain useful and compatible constructs.
-Osize	Optimize for size.
-dual_debug	Enables the generation of DWARF, COFF, or BSD debugging information in the object file
-G	Generates source level debugging information and allows procedure call from debugger's command line.
--no_exceptions	Disables support for exception handling
-Wundef	Generates warnings for undefined symbols in preprocessor expressions
-Wimplicit-int	Issues a warning if the return type of a function is not declared before it is called
-Wshadow	Issues a warning if the declaration of a local variable shadows the declaration of a variable of the same name declared at the global scope, or at an outer scope
-Wtrigraphs	Issues a warning for any use of trigraphs
-Wall	Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid even in conjunction with macros.
--prototype_errors	Generates errors when functions referenced or called have no prototype
--incorrect_pragma_warnings	Valid #pragma directives with wrong syntax are treated as warnings
-noslashcomment	C++ like comments will generate a compilation error
-preprocess_assembly_files	Preprocesses assembly files
-nostartfile	Do not use Start files
--short_enum	Store enumerations in the smallest possible type
-c	Produces an object file (called input-file.o) for each source file.
--no_commons	Allocates uninitialized global variables to a section and initializes them to zero at program startup.
-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. Produces an object file (called input-file.o) for each source file.
-list	Creates a listing by using the name of the object file with the .lst extension. Assembler option
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DDISABLE_MCAL_INTERMODULE_ASR_CHECK	-D defines a preprocessor symbol to disable the inter-module version check for AR_RELEASE versions. DISABLE_MCAL_INTERMODULE_ASR_CHECK: By default in the package, drivers are compiled to perform the inter-module version check as per Autosar BSW004. When the inter-module version check needs to be disabled then the DISABLE_MCAL_INTERMODULE_ASR_CHECK global define must be added to the list of compiler options.
-DGHS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GHS preprocessor symbol.

Table 3-2. Assembler Options

Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+
-c	Produces an object file (called input-file.o) for each source file.
-preprocess_assembly_files	Preprocesses assembly files
-asm=list	Creates a listing by using the name of the object file with the .lst extension. Assembler option

Table 3-3. Linker Options

Option	Description
-Mn	Map file numeric ordering
-delete	Removal from the executable of functions that are unused and unreferenced
-v	Display removed unused functions
-ignore_debug_references	Ignores relocations from DWARF debug sections when using -delete.
-map	Creates a detailed map file
-keepmap	Keep the map file in the event of a link error
-lstartup	Link libstartup library -Run-time environment startup routines
-lsys	Link libsys library -Run-time environment system routines
-larch	Link libarch library -Target-specific run-time support. Any file produced by the Green Hills Compiler may depend on symbols in this library.
-lansi	Link libansi library -the standard C library
-L(/lib/thumb2)	Link thumb2 library
-lutf8_s32	Include utf8_s32.a to use the Wide Character Functions

3.1.2 GCC Compiler/Linker/Assembler Options

Table 3-4. Compiler Options

Option	Description
-c	Produces an object file (called input-file.o) for each source file.
-Os	Use optimization for size.
-ggdb3	Produce debugging information for use by GDB. Level 3 includes extra information, such as all the macro definitions present in the program.
-mcpu=cortex-m4	Selects target processor: Arm Cortex M4
-mcpu=cortex-m0plus	Selects target processor: Arm Cortex M0+
-mthumb	Selects generating code that executes in Thumb state.
-ansi	Specifies ANSI C with extensions.
-mlittle-endian	Generate code for a processor running in little-endian mode.
-fomit-frame-pointer	Removes the frame pointer for all functions, which might make debugging harder.
-msoft-float	Use software floating-point instructions.

Table continues on the next page...

Table 3-4. Compiler Options (continued)

Option	Description
-fno-common	Specifies that the compiler should place uninitialized global variables in the data section of the object file, rather than generating them as common blocks.
-Wall	Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid even in conjunction with macros.
-Wextra	Enables some extra warning flags that are not enabled by '-Wall'.
-Wstrict-prototypes	Warn if a function is declared or defined without specifying the argument types.
-Wno-sign-compare	Do not warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.
-fstack-usage	Geneates an extra file that specifies the maximum amount of stack used, on a per-function basis.
-fdump-ipa-all	Enables all inter-procedural analysis dumps.
-Werror=implicit-function-declaration	Generates an error when the prototype of the function is not defined..
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DGCC	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GCC preprocessor symbol.

