User's Manual

CONFIDENTIAL

R-Car V4H AUTOSAR R19-11 MCAL

User's Manual

Driver Component

Embedded User's Manual

RTM8RC779GCMCL5DA0JCDRE
RTM8RC779GCMCL5QA0JCDRE
RTM8RC779GCSPL5DA0JCDRE
RTM8RC779GCSPL5QA0JCDRE
RTM8RC779GCCOM5DA0JCDRE
RTM8RC779GCCOM5QA0JCDRE
RTM8RC779GCCDD5DA0JCDRE
RTM8RC779GCCDD5QA0JCDRE

Target Device:
R-Car V4H

Renesas Electronics
www.renesas.com

Rev.3.00 Mar. 2025

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

List of Abbreviations and Acronyms

Abbreviation	Full Form
A/D	Analog / Digital
ANSI	American National Standards Institute
API	Application Programming Interface
ARXML/arxml	AutosaR eXtensible Mark-up Language
ASIC	Application Specific Integration Circuit
AUTOSAR	AUTomotive Open System ARchitecture
BSW	Basic SoftWare
BUS	BUS Network
CAN	Controller Area Network
CANIF/CanIf	CAN Interface
CC	Communication Controller
CDD	Complex Device Driver
CHI	Controller Host Interface
CPU	Central Processing Unit
CRAM	Cluster RAM
DA	Destination Address of the Mac Frame
DEM	Diagnostic Event Manager
DET/Det	Default Error Tracer
DIO	Digital Input Output
DMA	Direct Memory Access
DMON	Delay MONitor
EB	External Buffer
ECM	Error Control Module
ECU	Electronic Control Unit
ECUM/EcuM	ECU State Manager
EEPROM	Electrically Erasable Programmable Read-Only Memory
ETH	Ethernet
FLS	FLaSh Driver
GNU	GNU is Not Unix
GTM	Generic Timer Module
GW	Gateway
HW	HardWare
I/O	Input Output
IB	Internal Buffer
ID/Id	IDentifier
INTPx	Interruptx
IRQ	Interrupt ReQuest
ISR	Interrupt Service Routine
KB	Kilo Byte

Abbreviation	Full Form
LPdu/Lpdu	Data Link Protocol Datagram Unit
LPS	Low Power Sample
LRAM	Local RAM
MAC	Media Access Control
MCAL	MicroController Abstraction Layer
MCU	MicroController Unit
MHz	Mega Hertz
MI	Maskable Interrupt
MII	Media Independent Interface
Multi-core	Multiple master PEs
NA	Not Applicable
Nm	Network Management
NMI	Nonmaskable Interrupt
OS/Os	Operating System
OSTM	Operating System TiMer
PBG	PBUS Guard
PDF/pdf	Parameter Definition File
PDU	Protocol Data Unit
PEG	PE Guard
PID/Pid	Protected Identifier
PLL	Phase Locked Loop
POC	Protocol Operation Control
PPR	Port Pin Read
PWM	Pulse Width Modulation
MSN	<MSN> is the name of each module in the MCAL or CDD driver can, dio, eth, fls, gpt, mcu, port, spi, wdg, cddiccom, cddiic, cddcrc, cddths, cddrfso, cddemm, cddipmmu
ICCOM	Inter-CPU Communication
RFSO	Failure Self-Detection Output
IIC	Inter-Integrated Circuit
IPMMU	IP Memory Management Unit
THS	THermal Sensor
EMM	Error Management Module
CRC	Cyclic Redundancy Check

Abbreviation	Full Form
RAM	Random Access Memory
RMII	Reduced Media Independent Interface
ROM	Read Only Memory
RTE	RunTime Environment
Rx	Receive
SA	Source Address of a Mac Frame
SCHM/SchM	Scheduler Manager
SCI	Serial Communication Interface
SDU	Service Data Unit
SFR	Special Function Register
SG	Scan Group
SPAL	Software Peripheral Abstraction layer
SPI	Serial Peripheral Interface
SW	SoftWare
SWS	SoftWare Specification
Sync	Synchronous
T&H	Track and Hold
TAU	Timer Array Unit
Tx	Transmit
VMON	Voltage MONitor
WDG/wdg	Watchdog
WDGIF	Watchdog Interface
WDT	WatchDog Timer
WDTB	Window Watchdog Timer B
WDTBA	Window Watchdog Timer B unit A

Definitions

Term	Represented by
ECMEMK	ECM Error Mask Register
ECMIRCFG	ECM Internal Reset Configuration Register
ECMKCPROT	ECM Key Code Protection Register
ECMNMICFG	ECM FE Level Interrupt Configuration Register
ECMNMIDTCFG	ECM FE Level Interrupt Delay Timer Configuration Register
EIC	EI Level Interrupt Control Registers
FEINTC	FEINT Status Clear Register
FEINTF	FEINT Status Register
FEINTMSK	FEINT Event Mask Register
IMR	EI Level Interrupt Mask Registers
NA	Not Available
OSTM	OS Timer
Port	Represents a whole configurable port on a microcontroller device.
PORT channel	Numeric identifier linked to a hardware PORT
PORT Output State	Defines the output state for a PORT signal. It could be: High Low
PORT period	Defines the period of the PORT signal.
Port Pin	Represents a single configurable input or output pin on a microcontroller device.
PORT Polarity	Defines the starting output state of each PORT channel
Sl. No.	Serial Number
TAUD	Timer Array Unit D
TAUJ	Timer Array Unit J
TMU	Timer Unit
WDTBA	Window Watchdog Timer Unit the identified by index "A" with AWO domain
WDTBn	Window Watchdog Timer Unit the identified by index "n" with ISO domain
WDTBnEVAC/WDTBAEVAC	WDTB Enable Register for VAC
WDTBnMD/WDTBAMD	WDTB Mode Register
WDTBnREF/WDTBAREF	WDTB Reference Value Register
WDTBnWDTE/WDTBAWDTE	WDTB Enable Register for Fixed Activation Code
WDTBnWIS/WDTBAWIS	WDTB Interrupt Output Timing Setting Register
WDTBnWOST/WDTBAWOST	WDTB Window Open Start Register
√	It means that the item is valid
-	It means that the item is invalid

Notation for the Individual Descriptions by Product

Term	Description
[R-Car <Device Name>]	This description is used for the specific <Device Name>. (It does not depend on the compiler) . Example: [R-Car S4], [R-Car V4H], …
[R-Car <Device Name> (GHS)]	This description is used for the specific <Device Name> and GHS compiler. Example: [R-Car M3N (GHS)]
[R-Car <Device Name> (ARM)]	This description is used for the specific <Device Name> and ARM compiler. Example: [R-Car M3N (ARM)]

Table of Contents

List of Figures

List of Tables

1.Introduction

The purpose of this document is to describe the information related to the MCAL Driver Component This document shall be used as reference by the users of MCAL Driver Component. The system overview of complete AUTOSAR architecture is shown in the **Figure 1-1** below.

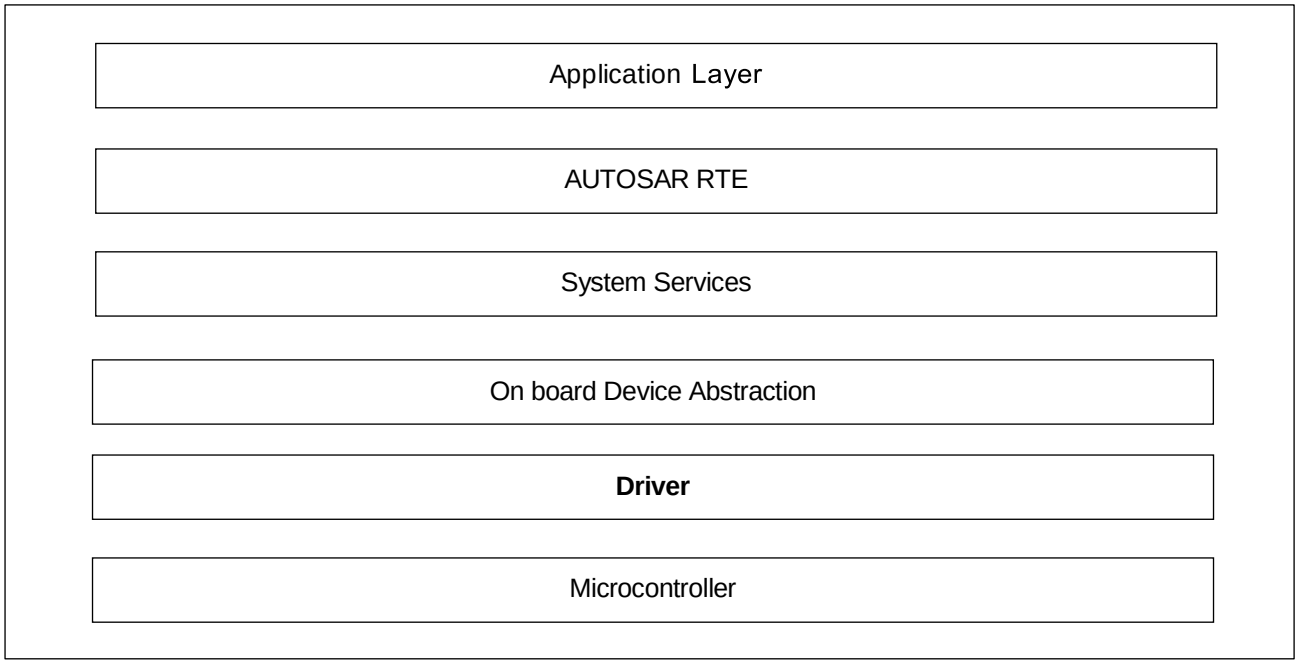


Figure 1-1 System Overview of AUTOSAR Architecture

The Driver is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.

The Figure 1-2 in the following page depicts the Driver as part of layered AUTOSAR MCAL Layer

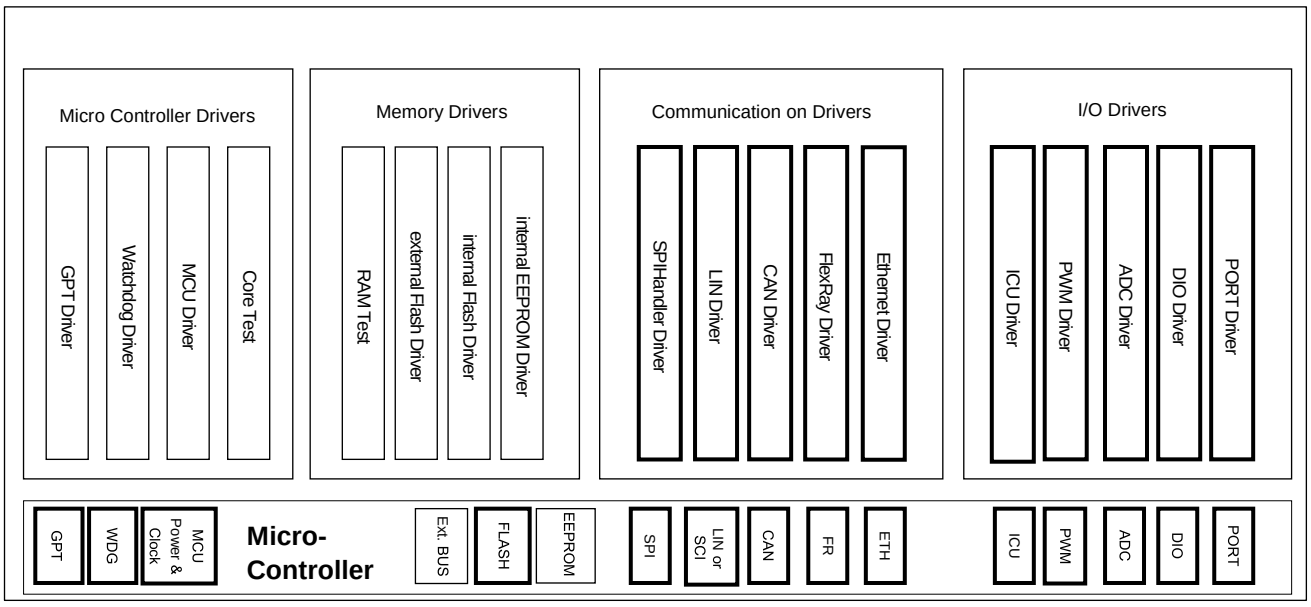


Figure 1-2 System Overview of the Driver in AUTOSAR MCAL Layer

The Driver Component comprises Embedded software and the Configuration Tool to achieve scalability and configurability.

The Driver Component code Generation Tool is a command line tool that accepts ECU configuration description files as input and generates source and header files. The configuration description is an ARXML file that contains information about the configuration for each Driver. The tool generates the <Msn>_PBcfg.c and <Msn>_Cfg.h

1.1 Supported MCAL Product Release Version

The Driver Component code Generation Tool is a command line tool that accepts ECU configuration description files as input and generates source and header files. The configuration description is an ARXML file that contains information about the configuration for each Driver. The tool generates the <Msn>_PBcfg.c and <Msn>_Cfg.h. The document corresponds to the MCAL Product Release Version shown in Table 1-1.

Table 1-1 Supported MCAL Product Release Version

Device	MCAL Product Release Version	No	Component Name	Version
R-Car V4H	Ver19.3.0	1	DIO	1.3.8
		2	PORT	1.1.12
		3	ICCOM	1.1.9
		4	IIC	1.0.12
		5	SPI	1.5.10
		6	FLS	1.1.10
		7	GPT	1.7.11
		8	MCU	1.1.13
		9	WDG	1.4.5
		10	CAN	1.1.13
		11	ETH	1.4.9
		12	THS	1.0.8
		13	IPMMU	1.0.8
		14	EMM	1.0.9
		15	RFSO	1.0.7
		16	CRC	1.0.8

1.2 Document Overview

The document has been segmented for easy reference. Table 1-2 provides user with an overview of the contents of each chapter:

Table 1-2 Document Overview

Chapter	Contents
Chapter 1 (Introduction)	This chapter provides an introduction and overview of each Driver Component.
Chapter 2 (Reference Documents)	This chapter lists the documents referred for developing this document.
Chapter 3 (Integration and Build Process)	This chapter explains the folder structure, and Makefile structure for each Driver Component. It also explains the Makefile descriptions, Integration of <MSN> Driver Component with other components and building the <MSN> Driver Component along with a sample application.
Chapter 4 (Generation Tool Guide)	This chapter provides guidelines for generation tools for each module.
Chapter 5 (CAN)	This chapter provides user's manual information for CAN driver components.
Chapter 6 (DIO)	This chapter provides user's manual information for DIO driver components.
Chapter 7 (PORT)	This chapter provides user's manual information for PORT driver components.
Chapter 8 (ETH)	This chapter provides user's manual information for ETH driver components.
Chapter 9 (FLS)	This chapter provides user's manual information for FLS driver components.
Chapter 10 (GPT)	This chapter provides user's manual information for GPT driver components.
Chapter 11 (ICCOM)	This chapter provides user's manual information for ICCOM driver components.
Chapter 12 (MCU)	This chapter provides user's manual information for MCU driver components.
Chapter 13 (IIC)	This chapter provides user's manual information for IIC driver components.
Chapter 14 (SPI)	This chapter provides user's manual information for SPI driver components.
Chapter 15 (WDG)	This chapter provides user's manual information for WDG driver components.
Chapter 16 (THS)	This chapter provides user's manual information for THS driver components.
Chapter 17 (IPMMU)	This chapter provides user's manual information for IPMMU driver components.
Chapter 18 (EMM)	This chapter provides user's manual information for EMM driver components.
Chapter 19 (RFSO)	This chapter provides user's manual information for RFSO driver components.
Chapter 20 (CRC)	This chapter provides user's manual information for CRC driver components.

This user's manual contains an appendix file with a deviation list and register access list for each module.
For more information, please refer to the following appendix file:
"V4H_Deviation_List.xlsx".

"V4H_Access_Register_List.xlsx".

2.Reference Documents

The documents referred to in this document is shown in Table 2-1 to Table 2-2.

Table 2-1 **Reference Documents(1/2)**

Sl. No.	Title	Version
[1]	R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview R-CarS4_V4H_V4M_MCAL_ModuleOverview.pdf	4.00
[2]	R-Car V4H AUTOSAR R19-11 MCAL User's Manual Driver Component Generation Tool User's Manual r11uz0131ej0300-rcarv4h-mcal-tum.pdf	3.00
[3]	R-Car V4H Series User's Manual: Hardware r19uh0186ej0110-r-carv4h.pdf	1.10
[4]	R-Car V4H Series User's Manual: Hardware Errata V4x_UM誤記_類件調査表_240930_1.xlsx	-

Table 2-2 Reference Documents(2/2)

Sl. No.	Title	Version
[4]	General Specification of Basic Software Modules AUTOSAR_EXP_CDDDesignAndIntegrationGuideline.pdf	R19-11
[5]	Specification of DIO Driver AUTOSAR_SWS_DIODriver.pdf	R19-11
[6]	Specification of Module Flash Driver AUTOSAR_SWS_FlashDriver.pdf	R19-11
[7]	Specification of GPT Driver AUTOSAR_SWS_GPTDriver.pdf	R19-11
[8]	Specification of MCU Driver AUTOSAR_SWS_MCUDriver.pdf	R19-11
[9]	Specification of PORT Driver AUTOSAR_SWS_PortDriver.pdf	R19-11
[10]	Specification of SPI Handler/Driver AUTOSAR_SWS_SPIHandlerDriver.pdf	R19-11
[11]	Specification of Watchdog Driver AUTOSAR_SWS_WatchdogDriver.pdf	R19-11
[12]	Specification of CAN Driver AUTOSAR_SWS_CANDriver.pdf	R19-11
[13]	Specification of Ethernet Driver AUTOSAR_SWS_EthernetDriver.pdf	R19-11
[14]	Specification of ECU Configuration AUTOSAR_TPS_ECUConfiguration.pdf	R19-11
[15]	Specification of Compiler Abstraction AUTOSAR_SWS_CompilerAbstraction.pdf	R19-11
[16]	General Specification of Basic Software Modules AUTOSAR_SWS_BSWGeneral.pdf	R19-11
[17]	Specification of RTE Software AUTOSAR_SWS_RTE.pdf	R19-11
[18]	Specification of Memory Mapping AUTOSAR_SWS_MemoryMapping.pdf	R19-11
[19]	Specification of Ethernet Interface AUTOSAR_SWS_EthernetInterface.pdf	R19-11

3.Integration and Build Process

3.1 Each Driver Component Makefile

Refer to “[1] R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 3.1 Driver Component Makefile.

3.1.1 Folder Structure

Refer to “[1] R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 3.2 Folder Structure and 3.3.<n>.3 Forder Structure.

Note <n>: is a number from 1 to 13.

4.Generation Tool Guide

For information on the Each Driver Component Code Generation Tool, refer to “[2] R-Car V4H AUTOSAR R19-11 MCAL User’s Manual Driver Component Generation Tool User’s Manual” document.

5.CAN

5.1 Overview

The purpose of this chapter is to describe the information related to CAN Driver Component.
The users of CAN Driver Component shall use this chapter as reference. This chapter describes the common features of CAN Driver Component.
This chapter is intended for the developers of ECU software using Application Programming Interfaces provided by AUTOSAR. The CAN Driver Component provides the following services:

- CAN Driver Component initialization
- CAN Driver mode control setting
- Tx and Rx of messages along with indication to upper layers
- Disable and enable the Interrupts
- Get Version information
- Call-out notification for L-PDU Receive Indication and Payload Length Error when CAN Pretended Networking activation.

The following diagram shows the system overview of the AUTOSAR Architecture. The CAN Driver initializes all the channels that are required to produce the CAN outputs.

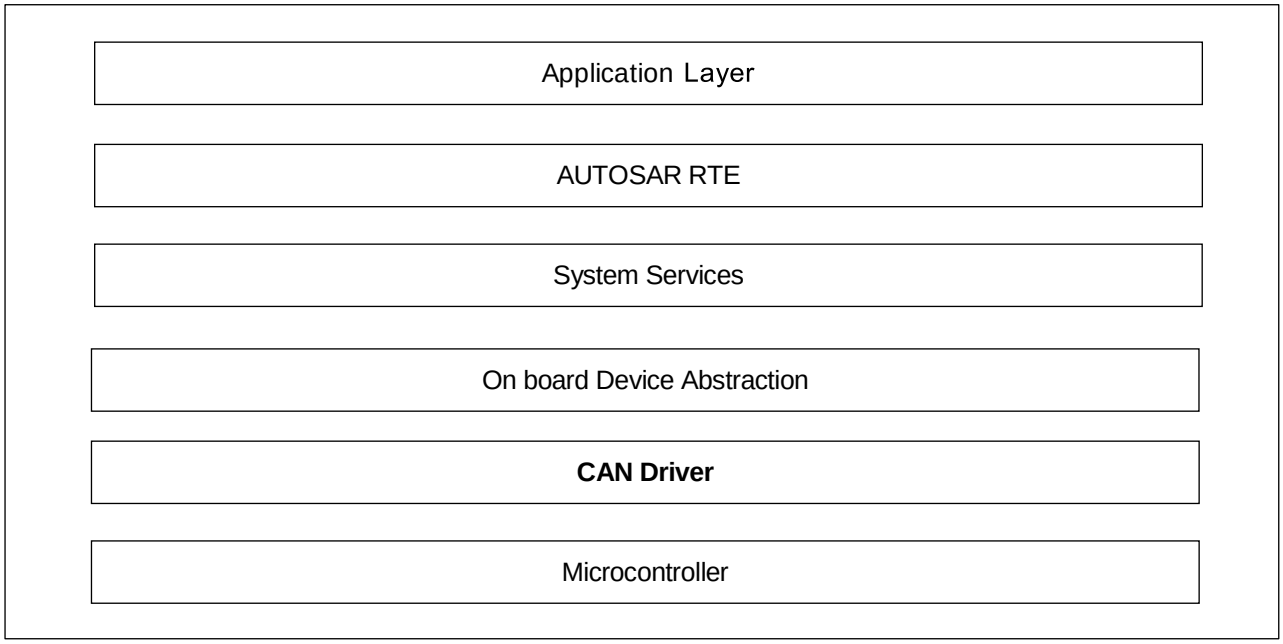


Figure 5-1 System Overview of AUTOSAR Architecture

The CAN Driver is part of the MCAL, the lowest layer of Basic Software in the AUTOSAR environment. The CAN Driver Component Generation Tool is a command line tool that accepts ECU Configuration Description files as input and generates C source files and C header files i.e. Can_PBcfg.c, Can_Lcfg.c, and Can_Cfg.h files. ECU Configuration Description files contain information about CAN Controller, Controller Base Address, Hardware Id, Controller Interrupt Mask Registers, Controller Baud rate, Controller Sync Jump Width, Time segment 1 and Time segment 2, HRH, HTH, CAN Filter Mask, Rx/Tx/BusOff/Wakeup processing, and CAN ID Types.

The figure below depicts the CAN Driver as part of layered AUTOSAR MCAL Layer:

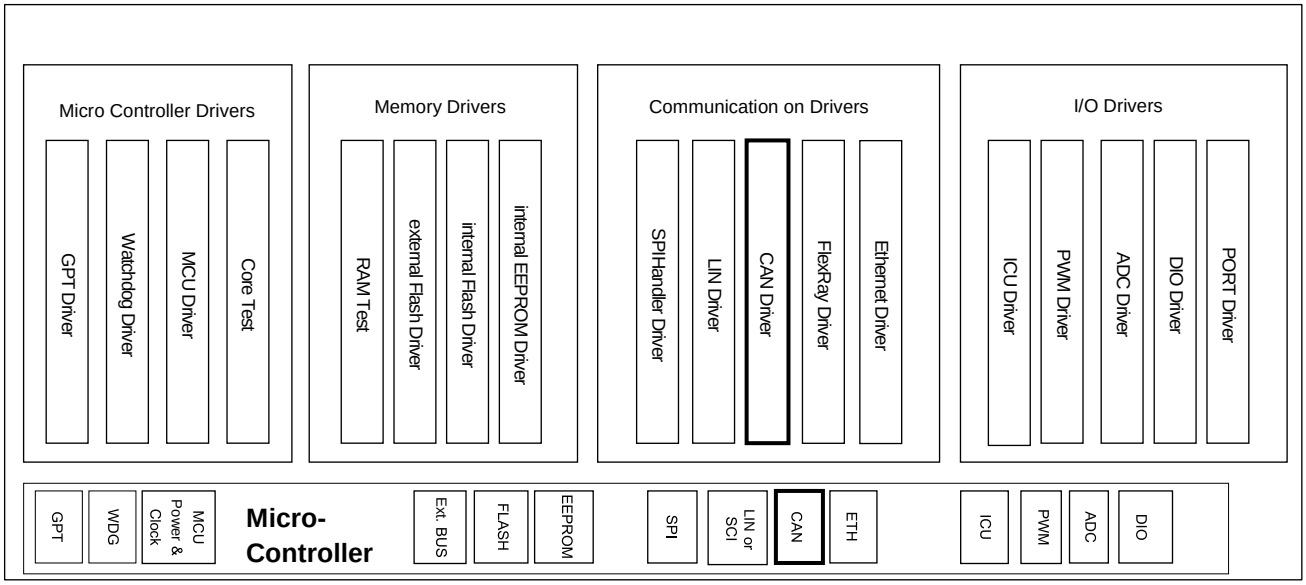


Figure 5-2 System Overview of the CAN Driver in AUTOSAR MCAL Layer

CAN Driver performs the hardware access and offers a hardware independent API to the upper layer. The only upper layer accessible to the CAN Driver is the CAN Interface. Several CAN Controllers can be controlled by the CAN Driver as long as they belong to the same CAN Hardware Unit.

The CAN Driver Component fulfills requirements of the upper layer communication components with respect to Initialization, Transmit confirmation, Receive indication, BusOff to CAN Interface layer and Wakeup notification to ECU State Manager.

The CAN Driver Component comprises two sections i.e., Embedded Software and the configuration tool to achieve scalability and configurability.

5.2 Forethoughts

5.2.1 General

The following information will help the user use the CAN Driver Component software efficiently:

- The start-up code is ECU specific. CAN Driver component does not implement the start-up code.
- MCU specific initializations such as reset registers, one-time writable registers, interrupt stack pointer, user stack pointer, internal watchdog, specific features of internal memory and registers is not implemented by CAN Driver. This initialization must be implemented by the start-up code.
- The ISR functions and the corresponding handler addresses are provided in Table 5-21. User must ensure that Interrupt Vector table is configured as per the information provided in the table.
- The CAN Driver uses the scheduler service SchM_Enter_Can function to get and SchM_Exit_Can function to release data protection to provide data integrity of shared resources.
- The CAN Driver does not support configuration of the mixed CAN-Ids (Standard as well as Extended CAN-Id types) for the same receive hardware object.
- The CAN Driver supports the interrupt service routine for all CAN Hardware Unit interrupts required. All unused interrupts in the CAN Controller are disabled by the CAN Driver. The configuration (i.e., priority) and the vector table is not made by the CAN Driver module.
- 3 common FIFO, 4 Queue and 8 Rx FIFO can be configured for a controller on Max.
- While configuring HOH Ids, HRH Ids should come first, followed by HTH Ids. The numbering of the HOH Ids must be consecutive without any gaps.
- While configuring Receive Rules only a single Receive Rule can be configured to a single or many receive message buffer (single HRH).
- To support DEEPSTOP functionality without resetting the microcontroller, the re initialization of the Driver using Can_Init API is supported. To achieve this functionality the 'CAN_E_ALREADY_INITIALIZED' DET error check is to be suppressed using ‘CanAlreadyInitDetCheck’ parameter when DET is enabled. When DET is disabled there is no impact of “CanAlreadyInitDetCheck” parameter (not supported).
- The CAN Driver uses the system service ‘GetCounterValue’ to detect the time-out of Global Mode/Channel Mode Transition.
- The Transmit History Interrupt is enabled by the parameter CanEnableTransmitHistoryInterrupt, and the interrupt occurs when 24 message histories are stored to the transmit history buffer. This setting prevents the loss of transmit confirmations in the interrupt mode.
- When the message matches more than one rule which belongs to the different CanHardwareObject simultaneously, the message is received by a CanHardwareObject with the smallest CanObjectID.
- It is the user’s responsibility to make sure input parameter when calling Can_GetVersionInfo() must be in range.
- As a hardware limitation, the maximum value of the PduId (swPduHandle) used within CAN module must not be greater than 255.
- As a hardware limitation, the maximum payload size is limited to BUFFER_MODE (both of TRANSMIT and RECEIVE) and TRANSMIT_QUEUE_MODE.
- In interrupt mode, the message standard buffers (having hardware objects configured as receive and memory mode as "BUFFER_MODE") are not operative, as they are not causing the interrupts. User can call Can_MainFunction_Read() to get messages from the standard message buffers having parameter ‘CanMainFunctionRWPeriod’ not configured.
- When CanPayloadOverflowModeSelect is configured as STORE and payload overflow occurs, the payload length notified to the upper layer should indicate the original payload length instead of the truncated length according to the buffer.

- If any controller interrupt occurs while controller interrupts are disabled by Can_DisableControllerInterrupts, the interrupt will be accepted right after the interrupts are enabled by Can_EnableControllerInterrupts.
- Relation of “CanIntervalTimerPrescalerSet” parameter and “CanTxRxFIFOTransmissionInterval” parameter and pclk is Interval Transmission Function. An idea for the calculation interval time can be retrieved from Section Interval Timer for FIFO Transmission referring to “[3] R-Car V4H Series User’s Manual: Hardware”.
- In case a CanHardwareObject is configured as a GW, CanControllerRef, CanGatewaySourceControllerRef and CanGatewayCopyObjectRef should not be configured as a reference to CanController or CanHardwareObject which belongs to a different CAN hardware unit.
- The CAN Driver supports Pretended Networking functionality. If Pretended Networking is activated CAN Driver shall call CanIf_RxIndication if and only if the received message matches the wakeup conditions of the CanIcomConfig.

5.2.2 Preconditions

- The following preconditions must be adhered by the user, for proper functioning of the CAN Driver Component:
- A mismatch in the version numbers will result in a compilation error. Ensure that the correct versions of the header and the source files are used. The version information is described in the 1.1 Supported MCAL Product Release Version.
- The Can_Cfg.h file generated by the CAN Driver Component Generation Tool should be compiled and linked with the CAN Driver Component source files.
- The application must be rebuilt, if there is any change in the Can_Cfg.h file is generated by the CAN Driver Component Generation Tool.
- It should be ensured that the controller is initialized and set to start mode, before transmitting an L-PDU.
- The CAN Driver component should be initialized before calling any other CAN Driver component API.
- All the CAN Driver APIs other than initialization should be invoked only through the CAN Interface component APIs.
- The internal data used by CAN Driver Component is read and written in the ISR(s) and Global APIs. The CAN Driver uses the scheduler service SchM_Enter_Can function to get and SchM_Exit_Can function to release data protection to provide data integrity of shared resources.
- The CAN controller channel must be in channel communication mode before invoking Can_SetIcomConfiguration API to activate Pretended Networking mode.
- To use CAN channel 0 to transmit/receive, port pin GP1_3 should be set as GPIO Output mode with HIGH Pin level value to enable the channel 0 transceiver on V4H White Hawk board. This setting can be configured by Configuration Description File and APIs provided by PORT driver.
- Can_EnableControllerInterrupts and Can_DisableControllerInterrupts APIs shall be called only after an ongoing transmission is completed. If Can_Write API is just called and the message is not confirmed to the user, these APIs shall not be called.
- CanHwFilter is not applicable for transmit CanHardwareObject. To avoid unexpected behavior, the user should not configure CanHwFilter to transmit CanHardwareObject.
- In interrupt mode, when calling Can_Write API transmits more than 32 CAN frames consecutively and a higher ISR occurs during transmission, possibly, there are lost notifications to CAN interface. Because the Transmit ISR has the capability to process up to 32 transmitted CAN frame at the same time only.

5.2.3 Data Consistency

To support the re-entrance and interrupt services, the AUTOSAR CAN component will ensure the data consistency while accessing its own RAM storage or hardware registers. The CAN component will use SchM_Enter_Can_<Exclusive Area> and SchM_Exit_Can_<Exclusive Area> functions. The SchM_Enter_Can_<Exclusive Area> function is called before the data needs to be protected and SchM_Exit_Can_<Exclusive Area> function is called after the data is accessed.

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

- CAN_RAM_DATA_PROTECTION
- CAN_INTERRUPT_CONTROL_PROTECTION

The functions SchM_Enter_Can_<Exclusive Area> and SchM_Exit_Can_<Exclusive Area> can be disabled by disabling the configuration parameter 'CanCriticalSectionProtection'. The flowchart indicates indicate the flow with the pre-compile option 'CanCriticalSectionProtection' enabled.

5.3 Architecture Details

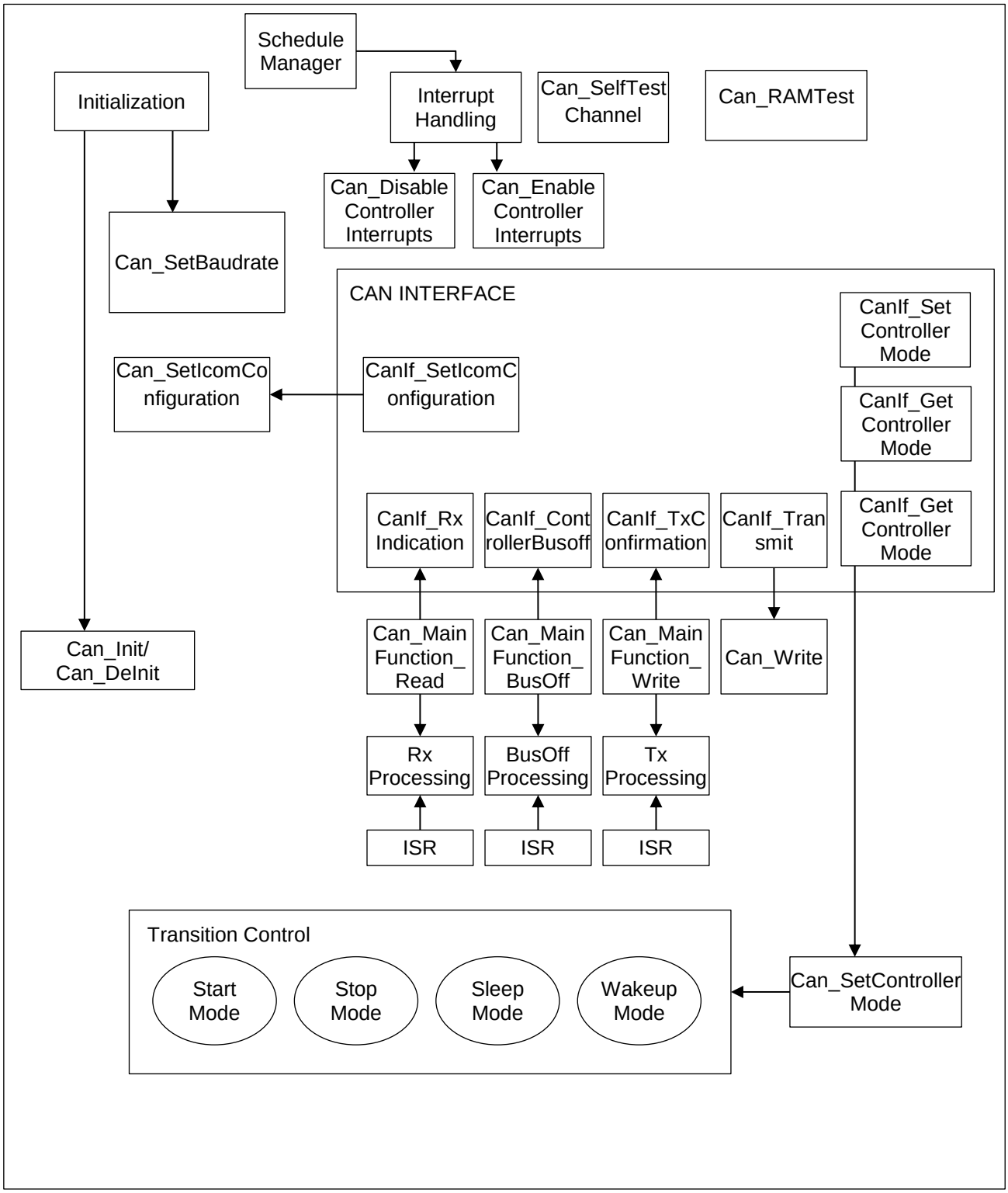


Figure 5-3 Driver Architecture

The component overview of the CAN Driver component details is given below:

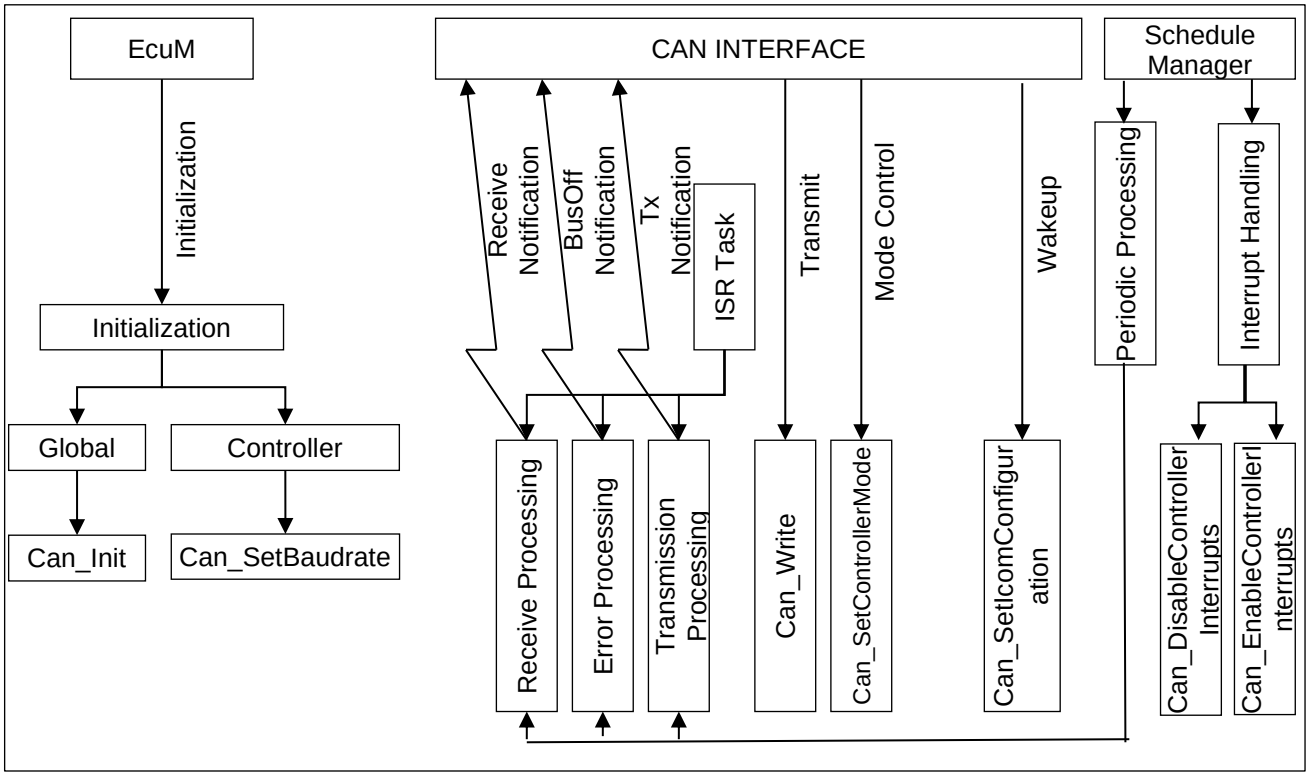


Figure 5-4 Component Overview of CAN Driver Component

The CAN Driver software component shall provide the following main features:

- The Initialization Module will provide the structures and APIs for both global and Controller specific initialization.
- In Mode Control Module service for each possible transition of CAN Controller will be provided. A state transition is triggered by software with the function “Can_SetControllerMode” with the required transition as parameter. The CAN Driver provides services to control each state of the CAN Controllers.
- The Periodical Task Module consists of the APIs required for periodical processing of Transmission, Receive indication and Bus Off.
- Transmit Processing Module processes Transmit requests of the upper layer. CAN Driver checks if the Hardware Transmit Object identified by HTH is free and all necessary control operations to initiate the transmission are done. If the Hardware Transmit Object is busy with another transmit request, it returns CAN_BUSY to the upper layer.
- Interrupt Processing Task Module implements the interrupt service routines for the entire CAN Hardware Unit interrupts required. All unused interrupts in the CAN Controller are disabled by the CAN Driver. The CAN Driver is responsible to reset the interrupt flag at the beginning of ISR.
- In Interrupt Control Module all the interrupts will be disabled or enabled for the particular CAN Controller. This module implements the interrupt service routines for all CAN Hardware Unit interrupts that will be required. All unused interrupts in the CAN Controller shall be disabled by the CAN Driver.
- CAN Pretended Networking shall be activated by calling Can_SetIcomConfiguration with a configuration ID not set to 0. If activation was successful, then CanIf_CurrentIcomConfiguration shall be called with the parameter Error set to ICOM_SWITCH_E_OK referring to the corresponding CAN controller with the abstract CanIf ControllerId. If activation was not successful, the parameter Error set to ICOM_SWITCH_E_FAILED.

5.4 CAN Driver Component Header and Source File Description

This section explains the CAN Driver Component's source and header files. These files must be included in the project application while integrating with other modules.

The C header file generated by CAN Driver Component Generation Tool:

- Can_Cfg.h

The C source file generated by CAN Driver Component Generation Tool:

- Can_PBcfg.c
- Can_Lcfg.c

5.5 Application Programming Interface

This section explains the Data types and APIs provided by the CAN Driver Component to the Upper layers.

5.5.1 Imported Types

This section explains the Data types imported by the CAN Driver Component and lists the dependency on other modules.

5.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed:

- Std_VersionInfoType
- Std_ReturnType

5.5.1.2 Other Module Types

In this section, all types included in the CanIf.h, ComStack_Types.h, EcuM.h and Os.h are listed below.

- CanIf_ControllerModeType
- PduIdType
- EcuM_WakeupSourceType
- CounterType
- StatusType
- TickType

5.5.2 Type Definitions

The Table 5-1 to Table 5-10 explain the type definitions of CAN Driver Component according to AUTOSAR Specification.

5.5.2.1 **Can_ConfigType**

Table 5-1 Can_ConfigType

Name:	Can_ConfigType	
Type:	Structure	
Element:	Implementation-Specific	The contents of the initialization data structure are CAN specific
Description:	This type of the external data structure shall contain the initialization data for the CAN Driver.	

5.5.2.2 **Can_PduType**

Table 5-2 Can_PduType

Name:	Can_PduType		
Type:	Structure		
Element:	**Type**	**Name**	**Explanation**
	uint8*	sdu	This is service data unit that is transported inside the PDU.
	Can_IdType	id	This is the identifier of PDU, either standard or extended type.
	PduIdType	swPduHandle	This is a unique handle that represents PDU within a module. The PDU Id range depends on implementation.
	uint8	length	This is the Data Length Code that describes the SDU length.
Description:	This type is used to provide ID, DLC and SDU from CAN Interface to CAN Driver.		

5.5.2.3 **Can_IdType**

Table 5-3 Can_IdType

Name:	Can_IdType		
Type	uint16 or uint32		
Range:	**Type**	**Name**	**Explanation**
	uint16	id	If only Standard IDs are used. Range: 0--- 0x7FF
	uint32	id	If only Extended IDs are used. Range: 0…0xFFFFFFF
Description:	Represents the identifier of an L-PDU. For external Ids the most significant bit is set.		

5.5.2.4 **Can_ControllerStateType**

Table 5-4 Can_ControllerStateType

Name:	Can_ControllerStateType	
Type	Enumeration	
Range:	CAN_CS_UNINIT	CAN controller state UNINIT.
	CAN_CS_STARTED	CAN controller state STARTED.
	CAN_CS_STOPPED	CAN controller state STOPPED.
Description:	States used by the several ControllerMode functions.	

5.5.2.5 **Can_ReturnType**

Table 5-5 Can_ReturnType

Name:	Can_ReturnType	
Type	Enumeration	
Range:	CAN_BUSY	transmit request could not be processed as no transmit object was available.
Description:	Overlayed return value of Std_ReturnType for CAN driver API Can_Write()	

5.5.2.6 **Can_HwHandleType**

Table 5-6 Can_HwHandleType

Name:	Can_HwHandleType		
Type	uint8 or uint16		
Element:	**Type**	**Name**	**Explanation**
	uint8	id	If only Standard IDs are used. Range: 0--- 0x0FF
	uint16	id	If only Extended IDs are used. Range: 0…0xFFFF
Description:	Represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects, use the extended range.		

5.5.2.7 **Can_ErrorStateType**

Table 5-7 Can_ErrorStateType

Name:	Can_ErrorStateType	
Type	Enumeration	
Range:	CAN_ERRORSTATE_ACTIVE	The CAN controller fully takes part in communication.
	CAN_ERRORSTATE_PASSIVE	The CAN controller takes part in communication, but does not send active error frames.
	CAN_ERRORSTATE_BUSOFF	The CAN controller does not take part in communication.
Description:	This type defines error states	

5.5.2.8 **Can_SelfTestType**

Table 5-8 Can_SelfTestType

Name:	Can_SelfTestType	
Type	Enumeration	
Range:	CAN_T_SELF_OFF	Request a leave from self-test.
	CAN_T_SELF_EXTERNAL	Request transition to channel self-test mode 0 (External loop back)
	CAN_T_SELF_INTERNAL	Request transition to channel self-test mode 1 (Internal loop back)
Description:	Request transition to or a leave from self-test.	

5.5.2.9 **Can_RamTestWalkType**

Table 5-9 Can_RamTestWalkType

Name:	Can_RamTestWalkType	
Type	Enumeration	
Range:	CAN_RAMTEST_WALK_0	The CAN Ram test walk.

	CAN_ERRORSTATE_PASSIVE	The CAN error state passive
Description:	This type deceleration for Can_RamTest.	

5.5.2.10 **Can_RamTestFillType**

Table 5-10 Can_RamTestFillType

Name:	Can_ RamTestFillType	
Type	Enumeration	
Range:	CAN_RAMTEST_FILL_0	The CAN Ram test fill.
	CAN_RAMTEST_FILL_1	The CAN Ram test fill.
Description:	This type deceleration for Can_RamTest.	

5.5.3 Function Definitions

The Table 5-11 describes the API provided by the CAN Driver Component.

Table 5-11 API provided by the CAN Driver Component

Sl.No.	API's
AUTOSAR API	
1	Can_Init
2	Can_GetVersionInfo
3	Can_DeInit
4	Can_GetControllerErrorState
5	Can_GetControllerMode
6	Can_SetControllerMode
7	Can_DisableControllerInterrupts
8	Can_EnableControllerInterrupts
9	Can_Write
10	Can_MainFunction_Write
11	Can_MainFunction_Read
12	Can_MainFunction_BusOff
13	Can_MainFunction_Mode
14	Can_SetBaudrate
15	Can_GetControllerTxErrorCounter
16	Can_GetControllerRxErrorCounter
17	Can_SetIcomConfiguration
Renesas Original API	
1	Can_SelfTestChannel
2	Can_RAMTest

5.5.3.1 Renesas Original API

Table 5-12 Can_SelfTestChannel

Function Name:	Can_SelfTestChannel		
Syntax:	FUNC(Std_ReturnType, CAN_RSCAN_PUBLIC_CODE) Can_SelfTestChannel (uint8 Controller, Can_SelfTestType TestTransition)		
Service ID:	0x23		
Sync/Async:	Synchronous		
Reentrancy:	Non Re-entrant		
Parameters (In):	Controller	CAN controller for which test mode shall be changed	
		Range:	0..(number of controllers – 1)
	TestTransition	Self-Test mode	

		Range:	CAN_T_SELF_OFF CAN_T_SELF_EXTERNAL CAN_T_SELF_INTERNAL
Parameters (In-Out):	N/A		
Parameters (Out):	N/A		
Return Value:	Std_ReturnType	E_OK: Transition to Self-Test mode successful. E_NOT_OK: Transition to Self-Test mode failed.	
Description:	This service initiates the transition of mode to execute a self-test in internal or external loop back mode.		
Preconditions:	The CAN Driver must be initialized, and the controller must be in CAN_T_START state. This function is available only if pre-compile option CAN_SELFTEST_API is STD_ON		
Remarks:	-		
DET/DEM Errors Handled:	CAN_E_UNINIT	CAN module has not been initialized.	
	CAN_E_PARAM_CONTROLLER	Specified controller ID is invalid.	
	CAN_E_TRANSITION	Prohibited state transition is specified.	
	CAN_E_TIMEOUT_FAILURE	Changing controller state is timeout.	

Table 5-13 Can_RAMTest

Function Name:	Can_RAMTest		
Syntax:	FUNC(Std_ReturnType, CAN_RSCAN_PUBLIC_CODE) (uint8 LucUnit, uint16 LusPageID)		
Service ID:	0x22		
Sync/Async:	Synchronous		
Reentrancy:	Non Re-entrant		
Parameters (In):	LusPageID	Identifier of the RAM page to be tested - API will check whether input parameter is invalid or not. - This parameter is mandatory to be provided correctly (it cannot be omitted)	
		Range:	0..868
	LucUnit	Index of the RSCAN(FD) unit	

		Range:	0..255
Parameters (In-Out):	**Input/Output**	**Global Variable Name**	**Description**
	Input	Can_GaaRegs[LucUnit].pCmn->ulGLOCKK	Write protection data to LOCKK register (LpRSCFDnLOCKKReg)
		Can_GaaRegs[LucUnit].pCmn->ulGTSTCTR	Write protection data to LOCKK register (LpRSCFDnLOCKKReg)
		Can_GaaRegs[LucUnit].pCmn->ulGTSTCFG	Write tested RAM page to CFG register
	Output	-	-
	Input/Output	Can_GaaRegs[LucUnit].pCmn->ulGTSTCTR	Disable/Enable Ram Test
Parameters (Out):	N/A		
Return Value:	Std_ReturnType	E_OK: Testing RAM page passed. E_NOT_OK: (A) Testing RAM page failed. (B) A development error occurred (see DET/DEM Errors handled).	
Description:	This API used for testing one RAM page in the foreground.		
Preconditions:	The CAN Driver must be in UN-INITIALIZED state. This function is available only if pre-compile option CAN_RAMTEST_API is STD_ON		
Remarks:	-		
DET/DEM Errors Handled:	CAN_RAMTEST_E_INITIALIZED	Report to DET, if module is already initialized .	
	CAN_RAMTEST_E_OUT_OF_RANGE	Report to DET, if RAM test page is out of supported range	
	CAN_E_TIMEOUT_FAILURE	Changing controller state is timeout.	

5.5.4 Preemption of APIs

The following table specify the preemption of each API that can be invoked at the same time as the API.

Table 5-14 Preemption table of APIs of the CAN Driver

	Can_Init	Can_GetVersionInfo	Can_DeInit	Can_GetControllerErrorState	Can_GetControllerMode	Can_SetControllerMode	Can_DisableControllerInterrupts	Can_EnableControllerInterrupts	Can_Write	Can_MainFunction_Write	Can_MainFunction_Read	Can_MainFunction_BusOff	Can_MainFunction_Mode	Can_SetBaudrate	Can_GetControllerTxErrorCount	Can_GetControllerRxErrorCount	Can_SelfTestChannel	Can_SetIcomConfiguration	Can_RAMTest
Can_Init	-																		
Can_GetVersionInfo	√	√																	
Can_DeInit	-	√	-																
Can_GetControllerErrorState	-	√	-	*1															
Can_GetControllerMode	-	√	-	√	-										√	√			
Can_SetControllerMode	-	√	-	√	√	-									√	√			
Can_DisableControllerInterrupts	-	√	-	√	√	√	√								√	√			
Can_EnableControllerInterrupts	-	√	-	√	√	√	√	√							√	√			
Can_Write	-	√	-	√	√	√	√	√	√						√	√			
Can_MainFunction_Write	-	√	-	√	√	√	√	√	√	-					√	√			
Can_MainFunction_Read	-	√	-	√	√	√	√	√	√	√	-				√	√			
Can_MainFunction_BusOff	-	√	-	√	√	*2	√	√	√	√	√	-			√	√			
Can_MainFunction_Mode	-	√	-	√	√	√	√	√	√	√	√	√	-		√	√			
Can_SetBaudrate	-	√	-	√	√	√	√	√	√	√	√	√	√	*1	√	√			
Can_GetControllerTxErrorCounter	-	√	-												*1	√			
Can_GetControllerRxErrorCounter	-	√	-													*1			
Can_SelfTestChannel	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Can_SetIcomConfiguration	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Can_RAMTest	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

-: can't be invoked at the same time

√: can be invoked at the same time

Note:

1 Re-entrant for different Controllers. Non Re-entrant for the same Controller.

2 If the state transition START to STOP has been started by Can_SetControllerMode, just clear error

flags and do not perform further operations.

5.6 Development and Production Errors

In this section, the development errors reported by the CAN Driver Component are tabulated. The development errors will be reported only when the pre-compiler option “CanDevErrorDetection” is enabled in the configuration. The production code errors are not supported by the CAN Driver Component.

5.6.1 CAN Driver Component Development Errors

The Table 5-15 to Table 5-16 contain the DET errors reported by the CAN Driver Component. These errors are reported to the Development Error Tracer Module when the CAN Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 5-15 DET Errors of CAN Driver Component (1/2)

Sl. No.	1
Error Code	CAN_E_PARAM_HANDLE
Value	0x02
Related API(s)	Can_Write()
Source of Error	When the Tx L-PDU is not Configured and transmitted by invoking the above-mentioned API.
Sl. No.	**2**
Error Code	CAN_E_PARAM_CONTROLLER
Value	0x04
Related API(s)	Can_SetControllerMode(), Can_SetBaudrate(), Can_EnableControllerInterrupts() and Can_DisableControllerInterrupts(), Can_SelfTestChannel().
Source of Error	When the above-mentioned APIs are invoked with out-of- range Controller Id.
Sl. No.	**3**
Error Code	CAN_E_UNINIT
Value	0x05
Related API(s)	Can_SetControllerMode(), Can_SetBaudrate(), Can_DisableControllerInterrupts(), Can_EnableControllerInterrupts(), Can_Write(), Can_MainFunction_Write(), Can_MainFunction_Read(), Can_MainFunction_BusOff(), Can_MainFunction_Mode(), Can_SelfTestChannel().
Source of Error	When the above-mentioned APIs are invoked without initialization of CAN Driver Component.
Sl. No.	**4**
Error Code	CAN_E_TRANSITION
Value	0x06
Related API(s)	Can_Init(),Can_DeInit, Can_SetBaudrate and Can_SetControllerMode(), Can_SelfTestChannel()
Source of Error	When the above-mentioned APIs are invoked with invalid transition for the Controller current Mode.
Sl. No.	**5**
Error Code	CAN_E_PARAM_POINTER
Value	0x01
Related API(s)	Can_GetVersionInfo(), Can_Write(), Can_GetControllerErrorState, Can_GetControllerMode and Can_Init()
Source of Error	When the API is invoked with NULL pointer.

Table 5-16 DET Errors of CAN Driver Component (2/2)

Sl. No.	6
Error Code	CAN_E_PARAM_DATA_LENGTH
Value	0x03
Related API(s)	Can_Write()
Source of Error	When the above-mentioned API is invoked with wrong DLC value.
Sl. No.	**7**
Error Code	CAN_E_DATALOST
Value	0x01
Related API(s)	CAN_GLOBAL_ISR(),CAN_GLOBAL_CAT2_ISR (), CAN_CHANNEL_ISR(), CAN_CHANNEL_CAT2_ISR() and Can_MainFunction_Read()
Source of Error	When the CAN message received is lost.
Sl. No.	**8**
Error Code	CAN_E_INVALID_DATABASE
Value	0xEF
Related API(s)	Can_Init()
Source of Error	When the above-mentioned API is invoked with invalid database address.
Sl. No.	**9**
Error Code	CAN_E_PARAM_BAUDRATE
Value	0x08
Related API(s)	Can_SetBaudrate()
Source of Error	When the above-mentioned API is invoked with invalid baudrate value.
Sl. No.	**10**
Error Code	CAN_RAMTEST_E_INITIALIZED
Value	0x0B
Related API(s)	Can_RAMTest()
Sl. No.	**11**
Error Code	CAN_RAMTEST_E_OUT_OF_RANGE
Value	0x0C
Related API(s)	Can_RAMTest()

5.6.2 CAN Driver Component Production Errors

The Table 5-17 contains the DEM errors reported by CAN Driver Component. These are the hardware errors reported during runtime.

Table 5-17 DEM Errors of CAN Driver Component

Sl. No.	1
Error Code	CAN_E_TIMEOUT_FAILURE
Related API(s)	Can_Init(), Can_DeInit(), Can_SelfTestChannel(), Can_SetIcomConfiguration(), Can_RAMTest().
Source of Error	DEM error is raised when any infinite loop is set in code, or register value write is failed.
Sl. No.	**2**
Error Code	CAN_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	CAN_CHANNEL_ISR(), CAN_CHANNEL_CAT2_ISR(), CAN_GLOBAL_ISR(), CAN_GLOBAL_CAT2_ISR()
Source of Error	DEM error is raised when unintended interrupt error occurs.

5.7 CAN Driver Component Runtime Errors

The following table contains the Runtime errors reported by CAN Driver Component. These errors are reported to Default Error Tracer Module when the CAN Driver Component APIs are invoked in time and the CAN Driver Component does not meet the requirement to execute the API

Table 5-18 Runtime Errors of CAN Driver Component

Sl. No.	Description
Error Code	CAN_E_DATALOST
Value (hex)	0x01
Related API(s)	Can_MainFunction_Read
Source of Error	When API service is called if Received CAN message is lost

5.8 Memory Organization

The following Table 5-19 and Table 5-20 depict a typical memory organization, which must be met for proper functioning of CAN Driver Component software.

Table 5-19 ROM sections of the CAN Driver

Section Name	Alignment (*1)	Description
CAN_PUBLIC_CODE_ROM	-	API(s) of CAN Driver Component that can be located in code memory.
CAN_PRIVATE_CODE_ROM	-	Internal functions of CAN Driver Component code that can be located in code memory.
CAN_FAST_CODE_ROM	-	Interrupt functions of CAN Driver Component code that can be located in code memory.
CONST_ROM_8BIT	-	This section is related to the static constant structure that can be located in code memory.
CAN_CFG_DBTOC_UNSPECIFIED	-	This section consists of CAN Driver Component database table of contents generated by the CAN Driver Component Generation Tool. It can be located in code memory.
CAN_CFG_DATA_8BIT	-	This section consists of CAN Driver Component constant configuration structures. It can be located in code memory.
CAN_CFG_DATA_UNSPECIFIED	-	This section consists of CAN Driver Component constant configuration structures. It can be located in code memory.
CAN_APPL_CODE_ROM	-	This section consists of CAN Driver Component code that to be used references on call back function pointers.

Note:
*1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

Table 5-20 RAM sections of the CAN Driver

Section Name	Alignment (*1)	Description
RAM_UNSPECIFIED	-	This section consists of the global RAM variables used internally by CAN Driver Component. It can be located in data memory.
RAM_32BIT	-	This section consists of the global RAM variables of 32-bit size used internally by CAN Driver Component. It can be located in data memory.
NOINIT_RAM_UNSPECIFIED	-	This section consists of the global RAM pointer variables used internally by CAN Driver Component. It can be located in data memory.
RAM_8BIT	-	This section consists of the global RAM variables of 8-bit size used internally by CAN Driver Component. It can be located in data memory.
RAM_1BIT	-	This section consists of the global RAM variables of 1-bit size used internally by CAN Driver Component. It can be located in data memory.
NOINIT_RAM_1BIT	-	This section consists of the global RAM variables of 1-bit size used internally by CAN Driver Component. It can be located in data memory.
NOINIT_RAM_8BIT	-	This section consists of the global RAM variables of 8-bit size used internally by CAN Driver Component. It can be located in data memory.
NOINIT_RAM_32BIT	-	This section consists of the global RAM variables of 32-bit size used internally by CAN Driver Component. It can be located in data memory.

Note:

*1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

5.9 Device-Specific Information

5.9.1 Interaction Between the User and CAN Driver Component

The details of the services supported by the CAN Driver Component to the upper layers users and the mapping of the channels to the hardware units is provided in the following sections:

5.9.1.1 Channel Mapping

The channel setting does not depend on H/W resources.

5.9.1.2 ISR Functions

The Table 5-21 provide the list of handlers corresponding to the hardware unit ISR(s) in CAN Driver Component. The user should configure the ISR functions mentioned below:

Table 5-21 ISR Handler Addresses

Interrupt Source	Name of the ISR Function
SPI412 CAN-FD Channel: • CANm transmit interrupt • CANm receive interrupt in a common FIFO mode(RX or GW mode) or TXQ(GW mode) • CANm error interrupt	CAN_CHANNEL_ISR
	CAN_CHANNEL_CAT2_ISR
SPI413 CAN-FD Global: • successful reception interrupt in 8 RX FIFO. • Global error interrupt	CAN_GLOBAL_ISR
	CAN_GLOBAL_CAT2_ISR

5.9.2 Multi-Core / Multi-Instantiation

CAN driver does not support multi-core and multi-instantiation for this device.

5.10 Non-AUTOSAR environment integration

The CAN Driver Components for Renesas R-Car Series, 4th Generation SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the CAN Driver Components to Non-AUTOSAR environment.

5.10.1 Stub modules handling

5.10.1.1 Os

The Os stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\Os

In the AUTOSAR environment, CAN Driver is relying on APIs of GetCounterValue() and GetElapsedValue() provided by the OS module to get the current count value of the specified counter.

StatusType GetCounterValue(CounterType CounterID, TickRefType Value);
StatusType GetCounterValue(CounterType CounterID, TickRefType Value, TickRefType ElapsedValue);

Current Os stub implementation uses SCMT (System Timer) IP to simulate an Os counter. This counter has OsSecondsPerTick ~ 0.0000076s and will always run with sample application. GetCounterValue will return the value of that counter at calling time, GetElaspedValue will return the value between current value of that counter and previously read value.

Non-AUTOSAR users can modify the provided GetCounterValue and GetElapsedValue APIs as needed e.g. with other hardware counter or unique CounterID for each MCAL module.

5.10.1.2 Det

The Det stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\Det

In the AUTOSAR environment, CAN Driver Components uses Det_ReportError and Det_ReportRuntimeError API provided by the DET module to report a development error e.g. CAN Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:

Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)
Std_ ReturnType Det_ReportRuntimeError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

5.10.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture. The SchM (Basic Software Scheduler) stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\Rte

CAN driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when CAN_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user CAN driver as following:

Write SchM functions with prototype as below:

void SchM_Enter_Can_CAN_INTERRUPT_CONTROL_PROTECTION(void);
void SchM_Exit_Can_CAN_INTERRUPT_CONTROL_PROTECTION(void);
void SchM_Enter_Can_CAN_RAM_DATA_PROTECTION(void);
void SchM_Exit_Can_CAN_RAM_DATA_PROTECTION(void);

5.10.1.4 Dem

The Dem stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\Dem

In the AUTOSAR environment, CAN Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error e.g. CAN Driver access failure, CAN frame is lost… The API prototype is as of follow:
Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

5.10.1.5 CanIf

The CanIf stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\CanIf

In the AUTOSAR environment, CanIf is the only upper layer of the CAN Driver Components. It provides a hardware independent interface to the Communication Service layers as specified in Figure 1.1: System Overview of AUTOSAR Architecture.

Non-AUTOSAR users can implement their own CAN stack using the APIs provided by the CAN Driver Component.

5.10.1.6 CanIf_RxIndication

CAN Driver invokes CanIf_RxIndication API provided by the CanIf module when a CAN Rx L-PDU is successfully received after passing all filters and validation checks. The API prototype is as of follow:

void CanIf_RxIndication(const Can_HwType* Mailbox,
const PduInfoType* PduInfoPtr);

Current implementation of this API does nothing.

5.10.1.7 CanIf_TxConfirmation

CAN Driver invokes CanIf_TxConfirmation API provided by the CanIf module when transmission of a CAN TxPDU is successfully processed. The API prototype is as of follow:
void CanIf_TxConfirmation(PduIdType CanTxPduId)

Current implementation of this API does nothing.

5.10.1.8 CanIf_ControllerBusOff

CAN Driver invokes CanIf_ControllerBusOff API provided by the CanIf module when a Controller BusOff event happens in the CAN Controller. The API prototype is as of follow:
void CanIf_ControllerBusOff(uint8 Controller)

Current implementation of this API does nothing.

5.10.1.9 CanIf_CurrentIcomConfiguration

CAN Driver invokes CanIf_CurrentIcomConfiguration API provided by the CanIf module when activation/deactivation Pretended Networking happens in the CAN Controller. The API prototype is as of follow:
void CanIf_CurrentIcomConfiguration(uint8 ControllerId, IcomConfigIdType ConfigurationId, IcomSwitch_ErrorType Error)

Current implementation of this API does nothing.

5.10.1.10 CanIf_ControllerModeIndication

CAN Driver invokes CanIf_ControllerBusOff API provided by the CanIf module when a Controller state transition happens in the CAN Controller. The API prototype is as of follow
void CanIf_ControllerModeIndication (uint8 Controller, Can_ControllerStateType ControllerMode)

Current implementation of this API does nothing.

5.10.1.11 EcuM

The EcuM stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\EcuM

In the AUTOSAR environment, CAN Driver Components uses EcuM_SetWakeupEvent, EcuM_CheckWakeup APIs provided by the EcuM module to report a production wakeup event (e.g check and set). The APIs prototype is as of follow:
void EcuM_CheckWakeup (EcuM_WakeupSourceType wakeupSource)

void EcuM_SetWakeupEvent (EcuM_WakeupSourceType sources)

Current implementation of this APIs does nothing.

5.10.2 Callback function usage

The CAN Driver Component does not provide any callback functions.

5.10.3 Scheduled function usage

In the AUTOSAR environment, CAN Driver Component scheduled functions shall be invoked periodically by the Scheduler Manager (see Figure 1.4: Driver Architecture) implemented by EcuM module. The CAN Driver Component scheduled functions (see Chapter 1.5.3) are listed below:

- Can_MainFunction_Write
- Can_MainFunction_Read
- Can_MainFunction_BusOff
- Can_MainFunction_Mode

Non AUTOSAR user can invoke above API periodically using their own implementation e.g. hardware timer to utilize those APIs functionality.

5.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:
\rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

6.DIO

6.1 Overview

The purpose of this chapter is to describe the information related to DIO Driver Component.

The users of DIO Driver Component shall use this chapter as reference.

This chapter shall be used as reference by the users of DIO Driver Component. The system overview of complete AUTOSAR architecture is shown in the following figure below:

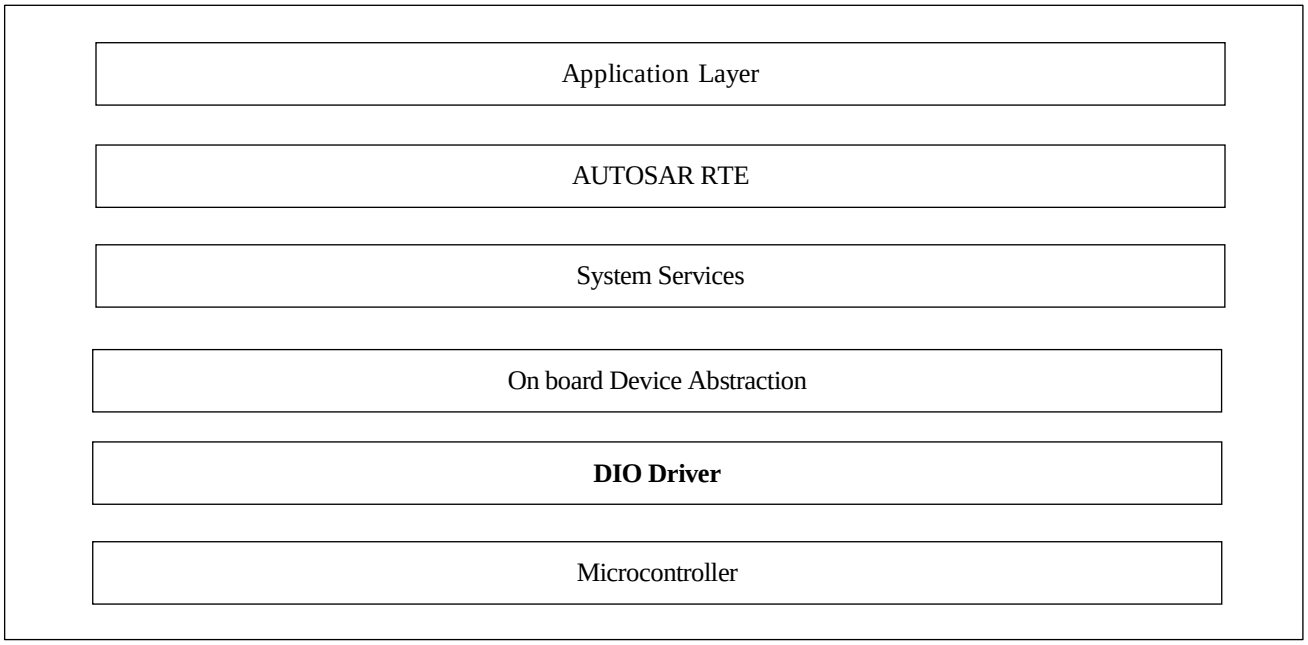


Figure 6-1 System Overview of AUTOSAR Architecture

The DIO Driver is part of the MCAL, the lowest layer of Basic Software in the AUTOSAR environment.

The following figure depicts the DIO Driver as part of layered AUTOSAR MCAL Layer:

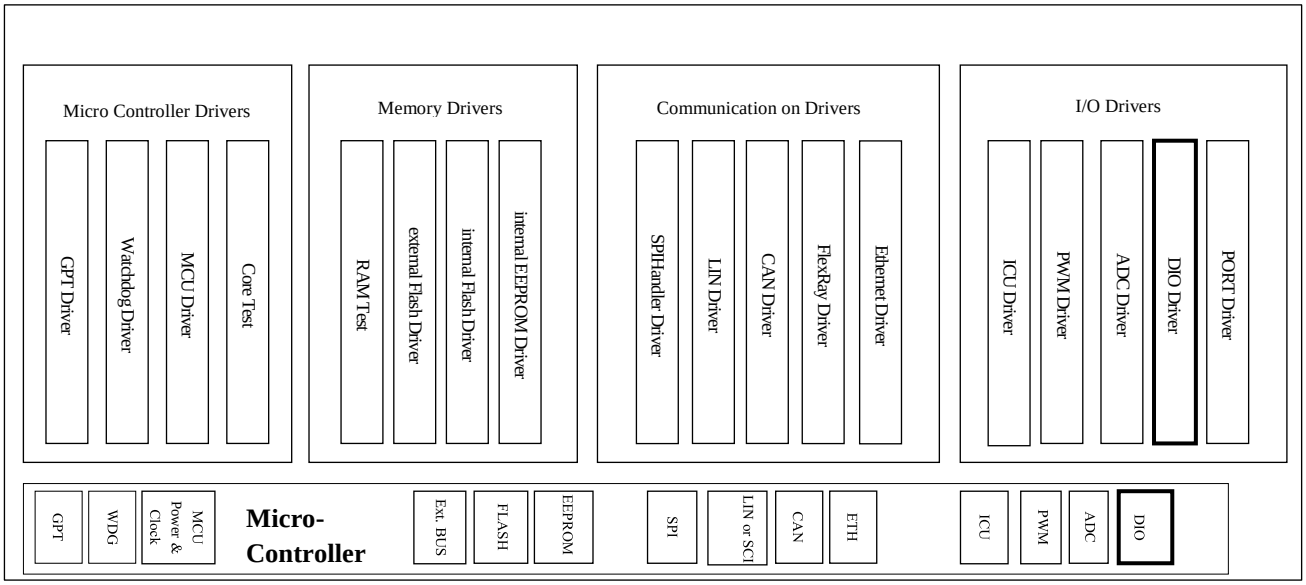


Figure 6-2 System Overview of the DIO Driver in AUTOSAR MCAL Layer

The DIO Driver Component provides the service for reading and writing to Channels, ChannelGroups, and Ports.

The DIO Driver Component comprises two sections, that is, Embedded Software and the Configuration Tool, to achieve scalability and configurability. The DIO Driver Component Code Generation Tool is a command line tool that accepts ECU configuration description file(s) as input and generates source and header file(s). The ECU configuration description is an XML file that contains information about the configuration for Port Pins. The tool generates the Dio_Lcfg.c and Dio_Cfg.h files.

6.2 Forethoughts

6.2.1 General

The following information will help the user use the DIO Driver Component software efficiently:

- The DIO Driver does not enable or disable the ECU or Microcontroller power supply. The upper layer should handle this operation.
- Start-up code is not implemented by the DIO Driver.
- The DIO Driver does not implement any callback notification functions.
- The DIO Driver does not implement any scheduled functions.
- The DIO Driver supports the Read back feature.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.

6.2.2 Preconditions

The following preconditions have to be adhered by the user, for proper functioning of the DIO Driver Component:

- The Dio_Cfg.h and Dio_Lcfg.c files generated by the DIO Driver Component Code Generation Tool have to be linked with the DIO Driver Component source files.
- File Dio_Lcfg.c generated by DIO Driver Component Generation Tool can be compiled and linked independently.
- The application has to be rebuilt, if there is any change in the Dio_Cfg.h and Dio_Lcfg.c file generated by the DIO Driver Component Generation Tool.
- The authorization of the user for calling the software triggering of a hardware reset is not checked in the DIO Driver. This will be the responsibility of the upper layer.
- The user should ensure that DIO Driver Component API requests are invoked with correct input arguments.
- Input parameters are validated only when the static configuration parameter ‘DioDevErrorDetect’ is enabled. Application should ensure that the right parameters are passed while invoking the APIs when DioDevErrorDetect is disabled.
- A mismatch in the version numbers of header and the source files will result in compilation error. User should ensure that the correct versions of the header and the source files are used.
- The DIO functions shall be called only after PORT Driver initialization. Otherwise, the DIO functions will exhibit undefined behavior.
- User/Integrator should ensure that the same Ports/Pins are configured for DIO Driver component.
- The Write Protection for MFIS that used in Exclusive Control feature is implemented in start-up code before executing DIO module. Need to disable The Write Protection for MFIS when Exclusive Control Feature is enabled in order to operate properly.
- The parameter "DioDomainId" is used to set the bus domain (0, 1, 2, and 3). If Users are set to a domain other than 0, Users must execute the Bus Domain Protection Registers setting to ensure that the target registers can be accessed. Please refer to “R-Car V4H Series User’s Manual: Hardware” 18.5.10 Bus Domain Protection for Bus Domain Protection and 7.1.4 Register Configuration for PFC/GPIO registers.

6.2.3 Data Consistency

To support the re-entrance and interrupt services, the AUTOSAR DIO component will ensure the data consistency while accessing its own RAM storage or hardware registers. The DIO component will use SchM_Enter_Dio_<Exclusive Area> and SchM_Exit_Dio_<Exclusive Area> functions. The SchM_Enter_Dio_<Exclusive Area> function is called before the data needs to be protected and SchM_Exit_Dio_<Exclusive Area> function is called after the data is accessed. Dio module will use the following macros:

```
#define DIO_ENTER_CRITICAL_SECTION(Exclusive_Area) \

SchM_Enter_Dio_##Exclusive_Area()

#define DIO_EXIT_CRITICAL_SECTION(Exclusive_Area) \

SchM_Exit_Dio_##Exclusive_Area()
```

The following *Table 6-1 DIO Driver Protected Resource List* shows exclusive area along with scheduler services is used to provide data integrity for shared resources:

Table 6-1 DIO Driver Protected Resource List

Sl. No.	APIs	Exclusive Area Type	Protected Resources
1	Dio_WritePort	DIO_INTERRUPT_CONTROL_PROTECTION	Registers: INDTn, OUTDTn, IOINTSELn, INOUTSELn, OUTDTSELn, OUTDTLn, OUTDTHn
2	Dio_WriteChannel	DIO_INTERRUPT_CONTROL_PROTECTION	Registers: INDTn, OUTDTn, IOINTSELn, INOUTSELn, OUTDTSELn, OUTDTLn, OUTDTHn
3	Dio_FlipChannel	DIO_INTERRUPT_CONTROL_PROTECTION	Registers: INDTn, OUTDTn, IOINTSELn, INOUTSELn, OUTDTSELn, OUTDTLn, OUTDTHn
4	Dio_WriteChannelGroup	DIO_INTERRUPT_CONTROL_PROTECTION	Registers: INDTn, OUTDTn, IOINTSELn, INOUTSELn, OUTDTSELn, OUTDTLn, OUTDTHn
5	Dio_MaskedWritePort	DIO_INTERRUPT_CONTROL_PROTECTION	Registers: INDTn, OUTDTn, IOINTSELn, INOUTSELn, OUTDTSELn, OUTDTLn, OUTDTHn

Note: *n: Port Group*

}

6.3 Architecture Details

The overview of the AUTOSAR DIO software architecture is given in the following *Figure 6-3 DIO Driver Architecture*. The DIO Driver Component accesses the hardware features directly. The upper layers call the functionalities provided by DIO Driver Component:

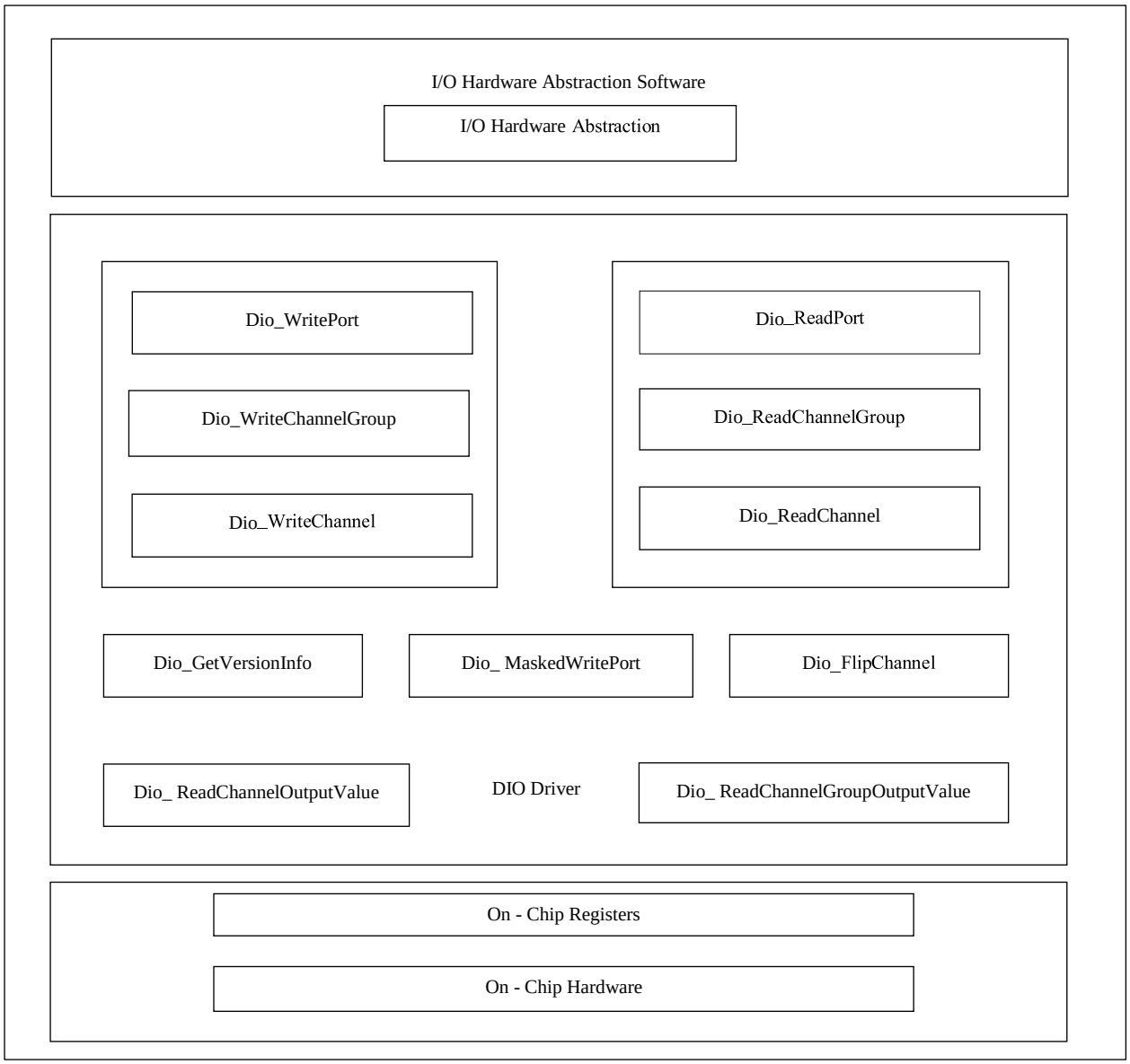


Figure 6-3 DIO Driver Architecture

DIO Driver Component:

The following *Figure 6-4DIO Driver Services* shows the various functionalities as offered by the DIO Driver.

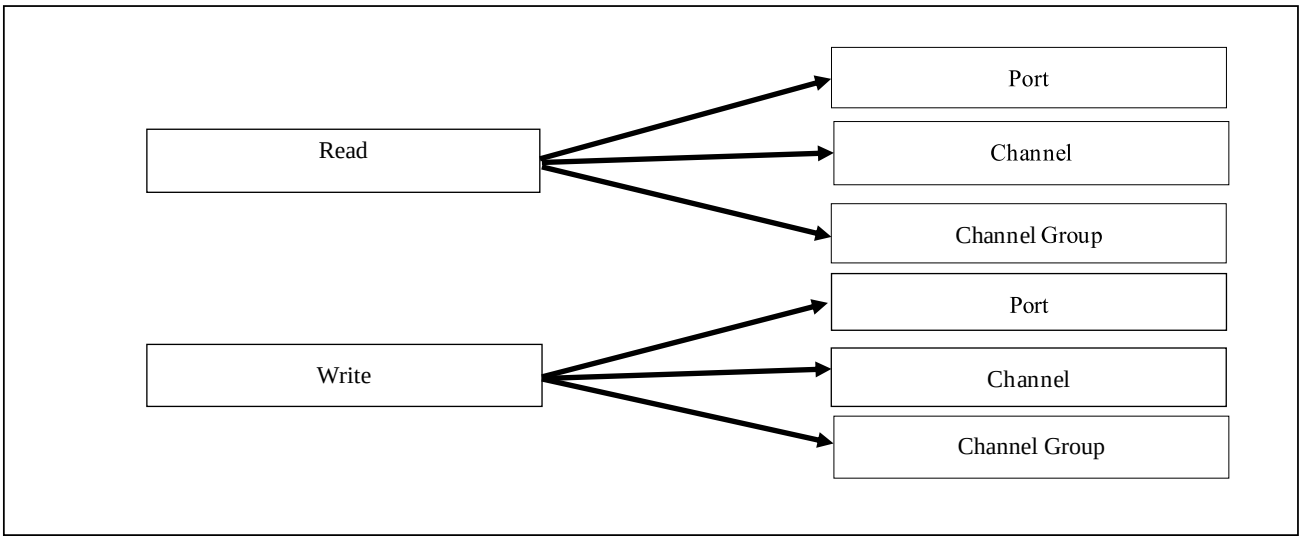


Figure 6-4 DIO Driver Services

The DIO Driver Component is divided into the following sub modules based on the functionality required:

- Port Read and Port Write
- Channel Read and Channel Write
- Channel Group Read and Channel Group Write

Dio_ReadPort:
The levels of all channels of a DIO Port will be read. A bit value '0' indicates the individual channels of the port to physical STD_LOW, and a bit value '1' indicates the individual channels of the port to physical STD_HIGH. The levels of all channels of Port are read simultaneously.

Dio_ReadChannel:
The level of a single DIO Channel is read as either STD_LOW or STD_HIGH.

Dio_ ReadChannelGroup:
The levels of all channels of a DIO Channel Group are read. A bit value '0' indicates the corresponding pin to physical STD_LOW, and a bit value '1' indicates the corresponding channel to physical STD_HIGH.

Dio_WritePort:
The levels of all channels of a Port will be set simultaneously without affecting the functionality of the input channels. A bit value '0' sets the corresponding channel to physical STD_LOW, and a bit value '1', to physical STD_HIGH.

Dio_WriteChannel:
The level of a single DIO Channel is set to STD_HIGH or STD_LOW.

Dio_ WriteChannelGroup:
All adjoining subset of DIO channels of a port are set simultaneously. A bit value '0' sets the corresponding channel to physical STD_LOW, and a bit value '1', to physical STD_HIGH.

Dio_FlipChannel:
The level of a single DIO channel will be set with compliment of current level. That is, if the current value is STD_HIGH then STD_LOW will be set and vice versa, then, the level of a single DIO Channel will be read.

Dio_GetVersionInfo:
The Dio_GetVersionInfo function shall return the version information of this module. The version information includes module ID, Vendor ID and Vendor-specific version numbers.

Dio_ MaskedWritePort:
The Dio_MaskedWritePort function shall set the specified value for the channels in the specified port, if the corresponding bit in Mask is '1'.

Dio_ReadChannelOutputValue:
The Dio_ReadChannelOutputValue function shall read and return the output data value of the specified channel of Port register.

Dio_ReadChannelGroupOutputValue:
The Dio_ReadChannelGroupOutputValue function shall read and return the output data value of the specified ChannelGroup of Port register.

6.4 DIO Driver Component Header and Source File Description

This section explains the DIO Driver Component's source and header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by DIO Driver Component Code Generation Tool:

- Dio_Cfg.h

The C source file generated by DIO Driver Component Code Generation Tool:

- Dio_Lcfg.c

The DIO Driver Component C header files and Component source files:
Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" - section *"3.3.1.3 Folder Structure"*.
The Stub C header files:
Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" - section *"3.2.9 Stubs File"*.

6.5 Application Programming Interface

This section explains the Data types and APIs provided by the DIO Driver Component to the Upper layers.

6.5.1 Imported Types

This section explains the Data types imported by the DIO Driver Component and lists the dependency on other modules.

6.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed:

- Std_VersionInfoType
- Std_ReturnType

6.5.1.2 Other Module Types

The DIO Driver Component module does not depend on other modules. Hence other module types are not imported.

6.5.2 Type Definitions

This section explains the type definitions of DIO Driver Component according to AUTOSAR Specification. Refer to "*Autosar Specification of DIO Driver R19-11*" for more type definitions.

6.5.2.1 Dio_ChannelType

The following *Table 6-2* shows explanation of Dio_ChannelType.

Table 6-2 Dio_ChannelType

Name:	Dio_ChannelType
Type:	uint16
Range:	0 to 65535
Description:	Numeric ID of a DIO channel.

6.5.2.2 Dio_PortType

The following *Table 6-3* shows explanation of Dio_PortType.

Table 6-3 Dio_PortType

Name:	Dio_PortType
Type:	uint8
Range:	0 to 255
Description:	Numeric ID of a DIO port.

6.5.2.3 Dio_LevelType

The following *Table 6-4* shows explanation of Dio_LevelType.

Table 6-4 Dio_LevelType

Name:	Dio_LevelType
Type:	uint8
Range:	0 to 255
Description:	These are the possible levels a DIO channel can have (input or output)

6.5.2.4 Dio_PortLevelType

The following *Table 6-5* shows explanation of Dio_PortLevelType.

Table 6-5 Dio_PortLevelType

Name:	Dio_PortLevelType
Type:	Uint32
Range:	0..4294967295
Description:	Type definition for Dio_PortLevelType used by the DIO APIs. If the µC owns ports of different port widths (e.g. 4, 8,16, 32...Bit) Dio_PortLevelType inherits the size of the largest port.

6.5.2.5 Dio_PortGroup

The following *Table 6-6* shows explanation of Dio_PortGroup.

Table 6-6 Dio_PortGroup

Name:	Dio_PortGroup		
Type:	struct		
Element:	**Type**	**Name**	**Explanation**
	volatile uint32	pPortAddress	Address of the IOINTSEL register
	uint32	ulModeMask	Useable pin position is 1 value in 32bit. When useable pin is 0-6, this parameter is 0x0000007F
Description:	This structure is used to access Dio Port Group.		

6.5.2.6 Dio_PortChannel

The following *Table 6-7* shows explanation of Dio_PortChannel.

Table 6-7 Dio_PortChannel

Name:	Dio_PortChannel		
Type:	struct		
Element:	**Type**	**Name**	**Explanation**
	uint32	ulMask	Bit position of the Channel
	uint8	ucPortIndex	PortGroup Index in the array Dio_GaaPortGroup[ ].
Description:	This structure is used to access DIO Channel.		

6.5.2.7 Dio_ChannelGroupType

The following *Table 6-8* shows explanation of Dio_ChannelGroupType.

Table 6-8 Dio_ChannelGroupType

Name:	Dio_ChannelGroupType

Type:	struct		
Element:	**Type**	**Name**	**Explanation**
	uint32	ulMask	Bit positions of the Channel Group
	uint8	ucOffset	Bit Position of Channel Group from LSB
	uint8	ucPortIndex	PortGroup Index in the array Dio_GaaPortGroup[ ].
Description:	This structure is used to access DIO Port Channel Group		

6.5.2.8 Dio_HwRegOffsetType

The following *Table 6-9* shows explanation of Dio_HwRegOffsetType.

Table 6-9 Dio_HwRegOffsetType

Name:	Dio_HwRegOffsetType		
Type:	struct		
Element:	**Type**	**Name**	**Explanation**
	uint32 *	ulIOINTSEL	value of IOINTSEL register offset
	uint32 *	ulINOUTSEL	value of INOUTSEL register offset
	uint32 *	ulOUTDT	value of OUTDT register offset
	uint32 *	ulINDT	value of INDT register offset
	uint32 *	ulOUTDTSEL	value of OUTDTSEL register offset
	uint32 *	ulOUTDTH	value of OUTDTH register offset
	uint32 *	ulOUTDTL	value of OUTDTL register offset
Description:	Structure for DIO HW Register Offset.		

6.5.2.9 Dio_HwFuncTableType

The following *Table 6-10* shows explanation of Dio_HwFuncTableType.

Table 6-10 Dio_HwFuncTableType

Name:	Dio_HwFuncTableType		
Type:	struct		
Element:	**Type**	**Name**	**Explanation**
	P2FUNC	pHwReadPort	Pointer to Read function
	P2FUNC	pHwWritePort	Pointer to Read function
	P2FUNC	pHwFlipChannel	Pointer to Flip function
	P2FUNC	pHwReadOutputValue	Pointer to Read value function
Description:	Structure for DIO HW-dependent function pointer table.		

6.5.2.10 **Dio_HwConfigType**

The following *Table 6-11* shows explanation of Dio_HwConfigType.

Table 6-11 Dio_HwConfigType

Name:	Dio_HwConfigType		
Type:	struct		
Element:	**Type**	**Name**	**Explanation**
	Dio_HwRegOffsetType *	pHwRegOffset	value of 32bit HW Register Offset
	Dio_HwFuncTableType *	pHwFunc	table of HW Functions.
Description:	Structure for DIO HW Configuration.		

6.5.2.11 **Dio_ExclusiveType**

The following *Table 6-12* shows explanation of Dio_ExclusiveType.

Table 6-12 Dio_ExclusiveType

Location:	Dio_LTTypes.h	
Type:	Enumeration	
Range:	DIO_GET_EXCLUSIVE = 0x00	0 (Get resource from another CPU)
	DIO_RELEASE_EXCLUSIVE =0x01	1 (Release resource)
Description:	This is the input data type of Dio_ExclusiveControl. Enumeration for selection of exclusive control	

6.5.3 **Function Definitions**

The following *Table 6-13* shows list of APIs Provided by the DIO Driver Component.

Table 6-13 APIs Provided by the DIO Driver Component

Sl.No	**APIs**
1.	Dio_ReadPort
2.	Dio_WritePort
3.	Dio_ReadChannel
4.	Dio_WriteChannel
5.	Dio_FlipChannel
6.	Dio_GetVersionInfo
7	Dio_ReadChannelGroup
8	Dio_WriteChannelGroup
9	Dio_MaskedWritePort
10	Dio_ReadChannelOutputValue
11	Dio_ReadChannelGroupOutputValue

6.5.3.1 **Renesas Original API**

6.5.3.1.1 Dio_ReadChannelOutputValue

The following *Table 6-14* explains the API of Dio_ReadChannelOutputValue.

Table 6-14 Dio_ReadChannelOutputValue

Function Name:	Dio_ReadChannelOutputValue		
Syntax:	FUNC(Dio_LevelType, DIO_PUBLIC_CODE) Dio_ReadChannelOutputValue (const Dio_ChannelType LddChannelId)		
Service ID:	0x14		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	ID of DIO channel	
		Range:	0.. DIO_MAXNOOFCHANNEL-1
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Dio_LevelType	The output value of the specified channel of Port register	
		Range:	0: Low level output 1: High level output
Description:	This service reads the output value of the specified channel of Port register. Structure type of Dio_PortGroup and Dio_PortChannel are used in this API.		
Preconditions:	None		
Remarks:	This API is available only if the pre-compile option DIO_READ_CHANNEL_OUTPUT_VALUE_API is STD_ON.		
DET/DEM Errors handled:	DIO_E_PARAM_INVALID_CHANNEL_ID		Parameter (In) LddChannelId is out of range.

6.5.3.1.2 Dio_ReadChannelGroupOutputValue

The following *Table 6-15* explains the API of Dio_ReadChannelGroupOutputValue

Table 6-15 Dio_ReadChannelGroupOutputValue

Function Name:	Dio_ReadChannelGroupOutputValue		
Syntax:	FUNC(Dio_PortLevelType, DIO_PUBLIC_CODE) Dio_ReadChannelOutputValue (P2CONST(Dio_ChannelGroupType, DIO_VAR, DIO_CONST) ChannelGroupIdPtr)		
Service ID:	0x15		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	ChannelGroupIdPtr	Pointer to ChannelGroup	
		Range:	Pointer of handles of DioChannelGroup
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Dio_PortLevelType	The output value of the specified ChannelGroup of Port Register.	
		Range:	0..0xFFFF
Description:	This service reads the output value of the specified ChannelGroup of Port Register. The return value is a subset of the adjoining bits of a port by masking the channel group and aligning to the LSB. Structure type of Dio_PortGroup and Dio_ChannelGroupType are used in this API.		
Preconditions:	DIO Driver must be initialized.		
Remarks:	This API is available only if the pre-compile option DIO_READ_CHANNEL_GROUP_OUTPUT_VALUE_API is STD_ON.		
DET/DEM Errors handled:	DIO_E_PARAM_POINTER	Parameter (In) ChannelGroupIdPtr is NULL pointer.	
	DIO_E_PARAM_INVALID_GROUP	Any Channel Group is not configured.	

6.5.4 Preemption of APIs

The following *Table 6-16* shows Preemption Table of APIs of the DIO Driver.

Table 6-16 Preemption Table of APIs of the DIO Driver

	Dio_ReadPort	Dio_WritePort	Dio_ReadChannel	Dio_WriteChannel	Dio_FlipChannel	Dio_ReadChannelGroup	Dio_WriteChannelGroup	Dio_GetVersionInfo	Dio_MaskedWritePort	Dio_ReadChannelOutputValue	Dio_ReadChannelGroupOutputValue
Dio_ReadPort	√										
Dio_WritePort	√	√									
Dio_ReadChannel	√	√	√								
Dio_WriteChannel	√	√	√	√							
Dio_FlipChannel	√	√	√	√	√						
Dio_ReadChannelGroup	√	√	√	√	√	√					
Dio_WriteChannelGroup	√	√	√	√	√	√	√				
Dio_GetVersionInfo	√	√	√	√	√	√	√	√			
Dio_MaskedWritePort	√	√	√	√	√	√	√	√	√		
Dio_ReadChannelOutputValue	√	√	√	√	√	√	√	√	√	√	
Dio_ReadChannelGroupOutputValue	√	√	√	√	√	√	√	√	√	√	√

-: cannot be invoked at the same time

√: can be invoked at the same time

Note: Operating conditions when DIO_CRITICAL_SECTION_PROTECTION = STD_ON

6.6 Development and Production Errors

In this section, the development errors reported by the DIO Driver Component are tabulated. The development errors will be reported only when the pre-compiler option ‘DIO_DEV_ERROR_DETECT’ is enabled in the configuration. The DIO Driver Component does not support any production errors.

6.6.1 DIO Driver Component Development Errors

The following *Table 6-17* contains the DET errors reported by DIO Driver Component. These errors are reported to Development Error Tracer Module when the DIO Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 6-17 DET Errors of DIO Driver Component

Sl. No.	1
Error Code	DIO_E_PARAM_INVALID_CHANNEL_ID
Value (hex)	0x0A
Related API(s)	Dio_ReadChannel, Dio_WriteChannel, Dio_FlipChannel, Dio_ReadChannelOutputValue
Source of Error	When the above-mentioned APIs are invoked with invalid Channel ID.
Sl. No.	**2**
Error Code	DIO_E_PARAM_INVALID_PORT_ID
Value (hex)	0x14
Related API(s)	Dio_ReadPort, Dio_WritePort, Dio_MaskedWritePort
Source of Error	When the above-mentioned APIs are invoked with invalid Port ID.
Sl. No.	**3**
Error Code	DIO_E_PARAM_INVALID_GROUP
Value (hex)	0x1F
Related API(s)	Dio_ReadChannelGroup, Dio_WriteChannelGroup, Dio_ReadChannelGroupOutputValue
Source of Error	When the above-mentioned APIs are invoked with invalid ChannelGroup ID.
Sl. No.	**4**
Error Code	DIO_E_PARAM_POINTER
Value (hex)	0x20
Related API(s)	Dio_ReadChannelGroup, Dio_WriteChannelGroup, Dio_GetVersionInfo, Dio_ReadChannelGroupOutputValue
Source of Error	When the above-mentioned API is invoked with the NULL parameter.

6.6.2 DIO Driver Component Production Errors

In this following *Table 6-18*, the DEM errors identified in the DIO Driver Component are listed. DIO Driver Component reports these errors to DEM by invoking Dem_SetEventStatus API. This API is invoked when the processing of the given API request fails.

Table 6-18 DEM Errors of DIO Driver Component

Sl. No.	1
Error Code	DIO_E_GET_CONTROL_FAILURE
Value	DemConf_DemEventParameter_<short name of DemEventParameter>
Related API(s)	Dio_WriteChannel, Dio_WritePort, Dio_WriteChannelGroup, Dio_FlipChannel, Dio_MaskedWritePort
Source of Error	The Exclusive Control functionality DEM error will be reported, if Dio Driver can’t get control registers.

6.7 DIO Driver Component Runtime Errors

None.

6.8 Memory Organization

The following *Table 6-19* depicts a typical memory organization, which must be met for proper functioning of DIO Driver Component software.

Table 6-19 ROM Sections of the DIO Driver

Section Name	Alignment (*1)	Description
DIO_PUBLIC_CODE_ROM	-	This section contains codes which belong to the API functions of the DIO Driver.
DIO_PRIVATE_CODE_ROM	-	This section contains codes which belong to the internal functions of the DIO Driver.
DIO_CFG_DATA_UNSPECIFIED	-	This section contains post-build config miscellaneous data of the DIO Driver.

6.9 Device-Specific Information

It supports the following devices:
Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" 5.1 Product

6.9.1 Multi-Core / Multi-Instantiation

DIO driver does not support multi-core and multi-instantiation for this device.

6.9.2 Interaction between the User and DIO Driver Component

6.9.2.1 Channel Mapping

DIO module does not support channel mapping.

6.9.2.2 ISR Function

DIO module does not support implementation of ISR.

6.10 Non-AUTOSAR Environment Integration

6.10.1 Stub Modules Handling

6.10.1.1 Det

Det (Default Error Tracer) provides functionality to support error detection and tracing of errors during the development and runtime of Software Components and other Basic Software Modules. For this purpose the Default Error Tracer receives and evaluates error messages from these components and modules.

DIO Driver needs DET module to report the development errors when DIO_DEV_ERROR_DETECT configured as STD_ON. With Non-AUTOSAR environment, user needs to write DET stub to use DIO Driver as following:

- Writing DET function with prototype as below:

Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId);

- Create Det_Error structure.

Example: refer Det.c, Det.h in “external\rel\common\generic\stubs\19_11\Det”.

6.10.1.2 **Basic Software Scheduler**

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture. DIO driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when DIO_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user DIO driver as following:
Write SchM functions with prototype as below:

SchM_Enter_Dio_DIO_INTERRUPT_CONTROL_PROTECTION ()

SchM_Exit_Dio_DIO_INTERRUPT_CONTROL_PROTECTION ()

Example: refer SchM_Dio.h, SchM_Dio.c in *“external\rel\common\generic\stubs\19_11\Rte\”*

6.10.1.3 **Dem**

The Dem stub files are organized in the following folder:
\external\rel\common\generic\stubs\<Autosar version>\Dem
In the AUTOSAR environment, DIO Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production. The API prototype is as of follow:
Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)
Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.
Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

6.10.2 Callback Function Usage

None.

6.10.3 Scheduled Function Usage

None.

6.10.4 Interrupt Handling Usage

None.

7. PORT

7.1 Overview

The purpose of this chapter is to describe the information related to PORT Driver Component.
The users of PORT Driver Component shall use this chapter as reference.
This chapter describes the common features of PORT Driver Component. This chapter is intended for the developers of ECU software using Application Programming Interfaces provided by AUTOSAR. The PORT Driver Component provides the following services:

• PORT Driver Component initialization
• PORT Pin Direction changing

The following diagram shows the system overview of the AUTOSAR Architecture. The PORT Driver initializes all the channels required to produce PORT outputs.

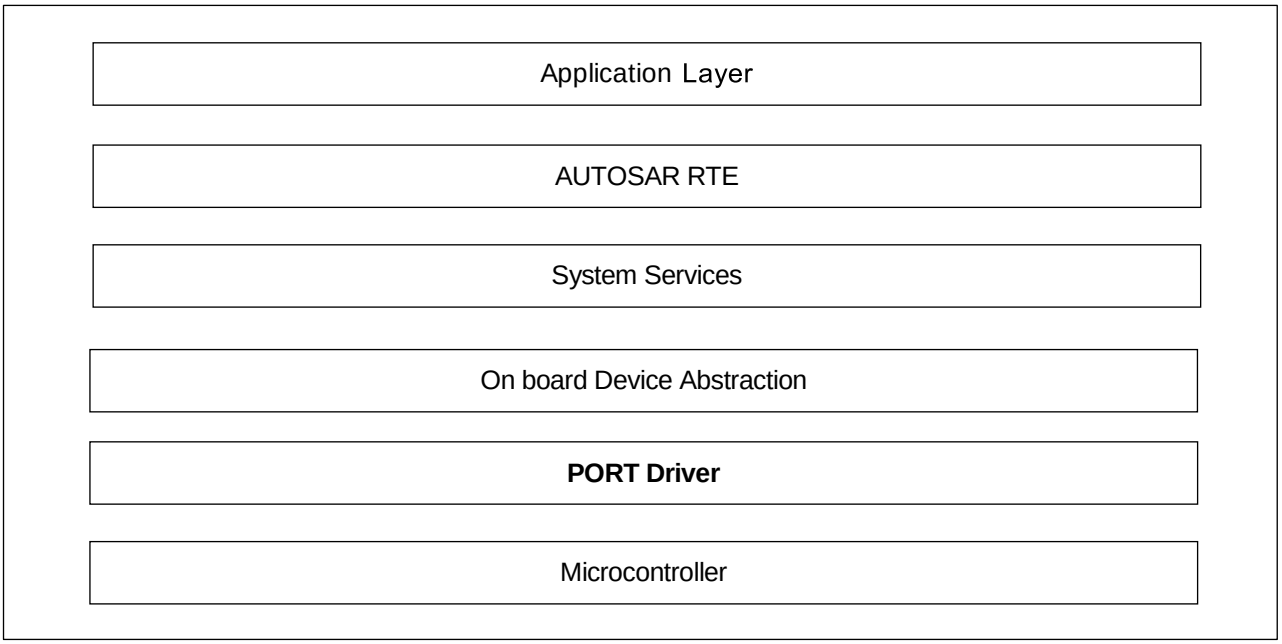


Figure 7-1 System Overview of AUTOSAR Architecture

The PORT Driver Component comprises two sections, that is, embedded software and the configuration tool to achieve scalability and configurability. The PORT Driver Component Code Generation Tool is a command line tool that accepts ECU configuration description files as input and generates C Source and C Header files. The configuration description is an ARXML file that contains information about the configuration for PORT channels. The tool generates Port_Cfg.h and Port_PBcfg.c files.

The Figure 7-1 in the following page depicts the PORT Driver as part of layered AUTOSAR MCAL Layer:

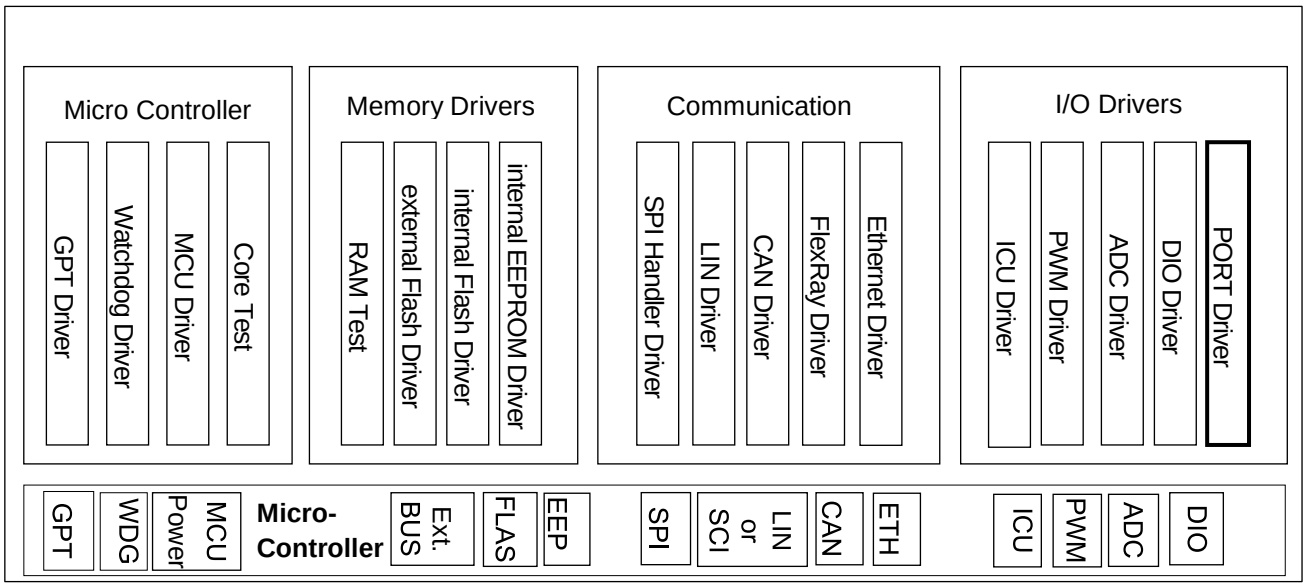


Figure 7-2 System Overview of the PORT Driver in AUTOSAR MCAL Layer

7.2 Forethoughts

7.2.1 General

The following information will help the user use the PORT Driver Component software efficiently:

- The PORT Driver Component does not enable or disable the ECU or Microcontroller power supply. The upper layer should handle this operation.
- Start-up code is not implemented by the PORT Driver Component.
- PORT Driver Component does not implement any callback notification functions.
- PORT Driver Component does not implement any scheduled functions.
- The PORT Driver Component is restricted to Post Build only.
- The authorization of the user to call the software triggering of a hardware reset is not checked in the PORT Driver Component. This is the responsibility of the upper layer.
- The PORT Driver Component supports the setting of Chattering Prevention. To figure out the different port filter arrangements, the device of User Manual should be taken as reference. If no configuration of a certain port filter is made in this Port Module, the device-specific default settings will take effect on this filter.
- The value of unused pins in Port registers is undefined.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- All production errors will be reported to DEM by using the Dem_SetEventStatus API provided by DEM.
- Level inversion is not supported by ‘PORT Driver Component’. The default value is set (not inverted).
- The PORT Driver does not have the API support to read the status of Port pins or Port registers. Hence PORT Driver does not support ‘Read back’ feature.
- The user shall take care if a particular mode is valid for a particular port pin while calling Port_SetPinMode API.
- The CANFD7 functionality will be reflected instead of CANFD5 in the PortGroup2/PortPin2, the PortGroup2/PortPin3 when the parameter “PortPinInitialMode” is configured as:
  PortGroup2/PortPin2/PortPinInitialMode: CANFD5_TX_ALT1_OUT_GPSR2_BIT2_IP0SR2_BIT8
  PortGroup2/PortPin3/PortPinInitialMode: CANFD5_RX_ALT1_IN_GPSR2_BIT3_IP0SR2_BIT12.

7.2.2 Preconditions

The following preconditions have to be adhered by the user, for proper functioning of the PORT Driver Component:

- The Port_PBcfg.c and Port_Cfg.h files generated by the PORT Driver Component Code Generation Tool must be compiled and linked along with PORT Driver Component source files.
- The application has to be rebuilt, if there is any change in the Port_Cfg.h file generated by the PORT Driver Component Generation Tool.
- File Port_PBcfg.c generated for single configuration set using PORT Driver Component Code Generation Tool should be compiled and linked independently.
- Symbolic names for all Port Pins are generated in Port_Cfg.h file which can be used as parameters for passing to PORT Driver Component APIs.
- For convenience with hardware design, Renesas design Port Pin container configuration following each Port Group. Therefore, current symbolic macro definition is generated in Port_Cfg.h according rule:
  PortConf_PortGroupX_PortPinY
  where X, Y is the number of PortGroup and PortPin
  e.g: symbolic macro for PortPin0 container inside PortGroup0
  #define PortConf_PortGroup0_PortPin0 (Port_PinType)0
- The PORT Driver Component needs to be initialized for all Port Pins before doing any operation on Port Pins. The Port_Init() API shall also be called after a reset in order to reconfigure the Port Pins of the microcontroller. If PORT Driver Component is not initialized properly, the behavior of Port Pins may be undetermined.
- The user should ensure that PORT Driver Component API requests are invoked with correct input arguments.
- The PORT Driver Component working with GPIO registers, hence user should ensure that MCU Driver is supplies clock for GPIO to make port pin that has mode other than alternative function works properly.
- Input parameters are validated only when the static configuration parameter ‘PORT_DEV_ERROR_DETECT’ is enabled. Application should ensure that the right parameters are passed while invoking the APIs when PORT_DEV_ERROR_DETECT is disabled.
- Values for production code Event Ids should be assigned externally by the configuration of the DEM.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used. The version information is described in the 1.1 Supported MCAL Product Release Version.
- The PORT Driver Component APIs, except Port_GetVersionInfo API, which are intended to operate on

Port Pins shall be called only after PORT Driver Component is initialized by invoking Port_Init() API. Otherwise Port Pin functions will exhibit undefined behavior.

- The API Port_SetPinDirection should be configured these params “PortSetPinDirectionApi and PortPinDirectionChangeable for any PortGroup/PortPin” at the same time to guarantee the operation of PORT Driver Component.

- All Port Pins and the functions should be configured by the Port configuration tool. It is the user/integrator’s responsibility to ensure that the same Port/Port Pin is not accessed/configured in parallel by different entities in the same system.
- The Write Protection for MFIS that used in Exclusive Control feature was implemented in start-up code before executing PORT module.

- According to caution note HWUM of R-Car V4H sections “7.3.2 Function Setting for Multiplexed Pins”. The API Port_SetPinMode should only activate one single pin to one given peripheral input function. Do not activate the same input function on multiple pins at the same time.
  - Example: Mode HSCK3 should not be configured respectively “PortGroup1_PortPin25, HSCK3_ALT0_IN_OUT” and “PortGroup1_PortPin3, HSCK3_ALT1_IN_OUT” in run-time Port_SetPinMode.

- The API Port_SetPinMode should be invoke again in peripheral function when direction of pins are changed difference with direction of mode in runtime.
  - Example:

    Case1: Mode RX
    - Step 1: direction of pin A is IN and mode of pin A is RX.
    - Step 2: Invoke the API Port_SetPinDirection to change pin A to OUT
    - In of this case, direction of pin A is incorrect as mode is RX.
    - Step 3: Invoke Port_SetPinMode for pin A as mode RX to keep consistency between direction of pin with direction of mode.

    Case 2: MSPInSC (for S4) or TSNn/AVBn (for V4H) change the pin from slave mode to master mode.
    - Step 1: direction of pin A is IN (Slave mode) and mode of pin A is MSPInSC (for S4) or TSNn/AVBn (for V4H).
    - Step 2: Invoke the API Port_SetPinDirection to change pin A to OUT.
    - In this case, the pin is not changed from slave mode to master mode as direction OUT of pin. User must consider to conduct Step3.
    - Step 3: Invoke Port_SetPinMode for pin A to change from slave mode to master mode MSPInSC (for S4) or TSNn/AVBn (for V4H) as direction OUT.

- The API Port_SetToAlternateMode does not support change direction as ALT mode. So user must consider to control direction of ALT mode.
- According to HWM of R-Car V4H sections “7.2.10 Module Select Register (MODSELn)”. The API Port_SetToAlternateMode does not support to correct MODSEL. So mode I2C should not be set in this API. User must consider to use The API Port_SetPinMode instead of The API Port_SetToAlternateMode for these modes.
- According to HWM of R-Car V4H attached “004_R-CarV4H_pin_function.xlsx“. The users shall consider available port pins to configure corresponding PortPinX and PortChatteringFilterInputX containers

according to following table:

Table 7-1 Available port pins

GPIO group	RCar V4H
GP0_n	n= 0..18
GP1_n	n= 0..28
GP2_n	n= 0..19
GP3_n	n= 0..29
GP4_n	n= 0..24
GP5_n	n= 0..20
GP6_n	n= 0..20
GP7_n	n= 0..20
GP8_n	n= 0..13

- The parameter "PortDomainId" is used to set the bus domain (0, 1, 2, and 3). If Users are set to a domain other than 0, Users must execute the Bus Domain Protection Registers setting to ensure that the target registers can be accessed. Please refer to “R-Car V4H Series User’s Manual: Hardware” 18.5.10 Bus Domain Protection for Bus Domain Protection and 7.1.4 Register Configuration for PFC/GPIO registers.

7.2.3 Data Consistency

To support the re-entrance and interrupt services, the AUTOSAR PORT component will ensure the data consistency while accessing its own RAM storage or hardware registers. The PORT component will use SchM_Enter_Port_<Exclusive Area> and SchM_Exit_Port_<Exclusive Area> functions. The SchM_Enter_Port_<Exclusive Area> function is called before the data needs to be protected, and SchM_Exit_Port_<Exclusive Area> function is called after the data is accessed. Port module will use the following macros:

```
#define PORT_ENTER_CRITICAL_SECTION(Exclusive_Area) \
SchM_Enter_Port_##Exclusive_Area()

#define PORT_EXIT_CRITICAL_SECTION(Exclusive_Area) \
SchM_Exit_Port_##Exclusive_Area()
```

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

Table 7-2 PORT Driver Protected Resource List

Sl. No.	APIs	Exclusive Area Type	Protected Resources
1	Port_SetPinDirection	PORT_INTERRUPT_CONTROL_PROTECTION	Registers: INOUTSELn, OUDTSELn, OUTDTLn, OUTDTHn, OUTDTn, MFISLCKRx
2	Port_RefreshPortDirection	PORT_INTERRUPT_CONTROL_PROTECTION	Registers: INOUTSELn
3	Port_SetPinDefaultDirection	PORT_INTERRUPT_CONTROL_PROTECTION	Registers: INOUTSELn, OUDTSELn, OUTDTLn, OUTDTHn, OUTDTn, MFISLCKRx
4	Port_SetToDioMode	PORT_INTERRUPT_CONTROL_PROTECTION	Registers: GPSRn, IOINTSELn
5	Port_SetToAlternateMode	PORT_INTERRUPT_CONTROL_PROTECTION	Registers: GPSRn
6	Port_SetPinMode	PORT_INTERRUPT_CONTROL_PROTECTION	Registers: INOUTSELn, OUDTSELn, OUTDTLn, OUTDTHn, OUTDTn, MFISLCKRx, GPSRn, IP*SRn, MODSELn, IOINTSELn, PMMRn

Note: n: Port Group, x: Registers

7.3 Architecture Details

The PORT Driver Component accesses the microcontroller Port Pins located in the On-Chip hardware. The basic architecture of the PORT Driver Component is illustrated below:

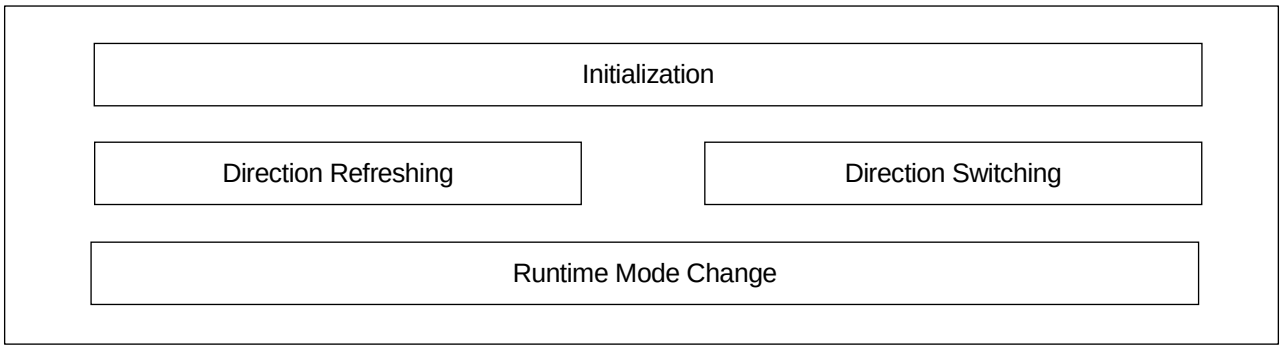


Figure 7-3 PORT Driver Architecture

The PORT Driver Component consists of the following sub modules based on the functionality:

- Port Initialization.
- Port Direction Refreshing.
- Port Pin Direction Switching.
- Port Pin Default Direction Switching.
- Port Pin Mode Changing.
- Port FUSE Monitoring Register.
- Port Unintended Module Stop Check.

Port Initialization:

This sub module provides the Port initialization functionality by the Port_Init() API. This API should be invoked before using any other APIs of PORT Driver Component. Port Initialization includes initializing Port Pin Initial Mode, Port Pin Detection Interrupt, Port Pin Direction, Port Pin Level Value, Port Pin Pull Control, Port Pin Pull Option, Port Pin Out Data Select, Port Pin Polarity Select and Port Chattering configuration.

Port Direction Refreshing:

This sub module provides the Port Direction Refreshing functionality by the Port_RefreshPortDirection() API. In this functionality, the PORT Driver Component refreshes the direction of all configured Port Pins except those configured as ‘Port Pin Direction Changeable during runtime’.

In this functionality, only Direction of Port Pins is refreshed.

Port Pin Direction Switching:

This sub module provides the Port Direction switching functionality at run time by the Port_SetPinDirection() API. In this functionality, the PORT driver Component allows the user to change the direction of Port Pins during runtime.

Port Pin Default Direction Switching:

This sub module provides the Port Default Direction switching functionality at run time by the Port_SetPinDefaultDirection() API. In this functionality, the PORT driver Component allows the user after changing the direction of Port Pins during runtime to the default mode set during the Port_Init().

Port Pin Mode Changing:

This sub module provides the Port Mode change functionality at run time by the Port_SetPinMode() API. In this functionality, the PORT driver Component allows the user to change the mode of Port Pins during runtime.

It returns the mode to the default value by specifying Default for the mode.

This sub module provides the Port Mode change functionality at run time by the Port_SetToDioMode() API. In this functionality, the PORT driver Component allows the user to change the mode of Port Pin to DIO mode during runtime.

This sub module provides the Port Mode change functionality at run time by the Port_SetToAlternateMode() API. In this functionality, the PORT driver Component allows the user to change the mode of Port Pin to alternate mode during runtime.

7.4 PORT Driver Component Header and Source File Description

This section explains the PORT Driver Component's C Source and C Header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by PORT Driver Generation Tool:

- Port_Cfg.h

The C source file generated by PORT Driver Generation Tool:

- Port_PBcfg.c

The PORT Driver Component C header files and Component source files:
Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.3.6.3 Folder Structure.

The Stub C header files:
Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.2.9 Stubs File.

7.5 Application Programming Interface

This section explains the Data types and APIs provided by the PORT Driver Component to the Upper layers.

7.5.1 Imported Types

This section explains the Data types imported by the PORT Driver Component and lists its dependency on other modules.

7.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed: Std_VersionInfoType

7.5.2 Type Definitions

This section explains the type definitions of PORT Driver Component according to AUTOSAR Specification.