Table 3-5. Assembler Options

Option	Description
-mcpu=cortex-m4	Selects target processor: Arm Cortex M4
-mcpu=cortex-m0plus	Selects target processor: Arm Cortex M0+
-c	Produces an object file (called input-file.o) for each source file.
-mthumb	This option specifies that the assembler should start assembling Thumb instructions.
-x assembler-with-cpp	Indicates that the assembly code contains C directives and the C preprocessor must be run.

Table 3-6. Linker Options

Option	Description
-Map=filename	Print a link map to the file mapfile.
-T scriptfile	Use scriptfile as the linker script. This script replaces ld's default linker script(rather than adding to it), so commandfile must specify everything necessary to describe the output file.

3.1.3 IAR Compiler/Linker/Assembler Options

Table 3-7. Compiler Options

Option	Description
--cpu=Cortex-M4	Selects target processor: Arm Cortex M4
--cpu=Cortex-M0+	Selects target processor: Arm Cortex M0+
--cpu_mode=thumb	Selects generating code that executes in Thumb state.
--endian=little	Specifies the endianess of core: little endian.
-Ohz	Sets the optimization level to High, favoring size.
-c	Produces an object file (called input-file.o) for each source file.
--no_clustering	Disables static clustering optimizations.
--no_mem_idioms	Makes the compiler to not optimize code sequences that clear, set, or copy a memory region.
--no_explicit_zero_opt	Places the zero initialized variables in data section instead of bss.
--debug	Makes the compiler include information in the object modules.
--diag_suppress=Pa050	Suppresses diagnostic messages (warnings) about non-standard line endings.
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DIAR	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the IAR preprocessor symbol.
--require_prototypes	Forces the compiler to verify that all functions have proper prototypes.
--no_wrap_diagnostics	Disables line wrapping of diagnostic messages issued by compiler.
--no_system_include	Disables the automatic search for system include files.
-e	Enables language extensions. This option is needed by FLS driver which uses _packed structures.

Table 3-8. Assembler Options

Option	Description
--cpu=Cortex-M4	Selects target processor: Arm Cortex M4
--cpu=Cortex-M0+	Selects target processor: Arm Cortex M0+
--cpu_mode=thumb	Selects generating code that executes in Thumb state.
-g	Use this option to disable the automatic search for system include files.

Table 3-9. Linker Options

Option	Description
--map filename	Produces a map file.
--no_library_search	Disables automatic runtime library search.
--entry _start	Treats the symbol _start as a root symbol and as the start of the application.
--enable_stack_usage	Enables stack usage analysis.
--skip_dynamic_initialization	Suppress dynamic initialization during system startup.

Table continues on the next page...

Table 3-9. Linker Options (continued)

Option	Description
--no_wrap_diagnostics	Disables line wrapping of diagnostic messages issued by linker.
--config	Specifies the configuration file to be used by the linker.

3.2 Files required for Compilation

This section describes the include files required to compile, assemble (if assembler code) and link the CRCU driver for S32K14X 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.

CRCU Files

- ..\Crcu_TS_T40D2M10I1R0\include\Crcu.h
- ..\Crcu_TS_T40D2M10I1R0\include\CDD_Crcu.h
- ..\Crcu_TS_T40D2M10I1R0\include\Crcu_Types.h
- ..\Crcu_TS_T40D2M10I1R0\include\Crcu_Ipw.h
- ..\Crcu_TS_T40D2M10I1R0\include\Crcu_Ipw_Types.h
- ..\Crcu_TS_T40D2M10I1R0\include\Crcu_Crcv2_Types.h
- ..\Crcu_TS_T40D2M10I1R0\include\Crcu_Crcv2.h
- ..\Crcu_TS_T40D2M10I1R0\include\Crcu_Reg_eSys_Crcv2.h
- ..\Crcu_TS_T40D2M10I1R0\src\CDD_Crcu.c
- ..\Crcu_TS_T40D2M10I1R0\src\Crcu_Ipw.c
- ..\Crcu_TS_T40D2M10I1R0\src\Crcu_Crcv2.c

CRCU Generated Files

- CDD_Crcu_Cfg.h
- CDD_Crcu_Cfg.c
- CDD_Crcu_PBcfg_<VariantName>.c

For driver compilation, CDD_Crcu_PBcfg_<VariantName>.c should be generated by the user using a configuration tool. The file contains the definition of the init pointer for the respective variant.

As a deviation from standard:

- Crcu_PBcfg_<VariantName>.c files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB)

- Crcu_Cfg.c file will contain the definition for all configuration structures containing only variables that are not variant aware, configured and generated only once. This file alone does not contain the whole structure needed by Crcu_Init function to configure the driver. Based on the number of variants configured in the EcuC, there can be more than one configuration structure for one module even for PreCompile variant.

Files from Mcl folder, if DMA is used:

- ..\Mcl_TS_T40D2M10I1R0\include\CDD_Mcl.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_Dma.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_Dma_Types.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_DmaMux_Types.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_EnvCfg.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_IPW.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_IPW_Notif.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_IPW_Types.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_Notif.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_TrgMux.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_TrgMux_Types.h
- ..\Mcl_TS_T40D2M10I1R0\include\Mcl_Types.h
- ..\Mcl_TS_T40D2M10I1R0\include\Reg_eSys_Dma.h
- ..\Mcl_TS_T40D2M10I1R0\include\Reg_eSys_DmaMux.h
- ..\Mcl_TS_T40D2M10I1R0\include\Reg_eSys_TrgMux.h
- ..\Mcl_TS_T40D2M10I1R0\src\CDD_Mcl.c
- ..\Mcl_TS_T40D2M10I1R0\src\Mcl_Dma.c
- ..\Mcl_TS_T40D2M10I1R0\src\Mcl_Dma_Irq.c
- ..\Mcl_TS_T40D2M10I1R0\src\Mcl_DmaMux.c
- ..\Mcl_TS_T40D2M10I1R0\src\Mcl_IPW.c
- ..\Mcl_TS_T40D2M10I1R0\src\Mcl_TrgMux.c
- CDD_Mcl_Cfg.h
- Mcl_DmaMux.h
- CDD_Mcl_Cfg.c
- CDD_Mcl_PBcfg_<VariantName>.c

Files from Base common folder

- ..\Base_TS_T40D2M10I1R0\include\Can_GeneralTypes.h
- ..\Base_TS_T40D2M10I1R0\include\Compiler.h
- ..\Base_TS_T40D2M10I1R0\include\Compiler_Cfg.h
- ..\Base_TS_T40D2M10I1R0\include\ComStack_Cfg.h
- ..\Base_TS_T40D2M10I1R0\include\ComStack_Types.h
- ..\Base_TS_T40D2M10I1R0\include\Eth_GeneralTypes.h
- ..\Base_TS_T40D2M10I1R0\include\Fr_GeneralTypes.h

- ..\Base_TS_T40D2M10I1R0\include\Lin_GeneralTypes.h
- ..\Base_TS_T40D2M10I1R0\include\Mcal.h
- ..\Base_TS_T40D2M10I1R0\include\Crcu_MemMap.h
- ..\Base_TS_T40D2M10I1R0\include\Platform_Types.h
- ..\Base_TS_T40D2M10I1R0\include\Reg_eSys.h
- ..\Base_TS_T40D2M10I1R0\include\RegLockMacros.h
- ..\Base_TS_T40D2M10I1R0\include\SilRegMacros.h
- ..\Base_TS_T40D2M10I1R0\include\Soc_Ips.h
- ..\Base_TS_T40D2M10I1R0\include\Std_Types.h
- ..\Base_TS_T40D2M10I1R0\include\StdRegMacros.h

Files from Det folder:

- ..\Det_TS_T40D2M10I1R0\include\Det.h
- ..\Det_TS_T40D2M10I1R0\src\Det.c

3.3 Setting up the Plug-ins

The CRCU driver was designed to be configured by using the EB Tresos Studio (version EB tresos Studio 23.0.0 b170330-0431 or later.)

Location of various files inside the module folder:

- VSMD (Vendor Specific Module Definition) file in EB tresos Studio XDM format:
  - ..\Crcu_TS_T40D2M10I1R0\config\Crcu.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k118_lqfp48.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k118_lqfp64.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k142_lqfp64.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k142_lqfp100.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k144_lqfp64.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k144_lqfp100.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k144_mapbga100.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k146_lqfp64.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k146_lqfp100.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k146_mapbga100.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k146_lqfp144.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k148_mapbga100.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k148_lqfp144.epd
  - ..\Crcu_TS_T40D2M10I1R0\autosar\Crcu_s32k148_lqfp176.epd
- Code Generation Templates for parameters without variation points:

- ..\Crcu_TS_T40D2M10I1R0\generate_PC\include\CDD_Crcu_Cfg.h
- ..\Crcu_TS_T40D2M10I1R0\generate_PC\src\CDD_Crcu_Cfg.c
- ..\Crcu_TS_T40D2M10I1R0\generate_PB\src\CDD_Crcu_PBcfg.c

Steps to generate the configuration:

1. Copy the module folders Crcu_TS_T40D2M10I1R0, Mcl_TS_T40D2M10I1R0, Base_TS_T40D2M10I1R0, Resource_TS_T40D2M10I1R0, Det_TS_T40D2M10I1R0 into the Tresos plugins folder.
2. Set the desired Tresos Output location folder for the generated sources and header files.
3. Use the EB tresos Studio GUI to modify ECU configuration parameters values.
4. Generate the configuration files.