7.5.2.1 Port_ConfigType

The **Table 7-3** shows explanation of Port_ConfigType.

Table 7-3 Port_ConfigType

Name:	Port_ConfigType		
Type:	struct		
Element:	**Type**	**Name**	**Explanation**
	uint32	ulStartOfDbToc	Database start value.
	Port_Regs	pPortNumRegs	Pointer to array of structure containing details of Numeric Port Group registers.
	Port_FuncCtrlRegs	pPortNumFuncCtrlRegs	Pointer to array of structure containing details of Numeric Function Control Port Group registers.
	Port_INOUTSELRegs	pPortNumINOUTSELRegs	Pointer to array of structure containing details of INOUTSEL registers.
	Port_FILONOFFRegs	pPortFILONOFFRegs	Pointer to array of structure containing details about FILONOFF registers.
	Port_MODSELRegs	pPortMODSELRegs	Pointer to array of structure containing details about MODSEL registers
	Port_PinsDirChangeable	pPinDirChangeable	Pointer to structure containing details about the pins whose direction can be changed during run time
	Port_PinDioAltChangeableDetails	pPinDioAltModeDetails	Pointer to structure containing details about the port Dio or alternate pin mode
	Port_PinModeChangeableDetails	pPinModeChangeableDetails	Pointer to structure containing details about the port pin mode as pin mode changeable during runtime

	Port_PinModeChangeableGroups	pPinModeChangeableGroups	Pointer to structure containing port groups register as pin mode changeable during runtime.
	Port_MODSELGroupInfo	pMODSELGroupInfo	Pointer to structure containing MODSEL registers as pin mode changeable during runtime
Description:	This is the type of the external data structure containing the initialization data for the PORT Driver Component. The user shall use the symbolic names defined in the PORT Driver Configuration Tool. The configuration of each Port Pin is Microcontroller-specific.		

7.5.2.2 Port_PinType

The **Table 7-4** shows explanation of Port_PinType.

Table 7-4 Port_PinType

Name:	Port_PinType
Type:	uint16
Range:	0 to 65535
Description:	The user shall use the symbolic names defined in the PORT Driver Configuration Tool. The configuration of each Port Pin is Microcontroller-specific.

7.5.2.3 Port_PinDirectionType

The **Table 7-5** shows explanation of Port_PinDirectionType.

Table 7-5 Port_PinDirectionType

Name:	Port_PinDirectionType	
Type:	Enumeration	
Range:	PORT_PIN_OUT	Output Direction
	PORT_PIN_IN	Input Direction
Description:	These are the possible directions; a port pin can have both input and output.	

7.5.2.4 Port_PinModeType

The **Table 7-6** shows explanation of Port_PinModeType.

Table 7-6 Port_PinModeType

Name:	Port_PinModeType	
Type:	Enumeration	
Range:	APP_ALT0	Alternative function 0
	APP_ALT1	Alternative function 1
	APP_ALT2	Alternative function 2
	APP_ALT3	Alternative function 3
	APP_ALT4	Alternative function 4
	DIO	Digital Input Output
	INTERRUPT	Interrupt mode
Description:	These are the possible modes; a port pin can have both input and output.	

7.5.3 Function Definitions

The **Table 7-7** shows list of APIs Provided by the PORT Driver Component.

Table 7-7 APIs Provided by the PORT Driver Component

Sl.No	APIs
1	Port_Init
2	Port_RefreshPortDirection
3	Port_GetVersionInfo
4	Port_SetPinDirection
5	Port_SetPinDefaultDirection
6	Port_SetToDioMode
7	Port_SetToAlternateMode
8	Port_SetPinMode
9	Port_FUSEMonitoring
10	Port_UnintendedModuleStopCheck

7.5.3.1 Renesas Original API

7.5.3.1.1 Port_SetPinDefaultDirection

The **Table 7-8** explains the API of Port_SetPinDefaultDirection.

Table 7-8 Port_SetPinDefaultDirection

Function Name:	Port_SetPinDefaultDirection		
Syntax:	FUNC (void, PORT_PUBLIC_CODE) Port_SetPinDefaultDirection (const Port_PinType LddPinNumber)		
Service Id:	0x08		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddPinNumber	Port Pin Id number	
		Range:	0.. PORT_TOTAL_NUMBER_OF_PINS-1 (Total number of PortPin)
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	void		
Description:	This service sets the Port pin direction during runtime. The PORT Driver module allows changing the direction of the pin to default direction set by the configuration at the time of Port_Init().		
Preconditions:	The PORT Driver module should be initialized.		
Remarks:	This API is available only if the pre-compile option PORT_SET_PIN_DEFAULT_DIRECTION_API is STD_ON and PORT_SET_PIN_DIRECTION_API is STD_ON.		
DET/DEM Errors handled:	PORT_E_UNINIT		PORT Driver is not initialized.
	PORT_E_PARAM_PIN		Parameter (In) PIN is invalid.
	PORT_E_DIRECTION_UNCHANGEABLE		Configuration table of the required PIN is NULL.

7.5.3.1.2 Port_SetToDioMode

The **Table 7-9** explains the API of Port_SetToDioMode.

Table 7-9 Port_SetToDioMode

Function Name:	Port_ SetToDioMode		
Syntax:	FUNC (void, PORT_PUBLIC_CODE) Port_SetToDioMode (const Port_PinType LddPinNumber,)		
Service Id:	0x05		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddPinNumber	Port Pin ID number	
		Range:	0.. PORT_TOTAL_NUMBER_OF_PINS-1 (Total number of PortPin)
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	void		
Description:	This function is used to set the DIO mode of a Port pin during runtime.		
Preconditions:	The PORT Driver module should be initialized.		
Remarks:	This API is available only if the pre-compile option PORT_SET_TO_DIO_ALT_MODE_API is STD_ON.		
DET/DEM Errors handled:	PORT_E_UNINIT		PORT Driver is not initialized.
	PORT_E_PARAM_PIN		Parameter (In) PIN is invalid.
	PORT_E_MODE_UNCHANGEABLE		Required PIN is not configured as mode changeable.

7.5.3.1.3 Port_SetToAlternateMode

The **Table 7-10** explains the API of Port_SetToAlternateMode.

Table 7-10 Port_SetToAlternateMode

Function Name:	Port_SetToAlternateMode		
Syntax:	FUNC (void, PORT_PUBLIC_CODE) Port_SetToAlternateMode (const Port_PinType LddPinNumber,)		
Service Id:	0x06		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddPinNumber	Port Pin ID number	
		Range:	0.. PORT_TOTAL_NUMBER_OF_PINS-1 (Total number of PortPin)
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	void		
Description:	This function used to set the Alternate mode of a Port pin during runtime.		
Preconditions:	The PORT Driver module should be initialized.		
Remarks:	This API is available only if the pre-compile option PORT_SET_TO_DIO_ALT_MODE_API is STD_ON.		
DET/DEM Errors handled:	PORT_E_UNINIT		PORT Driver is not initialized.
	PORT_E_PARAM_PIN		Parameter (In) PIN is invalid.
	PORT_E_MODE_UNCHANGEABLE		Required PIN is not configured as mode changeable

7.5.3.1.4 Port_FUSEMonitoring

The **Table 7-11** explains the API of Port_FUSEMonitoring.

Table 7-11 Port_FUSEMonitoring

Function Name:	Port_FUSEMonitoring		
Syntax:	FUNC(Std_ReturnType, PORT_PUBLIC_CODE) Port_FUSEMonitoring(void)		
Service Id:	0x0B		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	None		
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	E_NOT_OK	PORT driver detects difference between the actual value and expected value of FUSE Monitoring register.
		E_OK	No difference between the actual values and expected value of FUSE Monitoring register.
Description:	This service is used to check the values of configuration FUSE Monitoring register with the expected value. OTPMONITORm (m = 0, 3)		
Preconditions:	None		
Remarks:	This API is available only if the pre-compile option PORT_FUSE_MONITORING_API is STD_ON.		
DET/DEM Errors handled:	PORT_E_FUSE_MONITORING_FAILURE		PORT driver detects the difference between the actual value and expected value of configuration FUSE Monitoring register.

7.5.3.1.5 Port_UnintendedModuleStopCheck

The **Table 7-12** explains the API of Port_UnintendedModuleStopCheck.

Table 7-12 Port_UnintendedModuleStopCheck

Function Name:	Port_UnintendedModuleStopCheck	
Syntax:	FUNC(void, PORT_PUBLIC_CODE) Port_UnintendedModuleStopCheck(void)	
Service Id:	0x0C	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant	
Parameters (In):	None	
Parameters (In-Out):	None	
Parameters (Out):	None	
Return Value:	void	
Description:	This service is used to check the value of configuration registers with the expected value when unintended module stop occurs	
Preconditions:	The PORT Driver module should be initialized.	
Remarks:	The function is available if the pre-compile option PORT_UNINTENDED_MODULE_STOP_CHECK is STD_ON	
DET/DEM Errors handled:	PORT_E_UNINIT	PORT Driver is not initialized.
	PORT_E_UNINTENDED_MODULE_STOP_FAILURE	PORT driver detects fault by the difference between the read value and expected value of configuration registers (MODSEL8, FILCLKSEL8).

7.5.4 Preemption of APIs

The **Table 7-13** shows list of Preemption Table of APIs of the PORT Driver

Table 7-13 Preemption Table of APIs of the PORT Driver

	Port_Init	Port_SetPinDirection	Port_RefreshPortDirection	Port_GetVersionInfo	Port_SetPinMode	Port_SetPinDefaultDirection	Port_SetToDioMode	Port_SetToAlternateMode	Port_FUSEMonitoring	Port_UnintendedModuleStopCheck
Port_Init	-									
Port_SetPinDirection	-	√								
Port_RefreshPortDirection	-	√	-							
Port_GetVersionInfo	√	√	√	√						
Port_SetPinMode	-	√	-	√	√					
Port_SetPinDefaultDirection	-	√	-	√	√	√				
Port_SetToDioMode	-	√	-	√	√	√	√			
Port_SetToAlternateMode	-	√	-	√	√	√	√	√		
Port_FUSEMonitoring	√	√	√	√	√	√	√	√	√	√
Port_UnintendedModuleStopCheck	-	√	√	√	-	√	-	-	√	√

"-" : cannot be invoked at the same time

"√" : can be invoked at the same time

Note:

- Operating conditions when PORT_CRITICAL_SECTION_PROTECTION = STD_ON
- List of these API: Port_SetPinDirection, Port_SetPinDefaultDirection, Port_SetPinMode, Port_SetToDioMode and Port_SetToAlternateMode which just can be ensured the proper operation when these API is called with different input PortPin at the same time.

7.6 Development and Production Errors

In this section, the development and production errors reported by the PORT Driver Component are tabulated. The development errors will be reported only when the pre-compiler option 'PORT_DEV_ERROR_DETECT' is enabled in the configuration.

7.6.1 PORT Driver Component Development Errors

The following table contains the DET errors reported by PORT Driver Component. These errors are reported

to Development Error Tracer Module when the PORT Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 7-14 DET Errors of PORT Driver Component

Sl. No.	1
Error Code	PORT_E_PARAM_PIN
Value (hex)	0x0A
Related API(s)	Port_SetPinDirection, Port_SetPinDefaultDirection
Source of Error	API is invoked with invalid Pin number.
Sl. No.	**2**
Error Code	PORT_E_DIRECTION_UNCHANGEABLE
Value (hex)	0x0B
Related API(s)	Port_SetPinDirection, Port_SetPinDefaultDirection
Source of Error	API is invoked with Pin not configured as ‘Direction Changeable during run time’.
Sl. No.	**3**
Error Code	PORT_E_INIT_FAILED
Value (hex)	0x0C
Related API(s)	Port_Init
Source of Error	API is invoked with the config pointer value as NULL.
Sl. No.	**4**
Error Code	PORT_E_PARAM_POINTER
Value (hex)	0x10
Related API(s)	Port_GetVersionInfo
Source of Error	API is invoked with parameter as NULL pointer.
Sl. No.	**5**
Error Code	PORT_E_UNINIT
Value (hex)	0x0F
Related API(s)	Port_RefreshPortDirection, Port_SetPinDirection, Port_SetPinDefaultDirection, Port_SetToAlternateMode, Port_SetToDioMode, Port_SetPinMode, Port_UnintendedModuleStopCheck
Source of Error	APIs are invoked without the initialization of the PORT Driver Component.
Sl. No.	**6**
Error Code	PORT_E_INVALID_DATABASE
Value (hex)	0xEF
Related API(s)	Port_Init
Source of Error	Invalid database is found.
Sl. No.	**7**
Error Code	PORT_E_MODE_UNCHANGEABLE
Value (hex)	0x0E
Related API(s)	Port_SetToDioMode, Port_SetToAlternateMode, Port_SetPinMode
Source of Error	APIs are invoked without the initialization of the PORT Driver Component.
Sl. No.	**8**
Error Code	PORT_E_PARAM_INVALID_MODE
Value (hex)	0x0D
Related API(s)	Port_SetPinMode
Source of Error	API is invoked with invalid mode.

7.6.2 PORT Driver Component Production Errors

In this section, the DEM errors identified in the PORT Driver Component are listed. PORT Driver Component reports these errors to DEM by invoking Dem_SetEventStatus API. This API is invoked, when the processing of the given API request fails.

Table 7-15 DEM Errors of PORT Driver Component

Sl. No.	1
Error Code	PORT_E_GET_CONTROL_FAILURE
Value	DemConf_DemEventParameter_<short name of DemEventParameter>
Related API(s)	Port_Init, Port_RefreshPortDirection
Source of Error	The Exclusive Control functionality DEM error will be reported, if Port Driver can't get control registers.
Sl. No.	**2**
Error Code	PORT_E_FUSE_MONITORING_FAILURE
Value	DemConf_DemEventParameter_<short name of DemEventParameter>
Related API(s)	Port_FUSEMonitoring
Source of Error	The FUSE Monitoring functionality DEM error will be reported, if the actual value of registers OTPMONITORm (m= 0, 3) is mismatched with the expected value configured by the user.
Sl. No.	**3**
Error Code	PORT_E_UNINTENDED_MODULE_STOP_FAILURE
Value	DemConf_DemEventParameter_<short name of DemEventParameter>
Related API(s)	Port_UnintendedModuleStopCheck
Source of Error	The Unintended Module Stop Check functionality DEM error will be reported, if the configuration value of registers MODSEL8 or FILCLKSEL8 is mismatched expected value when unintended module stop occurs.

7.7 PORT Driver Component Runtime Errors

AUTOSAR does not define any runtime error code (see Chapter 7.2.2 of AUTOSAR Specification of PORT Driver (AUTOSAR_SWS_PORTDriver.pdf) [1]) for PORT Driver.

7.8 Memory Organization

The following tables depict a typical memory organization, which must be met for proper functioning of PORT Driver Component software.

Table 7-16 ROM Sections of the PORT Driver

Section Name	Alignment(*1)	Description
PORT_PUBLIC_CODE_ROM	-	This section contains codes which belong to the API functions of the PORT Driver.
PORT_PRIVATE_CODE_ROM	-	This section contains codes which belong to the internal functions of the PORT Driver.
PORT_CFG_DBTOC_UNSPECIFIED	-	This section consists of PORT Driver Component database table of contents generated by the PORT Driver Component Generation Tool. It can be located in code memory.
CONFIG_DATA_UNSPECIFIED	-	This section consists of PORT Driver Component constant configuration structures. IT can be located in code memory.

Table 7-17 RAM Sections of the PORT Driver

Section Name	Alignment(*1)	Description
VAR_NO_INIT_PTR	-	This section consists of the global RAM pointer variables used internally by PORT Driver Component. It can be located in data memory.
VAR_INIT_1	-	This section consists of the global RAM variables of boolean used internally by PORT Driver Component. It can be located in data memory.
VAR_NO_INIT_32	-	This section consists of the global RAM variables of uint32 used internally by PORT Driver Component. It can be located in data memory.

Note: *1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

7.9 Device-Specific Information

It supports the following devices:
Refer to “R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 5.1 Product.

7.9.1 Interaction between the User and PORT Driver Component

The following sections detail the services supported by the PORT Driver Component to the upper layer’s users and the mapping of the channels to the hardware units:

7.9.1.1 Channel Mapping

None.

7.9.1.2 ISR Function

None.

7.9.2 Multi-Core / Multi-Instantiation

PORT driver does not support multi-core and multi-instantiation for this device.

7.10 Non-AUTOSAR environment integration

7.10.1 Stub modules handling

7.10.1.1 Det

The Det stub files are organized in the following folder:

\RCar\common_platform\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, PORT Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g. PORT Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:

Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

7.10.1.2 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.

The SchM (Basic Software Scheduler) stub files are organized in the following folder:

\RCar\common_platform\generic\stubs\<Autosar version>\Rte

PORT driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when PORT_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user MSN driver as following:

Write SchM functions with prototype as below:

void SchM_Enter_Port_PORT_INTERRUPT_CONTROL_PROTECTION (void);

void SchM_Exit_Port_PORT_INTERRUPT_CONTROL_PROTECTION (void);

7.10.1.3 Dem

The Dem stub files are organized in the following folder:

\RCar\common_platform\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, PORT Driver Components uses Dem_SetEventStatusAPI provided by the DEM module to report a production. The API prototype is as of follow:

Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)
Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

7.10.2 Callback function usage

The PORT Driver Component does not provide any callback functions.

7.10.3 Scheduled function usage

The PORT Driver Component does not provide any scheduled functions.

7.10.4 Interrupt handling usage

The PORT Driver Component does not provide any interrupt handling functions.

8.ETH

8.1 Overview

The purpose of this chapter is to describe the information related to ETH Driver Component

This chapter shall be used as reference by the users of ETH Driver Component. The system overview of complete AUTOSAR architecture is shown in the figure below.

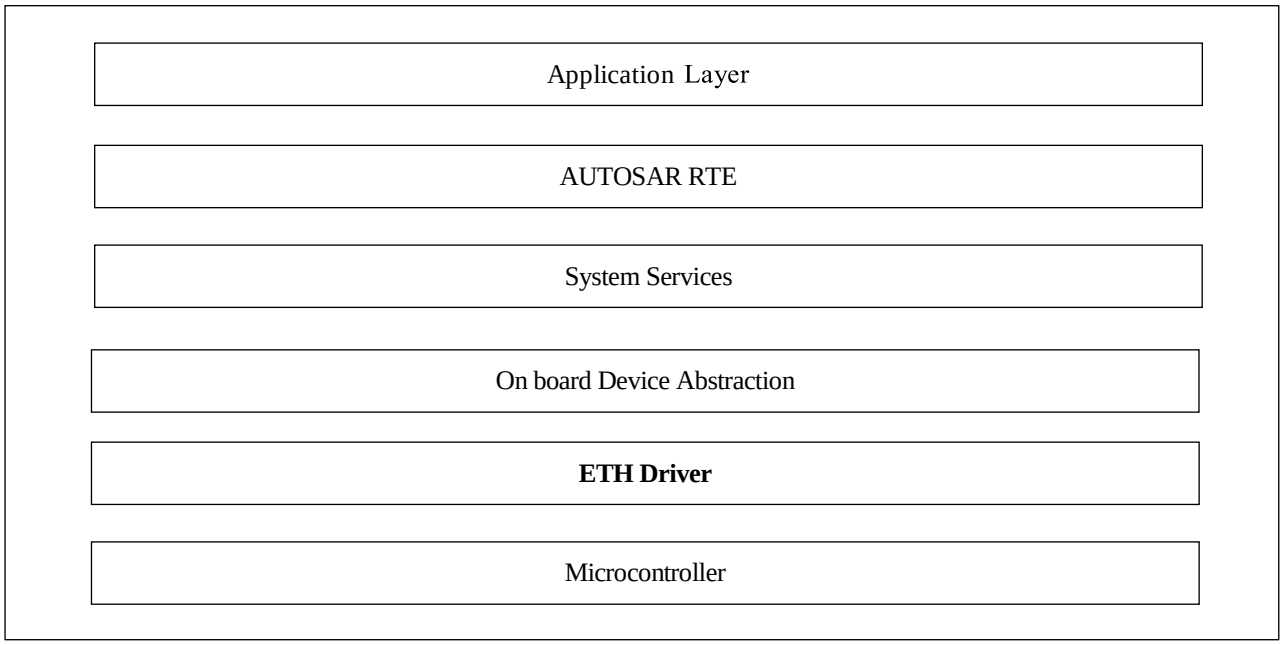


Figure 8.1 System Overview of AUTOSAR Architecture

The ETH Driver is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.

The figure in the following page depicts the ETH Driver as part of layered AUTOSAR MCAL Layer:

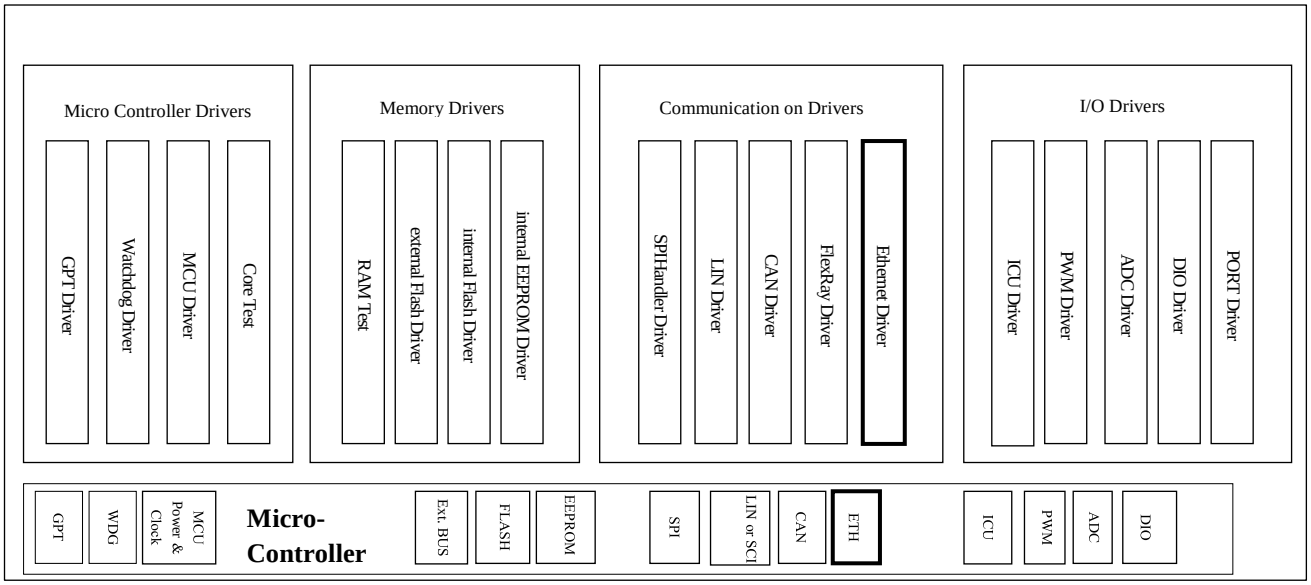


Figure 8.2 System Overview of the ETH Driver in AUTOSAR MCAL Layer

The ETH Driver Component comprises Embedded software and the Configuration Tool to achieve scalability and configurability.

The ETH Driver Component Code Generation Tool is a command line tool that accepts ECU configuration description files as input and generates source and header files. The configuration description is an ARXML file that contains information about the configuration for ETH Driver. The tool generates the Eth_PBcfg.c and Eth_Cfg.h

8.2 Forethoughts

8.2.1 General

The following information will help the user use the ETH Driver Component software efficiently:

- AUTOSAR R19-11 Specification.
- ETH Driver shall be integrated in the AUTOSAR R19-11 stack (e.g. version checking is still done for Det, Dem, and EthIf released in R19-11 Spec version).
- ETH Driver cannot be used in multithread. Use single thread as specified in AUTOSAR.
- Example code mentioned in this chapter shall be taken only as a reference for implementation.
- MCU specific initializations such as reset registers, one-time writable registers, interrupt stack pointer, user stack pointer, internal watchdog, specific features of internal memory and registers are not implemented by ETH Driver. These initializations must be implemented by the start-up code.
- Only Eth_GetVersionInfo and Eth_MainFunction APIs are allowed to be invoked before Eth_Init.
- Eth_UpdatePhysAddrFilter can be invoked while a controller is ACTIVE or DOWN. However, 00:00:00:00:00:00 and FF:FF:FF:FF:FF:FF are allowed only while a controller is DOWN.
- To terminate promiscuous mode turned on by Eth_UpdatePhysAddrFilter with FF:FF:FF:FF:FF:FF, Eth_UpdatePhysAddrFilter must be invoked with 00:00:00:00:00:00. During promiscuous mode, any multicast address passed to Eth_UpdatePhysAddrFilter is ignored.
- Up to 32 multicast addresses can be registered in the address filter list by Eth_UpdatePhysAddrFilter. Unicast address cannot be registered in the address filter.
- If 00:00:00:00:00:00 is passed to Eth_UpdatePhysAddrFilter, all registered multicasts addresses are removed.
- A payload length is passed to the parameter ‘LenByte’ of EthIf_RxIndication, instead of a frame length.
- A receive buffer is released right after EthIf_RxIndication returns, and it will be overwritten after a round of the receive buffer ring.
- A transmit buffer are released after a completion of a transmission regardless of the 4th argument TxConfirmation of Eth_Transmit.
- Even if Eth_Transmit returns E_NOT_OK, the Tx buffer will not be released. In this case, the application can continue to access the Tx buffer.
- In the polling mode, Eth_TxConfirmation must be invoked periodically even if EthIf_TxConfirmation is not required, since the transmitted Tx buffer is released by this function.
- EtherType is the host CPU byte order.
- Eth_SetPhysAddr must take a valid unicast address. Multicast addresses and the broadcast address are not allowed.
- Eth_SetControllerMode, Eth_ReadMii, and Eth_WriteMii are implemented as synchronous. Thus, the callback functions are invoked from these APIs directly.
- When ETH_MODE_DOWN is passed to Eth_SetControllerMode, it waits for the on-going transmission and is blocked until the transmission completes. The maximum blocking time is specified by EthTimeout parameter. The maximum number of frames that can be transmitted until the controller stops is not guaranteed.
- Eth_ProvideTxBuffer, Eth_Transmit, Eth_Receive and Eth_TxConfirmation are allowed only while a controller is ACTIVE.
- Eth_SetPhysAddr is allowed only while a controller is DOWN.
- The EthRamSize parameter is the size of the heap memory used in this module. Rx buffers and the descriptors, and Tx buffers and the descriptors are dynamically allocated using heap memory. Hence, the user shall make sure it configures an amount of Memory enough to allocate. This module can make configuration to 4 transmit queues by the EthTxQueueConfig container.
- The EthTxQueueIdx parameter specifies the transmit queue number to use. The features related to the numbers are as follows.
  0 : Best Effort.
  1 : Network Control for the Precision Time Protocol.
  2 : Class B for CBS algorithm by Ethernet AVB.
  3 : Class A for CBS algorithm by Ethernet AVB.
- If the EthQosSupport parameter is not enabled, Best Effort is selected for all transmit queues.

- The EthTxQueuePolicy parameter set to ETH_CBS can set the detailed parameters of the CBS algorithm by the EthCtrlTxQueueBwFraction parameter.
- The ETH_CBS setting applies only to Class A and Class B transmit queues.
- The EthCtrlTxQueueBwFraction parameter sets the bandwidth percentage of the transmit queue.
- The EthTxQueueBufs parameter sets the number of the Tx descriptors. Also, this parameter affects the throughput.
- This module can make configuration to the PCP (Priority Code Point) by the EthCtrlPriority container.
- The EthCtrlPriorityValue parameter specifies a priority value between 0 and 7.
  0 : Best Effort.
  1 : Background.
  2 : Excellent effort.
  3 : Critical applications.
  4 : Video, < 100ms latency and jitter.
  5 : Voice, < 10ms latency and jitter.
  6 : Internetwork control.
  7 : Network control.
- If the EthQosSupport parameter is not enabled, this feature cannot be used.
- The EthCtrlTxQueueRef parameter specifies the container name of the transmit queue.
- The EthCtrlTxDefaultQueueRef parameter specifies the name of the outbound queue container to select as the default. If EthCtrlTxQueueRef is not found, this parameter will be selected.
- This module can make configuration to 18 receive queues by the EthRxQueueConfig container.
- The EthRxQueueIdx parameter specifies the receive queue number to use. The features related to the numbers are as follows.
  0 : Best Effort.
  1 : Network Control for the Precision Time Protocol.
  2-17 : Stream for filtering the specific AVTP frame.
- The EthNwCntrlFiltering parameter set to true can be the receive LLDP frames into the Network Control.
- The EthPatternStream parameter sets the Stream ID of the AVTP frame to be separated. This parameter only applies to the Stream receive queues.
- If true is specified for the EthSRPTalkerFiltering parameter, only the Talker source address part (6 bytes) in the Stream ID will be compared.
- If ETH_SFDISABLE is set in the EthStreamFiltering parameter, the separating feature will be disabled.
- If true is specified for the EthBeTimeStampStore parameter, the time stamp information is included in the receive descriptor of the best effort.
- If true is specified for the EthStreamTimeStampStore parameter, the time stamp information is included in the receive descriptor of the stream.
- Network Control receive descriptors always include time stamp information.
- Timestamp information can be obtained by Eth_GetIngressTimeStamp.
- The EthRxQueueBufs parameter sets the number of the Rx descriptors. Also, this parameter affects the throughput.
- Condition for network control frames:
  The Ethernet destination address (DA) is 01:80:C2:00:00:0E.
  The Ethernet type (ET) is 88:F7.
- Condition for AVTP frames:
  The Ethernet destination address (DA) is within the range from 91:E0:F0:00:00:00 to 91:E0:F0:00:FE:FF.
  The VLAN-tagged TPID (tag protocol identifier) field (VL) is 81:00.
  The Ethernet type (ET) is 22:F0.
- The argument “FifoIdx” of Eth_Receive specifies the reception queue (correlating to EthRxQueueIdx parameter) to be checked for the new received frames.

8.2.2 Preconditions

The following preconditions must be adhered by the user, for proper functioning of the ETH Driver Component:

- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used. The version information is described in the 1.1 Supported MCAL Product Release Version.
- The Eth_Cfg.h file generated by the ETH Driver Generation Tool must be compiled and linked with the ETH Driver source files.
- The application must be rebuilt, if there is any change in the Eth_Cfg.h file generated by the Ethernet Driver Component Code Generation Tool.
- The users of this component shall ensure that all the ETH Driver Component APIs are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated by the ETH Driver Component APIs only when the configuration parameter 'EthDevErrorDetect' is enabled. Hence, it should be ensured that the right parameters are passed while invoking the APIs when EthDevErrorDetect is disabled.
- ETH Driver expect User / Ethernet Interface invoke the MII interface API to set up the physical link during the Initialization procedure, as the ETH Driver does not perform this on its own but offers the APIs. (Eth_WriteMii and Eth_ReadMii). User must provide a timer to check for the status of Link.
- When Eth_Transmit is invoked while the previous transmission is on-going on the hardware, the transmission request is queued and performed when the current transmission is completed. In the polling mode, the queue handling and Tx buffer release are performed in Eth_TxConfirmation. Thus Eth_TxConfirmation must be invoked periodically in the polling mode.
- The E-MAC of AVB0 supports a RGMII (100/1000base).
- The E-MAC of AVB1 supports a RGMII (100/1000base).
- The E-MAC of AVB2 supports a RGMII (100/1000base).
- Since signals of Ethernet have high frequency, the driving ability (driving strength) of output port pins may be set properly according to the board design.
- The following Table 8.1 shows the limitations applied depending on the peripheral clock settings.

Table 8.1 Limitation of Communication List

HW unit	Speed	CPU:400Mhz High Speed Peripheral clock:100Mhz
AVB0	100base	Support
	1000base	Support
AVB1	100base	Support
	1000base	Support
AVB2	100base	Support
	1000base	Support

- It is responsibility of user to make sure the consistency between immediate clock value (configured in EthInputClockRefImmediateValue).
- The gPTP module only access in secure mode. Therefore, CR52 must change to Secure mode before initialize Ethernet Driver when EthGlobalTimeSupport parameter configured true.
- To enable supply of clock signal to the gPTP module, bit 23 of the MSTPCR27 register must be changed to b0, this may be done either manually or by enable the McuTSNClockSupplyEnable parameter in the MCU module prior to initializing the Ethernet Driver. Due to the shared usage of the gPTP and TSN clock signals.
- The transmission time stamp information was handled by polling when the EthGlobalTimeSupport parameter was enabled, which causes a delay. The CR52 cannot handle transmission time stamp information through interrupt mode since it does not allow nested interrupts. Consequently, the reading back the egress time stamp on a dedicated message object always ETH_INVALID in TxConfirmation() function.

- After initializing the PHY interface, allow sufficient waiting time before invoking Eth_SetControllerMode() with ETH_MODE_ACTIVE.
  The required waiting time depends on the PHY chip used and the specifications of the "Media-Independent Interface" used.
  However, depending on the PHY chip, it may fail even if the required time constraints are adhered to.
  In this case, retry Eth_SetControllerMode() until it succeeds.
- When the configuration parameter EthSwitchManagementSupport is set as TRUE, the API Eth_ProvideTxBuffer() returns Ethernet payload data length via in-out parameter 'uint16* LenBytePtr'. It does not contain Ethernet Switch Management Information length (which is got from API EthSwt_EthTxAdaptBufferLength()).
  It is necessary to adjust the length of Tx buffer for the Management Information in EthIf_ProvideTxBuffer() and EthIf_Transmit().
  The following is the APIs related to the above, be careful when implementing them.
  Also, refer to the stub file provided as a reference example of implementation.
  EthIf_ProvideTxBuffer.
  EthIf_Transmit.
  EthSwt_EthTxAdaptBufferLength.
  EthSwt_EthTxPrepareFrame.
  EthSwt_EthTxProcessFrame.
- When the container EthCtrlConfig is configured with more than 1 configuration, it is necessary for user to make sure the parameter EthCtrlIdx is configured with sequential value.
- EthernetAVB-IF has a dedicated DMAC (Direct Memory Access Controller) to copy data from user memory to hardware for transmitting and opposite direction for receiving. To avoid cache coherency problem when L1 data cache of Cortex-R52 is enabled, user needs to provide some functions with following names and formats in Table 8.2 and Table 8.3 for cache maintenance. ETH driver will use them to maintain the cached memory area which it uses. The operation of ETH driver will not be guaranteed if L1 data cache is enabled but these functions are not implemented. The size of one cache line on Cortex-R52 is 32 bytes, so these functions should accept 32 bytes aligned input start address. The input size can be unaligned, these functions have to calculate proper number of cache lines based on this size. User also should implement these functions, at least, for driver compilation if L1 data cache is disabled because ETH driver will always call them regardless to status of L1 data cache.

Table 8.2 CR7_Invalidate_DCache_By_Addr

Function Name:	CR7_Invalidate_DCache_By_Addr	
Syntax:	CR7_Invalidate_DCache_By_Addr (uint32 start_addr, uint32 size)	
Parameters (In):	start_addr	Start address of the buffer that L1 data cache need to be maintained
	size	Number of bytes of above buffer
Parameters (In-Out):	None	
Parameters (Out):	None	
Return Value:	None	
Description:	This API shall invalidate L1 data cache by address and size of the buffer.	

Table 8.3 CR7_Flush_DCache_By_Addr

Function Name:	CR7_Flush_DCache_By_Addr	
Syntax:	CR7_Flush_DCache_By_Addr (uint32 start_addr, uint32 size)	
Parameters (In):	start_addr	Start address of the buffer that L1 data cache need to be maintained
	size	Number of bytes of above buffer
Parameters (In-Out):	None	
Parameters (Out):	None	
Return Value:	None	
Description:	This API shall clean and invalidate L1 data cache by address and size of the buffer.	

8.2.3 Data Consistency

To support the reentrancy and interrupt services, the ETH Driver Component will ensure the data consistency while accessing their own RAM storage or hardware registers.

```
#define ETH_ENTER_CRITICAL_SECTION (Exclusive_Area)
SchM_Enter_Eth_##Exclusive_Area()

#define ETH_EXIT_CRITICAL_SECTION (Exclusive_Area)
SchM_Exit_Eth_##Exclusive_Area()
```

In Table 8.4 the following exclusive areas along with scheduler services are used to provide data integrity for shared resources:

Table 8.4 ETH Driver Protected Resource List

API Name	Exclusive Area Type	Protected Resources
Eth_Transmit Eth_TxConfirmation Eth_ProvideTxBuffer ETH_AVBnDATAISR, with n equal 0..2	ETH_RAM_DATA_PROTECTION	Tx/Rx descriptor queue Hardware status information Dynamic buffer management

8.2.4 Continuous Transmission

This section explains the behavior of the ETH Driver when Eth_Transmit is invoked before the previous transmission completes.

8.2.4.1 **Interrupt Mode**

- When Eth_Transmit is invoked when no on-going transmission, the requested frame is started immediately.
- When Eth_Transmit is invoked when on-going transmission, the request will be set to an empty descriptor. (not supported)
- If there are no empty descriptors, the request is queued in the Tx pending queue.
- The transmission frame set in the descriptor starts one by one for each interruption.
- When ETH_TRUE is specified to TxConfirmation of Eth_Transmit, the EthIf_TxConfirmation is invoked for each transmission completion. If there are empty descriptors, and the request is in the Tx pending queue, set the request to an empty descriptor.

8.2.4.2 **Polling Mode**

- When Eth_Transmit is invoked when no on-going transmission and there is no unconfirmed transmission, the requested frame is started immediately.
- When Eth_Transmit is invoked when on-going transmission or there is unconfirmed transmission, the request will be set to an empty descriptor.
- If there are no empty descriptors, the request is queued in the Tx pending queue.
- The transmission frame set in the descriptor continues the frame transmission processing without invoking Eth_TxConfirmation.
- When Eth_TxConfirmation invoked after the transmission finished by the RMAC, then EthIf_TxConfirmation is invoked for each completed frame, if the confirmation is required. If there are empty descriptors, and the request is in the Tx pending queue, set the request to an empty descriptor.

8.2.5 ETH Driver Error List

This section explains the errors caused by the ETH Driver. The following Table 8.5 to Table 8.8 list them.

Table 8.5 ETH Driver Error List (1/4)

API Name	Error Code	Source of Error
Eth_Init	ETH_E_PARAM_POINTER	Parameter (In) CfgPtr is NULL.
	ETH_E_INVALID_DATABASE	Database is failed.
	ETH_E_ACCESS	Access error to E-MAC.
	ETH_E_REGISTER_CORRUPTION	Any of target registers have not expected values.
Eth_SetControllerMode	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_ACCESS	The E-MAC mode change failed.
Eth_GetControllerMode	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (out) CtrlModePtr is NULL.
Eth_GetPhysAddr	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (out) PhysAddrPtr is NULL.
Eth_SetPhysAddr	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (out) PhysAddrPtr is NULL.
	ETH_E_INV_MODE	Current Controller mode is not DOWN.
Eth_UpdatePhysAddrFilter	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (In) PhysAddrPtr is NULL.
	ETH_E_INV_MODE	00:00:00:00:00:00 or FF:FF:FF:FF:FF:FF is specified when the controller state is ACTIVE.
	ETH_E_INV_PARAM	The specified address with ETH_ADD_TO_FILTER is already registered.
		The specified address with ETH_REMOVE_FROM_FILTER is not registered.
		The filter array is full.
		The specified address is not a multicast.

Table 8.6 ETH Driver Error List (2/4)

API Name	Error Code	Source of Error
Eth_WriteMii	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_PARAM	Parameter (In) TrcvIdx or RegIdx is out of range.
Eth_ReadMii	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_PARAM	Parameter (In) TrcvIdx or RegIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) RegValPtr is NULL.
Eth_GetCurrentTime	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) timeQualPtr is NULL.
	ETH_E_INV_MODE	Current Controller mode is DOWN.
Eth_EnableEgressTimeStamp	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_PARAM	Parameter (In) BufIdx is an invalid value.
Eth_GetEgressTimeStamp	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) timeQualPtr or timeStampPtr is NULL.
	ETH_E_INV_MODE	Current Controller mode is DOWN.
Eth_GetIngressTimeStamp	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) DataPtr is NULL.
	ETH_E_INV_MODE	Current Controller mode is DOWN.
Eth_ProvideTxBuffer	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) BufIdxPtr or BufPtr or LenBytePtr is NULL.
	ETH_E_INV_MODE	Current Controller mode is DOWN.

Table 8.7 ETH Driver Error List (3/4)

API Name	Error Code	Source of Error
Eth_Transmit	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_PARAM	Parameter (In) BufIdx is out of range or not provided.
	ETH_E_PARAM_POINTER	Parameter (In) PhysAddrPtr is NULL.
	ETH_E_INV_MODE	The controller mode is not ACTIVE
Eth_Receive	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_MODE	The controller mode is not ACTIVE.
	ETH_E_PARAM_POINTER	Parameter (Out) RxStatusPtr is NULL.
	ETH_E_INV_PARAM	Parameter (In) FifoIdx is an invalid value.
Eth_TxConfirmation	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_MODE	The controller mode is not ACTIVE.
Eth_MainFunction	ETH_E_CRC	Receive CRC error
	ETH_E_ALIGNMENT	Receive alignment error
	ETH_E_OVERSIZEFRAME	Receive oversize frame
	ETH_E_UNDERSIZEFRAME	Receive undersize frame
	ETH_E_RX_FRAMES_LOST	Receive frame lost
	ETH_E_DMA_ERROR	DMA relevant errors
	ETH_E_REGISTER_CORRUPTION	Any of target registers have not expected values.
Eth_GetVersionInfo	ETH_E_PARAM_POINTER	Parameter (Out) VersionInfoPtr is NULL.
Eth_GetCounterValues	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) CounterPtr is NULL.
Eth_GetRxStats	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) RxStats is NULL.

Table 8.8 ETH Driver Error List (4/4)

API Name	Error Code	Source of Error
Eth_GetTxStats	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) TxStats is NULL.
Eth_GetTxErrorCounterValues	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (Out) TxErrorCounterValues is NULL.
Eth_UpdateStreamFilter	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_PARAM	Parameter (In) QueIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (In) StreamIdPtr is NULL.
	ETH_E_INV_MODE	The controller mode is STANDBY.
Eth_SetIncrementTimeForGptp	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_MODE	The controller mode is STANDBY.
Eth_SetOffsetTimeForGptp	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (In) pTimeOffsetPtr is NULL.
	ETH_E_INV_MODE	The controller mode is STANDBY.
ETH_AVBnDATAISR, ETH_AVBnERRORISR, ETH_AVBnMACISR with n equal 0..2	ETH_E_INTERRUPT_CONTROLLER_FAILURE	Unintended interrupt check.
ETH_AVBnERRORISR, with n equal 0..2	ETH_E_DMA_ERROR	Tx Buffer is corrupted in the E-MAC.

8.3 Architecture Details

The ETH Driver Component architecture is shown in the following **Figure 8.3**.

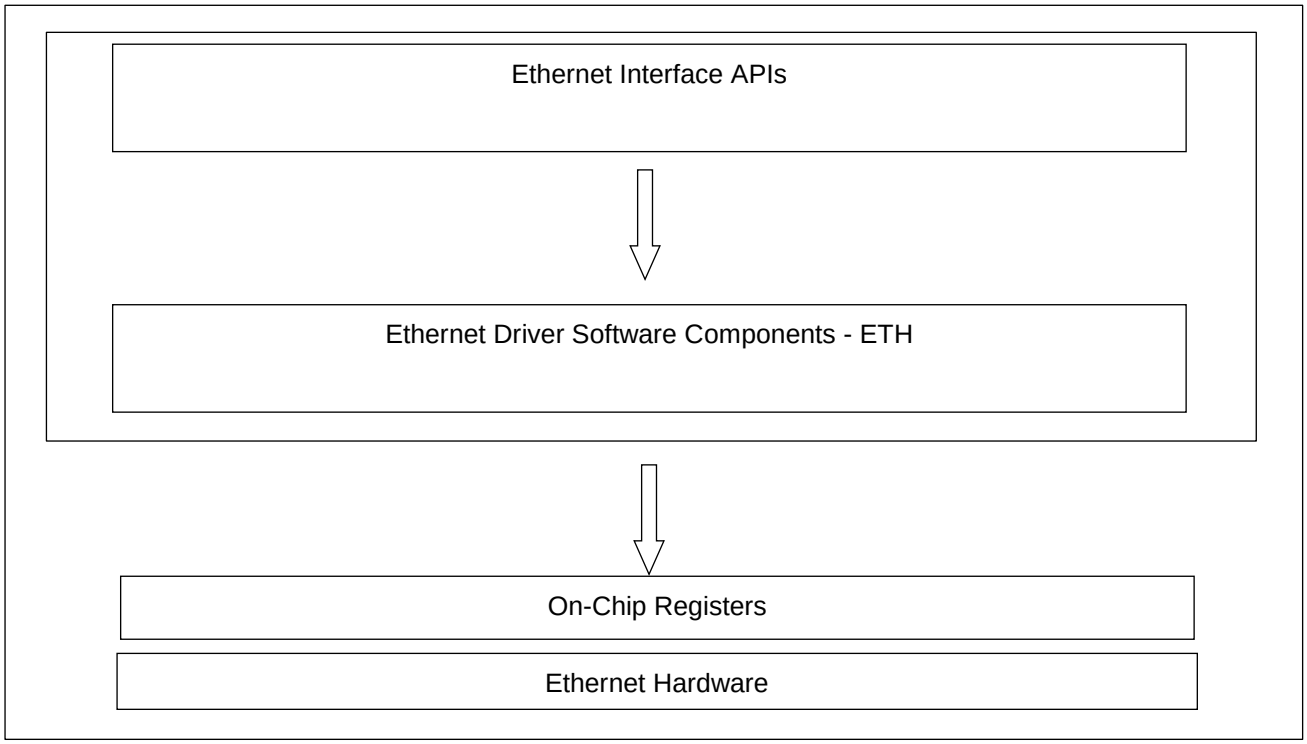


Figure 8.3 ETH Driver Component Architecture

The basic architecture of the ETH Driver Component is illustrated in the following **Figure 8.4**:

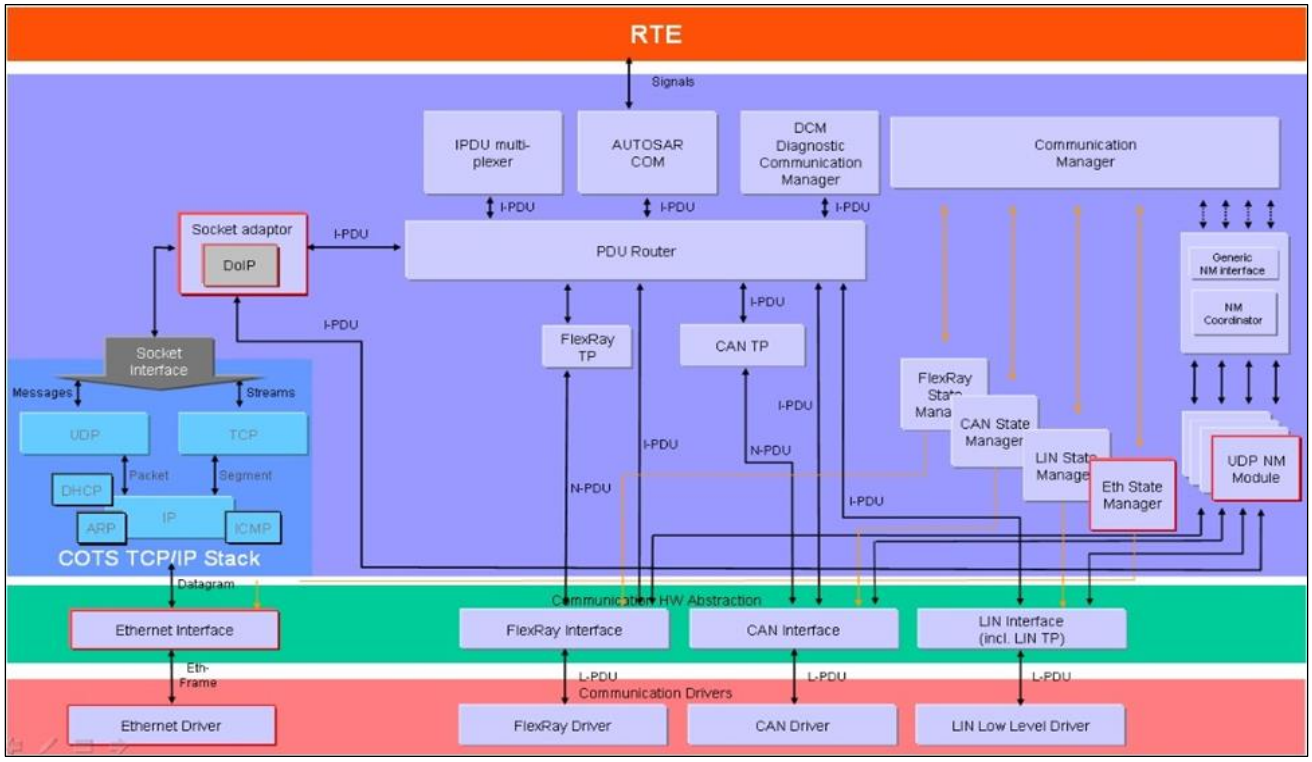


Figure 8.4 Component Overview of ETH Driver Component

The ETH Driver Component can be divided into the following subcomponents based on the functions performed by the ETH Driver:

- Driver Initialization
- Controller Initialization
- Setting and getting the Controller Mode
- Setting and getting the MAC Address of the Ethernet Controller
- Adding/ Removing Mac Address to/from the Rx Hardware Filter Table
- Writing MII Interface register
- Reading MII Interface register
- Getting the Counter State
- Providing Transmit Buffer Access
- Transmit Functionality
- Receive Functionality
- Transmit confirmation
- Frame Reception Interrupt handling
- Frame Transmission Interrupt handling
- Getting Version Information
- Getting Dropped Packets
- Getting Ethernet Statistic
- Providing timestamp settings for transmit frame
- Getting timestamp of transmitted frame
- Getting timestamp of received frame
- Getting current time by the gPTP timer
- Feature to adjust gPTP timer
- IEEE1722 packet filtering functionality

Driver Initialization

The driver initialization sub-module internally stores the configuration data address to enable subsequent API calls to access the configuration data and initializes the global variables used by ETH Driver Component.
The API related to this sub-module is Eth_Init().

Setting and Getting the Controller Mode

The API related to this sub-module are Eth_SetControllerMode() and Eth_GetControllerMode().
This sub-module provides the service for setting and getting the Controller mode as either ACTIVE, DOWN, STANDBY.

Adding/ Removing Mac Address to/from the Rx Hardware Filter Table
The driver provides the possibility to add /delete a new Mac Address to/from the HW Rx Filter Table.
The API related to this function is Eth_UpdatePhysAddrFilter().

Writing MII Interface Register
This sub-module provides the service for writing the Transceiver registers of the Physical MII Interface through the port control register of the Ethernet Controller. The maximum value that can be specified as a register is 32 from 0 to 31.
The MII interface mainly writes registers to set the communication speed, communication mode, and full-duplex mode.
The API related to this sub-module is Eth_WriteMii().

Reading MII Interface Register
This sub-module provides the service for reading the Transceiver registers of the Physical MII Interface through the port control register of the Ethernet Controller. The MII Interfaces Status register is read to monitor the status of the MII Interface.
The API related to this sub-module is Eth_ReadMii().

Providing Transmit Buffer Access
This sub-module provides the service to access the buffer used for the transmit. i.e. this service checks whether the status of the transmit buffer is valid or not. If so, it provides the buffer and the buffer index for the subsequent call for transmit The API related to this sub-module is Eth_ProvideTxBuffer().

Transmit Functionality
This sub-module provides the service to create and transmit the Ethernet frame.
Ethernet frame is created with the Frame header which includes destination address, source address, the frame type and the data payload for Transmission.
This service sets the Transmit descriptor valid for transmit, and the Ethernet frame to the buffer assigned for the corresponding descriptor, and activates the transmit operation.
The API related to this sub-module is Eth_Transmit().

Receive Functionality
This sub-module provides the service to receive the Ethernet frame and to provide the received frame to the upper layer through the Ethernet Interface during the polling mode. This service sets the receive descriptor valid for the receive.
The API related to this sub-module is Eth_Receive().

Transmit Confirmation
This sub-module provides the service to confirm the Transmission of Ethernet frame to the upper layer through the Ethernet interface during polling mode. When transmission is confirmed, the transmission buffer is released.
The API related to this sub-module is Eth_TxConfirmation().

Frame reception Interrupt handling
This sub-module provides the service to receive the Ethernet frame and to provide the received frame to the upper layer through the Ethernet Interface during the Interrupt mode.
The function related to this sub-module is Eth_Hw_Avb_RxIsrHdlr().

Frame Transmission Interrupt handling
This sub-module provides the service to confirm the Transmission of Ethernet frame to the upper layer through the Ethernet interface during Interrupt mode.
The function related to this sub-module is Eth_HwTxConfirmation().

Getting Version Information
The ETH Driver provides a service (implemented as C macro) to return version information details to the user.
The function related to this sub-module is Eth_GetVersionInfo ().

Getting Dropped Packets

The ETH Driver provide a service to read the list of dropped packets and return it to the user. It reads the underlying Ethernet Hardware registers and returns it to the user. The following Table 8.9 shows the register returned and the corresponding index.

Table 8.9 Dropped Packet List

DropCount[index]	Error Counter Register	Description
DropPktBufOverrun	FRECR	Buffer Overrun Frame Counter Maximum value: 0xFFFFFFFF
DropPktCrc	CEFCR	CRC Error Frame Counter Maximum value: 0xFFFFFFFF
UndersizePkt	TSFRCR	Undersize Frame Counter (frame length < 64 bytes) Maximum value: 0xFFFFFFFF
OversizePkt	TLFRCR	Oversize Frame Counter (frame length > 1522 bytes) Maximum value: 0xFFFFFFFF
AlgnmtErr	RFCR	Alignment Error Counter (frames received are not an integer number) Maximum value: 0xFFFFFFFF
SqeTestErr	None	Not supported, always 0xFFFFFFFF
DiscInbdPkt	None	Not supported, always 0xFFFFFFFF
ErrInbdPkt	None	Not supported, always 0xFFFFFFFF
DiscOtbdPkt	None	Not supported, always 0xFFFFFFFF
ErrOtbdPkt	None	Not supported, always 0xFFFFFFFF
SnglCollPkt	None	Not supported, always 0xFFFFFFFF
MultCollPkt	None	Not supported, always 0xFFFFFFFF
DfrdPkt	None	Not supported, always 0xFFFFFFFF
LatCollPkt	None	Not supported, always 0xFFFFFFFF
HwDepCtr0	None	Not supported, always 0xFFFFFFFF
HwDepCtr1	None	Not supported, always 0xFFFFFFFF
HwDepCtr2	None	Not supported, always 0xFFFFFFFF
HwDepCtr3	None	Not supported, always 0xFFFFFFFF

Getting Ethernet Statistics

The ETH Driver provide a service to return a list of the statistics compatible to IETF RFC2819.
The following Table 8.10 and Table 8.11 show the statistics available for each index.

Table 8.10 ETH Tx Statistics Counter List

TxStats/TxErrorCounterValues	Statistics Counter Register	Description
TxNumberOfOctets	Implemented by the software	The total number of octets of data transmitted Maximum value: 0xFFFFFFFF
TxNUcastPkts	Implemented by the software	The total number of non-unicast packets transmitted (including broadcast and multicast) Maximum value: 0xFFFFFFFF
TxUniCastPkts	Implemented by the software	The total number of unicast packets transmitted Maximum value: 0xFFFFFFFF
TxDroppedNoErrorPkts	None	Not supported, always 0xFFFFFFFF
TxDroppedErrorPkts	None	Not supported, always 0xFFFFFFFF
TxDeferredTrans	None	Not supported, always 0xFFFFFFFF
TxSingleCollision	None	Not supported, always 0xFFFFFFFF
TxMultipleCollision	None	Not supported, always 0xFFFFFFFF
TxLateCollision	None	Not supported, always 0xFFFFFFFF
TxExcessiveCollison	None	Not supported, always 0xFFFFFFFF

Table 8.11 ETH Rx Statistics Counter List

RxStats	Statistics Counter Register	Description
RxStatsDropEvents	None	Not supported, always 0xFFFFFFFF
RxStatsOctets	Implemented by the software	The total number of octets of data received Maximum value: 0xFFFFFFFF
RxStatsPkts	Implemented by the software	The total number of packets received Maximum value: 0xFFFFFFFF
RxStatsBroadcastPkts	Implemented by the software	The total number of broadcast packets received Maximum value: 0xFFFFFFFF
RxStatsMulticastPkts	MAFCR	The total number of multicast frame received Maximum value: 0xFFFFFFFF
RxStatsCrcAlignErrors	CEFCR + RFCR	The total number of CRC + Alignment Error frames received Maximum value: 0xFFFFFFFF
RxStatsUndersizePkts	TSFRCR	The total number of Undersize (<64 bytes) Frames received Maximum value: 0xFFFFFFFF
RxStatsOversizePkts	TLFRCR	The total number of oversize (> 1522 bytes) Frames received Note: 1522 bytes because VLAN is supported.
RxStatsFragments	None	Not supported, always 0xFFFFFFFF
RxStatsJabbers	None	Not supported, always 0xFFFFFFFF
RxStatsCollisions	None	Not supported, always 0xFFFFFFFF
RxStatsPkts64Octets	Implemented by the software	The total number of packets received that are 64 octets in length. Maximum value: 0xFFFFFFFF
RxStatsPkts65to127Octets	Implemented by the software	The total number of packets received that are between 65 and 127 octets in length. Maximum value: 0xFFFFFFFF
RxStatsPkts128to255Octets	Implemented by the software	The total number of packets received that are between 128 and 255 octets in length. Maximum value: 0xFFFFFFFF
RxStatsPkts256to511Octets	Implemented by the software	The total number of packets received that are between 256 and 511 octets in length. Maximum value: 0xFFFFFFFF
RxStatsPkts512to1023Octets	Implemented by the software	The total number of packets received that are between 512 and 1023 octets in length. Maximum value: 0xFFFFFFFF
RxStatsPkts1024to1518Octets	Implemented by the software	The total number of packets received that are between 1024 and 1522 octets in length. Note: 1522 octets because VLAN is supported.
RxUnicastFrames	Implemented by the software	The number of subnetwork-unicast packets delivered to a higher-layer protocol. Maximum value: 0xFFFFFFFF

Providing timestamp settings for transmit frame
This sub-module provides a service to enable the timestamp information of the transmission frame.
The API related to this sub-module is Eth_EnableEgressTimeStamp().

Getting timestamp of transmitted frame
This sub-module provides a service to provide the timestamp information of the transmitted frame.
The API related to this sub-module is Eth_GetEgressTimeStamp().

Getting timestamp of received frame
This sub-module provides a service to provide the timestamp information of the received frame.
The API related to this sub-module is Eth_GetIngressTimeStamp().

Getting current time by the gPTP timer
This sub-module provides a service to provide the current time information by the gPTP timer.
The API related to this sub-module is Eth_GetCurrentTime().

Feature to adjust gPTP timer
This sub-module provides a service to adjust the gPTP timer.
The sub-module to set the timer increment value is used to adjust the deviation from the grand master clock.
The sub-module to set the timer offset value is used to adjust the difference of the network time.
The API related to this sub-module is Eth_SetIncrementTimeForGptp() and Eth_SetOffsetTimeForGptp().

IEEE1722 packet filtering functionality
This sub-module provides a service to filter the IEEE1722 packets. The filtered frames are queued in a dedicated receive stream queue.
The API related to this sub-module is Eth_UpdateStreamFilter().

8.4 ETH Driver Component Header and Source File Description

This section explains the ETH Driver Component's source and header files. These files must be included in the project application while integrating with other modules.

The C header file generated by ETH Driver Generation Tool:

- Eth_Cfg.h

The C source file generated by ETH Driver Component Code Generation Tool:

- Eth_PBcfg.c

The ETH Driver Component C header files and Component source files:
Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.3.13.3 Folder Structure.

8.5 Application Programming Interface

This section explains the Data types and APIs provided by the ETH Driver Component to the Upper layers.

8.5.1 Imported Types

This section explains the Data types imported by the ETH Driver Component and lists its dependency on other modules.

8.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed:

- Std_VersionInfoType
- Std_ReturnType

8.5.1.2 Other Module Types

In this section all types included in the Dem.h are listed.

- Dem_EventIdType
- Dem_EventStatusType

In this section all types included in the ComStack_Types are listed.

- BufReq_ReturnType

8.5.1.3 Eth General Type

In this section all types included in the Eth_GeneralTypes.h are listed:

- Eth_DataType
- Eth_BufIdxType
- Eth_FilterActionType
- Eth_FrameType
- Eth_ModeType
- Eth_RxStatusType
- Eth_TimeStampQualType
- Eth_TimeStampType
- Eth_TimeIntDiffType
- Eth_CounterType
- Eth_RxStatsType
- Eth_TxStatsType
- Eth_TxErrorCounterValuesType

8.5.2 Type Definitions

This section explains the type definitions of ETH Driver Component according to AUTOSAR Specification. The following Table 8.12 lists them.

Table 8.12 Eth_ConfigType

Name:	Eth_ConfigType
Type:	Structure
Description	ETH Driver Initialization configuration

8.5.3 Function Definitions

The following Table 8.13 lists the available AUTOSAR APIs.

Table 8.13 API Provided by ETH Driver Component

API
Eth_Init
Eth_SetControllerMode
Eth_GetControllerMode
Eth_GetPhysAddr
Eth_SetPhysAddr
Eth_UpdatePhysAddrFilter
Eth_WriteMii
Eth_ReadMii
Eth_GetCurrentTime
Eth_EnableEgressTimeStamp
Eth_GetEgressTimeStamp
Eth_GetIngressTimeStamp
Eth_ProvideTxBuffer
Eth_Transmit
Eth_Receive
Eth_TxConfirmation
Eth_GetVersionInfo
Eth_MainFunction
Eth_GetCounterValues
Eth_GetRxStats
Eth_GetTxStats
Eth_GetTxErrorCounterValues

8.5.3.1 **Renesas Original API**

The following Table 8.14 lists the available RENESAS APIs.

Table 8.14 API Provided by ETH Driver Component for the Renesas Original

API
Eth_UpdateStreamFilter
Eth_SetIncrementTimeForGptp
Eth_SetOffsetTimeForGptp

Table 8.15 to Table 8.17 shows the specifications of each RENESAS APIs.

Table 8.15 Eth_UpdateStreamFilter

Function Name:	Eth_UpdateStreamFilter		
Syntax:	FUNC(Std_ReturnType, ETH_PUBLIC_CODE) Eth_UpdateStreamFilter (CONST(uint8, AUTOMATIC) CtrlIdx, CONST(uint8, AUTOMATIC) QueIdx, CONSTP2CONST(uint8, AUTOMATIC, ETH_APPL_DATA) StreamIdPtr)		
Service Id:	0xA0		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	CtrlIdx	Index of the controller	
		Range:	0.. The number of controllers configured – 1
	QueIdx	Index of the receive queue	
		Range:	2.. The number of AVB stream receive queue – 1
	StreamIdPtr	Update the stream id for separate filtering	
		Range:	Valid pointer
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	E_OK	filter was successfully changed	
	E_NOT_OK	filter could not be changed and the DET is ON	
Description:			
Preconditions:	Component requires the previous controller initialization using Eth_Init.		

Remarks:	This function is available if pre-compile configuration ETH_STREAM_FILTERING == STD_ON.	
DET/DEM Errors handled:	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_PARAM	Parameter (In) QueIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (In) StreamIdPtr is NULL.
	ETH_E_INV_MODE	Any of the followings: The controller mode is STANDBY.

Table 8.16 Eth_SetIncrementTimeForGptp

Function Name:	Eth_SetIncrementTimeForGptp		
Syntax:	FUNC(Std_ReturnType, ETH_PUBLIC_CODE) Eth_SetIncrementTimeForGptp (CONST(uint8, AUTOMATIC) CtrlIdx, CONST(uint32, AUTOMATIC) IncVal)		
Service Id:	0xA1		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	CtrlIdx	Index of the controller	
		Range:	0.. The number of controllers configured – 1
	IncVal	Increment value for gPTP timer	
		Range:	0x00000000..0xFFFFFFFF
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	E_OK	gPTP increment register was successfully changed	
	E_NOT_OK	gPTP increment register could not be changed and the DET is ON	
Description:			
Preconditions:	Component requires the previous controller initialization using Eth_Init.		
Remarks:	None		

DET/DEM Errors handled:	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_INV_MODE	Any of the followings: The controller mode is STANDBY.
	ETH_E_INV_PARAM	Parameter (In) IncVal is out of range.

Table 8.17 Eth_SetOffsetTimeForGptp

Function Name:	Eth_SetOffsetTimeForGptp		
Syntax:	FUNC(Std_ReturnType, ETH_PUBLIC_CODE) Eth_SetOffsetTimeForGptp (CONST(uint8, AUTOMATIC) CtrlIdx, CONSTP2CONST(Eth_TimeStampType, AUTOMATIC, ETH_APPL_DATA) pTimeOffsetPtr)		
Service Id:	0xA2		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	CtrlIdx	Index of the controller	
		Range:	0.. The number of controllers configured – 1
	pTimeOffsetPtr	Offset value for gPTP timer	
		Range:	Valid pointer pTimeOffsetPtr->nanoseconds: 0.. 0x3B9AC9FF pTimeOffsetPtr->seconds: 0.. 0xFFFFFFFF pTimeOffsetPtr->secondsHi: 0.. 0xFFFF
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	E_OK	gPTP offset register was successfully changed	
	E_NOT_OK	gPTP offset register could not be changed and the DET is ON	
Description:			

Preconditions:	Component requires the previous controller initialization using Eth_Init.	
Remarks:	None	
DET/DEM Errors handled:	ETH_E_UNINIT	The function called before Eth_Init has been called.
	ETH_E_INV_CTRL_IDX	Parameter (In) CtrlIdx is out of range.
	ETH_E_PARAM_POINTER	Parameter (In) pTimeOffsetPtr is NULL.
	ETH_E_INV_MODE	Any of the followings: The controller mode is STANDBY.

8.5.4 Preemption of APIs

The following Table 8.18 and Table 8.19 specify the preemption for each API that can be invoked at the same time as the API.

-: cannot be invoked at the same time
√: can be invoked at the same time

Table 8.18 Preemption Table of APIs of the ETH Driver (1/2)

	Eth_Init	Eth_SetControllerMode	Eth_GetControllerMode	Eth_GetPhysAddr	Eth_SetPhysAddr	Eth_UpdatePhysAddrFilter	Eth_WriteMii	Eth_ReadMii	Eth_GetCurrentTime	Eth_EnableEgressTimeStamp	Eth_GetEgressTimeStamp	Eth_GetIngressTimeStamp	Eth_GetCounterValues	Eth_GetRxStats	Eth_GetTxStats	Eth_TxErrorCounterValues	Eth_ProvideTxBuffer	Eth_Transmit	Eth_Receive	Eth_TxConfirmation	Eth_GetVersionInfo	Eth_MainFunction	Eth_UpdateStreamFilter	Eth_SetIncrementTimeForGptp	Eth_SetOffsetTimeForGptp
Eth_Init	-																								
Eth_SetControllerMode	-	-																							
Eth_GetControllerMode	-	-	-																						
Eth_GetPhysAddr	-	√	√	-																					
Eth_SetPhysAddr	-	-	√	-	-																				
Eth_UpdatePhysAddrFilter	-	-	√	√	√	-																			
Eth_WriteMii	-	-	√	√	√	√	-																		
Eth_ReadMii	-	-	√	√	√	√	-	-																	
Eth_GetCurrentTime	-	-	√	√	√	√	√	√	-																
Eth_EnableEgressTimeStamp	-	-	√	√	√	√	√	√	√	-															
Eth_GetEgressTimeStamp	-	-	√	√	√	√	√	√	√	√	-														
Eth_GetIngressTimeStamp	-	-	√	√	√	√	√	√	√	√	√	-													
Eth_GetCounterValues	-	-	√	√	√	√	√	√	√	√	√	√	-												
Eth_GetRxStats	-	-	√	√	√	√	√	√	√	√	√	√	√	-											
Eth_GetTxStats	-	-	√	√	√	√	√	√	√	√	√	√	√	√	-										
Eth_GetTxErrorCounterValues	-	-	√	√	√	√	√	√	√	√	√	√	√	√	√	-									

Table 8.19 Preemption Table of APIs of the ETH Driver (2/2)

	Eth_Init	Eth_SetControllerMode	Eth_GetControllerMode	Eth_GetPhysAddr	Eth_SetPhysAddr	Eth_UpdatePhysAddrFilter	Eth_WriteMii	Eth_ReadMii	Eth_GetCurrentTime	Eth_EnableEgressTimeStamp	Eth_GetEgressTimeStamp	Eth_GetIngressTimeStamp	Eth_GetCounterValues	Eth_GetRxStats	Eth_GetTxStats	Eth_TxErrorCounterValues	Eth_ProvideTxBuffer	Eth_Transmit	Eth_Receive	Eth_TxConfirmation	Eth_GetVersionInfo	Eth_MainFunction	Eth_UpdateStreamFilter	Eth_SetIncrementTimeForGptp	Eth_SetOffsetTimeForGptp
Eth_ProvideTxBuffer	-	-	√	√	-	√	√	√	√	√	√	√	√	√	√	√	√								
Eth_Transmit	-	-	√	√	-	√	√	√	√	-	√	√	√	√	√	√	√	-							
Eth_Receive	-	-	√	√	-	-	√	√	√	√	√	√	√	√	√	√	√	√	-						
Eth_TxConfirmation	-	-	√	√	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	-					
Eth_GetVersionInfo	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√				
Eth_MainFunction	√	*1	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	-			
Eth_UpdateStreamFilter	-	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	-		
Eth_SetIncrementTimeForGptp	-	-	√	√	√	√	√	√	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	-	
Eth_SetOffsetTimeForGptp	-	-	√	√	√	√	√	√	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	-

Note:
*1: Eth_SetControllerMode must not preempt Eth_MainFunction, but Eth_MainFunction can preempt Eth_SetControllerMode.
In other words, Eth_SetControllerMode should be invoked in the lower priority task than Eth_MainFunction or sequentially.

8.6 Development and Production Errors

8.6.1 ETH Driver Component Development Errors

The following Table 8.20 and Table 8.21 contain the DET errors reported by ETH Driver Component. These errors are reported to the Default Error Tracer Module, when the ETH Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 8.20 DET Errors of ETH Driver Component (1/2)

Sl. No.	1
Error Code	ETH_E_INV_CTRL_IDX
Value	0x01
Related API(s)	Eth_SetControllerMode, Eth_GetControllerMode, Eth_GetPhysAddr, Eth_SetPhysAddr, Eth_UpdatePhysAddrFilter, Eth_WriteMii, Eth_ReadMii, Eth_GetCurrentTIme, Eth_EnableEgressTimeStamp, Eth_GetEgressTimeStamp, Eth_GetIngressTimeStamp, Eth_ProvideTxBuffer, Eth_Transmit, Eth_Receive, Eth_TxConfirmation, Eth_GetCounterValues, Eth_GetRxStats, Eth_GetTxStats, Eth_GetTxErrorCounterValues, Eth_UpdateStreamFilter, Eth_SetIncrementTimeForGptp, Eth_SetOffsetTimeForGptp
Source of Error	When the API service is invoked with invalid CtrlIdx.
Sl. No.	**2**
Error Code	ETH_E_UNINIT
Value	0x02
Related API(s)	Eth_SetControllerMode, Eth_GetControllerMode, Eth_GetPhysAddr, Eth_SetPhysAddr, Eth_UpdatePhysAddrFilter, Eth_WriteMii, Eth_ReadMii, Eth_GetCurrentTIme, Eth_EnableEgressTimeStamp, Eth_GetEgressTimeStamp, Eth_GetIngressTimeStamp, Eth_ProvideTxBuffer, Eth_Transmit, Eth_Receive, Eth_TxConfirmation, Eth_GetCounterValues, Eth_GetRxStats, Eth_GetTxStats, Eth_GetTxErrorCounterValues, Eth_UpdateStreamFilter, Eth_SetIncrementTimeForGptp, Eth_SetOffsetTimeForGptp
Source of Error	When the API service is invoked before initialization.
Sl. No.	**3**
Error Code	ETH_E_PARAM_POINTER
Value	0x03
Related API(s)	Eth_Init, Eth_GetControllerMode, Eth_GetPhysAddr, Eth_SetPhysAddr, Eth_UpdatePhysAddrFilter, Eth_ReadMii, Eth_GetCurrentTIme, Eth_GetEgressTimeStamp, Eth_GetIngressTimeStamp, Eth_ProvideTxBuffer, Eth_Transmit, Eth_Receive, Eth_GetVersionInfo, Eth_GetCounterValues, Eth_GetRxStats, Eth_GetTxStats, Eth_GetTxErrorCounterValues, Eth_UpdateStreamFilter, Eth_SetOffsetTimeForGptp
Source of Error	When the API service is invoked with Invalid pointer/Address.

Table 8.21 DET Errors of ETH Driver Component (2/2)

Sl. No.	4
Error Code	ETH_E_INV_PARAM
Value	0x04
Related API(s)	Eth_UpdatePhysAddrFilter, Eth_WriteMii, Eth_ReadMii, Eth_EnableEgressTimeStamp, Eth_Transmit, Eth_Receive, Eth_UpdateStreamFilter, Eth_SetIncrementTimeForGptp
Source of Error	When the API service is invoked with invalid parameter.
Sl. No.	**5**
Error Code	ETH_E_INV_MODE
Value	0x05
Related API(s)	Eth_SetPhysAddr, Eth_UpdatePhysAddrFilter, Eth_Transmit, Eth_Receive, Eth_TxConfirmation, Eth_UpdateStreamFilter, Eth_SetIncrementTimeForGptp, Eth_SetOffsetTimeForGptp, Eth_ProvideTxBuffer, Eth_GetCurrentTime, Eth_GetEgressTimeStamp, Eth_GetIngressTimeStamp
Source of Error	When the API service is invoked in the unallowed controller mode. See 8.2.1 for the allowed controller modes.
Sl. No.	**6**
Error Code	ETH_E_INVALID_DATABASE
Value	0xEF
Related API(s)	Eth_Init
Source of Error	When the API service is incorrect database.

8.6.2 ETH Driver Component Production Errors

The following information will help the user use the ETH Driver Component software efficiently:

Eth_Init should be not called multiple times to prevent unwanted change to configured value.

The following Table 8.22 and Table 8.23 contain the DEM errors reported by ETH Driver Component. These are the hardware errors reported during runtime.

Table 8.22 DEM Errors of ETH Driver Component (1/2)

Sl. No.	1
Error Code	ETH_E_RX_FRAMES_LOST
Related API(s)	Eth_MainFunction
Source of Error	Reported when Register 'FRECR' in underlying MAC block increased the previous call of Eth_MainFunction. Note that this error means that the frames(s) are dropped due to the overflow of the internal FIFO. The dropped frames due to the receive buffer shortage cannot be detected.
Sl. No.	**2**
Error Code	ETH_E_CRC
Related API(s)	Eth_MainFunction
Source of Error	Reported when Register 'CEFCR' in underlying MAC block contains a value different than zero.
Sl. No.	**3**
Error Code	ETH_E_UNDERSIZEFRAME
Related API(s)	Eth_MainFunction
Source of Error	Reported when Register 'TSFRCR' in underlying MAC block contains a value different than zero.
Sl. No.	**4**
Error Code	ETH_E_OVERSIZEFRAME
Related API(s)	Eth_MainFunction
Source of Error	Reported when Register 'TLFRCR' in underlying MAC block contains a value different than zero.
Sl. No.	**5**
Error Code	ETH_E_ALIGNMENT
Related API(s)	Eth_MainFunction
Source of Error	Reported when Register 'RFCR' in underlying MAC block contains a value different than zero.

Table 8.23 DEM Errors of ETH Driver Component (2/2)

Sl. No.	6
Error Code	ETH_E_REGISTER_CORRUPTION
Related API(s)	Eth_Init, Eth_MainFunction
Source of Error	Reported when the register stuck, or the register corruption is detected on any register.
Sl. No.	**7**
Error Code	ETH_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	ETH_AVBnDATAISR, ETH_AVBnERRORISR, ETH_AVBnMACISR, ETH_AVBnDATAISR_CAT2, ETH_AVBnERRORISR_CAT2, ETH_AVBnMACISR_CAT2 (n =0..2)
Source of Error	Reported when unintended interrupt occurred.
Sl. No.	**8**
Error Code	ETH_E_DMA_ERROR
Related API(s)	ETH_AVBn_ERROR_ISR, ETH_AVBn_ERROR_ISR_CAT2 (n =0..2)
Source of Error	Reported when the DMA address error when interrupt occurs.
Sl. No.	**9**
Error Code	ETH_E_ACCESS
Related API(s)	Eth_Init, Eth_SetControllerMode
Source of Error	Reported when the E-MAC access error occurs.

8.7 ETH Driver Component Runtime Errors

ETH module does not support runtime errors.

8.8 Memory Organization

The following table depict a typical memory organization, which must be met for proper functioning of ETH Driver Component software.

Table 8.24 ROM Sections of the ETH Driver

Section Name	Alignment (*1)	Description
ETH_PUBLIC_CODE_ROM	-	This section contains codes which belong to the API functions of the ETH Driver.
ETH_PRIVATE_CODE_ROM	-	This section contains codes which belong to the internal functions of the ETH Driver.
ETH_FAST_CODE_ROM	-	This section contains codes which belong to the ISRs of the ETH Driver.
CONST_ROM_32BIT	-	This section contains 32bits constants of the ETH Driver.
ETH_CFG_DBTOC_UNSPECIFIED	-	This section contains post-build config data table of the ETH Driver.
ETH_CFG_DATA_UNSPECIFIED	-	This section contains post-build config data table of the ETH Driver.

Table 8.25 RAM sections of the ETH Driver (1/2)

Section Name	Alignment (*1)	Description
RAM_UNSPECIFIED	-	This section contains the variables with environment-dependent size and with the initial value of the ETH Driver. This section should be initialized with the initial values before Eth_Init.
RAM_16BIT	-	This section contains 16bits variables that need to be initialized before Eth_Init. This section should be initialized with the initial values before Eth_Init.
RAM_32BIT	-	This section contains 32bits variables that need to be initialized before Eth_Init. This section should be initialized with the initial values before Eth_Init.
NOINIT_RAM_UNSPECIFIED	-	This section contains miscellaneous variables that need not be initialized before Eth_Init. However, for the ECC feature, this section should be initialized by 32bits before Eth_Init.
NOINIT_RAM_32BIT	-	This section contains 32bits variables that need not be initialized before Eth_Init.
ETH_PORT_RAM_0	-	This section contains the post-build heap memory of the ETH PORT0. It is used to store transmission and reception buffer. This section does not need to be initialized.
ETH_PORT_RAM_1	-	This section contains the post-build heap memory of the ETH PORT1. It is used to store transmission and reception buffer. This section does not need to be initialized.
ETH_PORT_RAM_2	-	This section contains the post-build heap memory of the ETH PORT2. It is used to store transmission and reception buffer. This section does not need to be initialized.

Note: *1 means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

Table 8.26 RAM sections of the ETH Driver (2/2)

Section Name	Alignment (*1)	Description
ETH_PORT_DESCRIPTOR_0	-	This section contains the post-build heap memory of the ETH PORT0. It is used exclusively to store EAVB-IF Descriptors. This must be allocated into non-cached area to avoid data incoherence. This section does not need to be initialized.
ETH_PORT_DESCRIPTOR_1	-	This section contains the post-build heap memory of the ETH PORT1. It is used exclusively to store EAVB-IF Descriptors. This must be allocated into non-cached area to avoid data incoherence. This section does not need to be initialized.
ETH_PORT_DESCRIPTOR_2	-	This section contains the post-build heap memory of the ETH PORT2. It is used exclusively to store EAVB-IF Descriptors. This must be allocated into non-cached area to avoid data incoherence. This section does not need to be initialized.

Note: *1 means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

8.9 Device-Specific Information

The device supports the following devices:
Refer to "[1] R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 5.1 Product.

8.9.1 Interaction between the User and ETH Driver Component

The following sections detail the services supported by the ETH Driver Component to the upper layer users and the mapping of the channels to the hardware units:

8.9.1.1 Channel Mapping

The following Table 8.27 shows the ETH channel information.

Table 8.27 HW ETH Channel Mapping

Sl. No.	Hardware Channel
1	AVB0
2	AVB1
3	AVB2

8.9.1.2 ISR Functions

The following Table 8.28 shows the ISR information.

Table 8.28 ISR Function for the Device

Interrupt Source	Name of the ISR Function
AVB0	ETH_AVB0DATAISR
	ETH_AVB0ERRISR
	ETH_AVB0MACISR
	ETH_AVB0DATAISR_CAT2
	ETH_AVB0ERRISR_CAT2
	ETH_AVB0MACISR_CAT2
AVB1	ETH_AVB1DATAISR
	ETH_AVB1ERRISR
	ETH_AVB1MACISR
	ETH_AVB1DATAISR_CAT2
	ETH_AVB1ERRISR_CAT2
	ETH_AVB1MACISR_CAT2
AVB2	ETH_AVB2DATAISR
	ETH_AVB2ERRISR
	ETH_AVB2MACISR
	ETH_AVB2DATAISR_CAT2
	ETH_AVB2ERRISR_CAT2
	ETH_AVB2MACISR_CAT2

8.9.1.3 Translation Header File

Refer to “R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 3.2.4 Translation Header File.

8.9.1.4 Parameter Definition File

Refer to “[1] R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” section 3.2.3 Parameter Definition File.

8.9.2 Multi-Core / Multi-Instantiation

ETH driver does not support multi-core and multi-instantiation for this device.

8.10 Non-AUTOSAR environment integration

The ETH Driver Components for Renesas R-Car Series, 4th Generation SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the ETH Driver Components to Non-AUTOSAR environment.

8.10.1 Stub modules handling

8.10.1.1 Os

The Os stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar_version>\Os

In the AUTOSAR environment, ETH Driver is relying on APIs of GetCounterValue() and GetElapsedValue() provided by the OS module to get the current count value of the specified counter.

StatusType GetCounterValue(CounterType CounterID, TickRefType Value)

Current Os stub implementation uses SCMT (System Timer) IP to simulate an Os counter. This counter has OsSecondsPerTick ~ 0.0000076s and will always run with sample application. GetCounterValue will return the value of that counter at calling time, GetElaspedValue will return the value between current value of that counter and previously read value.

Non-AUTOSAR users can modify the provided GetCounterValue and GetElapsedValue APIs as needed e.g. with other hardware counter or unique CounterID for each MCAL module.

8.10.1.2 Det

The Det stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\Det
In the AUTOSAR environment, ETH Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g. ETH Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:
Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)
Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.
Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

8.10.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.
The SchM (Basic Software Scheduler) stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\Rte
ETH driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when ETH_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON.
With Non-AUTOSAR environment, user needs to prepare SchM stub to user ETH driver as following:
Write SchM functions with prototype as below:
void SchM_Enter_Eth_ETH_INTERRUPT_CONTROL_PROTECTION(void);
void SchM_Exit_Eth_ETH_INTERRUPT_CONTROL_PROTECTION(void);
void SchM_Enter_Eth_ETH_RAM_DATA_PROTECTION(void);
void SchM_Exit_Eth_ETH_RAM_DATA_PROTECTION(void);

8.10.1.4 Dem

The Dem stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\Dem
In the AUTOSAR environment, ETH Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error e.g. ETH Driver access failure, ETH frame is lost… The API prototype is as of follow:

Std_ReturnType Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)
Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.
Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

8.10.1.5 **EthIf**

The EthIf stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar_version>\EthIf
In the AUTOSAR environment, EthIf is the only upper layer of the ETH Driver Components. It provides a hardware independent interface to the Communication Service layers (TCP/IP, EthSM, CDD) as specified in Figure 1-1: System Overview of AUTOSAR Architecture.
Non-AUTOSAR users can implement their own ETH stack using the APIs provided by the ETH Driver Component.

8.10.1.6 **EthIf_RxIndication**

ETH Driver invokes EthIf_RxIndication API provided by the EthIf module when an Ethernet frame is successfully received. The API prototype is as of follow:
void EthIf_RxIndication(uint8 CtrlIdx, Eth_FrameType FrameType, boolean IsBroadcast, const uint8* PhysAddrPtr, const Eth_DataType* DataPtr, uint16 LenByte)
Current implementation of this API simply copies the content pointed by DataPtr to character array RxEthFrame.

8.10.1.7 **EthIf_TxConfirmation**

ETH Driver invokes EthIf_TxConfirmation API provided by the EthIf module when an Ethernet frame is successfully transmitted. The API prototype is as of follow:
void EthIf_TxConfirmation(uint8 CtrlIdx, Eth_BufIdxType BufIdx, Std_ReturnType Result)
Current implementation of this API simply toggles the global variable TxConfirmed to 1.

8.10.1.8 **EthIf_CtrlModeIndication**

ETH Driver invokes EthIf_CtrlModeIndication API provided by the EthIf module when the controller mode changed. The API prototype is as of follow:
void EthIf_CtrlModeIndication(uint8 CtrlIdx, Eth_ModeType CtrlMode)
Current implementation of this API simply increases the global variables EthModeDownCnt and EthModeActiveCnt.

8.10.1.9 **EthTrcv**

The EthTrcv stub files are organized in the following folder:
\rel\common \generic\stubs\<Autosar version>\EthTrcv
In the AUTOSAR environment, EthTrcv belongs to the Microcontroller Abstraction Layer. It provides to the upper layer (Ethernet Interface) a hardware independent interface comprising multiple equal transceivers.
Non-AUTOSAR users can implement their own Ethernet stack using the APIs provided by the ETH Driver Component.

8.10.1.10 **EthTrcv_ReadMiiIndication**

ETH Driver invokes EthTrcv_ReadMiiIndication API provided by the EthTrcv module when the MII access finished. The API prototype is as of follow:
void EthTrcv_ReadMiiIndication (uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx, uint16 RegVal)
Current implementation of this API simply copies the register Id and register value of the MII read access to global variables.

8.10.1.11 **EthTrcv_WriteMiiIndication**

ETH Driver invokes EthTrcv_ReadMiiIndication API provided by the EthTrcv module when the MII access finished. The API prototype is as of follow:
void EthTrcv_WriteMiiIndication (uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx)
Current implementation of this API simply copies the register Id of the MII write access to a global variable.

8.10.1.12 **EcuM_SetWakeupEvent**

Current implementation of this API simply sets the wakeup event.
void EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType sources)

8.10.1.13 **EcuM_ValidateWakeupEvent**

After wakeup, the ECU State Manager will stop the process during the WAKEUP VALIDATION state/sequence to wait for validation of the wakeup event.This API service is used to indicate to the ECU Manager module that the wakeup events indicated in the sources parameter have been validated.
void EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType sources)

8.10.1.14 **EcuM_CheckWakeup**

This callout is called by the EcuM to poll a wakeup source. It shall also be called by the ISR of a wakeup source to set up the PLL and check other wakeup sources that may be connected to the same interrupt.
void EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)

8.10.2 Callback function usage

The ETH Driver Component does not provide any callback functions.

8.10.3 Scheduled function usage

In the AUTOSAR environment, ETH Driver Component scheduled functions shall be invoked periodically by the Scheduler Manager implemented by EcuM module. The ETH Driver Component scheduled functions are listed below:

- Eth_MainFunction

Non AUTOSAR user can invoke above API periodically using their own implementation e.g. hardware timer to utilize those APIs functionality.

8.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:

V4H: \rel\V4H\common_family\src\arm\Interrupt_VectorTable.c
\rel\V4H\common_family\include\arm\Interrupt_Cfg.h

9.FLS

9.1 Overview

The purpose of this chapter is to describe the information related to FLS Driver Component.

This chapter shall be used as reference by the users of FLS Driver Component. The system overview of complete AUTOSAR architecture is shown in the Figure 9.1 below.

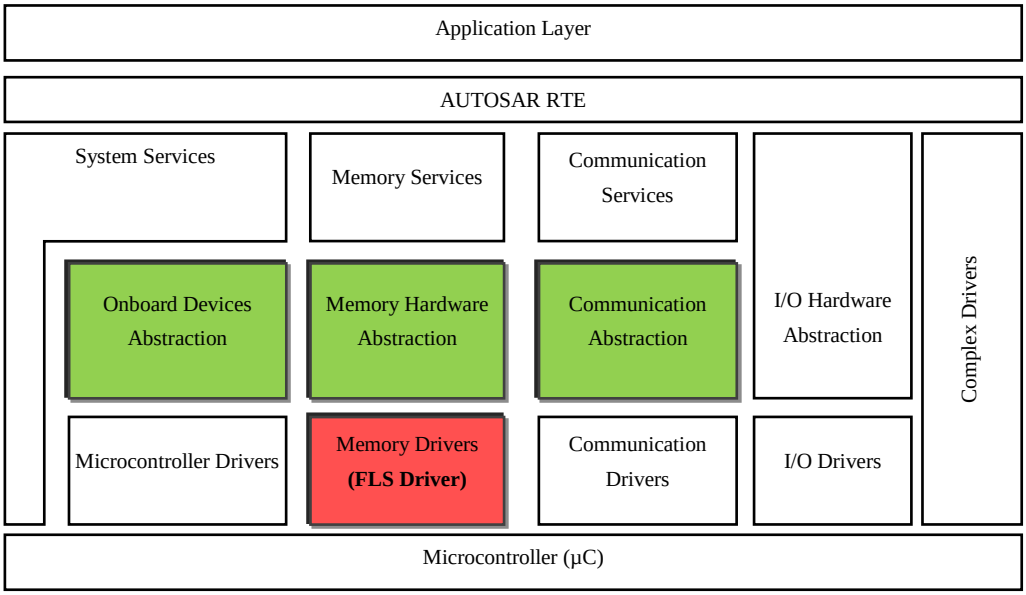


Figure 9.1 System Overview of AUTOSAR Architecture

The FLS Driver is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment. As shown above, the FLS Driver is accessed by “Onboard Device Abstraction, Memory HW Abstraction and Communication HW Abstraction layers.
The Figure in the following page depicts the FLS Driver as part of layered AUTOSAR MCAL Layer:

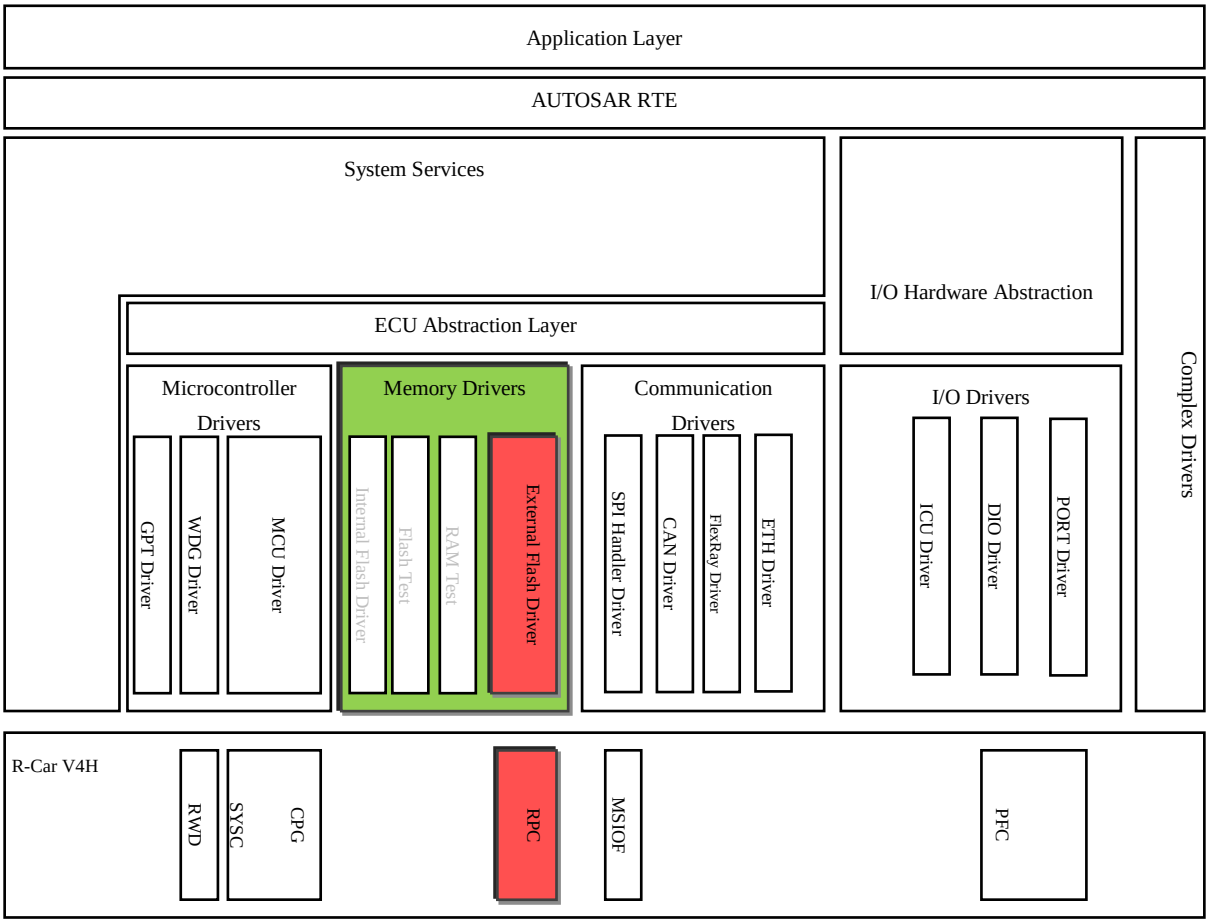


Figure 9.2 System Overview of the FLS Driver in AUTOSAR MCAL Layer

The FLS Driver Component fulfills requirements of upper layer communication components with respect to mentioned services above. The FLS Driver Component comprises embedded software and the configuration tool to achieve scalability and configurability. It performs the hardware access and offers a hardware independent AUTOSAR API (see AUTOSAR_SWS_FlashDriver.pdf) to the upper layer.

The FLS driver provides services for reading from and writing to devices connected through SPI multi I/O bus controller (RPC).

9.2 Forethoughts

9.2.1 General

Following information will aid the user to use the FLS Driver Component software efficiently:

- The FLS Driver Component does not enable or disable the SoC or Microcontroller power supply. The upper layer should handle this operation.
- The FLS Driver Component does not provide functionalities for setting of Port pins. The Port modules should handle it.
- Start-up code is not implemented by the FLS Driver Component.
- Example code mentioned in this document shall be taken only as a reference for implementation.
- All development errors will be reported to DET by using the API Det_ReportError() provided by DET.
- All runtime errors will be reported to DET by using the API Det_ReportRuntimeError() provided by DET.
- All transient faults will be reported to DET by using the API Det_ReportTransientFault() provided by DET.
- All production errors will not be handled by FLS Driver Component and will be reported to DEM by upper layers of FLS.
- The FLS Driver Component is restricted to Post Build only
- The FLS Driver Component does not provide functionalities for setting of protection flags, boot cluster size, swapping of boot block and flashing of boot block and they are out of scope for FLS Driver Component implementations.
- The FLS Driver Component's job processing function (Fls_MainFunction() API) is a polled function.
- Program execution from external flash is prohibited during flash programming. Therefore, all FLS Components are located in RAM. The FLS components will be copied from external flash to RAM during the startup. The FLS user has to assure that the application for programming control is also located to RAM area during ongoing flash programming operations.
- During activated flash environment, the access to flash is not possible. Hence the user should ensure that all the application and supporting components code that needs to be executed during flash operation need to locate in RAM.
- The FLS Driver Component will only invoke the call back notification functions. However, the implementation of the call back functions is the responsibility of the upper layer.
- The component will support only the user mode of flash memory. Internal mode is not in the scope of this implementation.
- The Fls_Erase() API computes the sectors that need to be erased based on the provided target address and length. When DET is enabled the error will be reported if the length of the bytes to be erased is not in multiples of flash sector size.
- The length of the data that must be programmed on to the flash should be in multiples of flash page. The FLS Driver Component does not pad bytes if the length is not in multiples of flash page. It is the responsibility of the application to pad bytes such that the length of the data is in multiples of flash page.
- The FLS Driver Component is developed to support external devices: serial flash: S25FS512S and hyper flash: S26KS512S of Cypress. Hence the parameters related to other devices are ignored by the FLS Generation Tool.
- In this version, FLS Driver Component only supports one flash programming device type, which setting by the parameter FlsAccess:

  + FLS_SERIAL_FLASH_DEVICE: Serial flash connected to R-Car V4H

  + FLS_HYPER_FLASH_DEVICE: Hyper flash connected to R-Car V4H

- User configures the number of devices via the parameter FlsSelectPinGroup:

  + Serial flash:

    - FLS_QSPI0_PIN_GROUP: Single flash mode is selected. Only one serial flash connected to QSPI0 pin group of R-Car V4H
    - FLS_QSPI0_QSPI1_PIN_GROUP: Dual flash mode is selected. Two serial flash devices connected to QSPI0 and QSPI1 pin group of R-Car V4H (not supported)

+ Hyper flash:
    - FLS_QSPI0_QSPI1_PIN_GROUP: Hyper flash device connected to R-Car V4H.Because of R-Car V4H board limitations, FLS Driver is tested for the following device: ([R-Car V4H]: S25FS512S, [R-Car V4H]: S26KS512S of Cypress vendor. In the case of R-Car V4H, the details of the test constitution of the device become S25FS512S x 1 (single flash) or S26KS512S x 1 (hyper flash).

- All user's configurations for serial flash device place in sub-containers of the container FlsSerialFlashConfig:
    + FlsSerialFlashDDRPattern: include all configurations for DDR pattern.
    + FlsSerialFlashDDRVerify: include all configurations for SDR read operation. It is used for the purpose of verifying values of the DDR pattern area (and other verify processing such as compare data and blank check – when the parameter FlsDDRCalibration configured as true).
    + FlsSerialFlashErase: include all configurations for erase operation.
    + FlsSerialFlashProgram: include all configurations for program (also known as write) operation.
    + FlsSerialFlashRead: include all configurations for read operation.
    + FlsSerialFlashReadHwId: include all configurations for reading the Flash device's hardware identifies operation.
    + FlsSerialFlashReadStatusReg: include all configurations for reading status register operation. The Status Register is where the status of an operation is stored after it is completed. FLS Driver only supported to read a status register. Therefore, it should be containing the following status bits Write Enable Latch (WEL) and Write in Progress (WIP). If a serial flash device supports the error bits of erase and program operation on same a status register, such as S25FS512S (for R-Car V4H) Cypress serial flash, the configuration for these bits should be enabled by the parameters FlsEnableEraseErrorBit and FlsEnableProgramErrorBit in the container FlsSerialFlashGeneral.
    + FlsSerialFlashWriteEnable: include all configurations for enabling the Flash device's Write Enable Latch (WEL) bit operation.
- All user's configurations for hyper flash device place in sub-containers of the container FlsHyperFlashConfig:
    + FlsHyperFlashErase: include all configurations for erase operation.
    + FlsHyperFlashProgram: include all configurations for program (also known as write) operation.
    + FlsHyperFlashRead: include all configurations for read operation.
    + FlsHyperFlashReadHwId: include all configurations for reading the Flash device's hardware identifies operation.
    + FlsHyperFlashReadStatusReg: include all configurations for reading status register operation. The Status Register is where the status of an operation is stored after it is completed. FLS Driver only supported to read a status register. Therefore, it should be containing the following status bit: Device Ready Bit (DRB). If a hyper flash device supports the error bits of erase and program operation on same a status register, such as S26KS512S (for R-Car V4H) Cypress hyper flash, the configuration for these bits should be enabled by the parameters FlsEnableEraseErrorBit and FlsEnableProgramErrorBit in the container FlsHyperFlashGeneral.

- The FLS Driver is only supported and tested by non-secure mode.

- FLS Driver provides the Fls_Suspend() API to suspend the ongoing job (it is a read, write, erase, blank check or compare job) in main function. The suspended jobs can be called to continue by the Fls_Resume() API. If the user does not want to continue doing the suspended jobs, they absolutely can cancel them by the Fls_Cancel() API.
  + Fls_Suspend() API backup the address, command, pointer to data and status of the current job to resume later. The suspend operation processed by Fls_MainFunction() API. A main processing will usually be split into several jobs. When it receives a request to suspend the job, the job part before receiving the request must be done, the rest will be suspended.
  + Fls_Resume() API restore the backup data from Fls_Suspend() API when it called. The resume operation also processed by Fls_MainFunction() API. When it receives a request to resume the suspended jobs, these jobs will be ongoing.
- To support settings specific features of external flash device (such as enable: Quad mode for S25FS512S Cypress serial flash; setting latency code for hyper flash) FLS Driver provides Fls_SendSpecificConfig() API for user. This API can be enabled / disabled via the parameter FlsSpecificConfigApi in the container FlsGeneral. Pointer to configuration set of the container FlsSerialFlashSpecificConfig or FlsHyperFlashSpecificConfig used as an input parameter for this API base on the value of FlsAccess parameter in FlsGeneral container.
  + This API can be configured to send a command / sequence of commands to the external flash device. It is also possible to configure a function to read / write data from any address of an external flash device, including address registers and data regions. This is done through a parameter called FlsOperation of the container FlsSerialFlashSpecificConfig or FlsHyperFlashSpecificConfig:
    - FLS_READ_OPERATION: Fls_SendSpecificConfig() API configured to send a read command to external flash device.
    - FLS_WRITE_OPERATION: Fls_SendSpecificConfig() API configured to send a write command to external flash device.
- FLS Driver provides three APIs for DDR calibration feature, they are Fls_DDRWritePattern(), Fls_DDRVerifyPattern() and Fls_DDRCalibrate().

9.2.2 Preconditions

The following preconditions must be adhered by the user, for proper functioning of the FLS Driver Component:

- User must initialize some other hardware IPs as below before initializing FLS driver.

Table 9.1 Hardware IPs initialization before initializing FLS driver

Related IPs	Values	Description
CPG Write Protect Register (CPGWPR)	CPGWPCR.WPE=1	Enable the write protection.
Clock Pulse Generator (CPG)	RPCCKCR [8:9] = 00	MCU driver needs clear the RPCCKCR [8:9] bits to supply the clock before the RPC-IF module is being used by the FLS driver.
Write Protection Control Register (MFISWPCNTR)	MFISWPCNTR.WPD = 1	Disable the write protection.

- The Fls_PBcfg.c and Fls_Cfg.h files generated by the FLS Generation Tool must be compiled and linked along with FLS Driver Component source files.
- The application must be rebuilt, if there is any change in the Fls_Cfg.h file generated by the FLS Generation Tool.
- The user should ensure that FLS Driver Component APIs requests are invoked with correct input arguments.
- The FLS Driver Component needs to be read the hardware identify of flash device before doing any operation of it, user must disable interrupt before calling Fls_Init() API. If FLS Driver Component is not initialized properly, the behavior of flash operation may be undetermined.

- Since FLS Driver Component support external devices, the configuration parameter FlsExpectedHwId in the configuration container FlsPublishedInformation must be configured. The four bytes in "Manufacturer and Device ID" information of the Flash Device’s Hardware User’s Manual will be used as Flash Device’s Hardware ID.
- The configuration parameter FlsExpectedHwIdMask in the configuration container Fls_General must be configured. This parameter is used as mask value for verifying the value of Flash Device’s Hardware ID by FlsExpectedHwId.
- For example:
- Case 1: The first four bytes to use
  - FlsExpectedHwId to set “0x04030201”.
  - FlsExpectedHwIdMask to set “0xFFFFFFFF”.
- Case 2: The first three bytes to use
  - FlsExpectedHwId to set “0x030201”.
  - FlsExpectedHwIdMask to set “0xFFFFFF”.
- Validation of input parameters is done only when the configured value of the parameter FlsDevErrorDetect is ’true’. Application should ensure:

  + Correct parameters are passed while invoking the APIs when the configured value of FlsDevErrorDetect is ’false’.

  + APIs are called in correct sequences.
- The validation of runtime errors is done when the static configuration parameter FlsRuntimeErrorDetect is enabled.
- The validation of transient faults is done when the static configuration parameter FLS_TRANSIENT_FAULT_DETECT is enabled.
- A mismatch in the version numbers will result in compilation error. Ensure that the correct versions of the header and the source files are used.
- The FLS Driver Component needs to be initialized by calling Fls_Init() API before calling any other FLS functions. Exception is the function Fls_GetVersionInfo() API.
- The Fls_MainFunction() API should be invoked regularly by the Basic Scheduler. Though not specified by AUTOSAR, calling Fls_MainFunction by polling mechanism is also possible. Ensure that the FLS Driver Component is initialized before enabling the invocation of this scheduled function.
- Fls_SetMode() API only valid when external space read enabled. To enable it, user configures the parameter FlsExternalSpaceReadMode to ‘true’. When external space read mode disabled, this API only used to change the maximum read bytes per a main job processing, and to configure this value, continue reading the description below.
- The maximum read bytes per a main job processing configured by the parameters FlsMaxReadFastMode and FlsMaxReadNormalMode. After FLS Driver initialized, the configured value of the parameter FlsDefaultMode will be used for selection this value:

  + MEMIF_MODE_FAST: the configured value of the parameter FlsMaxReadFastMode is the maximum read bytes per a main job processing.

  + MEMIF_MODE_SLOW: the configured value of the parameter FlsMaxReadNormalMode is the maximum read bytes per a main job processing.
- The maximum read bytes per main job processing is used for reading processing. If this value is less than the input parameter length multiple times, job performance will be reduced.

  E.g.: For the maximum read bytes per a main job processing: The configured value of the parameter FlsMaxReadFastMode is ‘A’ value and FlsDefaultMode is ‘MEMIF_MODE_FAST’. When user sets the input parameter ‘Length’ the Fls_Read() API is ‘Length_Value’, then invokes Fls_MainFunction() API to it process the read job, that means Fls_MainFunction() API will break the read job into ‘Length_Value divisible by A’ parts and execute them sequentially.
- The maximum written bytes per a main job processing configured by the parameter FlsMaxWriteNormalMode.
- To get the best performance, the configuration value for the parameter FlsMaxWriteNormalMode and the input parameter ‘Length’ of Fls_Write() API should be a number which is divisible by four.
- FLS Driver Component does not have the configuration for the maximum erased bytes per a main job processing, the default value is one sector per a main job processing.
- Before executing a write operation, please make sure the given address range is erased.

- FLS Driver does not provide any error checking for compatibility of the input parameters: buffer size and length of data. User must ensure that the data length is less than or equal to the buffer size.

  E.g: Fls_Write(Fls_AddressType TargetAddress, P2CONST (uint8, AUTOMATIC, FLS_APPL_CONST) SourceAddressPtr, Fls_LengthType Length). When size of buffer at SourceAddressPtr less than Length, this means that the Length is out of range the SourceAddressPtr; unwanted values may be written to the Flash device or worse, unwanted hardware errors will occur. This case should be avoided.
- If a cancel request is accepted, during an on-going job operation and a previous operation is already suspended, then both operations will be cancelled.
- Reentrant calls of any FLS non-reentrant APIs must be avoided. FLS Driver Component only provides reentrant APIs: Fls_GetStatus(), Fls_GetJobResult(), Fls_GetVersionInfo().
- When HyperFlash is connected, the address extension with DRENR.ADE[3] is not available, and internal address bits [25:0] (64 Mbytes area) are used. Use manual mode when access to the bigger area than 64 Mbytes area.
- It is not possible to nest suspend and/or stand-by and should not suspend then use erase or write to the same memory area with suspended job.

  E.g: Any operation ► suspend ► suspend – is not possible.

  Write or Erase ► suspend ► write or erase to same memory area with suspended job – is not possible.

  Any operation ► suspend ► other operation ► suspend – is not possible.
- The parameter FlsDDRCalibration in the container FlsSerialFlashGeneral is used to enable/disable Fls_DDRVerifyPattern(), Fls_DDRCalibrate() and Fls_DDRWritePattern() APIs. The configured value of the parameter FlsDDRCalibration is described as below:
  - True: Enable Fls_DDRVerify(), Fls_DDRCalibrate() and Fls_DDRWritePattern() APIs.
  - False: Disable Fls_DDRVerify(), Fls_DDRCalibrate() and Fls_DDRWritePattern() APIs. FlsSerialFlashDDRPattern and FlsSerialFlashDDRVerify containers are not used. The configuration of FlsSerialFlashRead container is used for verification and read processes such as blank check, compare data and read data.
- A sector will be used as DDR Pattern Area. This area is used for DDR read to verify the result. DDR Pattern Area cannot be erased/written (programmed). When user calls FLS Driver to erase/write to this area, the FLS Driver will report error ‘FLS_E_PARAM_ADDRESS’ to DET. Other operations are not affected.
- In FlsSerialFlashDDRPattern container, the address of DDR Pattern Area is configured by FlsPatternAddress parameter. A pattern on this area has four bytes, and its value is configured by FlsPatternValue parameter. Number of patterns is also available to configure by FlsNumberOfPattern parameter. As address of the pattern must be a sector start address, it is same as the address in input parameter of Fls_Erase() API. The number of patterns must be less than the value of sector size divided by four (because a pattern has four bytes).
- FLS Driver provides Fls_DDRVerifyPattern() API for user to check the pattern value. In case DDR Pattern Area has been written and this data matched with the value configured by the parameter FlsPatternValue, Fls_DDRVerifyPattern() API returns ‘E_OK’, else it returns ‘E_NOT_OK’. Details for Fls_DDRVerifyPattern() API, see the section 9.5.3.1.
- Fls_DDRCalibrate() API must be invoked before starting DDR transfer, then the value of FlsInternalStrobeDelay will be replaced by new Strobe Delay value as result of DDR Calibration. Otherwise, Fls Driver will get value of FlsInternalStrobeDelay for Strobe Delay as default, but the DDR transfer is not guaranteed.
- Fls_DDRCalibrate() API must be re-invoked when the call to Fls_DDRVerifyPattern() API is completed and the returned result is ‘E_NOT_OK’. This is necessary because DDR timing of data strobe changes with PVT (Process-Voltage-Temperature) condition in both this LSI and the connected device. Fls_DDRCalibrate() API will help user correct these values. In case DDR Pattern Area is not written, ‘E_NOT_OK’ will be returned. Details for Fls_DDRCalibrate() API, see the section 9.5.3.1.
- When using the Fls_SendSpecificConfig() API, user should be sure that all configuration in the container FlsSerialFlashSpecificConfig/ FlsHyperFlashSpecificConfig are correctly. A wrong configuration or a special setting for serial flash may affect its operation.

  E.g.: For S25FS512S Cypress serial flash: Fls_SendSpecificConfig() API invoked to change the address mode of this flash device from 3-bytes to 4-bytes. After mode is successfully installed, the user must configure all the commands

so that they work with 4-bytes mode.

- User must initialize the following hardware IPs as below before initializing FLS module:

Table 9.2 Precondition Hardware IPs before FLS Driver Initialization

Hardware IP	Module Name	Expected Setting	Description
AXI-bus	-	Enable the AXI BUS since RPC-IF communicates with CPU via main memory AXI BUS refer to chapter 18 in HW-UM	FLS is designed assuming the setting of SDRAM address mapping is corrected, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since FLS operation is under controlled by DRAM for memory control.
CPG	MCU	- **CPGWPCR**: Set/clear bit0 of this register to enable/disable write protection of register in CPG. By default, this bit is set (enable write protection). - **CPGWPR**: When write protection is enabled (via register CPGWPCR), user shall write inverse value (which is used to set to the CPG register) to this register before writing the value to CPG register. FLS driver will access this register to set clock for RPC-IF (via register RPCCKCR). If two or more than two applications want to access to this register, then they should use exclusive control and critical section to prevent the conflict accessing. - **RPCCKCR:** control the RPC-IF clock, FLS driver will access this register to set clock for RPC-IF. MCU driver does not need to set this register before executing FLS driver.	This IP enable/disable clock signal of modules (see section 8 Clock Pulse Generator of HW UM The setting of these register is provided in MCU Driver.

		If two or more than two applications want to access to this register, then they should use exclusive control and critical section to prevent the conflict accessing. User only configures the bus domain via module MCU to support domain ID setting.	
MFIS	-	Write Access protection in MFIS should be disable. (refer to 16.2.16 Write Protection Control Register (MFISWPCNTR) in HW-UM)	FLS is designed assuming the setting of this protection mechanism is disabled, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed since it may not write to registers of MFIS.
PFC	PORT	**- PMMR**: To access GPSR and IPSR register, user shall write inverse value of GPSR/IPSR's value to this register. - **GPSR2**: bits 0, 1 and 2 FLS driver doesn't use functionality of these bit, they should not be changed when FLS driver is executing.	This IP set the function of multiplexed pins and controlling the pull-up on each LSI pin section 7 Pin Function The setting of these register can be configured by Configuration Description File and APIs provided by PORT Driver.

9.2.3 Data Consistency

To support the reentrancy and interrupt services, the FLS Software component will ensure the data consistency while accessing their own RAM storage or hardware registers. The FLS module will use the following macro for the respective higher and lower version.

```
#define FLS_ENTER_CRITICAL_SECTION (Exclusive_Area)
SchM_Enter_Fls_##Exclusive_Area()

#define FLS_EXIT_CRITICAL_SECTION (Exclusive_Area)
SchM_Exit_Fls_##Exclusive_Area()
```

The functions SchM_Enter_Fls_<Exclusive Area> and SchM_Exit_ Fls_<Exclusive Area> can be disabled by disabling the configuration parameter 'FlsCriticalSectionProtection'. The flowchart will indicate the flow with the pre-compile option 'FlsCriticalSectionProtection' enabled.
The following exclusive areas along with scheduler services are used to provide data integrity for shared resources:

- FLS_RAM_DATA_PROTECTION

9.3 Architecture Details

The FLS Software architecture is shown in the following figure. The FLS user shall directly use the APIs to configure and execute the FLS conversions:

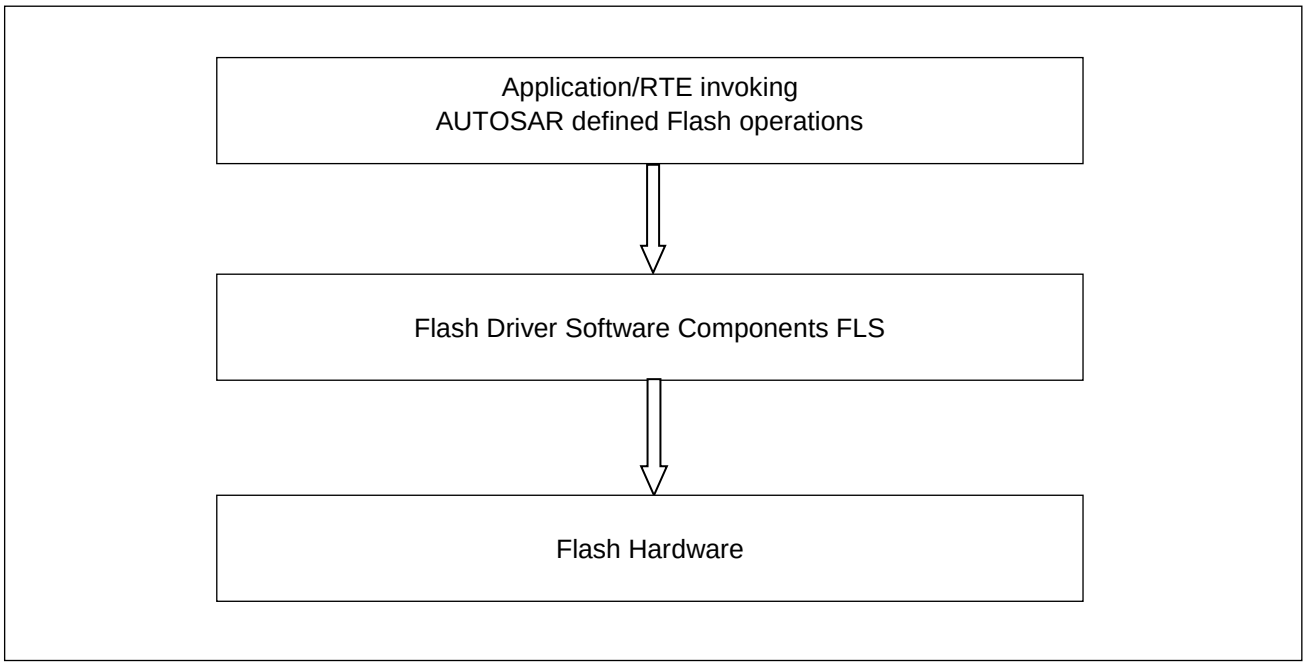


Figure 9.3 FLS Driver Component Architecture

The basic architecture of the FLS Driver Component is illustrated in the following figure:

FLS Driver Component:

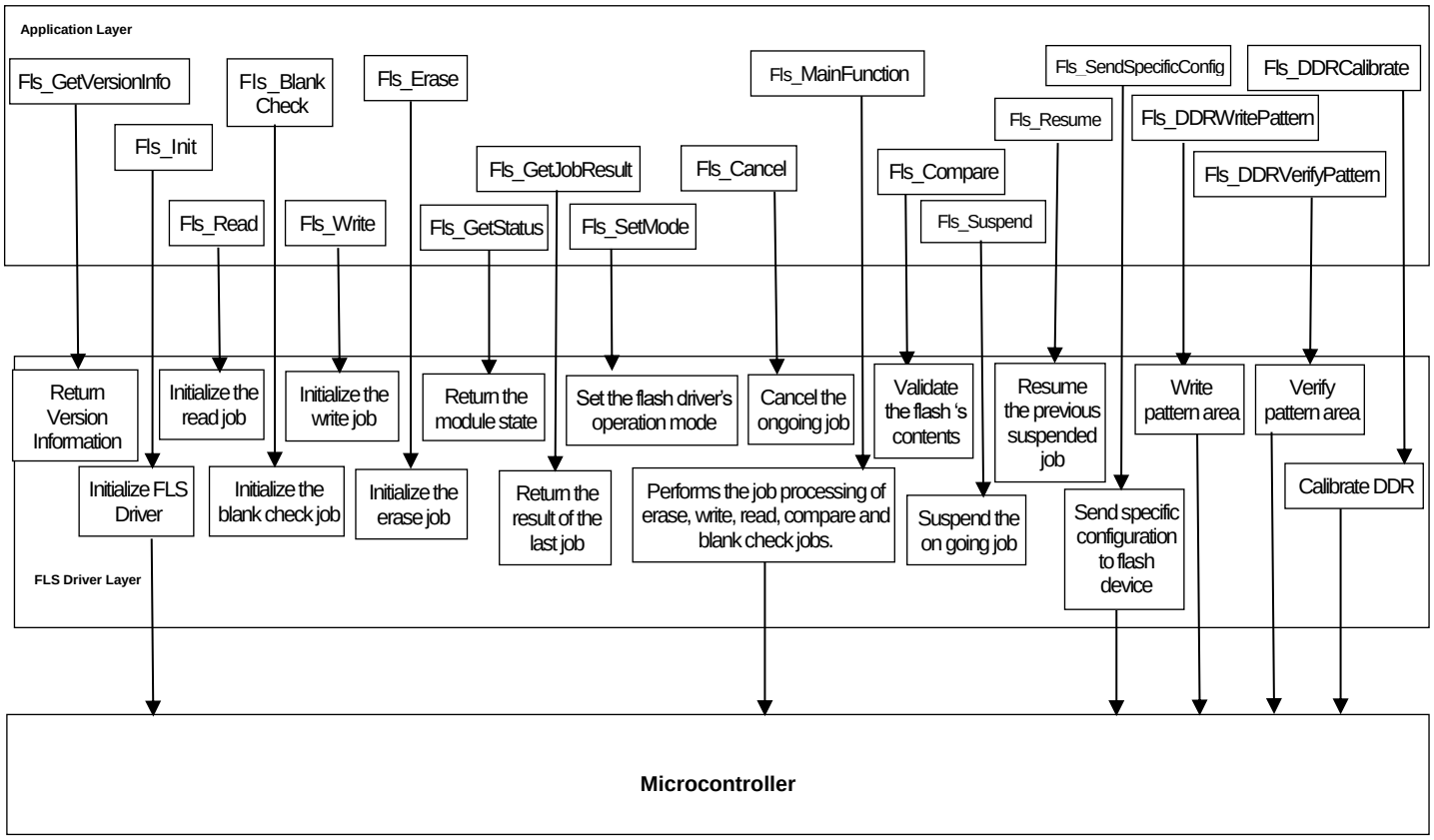


Figure 9.4 Component Overview of FLS Driver Component

The internal architecture of FLS Driver Component is shown in the figure above. The FLS Driver Component Software Component provides the following services.

The FLS Driver Component is divided into the following sub modules based on the functionality required:

- Initialization
- Erasing the flash memory
- Writing to the flash memory
- Reading the flash memory
- Reading result and status information
- Module version information
- Blank check of flash memory
- Validating contents of flash memory
- Job Processing
- Cancelling of a request
- Setting the FLS Component Driver's operation mode
- Suspend the on-going job
- Resume the previous suspended job
- Specific configuration for external flash device
- Fls DDR Verify
- Fls DDR write pattern
- Fls DDR Calibration

The initialization sub-module provides the service for initialization of the flash driver and initializes the global variables used by the FLS Component.