Chapter 4
Function calls to module

4.1 Function Calls during Start-up

None.

4.2 Function Calls during Shutdown

None.

4.3 Function Calls during Wake-up

None.

Chapter 5
Module requirements

5.1 Exclusive areas to be defined in BSW scheduler

In the current implementation, CRCU is using the services of Run-Time Environment (RTE) for entering and exiting the critical regions. RTE implementation is done by the integrators of the MCAL using OS or non-OS services.

The following critical regions are used in the CRCU driver:

5.1.1 CRCU_EXCLUSIVE_AREA_00

Used in function Crcu_ValidateChannelStateAndSetBusyIfIdle().

5.1.2 CRCU_EXCLUSIVE_AREA_01

Used in function Crcu_SetChannelStateIdle().

Critical Region Exclusive Matrix

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

Table 5-1. Exclusive Areas

	CRCU_EXCLUSIVE_AREA_00	CRCU_EXCLUSIVE_AREA_01
CRCU_EXCLUSIVE_AREA_00	X	
CRCU_EXCLUSIVE_AREA_01		X

5.2 Peripheral Hardware Requirements

The CRCU driver uses CRCv2 peripheral.

Information about the CRCv2 peripheral is available in the S32K14X Reference Manual.

5.3 ISR to configure within OS – dependencies

If DMA transfer mode is used, MCL_DMA_CH_x_ISR routines should be used for each DMA channel. It depends on the MCL configuration. In this case, Crcu_DmaTransferComplete0**x** function should be configured in MCL plugin as DMA notification parameter. This is required to update Crcu driver internal statuses.

The following function should be configured for DMA as notification parameter in MCL:

Table 5-2. DMA notification parameter

Function Name	Observations
Crcu_DmaTransferComplete0	DMA notification parameter

5.4 ISR Macro

None.

5.5 Other AUTOSAR modules - dependencies

- **Det:** 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 'CrcuDevErrorDetect' configuration parameter.
- **Base:** This module is necessary for a reference to the Wakeup source for this controller as defined in the ECU State Manager.
- **Resource:** Sub-Derivative model is selected from Resource configuration.
- **Rte:**The Rte module is used to manage the exclusive area inside CRCU driver.
- **Mcl:**In DMA mode, the Mcl is used to configure the DMA channels which used for transfering data from app to CRC IP, using DMA.

5.6 Data Cache Restriction

In the DMA transfer mode, DMA transfers may issue cache coherency problems. To avoid possible coherency issues when D-CACHE is enabled, the user shall ensure that the buffers used as TCD source and destination are allocated in the NON-CACHEABLE area (by means of Memmap).

5.7 User Mode support

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

Chapter 6
Main API Requirements

6.1 Main functions 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 Crcu Driver provides a Crc complete notification callback per Crcu channel that is called whenever the Crc calculation initiated by a call to Crcu_AsyncCalculateChannelCrc() function is complete..

The syntax of this function is as follows:

```
            void Crcu_AsyncCalculateCrcCompleteNotif(Crcu_ChannelIdType channelId,
Crcu_ValueType crc)
```

An extern declaration of this function is available in CDD_Crcu_Cfg.c or CDD_Crcu_PBcfg.c, depending on the variant used. The function has to be implemented by the user.

Chapter 7
Memory Allocation

7.1 Sections to be defined in MemMap.h

Table 7-1. Sections to be defined in MemMap.h

Section Name	Type Of Section	Description
CRCU_START_SEC_CODE	Code	Start of memory Section for Code
CRCU_STOP_SEC_CODE	Code	End of memory Section for Code
CRCU_START_SEC_VAR_NO_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 never cleared and never initialized by start-up code.
CRCU_STOP_SEC_VAR_NO_INIT_UNSPECIFIED	Variables	End of above section.
CRCU_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.
CRCU_STOP_SEC_VAR_INIT_UNSPECIFIED	Variables	End of above section.
CRCU_START_SEC_CONST_8	Constant Data	Used for constants to be aligned to 8 bit. For instance used for constants of size 8 bit or used for composite data types: arrays, structs containing elements of maximum 8 bits.
CRCU_STOP_SEC_CONST_8	Constant Data	End of above section.
CRCU_START_SEC_CONST_32	Constant Data	Used for constants to be aligned to 32 bit. For instance used for constants of size 32 bit or used for composite data types: arrays, structs containing elements of maximum 32 bits.
CRCU_STOP_SEC_CONST_32	Constant Data	End of above section.
CRCU_START_SEC_CONST_UNSPECIFIED	Constant Data	Used for constants that does not fit the criteria of 8,16 or 32 bit.
CRCU_STOP_SEC_CONST_UNSPECIFIED	Constant Data	End of above section.
CRCU_START_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data
CRCU_STOP_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	End of above section.