This sub-module related to the Fls_Init() API.

Erasing the flash memory
This sub-module provides the service for erasing the blocks of the flash memory. The request will be processed by the Fls_EraseJobHandler() job processing in the Fls_MainFunction() API. This job is called to erases the requested flash memory sectors. It is processed till the requested numbers of sectors are erased in the flash memory. Number of sectors depending on the parameter “Length” at the Fls_Erase() API, “Length” must be greater than zero and divisible by size of one sector.

This sub-module related to the Fls_Erase() API.

Writing to the flash memory
This sub-module provides the service for writing to the flash memory. The request will be processed by the Fls_WriteJobHandler() job processing in the Fls_MainFunction() API. This job called to writes the specified number of words from buffer to consecutive Flash addresses starting at the specified address. In single cycle of the Fls_MainFunction() API, the Fls_WriteJobHandler() job processing writes the data from target buffer to flash addresses depending on configuration of the parameter FlsMaxWriteNormalMode. The job is processed till the requested number of bytes is written to the flash memory.

This sub-module related to the Fls_Write() API.

Reading the flash memory
This sub-module provides the service for reading the contents of the flash memory. The request will be processed by the Fls_ReadJobHandler() job processing in the Fls_MainFunction() API. This job reads the specified number of words from consecutive Flash addresses starting at the specified address and writes it into a buffer. In single cycle of the Fls_MainFunction() API, the Fls_ReadJobHandler() job processing will read the data from the flash memory depending on configuration of the parameter FlsMaxReadNormalMode/ FlsMaxReadFastMode. The job is processed till the requested bytes of length are copied into the application buffer.

The configuration parameter FlsMaxReadNormalMode is used when the default mode is MEMIF_MODE_SLOW or the Fls_SetMode() API called to set mode to MEMIF_MODE_SLOW.
The configuration parameter FlsMaxFastNormalMode is used when the default mode is MEMIF_MODE_FAST or the Fls_SetMode() API called to set mode to MEMIF_MODE_FAST.

This sub-module related to the Fls_Read() API.

All configurations for read operation are available at the containers FlsSerialFlashRead and FlsSerialFlashDDRVerify. The FlsSerialFlashRead container is used for all read operations when the calibration is disabled (FlsDDRCalibration is ‘false’). Otherwise, when calibration is enabled, the FlsSerialFlashRead container is only used for the reading operation; the FlsSerialFlashDDRVerify container will be used for the operations verifying data, blank checking and comparing.

Reading result and status information
This sub-module provides the services for getting the current status of the module or results of the initiated job request or the response to previously issued command and return the current status of the current job execution.

This sub-module related to the Fls_GetStatus() and Fls_GetJobResult() APIs.

Cancelling of a request
This sub-module provides the service for cancelling an ongoing memory request. After aborting the current ongoing memory operations this sub- module prepares internal variables to accept the next Read/Write/Erase/Compare/Blank Check command. The cancel request will be synchronous, and a new job can be requested immediately after the return from this function.

This sub-module related to the Fls_Cancel() API.

Module version information
This module provides API for reading Module Id, Vendor Id, and vendor-specific version numbers.

This sub-module related to the Fls_GetVersionInfo() API.

Job Processing
The command requests are always processed by the main function (Fls_MainFunction API) that is invoked cyclically by the scheduler. This function will invoke the status check while processing the flash operations requests. This API derives the internal driver status. Completion of the flash operation needs to be checked in order to continue the reprogramming flow.

This sub-module related to the Fls_MainFunction() API.

Blank checking of flash memory
This sub-module provides the service for performing blank check of the flash memory words. The request will be processed by the Fls_BlankCheckJobHandler() job processing in the Fls_MainFunction() API. This function is invoked to perform the blank check of the single word. The job is processed till the requested numbers of words are performed with the blank check in the flash memory.

This sub-module related to the Fls_BlankCheck() API.

Setting the FLS Component Driver's operation mode
This sub-module provides the service to sets the flash driver's operation mode. User has two modes to switch:
- MEMIF_MODE_SLOW: Slow read access / normal SPI access.
- MEMIF_MODE_FAST: Fast read access / SPI burst access.

This sub-module related to the Fls_SetMode() API.

Validating the contents of Flash memory.
This sub-module provides the service for comparing the contents of the flash memory with the application buffer. The request will be processed by the Fls_CompareJobHandler() job processing in the Fls_MainFunction() API. This job reads defined words in flash and compares them with the data in application buffer. In single cycle of the Fls_MainFunction() API, the Fls_ CompareJobHandler() job processing will read the data from the flash memory depending on configuration of parameter FlsMaxReadNormalMode/ FlsMaxReadFastMode. The job is processed till the requested number of bytes are read and compared with the application buffer.

The configuration parameter FlsMaxReadNormalMode is used when the default mode is MEMIF_MODE_SLOW or the Fls_SetMode() API called to set mode to MEMIF_MODE_SLOW.

The configuration parameter FlsMaxFastNormalMode is used when the default mode is MEMIF_MODE_FAST or the Fls_SetMode() API called to set mode to MEMIF_MODE_FAST.

This sub-module related to the Fls_Compare() API.

Suspend the on-going job.
This sub-module provides the service of suspending the ongoing job.
This sub-module related to the Fls_Suspend() API.
The Fls_Suspend() is synchronous API.

Resume the previous suspended job.
This sub-module provides the service of resume of previous suspended job.
This sub-module related to the Fls_Resume() API.
The Fls_Resume() is synchronous API.

Specific configuration for external flash device.
This sub-module provides the service to setting specific configuration for external flash device.
This sub-module related to the Fls_SendSpecificConfig() API.

Fls DDR write pattern
This sub-module provides the service for DDR write pattern values.

This sub-module is related to the Fls_DDRWritePattern() API.

Fls DDR Verify
This sub-module provides the service to verify the DDR Pattern Area.
This sub-module is related to the Fls_DDRVerify() API.

Fls DDR Calibration
This sub-module provides the service to calibrate DDR.
This sub-module is related to the Fls_DDRCalibrate() API.

9.4 FLS Driver Component Header and Source File Description

This section explains the FLS Driver Component's C Source and C Header files. These files have to be included in the project application while integrating with other modules.
The C header file generated by FLS Software Generation Tool:

- Fls_Cfg.h

The C source file generated by FLS Driver Generation Tool:

- Fls_PBcfg.c

The FLS Driver Component C header files and Component source files:
Refer to "[1] R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.3.2.3 Folder Structure.

The Stub C header files:
Refer to "[1] R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.2.9 Stubs File.

9.5 Application Programming Interface

This section explains the Data types and APIs provided by the FLS Driver Component to the Upper layers.

9.5.1 Imported Types

This section explains the Data types imported by the FLS Driver Component and lists its dependency on other modules.

9.5.1.1 Standard Types

In this section all types included from the Std_Types.h are listed
Std_ReturnType
Std_VersionInfoType

9.5.1.2 Other Module Types

In this section, all types included in the MemIf_Types.h are listed:
• MemIf_ModeType
• MemIf_JobResultType
• MemIf_StatusType

,

9.5.2 Type Definitions

This section explains the type definitions of FLS Driver Component according to AUTOSAR Specification.

9.5.2.1 Fls_ConfigType

The Table 9.3 shows the FLS Config Type.

Table 9.3 Fls_ConfigType

Name:	Fls_ConfigType		
Type:	Structure		
Elements:	ulStartOfDbToc	Database start value	
		Range:	Version of configuration structure
	pJobEndNotificationPointer	Pointer to job end callback notification	
	pJobErrorNotificationPointer	Pointer to job error callback notification	
	enFlsDefaultMode	Enumeration to store default mode of flash.	
		Range:	MEMIF_MODE_SLOW: Slow read access / normal SPI access. MEMIF_MODE_FAST: Fast read access / SPI burst access.
	ulFlsMaxFastReadBytes	Variable to store the number of bytes will be read per a main cycle in fast mode.	
		Range:	0 to 4294967295

	ulFlsMaxSlowReadBytes	Variable to store the number of bytes will be read per a main cycle in slow mode.	
		Range:	0 to 4294967295
	ulFlsMaxWriteBytes	Variable to store the number of bytes will be written per a main cycle.	
		Range:	0 to 4294967295
	ulFlsProtection	Variable to store the Fls Protection value in bytes	
		Range:	0 to 4294967295
	ulNumberOfSectorPartition	The number of Sector Partition.	
		Range:	0 to 4294967295
	ulSectorPartitionIndex	Index of Sector Partition on the constant structure Fls_SectorMap	
		Range:	0 to 4294967295
	pSfDdrPatternConfig	Pointer to the constant structure Fls_SfDdrPatternConfig	
	pSfConfig	Pointer to the constant structure of serial flash device	
	pHfConfig	Pointer to the constant structure of hyper flash device	
Description:	A pointer to such a structure is provided to the flash driver initialization routine for configuration of the driver and flash memory hardware.		

9.5.2.2 **Fls_AddressType**

The Table 9.4 shows the FLS Address Type.

Table 9.4 Fls_AddressType

Name:	Fls_AddressType
Type:	uint32
Range:	0 to 4294967295
Description:	Address offset from the configured flash base address to access a certain flash memory area.

9.5.2.3 **Fls_LengthType**

The Table 9.5 shows the FLS Length Type.

Table 9.5 Fls_LengthType

Name:	Fls_LengthType

Type:	uint32
Range:	0 to 4294967295
Description:	Specifies the number of bytes to read/write/erase/blank check.

9.5.2.4 **Fls_SfSpecificConfigType**

The Table 9.6 shows the FLS SfSpecificConfig Type.

Table 9.6 Fls_SfSpecificConfigType

Name:	Fls_SfSpecificConfigType		
Type:	Structure		
Elements:	ulPHYCNTRegValue	Variable to store the configured value of PHYCNT register.	
		Range:	0 to 4294967295
	ulCMNCRRegValue	Variable to store the configured value of CMNCR register.	
		Range:	0 to 4294967295
	ulSMCMRRegValue	Variable to store the configured value of SMCR register.	
		Range:	0 to 4294967295
	ulSMADRRegValue	Variable to store the configured value of SMADR register.	
		Range:	0 to 4294967295
	ulSMENRRegValue	Variable to store the configured value of SMENR register.	
		Range:	0 to 4294967295
	ulSSLDRRegValue	Variable to store the configured value of SSLDR register.	
		Range:	0 to 4294967295
	ulSMDMCRValue	Variable to store the configured value of SMDMCR register.	
		Range:	0 to 4294967295
	ulSMDRENRValue	Variable to store the configured value of SMDRENR register.	
		Range:	0 to 4294967295

	ulSMOPRRegValue	Variable to store the configured value of SMOPR register.	
		Range:	0 to 4294967295
	ulSMCRRegValue	Variable to store the configured value of SMCR register.	
		Range:	0 to 4294967295
	ulPHYOFFSET1RegValue	Variable to store the configured value of PHYOFFSET1 register.	
		Range:	0x20000000
	ucRPCCKCRRegValue	Variable to store the configured value of RPCCKCR register.	
		Range:	17, 19, 21, 23, 25, 27, 29, 31
	ucDataLength	Variable to store the data length value.	
		Range:	0 to 255
Description:	Structure to store the specific user configuration for serial flash.		

9.5.2.5 **Fls_SpecificConfigType**

The Table 9.7 shows the FLS SpecificConfig Type.

Table 9.7 Fls_SpecificConfigType

Name:	Fls_SpecificConfigType
Elements:	Same with Fls_SfSpecificConfigType if value of FlsAccess parameter in FlsGeneral container is FLS_SERIAL_FLASH_DEVICE. Same with Fls_HfSpecificConfigType if value of FlsAccess parameter in FlsGeneral container is FLS_HYPER_FLASH_DEVICE.
Description:	Structure to store the specific user configuration.

9.5.2.6 **Fls_HfSpecificConfigType**

The Table 9.8 shows the FLS HfSpecificConfig Type.

Table 9.8 Fls_HfSpecificConfigType

Name:	Fls_HfSpecificConfigType	
Type:	Structure	
Elements:	pHfSpecificSetupCycle	Pointer to the constant setup cycle structure of hyper flash device.
	ulPHYCNTRegValue	Variable to store the configured value of PHYCNT register.

		Range:	0 to 4294967295
	ulCMNCRRegValue	Variable to store the configured value of CMNCR register.	
		Range:	0 to 4294967295
	ulSMADRRegValue	Variable to store the configured value of SMADR register.	
		Range:	0 to 4294967295
	ulSMENRRegValue	Variable to store the configured value of SMENR register.	
		Range:	0 to 4294967295
	ulSSLDRRegValue	Variable to store the configured value of SSLDR register.	
		Range:	0 to 4294967295
	ulSMDMCRValue	Variable to store the configured value of SMDMCR register.	
		Range:	0 to 4294967295
	ulSMDRENRValue	Variable to store the configured value of SMDRENR register.	
		Range:	0 to 4294967295
	ulSMCRRegValue	Variable to store the configured value of SMCR register.	
		Range:	0 to 4294967295
	ulPHYOFFSET1RegValue	Variable to store the configured value of PHYOFFSET1 register.	
		Range:	0x20000000
	ucRPCCKCRRegValue	Variable to store the configured value of RPCCKCR register.	
		Range:	17, 19, 21, 23, 25, 27, 29, 31
	ucHfSpecificNumberCycle	The number of setup cycle.	
		Range:	0 to 255
	ucDataLength	Variable to store the data length value.	
		Range:	0 to 255
Description:	Structure to store the specific user configuration for hyper flash.		

9.5.3 Function Definitions

This section explains the function definitions of FLS Driver Component.

Table 9.9 Function Definitions

Sl. No	API
1.	Fls_Init
2.	Fls_Erase
3.	Fls_Write
4.	Fls_Read
5.	Fls_GetVersionInfo
6.	Fls_BlankCheck
7.	Fls_MainFunction
8.	FlsJobEndNotification
9.	FlsJobErrorNotification
10.	Fls_GetStatus
11.	Fls_GetJobResult
12.	Fls_SetMode
13.	Fls_Cancel
14	Fls_Compare
15	Fls_Suspend
16	Fls_Resume
17	Fls_SendSpecificConfig
18	Fls_DDRWritePattern
19	Fls_DDRVerifyPattern
20	Fls_DDRCalibrate

9.5.3.1 Renesas Original API

Table 9.10 Fls_Suspend

Function Name:	Fls_Suspend	
Syntax:	FUNC(void, FLS_CODE_SLOW) Fls_Suspend(void)	
Service Id:	0xFE	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant	
Parameters (In):	None	None
Parameters (Out):	None	None
Parameters (in/out):	None	None
Return value:	None	None
Description:	This API performs the suspend of the ongoing job	

Preconditions:	FLS Driver's components should be initialized.	
Remarks:	This API is pre-compile time configurable (STD_ON/ STD_OFF) by the configuration parameter FlsSuspendApi Pre-compile options: FLS_SUSPEND_API	
DET/DEM Errors handled:	FLS_E_UNINIT	FLS Driver's components are not initialized.
	FLS_E_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.
	FLS_E_RELEASE_SEMAPHORE_FAILURE	Cannot release semaphore resource.

Table 9.11 Fls_Resume

Function Name:	Fls_Resume	
Syntax:	FUNC(void, FLS_CODE_SLOW) Fls_Resume(void)	
Service Id:	0xFD	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant	
Parameters (In):	None	None
Parameters (Out):	None	None
Parameters (in/out):	None	None
Return value:	None	None
Description:	This API performs the resume of the previous suspending job.	
Preconditions:	FLS Driver's components should be initialized.	
Remarks:	This function is pre-compile time configurable (STD_ON/ STD_OFF) by the configuration parameter FlsResumeApi Pre-compile options: FLS_RESUME_API	
DET/DEM Errors handled:	FLS_E_UNINIT	FLS Driver's components are not initialized.
	FLS_E_GET_SEMAPHORE_FAILURE	Cannot get semaphore resource.
	FLS_E_GET_CONTROL_FAILURE	Cannot get exclusive control.

Table 9.12 Fls_DDRWritePattern

Function Name:	Fls_DDRWritePattern
Syntax:	FUNC(Std_ReturnType, FLS_CODE_SLOW) Fls_DDRWritePattern(void)
Service Id:	0xFC
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant

Parameters (In):	None	None
Parameters (Out):	None	None
Parameters (in/out):	None	None
Return value:	E_OK	DDR pattern is written successfully
	E_NOT_OK	DDR write pattern is failed
Description:	This API performs write pattern values to DDR pattern.	
Preconditions:	FLS Driver's components should be initialized.	
Remarks:	This function is pre-compile time configurable (STD_ON/ STD_OFF) by the configuration parameter FlsDDRCalibration Pre-compile options: FLS_DDR_CALIB_FEATURE	
DET/DEM Errors handled:	FLS_E_UNINIT	FLS Driver's components are not initialized.
	FLS_E_BUSY	FLS Driver's state is MEMIF_BUSY
	FLS_E_GET_CONTROL_FAILURE	Cannot get exclusive control.
	FLS_E_GET_SEMAPHORE_FAILURE	Cannot get semaphore resource.
	FLS_E_CPG_GET_CONTROL_FAILURE	Cannot get exclusive control.
	FLS_E_TIMEOUT	Timeout exceeded.
	FLS_E_RELEASE_SEMAPHORE_FAILURE	Cannot release semaphore resource.
	FLS_E_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.
	FLS_E_VERIFY_WRITE_FAILED	A flash write verification (compare) error.
	FLS_E_PARAM_DATA	The <CounterID> was not valid
	FLS_E_ERASE_FAILED	A flash erase job fails due to a hardware error
	FLS_E_VERIFY_ERASE_FAILED	A flash erase verification (blankcheck) error
	FLS_E_WRITE_FAILED	A flash write job fails due to a hardware error
	FLS_E_CLOCK_SET_FAILURE	Cannot set frequency value to RPCCKCR register.
	FLS_E_CPG_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.

Table 9.13 Fls_DDRVerifyPattern

Function Name:	Fls_DDRVerifyPattern
Syntax:	FUNC(Std_ReturnType, FLS_CODE_SLOW) Fls_DDRVerifyPattern(void)
Service Id:	0xFB
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant

Parameters (In):	None	None
Parameters (Out):	None	None
Parameters (in/out):	None	None
Return value:	E_OK	DDR pattern value is consistent.
	E_NOT_OK	DDR pattern value is inconsistent or verification is failed
Description:	This API verify DDR pattern value.	
Preconditions:	FLS Driver's components should be initialized.	
Remarks:	This function is pre-compile time configurable (STD_ON/ STD_OFF) by the configuration parameter FlsDDRCalibration Pre-compile options: FLS_DDR_CALIB_FEATURE	
DET/DEM Errors handled:	FLS_E_UNINIT	FLS Driver's components are not initialized.
	FLS_E_BUSY	FLS Driver's state is MEMIF_BUSY
	FLS_E_GET_CONTROL_FAILURE	Cannot get exclusive control.
	FLS_E_GET_SEMAPHORE_FAILURE	Cannot get semaphore resource.
	FLS_E_CPG_GET_CONTROL_FAILURE	Cannot get exclusive control.
	FLS_E_TIMEOUT	Timeout exceeded.
	FLS_E_CLOCK_SET_FAILURE	Cannot set frequency value to RPCCKCR register.
	FLS_E_CPG_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.
	FLS_E_RELEASE_SEMAPHORE_FAILURE	Cannot release semaphore resource.
	FLS_E_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.

Table 9.14 Fls_SendSpecificConfig

Function Name:	Fls_SendSpecificConfig
Syntax:	FUNC(Std_ReturnType, FLS_CODE_SLOW) Fls_SendSpecificConfig (P2CONST(Fls_SpecificConfigType, AUTOMATIC, FLS_APPL_CONST) SpecificConfigPtr, P2VAR(uint8, AUTOMATIC, FLS_APPL_CONST) DataAddressPtr, const uint8 Length)
Service Id:	0xFA
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant

Parameters (In):	SpecificConfigPtr	Pointer to user's specific configuration set - When FlsDevErrorDetect parameter is True, API will check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user - This parameter is mandatory to be provided correctly(it cannot be omitted)	
		Range:	Valid pointer
	Length	Length of buffer data. - When FlsDevErrorDetect parameter is True, API will check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user - This parameter is mandatory to be provided correctly(it cannot be omitted)	
		Range:	0.. (Data length) Data length is a value which configured at the parameter FlsDataLength in the container FlsSerialFlashSpecificConfig/ FlsHyperFlashSpecificConfig.
Parameters (Out):	None	None	
Parameters (in/out):	DataAddressPtr	Pointer to the buffer data of user. When the parameter FlsOperation in the container FlsSerialFlashSpecificConfig/ FlsHyperFlashSpecificConfig configured as FLS_READ_OPERATION and the value of "Length" is not equal 0, this pointer is an output parameter. It stores the read data. When the parameter FlsOperation in the container FlsSerialFlashSpecificConfig/ FlsHyperFlashSpecificConfig configured as FLS_WRITE_OPERATION and the value of "Length" is not equal 0, this pointer is an input parameter. It stores the write data will be sent to flash device. When the value of "Length" equal 0, this pointer is ignored. - When FlsDevErrorDetect parameter is True, API will check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user	

		- This parameter is mandatory to be provided correctly(it cannot be omitted)
		Range: Valid pointer
Return value:	E_OK	Sets specific configuration has been finished without errors.
	E_NOT_OK	Sets specific configuration has been finished with errors.
Description:	This API sets the specific configuration to flash device.	
Preconditions:	FLS Driver's components should be initialized.	
Remarks:	This function is pre-compile time configurable (STD_ON/ STD_OFF) by the configuration parameter FlsSendSpecificConfigApi Pre-compile options: FLS_SEND_SPEC_CONFIG_API	
DET/DEM Errors handled:	FLS_E_UNINIT	Check if API service called without module initialization.
	FLS_E_PARAM_LENGTH	Fls_SendSpecificConfig shall raise this error when the value of input parameter "Length" larger than the value configured at the parameter FlsDataLength in the container FlsSerialFlashSpecificConfig/ FlsHyperFlashSpecificConfig.
	FLS_E_PARAM_DATA	DataAddressPtr is a null pointer.
	FLS_E_PARAM_CONFIG	SpecificConfigPtr is a null pointer.
	FLS_E_TIMEOUT	Timeout exceeded.
	FLS_E_BUSY	FLS Driver's state is MEMIF_BUSY or MEMIF_BUSY_INTERNAL
	FLS_E_RELEASE_SEMAPHORE_FAILURE	Cannot release semaphore resource.
	FLS_E_CLOCK_SET_FAILURE	Cannot set frequency value to RPCCKCR register.
	FLS_E_CPG_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.
	FLS_E_CPG_GET_CONTROL_FAILURE	Cannot get exclusive control.
	FLS_E_WRITE_VERIFY_FAILURE	Detects the difference between the RPC's register values and the compare data.
	FLS_E_GET_CONTROL_FAILURE	Cannot get exclusive control.
	FLS_E_GET_SEMAPHORE_FAILURE	Cannot get semaphore resource.

Table 9.15 Fls_DDRCalibrate

Function Name:	Fls_DDRCalibrate
Syntax:	FUNC(Std_ReturnType, FLS_CODE_SLOW) Fls_DDRCalibrate(void)
Service Id:	0xF9
Sync/Async:	Synchronous

Reentrancy:	Non-Reentrant	
Parameters (In):	None	None
Parameters (Out):	None	None
Parameters (in/out):	None	None
Return value:	E_OK	DDR calibration is completed successfully
	E_NOT_OK	DDR calibration is not completed successfully
Description:	This API performs DDR calibration.	
Preconditions:	FLS Driver's components should be initialized.	
Remarks:	This function is pre-compile time configurable (STD_ON/ STD_OFF) by the configuration parameter FlsDDRCalibration Pre-compile options: FLS_DDR_CALIB_FEATURE	
DET/DEM Errors handled:	FLS_E_UNINIT	Check if API service called without module initialization.
	FLS_E_BUSY	FLS Driver's state is MEMIF_BUSY
	FLS_E_GET_CONTROL_FAILURE	Cannot get exclusive control.
	FLS_E_GET_SEMAPHORE_FAILURE	Cannot get semaphore resource.
	FLS_E_RELEASE_SEMAPHORE_FAILURE	Cannot release semaphore resource.
	FLS_E_CPG_GET_CONTROL_FAILURE	Cannot get exclusive control for CPG.
	FLS_E_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.
	FLS_E_CPG_WRITE_VERIFY_FAILURE	Detects the difference between the register value and the compare data.
	FLS_E_CLOCK_SET_FAILURE	Cannot set frequency value to RPCCKCR register.
	FLS_E_TIMEOUT	Timeout exceeded.

9.5.3.2 **Renesas Original callback function**

None

9.5.4 Preemption of APIs

The Table 9.16 specifies the preemption of each API that can be invoked at the same time as the API.

Table 9.16 Preemption Table of APIs of the FLS Driver

	Fls_Init	Fls_Erase	Fls_Write	Fls_MainFunction	Fls_Read	Fls_GetVersionInfo	Fls_BlankCheck	Fls_GetStatus	Fls_GetJobResult	Fls_SetMode	Fls_Cancel	Fls_Compare	Fls_Suspend	Fls_Resume	Fls_SendSpecificConfig	Fls_DDRWritePattern	Fls_DDRVerifyPattern	Fls_DDRCalibrate
Fls_Init	-																	
Fls_Erase	-	-																
Fls_Write	-	-	-															
Fls_MainFunction	-	-	-	-														
Fls_Read	-	-	-	-	-													
Fls_GetVersionInfo	√	√	√	√	√	√												
Fls_BlankCheck	-	-	-	-	-	√	-											
Fls_GetStatus	-	√	√	√	√	√	√	-										
Fls_GetJobResult	-	√	√	√	√	√	√	√	-									
Fls_SetMode	-	-	-	-	-	√	-	√	√	-								
Fls_Cancel	-	-	-	-	-	√	-	√	√	-	-							
Fls_Compare	-	-	-	-	-	√	-	√	√	-	-	-						
Fls_Suspend	-	-	-	-	-	√	-	√	√	-	-	-	-					
Fls_Resume	-	-	-	-	-	√	-	√	√	-	-	-	-	-				
Fls_SendSpecificConfig	-	-	-	-	-	√	-	√	√	√	-	-	-	-	-			
Fls_DDRWritePattern	-	-	-	-	-	√	-	√	√	√	-	-	-	-	-	-		
Fls_DDRVerifyPattern	-	-	-	-	-	√	-	√	√	√	-	-	-	-	-	-	-	
Fls_DDRCalibrate	-	-	-	-	-	√	-	√	√	√	-	-	-	-	-	-	-	-

-: cannot be invoked at the same time
√: can be invoked at the same time

9.6 Development and Production Errors

In this section, the development errors reported by the FLS Driver Component are tabulated.
The development errors other than the above will be reported only when the pre-compiler option 'FlsDevErrorDetect' is enabled in the configuration. The production code errors are not supported by the FLS Driver Component.

9.6.1 FLS Driver Component Development Errors

The following table contains the DET errors reported by the FLS Driver Component. These errors are reported to the Development Error Tracer Module when the FLS Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 9.17 DET Errors of FLS Driver Component

Sl. No.	**1**
ErrorCode	FLS_E_UNINIT
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_BlankCheck, Fls_GetStatus, Fls_GetJobResult, Fls_SetMode, Fls_Cancel, Fls_Compare, Fls_Suspend, Fls_Resume, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When the API service is invoked before initialization.
Origin	AUTOSAR
Sl. No.	**2**
Error Code	FLS_E_PARAM_ADDRESS
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_BlankCheck, Fls_Compare
Source of Error	When the API service is invoked with a wrong address.
Origin	AUTOSAR
Sl. No.	**3**
Error Code	FLS_E_PARAM_LENGTH
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_BlankCheck, Fls_Compare, Fls_SendSpecificConfig
Source of Error	When the API service is invoked with a wrong length.
Origin	AUTOSAR
Sl. No.	**4**
ErrorCode	FLS_E_PARAM_DATA
RelatedAPI(s)	Fls_Write, Fls_Read, Fls_Compare, Fls_SendSpecificConfig, Fls_DDRWritePattern
Source of Error	When the API service is invoked with a NULL buffer address.
Origin	AUTOSAR
Sl. No.	**5**
ErrorCode	FLS_E_PARAM_CONFIG
RelatedAPI(s)	Fls_Init, Fls_SendSpecificConfig
Source of Error	API initialization service invoked with wrong parameter.
Origin	AUTOSAR
Sl. No.	**6**
Error Code	FLS_E_BUSY

Origin	AUTOSAR
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_BlankCheck, Fls_SetMode, Fls_Init, Fls_Compare, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When the API service is invoked when the driver is still busy.
Sl. No.	**10**
Error Code	FLS_E_PARAM_POINTER
RelatedAPI(s)	Fls_GetVersionInfo
Source of Error	API service API Fls_GetVersionInfo reports the error FLS_E_PARAM_POINTER to DET, when called with null pointer.
Origin	AUTOSAR
Sl. No.	**11**
Error Code	FLS_E_INVALID_DATABASE
RelatedAPI(s)	Fls_Init
Source of Error	API service API Fls_Init reports the error FLS_E_INVALID_DATABASE to DET, when called the invalid database.
Origin	AUTOSAR

9.6.2 FLS Driver Component Production Errors

In this section the DEM errors identified in the FLS Driver Component are listed. FLS Driver Component reports these errors to DEM by invoking Dem_SetEventStatus API. All errors are come from Renesas.

Table 9.18 DEM Error of FLS Driver Component

Sl. No.	**1**
Error Code	FLS_E_GET_CONTROL_FAILURE
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_BlankCheck, Fls_Init, Fls_Compare, Fls_Resume, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When cannot get exclusive resource control.
Origin	Renesas
Sl. No.	**2**
Error Code	FLS_E_GET_SEMAPHORE_FAILURE
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_BlankCheck, Fls_Init, Fls_Compare, Fls_Resume, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When cannot get semaphore resource control.
Origin	Renesas
Sl. No.	**3**

Error Code	FLS_E_RELEASE_SEMAPHORE_FAILURE
Related API(s)	Fls_MainFunction, Fls_Cancel, Fls_Init, Fls_Suspend, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When cannot release semaphore resource control.
Origin	Renesas
Sl. No.	**4**
Error Code	FLS_E_CPG_GET_CONTROL_FAILURE
Related API(s)	Fls_MainFunction, Fls_Init, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When cannot get resource control.
Origin	Renesas
Sl. No.	**5**
Error Code	FLS_E_CLOCK_SET_FAILURE
Related API(s)	Fls_MainFunction, Fls_Cancel, Fls_Init, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When the value comparison timed out.
Origin	Renesas
Sl. No.	**6**
Error Code	FLS_E_CPG_WRITE_VERIFY_FAILURE
Related API(s)	Fls_MainFunction, Fls_Cancel, Fls_Init, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When a register of RPC read back failure was detected
Origin	Renesas
Sl. No.	**7**
Error Code	FLS_E_WRITE_VERIFY_FAILURE
Related API(s)	Fls_MainFunction, Fls_Cancel, Fls_Init, Fls_Suspend, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	When there is difference between the register value and the compared data.
Origin	Renesas

9.7 FLS Driver Component Runtime Errors

The following table contains the DET runtime errors that are reported by FLS driver component. These are the hardware errors reported during runtime.

Table 9.19 DET Runtime Errors of FLS Driver Component

Sl. No.	1
Error Code	FLS_E_VERIFY_ERASE_FAILED
RelatedAPI(s)	Fls_MainFunction, Fls_DDRWritePattern
Source of Error	When the erase verification fails.
Origin	AUTOSAR
Sl. No.	2
Error Code	FLS_E_VERIFY_WRITE_FAILED
RelatedAPI(s)	Fls_MainFunction, Fls_DDRWritePattern
Source of Error	When the write verification fails.
Origin	AUTOSAR
Sl. No.	3
Error Code	FLS_E_TIMEOUT
RelatedAPI(s)	Fls_MainFunction, Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate
Source of Error	API service invoked when time out supervision of a read, write, erase or compare job failed.
Origin	AUTOSAR

9.8 FLS Driver Component Transient Faults

The following table contains the DET transient faults that are reported by FLS driver component.

Table 9.20 DET Transient Faults of FLS Driver Component

Sl. No.	**1**
Error Code	FLS_E_ERASE_FAILED
Value	0x01
Related API(s)	Fls_MainFunction, Fls_DDRWritePattern
Source of Error	The flash erase function failed.
Sl. No.	**2**
Error Code	FLS_E_WRITE_FAILED
Value	0x02
Related API(s)	Fls_MainFunction, Fls_DDRWritePattern
Source of Error	The flash write function failed.
Sl. No.	**5**
Error Code	FLS_E_UNEXPECTED_FLASH_ID
Value	0x05
Related API(s)	Fls_Init
Source of Error	During the initialization of the external flash driver, the FLS module shall check the hardware ID of the external flash device against the corresponding published parameter. If a hardware ID mismatch occurs, the FLS module shall report the error code FLS_E_UNEXPECTED_FLASH_ID to the Default Error Tracer (DET).

9.9 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of FLS Driver Component software.

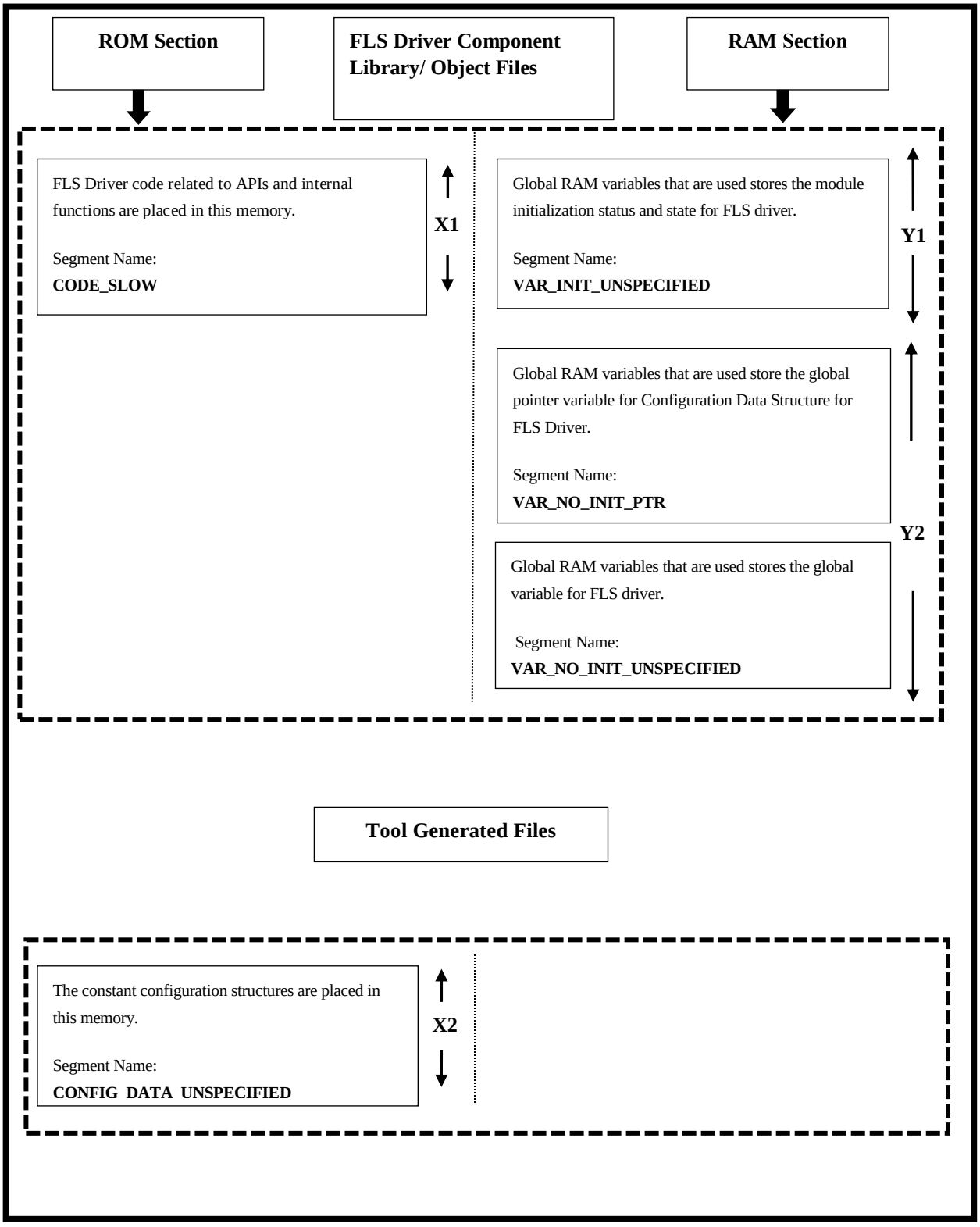


Figure 9.5 FLS Driver Component Overview Memory Organization

ROM Section (X1, X2)

CODE_SLOW(X1): API(s) and Internal functions of FLS Driver Component, which can be located in code memory.

CONFIG_DATA_UNSPECIFIED (X2): This section consists of FLS Driver Component constant configuration structures. This can be located in code memory.

Note: ROM in R-Car is the Read-only Section in memory (DRAM)

RAM Section (Y1, Y2):

VAR_INIT_UNSPECIFIED (Y1): This section consists of the global RAM variables that are used store the module initialization status and the state for FLS driver. This can be located in data memory.

VAR_NO_INIT_UNSPECIFIED and VAR_NO_INIT_PTR (Y2): This section consists of the global variable and the global pointer variable for FLS driver. This can be located in data memory.

Note: User must ensure that none of the memory areas overlap with each other. Even ‘debug’ information should not overlap.

In external address space read mode (Configuration parameter FlsExternalSpaceReadMode set to true) FLS driver using memories from 0x08000000 to 0x0BFFFFFF. This memory area should be configured as non-cacheable.

9.10 Device-Specific Information

9.10.1 Interaction between the User and FLS Driver Component

The details of the services supported by the FLS Driver Component to the upper layer users and the mapping of the channels to the hardware units is provided in the following sections:

9.10.1.1 **Channel Mapping**

None

9.10.1.2 **ISR Functions**

None

9.10.2 Multi-Core / Multi-Instantiation

FLS driver does not support multi-core and multi-instantiation for this device.

9.11 Non-AUTOSAR environment integration

9.11.1 Stub modules handling

9.11.1.1 Os

The Os stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar_version>\Os

In the AUTOSAR environment, FLS Driver is relying on APIs of GetCounterValue() and GetElapsedValue provided by the OS module to get the current count value of the specified counter.

StatusType GetCounterValue(CounterType CounterID, TickRefType Value);

StatusType GetCounterValue(CounterType CounterID, TickRefType Value, TickRefType ElapsedValue);

Current Os stub implementation uses SCMT (System Timer) IP to simulate an Os counter. This counter has OsSecondsPerTick ~ 0.0000076s and will always run with sample application. GetCounterValue will return the value of that counter at calling time, GetElaspedValue will return the value between current value of that counter and previously read value.

Non-AUTOSAR users can modify the provided GetCounterValue and GetElapsedValue APIs as needed e.g. with other hardware counter or unique CounterID for each MCAL module.

9.11.1.2 Det

The Det stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar_version>\Det

In the AUTOSAR environment, FLS Driver Components uses Det_ReportError, Det_ReportRuntimeError and Det_ReportTransientFault API provided by the DET module to report a development error or runtime error, e.g., FLS Driver has not been initialized, API is provided with invalid parameter. The API prototype is as of follow:

Std_ReturnType Det_ReportError(uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Std_ReturnType Det_ReportRuntimeError(uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Std_ReturnType Det_ReportTransientFault(uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 FaultId）

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError, Det_ReportRuntimeError and Det_ReportTransientFault API with their current error handling strategy.

9.11.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.

The SchM (Basic Software Scheduler) stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar_version>\Rte

MSN driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when FlsCriticalSectionProtection parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user MSN driver as following:

void SchM_Enter_Fls_FLS_RAM_DATA_PROTECTION (void)

void SchM_Exit_Fls_FLS_RAM_DATA_PROTECTION (void)

9.11.1.4 **Dem**

The Dem stub files are organized in the following folder:

rel\common\generic\stubs\<Autosar_version>\Dem

In the AUTOSAR environment, FLS Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error e.g., FLS Driver access failure… The API prototype is as of follow:

Dem_SetEventStatus(Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

9.11.1.5 **MemIf**

The MemIf (Memory Abstraction Interface) stub files are organized in the following folder:

rel\common\generic\stubs\<Autosar_version>\MemIf

FLS driver uses MemIf_JobResultType of MemIf as return type for Fls_GetJobResult API.

FLS driver uses MemIf_StatusType of MemIf as return type for Fls_GetStatus API.

FLS driver uses type MemIf_ModeType of MemIf for Parameter “Mode” of Fls_SetMode API

9.11.1.6 **Fee**

The Fee (Flash EEPROM Emulation) stub files are organized in the following folder:

rel\common\generic\stubs\<Autosar_version>\Fee

FLS driver uses Fee_JobEndNotification() and Fee_JobErrorNotification() of Fee.

FLS driver call Fee_JobEndNotification() of Fee when a job has been completed with a positive result.

FLS driver call Fee_JobErrorNotification() of Fee when a job has been canceled or finished with negative result.

9.11.2 Callback function usage

The FLS Driver Component does not provide any callback functions.

9.11.3 Scheduled function usage

Period of Fls_MainFunction should be greater than its throughput. User should consider specific period based on the performance and respond time of main function. If the period is too small, the performance will be decreased. If the period is too large, Fls_MainFunction will need a long time to respond.

9.11.4 Interrupt handling usage

None

10.GPT

10.1 Overview

The purpose of this chapter is to describe the information related to GPT Driver Component.
The users of GPT Driver Component shall use this chapter as reference.
This chapter describes the common features of GPT Driver Component. This chapter is intended for the developers of ECU software using Application Programming Interfaces provided by AUTOSAR. The GPT Driver Component provides the following services:

- Initialization
- De-Initialization
- Start Timer
- Stop Timer
- Reading timer values (time remaining and time elapsed)
- Notification Services
- Reading Predef timer value
- Version Information
- Wakeup trigger Services (This service depends on hardware)

The **Figure 10.1** shows the system overview of the AUTOSAR Architecture. The GPT Driver initializes all the channels required to produce the GPT outputs.

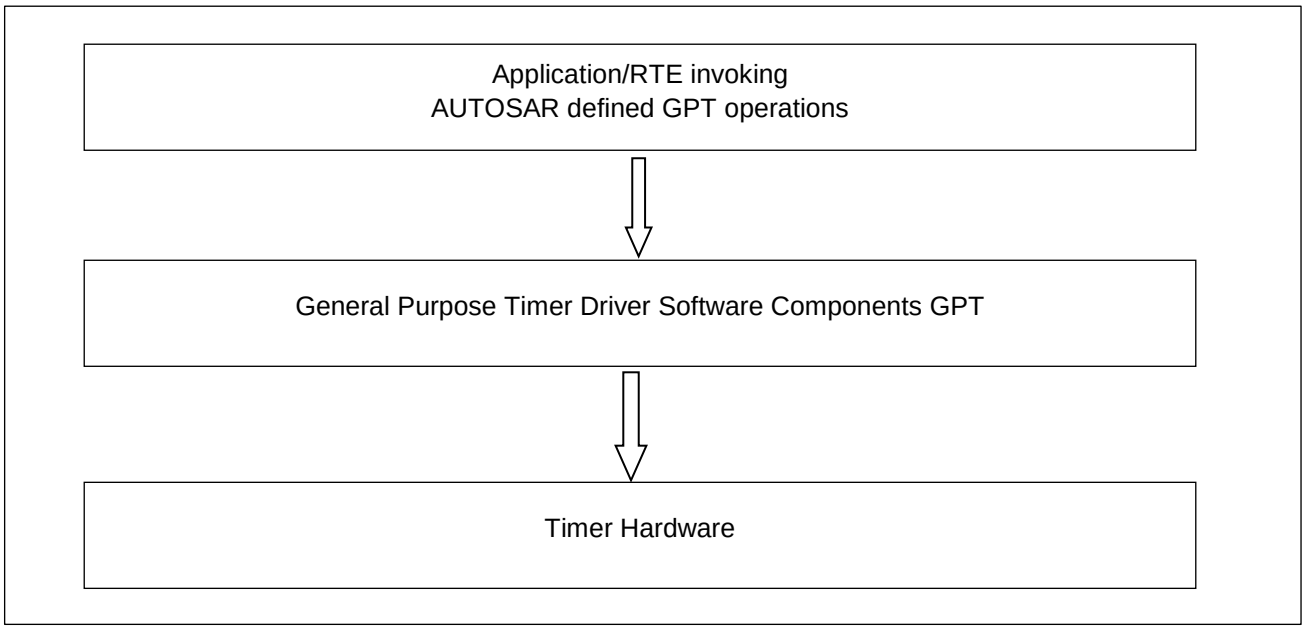


Figure 10.1 System Overview Of AUTOSAR Architecture

The GPT Driver Component comprises two sections, that is, Embedded Software and the Configuration Tool to achieve scalability and configurability.

The GPT Driver Component Code Generation Tool is a command line tool that accepts ECU configuration description files as input and generates C Source and C Header files. The configuration description is an ARXML file that contains information about the configuration for GPT channels. The tool generates Gpt_Cfg.h, Gpt_Cbk.h, and Gpt_PBcfg.c files.

The **Figure 10.2** depicts the GPT Driver as part of layered AUTOSAR MCAL Layer:

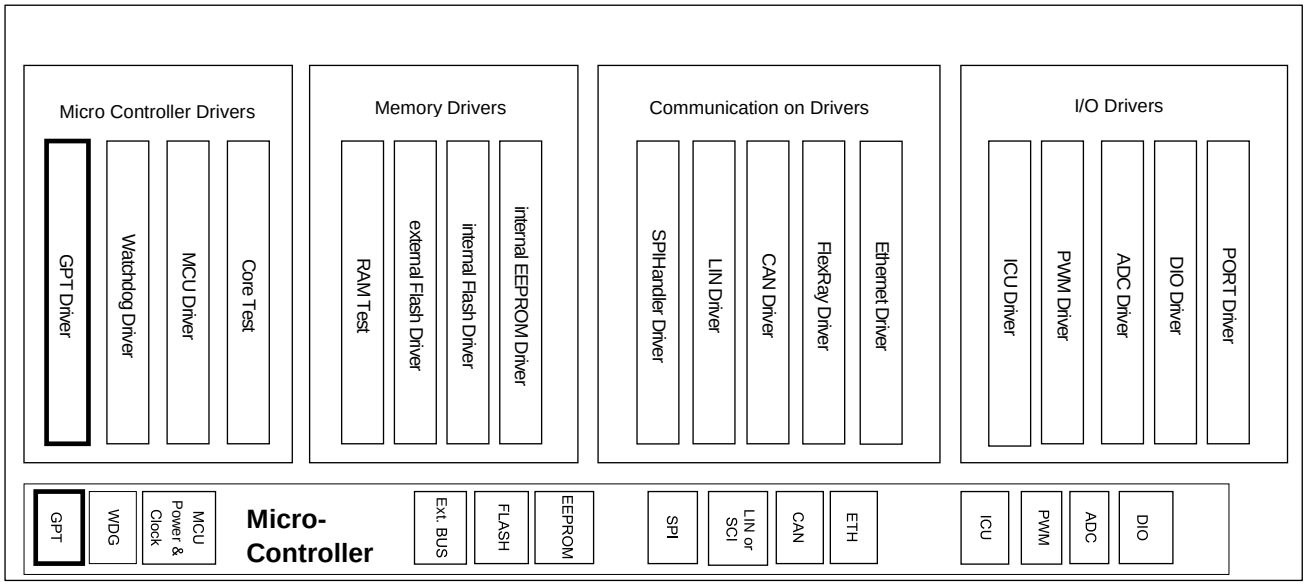


Figure 10.2 System Overview Of The GPT Driver In AUTOSAR MCAL Layer

10.2 Forethoughts

10.2.1 General

The following information will help the user use the GPT Driver Component software efficiently:

- The start-up code is ECU-specific. GPT Driver does not implement the start- up code.
- The GPT Driver only generates time bases and does not serve as an event counter.
- The GPT Driver does not support timeout period that exceed the maximum value restricted by the clock source, prescaler and width of the timer register. The user should ensure that the values are configured properly.
- The GPT Driver Component depends on the system clock, prescaler(s), and PLL. Thus, any change in the system clock (For example, PLL on -> PLL off) affects the clock settings of the GPT hardware. GPT Driver Component does not take care of setting the registers configuring the clock, prescaler(s), and PLL in its initialization function.
- The GPT driver used clock divider SASYNC and SASYNCD2 as base clock (SASYNCRT and SASYNCPERD2 in HW)
- User needs to configure at least one prescaler unit in Timer modules (GPT or ICU or PWM) in order to enable the timer functionality.
- The GPT Driver does not implement the scheduled functions.
- Example code mentioned in this section shall be taken only as a reference for implementation.
- GPT Module and ICU Module share HW resource. The user should ensure that one subblock should be used by one module.
- The GPT Driver Generation tool generates the Channel Id handles of the Channels configured for GPT functionality.
- The timers required for the GPT Driver Component, PWM Driver Component and ICU Driver Component should be selected during the configuration. These timer selections are pre-compile-configured. The assignment of GPT channels to timer resources is static.
- The number of channels configured should be maintained across multiple configuration sets and for single post-build configuration set. If additional channel is added, or if any channel is removed, the GPT Driver Component should be compiled and linked again. User should ensure that the PWM Driver and ICU Driver Components are also recompiled, in case of any modification in the channels configured for these components.
- The GPT Driver Component Generation Tool generates the addresses in ROM (FLASH) for all structures related to the channel based on the configuration in ECU Configuration description files(s).
- The GPT Driver Component Generation Tool generates the extern declarations of callback notification function prototypes configured through Gpt_Notification (per channel configurable field) parameter in Gpt_Cbk.h. The application shall include Gpt_Cbk.h file to obtain the extern declarations to implement these callback functions.
- The timer to channel mapping is fixed as per the Table Channel to Timer mapping. The Channel Id handles generated by the GPT Driver Generation tool can be used in the API calls to operate on the particular channel.
- The ISR functions and the corresponding handler addresses are provided in Table ISR Handler Addresses of Timers. User should ensure that Interrupt Vector table is configured according to the information provided in the table.
- Timer channel configured for one-shot mode with no callback function configured will stop implicitly after initial timeout without invoking a stop conversion. However, in the timer channel configured for Continuous

mode, as the timers keep operating even after the target value is reached and it has multiple notifications (if enabled), user needs to invoke “Gpt_StopTimer” explicitly to stop timer in Continuous mode.

- The supply clock of Timer IP from MCU module must set maximum clock frequency when using the GPT Predef Timer function.
- One channel cannot enable wakeup and notification at the same time as the timing of notification cannot be guaranteed.
- Start timer for channels should be called in normal mode regardless of the wakeup channels or notification channels.

10.2.2 Preconditions

The following preconditions have to be adhered by the user, for proper functioning of the GPT Driver Component:

- Before Gpt_Init function call, all registers are expected to be the value after reset.
- The user should ensure that GPT Component API requests are invoked in the correct and expected sequence along with correct input arguments.
- Validation of input parameters is performed only when the static configuration parameter 'GPT_DEV_ERROR_DETECT' is enabled. Application should ensure that the right parameters are passed while invoking the APIs when GPT_DEV_ERROR_DETECT is disabled.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used. The version information is described in the 1.1 Supported MCAL Product Release Version.
- All timer units used within the API services of the GPT Driver shall be in ticks. (expect Predef Timer channel)
- The files Gpt_Cfg.h, Gpt_Cbk.h, and Gpt_PBcfg.c generated using the GPT Driver Generation Tool have to be linked with the GPT Driver Component source files.
- The application has to be rebuilt, if there is any change in the Gpt_Cfg.h, Gpt_Cbk.h files generated by the GPT Driver Generation Tool.
- The GPT Driver Component needs to be initialized before accepting any API requests. Gpt_Init should be called to initialize the GPT Driver Component.

10.2.3 Data Consistency

To support the re-entrance and interrupt services, the AUTOSAR GPT Module will ensure the data consistency while accessing its own RAM storage or hardware registers. The GPT module will use the following macros:

```
#define GPT_ENTER_CRITICAL_SECTION (Exclusive_Area)
SchM_Enter_Gpt_##Exclusive_Area ()

#define GPT_EXIT_CRITICAL_SECTION (Exclusive_Area)
SchM_Exit_Gpt_##Exclusive_Area ()
```

The following exclusive area along with scheduler services is used to provide data integrity for shared resources.

- GPT_RAM_DATA_PROTECTION
- GPT_INTERRUPT_CONTROL_PROTECTION

The protection area GPT_RAM_DATA_PROTECTION is used to protect the RAM Data.
e.g. GPT status variable accessing.

The protection area GPT_INTERRUPT_CONTROL_PROTECTION is used to protect the register.

These functions can be disabled by disabling the configuration parameter ‘GptCriticalSectionProtection’.
The flowchart indicates the flow with the precompile option ’GPT_CRITICAL_SECTION_PROTECTION’.

10.3 Architecture Details

The GPT Driver architecture is shown in the following figure. The GPT user shall directly use the APIs to configure and execute the GPT conversions.

The **Figure 10.3** shows the GPT Driver Component as defined in AUTOSAR architecture.

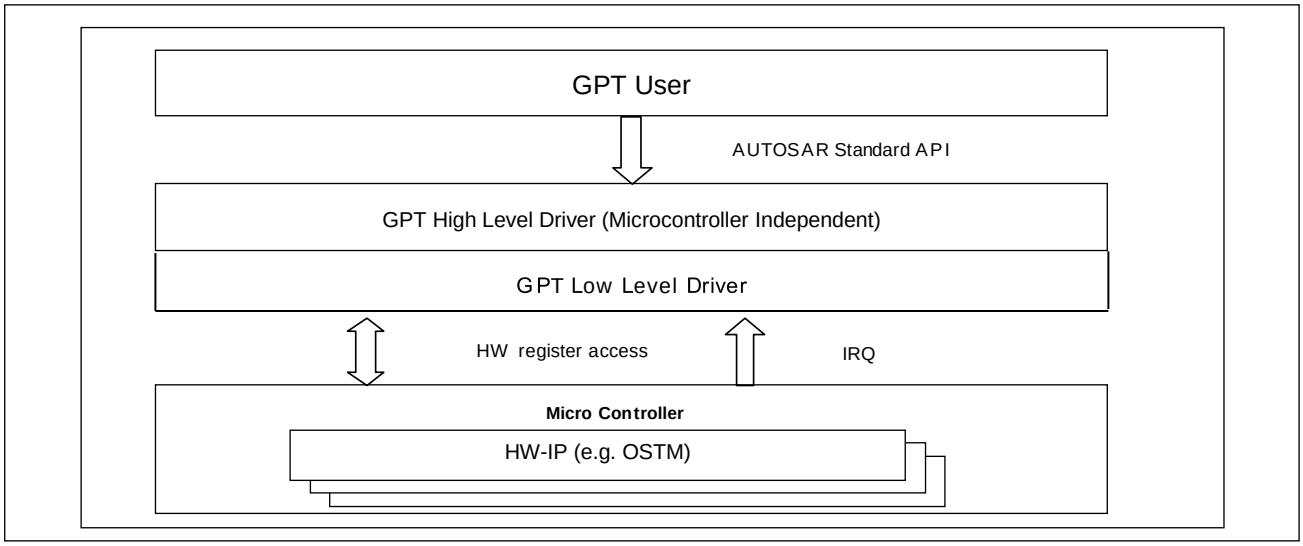


Figure 10.3 GPT Driver Architecture

The GPT Driver Component is divided into GPT High Level Driver and GPT Low Level Driver. Registers are accessed within the GPT Low Level Driver.

The GPT High Level Driver is used to check for the DET errors.

The GPT channel can be configured in either Continuous Mode or One-shot Mode. In Continuous Mode, the timers keep operating even after the target value is reached and it has multiple notifications if the configured channel is enabled.

Timers are used in the GPT Driver Component to generate timeout periods, except for channels that have configured the Predef Timer channel.

The GPT Driver component should provide the following services based on the functions performed by the GPT Driver:

- Initialization/De-Initialization
- Read timer value
- Start/Stop timer
- Notification
- Get version information
- Wakeup
- Read Predef timer values

The Table 10.1 shows the supported service list of the device.

Table 10.1 Supported Service List of GPT Module

Sl. No.	Service	TMU
1	Initialization/De-Initialization	√
2	Start/Stop timer	√
3	Notification	√
4	Wakeup	√
5	Read timer values	√
6	Get version information	No use Timer
7	Read Predef timer values	√

√: Service supported by the HW-IP.

-: Service not supported by the HW-IP.

- **Initialization/De-Initialization**

Initialization

This sub module provides the structures and APIs for both global and controller-specific initialization. It provides the services for initialization of the drivers before the timers are put into operation. This module also checks if the database is flashed. It is also responsible for selection of the prescaler value for the timer and setting the timer mode. The GPT Driver is initialized by invoking Gpt_Init () API. All the notifications are disabled after initialization. If the driver has already been initialized, all channels are stopped and the driver is reinitialized. The Gpt Predef Timer initializes and starts counter.

De-Initialization

This sub module provides the service to de-initialize the GPT Module. This service includes settings for all those peripherals configured statically. The GPT Driver Component is de-initialized by invoking Gpt_DeInit () API.

The Gpt Predef Timer stops counter and the driver is de-initialized.

After de-initialization, all GPT interrupts and notifications are disabled. Hence all configured GPT timers and the timer interrupts are disabled.

- **Read timer values**

Get Elapsed Time

This sub module provides the service to get elapsed time of the GPT channel. The API provides the services to get the time elapsed at a particular instant from the start of the timer. The parameter for this API includes the channel for which the elapsed time has to be calculated.

The elapsed time of the channel is read by invoking Gpt_GetTimeElapsed () API. This function calculates the elapsed time value by using the compare match register and timer counter register.

Get Remaining Time

This sub module provides the service to get remaining timer value until the next timeout period of the GPT channel expires. The remaining time of the channel is read by invoking Gpt_GetTimeRemaining () API.

This function calculates the remaining time value by using the compare match register and timer counter register.

- **Start/Stop timer**

Start Timer

This sub module provides the service to start the timer channel of the GPT with defined period. The GPT channel timer can be started by invoking Gpt_StartTimer () API.

Stop Timer

This sub module provides the service to stop the timer channel of the GPT with defined period. The GPT channel timer can be stopped by invoking Gpt_StopTimer () API.

- **Notification**

Enable Notification

This sub module provides the service to enable the timeout period notification of a channel according to the required notification parameter. This service enables the corresponding GPT channel notification flags. The notifications can be enabled by invoking the Gpt_EnableNotification () API.

Note: Wakeup and notification cannot be enabled on one channel at that same time.

Disable Notification

This sub module provides the service to disable the timeout period notification of a channel. This service disables the corresponding GPT channel notification flags. The notifications can be disabled by invoking the Gpt_DisableNotification () API.

- **Get version information**

Get Version Information

This sub module provides the service to get the version information of GPT Driver Component. The version information is available by invoking the Gpt_GetVersionInfo () API. No HW setting is required for this function.

- **Wakeup**

Set Mode

This sub module provides the service to set the operation mode of the GPT Driver. The mode can be switched by invoking Gpt_SetMode () API.

After initialization, the mode is “normal mode”. The mode can be switched to “sleep mode” by this service and vice versa.

Disable Wakeup

This sub module provides the service to disable the wakeup interrupt. This service disables the interrupt of the referenced channel configured for wakeup. The interrupt can be disabled by invoking Gpt_DisableWakeup () API.

This service is relevant to the operation mode of the GPT Driver.

Enable Wakeup

This sub module provides the service to enable the wakeup interrupt. This service enables the interrupt of the referenced channel configured for wakeup. The interrupt can be enabled by invoking Gpt_EnableWakeup () API.

This service is relevant to the operation mode of the GPT Driver.

Check Wakeup

This sub module provides the service to check the wakeup source. By invoking Gpt_CheckWakeup () API, it can check whether the source can be available for wakeup event.

- **Reading of Predef timer values**

Get Predef Timer Value

This sub module provides the service to get the Predef Timer value of the GPT. The GPT Predef Timer is a free counter and pre-defined timer.

The Predef Timer has two kinds of tick durations, 1 and 100 microseconds.

The service can get the current Predef Timer value by invoking Gpt_GetPredefTimerValue () API.

Note: When Gpt_Init () API is invoked, the Predef Timer starts counter. When Gpt_DeInit () API is invoked, the Predef Timer stops counter.

10.4 GPT Driver Component Header And Source File Description

This section explains the GPT Driver Component's C Source and C Header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by GPT Driver Generation Tool:

- Gpt_Cbk.h
- Gpt_Cfg.h

The C source file generated by GPT Driver Generation Tool:

- Gpt_PBcfg.c

The GPT Driver Component C header files and Component source files:
Refer to "[1] R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.3.4.3 Folder Structure.

The Stub C header files:
Refer to "[1] R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.2.9 Stubs File.

10.5 Application Programming Interface

This section explains the Data types and APIs provided by the GPT Driver Component to the Upper layers.

10.5.1 Imported Types

This section explains the Data types imported by the GPT Driver Component and lists the dependency on other modules.

10.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed:

- Std_ReturnType
- Std_VersionInfoType

10.5.1.2 Other Module Types

In this section, all types included in the module EcuM are listed:

- EcuM_WakeUpSourceType

10.5.2 Type Definitions

This section explains the type definitions of GPT Driver Component according to AUTOSAR Specification.

10.5.2.1 Gpt_ConfigType

The shows explanation of Gpt_ConfigType.

Table 10.2 Gpt_ConfigType

Name:	Gpt_ConfigType		
Type:	Structure		
Element	**Type**	**Element**	**Type**
	uint32	ulStartOfDbToc	Database start value.
	const Gpt_ChannelConfigType *	pChannelConfig	Pointer to GPT driver channels configuration.
	Gpt_ChannelRamDataType *	pChannelRamData	Pointer to the address of GPT driver status in RAM data.
	const void*	pPredefTimerConfig	Pointer to GPT driver Predef Timer configuration.
Description:	This structure exists for GPT Driver configuration.		

10.5.2.2 Gpt_ChannelType

The shows explanation of Gpt_ChannelType.

Table 10.3 Gpt_ChannelType

Name:	Gpt_ChannelType
Type:	uint8
Range:	0 to Max number of Channel ID.
Description:	Numeric ID of a GPT channel.

10.5.2.3 Gpt_ValueType

The shows explanation of Gpt_ValueType.

Table 10.4 Gpt_ValueType

Name:	Gpt_ValueType
Type:	uint32
Range:	0 to 4294967295
Description:	Used to read the current timer value/set the periodic timer values (number of ticks)

10.5.2.4 Gpt_PredefTimerType

The shows explanation of Gpt_PredefTimerType.

Table 10.5 Gpt_PredefTimerType

Name:	Gpt_PredefTimerType	
Type:	Enumeration	
Range:	GPT_PREDEF_TIMER_1US_16BIT	GPT Predef Timer with tick duration 1µs and range 16bit
	GPT_PREDEF_TIMER_1US_24BIT	GPT Predef Timer with tick duration 1µs and range 24bit
	GPT_PREDEF_TIMER_1US_32BIT	GPT Predef Timer with tick duration 1µs and range 32bit
	GPT_PREDEF_TIMER_100US_32BIT	GPT Predef Timer with tick duration 100µs and range 32bit
Description:	Type for GPT Predef Timers	

10.5.3 Function Definitions

The Table 10.6 explains the APIs provided by the GPT Driver Component.

Table 10.6 APIs provided by the GPT Driver Component

Sl. No	APIs
AUTOSAR API	
1	Gpt_GetVersionInfo
2	Gpt_Init
3	Gpt_DeInit
4	Gpt_StartTimer
5	Gpt_StopTimer
6	Gpt_SetMode
7	Gpt_EnableNotification
8	Gpt_DisableNotification
9	Gpt_GetTimeElapsed
10	Gpt_GetTimeRemaining
11	Gpt_DisableWakeup
12	Gpt_EnableWakeup
13	Gpt_CheckWakeup
14	Gpt_GetPredefTimerValue
Renesas Original API	
None	

10.5.3.1 Renesas Original API

None

10.5.4 Preemption of APIs

The following table specifies the preemption of each API that can be invoked at the same time as the API.
The following Table 10.7 shows Preemption Table of APIs of the GPT Driver.

Table 10.7 Preemption of APIs in the GPT Driver

	Gpt_GetVersionInfo	Gpt_Init	Gpt_DeInit	Gpt_GetTimeRemaining	Gpt_GetTimeElapsed	Gpt_StartTimer	Gpt_StopTimer	Gpt_EnableNotification	Gpt_DisableNotification	Gpt_SetMode	Gpt_DisableWakeup	Gpt_EnableWakeup	Gpt_CheckWakeup	Gpt_GetPredefTimerValue
Gpt_GetVersionInfo	√													
Gpt_Init	√	-												
Gpt_DeInit	√	-	-											
Gpt_GetTimeRemaining	√	-	-	-										
Gpt_GetTimeElapsed	√	-	-	√	-									
Gpt_StartTimer	√	-	-	√	-	*1								
Gpt_StopTimer	√	-	-	√	√	*1	*1							
Gpt_EnableNotification	√	-	-	√	√	√	√	*1						
Gpt_DisableNotification	√	-	-	√	√	√	√	*1	*1					
Gpt_SetMode	√	-	-	-	-	-	-	-	-	-				
Gpt_DisableWakeup	√	-	-	√	√	√	√	√	√	-	*1			
Gpt_EnableWakeup	√	-	-	√	√	√	√	√	√	-	*1	*1		
Gpt_CheckWakeup	√	-	-	√	√	√	√	√	√	-	√	√	√	
Gpt_GetPredefTimerValue	√	-	-	√	√	√	√	√	√	-	√	√	√	√

Note:
-: cannot be invoked at the same time
√: can be invoked at the same time
*1: The same channel cannot be invoked at the same time

10.6 Development And Production Errors

In this section, the development errors reported by the GPT Driver Component are tabulated. The development errors will be reported only when the pre-compiler option ‘GptDevErrorDetect’ is enabled in the configuration. The production code errors are not supported by GPT Driver Component.

10.6.1 GPT Driver Component Development Errors

The Table 10.8 and Table 10.9 contain the DET errors reported by GPT Driver Component. These errors are reported to Default Error Tracer Module when the GPT Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 10.8 DET Errors Of GPT Driver Component (1/2)

Sl. No.	**1**
Error Code	GPT_E_UNINIT
Value (hex)	0x0A
Related API(s)	Gpt_DeInit, Gpt_StartTimer, Gpt_StopTimer, Gpt_SetMode, Gpt_EnableNotification, Gpt_DisableNotification, Gpt_GetTimeElapsed, Gpt_GetTmeRemaining, Gpt_CheckWakeup, Gpt_DisableWakeup, Gpt_EnableWakeup, Gpt_GetPredefTimerValue
Source of Error	When the APIs are invoked without the initialization of the GPT Driver Component.
Sl. No.	**2**
Error Code	GPT_E_BUSY
Value (hex)	0x0B
Related API(s)	Gpt_DeInit and Gpt_StartTimer.
Source of Error	When the function is called while timer is already running.
Sl. No.	**3**
Error Code	GPT_E_PARAM_CHANNEL
Value (hex)	0x14
Related API(s)	Gpt_StartTimer, Gpt_StopTimer, Gpt_EnableNotification, Gpt_DisableNotification, Gpt_GetTimeElapsed, Gpt_GetTmeRemaining, Gpt_EnableWakeup, Gpt_DisableWakeup.
Source of Error	When the APIs are invoked with the invalid channel identifier.
Sl. No.	**4**
Error Code	GPT_E_PARAM_VALUE
Value (hex)	0x15
Related API(s)	Gpt_StartTimer
Source of Error	When the API is invoked with the invalid timer value.
Sl. No.	**5**
Error Code	GPT_E_ALREADY_DISABLED
Value (hex)	0xF1
Related API(s)	Gpt_DisableNotification.
Source of Error	When the API is invoked for a channel and its notification was already disabled.
Sl. No.	**6**
Error Code	GPT_E_ALREADY_ENABLED

Value (hex)	0xF2
Related API(s)	Gpt_EnableNotification
Source of Error	When the API is invoked for a channel and its notification was already enabled.
Sl. No.	**7**
Error Code	GPT_E_PARAM_POINTER
Value (hex)	0x16
Related API(s)	Gpt_Init, Gpt_GetVersionInfo, Gpt_GetPredefTimerValue.
Source of Error	When ConfigPtr is NULL pointer or the (hardware-specific) contents of the given configuration are not within the allowed boundaries.

Table 10.9 DET Errors Of GPT Driver Component (2/2)

Sl. No.	8
Error Code	GPT_E_INVALID_DATABASE
Value (hex)	0xEF
Related API(s)	Gpt_Init
Source of Error	When the API is invoked with the Invalid database.
Sl. No.	**9**
Error Code	GPT_E_ALREADY_INITIALIZED
Value (hex)	0x0D
Related API(s)	Gpt_Init
Source of Error	When the API is invoked when the module is already initialized.
Sl. No.	**10**
Error Code	GPT_E_INIT_FAILED
Value (hex)	0x0E
Related API(s)	Gpt_Init
Source of Error	When the Initialize function is failed.
Sl. No.	**11**
Error Code	GPT_E_PARAM_MODE *1
Value (hex)	0x1F
Related API(s)	Gpt_SetMode
Source of Error	When the mode parameter is invalid.
Sl. No.	**12**
Error Code	GPT_E_PARAM_PREDEF_TIMER
Value (hex)	0x17
Related API(s)	Gpt_GetPredefTimerValue
Source of Error	When the API parameter checking invalid Predef Timer.

*1: Those error codes are only used in wakeup function.

10.6.2 GPT Driver Component Production Errors

Table 10.10 DEM Errors Of GPT Driver Component

Sl. No.	1
Error Code	GPT_E_INTERRUPT_CONTROLLER_FAILURE
Value	Depend on configuration.
Related Function	Gpt_HW_<HW-IP>_CbkNotification (<HW-IP>_CH_ISR)
Source of Error	When unintended interrupt occurs, this error is reported.

10.7 GPT Driver Component Runtime Errors

The following table contains the Runtime errors reported by GPT Driver Component. These errors are reported to Default Error Tracer Module when the GPT Driver Component APIs are invoked in time and the GPT Driver Component does not meet the requirement to execute the API.

Table 10.11 Runtime Errors of GPT Driver Component for V4H devices

Sl. No.	1
Error Code	GPT_E_BUSY
Value (hex)	0x0B
Related API(s)	Gpt_DeInit Gpt_StartTimer
Source of Error	API service called when timer channel is still busy (running).

Table 10.12 Runtime Errors of GPT Driver Component for V4H devices

Sl. No.	1
Error Code	GPT_E_MODE
Value (hex)	0x0C
Related API(s)	Gpt_GetPredefTimerValue
Source of Error	API service called when driver is in wrong mode.

10.8 Memory Organization

The following table depicts a typical memory organization, which must be met for proper functioning of GPT Driver Component software.

Table 10.13 ROM / RAM sections of the GPT Driver

Section Name	Alignment (*1)	Description
GPT_PUBLIC_CODE_ROM	-	This section contains codes which belong to the API functions of the GPT Driver.
GPT_PRIVATE_CODE_ROM	-	This section contains codes which belong to the internal functions of the GPT Driver.
GPT_FAST_CODE_ROM	-	This section contains codes which belong to the interrupt function of the GPT Driver.
GPT_CFG_DATA_UNSPECIFIED	-	This section contains post-build config miscellaneous data of the GPT Driver.
GPT_CFG_DBTOC_UNSPECIFIED	-	This section contains post-build config data table of the GPT Driver.
RAM_1BIT	-	This section contains variables that need to be initialized before Gpt_Init.
GPT_CFG_RAM_UNSPECIFIED	-	This section contains miscellaneous variables that do not need to be initialized before Gpt_Init.
VAR_NO_INIT_PTR	-	This section contains the pointer variables that do not need to be initialized before Gpt_Init.
NOINIT_RAM_8BIT	-	This section contains 8-bits variables that do not need to be initialized before Gpt_Init.

Note:
*1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

10.9 Device-Specific Information

The device supports the following devices:
Refer to “[1] R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 5.1 Product.

10.9.1 Multi-Core / Multi-Instantiation

GPT driver does not support multi-core and multi-instantiation for this device.

10.9.2 Interaction Between The User And GPT Driver Component

10.9.2.1 Channel Mapping

The hardware timer channels available for V4H are as given in the Table 10.13.

Table 10.14 HW Timers Channel Mapping

Sl.No	Hardware Channel
1	TMU_CHmm(mm = 00..14)

10.9.2.2 ISR Functions

The table below provides the list of handlers corresponding to the hardware unit ISR(s) in GPT Driver Component. The user should configure the ISR functions mentioned in Table 10.14.

Table 10.15 ISR Handler Addresses.

Interrupt Source	Name of the ISR Function
GPT_TMU_CHn_ISR_API (n = 0.. 14)	GPT_TMU_CHn_ISR_API

10.9.2.3 Translation Header File

Refer to “[1] R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 3.2.4 Translation Header File.

10.9.2.4 Parameter Definition File

Refer to “[1] R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 3.2.3 Parameter Definition File.

10.10 Non-AUTOSAR environment integration

The GPT Driver Components for Renesas R-Car Series, 4th Generation is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the GPT Driver Components to Non-AUTOSAR environment.

10.10.1 Stub modules handling

10.10.1.1 Os

The Os stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar version>\Os
In the AUTOSAR environment, GPT Driver is relying on OS module version information and for interrupt category definitions.

10.10.1.2 Det

The Det stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar version>Det
In the AUTOSAR environment, GPT Driver Components uses Det_ReportError and Det_ReportRuntimeError API(s) provided by the DET module to report a development error e.g. GPT Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:
Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)
Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.
Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

10.10.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.
The SchM (Basic Software Scheduler) stub files are organized in the following folder:
rel\common\generic\stubs\Rte\
GPT driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when GPT_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON.
With Non-AUTOSAR environment, user needs to prepare SchM stub to user GPT driver as following:
Write SchM functions with prototype as below:
void SchM_Enter_Gpt_GPT_ GPT_RAM_DATA_PROTECTION (void);
void SchM_Exit_Gpt_GPT_ GPT_RAM_DATA_PROTECTION (void);
void SchM_Enter_Gpt_GPT_INTERRUPT_CONTROL_PROTECTION_AREA(void);
void SchM_Exit_Gpt_GPT_INTERRUPT_CONTROL_PROTECTION_AREA(void);

10.10.1.4 Dem

The Dem stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar version>\Dem
In the AUTOSAR environment, GPT Driver Components uses Dem_ReportErrorStatus and Dem_SetEventStatus API provided by the DEM module to report a production error. The API prototype is as of follow:
Dem_ReportErrorStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)
Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)
Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.
Non-AUTOSAR users can modify the provided Dem_ReportErrorStatus and Dem_SetEventStatus API with their current error handling strategy.

10.10.2 Callback function usage

The GPT Driver Component provide the callback functions as described at section 10.2.1.
Example: void Gpt_Notification_0()

10.10.3 Scheduled function usage

AUTOSAR does not define any runtime error code (see Chapter 8.6 of AUTOSAR Specification of GPT Driver (AUTOSAR_SWS_GPTDriver.pdf)

10.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:
\external\rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

11.ICCOM

11.1 Overview

The purpose of this document is to describe the information related to ICCOM Complex Device Driver Component.

This document is intended for the developers of ECU software on R-Car Series, 4th Generation SoC using Application Programming Interfaces provided by AUTOSAR specification for ICCOM Complex Device Driver. The ICCOM Complex Device Driver Component provides the following services:

- ICCOM Complex Device Driver Component initialization.
- Transmission data to AP-System Core domain.
- Notification back to ASWC.
- Reception data from AP-System Core domain.
- Periodic send processing to resend pending request (Send Data or ACK).
- Getting Version Information.

The ICCOM Complex Device Driver Component comprises of two parts: Embedded Software and Generation Tool to achieve scalability and configurability.

The purpose of this document is to describe the information related to Embedded Software part of the ICCOM Complex Device Driver Component for R-Car Series, 4th Generation SoC. Please refer to ICCOM in "*R-Car <DeviceName> AUTOSAR R19-11 MCAL Driver Component Generation Tool User's Manual*" for the detail of Generation Tool part.

The below figure depicts the ICCOM Complex Device Driver as part of layered AUTOSAR MCAL Layer:

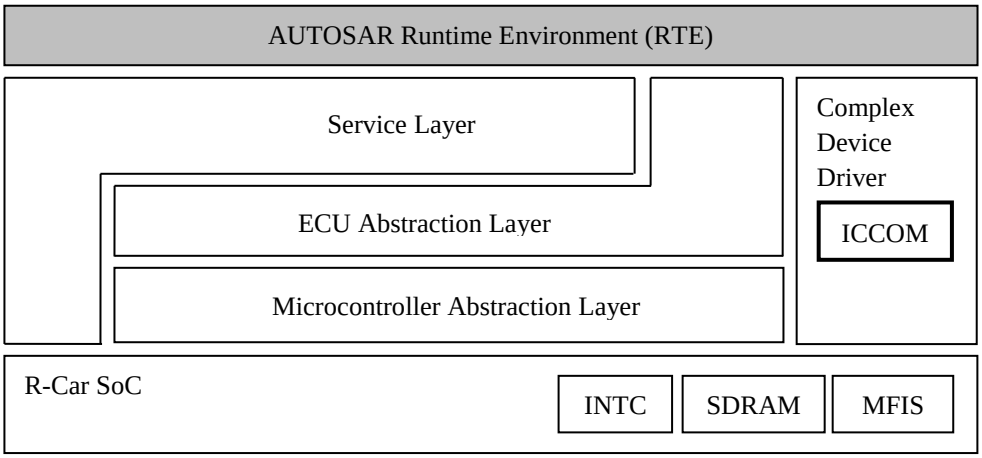


Figure 11.1 System Overview of the ICCOM Complex Device Driver in AUTOSAR Layer Architecture

The following diagram shows the system overview of the AUTOSAR Architecture for ICCOM Complex Device Driver:

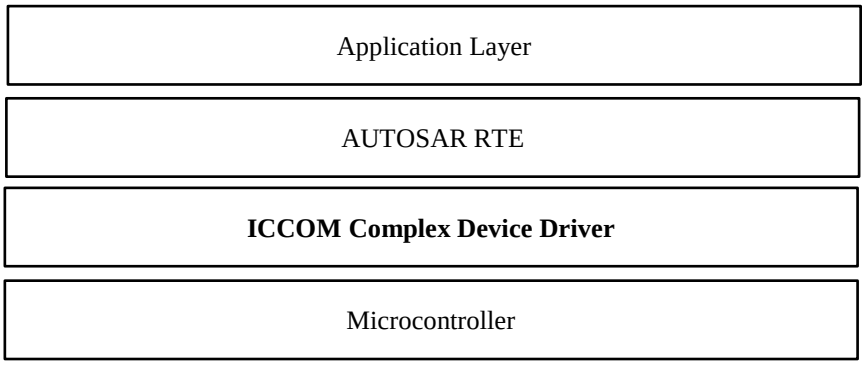


Figure 11.2 System Overview of AUTOSAR Architecture

The ICCOM Complex Device Driver Component is Vendor Specific Module provided by Renesas to communicate between the AP-System core and the Cortex-R52 core domain.

Sample Application is available for user to understand and test ICCOM Complex Device Driver without proper AUTOSAR upper layer, which is described in section *"3.4.1.7 Sample Application"* of *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"*.

Please refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* - section 3.7.2.2 How to build the Sample Application for more information on how to configure, compile and flash the ICCOM Complex Device Driver to R-Car SoC.

11.2 Forethoughts

11.2.1 General

Following information will aid the user to use the ICCOM Complex Device Driver Component software efficiently:

- ICCOM Complex Device Driver is used to interface with AP-System Core. Usage of ICCOM Complex Device Driver requires proper ICCOM driver/application from AP-System Core.
- MCU specific initializations such as reset registers, one-time writable registers, interrupt stack pointer, user stack pointer, internal watchdog, specific features of internal memory and registers is not implemented by ICCOM Complex Device Driver. These initializations must be implemented by the start-up code.
- The ISR functions and the corresponding handler addresses are provided in section *"ISR Functions"* of this document. User must ensure that Interrupt Vector table configuration is done as per the information provided in the table.
- Porting specific information such as compiler details, timing details and memory consumption are provided in chapter *"Device Specific Information"* of this document.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- If any development errors are reported to DET, the normal flow of execution of driver will be aborted.
- All production errors will be reported to DEM by using the API Dem_SetEventStatus provided by DEM.
- All interrupt-related functions are mapped to CODE_FAST section (see Table 7.12 of *AUTOSAR Specification of Memory Mapping*). Other functions are mapped to CODE_SLOW section (see Table 7.13 of *AUTOSAR Specification of Memory Mapping*).
- The number of channels that are configured should be maintained in single post-build configuration set.
- ICCOM Complex Driver supports one instance, therefore, configuration parameter CddInstanceId should be configured as 0. In the implementation, instance ID is always used as 0 without using CddInstanceId parameter in CDF setting.

11.2.2 Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the ICCOM Complex Device Driver Component:

- The CDD_Iccom_Cfg.h file is generated by the ICCOM Complex Device Driver Component Code Generation Tool must be compiled and linked along with ICCOM Complex Device Driver Component source files.
- The application has to be rebuilt, if there is any change in the CDD_Iccom_Cfg.h files generated by the ICCOM Complex Device Driver Component Generation Tool.
- The CDD_Iccom_PBcfg.c is generated for single configuration using ICCOM Complex Device Driver Component Generation Tool can be compiled and linked independently.
- The ICCOM Complex Device Driver Component needs to be initialized before accepting any request. The API CddIccom_Init should be invoked to initialize ICCOM Complex Device Driver Component before calling any other ICCOM Complex Device Driver API.
- Before transmission or reception, other domain (e.g. AP-System Core) shall send an initial negotiation to confirm that the initialization has been done at others domain.
- Input parameters are validated only when the static configuration parameter CddIccomDevErrorDetect is enabled. Application should ensure that the right parameters are passed while invoking the APIs when CddIccomDevErrorDetect is disabled.
- ICCOM Complex Device Driver included maximum of

+ 04 channel for communication with AP-System Core (core 0..3) from CR52 domain (core 0..2).

Each channel shall be configurable to use a specific MFIS Communication pair registers.

- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used.

- CDDICCOM driver provides Write Verify Check feature to verify the execution of control registers' write operation in registers MFISARIMBR<m><n> and MFISAREMBR<m><n> registers (<m>=0..2, <n>=0..3) when the static configuration parameter CddIccomWriteVerifyCheck is enable. After writing to a control register, value of that register is read back to make sure that register has been written correctly.
- The ICCOM driver need at least 1400 ticks to wait for timeout and the ticks is calculate by formula: CddIccomChannelTimeoutDuratiuon/OsSecondsPerTick. User should take care of CddIccomChannelTimeoutDuratiuon and OsSecondsPerTick value to satisfy the condition.
- Upper layer should have some timing protection mechanisms to ensure the performance of ICCOM Complex Device Driver.
- ISR priority should be configured to be greater than priority of the thread/task which invoke the driver's APIs. Otherwise, the driver operation is not guaranteed.
- CDDICCOM driver use a Common Transmission Area (CTA) to communicate among CPUs. One domain copy data from user memory to CTA and opposite direction for other domain in case of receiving. To avoid cache coherence problem when L1 data cache of Arm Cortex-R enable, user need to provide some functions with following names and formats for cache maintenance. CDDICCOM driver will use them to maintain the cached memory area which it uses. The operation of CDDICCOM driver will not be guaranteed if L1 data cache is enabled but these functions are not implemented. The size of one cache line on Arm Cortex-R is 32 bytes, so these functions should accept 32 bytes aligned input start address. The input size can be unaligned, these functions have to calculate proper number of cache lines based on this size. For recommended usage, when L1 data cache is enabled, the start address of CTA area (CddIccomChannelCtaAddress) and the size of each partition (CddIccomChannelPartitionSize) should aligned with the size of one L1 data cache line. User also should implement these functions, at least, for driver compilation if L1 data cache is disabled because CDDICCOM driver will always call them regardless to status of L1 data cache.

Function Name	CR7_Flush_DCache_By_Addr	
Syntax:	CR7_Flush_DCache_By_Addr (uint32 start_addr, uint32 size)	
Parameters (In):	start_addr	Start address of the buffer that D-Cache need to be maintained.
	size	Number of bytes of above buffer.
Parameters (In-Out):	None.	
Parameters (Out):	None.	
Return Value:	None.	
Description:	This API shall clean and invalidate L1 data cache by address and size need to be maintained.	

Function Name:	CR7_Invalidate_DCache_By_Addr	
Syntax:	CR7_Invalidate_DCache_By_Addr (uint32 start_addr, uint32 size)	
Parameters (In):	start_addr	Start address of the buffer that L1 data cache need to be maintained.
	size	Number of bytes of above buffer.
Parameters (In-Out):	None.	
Parameters (Out):	None.	
Return Value:	None.	
Description:	This API shall invalidate L1 data cache by address and size of the buffer.	

- According to ordering requirement for memory accesses on ARMv8, the memory type of Common Transmission Area (CTA) should be configured as Device/Strongly ordered (when MPU is on) to ensure the program order for instruction execution.
- User must initialize the following hardware IPs as below before initializing CDDICCOM module.

Table 11.1 List of hardware IP affect to CDDICCOM

Hardware IP	Module Name	Expected Setting	Description
CPG	MCU	User has to check to confirm that the clock is supplied for CDDICCOM module (MFIS HW IP).	MCU handle the clock supply control of CDDICCOM module, please refer to User Manual of MCU module for detail setting.
MFIS	-	Write Access protection in MFIS should be disabled. **MFISWPCNTR**: Bit0 (WPD) should be set to '1' to disable write access protection.	CDDICCOM for the R-Car Gen4 is designed assuming the setting of this protection mechanism is disabled, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed since it may not write to registers of MFIS.
AXI-bus	-	SDRAM address mapping in HW should be configured according to Chapter 18 in [3] HW UM.	CDDICCOM is designed assuming the setting of SDRAM address mapping is corrected, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since ICCOM operation is under controlled by DRAM for memory control.
DBSC5	-	The maximum use of SDRAM bus bandwidth will be enabled by DBSC5 according to Chapter 33 in [3]HW UM.	CDDICCOM is designed assuming the configuration of SDRAM bus bandwidth is suitable for R-Car V4H, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since CDDICCOM operation is under controlled by DRAM for memory control.

11.2.3 Data Consistency

To support the re-entrance and interrupt services, the ICCOM Complex Device Driver Component will ensure the data consistency while accessing its own RAM storage or hardware registers. The ICCOM Complex component will use SchM_Enter_CddIccom_<Exclusive Area> and SchM_Exit_CddIccom_<Exclusive Area> functions. The SchM_Enter_CddIccom_<Exclusive Area> function is called before accessing the data needs to be protected and SchM_Exit_CddIccom_<Exclusive Area> function is called after the data is accessed.

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

- CDDICCOM_INTERRUPT_CONTROL_PROTECTION<n> (<n> = 0..3).
- CDDICCOM_RAM_DATA_PROTECTION<n> (<n> = 0..3).

These functions can be disabled by disabling the configuration parameter 'CddIccomCriticalSectionProtection'.

11.3 Architecture Details

The ICCOM Complex Device Driver architecture is shown in the following figure:

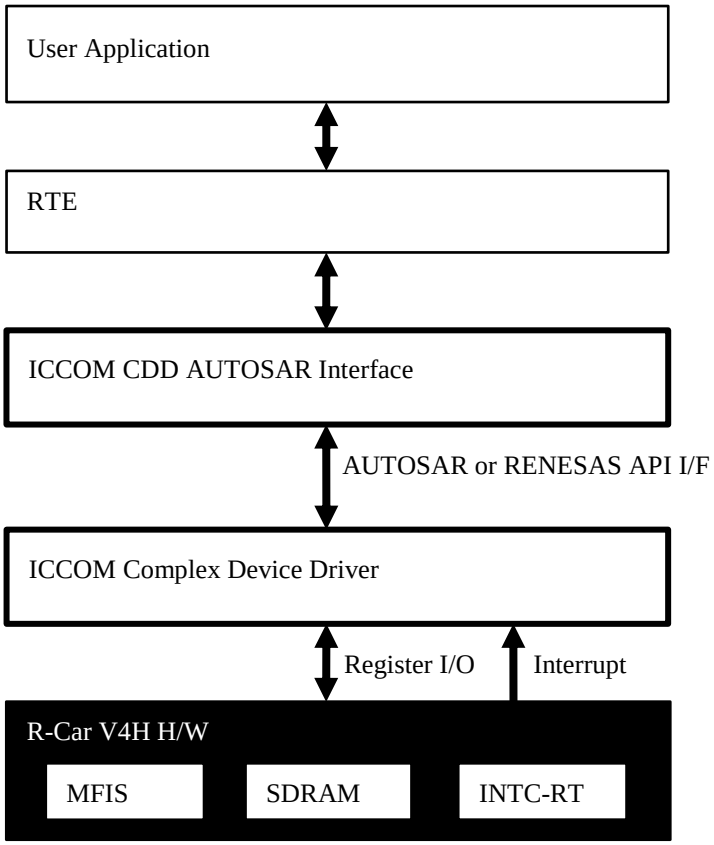


Figure 11.3 ICCOM Complex Device Driver Component Architecture

The ICCOM Complex Device Driver Component can be divided into following sub-component based on the functions performed by the ICCOM Complex Device Driver:

- Driver Initialization.
- Send data to AP-System core domain.
- Notify data received event to AUTOSAR SWC.
- Receive data from AP-System core domain.
- Schedule internal function.
- Get Version Information

Driver Initialization

The driver initialization sub-module internally stores the configuration data address to enable subsequent API calls to access the configuration data and initializes the global variables used by ICCOM Complex Device Driver Component.

The API related to this sub-module is **CddIccom_Init**.

Send data processing

The ICCOM Complex Device Driver Component (run on CR52) provides a service to send data to CA System Core.

Following AUTOSAR interface are provided:

- Interface: If_IccomSend (Client-Server)
- Provided Port: P_Ch<n>Send (<n> = 0..3)
- Runnable Entity: CddIccom_Ch<n>SendRun (<n> = 0..3)

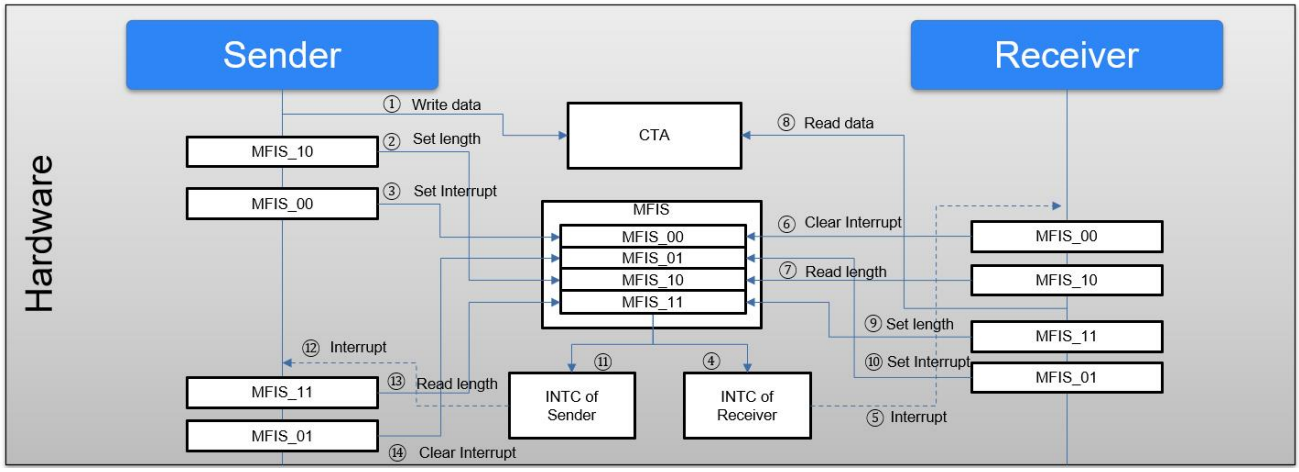


Figure 11.4 ICCOM Transmission Overall Structure

Table 11.2 ICCOM Complex Device Driver MFIS HW registers use for transmission process

	CR52 sends data to AP-System core (<m>=0..2, <n>=0..3)
MFIS_00	MFISAREICR<m><n>
MFIS_01	MFISARIICR<m><n>
MFIS_10	MFISAREMBR<m><n>
MFIS_11	MFISARIMBR<m><n>

Notify data received event to AUTOSAR SWC

The ICCOM Complex Device Driver provides a Sender-Receiver Port to notify data received event to AUTOSAR SWC.

Following AUTOSAR interface are provided:

- Interface: If_IccomNotice (Sender-Receiver).
- Provided Port: P_Ch<n>Notice (<n> = 0..3).

Receive data processing

The ICCOM Complex Device Driver Component (run on CR52 domain) provides a service to receive data to another partner.

Following AUTOSAR interface are provided:

- Interface: If_IccomReceive (Client-Server)
- Provided Port: P_Ch<n>Receive (<n> = 0..3)

- Runnable Entity: CddIccom_Ch<n>ReceiveRun (<n> = 0..3)

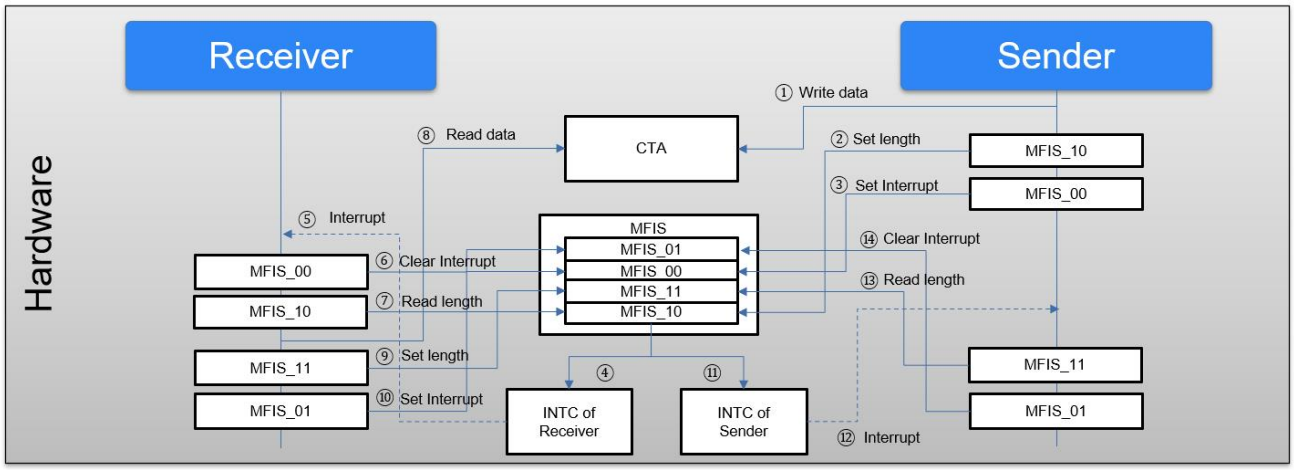


Figure 11.5 ICCOM Reception Overall Structure

Table 11.3 ICCOM Complex Device Driver MFIS HW registers use for reception process

	CR52 receives data from AP-System core (<m>=0..2, <n>=0..3)
MFIS_00	MFISARIICR<m><n>
MFIS_01	MFISAREICR<m><n>
MFIS_10	MFISARIMBR<m><n>
MFIS_11	MFISAREMBR<m><n>

Schedule internal function

The ICCOM Complex Driver needs to be scheduled to work properly.
Following scheduled function is provided to be invoked by SchM: **CddIccom_MainFunction_Send**.

Retrieval Version Information

The ICCOM Complex Device Driver provides a service to return version information details to the User.
The API related to this sub-module is **CddIccom_GetVersionInfo**.

11.4 ICCOM Complex Device Driver Component Header And Source File Description

This chapter explains the ICCOM Complex Device Driver Component's source and header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by ICCOM Complex Device Driver Generation Tool:

- CDD_Iccom_Cfg.h
- CDD_Iccom_Cbk.h

The C source file generated by ICCOM Complex Device Driver Generation Tool:

- CDD_Iccom_PBcfg.c

The ICCOM Complex Device Driver Component C header files and Component source files:

Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* – section 3.4.1.3 Folder Structure.

The Stub C header files and source file:

Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* – section 3.2.9 Stubs File.

11.5 Application Programming Interface

This chapter explains the Data types and APIs provided by the ICCOM Complex Device Driver Component to the Upper layers.

11.5.1 Imported Types

This chapter explains the Data types imported by the ICCOM Complex Device Driver Component and lists its dependency on other modules.

11.5.1.1 Standard Types

In this chapter all types included from the Std_Types.h (according to "*AUTOSAR Specification of Standard Types"*) are listed:

- Std_ReturnType.
- Std_VersionInfoType.

11.5.1.2 OS Types

In this chapter all types included from the Os.h (according to "*AUTOSAR Specification of Operating System"*) are listed.

- CounterType.
- TickType.
- TickRefType.

11.5.1.3 Dem Types

In this chapter all types included from the Dem.h (according to "*AUTOSAR Specification of Diagnostic Event Manager"*) are listed:

- Dem_EventIdType.
- Dem_EventStatusType.

11.5.2 Type Definitions

This chapter explain the type definitions of ICCOM Complex Device Driver Component according to AUTOSAR Specification.

11.5.2.1 CddIccom_ChannelType

Table 11.4 CddIccom_ChannelType

Name:	CddIccom_ChannelType	
Type:	typedef	
Range:	uint8	0..3
Description:	Represents the identifier of ICCOM channel	

11.5.2.2 **CddIccom_ReturnType**

Table 11.5 CddIccom_ReturnType

Name:	CddIccom_ReturnType	
Type:	Enumeration	
Range:	CDDICCOM_E_OK	No error status.
	CDDICCOM_E_ERR	Generic error, no specific.
	CDDICCOM_E_BUSY	ICCOM channel busy status.
	CDDICCOM_E_NO_DATA	No receiving data status.
	CDDICCOM_E_SIZE	Invalid data size status.
Description:	Enumeration for CDDICCOM APIs call return value.	

11.5.2.3 **CddIccom_NoticeType**

Table 11.6 CddIccom_NoticeType

Name:	CddIccom_NoticeType	
Type:	Enumeration	
Range:	CDDICCOM_NOTICE_INVALID	Invalid ICCOM notification.
	CDDICCOM_NOTICE_INIT	ICCOM initial negotiation notification.
	CDDICCOM_NOTICE_DATA	ICCOM data notification.
	CDDICCOM_NOTICE_ACK	ICCOM acknowledgement notification.
Description:	Enumeration for the ICCOM notice type.	

11.5.2.4 **CddIccom_ChStaType**

Table 11.7 CddIccom_ChStaType

Name:	CddIccom_ChStaType	
Type:	Enumeration	
Range:	CDDICCOM_CH_WAITINIT	Channel is waiting for initial negotiation.
	CDDICCOM_CH_READY	Channel is ready.
Description:	Enumeration for the ICCOM channel state.	

11.5.2.5 **CddIccom_CtaPartType**

Table 11.8 CddIccom_CtaPartType

Name:	CddIccom_CtaPartType	
Type:	Enumeration	
Range:	CDDICCOM_CTA_UPPER	Upper partition of CTA.
	CDDICCOM_CTA_BOTTOM	Bottom partition of CTA.
	CDDICCOM_CTA_NONE	No CTA partition specified.
Description:	Enumeration for the ICCOM CTA partition indicator.	

11.5.2.6 **CddIccom_SndStaType**

Table 11.9 CddIccom_SndStaType

Name:	CddIccom_SndStaType	
Type:	Enumeration	
Range:	CDDICCOM_SND_IDLE	Send operation is in idle state.
	CDDICCOM_SND_REQUEST	Send operation requested.
	CDDICCOM_SND_WAITACK	Send operation is waiting for acknowledgement.
	CDDICCOM_SND_TIMEOUT	Timeout occurred in send operation.
Description:	Enumeration for the ICCOM channel send state.	

11.5.2.7 **CddIccom_RcvStaType**

Table 11.10 CddIccom_RcvStaType

Name:	CddIccom_RcvStaType	
Type:	Enumeration	
Range:	CDDICCOM_RCV_IDLE	Send operation is in idle state.
	CDDICCOM_RCV_REQUEST	Send operation requested.
	CDDICCOM_RCV_SNDACK	Send operation is waiting for acknowledgement.
Description:	Enumeration for the ICCOM channel receive state.	

11.5.2.8 **CddIccom_ChannelStatusType**

Table 11.11 CddIccom_ChannelStatusType

Name:	CddIccom_ChannelStatusType	
Type:	Structure	
Elements:	VAR(CddIccom_ChStaType, TYPEDEF)	enChSta
	VAR(CddIccom_CtaPartType, TYPEDEF)	enRcvCtaCurr
	VAR(CddIccom_CtaPartType, TYPEDEF)	enRcvCtaNext
	VAR(CddIccom_CtaPartType, TYPEDEF)	enSndCtaCurr
	VAR(CddIccom_CtaPartType, TYPEDEF)	enSndCtaNext
	VAR(CddIccom_SndStaType, TYPEDEF) [CDDICCOM_CTA_CNT]	enSndSta
	VAR(CddIccom_RcvStaType, TYPEDEF) [CDDICCOM_CTA_CNT]	enRcvSta
	VAR(TickType, TYPEDEF) [CDDICCOM_CTA_CNT]	ulTimeoutCntTick
	VAR(uint32, TYPEDEF) [CDDICCOM_CTA_CNT]	ulSndSize
	VAR(uint32, TYPEDEF) [CDDICCOM_CTA_CNT]	ulRcvSize
Description:	Identify the desired status information for the ICCOM channel.	

11.5.2.9 **CddIccom_ChannelConfigType**

Table 11.12 CddIccom_ChannelConfigType

Name:	CddIccom_ChannelConfigType	
Type:	Structure	
Elements:	P2VAR(uint32, TYPEDEF, REGSPACE)	pMFISxICRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pMFISyICRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pMFISxMBRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pMFISyMBRnReg
	P2CONST(uint8, TYPEDEF, CDDICCOM_APPL_DATA)	pCtaRxUpper
	P2CONST(uint8, TYPEDEF, CDDICCOM_APPL_DATA)	pCtaRxBottom
	P2VAR(uint8, TYPEDEF, CDDICCOM_APPL_DATA)	pCtaTxUpper
	P2VAR(uint8, TYPEDEF, CDDICCOM_APPL_DATA)	pCtaTxBottom
	VAR(uint32, TYPEDEF)	uiCtaPartSize
	VAR(CounterType, TYPEDEF)	ddTimeoutCntId
	VAR(uint32, TYPEDEF)	uiTimeout
	P2FUNC(void, TYPEDEF, pNotification) (IccomNoticeType, uint32)	pNotification
	P2FUNC(void, TYPEDEF, pEnterRegProtect)(void)	pEnterRegProtect
	P2FUNC(void, TYPEDEF, pExitRegProtect)(void)	pExitRegProtect
	P2FUNC(void, TYPEDEF, pEnterGlbProtect)(void)	pEnterGlbProtect
	P2FUNC(void, TYPEDEF, pExitGlbProtect)(void)	pExitGlbProtect
Description:	Identify the desired configuration for ICCOM channel.	

11.5.2.10 **CddIccom_ConfigType**

Table 11.13 CddIccom_ConfigType

Name:	CddIccom_ConfigType	
Type:	Structure	
Elements:	VAR(uint32, TYPEDEF)	ulStartOfDbToc
	P2CONST(CddIccom_ChannelConfigType, TYPEDEF , CDDICCOM_APPL_CONST)	pChannelConfig
	P2VAR(CddIccom_ChannelStatusType, TYPEDEF, CDDICCOM_APPL_DATA)	pChannelStatus
Description:	Identify the desired configuration for the ICCOM Complex Device Driver	

11.5.3 Function Definitions

Table 11.14 APIs provided by the ICCOM Complex Device Driver Component

Sl. No.	APIs
AUTOSAR API	
1	CddIccom_Init
2	CddIccom_GetVersionInfo
RENESAS API	
3	CddIccom_Ch<n>SendRun (<n>=0..3)
4	CddIccom_Ch<n>ReceiveRun (<n>=0..3)
5	CddIccom_MainFunction_Send
6	CddIccom_Ch<n>NoticeCallback (<n>=0..3)

11.5.3.1 **CddIccom_Ch<n>SendRun**

Name:	CddIccom_Ch<n>SendRun (<n> = 0..3)		
Prototype:	FUNC(Std_ReturnType, CddIccom_CODE) CddIccom_Ch<n>SendRun (IccomDataPtr LpData, uint32 LulSize)		
Service Id:	0x03 + <n> (<n> = 0..3)		
Sync/Async:	Asynchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	IccomDataPtr	LpData	Pointer to data buffer location allocated by application SWC != NULL_PTR.
	uint32	LulSize	1..<CTA partition size>
Parameters InOut:	None	None	None
Parameters Out:	None	None	None
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	RTE_E_OK RTE_E_INVALID RTE_E_LIMIT	
Description:	This function is the C implementation of Runnable Entity Ch<n>SendRun of ICCOM Complex Device Driver Component. This function performs send operation to send data to other domain.		
Configuration Dependency:	Dependency of **CddIccomChannelId** parameter: - This function only available if CddIccomChannelId is configured correctly. - <n> indicator is specified by CddIccomChannelId configured value. Dependency of **CddIccomChannelCtaPartitionSize** parameter: - Maximum value range of LulSize parameter is the value configured in CddIccomChannelCtaPartitionSize.		
Preconditions:	The ICCOM Complex Device Driver Component must be initialized. CddIccomChannel must be configured with proper CddIccomChannelId.		

11.5.3.2 **CddIccom_Ch<n>ReceiveRun**

Name:	CddIccom_Ch<n>ReceiveRun (<n> = 0..3)		
Prototype:	FUNC(Std_ReturnType, CddIccom_CODE) CddIccom_Ch<n>ReceiveRun (IccomDataPtr LpData, uint32 LulSize)		
Service Id:	0x0B + <n> (<n> = 0..3)		
Sync/Async:	Asynchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint32	LulSize	1..<CTA partition size>
Parameters InOut:	None	None	None
Parameters Out:	IccomDataPtr	LpData	Pointer to data buffer location allocated by application SWC != NULL_PTR.
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	RTE_E_OK, RTE_E_INVALID, RTE_E_NO_DATA.	
Description:	This function is the C implementation of Runnable Entity Ch<n>ReceiveRun of ICCOM Complex Device Driver Component. This function performs receive operation to receive data from other domain.		
Configuration Dependency:	Dependency of **CddIccomChannelId** parameter: - This function only available if CddIccomChannelId is configured correctly. - <n> indicator is specified by CddIccomChannelId configured value. Dependency of **CddIccomChannelCtaPartitionSize** parameter: - Maximum value range of LulSize parameter is the value configured in CddIccomChannelCtaPartitionSize.		
Preconditions:	The ICCOM Complex Device Driver Component must be initialized. CddIccomChannel must be configured with proper CddIccomChannelId.		

11.5.3.3 **CddIccom_MainFunction_Send**

Name:	CddIccom_MainFunction_Send		
Prototype:	FUNC(void, CDDICCOM_CODE_SLOW) CddIccom_MainFunction_Send(void)		
Service Id:	0x02		
Sync/Async:	Synchronous in normal operation. Asynchronous in case timeout detected.		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	None	None
Parameters InOut:	None	None	None
Parameters Out:	None	None	None
Return Value:	**Type**	**Possible Return Values**	
	void	None	
Description:	This function performs periodic check and re-transmission of any pending request.		
Configuration Dependency:	No dependency.		
Preconditions:	The ICCOM Complex Device Driver Component must be initialized. This function must be called periodically.		

11.5.3.4 **CddIccom_Ch<n>NoticeCallback**

Name:	CddIccom_Ch<n>NoticeCallback (<n> = 0..3)		
Prototype:	FUNC(void, CddIccom_CODE) CddIccom_Ch<n>NoticeCallback (IccomNoticeType LenType, uint32 LulMsg)		
Service Id:	None		
Sync/Async:	None (User can implement their own Callback function.)		
Reentrancy:	Non-Reentrant (User can implement their own Callback function.)		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	IccomNoticeType	LenType	ICCOM_NOTICE_DATA, ICCOM_NOTICE_FATAL
	uint32	LulMsg	Store the message from CA side. 1..<CTA partition size>
Parameters InOut:	None	None	None
Parameters Out:	None	None	None
Return Value:	**Type**	**Possible Return Values**	
	void	None	
Description:	This function is the C implementation of Runnable Entity Ch<n>NoticeCallback of ICCOM Complex Device Driver Component. This callback function invokes RTE APIs to send notification to application SWC via Ch<n>Notice sender port.		
Configuration Dependency:	Dependency of **CddIccomChannelId** parameter: - This function only available if CddIccomChannelId is configured correctly. - <n> indicator is specified by CddIccomChannelId configured value. Dependency of **CddIccomChannelNotification** parameter: - The value of parameter CddIccomChannelNotification shall be same with this function name.		
Preconditions:	The ICCOM Complex Device Driver Component must be initialized. CddIccomChannel must be configured with proper CddIccomChannelId. CddIccomChannelNotification parameter value is configured with this function name.		

11.5.4 Preemption of APIs

The **Table 11.15** shows list of Preemption Table of APIs of the ICCOM Driver
-: cannot be invoked at the same time
√: can be invoked at the same time

Table 11.15 Preemption Table of APIs of the ICCOM Driver

	CddIccom_Init	CddIccom_GetVersionInfo	CddIccom_Ch<n>SendRun (<n> = 0..3)	CddIccom_Ch<n>ReceiveRun (<n> = 0..3)	CddIccom_MainFunction_Send
CddIccom_Init	-				
CddIccom_GetVersionInfo	√	√	√	√	√
CddIccom_Ch<n>SendRun (<n> = 0..3)	-	√	√	√	√
CddIccom_Ch<n>ReceiveRun (<n> = 0..3)	-	√	√	√	√
CddIccom_MainFunction_Send	-	√	√	√	

11.6 Development And Production Errors

In this chapter the development errors that are reported by the ICCOM Complex Device Driver Component are tabulated. The development errors will be reported only when the pre-compiler option *CddIccomDevErrorDetect* is enabled in the configuration. The production code errors are not supported by ICCOM Complex Device Driver Component.

11.6.1 ICCOM Complex Device Driver Component Development Errors

The following table contains the DET errors that are reported by ICCOM Complex Device Driver Component. These errors are reported to DET Module when the ICCOM Complex Device Driver Component APIs is invoked with wrong input parameters or without initialization of the driver.

Table 11.16 DET Errors of ICCOM Complex Device Driver Component

Sl. No.	1
Error Code	CDDICCOM_E_UNINIT
Value (hex)	0x0A
Related API(s)	CddIccom_Ch<n>SendRun, CddIccom_Ch<n>ReceiveRun (<n> = 0..3)
Source of Error	When the API service is invoked before initialization.
Sl. No.	**2**
Error Code	CDDICCOM_E_ALREADY_INITIALIZED
Value (hex)	0x0D
Related API(s)	CddIccom_Init
Source of Error	When CddIccom_Init is invoked while ICCOM Complex Device Driver already initialized.
Sl. No.	**3**
Error Code	CDDICCOM_E_PARAM_VALUE
Value (hex)	0x15
Related API(s)	CddIccom_Ch<n>SendRun, CddIccom_Ch<n>ReceiveRun (<n> = 0..3)
Source of Error	When the API service is invoked with invalid value parameter.
Sl. No.	**4**
Error Code	CDDICCOM_E_PARAM_POINTER
Value (hex)	0x16
Related API(s)	CddIccom_Init, CddIccom_GetVersionInfo, CddIccom_Ch<n>SendRun, CddIccom_Ch<n>ReceiveRun (<n> = 0..3)
Source of Error	When the API service is invoked with Invalid pointer/Address.
Sl. No.	**5**
Error Code	CDDICCOM_E_INVALID_DATABASE
Value (hex)	0xEF
Related API(s)	CddIccom_Init
Source of Error	When the API service is invoked with invalid database address

11.6.2 ICCOM Complex Device Driver Component Production Errors

The following table contains Renesas specific DEM errors that are reported by ICCOM Complex Device Driver Component.

Table 11.17 DEM Errors of ICCOM Complex Device Driver Component

Sl. No.	1
Error Code	CDDICCOM_E_FATAL
Related API(s)	None.
Source of Error	Monitors the fatal errors detected of ICCOM Complex Device Driver. CDDICCOM fatal errors including: CTA Data read/write failure, invalid notification. • DEM_EVENT_STATUS_FAILED: When fatal error occurred.
Sl. No.	**2**
Error Code	CDDICCOM_E_INIT_NEGOTIATION
Related API(s)	None.
Source of Error	Monitors the initial negotiation notification from another domain. • DEM_EVENT_STATUS_FAILED: When invalid initial negotiation notification from other domain.
Sl. No.	**3**
Error Code	CDDICCOM_E_TIMEOUT
Related API(s)	CddIccom_Init, CddIccom_MainFunction_Send.
Source of Error	Monitors the send operation status. • DEM_EVENT_STATUS_FAILED: When reception data domain does not acknowledge to data sent or does not respond incase initialization driver CR52 domain within configure timeout duration.
Sl. No.	**4**
Error Code	CDDICCOM_E_WRITE_VERIFY_FAILURE
Related API(s)	CddIccom_Init, CddIccom_Ch<n>SendRun, CddIccom_Ch<n>ReceiveRun. (<n> = 0..3)
Source of Error	Monitors register write failure. • DEM_EVENT_STATUS_FAILED: When register read-back value does not match with written value.
Sl. No.	**5**
Error Code	CDDICCOM_E_INVALID_ACK
Related API(s)	None.
Source of Error	Monitors received invalid ACK. • DEM_EVENT_STATUS_FAILED: When ICCOM complex device driver received an invalid ACK.
Sl. No.	**6**
Error Code	CDDICCOM_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	None.
Source of Error	Monitors the Interrupt Unintended failure. • DEM_EVENT_STATUS_FAILED: When ICCOM complex device driver received an Unintended Interrupt Check Failure.

11.7 ICCOM Driver Component Runtime Errors

AUTOSAR does not define any runtime error code for ICCOM Driver.

11.8 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of ICCOM Complex Device Driver Component software.

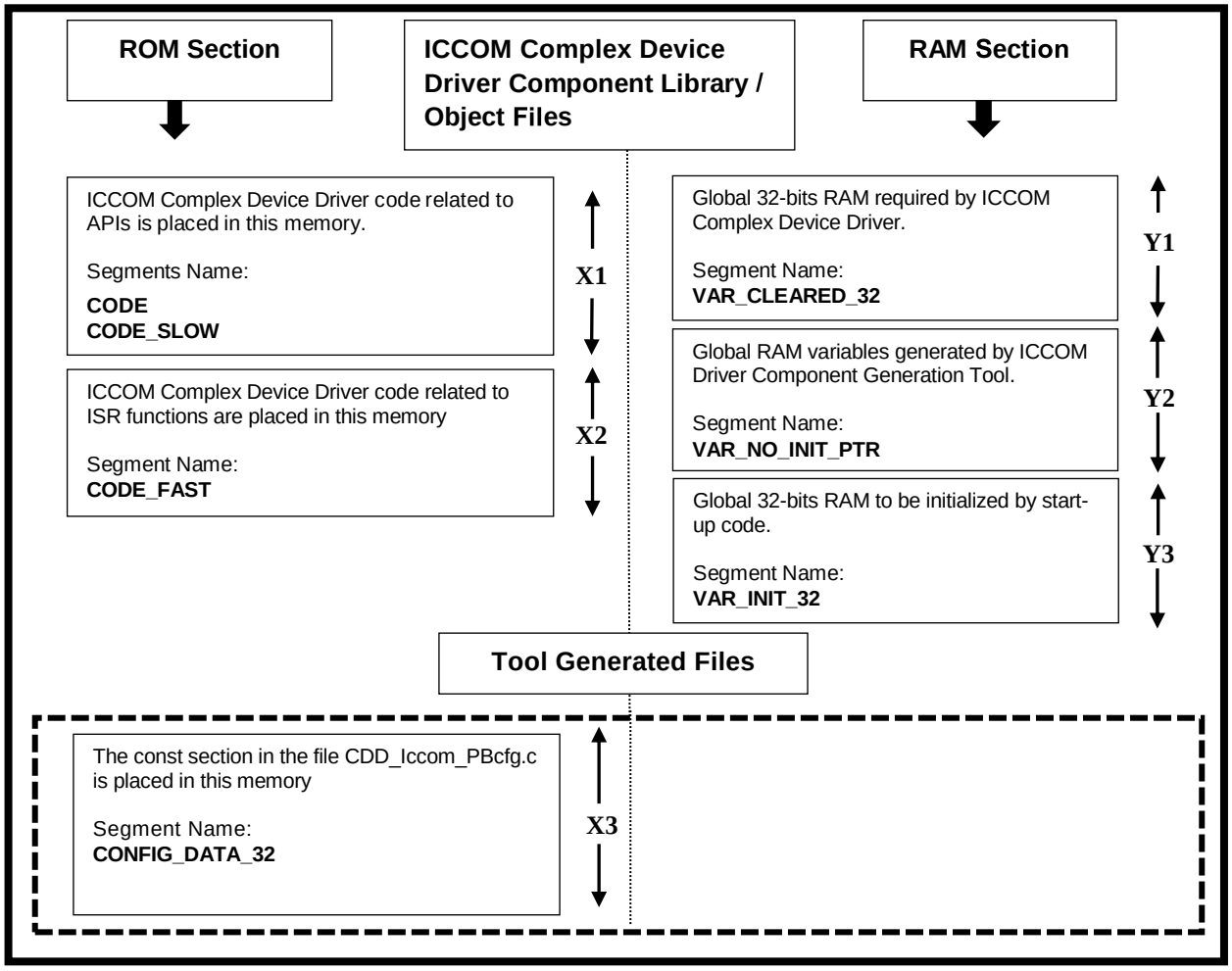


Figure 11.6 ICCOM Complex Device Driver Component Memory Organization

ROM Section (X1, X2 and X3):

CODE_SLOW, CODE (X1): API(s) of ICCOM Complex Device Driver Component except for ISR functions.

CODE_FAST (X2): This section consists of MFIS ISR functions that can be located in code memory.

CONFIG_DATA_32 (X3): This section consists of ICCOM Complex Device Driver Component controller specific configuration constant structures generated by ICCOM Complex Device Driver Component Generation Tool. This can be located in the code memory.

Note: ROM in R-Car is the Read-only Section in memory (DRAM).

RAM Section (Y1, Y2 and Y3):

VAR_CLEARED_32 (Y1): This section consists of the global RAM variables of 32-bits size that are used internally by ICCOM Complex Device Driver Component. This can be located in data memory.

VAR_NO_INIT_PTR (Y2): This section consists of the global RAM variables generated by ICCOM Driver Component Generation Tool. This can be located in data memory.

VAR_INIT_32 (Y3): This section consists of the global RAM variables of 32-bits size that are used internally by ICCOM Complex Device Driver Component. This can be located in data memory.

Note: User must ensure that none of the memory areas overlap with each other. Even 'debug' information should not overlap.

11.9 Device Specific Information

The device supports the following devices:

Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* – section 5.1 Product.

11.9.1 Interaction Between The User And ICCOM Complex Device Driver Component

The detail of the services supported by the ICCOM Complex Device Driver Component to the upper layers users is provided in the following chapter:

11.9.1.1 Channel Mapping

Table 11.18 Hardware CDDICCOM channel mapping

SI. No	Hardware channel
1	AP_MFIS0 AP_MFIS1 AP_MFIS2 AP_MFIS3

11.9.1.2 ISR Functions

The table below provides the list of handlers corresponding to the hardware unit ISR(s) in ICCOM Complex Device Driver Component. The user should configure the ISR functions mentioned below:

Table 11.19 ISR Handler Addresses

Interrupt Source	Name of the ISR Function (<n>=0..3)
MFIS	MFIS_xIICR<n>_ISR
	MFIS_xIICR<n>_CAT2_ISR

11.9.2 Multi-Core / Multi-Instantiation

ICCOM driver does not support multi-core and multi-instantiation for this device.

11.10 Non-AUTOSAR environment integration

The ICCOM Complex Driver Components for Renesas R-Car Gen4 SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the ICCOM Complex Driver Components to Non-AUTOSAR environment.

11.10.1 Stub modules handling

11.10.1.1 Os

The Os stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar version>\Os

In the AUTOSAR environment, ICCOM Complex Driver Components uses GetCounterValue and GetElapsedValue APIs provided by the OS module to get the current count value of the specified counter. The API prototype is as of follow:

StatusType GetCounterValue(CounterType CounterID, TickRefType Value)

Current Os stub implementation simply increases the Value each time the GetCounterValue API is invoked.