7.2 Linker command file

Memory shall be allocated for every section defined in CRCU_MemMap.h

Chapter 8
Configuration parameters considerations

Configuration parameter class for Autosar CRCU driver fall into the following variants as defined below:

8.1 Configuration Parameters

Configuration parameter class for CRCU driver fall into the following variants as defined below:

Table 8-1. Configuration Parameters

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
CrcuGeneral			
	CrcuDevErrorDetect	Pre Compile parameter for All variants of configuration	Pre Compile
	CrcuVersionInfoApi	Pre Compile parameter for All variants of configuration	Pre Compile
	CrcuDmaUsed	Pre Compile parameter for All variants of configuration	Pre Compile
	CrcuEnableUserModeSupport	Pre Compile parameter for All variants of configuration	Pre Compile
CrcuChannel			
	CrcuChannelId	VariantPC or VariantPB	Post Build
	CrcChannel	VariantPC or VariantPB	Post Build
	CrcuDmaChannelRef	VariantPC or VariantPB	Post Build
	CrcuAsyncCalculateCrcComplete Notification	VariantPC or VariantPB	Post Build
CrcuChannelConfig			
	CrcuChannelConfigId	VariantPC or VariantPB	Post Build
	Crcu_Width	VariantPC or VariantPB	Post Build

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	Crcu_Polynom	VariantPC or VariantPB	Post Build
	Crcu_In_Swap	VariantPC or VariantPB	Post Build
	Crcu_Out_Swap	VariantPC or VariantPB	Post Build
	Crcu_Out_Inversion	VariantPC or VariantPB	Post Build
CommonPublishedInformation			
	ArReleaseMajorVersion	VariantPC or VariantPB	Post Build
	ArReleaseMinorVersion	VariantPC or VariantPB	Post Build
	ArReleaseRevisionVersion	VariantPC or VariantPB	Post Build
	ModuleId	VariantPC or VariantPB	Post Build
	SwMajorVersion	VariantPC or VariantPB	Post Build
	SwMinorVersion	VariantPC or VariantPB	Post Build
	SwPatchVersion	VariantPC or VariantPB	Post Build
	VendorApiInfix	VariantPC or VariantPB	Post Build
	VendorId	VariantPC or VariantPB	Post Build

Chapter 9
Integration Steps

This section gives a brief overview of the steps needed for integrating Cyclic Redundancy Check Unit :

- Generate the required CRCU configurations. For more details refer to section Files required for Compilation
- Allocate proper memory sections in CRCU_MemMap.h and linker command file. For more details refer to section
- Compile & build the CRCU with all the dependent modules. For more details refer to section Building the Driver

Chapter 10
External Assumptions for CRCU driver

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

[SWS_CDD_CRCU_00031]

<< Application shall provide the value of the seed according with the configured CRC protocol width. The Crcu driver will ignore the MSB bits that exceed the width of the CRC protocol. Thus, if CRCU_WIDTH_16 is selected in the channel configuration, the MSB [16-31] bits in the seed will be ignored. >>

[SWS_CDD_CRCU_00039]

<< Before calling Crcu_SyncCalculateChannelCrc(), the application is responsible for initializing the Crcu driver (by calling Crcu_Init()), configuring the channel (by calling Crcu_SetChannelConfig()) and setting the initial CRC value (by calling Crcu_SetChannelSeed()). >>

[SWS_CDD_CRCU_00053]

<< When calling the API functions of CRCU driver, the calling entity shall use the generated symbolic names for CRCU channels and channel configurations. >>

[SWS_CDD_CRCU_00068]

<< Before calling Crcu_AsyncCalculateChannelCrc(), the application is responsible for initializing the Crcu driver (by calling Crcu_Init()), configuring the channel (by calling Crcu_SetChannelConfig()) and setting the initial CRC value (by calling Crcu_SetChannelSeed()). >>

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number IM2CRCUASR4.2 Rev0002R1.0.1
Revision 1.0