Non-AUTOSAR users can modify the provided GetCounterValue and GetElapsedValue APIs as needed e.g. with hardware counter or unique CounterID for each MCAL module.

11.10.1.2 Det

The Det stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, ICCOM Complex Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g. ICCOM Complex Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:

Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

11.10.1.3 Dem

The Dem stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, ICCOM Complex Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error. The API prototype is as of follow:

Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

11.10.2 Callback function usage

The ICCOM Complex Driver Component has a notification callback function being used to notify data/error to application SWC.

The notification callback function which will be configured in the parameter “CddIccomChannelNotification”. The configured value of CddIccomChannelNotification parameter must follow the following naming convention:

CddIccom_Ch<n>NoticeCallback

Where <n> is the channel number configured in parameter CddIccomChannelId within same CddIccomChannel container.

11.10.3 Scheduled function usage

In the AUTOSAR environment, ICCOM Complex Driver Component scheduled functions shall be invoked periodically by the Scheduler Manager implemented by EcuM module (Refer to *“AUTOSAR_SWS_ECUStateManager.pdf”*). The ICCOM Complex Driver Component scheduled functions are listed below:

CddIccom_MainFunction_Send

Non AUTOSAR user can invoke above API periodically using their own implementation e.g. hardware timer to utilize those APIs functionality.

11.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:

\rel\V4H\common_family\include\arm\Interrupt_VectorTable.h

\rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

Non-AUTOSAR users shall use MFIS_xIICR<Channel Number>_ISR ISR as specified in section 11.9.1.2 ISR Functions

12.MCU

12.1 Overview

The purpose of this chapter is to describe the information related to MCU Driver Component for Renesas microcontrollers.

This chapter shall be used as reference by the users of MCU Driver Component. The system overview of complete AUTOSAR architecture is shown in the Figure below:

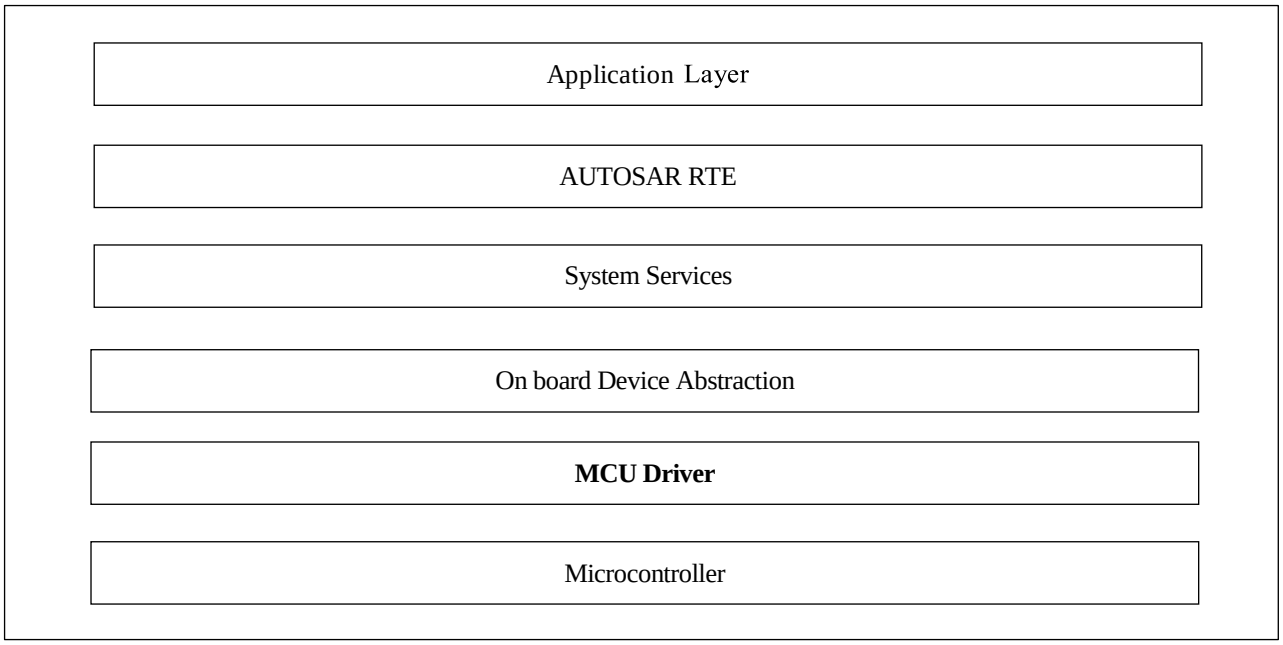


Figure 12.1 System Overview of AUTOSAR Architecture

The MCU Driver Component is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.

The Figure below depicts the MCU Driver as part of layered AUTOSAR MCAL Layer:

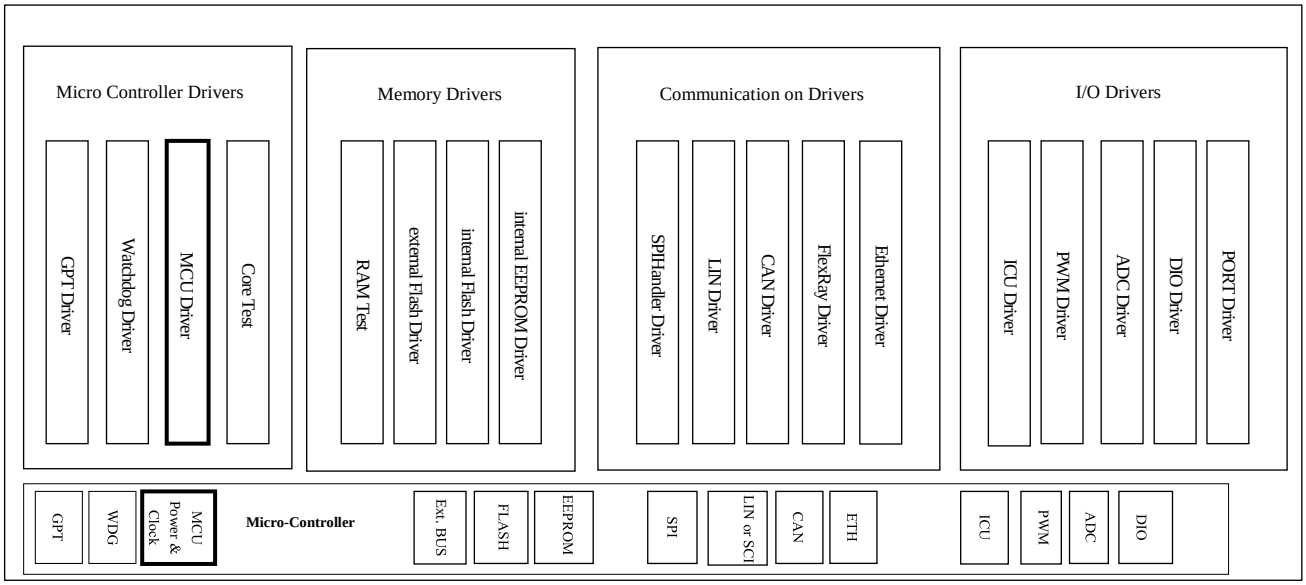


Figure 12.2 System Overview of The MCU Driver in AUTOSAR MCAL Layer

The MCU Driver Component provides services for basic microcontroller Initialization, power down functionality, reset functionality and microcontroller-specific functions required by other SPAL components.

This chapter describes the common features of MCU Driver Component. The diagram above shows the MCU Driver Component specification as defined in AUTOSAR. The AUTOSAR standardizes the functionality and API for the MCU driver Component, considering the scalability to different vehicle and platform variants.

This chapter is intended for the developers of ECU software using Application Programming Interfaces provided by AUTOSAR.

The MCU Driver Component comprises two sections, Embedded Software and the Generation Tool, to achieve scalability and configurability.

The MCU Driver Component Code Generation Tool is a command line tool that accepts ECU configuration description files as input, and generates C source and C header files. The configuration description is an ARXML file that contains information about the configuration for RAM Initialization, clock setting and power down mode setting.

The tool generates Mcu_Cfg.h, and Mcu_PBcfg.c.

12.2 Forethoughts

12.2.1 General

The following information will help the user use the MCU Driver Component software efficiently:

- The MCU Driver Component does not enable or disable the ECU or Microcontroller power supply. The upper layer should handle this operation.
- The MCU Driver does not enable or disable the Interrupts. The upper layer should handle this operation related to Interrupt control registers.
- The start-up code is ECU and MCU-specific. MCU Driver Component does not implement the start-up code.
- MCU specific initializations such as reset registers, one-time writable registers, interrupt stack pointer, user stack pointer and MCU internal watchdog, MCU-specific features of internal memory and registers are not implemented by MCU Driver Component. These initializations should be implemented by the start-up code.
- MCU Driver Component does not implement scheduled functions.
- MCU Driver Component is implemented as a post-build variant.
- The value configured for the parameter 'McuRamSectionBaseAddress' and 'McuRamSectionSize' in the container 'McuRamSectorSettingConf' should be aligned with each target RAM area.
- The parameter McuRamSectionWriteSize must be selected as "4" when RAM ECC is enabled; otherwise, an ECC error may be detected. To initialize RAM section in which the bit length is 8-bit or 16-bit access, the RAM ECC must be disabled.
- To use API Mcu_DistributePllClock, as the setting for peripheral clocks, external clock output and on-chip clock monitor are also supported by API Mcu_DistributePllClock as the additional features, if any of these features are used, API Mcu_DistributePllClock needs to be invoked (including the case that Internal OSC clock is selected as clock source of system clock).
- The parameter McuClockStabilityWaitingTime is used to generate the maximum count to satisfy all stabilization time among CLOCKs and PLLs after writing corresponding control registers. User must consider the following factors to configure to an appropriate value:
  - It should be big enough to cover the worst-case scenario in the driver configuration.
  - Compiler optimization level.
  - It is recommended to add additional margin to the timeout based on user experience.

  Since this is mere busy loop converted from the configured CPU clock, it is not guaranteed that the timeout occurs at the configured value.
- Since all McuPll<n>ClockSetting containers are mandatory, if initialization sequence use PLL for setting, the application should be setting as the same.
- In MCU module power save mode, MCU driver supports transition into Sleep mode. In Sleep mode, CPU executes "WFI" instruction. Clock to CPU is stopped to save the power consumption until CPU receives a wake-up request.
- Because of the limitation of hardware, the API Mcu_PerformReset does not support internal reset by using the hardware feature of the microcontroller. External reset via MCU_RESET_CALLOUT should be used.
- MCU driver setting of the PLL2VCO coordinates to the Normal mode. The settings can be implied for the High Performance mode despite the significant frequency differentiation. Refer to *"[3] R-Car V4H Series User's Manual: Hardware"* 8.1.5.2 PLL2 Multiplication Ratio.
- The ZB3φ clock range in the MCU Driver was calculated based on LPDDR5 setting, in which the maximum value is equal to 800 MHz. In case LPDDR4X is used, the ZB3φ clock is converted automatically by the setting of MD27:22 and MD19:17. Refer to *"[3] R-Car V4H Series User's Manual: Hardware"* 8.1.5.3 PLL3 Multiplication Ratio.
- MCU driver does not handle RAM errors, so the API Mcu_GetRamState always return VALID RAM status.

- User should choose McuClockDivider a reasonable divider to ensure that the value of McuZGClk does not exceed 600MHz.
- To make sure the output clock from the PLL5 stable, the parameters in the McuPll5ClockSetting container must be configured following **Table 12.1** or **Table 12.2** depend on the value of McuMainOsc parameter. In addition, the value of all parameters of the McuPllStopConditions container in the McuPll5ClockSetting container must be set as False (Default value).

Table 12.1 McuPll5ClockSetting limitation(1/2)

Container	Parameters	Value
McuPll5ClockSetting	McuMainOsc	CLOCK_FREQUENCY_16_66_MHZ or CLOCK_FREQUENCY_33_33_MHZ
	McuFreqDitherMode	INTEGER_FIXED_FREQUENCY_MODE_ID_0
	McuPllFrequency	3198720000
	McuMultiplicationRatio	95
	McuPllCircuitEnable	True

Table 12.2 McuPll5ClockSetting limitation(2/2)

Container	Parameters	Value
McuPll5ClockSetting	McuMainOsc	CLOCK_FREQUENCY_20_MHZ
	McuFreqDitherMode	INTEGER_FIXED_FREQUENCY_MODE_ID_0
	McuPllFrequency	3200000000
	McuMultiplicationRatio	79
	McuPllCircuitEnable	True

- All parameters in **Table 12.3** for the Module Stop Control Register must be always configured as True (Default value).

Table 12.3 McuModuleClockSupplySetting limitation

Container	Parameters	Value
McuModuleClockSupply Setting	McuINTAPClockSupplyEnable	True
	McuINTTPClockSupplyEnable	True
	McuIRQCClockSupplyEnable	True
	McuVDSP1MSTPCR28b31ClockSupplyEnable	True
	McuVDSP0MSTPCR28b28ClockSupplyEnable	True
	McuVDSP0MSTPCR28b27ClockSupplyEnable	True
	McuVDSP0MSTPCR28b26ClockSupplyEnable	True

	McuVDSP0MSTPCR28b25ClockSupplyEnable	True
	McuVDSP0MSTPCR28b24ClockSupplyEnable	True
	McuVDSP0MSTPCR28b23ClockSupplyEnable	True
	McuVDSP0MSTPCR28b22ClockSupplyEnable	True
	McuPAPBUSClockSupplyEnable	True

- All parameters in **Table 12.4** for the Module Stop Control Register must be always configured as False.

Table 12.4 McuModuleClockSupplySetting limitation

Container	Parameters	Value
McuModuleClockSupplySetting	McuVDSP3MSTPCR29b23ClockSupplyEnable	False
	McuVDSP3MSTPCR29b22ClockSupplyEnable	False
	McuVDSP3MSTPCR29b21ClockSupplyEnable	False
	McuVDSP3MSTPCR29b20ClockSupplyEnable	False
	McuVDSP3MSTPCR29b19ClockSupplyEnable	False
	McuVDSP3MSTPCR29b18ClockSupplyEnable	False
	McuVDSP3MSTPCR29b17ClockSupplyEnable	False
	McuVDSP2MSTPCR29b14ClockSupplyEnable	False
	McuVDSP2MSTPCR29b13ClockSupplyEnable	False
	McuVDSP2MSTPCR29b12ClockSupplyEnable	False
	McuVDSP2MSTPCR29b11ClockSupplyEnable	False
	McuVDSP2MSTPCR29b10ClockSupplyEnable	False
	McuVDSP2MSTPCR29b09ClockSupplyEnable	False
	McuVDSP2MSTPCR29b08ClockSupplyEnable	False
	McuVDSP1MSTPCR29b05ClockSupplyEnable	False
	McuVDSP1MSTPCR29b04ClockSupplyEnable	False
	McuVDSP1MSTPCR29b03ClockSupplyEnable	False
	McuVDSP1MSTPCR29b02ClockSupplyEnable	False
	McuVDSP1MSTPCR29b01ClockSupplyEnable	False
	McuVDSP1MSTPCR29b00ClockSupplyEnable	False

12.2.2 Preconditions

The following preconditions must be adhered by the user, for proper functioning of the MCU Driver Component:

- The Mcu_Cfg.h and Mcu_PBcfg.c files generated by the MCU Driver Component Code Generation Tool must be compiled and linked with the MCU Driver component source files.
- The application must be rebuilt, if there is any change in the Mcu_Cfg.h file generated by the MCU Driver Component Code Generation Tool.
- The Mcu_PBcfg.c generated for single configuration set using MCU Driver Component Code Generation Tool can be compiled and linked independently.
- The authorization of the user to call the software triggering of a hardware reset is not checked in the MCU Driver Component. This is the responsibility of the upper layer.
- The MCU Driver Component needs to be initialized before accepting any request. The API Mcu_Init should be called by the ECU State Manager Module to initialize MCU Driver Component.
- The user should ensure that MCU Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated only when the static configuration parameter 'MCU_DEV_ERROR_DETECT' is enabled. Application should ensure that the right parameters are passed while invoking the APIs when MCU_DEV_ERROR_DETECT is disabled.
- If the handle of clock setting passed to the API Mcu_InitClock is not configured to any one of the supported clock settings, then the Development Error Detection function is invoked if the static configuration parameter MCU_DEV_ERROR_DETECT is enabled.
- The MCU Driver Component initializes the clock generator as per the required configuration settings and provides the configured clock sources for the peripherals as needed. It is the responsibility of the individual drivers to select and initialize the respective driver-specific registers required for their functionality with reference to the clock source provided by the MCU Driver Component.
- The API Mcu_InitClock is implemented considering its invocation at run time. Hence, there is a possibility of change in the baud rate set by the peripheral drivers if the clock setting is different. Hence, the initialization of the respective drivers after the invocation of Mcu_InitClock, is the responsibility of the user of MCU Driver Component services.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used.
- The containers 'Mcu<ResetReasonName>Conf' and the parameter 'McuResetReason' in each container are used to represent for each reset type referred by EcuMResetReason parameters in the ECU State manager. The containers 'Mcu<ResetReasonName>Conf' are mandatory.
- Before the API Mcu_Init function call, all registers are expected to be the value after reset.
- The parameter "McuDomainId" is used to set the bus domain (0, 1, 2, and 3). If the parameter is set to a domain other than 0, Users must execute the Bus Domain Protection Registers setting to ensure that the target registers can be accessed. For the details of Bus Domain Protection, please refer to *"[3] R-Car V4H Series User's Manual: Hardware"* 8.4.2 Bus Access Protection for CPG registers, 9.4.2 Bus Access Protect for Module Standby and Software Reset registers, 14.4.1 Bus Access Protection for Reset (RST) registers.

12.2.3 Data Consistency

To support the re-entrance and interrupt services, the MCU Driver Component will ensure the data consistency while accessing its own Stand-by Control register or RAM storage or shared hardware registers. The MCU Driver Component will use SchM_Enter_Mcu_<Exclusive Area> and SchM_Exit_Mcu_<Exclusive Area> functions. The SchM_Enter_Mcu_<Exclusive Area> function is called before the data needs to be protected,

and SchM_Exit_Mcu_<Exclusive Area> function after the data is accessed. The flowchart indicates the flow with the precompile option "McuCriticalSectionProtection" enabled. The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

MCU_RAM_DATA_PROTECTION

It is used to exclude access to the global variable and the module-specific register.
It does not prohibit interrupt, but it prevents more than one context stored in the same critical section at the same time.
For the OS existence case, GetResource can be used.
If there is no OS and CAT1 ISR is used, it is necessary to disable interrupt.

MCU_INTERRUPT_CONTROL_PROTECTION

It is used to disable interrupt and task dispatch.
It is used when doing read modify write for shared resource register and disabling interruption such as changing the interrupt mask during peripheral operation.
The functions SchM_Enter_Mcu_<Exclusive Area> and SchM_Exit_Mcu_<Exclusive Area> can be disabled by disabling the configuration parameter 'McuCriticalSectionProtection'. MCU Module will use the following macros:

```
#define MCU_ENTER_CRITICAL_SECTION(Exclusive_Area) \
SchM_Enter_Mcu_##Exclusive_Area()

#define MCU_EXIT_CRITICAL_SECTION(Exclusive_Area) \
SchM_Exit_Mcu_##Exclusive_Area()
```

12.2.4 Callout API

The MCU_RESET_CALLOUT() API is the callout API from the MCU module that will be called by the Mcu_PerformReset() API for the software reset when configuration parameter 'McuSwResetCall' Api is true. The callout API needs to be filled by user to reset the software. If the configuration parameter 'McuSwResetCall' Api is false, the callout shall not be available and the software reset shall be handled by the MCU itself using HW feature of the SW reset.

12.2.5 EI Level Interrupt Conflict with ADC Driver

None.

12.2.6 FE Level Interrupt Conflict with WDG Driver

None.

12.2.7 Deviation List

The Table 12.5 shows the MCU Driver Deviation List.

Table 12.5 MCU Driver Deviation List

Sl. No.	Description	AUTOSAR Bugzilla
1.	The parameter ‘McuResetSetting’ (deviated from ECUC_Mcu_00173) in the sub-container ‘McuModuleConfiguration’ is not considered.	-
2.	The parameters ‘McuRamSectors’/ ’McuNumberOfMcuModes’/‘McuClockSrcFailureNotification’ (deviated from ECUC_Mcu_00172/ECUC_Mcu_00171/ECUC_Mcu_00170) and sub- container. ‘McuRamSectorSettingConf’ in the container ‘McuModuleConfiguration’ are not considered.	-
3.	The sub-container ‘McuClockReferencePoint’ (deviated from ECUC_Mcu_00174) in the Clock setting configuration is not used, as the reference frequencies specific to various peripheral devices need to be published by MCU Driver component.	-
4.	The parameter ‘McuClockReferencePointFrequency’ (devidated from ECUC_Mcu_00175) in the Clock setting configuration is not used, as the reference frequencies specific to various peripheral devices need to be published by MCU Driver component.	-
5.	Renesas MCU Driver does not follow SWS_Mcu_00126 regarding to Mcu_Init API does not support NULL pointer as a parameter, but reports DET.	-
6.	Renesas MCU driver does not follow the TPS_ECUC_08012, TPS_ECUC_08016 and TPS_ECUC_08015 AUTOSAR requirements regarding support for Different Post-build variants. It should be configured only one POST-BUILD variant on a parameter.	-
7.	Renesas MCU driver doesn't follow ECUC_Mcu_00185 and ECUC_Mcu_00186 Autosar requirements. The standard 'McuResetReasonConf' container is replaced by many vendor specific containers and parameters for each reset reason.	-
8.	Mcu driver does not support Mcu_PerformReset - internal reset (SWS_Mcu_00160, SWS_Mcu_00143, SWS_Mcu_00144, SWS_Mcu_00145, SWS_Mcu_00146) because the microcontroller does not support the software reset feature. User should use an external reset to reset system via MCU_RESET_CALLOUT().	-
9.	In the hard power-on case, Mcu_GetResetReson (SWS_Mcu_00134, SWS_Mcu_00005) shall return MCU_NONE_RESET, not MCU_POWER_ON_RESET.	-

12.3 Architecture Details

The MCU Driver Component architecture is shown in the following **Figure 12.3**.

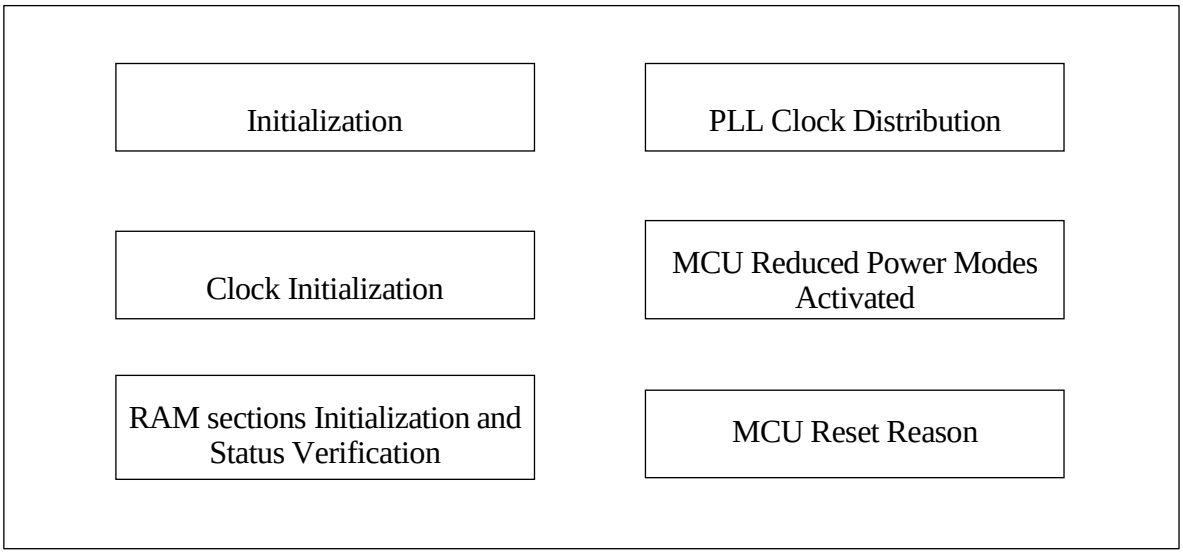


Figure 12.3 MCU Driver Architecture

The MCU Driver Component accesses the microcontroller hardware directly and is in the MCAL. It provides the functionalities related to PLL Initialization, Clock Initialization and Distribution, RAM sections Initialization, MCU Reduced Power Modes Activation, and MCU Reset Activation and Reason.

The MCU Driver Component consists of the following sub modules based on the functionality:

- Initialization
- Clock Initialization
- PLL Clock Distribution
- RAM sections Initialization and Status Verification
- MCU Reset Reason
- MCU Reset Activation
- MCU Reduced Power Modes Activated
- Version Information

Initialization

This sub module provides the structures and APIs for both global and controller-specific initialization. MCU-specific initialization is required to ensure different startup behaviors of the microcontroller. It also checks if the data base is flashed.

Clock Initialization

The clock initialization sub module provides the functionality to generate all the required clock signals for microcontroller operation from any one of the available sources. It allows selecting the individual clock source for the groups of CPU and peripherals.

RAM sections Initialization

This sub module provides the functionality to initialize the RAM with the any given value, at the selected blocks of the RAM.

MCU Reset Activation and Reason

The microcontroller reset activation will be performed by forcing a software reset. This functionality will be done by using software reset register.

To provide the reset reason, this sub module captures the information available with Reset flag/factor register. These registers contain information about the type of resets that has occurred.

MCU Reduced Power Modes Activated

This sub module provides the functionality for reduced power modes.

Version Information

This module provides APIs to read Module Id, Vendor Id and vendor-specific version numbers.

12.4 MCU Driver Component Header and Source File Description

This section explains the MCU Driver Component's C Source and C Header files. These files must be included in the project application while integrating with other modules.

The MCU Driver Component C header files:
Refer to "[1] AUTOSAR MCAL User's Manual Modules Overview" 3.3.5.3 Folder Structure.

The MCU Driver Component source files:
Refer to "[1] AUTOSAR MCAL User's Manual Modules Overview" 3.3.5.3 Folder Structure.

The C header file generated by MCU Driver Component Code Generation Tool:
Refer to "[1] AUTOSAR MCAL User's Manual Modules Overview" 3.3.5.3 Folder Structure.

The Stub C header files and source file:
Refer to "[1] AUTOSAR MCAL User's Manual Modules Overview" 3.2.9 Stubs File.

12.5 Application Programming Interface

This section explains the Data types and APIs provided by the MCU Driver Component to the Upper layers.

12.5.1 Imported Types

This section explains the Data types imported by the MCU Driver Component and lists its dependency on other modules.

12.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed:

- Std_ReturnType
- Std_VersionInfoType

12.5.1.2 Other Module Types

In this section, all types included in the Dem_types.h are listed:

- Dem_EventIdType
- Dem_EventStatusType

12.5.2 Type Definitions

This section explains the type definitions of MCU Driver Component according to AUTOSAR Specification.

12.5.2.1 Mcu_ClockType

The Table 12.6 shows explanation of Mcu_ClockType.

Table 12.6 Mcu_ClockType

Name:	Mcu_ClockType
Type:	uint8
Range:	0 to 1
Description:	Type definition for Mcu_ClockType used by the API Mcu_InitClock.

12.5.2.2 Mcu_RawResetType

The Table 12.7 shows explanation of Mcu_RawResetType.

Table 12.7 Mcu_RawResetType

Name:	Mcu_RawResetType
Type:	uint32
Range:	0 to 0x0000000F
Description:	Type definition for Mcu_RawResetType used by the API Mcu_GetResetRawValue.

12.5.2.3 Mcu_RamSectionType

The Table 12.8 shows explanation of Mcu_RamSectionType.

Table 12.8 Mcu_RamSectionType

Name:	Mcu_RamSectionType
Type:	uint8
Range:	0 to 255
Description:	Type definition for Mcu_RamSectionType used by the API Mcu_InitRamSection.

12.5.2.4 Mcu_ModeType

The Table 12.9 shows explanation of Mcu_ModeType.

Table 12.9 Mcu_ModeType

Name:	Mcu_ModeType
Type:	uint8
Range:	0 to 255
Description:	Type definition for Mcu_ModeType used by the API Mcu_SetMode.

12.5.2.5 Mcu_PllStatusType

The Table 12.10 shows explanation of Mcu_PllStatusType.

Table 12.10 Mcu_PllStatusType

Name:	Mcu_PllStatusType	
Type:	Enumeration	
Range:	MCU_PLL_LOCKED	PLL is locked
	MCU_PLL_UNLOCKED	PLL is unlocked.
	MCU_PLL_STATUS_UNDEFINED	PLL status is unknown
Description:	Status value returned by the API Mcu_GetPllStatus.	

12.5.2.6 **Mcu_RamStateType**

The Table 12.11 shows explanation of Mcu_RamStateType.

Table 12.11 Mcu_RamStateType

Name:	Mcu_RamStateType	
Type:	Enumeration	
Range:	MCU_RAMSTATE_INVALID	RAM State is invalid.
	MCU_RAMSTATE_VALID	RAM State is valid.
Description:	Status value returned by the API Mcu_GetRamState.	

12.5.2.7 **Mcu_ResetType**

The Table 12.12 shows the explanation of Mcu_ResetType.

Table 12.12 Mcu_ResetType

Name:	Mcu_ResetType		
Type:	Enumeration		
Range:	**Reset factor**	**ID**	**Description**
	MCU_POWER_ON_RESET	0	Power On reset
	MCU_RWDT_RESET	1	RWDT Reset
	MCU_SWDT_RESET	2	SWDT Reset
	MCU_SW_RESET	3	Soft Power on Reset (Application domain)
	MCU_MULTI_RESET	4	Multiple Reset reasons
	MCU_NONE_RESET	5	Reset is not occurred
	MCU_RESET_UNDEFINED	6	Reset is undefined
	MCU_RESERVED_RESET	7	Reserved Reset reason
Description:	Type of reset reasons supported by the hardware		

12.5.2.8 **Mcu_ConfigType**

The Table 12.13 shows the explanation of Mcu_ConfigType.

Table 12.13 Mcu_ConfigType

Location:	Mcu_Types.h	
Element:	uint32 ulStartOfDbToc	start of database table of contents
	uint8 ucNofClockSettings	The number of clock settings configured
	uint8 ucNofRamSettings	The number of ram settings configured
	uint8 ucNofModeSettings	The number of ram settings configured
	P2CONST(void, AUTOMATIC, MCU_CONFIG_CONST) pClockSetting	pointer to Clock setting configuration
	P2CONST(void, TYPEDEF, MCU_CONFIG_DATA) pRamSetting	Pointer to MCU Ram Setting configuration
	P2CONST(void, TYPEDEF, MCU_CONFIG_DATA) pModeSetting	Pointer to MCU Mode Setting configuration
	P2CONST(void, TYPEDEF, MCU_CONFIG_DATA) pClockHwInfo	Pointer to HW Information of clock controller
	P2CONST(void, TYPEDEF, MCU_CONFIG_DATA) pRstHwInfo	Pointer to HW Information of reset controller
	P2CONST(void, TYPEDEF, MCU_CONFIG_DATA) pHwFunc	Pointer to HW dependent functions of all HW IPs
Description:	This is the type of data structure containing the initialization data for the MCU driver.	

12.5.3 Function Definitions

This section mentions the APIs provided by the MCU Driver Component.
The Table 12.14 shows list of APIs and callback functions.

Table 12.14 APIs Provided by the MCU Driver Component

Sl. No.	API's name
AUTOSAR API	
1.	Mcu_Init
2.	Mcu_InitRamSection
3.	Mcu_InitClock
4.	Mcu_DistributePllClock
5.	Mcu_GetPllStatus
6.	Mcu_GetResetReason
7.	Mcu_GetResetRawValue
8.	Mcu_PerformReset
9.	Mcu_GetVersionInfo
10.	Mcu_GetRamState
11.	Mcu_SetMode

12.5.4 Preemption of APIs

The **Table 12.15** specifies the preemption of each API that can be invoked at the same time with the API.

Table 12.15 Preemption Table of APIs of the MCU Driver

	Mcu_Init	Mcu_InitRamSection	Mcu_InitClock	Mcu_GetPllStatus	Mcu_GetResetReason	Mcu_GetResetRawValue	Mcu_PerformReset	Mcu_GetRamState	Mcu_GetVersionInfo	Mcu_DistributePllClock	Mcu_SetMode
Mcu_Init	-										
Mcu_InitRamSection	-	-									
Mcu_InitClock	-	-	-								
Mcu_GetPllStatus	-	√	√	√							
Mcu_GetResetReason	-	√	√	√	√						
Mcu_GetResetRawValue	-	√	√	√	√	√					
Mcu_PerformReset	-	-	-	√	√	√	-				
Mcu_GetRamState	-	-	√	√	√	√	√	√			
Mcu_GetVersionInfo	√	√	√	√	√	√	√	√	√		
Mcu_DistributePllClock	-	-	-	√	√	√	-	√	√	-	
Mcu_SetMode	-	-	-	√	√	√	-	√	√	√	-

-: cannot be invoked at the same time

√: can be invoked at the same time

12.6 Development and Production Errors

In this section, the development errors reported by the MCU Driver Component are tabulated. The development errors will be reported only when the pre-compiler option 'McuDevErrorDetect' is enabled in the configuration. The production code errors are not supported by MCU Driver Component.

12.6.1 MCU Driver Component Development Errors

The **Table 12.16** and **Table 12.17** contain the DET errors reported by MCU Driver Component. These errors are reported to Development Error Tracer Module when the MCU Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 12.16 DET Errors of MCU Driver Component (1/2)

Sl. No.	**1**
Error Code	MCU_E_INIT_FAILED
Value (hex)	0x11
Related API(s)	Mcu_Init
Source of Error	When ConfigPtr is NULL pointer or the (hardware specific) contents of the given configuration is not within the allowed boundaries.
Sl. No.	**2**
Error Code	MCU_E_INVALID_DATABASE
Value (hex)	0xEF
Related API(s)	Mcu_Init
Source of Error	When the API is invoked with no database.
Sl. No.	**3**
Error Code	MCU_E_UNINIT
Value (hex)	0x0F
Related API(s)	Mcu_InitRamSection, Mcu_InitClock, Mcu_GetPllStatus, Mcu_GetResetReason, Mcu_GetResetRawValue, Mcu_PerformReset, Mcu_GetRamState
Source of Error	When the APIs are invoked without the initialization of the MCU Driver Component.
Sl. No.	**4**
Error Code	MCU_E_PARAM_RAMSECTION
Value (hex)	0x0D
Related API(s)	Mcu_InitRamSection
Source of Error	When RamSection is not within the sections defined in the configuration data structure.
Sl. No.	**5**
Error Code	MCU_E_PARAM_CLOCK
Value (hex)	0x0B
Related API(s)	Mcu_InitClock
Source of Error	When ClockSetting is not within the settings defined in the configuration data structure.

Table 12.17 DET Errors of MCU Driver Component (2/2)

Sl. No.	**6**
Error Code	MCU_E_PARAM_POINTER
Value (hex)	0x10
Related API(s)	Mcu_GetVersionInfo
Source of Error	When the above-mentioned API is invoked with Null Pointer versioninfo.
Sl. No.	**7**
Error Code	MCU_E_PARAM_MODE
Value (hex)	0x0C
Related API(s)	Mcu_SetMode
Source of Error	When McuMode is not within the settings defined in the configuration data structure.
Sl. No.	**8**
Error Code	MCU_E_PLL_NOT_LOCKED
Value (hex)	0x0E
Related API(s)	Mcu_DistributePllClock
Source of Error	When the above-mentioned API is invoked but PLL is not locked.

12.6.2 MCU Driver Component Production Errors

The **Table 12.18** shows the list of DEM errors identified in the MCU Driver component. MCU Driver component reports these errors to DEM by invoking Dem_ReportErrorStatus API. This API is invoked when the processing of the given API request fails.

Table 12.18 DEM Errors Generated by MCU Driver Component

Sl. No.	**1**
Error Code	MCU_E_CLOCK_FAILURE
Related API(s)	Mcu_InitClock
Source of Error	When the clock is not stable and the time-out occurs after setting clock controller registers.

12.7 MCU Driver Component Runtime Errors

None.

12.8 Memory Organization

The following **Table 12.19** depicts a typical memory organization, which must be met for proper functioning of MCU Driver Component software.

Table 12.19 ROM / RAM Sections of the MCU Driver

Section Name	Alignment*1	Description
MCU_PUBLIC_CODE_ROM	-	This section contains codes which belong to the API functions of the MCU Driver.
MCU_PRIVATE_CODE_ROM	-	This section contains codes which belong to the internal functions of the MCU Driver.
CONST_UNSPECIFIED	-	This section contains miscellaneous constants of the MCU Driver.
CONFIG_DATA_UNSPECIFIED	-	This section contains post-build config data of the MCU Driver.
CONFIG_DBTOC_DATA_UNSPECIFIED	-	This section contains post-build config data of the MCU Driver.
CONFIG_DATA_8	-	This section contains post-build config 8bit data table of the MCU Mode setting.
CONFIG_DATA_32	-	This section contains post-build config 32bit data table of the MCU Mode setting.
VAR_INIT_1	-	This section contains boolean variables which need to be initialized before Mcu_Init.
VAR_NO_INIT_32	-	This section contains 32bits variables which do not need to be initialized before Mcu_Init.
VAR_NO_INIT_PTR	-	This section contains the pointer variables which do not need to be initialized before Mcu_Init.

Note *1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

12.9 Device Specific Information

The device supports the following devices:

Refer to “[1] ASR MCAL User’s Manual Modules Overview” 5.1 Product.

12.9.1 Interaction between the User and MCU Driver Component

The following sections detail the services supported by the MCU Driver Component to the user upper layers and the mapping of the channels to the hardware units:

12.9.1.1 Channel Mapping

The channel setting does not depend on H/W resources.

12.9.2 Multi-Core / Multi-Instantiation

MCU driver does not support multi-core and multi-instantiation for this device.

12.10 Non-AUTOSAR environment integration

12.10.1 Stub Modules Handling

12.10.1.1 Det

In the AUTOSAR environment, MCU Driver Components uses Det_ReportError API provided by the DET module to report a development error, e.g. MCU Driver has not been initialized, API is provided with invalid parameter,…

The API prototype is as of follow:
Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

Example: refer Det.c, Det.h in “external\rel\common\generic\stubs\19_11\Det”.

12.10.1.2 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.

MCU driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when MCU_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON.

With Non-AUTOSAR environment, user needs to prepare SchM stub to user MCU driver as following:
void SchM_Enter_Mcu_MCU_INTERRUPT_CONTROL_PROTECTION (void)
void SchM_Exit_Mcu_MCU_INTERRUPT_CONTROL_PROTECTION (void)

Example: refer SchM_Mcu.h, SchM_Mcu.c in “external\rel\common\generic\stubs\19_11\Rte\”

12.10.1.3 Dem

In the AUTOSAR environment, MCU Driver Components uses Dem_ReportErrorStatus API provided by the DEM module to report a production.

The API prototype is as of follow:
Dem_ReportErrorStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_ReportErrorStatus API with their current error handling strategy.

Example: refer Dem.h, Dem.c in “external\rel\common\generic\stubs\19_11\Dem\”

12.10.2 Callback Function Usage

The MCU Driver Component does not provide any callback functions.

12.10.3 Scheduled Function Usage

The MCU Driver Component does not provide any scheduled functions.

12.10.4 Interrupt handling usage

The MCU Driver Component does not provide any interrupt handling functions.

13.IIC

13.1 Overview

The purpose of this document is to describe the information related to IIC Complex Driver Component.

This document is intended for the developers of ECU software on R-Car Series, 4th Generation SoC using Application Programming Interfaces provided by AUTOSAR specification for IIC Complex Driver. The IIC Complex Driver Component provides the following services:

- IIC Complex Driver Component initialization.
- Perform transmission operation in Master/Slave mode.
- Perform reception operation in Master/Slave mode.
- Notify back to application layer.
- Return the version information of IIC module.
- Perform slave initialization.
- Perform transmission-reception operation in master mode.

The IIC Complex Driver Component comprises of two parts: Embedded Software and Generation Tool to achieve scalability and configurability.

The purpose of this document is to describe the information related to Embedded Software part of the IIC Complex Driver Component for R-Car Series, 4th Generation SoC. Please refer to Chapter 6 (IIC) in "*R-Car V4H AUTOSAR R19-11 MCAL Generation Tool User's Manual"* for the detail of Generation Tool part.

The below figure depicts the IIC Complex Device Driver as part of layered AUTOSAR MCAL Layer:

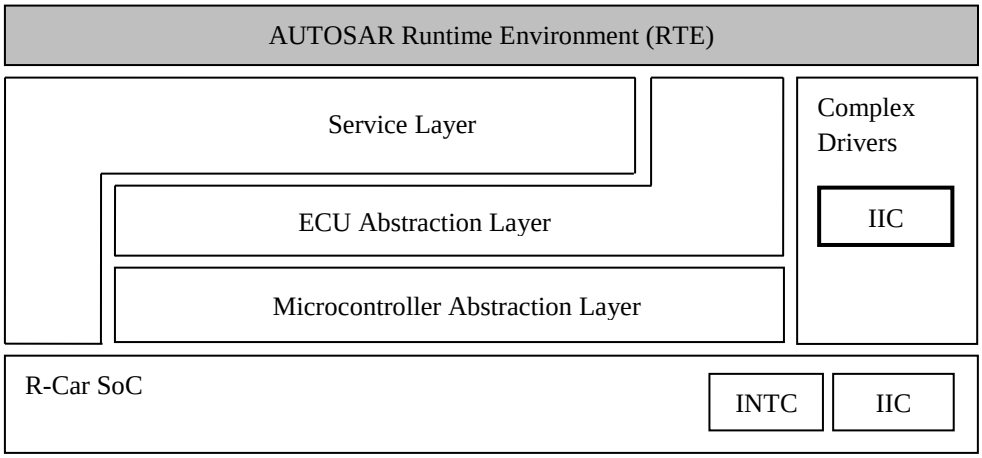


Figure 13-1 System Overview of the IIC Complex Device Driver in AUTOSAR Layer Architecture

The following diagram shows the system overview of the AUTOSAR Architecture for IIC Complex Driver :

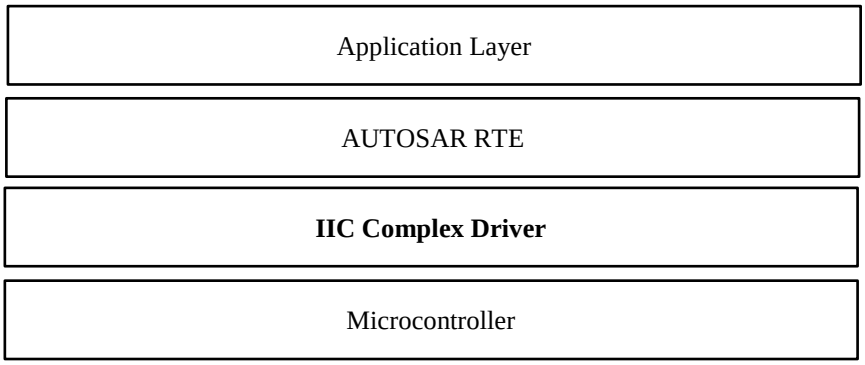


Figure 13-2 System Overview of AUTOSAR Architecture

The IIC CDD Component is Vendor Specific Module provided by Renesas to communicate with devices connected through IIC buses with R-Car SoC.

Sample Application is available for user to understand and test IIC Complex Device Driver without proper AUTOSAR upper layer, which is described in section *"Sample Application"* of this document.

Please refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* - section 3.7.2.2 How to build the Sample Application for more information on how to configure, compile and flash the IIC Complex Driver to R-Car SoC.

13.2 Forethoughts

13.2.1 General

Following information will aid the user to use the IIC Complex Driver Component software efficiently:

- IIC Complex Driver is used to interface with devices connected to IIC bus. Complex Driver requires proper devices that support IIC connected on the bus.
- MCU specific initializations such as reset registers, one-time writable registers, interrupt stack pointer, user stack pointer, internal watchdog, specific features of internal memory and registers is not implemented by IIC Complex Driver. These initializations must be implemented by the start-up code.
- The ISR functions and the corresponding handler addresses are provided in section *"ISR Functions"* of this document. User must ensure that Interrupt Vector table configuration is done as per the information provided in the table.
- Porting specific information such as compiler details, timing details and memory consumption are provided in in chapter *"Device Specific Information"* of this document.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- If any development errors are reported to DET, the normal flow of execution of driver will be aborted.
- All production errors will be reported to DEM by using the API Dem_SetEventStatus provided by DEM.
- All interrupt-related functions are mapped to CODE_FAST section (see Table 7.12 of *AUTOSAR Specification of Memory Mapping*). Other functions are mapped to CODE_SLOW section (see Table 7.13 of *AUTOSAR Specification of Memory Mapping*).
- The number of channels that are configured should be maintained in single post-build configuration set.

13.2.2 Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the IIC Complex Driver Component:

- The CDD_Iic_Cfg.h file generated by the IIC Complex Driver Component Code Generation Tool must be compiled and linked along with IIC Complex Driver Component source files.

- The application must be rebuilt, if there is any change in the CDD_Iic_Cfg.h files generated by the IIC Complex Driver Component Code Generation Tool.
- The CDD_Iic_Cbk.h file generated by the IIC Complex Driver Component Code Generation Tool must be compiled and linked along with IIC Complex Driver Component source files.
- CDD_Iic_PBcfg.c generated for single configuration set using IIC Complex Driver Component Code Generation Tool can be compiled and linked independently.
- The IIC Complex Driver Component needs to be initialized before accepting any request. The API CddIic_Init should be invoked to initialize IIC Complex Driver Component before calling any other IIC Complex Driver API.
- User should ensure that IIC Complex Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated only when the static configuration parameter CddIicDevErrorDetect is enabled. Application should ensure that the right parameters are passed while invoking the APIs when CddIicDevErrorDetect is disabled.
- The IIC Complex Driver states are checked only when the static configuration parameter CddIicDevErrorDetect is enabled. Application should ensure the recommended sequence of invoking APIs is satisfied when CddIicDevErrorDetect is disabled.
- IIC Complex Driver included maximum of 6 channels. Each channel shall be configurable to use a specific IIC hardware channel IICn "n" (0…5).
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used.
- IIC driver provides Register Write-Verify feature to verify the execution of control registers' write operation. After writing to a control register, value of that register is read back to make sure that register has been written correctly.
- Upper layer should have some timing protection mechanisms to ensure the performance of IIC Complex Driver.
- ISR priority should be configured to be greater than priority of the thread/task which invoke the driver's APIs. Otherwise, the driver operation is not guaranteed.
- Be sure to reset I2C with Software Reset Register SRCRn (n = 5 for V4H) at the beginning of transmission and reception procedure. A software reset can be set by controlling the Software Reset Register SRCRn (n = 5 for V4H) and Software Reset Clearing Register SRSTCLRn (n = 5 for V4H) of the "9. Module Standby, Software Reset" in R-CarV4H Hardware User's Manual.
  - Reset Sequence:

    Set the target I2C bit of SRCRn to 1. (**) → Wait 1 micro second → Set the target I2C bit of SRSTCRn to 1. (**)

    →Wait for the target bit of SRCSRCRn to 0. → End sequence

    **: The other bits should be written 0.
- IIC hardware channel 0, 1, 2, 3, 4, 5 use SYS-DMAC.
- In cache enable environment, user should invoke operation clear/invalidate cached when write/read data from memory. To avoid cache coherency problem when L1 data cache of Cortex-R7 enable, user need to provide some functions for cache maintenance.
- The usage of IIC Complex Driver shall follow the general IIC specification.
- For slave interface, the number of data transmission/reception shall not exceed the maximum values which configured in CddIic_ChnSlaveInit.
- IIC Complex Driver clock communication could be configured by 2 modes: Fix Duty and Variable Duty

  In **Fix Duty** mode, SCL frequency is calculated by below equation:

  $$SCLfreq = I2Cck/(20 + SCGD \times 8 + F[(tICF + tr + IntDelay) \times I2Cck])$$

  - I2Cck: IIC internal clock frequency
  - tICF: IIC SCL falling time (depending on external load)
  - tr: I2C SCL rising time (depending on external load)
  - IntDelay: LSI internal delay corresponds to output buffer type.
    Open drain buffer: 50 ns (typical), 110 ns (maximum)
    LVTTL (low drive only) buffer: 5 ns (typical), 6 ns (maximum).
  - F[n]: n rounded down to an integer
  - The I2C internal clock frequency is 19 MHz

  In order to configure communication clock in Fix Duty mode, user shall input the SCLfreq (target

communication clock) by parameter CddIicClockFrequency, tICF by parameter CddIicFallingTime, tr by parameter CddIicRisingTime and IntDelay by parameter CddIicIntDelay. According to above equation, user can calculate the value of SCGD (SCL clock generation divider).

Note:

1. **The range of SCGD should be in range of [0…63]**
2. **The value of tICF, tr, IntDelay is different from specific device. User should measure again according to actually device provide the value for these parameters**
3. **In Fix Duty mode, the maximum frequency value of configuration parameter CddIicClockFrequency is ~900000Hz. The actual maximum *SCL* frequency is ~720000Hz. If want to get a higher communication frequency, please use Variable Duty mode by setting the configuration parameter CddIicClockModeSelection as "VARIABLE_DUTY" (refer detailed setting of "Variable Duty mode" as the below information)**

In **Variable Duty** mode, IIC communication clock is controlled by 2 factors: SCL high period time and SCL low period time. These factors could be configured by 2 parameters CddIicSclHighPeriod and CddIicSclLowPeriod, accordingly. The SCL frequency is calculated by the following equation:
SCL freq = MOD_CLK/(8 + 2 × SMD + SCLD + SCHD + F[(tICF + tr + IntDelay) × MOD_CLK])

- MOD_CLK: Mode Clock frequency (133.33 MHz)
- tICF: IIC SCL falling time (depending on external load)
- tr: I2C SCL rising time (depending on external load)
- IntDelay: LSI internal delay corresponds to output buffer type.
  Open drain buffer: 50 ns (typical), 110 ns (maximum)
  LVTTL (low drive only) buffer: 5 ns (typical), 6 ns (maximum).
- F[n]: n rounded down to an integer

User also shall input the value of rising time, falling time and internal delay by the way same with Fix Duty mode.

- The maximum transmission byte and maximum reception byte in Slave operation is also used to configure for DMAC transferring. User shall ensure that the number use for DMAC transfer is same as number of data byte which requested write/read from Master device
- User must initialize the following hardware IPs as below before initializing IIC module.

Table 13-1 List of hardware IP affect to CDDIIC

Hardware IP	Module Name	Expected Setting	Description
AXI-bus	-	SDRAM address mapping in HW should be configured according to Section 18.3.1 in [3] HW UM.	IIC is designed assuming the setting of SDRAM address mapping is corrected, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since IIC operation is under controlled by DRAM for memory control.
DBSC4	-	The maximum use of SDRAM bus bandwidth will be enabled by DBSC4 according to Section 33.3 in [3] HW UM.	IIC is designed assuming the configuration of SDRAM bus bandwidth is suitable, so they should be configured like that

			by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since IIC operation is under controlled by DRAM for memory control.
CPG, Module Standby, SW Reset	MCU	- **CPGWPCR**: Set/clear bit0 of this register enable/disable write protection of register in CPG. By default, this bit is set (enable write protection). - **SRSTCLR5**: Set bit Bit18, Bit19, Bit20, Bit21, Bit22, Bit23 (channel 0 to channel 5, accordingly) to 1 whenever releasing the reset for IIC, otherwise, keep it as 0 - **SRCR5**: Set Bit18, Bit19, Bit20, Bit21, Bit22, Bit23 (channel 0 to channel 5, accordingly) will reset IIC module.	This IP enable/disable clock signal of modules (see section 8 Clock Pulse Generator, 9 Module Standby, SW Reset of [3] HW UM. The setting of these register is provided in MCU Driver.
PFC	PORT	**- PMMR:** To access GPSR and IPSR register, user shall write inverse value of GPSR/IPSR's value to this register. **- GPSR8:** It is required to set bits 0, bit 1 (IIC0), bit 2, bit 3 (IIC1), bit 4, bit 5 (IIC2), bit6, bit7 (IIC3), bit8, bit9 (IIC4), bit10, bit11 (IIC5) in order to use according IIC channel. **- IP0SR8:** It is required to + Clear bit [3:0] for SCL0 (IIC0), [7:4] for SDA0(IIC0) + Clear bit [11:8] for SCL1 (IIC1), [15:12] for SDA1 (IIC1) + Clear bit [19:16] for SCL2 (IIC2), [23:20] for SDA2 (IIC2) + Clear bit [27:24] for SCL3 (IIC3), [21:28] for SDA3 (IIC3) **- IP1SR8:** It is required to + Clear bit [3:0] for SCL4 (IIC4), [7:4] for SDA4 (IIC4) + Clear bit [11:8] for SCL5 (IIC5), [15:12] for SDA5 (IIC5)	This IP set the function of multiplexed pins and controlling the pull-up on each LSI pin see section 7 Pin Function Controller of [3] HW UM. The setting of these register can be configured by Configuration Description File and APIs provided by PORT Driver.
IPMMU	IPMMU	When using, Address translation to processing units and interconnect networks should be set according to Section 19.3.4 in [3] HW UM.	SW module CDDIPMMU handle the address translation, please refer to User Manual of IPMMU for detail setting.

13.2.3 Data Consistency

To support the re-entrance and interrupt services, the IIC Complex Device Driver Component will ensure the data consistency while accessing its own RAM storage or hardware registers. The IIC Complex component will use SchM_Enter_CddIic_<Exclusive Area> and SchM_Exit_CddIic_<Exclusive Area> functions. The SchM_Enter_CddIic_<Exclusive Area> function is called before accessing the data needs to be protected and SchM_Exit_CddIic_<Exclusive Area> function is called after the data is accessed.

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

- CDDIIC_RAM_DATA_PROTECTION
- CDDIIC_INTERRUPT_CONTROL_PROTECTION

The functions SchM_Enter_CddIic_<Exclusive Area> and SchM_Exit_CddIic_<Exclusive Area> can be disabled by disabling the configuration parameter 'CddIicCriticalSectionProtection'. The flowchart indicates indicate the flow with the pre-compile option 'CddIicCriticalSectionProtection' enabled.

13.3 Architecture Details

The IIC Complex Driver architecture is shown in the following figure:

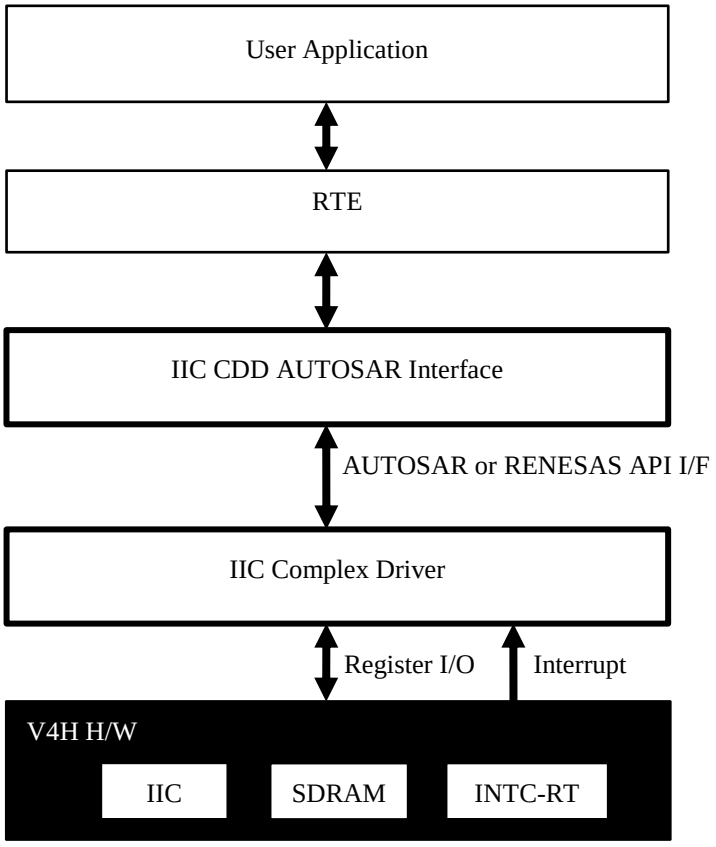


Figure 13-3 IIC Complex Device Driver Component Architecture

The IIC Complex Driver component can be divided into following substitute components based on the functions performed by the IIC Complex Driver:

- Driver Initialization
- Clock Configuration
- Send data to slave device (master mode)

- Receive data from slave device (master mode)
- Get Version Information

Driver Initialization

The driver initialization sub-module internally stores the configuration data address to enable subsequent API calls to access the configuration data and initializes the global variables used by IIC Complex Driver Component.
The API related to this sub-module is CddIic_Init.

Send data to slave device

The IIC complex driver provides a service to send data to slave device.
Following AUTOSAR interface are provided:

- Interface: If_IicWrite (Client-Server)
- Provided Port: P_Ch<n>Write (n = 0…5)
- Runnable Entity: CddIic_Ch<n>Write (n = 0...5)

Read data from slave device

The IIC complex driver provides a service to read data from slave device.
Following AUTOSAR interface are provided:

- Interface: If_IicRead (Client-Server)
- Provided Port: P_Ch<n>Read (n = 0…5)
- Runnable Entity: CddIic_Ch<n>Read (n = 0...5)

Write then Read data from slave device with repeat START

The IIC complex driver provides a service to send data to slave device.
Following AUTOSAR interface are provided:

- Interface: If_IicWriteRead (Client-Server)
- Provided Port: P_Ch<n>WriteRead (n = 0…5)
- Runnable Entity: CddIic_Ch<n>WriteRead (n = 0...5)

Slave initialization for IIC complex driver

The IIC complex driver provides a service to initialize for slave interface
Following AUTOSAR interface are provided:

- Interface: If_IicSlaveInit (Client-Server)
- Provided Port: P_Ch<n>SlaveInit (n = 0…5)
- Runnable Entity: CddIic_Ch<n>SlaveInit (n = 0...5)

Notify data received event to AUTOSAR SWC

The IIC complex driver provides a Sender-Receiver Port to notify data received event to AUTOSAR SWC.
Following AUTOSAR interface are provided:

- Interface: If_IicNotice (Sender-Receiver)
- Provided Port: P_ChnNotice (n = 0…5)

Retrieval Version Information

The IIC Complex Driver provides a service to return version information details to the User.
The API related to this sub-module is CddIic_GetVersionInfo.

13.4 IIC Complex Driver Component Header And Source File Description

This chapter explains the IIC Complex Driver Component’s source and header files. These files have to be

included in the project application while integrating with other modules.

The C header file generated by IIC Complex Driver Component Code Generation Tool:

- CDD_Iic_Cfg.h
- CDD_Iic_Cbk.h

The C source file generated by IIC Complex Driver Component Code Generation Tool:

- CDD_Iic_PBcfg.c

The IIC Complex Device Driver Component C header and Component C source files:
Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" – section 3.4.3.3 Folder Structure

The Stub C header files and source file :
Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" – section 3.2.9 Stubs File.

13.5 Application Programming Interface

This chapter explains the Data types and APIs provided by the IIC Complex Driver Component to the Upper layers.

13.5.1 Imported Types

This chapter explains the Data types imported by the IIC Complex Driver Component and lists its dependency on other modules.

13.5.1.1 Standard Types

In this chapter all types included from the Std_Types.h (according to "*AUTOSAR Specification of Standard Types"*) are listed:

- Std_ReturnType.
- Std_VersionInfoType.

13.5.1.2 Dem Types

In this chapter all types included from the Dem.h (according to "*AUTOSAR Specification of Diagnostic Event Manager"*) are listed:

- Dem_EventIdType.
- Dem_EventStatusType.

13.5.2 Type Definitions

This chapter explains the type definitions of IIC Complex Driver Component.

13.5.2.1 CddIic_ChannelType

Table 13-2 CddIic_ChannelType

Name:	CddIic_ChannelType	
Type:	typedef	
Range:	uint8	0…5
Description:	Represents the identifier of IIC channel.	

13.5.2.2 IicMessageType

Table 13-3 IicMessageType

Name:	IicMessageType	
Type:	uint8	
Range:	IIC_NOTICE_IDLE	IIC transmission/reception is idled
	IIC_NOTICE_REPEAT_START	IIC issue the repeat START to IIC bus
	IIC_NOTICE_NON_ACK	IIC data notification.
	IIC_NOTICE_END	IIC transmission/reception have been finished
	IIC_NOTICE_OVERRUN	IIC slave transmit/receive more than range of slave buffers
	IIC_NOTICE_DATA	IIC is transfer data
Description:	Enumeration for the IIC notice type.	

13.5.2.3 IicOperationType

Table 13-4 IicOperationType

Name:	IicOperationType	
Type:	uint8	
Range:	IIC_NOTICE_WRITE_OPERATION	IIC is on write operation
	IIC_NOTICE_READ_OPERATION	IIC is on read operation
Description:	Enumeration for the IIC notice type.	

13.5.2.4 CddIic_ChStaType

Table 13-5 CddIic_ChStaType

Name:	CddIic_ChStaType	
Type:	Enumeration	
Range:	CDDIIC_CH_SENDING	Channel is transmitting
	CDDIIC_CH_RECEIVING	Channel is receiving
	CDDIIC_CH_IDLE	Channel is ready to use.
	CDDIIC_CH_CONFIGURE	Channel is configuring for transmission/reception
Description:	Enumeration for the IIC channel state.	

13.5.2.5 CddIic_ChannelStatusType

Table 13-6 CddIic_ChannelStatusType

Name:	CddIic_ChannelStatusType	
Type:	Structure	
Elements:	VAR(volatile CddIic_ChStaType, TYPEDEF)	enChSta
	VAR(uint32, TYPEDEF)	ulSndByteNumber
	VAR(uint32, TYPEDEF)	ulSndByteSent
	VAR(uint32, TYPEDEF)	ulRcvByteNumber
	VAR(uint32, TYPEDEF)	ulRcvByteReceieved
	VAR(CddIic_AddressType, TYPEDEF)	enSlaveAddressMode
	VAR(CddIic_DmaStatusType, TYPEDEF)	enDmaStatus
	VAR(CddIic_RepeatStartStatus, TYPEDEF)	enRepeatStartSta
Description:	Identify the desired status information for the IIC channel.	

13.5.2.6 Cddlic_BufferAddressType

Table 13-7 CddIic_BufferAddressType

Name:	Cddlic_BufferAddressType	
Type:	Structure	
Elements:	P2CONST (uint8, TYPEDEF, CDDIIC_APPL_DATA)	pSndBuffer
	P2VAR(uint8, TYPEDEF, CDDIIC_APPL_DATA)	pRcvBuffer
Description:	Identify the send and receive buffer type.	

13.5.2.7 Cddlic_ClockModeType

Table 13-8 CddIic_ClockModeType

Name:	Cddlic_ClockModeType	
Type:	enumeration	
Range:	CDDIIC_FIXED_DUTY	Selected clock mode is fixed duty
	CDDIIC_VARIABLE_DUTY	Selected clock mode is variable duty
Description:	Identify the selected clock mode.	

13.5.2.8 Cddlic_AddressType

Table 13-9 CddIic_AddressType

Name:	Cddlic_AddressType	
Type:	Enumeration	
Range:	CDDIIC_SEVEN_BIT_ADDR	Limit length of slave address is 7-bit
	CDDIIC_TEN_BIT_ADDR	Limit length of slave address is 10-bit
Description:	Identify the slave address mode.	

13.5.2.9 Cddlic_OperationInterfaceType

Table 13-10 CddIic_OperationInterfaceType

Name:	Cddlic_OperationInterfaceType	
Type:	Enumeration	
Range:	CDDIIC_MASTER_IF	IIC operate in Master Interface
	CDDIIC_SLAVE_IF	IIC operate in Slave Interface
Description:	Identify the operation interface (master or slave)	

13.5.2.10 Cddlic_ChannelConfigType

Table 13-11 CddIic_ChannelConfigType

Name:	Cddlic_ChannelConfigType	
Type:	Structure	
Elements:	P2VAR(uint32, TYPEDEF, REGSPACE)	pICMCRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICMSRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICMIERnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICMARnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICTXRXDnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICSCRnReg

	P2VAR(uint32, TYPEDEF, REGSPACE)	pICSSRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICSIERnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICSARnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICCCRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICCCR2nReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICMPRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICHPRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICLPRnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICDMAERnReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pICFBSCRnReg
	P2VAR(CddIic_DmaConfigType, TYPEDEF, CDDIIC_DATA)	pDmaConfiguration
	P2FUNC(void, CDDIIC_APPL_CODE, pNotification)(IicNotification LstNoticInfo)	LstNoticInfo
	VAR(CddIic_OperationInterfaceType, TYPEDEF)	enInterface
	VAR(CddIic_DmaType, TYPEDEF)	enDmaMode
	P2FUNC(void, CDDIIC_APPL_CODE, pEnterRegProtect)(void)	pEnterRegProtect
	P2FUNC(void, CDDIIC_APPL_CODE, pExitRegProtect)(void)	pExitRegProtect
	P2FUNC(void, CDDIIC_APPL_CODE, pEnterGlbProtect)(void)	pEnterGlbProtect
	P2FUNC(void, CDDIIC_APPL_CODE, pExitGlbProtect)(void)	pExitGlbProtect
Description:	Identify the desired configuration for IIC channel.	

13.5.2.11 Cddlic_ConfigType

Table 13-12 CddIic_ConfigType

Name:	CddIic_ConfigType	
Type:	Structure	
Elements:	VAR(uint32, TYPEDEF)	ulStartOfDbToc
	P2CONST(void, TYPEDEF, CDDIIC_CONFIG_CONST)	pChannelConfig
	P2VAR(void, TYPEDEF, CDDIIC_CONFIG_DATA)	pChannelStatus
Description:	Identify the desired configuration for the IIC CDD.	

13.5.2.12 Cddlic_WriteType

Table 13-13 CddIic_WriteType

Name:	CddIic_WriteType	
Type:	Structure	
Elements:	P2CONST(uint8, TYPEDEF, CDDIIC_CODE)	pWriteBuff
	VAR(uint32, TYPEDEF)	ulWriteNumber
Description:	Identify the desired configuration for the IIC CDD.	

13.5.2.13 Cddlic_ReadType

Table 13-14 CddIic_ReadType

Name:	Cddlic_ReadType	
Type:	Structure	
Elements:	P2VAR (uint8, TYPEDEF, CDDIIC_CODE)	pReadBuff
	VAR(uint32, TYPEDEF)	ulReadNumber
Description:	Identify the desired configuration for the IIC CDD.	

13.5.2.14 Cddlic_DmaType

Table 13-15 CddIic_DmaType

Name:	Cddlic_DmaType	
Type:	Enumeration	
Range:	CDDIIC_DMA_ENABLED	DMAC have been enabled
	CDDIIC_DMA_DISABLED	DMAC have been disabled
Description:	Identify DMAC enabled/disabled	

13.5.2.15 Cddlic_SlaveInterfaceType

Table 13-16 CddIic_SlaveInterfaceType

Name:	Cddlic_SlaveInterfaceType	
Type:	Structure	
Elements:	P2VAR(uint32, TYPEDEF, RTE_APPL_DATA)	pTxBuff
	P2VAR(uint32, TYPEDEF, RTE_APPL_DATA)	pRxBuff
	VAR(uint32, TYPEDEF)	ulMaxTxByte
	VAR(uint32, TYPEDEF)	ulMaxRxByte
Description:	Identify the desired configuration for slave interface	

13.5.2.16 Cddlic_DmaConfigType

Table 13-17 CddIic_DmaConfigType

Name:	Cddlic_DmaConfigType	
Type:	Structure	
Elements:	P2VAR(uint32, TYPEDEF, REGSPACE)	pTxDmaBaseAddr
	P2VAR(uint32, TYPEDEF, REGSPACE)	pRxDmaBaseAddr
	VAR(uint8, TYPEDEF)	ucTxDmaMapping
	VAR(uint8, TYPEDEF)	ucRxDmaMapping
	Identify the configuration set of DMA	
Description:		

13.5.2.17 Cddlic_DmaModeType

Table 13-18 CddIic_DmaModeType

Name:	Cddlic_DmaModeType	
Type:	Enumeration	
Range:	CDDIIC_DMA_MODE_MASTER_TRANSMIT	Enable DMA mode master transmit for IIC

	CDDIIC_DMA_MODE_MASTER_RECEIVE	Enable DMA mode master receive for IIC
	CDDIIC_DMA_MODE_SLAVE_TRANSMIT	Enable DMA mode slave transmit for IIC
	CDDIIC_DMA_MODE_SLAVE_RECEIVE	Enable DMA mode slave receiver for IIC
Description:	Identify DMA mode type for IIC complex driver	

13.5.2.18 Cddlic_SlaveConfigType

Table 13-19 CddIic_SlaveConfigType

Name:	Cddlic_SlaveConfigType	
Type:	Structure	
Elements:	VAR(uint32, TYPEDEF)	ulSCLHighPeriod
	VAR(uint32, TYPEDEF)	ulSCLLowPeriod
	VAR(uint32, TYPEDEF)	ulTargetClockFreq
	VAR(uint32, TYPEDEF)	ulExtLoadTime
	VAR(uint16, TYPEDEF)	usSlaveAddress
	VAR(uint16, TYPEDEF)	ucSlaveID
	VAR(uint8, TYPEDEF)	ucFisrtBitSetupCycle
	VAR(uint8, TYPEDEF)	ucSclClkGenDiv
	VAR (Cddlic_ClockModeType, TYPEDEF)	enClockMode
	VAR(Cddlic_AddressType, TYPEDEF)	enAddressMode
Description:	Identify the configuration slave	

13.5.3 Function Definitions

Table 13-20 APIs provided by the IIC Complex Driver Component

Sl. No.	APIs
AUTOSAR API	
1	Cddlic_Init
2	Cddlic_GetVersionInfo
RENESAS API	
3	Cddlic_Ch<n>Write (<n>=0..5)
4	Cddlic_Ch<n>Read (<n>=0..5)
5	Cddlic_Ch<n>WriteRead (<n>=0..5)
6	Cddlic_Ch<n>SlaveInit (<n>=0..5)
7	Cddlic_Ch<n>NoticeCallback (<n> = 0..5)

13.5.3.1 Cddlic_Ch<n>Write

Name:	Cddlic_Ch<n>Write n = 0...5
Prototype:	FUNC(Std_ReturnType , Cddlic_CODE) Cddlic_ChnWrite (licSndDataPtr LpData, uint32 LulSndByteNumber, licSlaveConfigPtr LpSlaveConfig)

Service Id:	0x02 + <n> n = 0...5		
Sync/Async:	Sync		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	IicSndDataPtr	LpData	Valid address configured by SWC application.
	uint32	LulSndByteNumber	Number of data use for transmission
	IicSlaveConfigPtr	LpSlaveConfig	Valid address configured by SWC application.
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	RTE_E_OK RTE_E_INVALID RTE_E_COM_BUSY	
Description:	This function performs send operation to write data to slave via IIC channel n. n = 0...5		
Configuration Dependency:	Dependency of CddIicChannelId parameter: - This function only available if CddIicChannelId is configured correctly.		
Preconditions:	The IIC Complex Driver must be initialized. CddIicChannel must be configured with proper CddIicChannelId		

13.5.3.2 Cddlic_Ch<n>Read

Name:	Cddlic_Ch<n>Read n = 0...5		
Prototype:	FUNC(Std_ReturnType , Cddlic_CODE) Cddlic_ChnRead (IicRcvDataPtr LpData, uint32 LulRcvByteNumber, IicSlaveConfigPtr LpSlaveConfig)		
Service Id:	0x0A + <n> n = 0...5		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint32	LulRcvByteNumber	Number of data read from Slave
	IicSlaveConfigPtr	LpSlaveConfig	Valid address configured by SWC application.
Parameters InOut:	None		
Parameters out:	IicRcvDataPtr	LpData	Valid address configured by SWC application.
Return Value:	**Type**	**Possible Return Values**	

	Std_ReturnType	RTE_E_OK, RTE_E_COM_BUSY RTE_E_INVALID
Description:	This function performs receive operation to read data from slave via IIC channel n. n = 0...5	
Configuration Dependency:	Dependency of CddIicChannelId parameter: - This function only available if CddIicChannelId is configured correctly.	
Preconditions:	The IIC Complex Driver must be initialized. CddIicChannel must be configured with proper CddIicChannelId.	

13.5.3.3 Cddlic_Ch<n>WriteRead

Name:	Cddlic_Ch<n>WriteRead n = 0...5		
Prototype:	FUNC(Std_ReturnType , Cddlic_CODE) Cddlic_Ch<n>WriteRead (IicSndDataPtr LpWriteBuff, uint32 LulWriteNumber, IicRcvDataPtr LpReadBuff, uint32 LulReadNumber, IicSlaveConfigPtr LpSlaveConfig)		
Service Id:	0x30 + <n> n = 0...5		
Sync/Async:	Sync		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	IicSndDataPtr	LpWriteBuff	Valid address configured by SWC application.
	uint32	LulWriteNumber	Number of data use for transmission
	uint32	LulReadNumber	Number of data read from Slave
	IicSlaveConfigPtr	LpSlaveConfig	Valid address configured by SWC application.
Parameters InOut:	None	NA	NA
Parameters out:	IicRcvDataPtr	LpReadBuff	Valid address configured by SWC application.
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	RTE_E_OK RTE_E_INVALID RTE_E_COM_BUSY	
Description:	This function performs send and receive operation to write and read data to slave via IIC channel n. n = 0...5		
Configuration Dependency:	Dependency of CddIicChannelId parameter: - This function only available if CddIicChannelId is configured correctly.		

Preconditions:	The IIC Complex Driver must be initialized. CddIicChannel must be configured with proper CddIicChannelId

13.5.3.4 Cddlic_Ch<n>SlaveInit

Name:	Cddlic_Ch<n>SlaveInit n = 0...5		
Prototype:	FUNC(Std_ReturnType , Cddlic_CODE) Cddlic_ChnSlaveInit (uint8 LucSlaveOwnAdress, IicSlaveIfPtr LpSlaveInterface)		
Service Id:	0x20+ <n>		
Sync/Async:	Synchronous		
Reentrancy:	Non reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	LucSlaveOwnAdress	0…127
	IicSlaveIfPtr	LpSlaveInterface	Valid address configured by SWC application.
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	RTE_E_OK, RTE_E_INVALID	
Description:	This API perform slave initialization.		
Configuration Dependency:	Dependency of CddIicChannelId parameter: - This function only available if CddIicChannelId is configured correctly.		
Preconditions:	The IIC Complex Driver must be initialized.		

13.5.3.5 CddIic_Ch<n>NoticeCallback

Name:	CddIic_Ch<n>NoticeCallback (V4H: n = 0..5)
Prototype:	FUNC(void, CddIic_CODE) CddIic_ChnNoticeCallback (n = 0..5) (IicNotification LstNoticInfo)
Service Id:	None
Sync/Async:	Asynchronous (User can implement their own Callback function.)
Reentrancy:	Reentrant (User can implement their own Callback function.)
Parameters In:	The parameter is defined by IIC's user, it is out of scope of CDDIIC module
Parameters InOut:	The parameter is defined by IIC's user, it is out of scope of CDDIIC module
Parameters out:	The parameter is defined by IIC's user, it is out of scope of CDDIIC module

Return Value:	None
Description:	This function is the C implementation of Runnable Entity Ch<n>NoticeCallback of CddIic Complex Driver Component. This callback function invoke RTE APIs to send notification to application SWC via ChnNotice sender port.
Configuration Dependency:	Dependency of CddIicChannelId parameter: - This function only available if CddIicChannelId is configured correctly.
Preconditions:	The IIC Complex Driver must be initialized. CddIicChannel must be configured with proper CddIicChannelId. CddIicChannelNotification parameter value is configured with this function name.

13.5.4 Preemption of APIs

The following table specifies the preemption of each API that can be invoked at the same time as the API.

Table 13-21 Preemption Table of APIs of the IIC Driver

	Cddlic_Init	Cddlic_GetVersionInfo	Cddlic_Ch<n>Write (<n> = 0..5)	Cddlic_Ch<n>Read (<n> = 0..5)	Cddlic_Ch<n>WriteRead (<n> = 0..5)	Cddlic_Ch<n>SlaveInit (<n> = 0..5)
Cddlic_Init	-					
Cddlic_GetVersionInfo	✓	✓	✓	✓	✓	✓
Cddlic_Ch<n>Write (<n> = 0..5)	-	✓	-	✓	✓	-
Cddlic_Ch<n>Read (<n> = 0..5)	-	✓	✓	-	✓	-
Cddlic_Ch<n>WriteRead (<n> = 0..5)	-	✓	✓	✓	-	-
Cddlic_Ch<n>SlaveInit (<n> = 0..5)	-	✓	-	-	-	✓

✓: can be invoked at the same time
-: cannot be invoked at the same time

13.6 Development And Production Errors

In this chapter the development errors that are reported by the IIC Complex Driver Component are tabulated. The development errors will be reported only when the pre-compiler option *CddIicDevErrorDetect* is enabled in the configuration. The production code errors are not supported by IIC Complex Driver Component.

13.6.1 IIC Complex Device Driver Component Development Errors

The following table contains the DET errors that are reported by IIC Complex Driver Component. These errors are reported to DET Module when the IIC Complex Driver Component APIs is invoked with wrong input parameters or without initialization of the driver.

Table 13-22 DET Errors of IIC Complex Driver Component

Sl. No.	**1**
Error Code	CDDIIC_E_UNINITIALIZED
Value (hex)	0x14
Related API(s)	Cddlic_Ch<n>Write, Cddlic_Ch<n>Read, Cddlic_Ch<n>WriteRead, Cddlic_Ch<n>SlaveInit (<n> = 0...5)
Source of Error	When the API service is invoked before initialization.
Sl. No.	**2**
Error Code	CDDIIC_E_ALREADY_INITIALIZED
Value (hex)	0x15
Related API(s)	Cddlic_Init
Source of Error	When Cddlic_Init is invoked while IIC Complex Driver already initialized.
Sl. No.	**3**
Error Code	CDDIIC_E_PARAM_VALUE
Value (hex)	0x16
Related API(s)	Cddlic_Ch<n>Write, Cddlic_Ch<n>Read, Cddlic_Ch<n>WriteRead, Cddlic_Ch<n>SlaveInit

	<n> = 0…5
Source of Error	When the API service is invoked with invalid value parameter
Sl. No.	**4**
Error Code	CDDIIC_E_PARAM_POINTER
Value (hex)	0x17
Related API(s)	Cddlic_Ch<n>Write, Cddlic_Ch<n>Read, Cddlic_Ch<n>WriteRead, Cddlic_Ch<n>SlaveInit <n> = 0…5
Source of Error	When the API service is invoked with Invalid pointer/Address.
Sl. No.	**5**
Error Code	CDDIIC_E_INVALID_DATABASE
Value (hex)	0xEF
Related API(s)	Cddlic_Init
Source of Error	When the API service is invoked with Invalid database.

13.6.2 IIC Complex Device Driver Component Production Errors

The following table contains Renesas specific DEM errors that are reported by IIC Complex Driver Component.

Table 13-23 DEM Errors of IIC Complex Driver Component

Sl. No.	**1**
Error Code	CDDIIC_E_NON_ACKNOWLEGEMENT
Related API(s)	IIC_HW_CHn_ISR <n> = 0…5
Source of Error	Monitors the write and read operation status. • DEM_EVENT_STATUS_FAILED: When slave does not acknowledge to request of Master device.
Sl. No.	**2**
Error Code	CDDIIC_E_WRITE_VERIFY
Related API(s)	Cddlic_Init, Cddlic_Ch<n>Write, Cddlic_Ch<n>Read, Cddlic_Ch<n>WriteRead, Cddlic_Ch<n>SlaveInit <n> = 0…5
Source of Error	Monitors register write failure. • DEM_EVENT_STATUS_FAILED: When register read-back value does not match with written value.
Sl. No.	**3**
Error Code	CDDIIC_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	IIC_HW_CHn_ISR <n> = 0…5 CDDIIC_DMA<n>_ISR n = 00...31
Source of Error	Monitors register write failure. • DEM_EVENT_STATUS_FAILED: When IIC interrupt service routine was not triggered by interrupt source

13.7 IIC Driver Component Runtime Errors

AUTOSAR does not define any runtime error code for IIC Driver

13.8 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of IIC Complex Driver Component software.

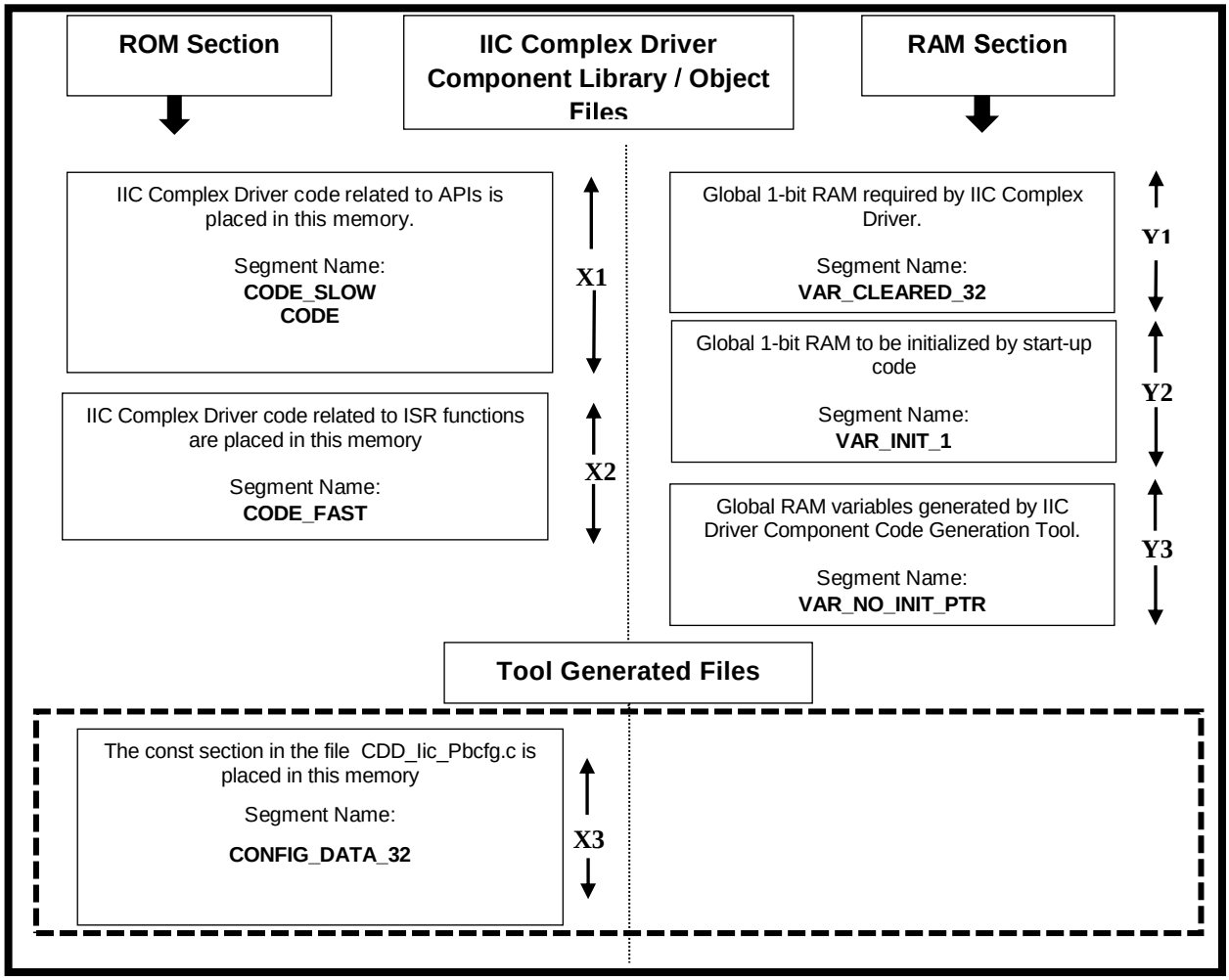


Figure 13-4 IIC Complex Driver Component Memory Organization

ROM Section (X1, X2 and X3):

CODE_SLOW, CODE (X1): API(s) of IIC Complex Device Driver Component except for ISR functions.

CODE_FAST (X2): This section consists of IIC ISR functions that can be located in code memory.

CONFIG_DATA_32 (X3): This section consists of IIC Complex Driver Component controller specific configuration constant structures generated by IIC Complex Driver Component Code Generation Tool. This can be located in the code memory.

Note: ROM in RCar-V4H is the Read-only Section in memory (DRAM).

RAM Section (Y1, Y2 and Y3):

VAR_CLEARED_32 (Y1): This section consists of the global RAM variables of 32-bit size that are used internally by IIC Complex Driver Component. This can be located in data memory.

VAR_INIT_1 (Y2): This section consists of the global RAM variables of 1-bit size that are used internally by IIC Complex Driver Component. This can be located in data memory.

VAR_NO_INIT_PTR (Y3): This section consists of the global RAM variables generated by IIC Driver

Component Code Generation Tool. This can be located in data memory.

Note: User must ensure that none of the memory areas overlap with each other. Even 'debug' information should not overlap.

13.9 Device Specific Information

The device supports the following devices:

Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* – section 5.1 Product.

13.9.1 Interaction Between The User And IIC Complex Device Driver Component

The detail of the services supported by the IIC Complex Driver Component to the upper layers users is provided in the following chapter:

13.9.1.1 Channel Mapping

Table 13-24 Hardware CDDIIC channel mapping

Sl. No	Hardware channel(<n>=0..5)
1	I2C<n>

13.9.1.2 ISR Functions

The table below provides the list of handlers corresponding to the hardware unit ISR(s) in IIC Complex Device Driver Component. The user should configure the ISR functions mentioned below:

Table 13-25 ISR Handler Addresses

Interrupt Source	Name of the ISR Function (<n>=0..5,<m>=00..31)
IIC	IIC_HW_CH<n>_ISR
	IIC_HW_CH<n>_ISR_CAT2_ISR
	CDDIIC_DMA<m>_ISR
	CDDIIC_DMA<m>_CAT2_ISR

13.9.2 Multi-Core / Multi-Instantiation

IIC driver does not support multi-core and multi-instantiation for this device.

13.10 Non-AUTOSAR environment integration

The IIC Complex Driver Components for Renesas R-Car Gen4 SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the IIC Complex Driver Components to Non-AUTOSAR environment.

13.10.1 Stub modules handling

13.10.1.1 Det

The Det stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, IIC Complex Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g. IIC Complex Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:

Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

13.10.1.2 Dem

The Dem stub files are organized in the following folder:

\rel\common\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, IIC Complex Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error. The API prototype is as of follow:

Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Currently, Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

13.10.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.

The SchM (Basic Software Scheduler) stub files are organized in the following folder:

RCar\common_platform\generic\stubs\<Autosar version>\Rte\

The IIC Complex Driver Component needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when *CddIicCriticalSectionProtection* parameter is configured as *true*.

With Non-AUTOSAR environment, user needs to prepare SchM stub to user IIC Complex Driver Component as following:

Write SchM functions with prototype as below:

void SchM_Enter_CddIic_CDDIIC_RAM_DATA_PROTECTION(void);

void SchM_Exit_CddIic_CDDIIC_RAM_DATA_PROTECTION(void);

void SchM_Enter_CddIic_CDDIIC_INTERRUPT_CONTROL_PROTECTION(void);

void SchM_Exit_CddIic_CDDIIC_INTERRUPT_CONTROL_PROTECTION(void);

13.10.2 Callback function usage

The IIC Complex Driver Component has a notification callback function being used to notify data/error to application SWC.

The notification callback function which will be configured in the parameter “CddIicChannelNotification”. The configured value of CddIicChannelNotification parameter must follow the following naming convention:

CddIic_Ch<n>NoticeCallBack

Where <n> is the channel number configured in parameter CddIicChannelId within same CddIicChannel container.

13.10.3 Scheduled function usage

The IIC Complex Driver Component does not have Scheduled function.

13.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:

\rel\V4H\common_family\include\arm\Interrupt_VectorTable.h

\rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

Non-AUTOSAR users shall use IIC_HW_CH<Channel Number>_ISR, CDDIIC_DMA< DMA Channel Number >_ISR. ISR as specified in section 13.9.1.2 ISR Function.

14.SPI

14.1 Overview

The purpose of this chapter is to describe the information related to SPI Driver Component.

This chapter shall be used as reference by the users of SPI Driver Component. The system overview of complete AUTOSAR architecture is shown in the **Figure 14.1**:

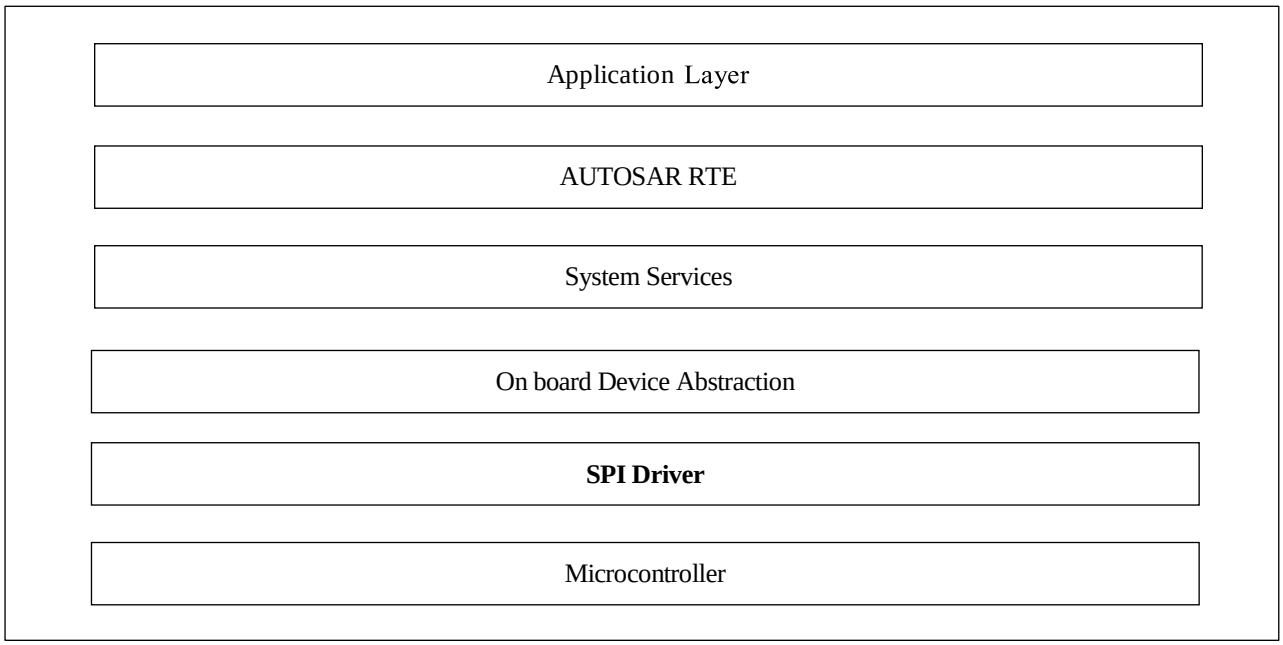


Figure 14.1 System Overview of AUTOSAR Architecture

The SPI Driver is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.
The

Figure 14.2 depicts the SPI Driver as part of layered AUTOSAR MCAL Layer:

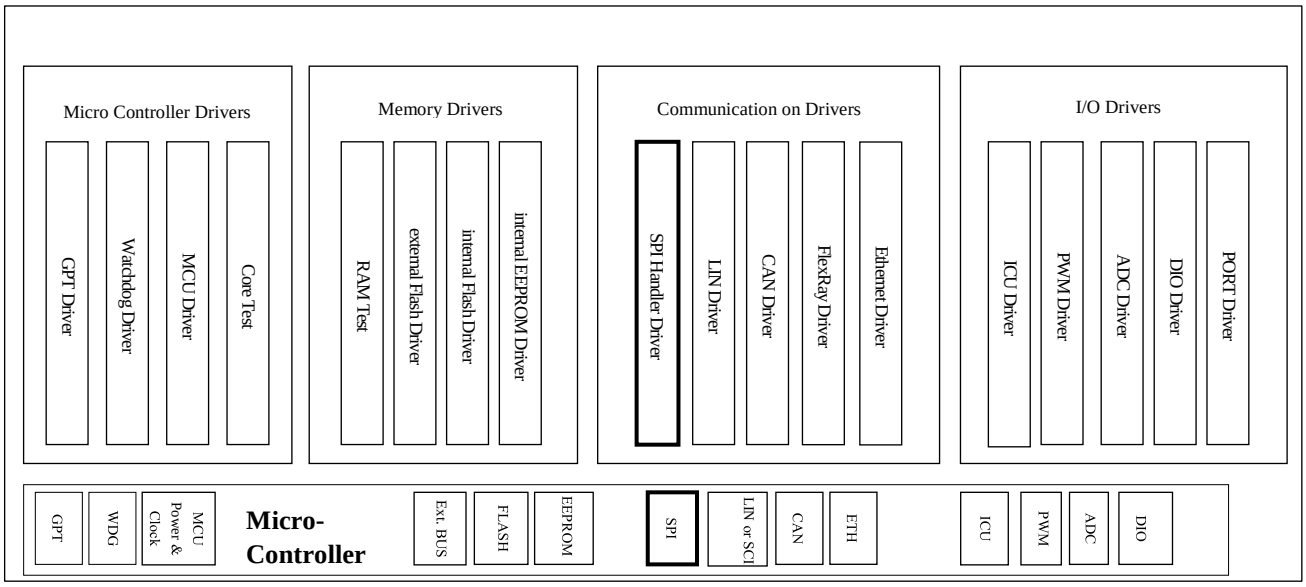


Figure 14.2 System Overview of the SPI Driver in AUTOSAR MCAL Layer

The SPI Driver Component comprises Embedded software and the Configuration Tool to achieve scalability and configurability.

The SPI Driver Component code Generation Tool is a command line tool that accepts ECU configuration description files as input, and generates source and header files. The configuration description is an ARXML file that contains information about the configuration for SPI Driver. The tool generates the Spi_PBcfg.c, Spi_Lcfg.c and Spi_Cfg.h.

The SPI Driver provides services for reading from and writing to devices connected through SPI buses. It provides access to SPI communication to several users (For example, EEPROM, I/O ASICs). It also provides the required mechanism to configure the on-chip SPI peripheral.

14.2 Forethoughts

14.2.1 General

The following information will help the user use the SPI Driver Component software efficiently:

- SPI Driver Component does not take care of setting the registers that configure clock, pre-scaler and PLL.
- SPI Driver Component handles Master mode and Slave mode.
- SPI Driver Component only supports full-duplex mode.
- Maximum number of channels and sequences configurable is 255, and jobs configurable is 4095.
- The identifiers for channels, jobs and sequences entered by the user should start from 0 and should be contiguous.
- The width of the transmitted data unit is configurable, and the valid values are 8 bits, 16 bits, 24 bits or 32 bits.
- When the user selects the GPIO chip, the user must be configure the parameter PortDomainID in the PORT module since the address of the common output register of the GPIO CS port will be generated based on the domain protection of regionID via the parameter PortDomainID.
- The channels, jobs and sequences cannot be deleted or added at post-build time.
- The SPI hardware unit cannot be deleted or added at post-build time. Also, the SPI hardware units cannot be reassigned to different jobs at post-build time.
- The DMA unit cannot be deleted or added at post-build time. Also, the DMA units cannot be reassigned to the SPI hardware units at post-build time.
- When the level of scalable functionality is configured as 2 and both synchronous and asynchronous transmissions are used, at least 2 SPI buses with separate hardware units are required. In this case, the SPI bus is dedicated to synchronous transmission is configurable.
- When the level of scalable functionality is configured as 2, two modes of asynchronous communication using polling or interrupt mechanism are possible. These modes are selectable during execution time.
- Both synchronous and asynchronous transmission and reception will be performed using the on-chip DMA unit only if the DMA mode is enabled through the configuration.
- The API Spi_Init must be invoked before Port_Init function to avoid unexpected signal being output.
- For availability of Data Consistency Check on the port pins, refer to the respective microcontroller user manual.
- In order to support DEEPSTOP functionality without resetting the microcontroller, the re-initialization of the Driver using the API Spi_Init is supported. To achieve this functionality, the ‘SPI_E_ALREADY_INITIALIZED’ Det error check is suppressed using the parameter “SpiAlreadyInitDetCheck” when DET is enabled. When DET is disabled, there is no impact of the parameter “SpiAlreadyInitDetCheck”.
- The parameter “SpiEnablePersistentHwConfiguration” determines whether hardware configuration is static or dynamic. This is applicable for both synchronous and asynchronous communication and all memory modes.
- If precompile parameter “SpiEnablePersistentHwConfiguration” is configured as true, HW unit is initialized statically. Otherwise, it is initialized dynamically.

- Registers contained in **Table 14.1** are set once in the API Spi_Init and never re-written during the operation when the parameter “SpiEnablePersistentHwConfiguration” is enabled. It reduces the execution time of transmission APIs and ISRs.
-

Table 14.1 Registers to be Configured for Static Configuration

MSIOF HW Unit
SITSCR SITMDR1 SIRMDR1

- All the parameters in channel/job/external devices containers mentioned in **Table 14.2** and **Table 14.3** should be the same for each hardware unit when the parameter “SpiEnablePersistentHwConfiguration” is enabled.

Table 14.2 Channel Container Parameters

Parameter in channel container	Registers linked MSIOF
SpiDataWidth	SITMDR2 (bit 28 to 24, Data Size (8 to 32 bits) SIRMDR2 (bit 28 to 24, Data Size (8 to 32 bits) The word size (bits) of Group 1 is set to the value specified in these bits + 1. Either 8, 16, 24, or 32 bits can be specified as word size)
SpiTransferStart	SITMDR1 (bit 24) SIRMDR1 (bit 24) 0: MSB first 1: LSB first

Table 14.3 Job Container Parameters

Parameter in job container	Registers linked MSIOF
SpiPortPinSelect	SITMDR1 bit 27 to 26 Synchronization Signal Channel Select These bits are valid only in master mode. 00: The frame synchronization signal output at MSIOF_SYNC. 01: The frame synchronization signal output at MSIOF_SS1. 10: The frame synchronization signal output at MSIOF_SS2. 11: Setting prohibited

- To cancel on-going asynchronous sequence, the API Spi_Cancel is used to cancel asynchronous sequence transmission.
- The API Spi_Cancel cancels a sequence at the next job end timing, and on-going job is not stopped.
- The notification functions are invoked after scheduling. Thus, the order of the notification functions is as below:
  1. Invoke SeqStartNotification if necessary.
  2. Start transmission of the first job in the sequence.
  3. Job transmission is completed.
  4. Invoke JobEndNotification if necessary.
  5. If there is the next job in the sequence, start it.
  6. Invoke SeqEndNotification if necessary, for completed sequence.
  7. Invoke SeqStartNotification if necessary, for the next sequence.
  8. If there is the next sequence, start it.
- In the slave mode, SeqStartNotification is invoked when a sequence starts to wait a transfer from a master, not at the actual start time of a transfer.
- The SPI Driver does not handle DMA errors on the SYS-DMAC hardware unit. If any DMA error occurs during transmission, the SPI Driver may stall.

  For MSIOF, errors that occur on DMA (address error, and so on) are not handled by SPI driver.
- To recover from the error state, user shall invoke the API Spi_ForceCancel to cancel the on-going transmission before starting the next transmission.
- When the sequence is canceled by the API Spi_ForceCancel, the result of the on-going job will be SPI_JOB_FAILED, and the result of the sequence will be SPI_SEQ_CANCELED. However, if any error has occurred, the result of the sequence will be SPI_SEQ_FAILED. JobEndNotification of the on-going job and SeqEndNotification of the sequence are invoked.
- If the sequence before the start is canceled by the API Spi_Cancel or Spi_ForceCancel, only SeqEndNotification is invoked, and SeqStartNotification is not invoked.

- If multiple events (transmission complete, cancelation, and hardware error) occur at the same time, the result will be determined as the following order:
  SPI_SEQ_FAILED > SPI_SEQ_OK > SPI_SEQ_CANCELED
- For the parameter “SpiDataShiftEdge”, the meaning of values is defined as below regardless of the parameter “SpiCsPolarity”:
  LEADING: sample MISO at rising edges and setup MOSI at falling edges
  TRAILING: sample MISO at falling edges and setup MOSI at rising edges
- The APIs Spi_WriteIB, Spi_ReadIB, and Spi_SetupEB shall be invoked when the sequence or job containing the specified channel is not on-going on the HW unit. The user should invoke the APIs Spi_GetSequenceResult, Spi_GetJobResult and Spi_GetHWUnitStatus to check the status before invoking the above APIs.
- The argument "Length" of Spi_SetupEB is number of data elements to transmit in SpiDataWidth units.

  When transmit 12 bytes of data:

  If SpiDataWidth is 8, Length is 12.

  If SpiDataWidth is 16, Length is 6.

  If SpiDataWidth is 32, Length is 3.
- When using callback functions, the header file containing the declaration of the function must be configured to parameter SpiUserCallbackHeaderFile.

SpiCsSetupTime and SpiCsHoldTiming are used to configure data pin bit delay and frame synchronization signal timing delay

In case GPIO as chip select mode is enabled, use parameter SpiClk2CsCount to configure the timing between clock and chip select signal

The Baudrate should be calculated with the following parameters:

- SpiInputClockSelect
- SpiBaudrateConfiguration

14.2.2 Preconditions

The following preconditions must be adhered by the user, for proper functioning of the SPI Driver Component:

- The Spi_Lcfg.c, Spi_PBcfg.c and Spi_Cfg.h files generated by the SPI Driver Component Code Generation Tool must be compiled and linked with the SPI Driver Component source files.
- The application has to be rebuilt, if there is any change in the Spi_Lcfg.c, Spi_PBcfg.c and Spi_Cfg.h files generated by the SPI Driver Component Generation Tool.
- File Spi_PBcfg.c generated using SPI Driver Component Generation Tool can be compiled and linked independently.
- The authorization of the user to call the software triggering of a hardware reset is not checked in the SPI Driver. It is the responsibility of the upper layer.
- The SPI Driver Component needs to be initialized before accepting any request. The API Spi_Init should be invoked to initialize SPI Driver Component.
- The user should ensure that SPI Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated only when the static configuration parameter “SpiDevErrorDetect” is enabled. Application should ensure that the right parameters are passed while invoking the APIs when the parameter “SpiDevErrorDetect” is disabled.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used. The version information is described in the 1.1 Supported MCAL Product Release Version.
- The ISR functions and the corresponding handler addresses are provided in Table ISR Handler Addresses. User should ensure that Interrupt Vector table configuration is done as per the information provided in the table.
- Within the callback notification functions, only the following APIs are allowed.
  Spi_ReadIB, Spi_WriteIB, Spi_SetupEB, Spi_GetJobResult, Spi_GetSequenceResult, Spi_GetHWUnitStatus, Spi_Cancel.
  All other SPI Handler/Driver API calls are not allowed.
- The parameter “SpiTimeoutWaitingTime” is used to generate the loop count of timeout value that determines the number of times in a loop which will be executed while polling for each data transmission in Synchronous Transmission. To configure the parameter “SpiTimeoutWaitingTime”, user must consider these factors:
  - Data transmission time strongly depends on the data length and baud rate configured.
  - The parameter “SpiTimeoutWaitingTime” should be big enough to cover the worst-case scenario in the driver configuration.
  - MCU clock.
    Compiler optimization level.
  - It is recommended to add additional margin to the timeout based on user experience.
  - Since this is mere busy loop, it is not guaranteed that the timeout occurs at this time certainly.
- Spi_DataBufferType is defined as uint8, but for external use only.
  The pointer passed to the APIs Spi_ReadIB, Spi_WriteIB and Spi_SetupEB shall be aligned as per the parameter “SpiDataWidth”. If the parameter “SpiDataWidth” is configured as 9-16 bits, the pointer shall be 16-bit aligned. If the parameter “SpiDataWidth” is configured as 17-32 bits, the pointer shall be 32-bit aligned.
- The baud rate should be set within the range allowed by HW unit. If the baud rate outside the range is set, operation cannot be guaranteed.
- A synchronous transmission may be started during another synchronous transmission even though a parameter : SpiSupportConcurrentSyncTransmit is configured as false. Therefore API : Spi_SyncTransmit() shall not be called re-entrantly as a workaraound when the parameter : SpiSupportConcurrentSyncTransmit is configured as false.
- SDMA0_<m> (m=0-15) is selected to configure for SYS-DMAC[1] in Channel[m] (m=0-15) and register SDMCMP_STATUS_[1].
- SDMA1_<m> (m=0-15) is selected to configure for SYS-DMAC[2] in Channel[m] (m=0-15) and register SDMCMP_STATUS_[2].

- SPI_DMA0_nn_ISR (nn: 00..15) is an interrupt function for SYS-DMAC[1] in Channel[m] (m=0-15).
- SPI_DMA1_nn_ISR (nn: 00..15) is an interrupt function for SYS-DMAC[2] in Channel[m] (m=0-15).
- The priority of all ISRs in the SPI Driver should be configured as the same value to prevent the ISRs from preempting each other.
- All APIs of the SPI Driver must not be invoked in contexts which have higher priority than the ISRs of the SPI Driver.
- The SPI Driver does not support exclusive control for the function chip select handled via GPIO SPI driver. GPIO to be used as SPI SS shall be reserved for SPI MCAL. User shall not change the status of the GPIO pin configured for SPI SS from different core and from different ECU which use MCAL/DIO to set the GPIO pin.
- MSIOF has a dedicated SYS-DMAC (Direct Memory Access Controller) to copy data from user memory to hardware for transmitting and opposite direction for receiving.
- When using DMA for SPI transmission, cache memory must not be enabled.

14.2.3 Data Consistency

To support the re-entrance and interrupt services, the SPI Driver Component will ensure the data consistency while accessing its own RAM storage or hardware registers.

```
#define SPI_ENTER_CRITICAL_SECTION(Exclusive Area)
SchM_Enter_Spi_##Exclusive_Area()

#define SPI_EXIT_CRITICAL_SECTION(Exclusive Area)
SchM_Exit_Spi_##Exclusive_Area()
```

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:
SPI_RAM_DATA_PROTECTION
SPI_INTERRUPT_CONTROL_PROTECTION

These functions can be disabled by setting the configuration parameter “SpiCriticalSectionProtection” as false.

14.2.4 Parallel Transmission

When the parameter “SpiSupportConcurrentAsyncTransmit” is configured as true, the SPI Driver works with the parallel transmission mode during asynchronous transmission.
In this mode, job queues are created for each hardware unit, and transmissions are performed simultaneously on each hardware unit.
If a sequence has jobs linked to multiple hardware units, each job is performed simultaneously regardless of their priorities.
If a sequence is non-interruptible, only jobs which belong to the same hardware unit are transmitted continuously.

14.2.5 Memory Modes

14.2.5.1 FIFO Mode

- FIFO mode by default is supported and provided by MSIOF HW unit.

14.3 Architecture Details

To minimize the effort and to optimize the reuse of the software developed on different platforms, the SPI Driver is split as High Level Driver and Low Level Driver. The SPI Driver architecture is shown in the **Figure 14.3**:

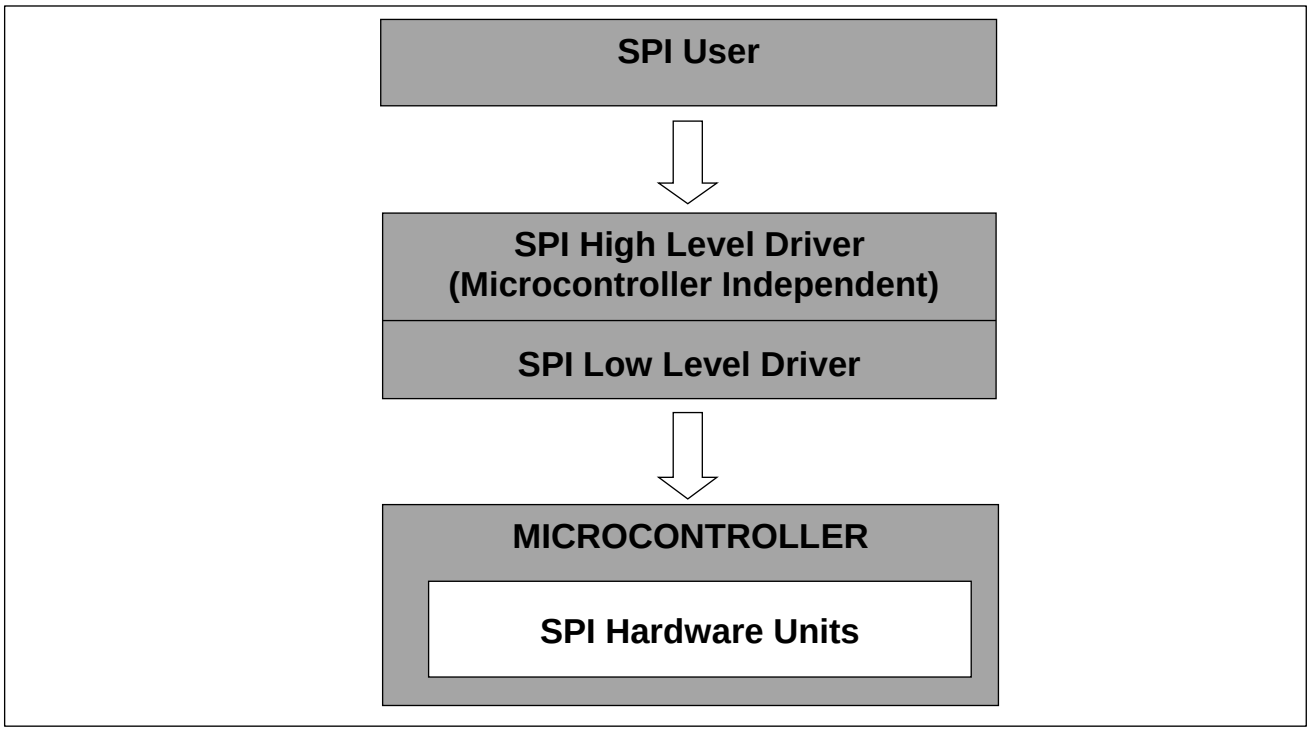


Figure 14.3 SPI Driver Architecture

The High Level Driver exports the AUTOSAR API towards upper modules and it will be designed to allow the compilation for different platforms without or only slight modifications, i.e. no reference to the specific microcontroller features or registers will appear in the High Level Driver. All these references are moved inside a µC specific Low Level Driver. The Low Level Driver interface extends the High Level Driver types and methods in order to adapt it to the specific target microcontroller.

SPI Driver Component:
The SPI Driver provides services for reading and writing to devices connected via SPI buses. It provides access to SPI communication to several users such as EEPROM, Watchdog, and I/O ASICs. It also provides the required mechanism to configure the on-chip SPI peripheral.

The SPI Driver Component is divided into the following sub modules based on the functionality required:

Initialization
De-initialization
Buffer Management
Communication
Notification
Status Information
Version information

The basic architecture of the SPI Driver Component is illustrated in the **Figure 14.4**:

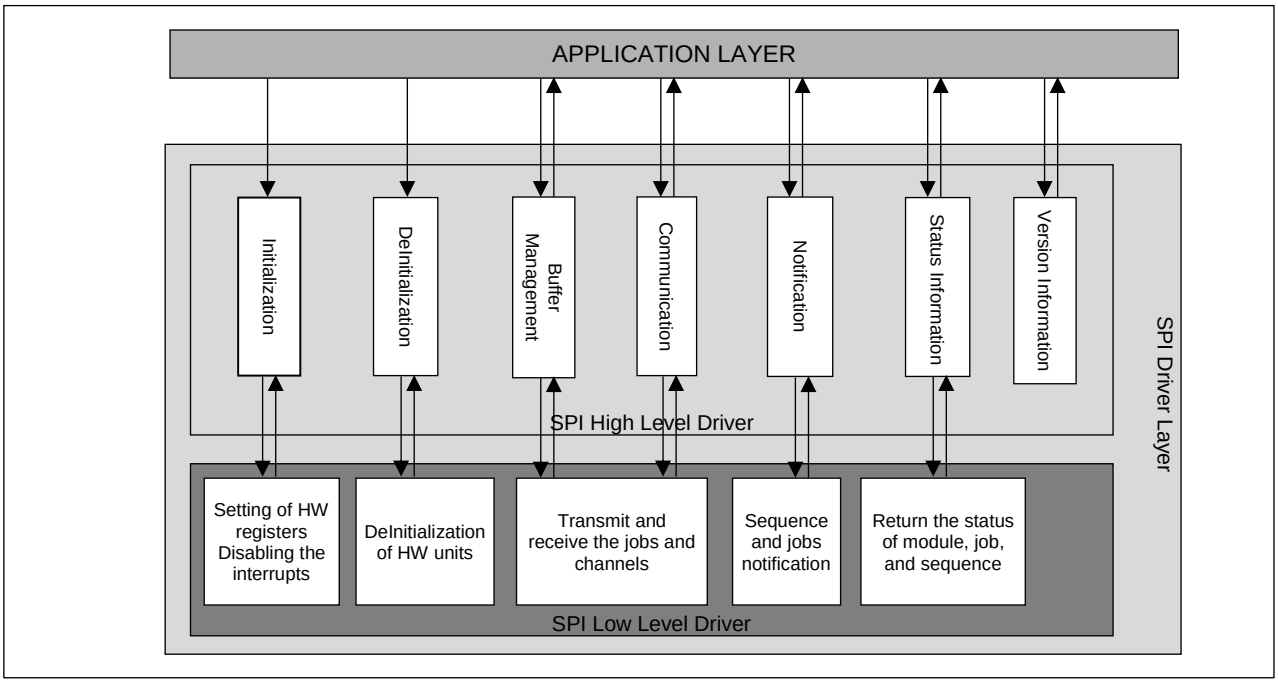


Figure 14.4 Component Overview of SPI Driver Component

SPI Driver Initialization and De-Initialization module
This module initializes and De-initializes the SPI Driver. It provides the APIs Spi_Init and Spi_DeInit. The API Spi_Init should be invoked before the usage of any other APIs of SPI Driver Module. This API should be invoked prior to Port_Init. De-initialization function puts all microcontroller SPI peripherals in the same state such as Power On Reset.

Buffer Management
This module provides the services for reading and writing the internal buffers and setting up the external buffer. The type of buffer for each channel is configurable as either internal or external.

The APIs related to this module are Spi_WriteIB, Spi_ReadIB and Spi_SetupEB.

Communication
This module provides the services for the transmission of data on the SPI bus both synchronously and asynchronously, canceling the on-going transmission and setting the asynchronous transfer mode.

The synchronous mode is based on polling mechanism. However, for the asynchronous mode, the possible mechanisms are Polling and Interrupt mode. One of these modes is selectable during execution by one of the services provided by this sub-module.

The APIs related to this module are Spi_SyncTransmit, Spi_AsyncTransmit, Spi_SetAsyncMode, Spi_Cancel and Spi_ForceCancel.

Status Information
This module provides the services for getting the status of the SPI Driver and hardware unit. The services also include getting the result of the specified job and specified sequence.

The APIs related to this module are Spi_GetStatus, Spi_GetHWUnitStatus, Spi_GetJobResult and Spi_GetSequenceResult.

Module Version Information
This module provides APIs for reading module ID, vendor ID, and vendor-specific version numbers.

The API related to this module is Spi_GetVersionInfo.

Hardware Error Diagnosis
There are different hardware errors present in the hardware that are used to report production errors if any of the following scenario occurs:

1. **Transmit Overflow:**
A transmit overflow occurs when there is an attempt to write to Transmit FIFO Data Register when the transmit FIFO is full

2. **Transmit Underflow:**
A transmit underflow occurs when loading for transmission has occurred but the transmit FIFO is empty

3. **Receive Underflow:**
A receive underflow occurs when reading of Receive FIFO Data Register has occurred when the receive FIFO is empty

4. **Receive Overflow:**
A receive overflow occurs when writing has been caused by receiving operation when the receive FIFO is full

Table 14.4 Relationship with Configuration Parameter

Configuration Parameter	Related Registers	Related API Name
SPI_E_DATA_TX_TIMEOUT_FAILURE	-	Spi_SyncTransmit

14.4 SPI Driver Component Header and Source File Description

This section explains the SPI Driver Component's source and header files. These files must be included in the project application while integrating with other modules.

The C header file generated by SPI Driver Generation Tool:

- Spi_Cfg.h

The C source file generated by SPI Driver Generation Tool:

- Spi_PBcfg.c
- Spi_Lcfg.c

The SPI Driver Component C header files and Component source files:
Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.3.8.3 Folder Structure.

The Stub C header files:
Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.2.9 Stubs File.

14.5 Application Programming Interface

This section explains the Data types and APIs provided by the SPI Driver Component to the Upper layers.

14.5.1 Imported Types

This section explains the Data types imported by the SPI Driver Component and lists its dependency on other modules.

14.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed:

Std_ReturnType
Std_VersionInfoType

14.5.1.2 Other Module Types

In this section, all types included in the Dem.h are listed:

Dem_EventIdType
Dem_EventStatusType

14.5.2 Type Definitions

This section explains the type definitions of SPI Driver Component according to AUTOSAR Specification.
The **Table 14.5** to **Table 14.16** show the definition specifications.

14.5.2.1 Spi_ConfigType

Table 14.5 Spi_ConfigType

Name:	Spi_ConfigType	
Type:	Structure	
Range:	Implementation Specific	The contents of the initialization data structure are SPI-specific.
Description:	This type of the external data structure shall contain the initialization data for the SPI Handler/Driver.	

14.5.2.2 Spi_StatusType

Table 14.6 Spi_StatusType

Name:	Spi_StatusType	
Type:	Enumeration	
Range:	SPI_UNINIT	The SPI Handler/Driver is not initialized or not usable.
	SPI_IDLE	The SPI Handler/Driver is not currently transmitting any job.
	SPI_BUSY	The SPI Handler/Driver is performing a SPI job (transmit).
Description:	This type defines a range of specific status for SPI Handler/Driver.	

14.5.2.3 Spi_JobResultType

Table 14.7 Spi_JobResultType

Name:	Spi_JobResultType	
Type:	Enumeration	
Range:	SPI_JOB_OK	The last transmission of the job has been finished successfully.
	SPI_JOB_PENDING	The SPI Handler/Driver is performing a SPI job. The meaning of this status is equal to SPI_BUSY.
	SPI_JOB_FAILED	The last transmission of the job has failed.
	SPI_JOB_QUEUED	An asynchronous transmit job has been accepted, while actual transmission for this job has not started yet.
Description:	This type defines a range of specific jobs status for SPI Handler/Driver.	

14.5.2.4 Spi_SeqResultType

Table 14.8 Spi_SeqResultType

Name:	Spi_SeqResultType	
Type:	Enumeration	
Range:	SPI_SEQ_OK	The last transmission of the sequence has been finished successfully.
	SPI_SEQ_PENDING	The SPI Handler/Driver is performing a SPI sequence. The meaning of this status is equal to SPI_BUSY.
	SPI_SEQ_FAILED	The last transmission of the sequence has failed.
	SPI_SEQ_CANCELED	The last transmission of the sequence has been canceled by user.
Description:	This type defines a range of specific sequences status for SPI Handler/Driver.	

14.5.2.5 Spi_DataBufferType

Table 14.9 Spi_DataBufferType

Name:	Spi_DataBufferType	
Type:	uint8	
Range:	0 to 255	This type is only used as pointer to pass the top address of buffers to Spi_ReadIB, Spi_WriteIB and Spi_SetupEB. The actual type of buffers depends on “SpiDataWidth”.
Description:	Type of application data buffer elements.	

14.5.2.6 Spi_NumberOfDataType

Table 14.10 Spi_NumberOfDataType

Name:	Spi_NumberOfDataType
Type:	uint16
Range:	0 to 65535
Description:	Type that defines the number of data elements of the type Spi_DataBufferType to send and/or receive by channel.

14.5.2.7 Spi_ChannelType

Table 14.11 Spi_ChannelType

Name:	Spi_ChannelType
Type:	uint8
Range:	0 to 254
Description:	Specifies the identification (ID) for a channel.

14.5.2.8 **Spi_JobType**

Table 14.12 Spi_JobType

Name:	Spi_JobType
Type:	uint16
Range:	0 to 4094
Description:	Specifies the identification (ID) for a job.

14.5.2.9 **Spi_SequenceType**

Table 14.13 Spi_SequenceType

Name:	Spi_SequenceType
Type:	uint8
Range:	0 to 254
Description:	Specifies the identification (ID) for a sequence of jobs.

14.5.2.10 **Spi_HWUnitType**

Table 14.14 Spi_HWUnitType

Name:	Spi_HWUnitType
Type:	uint8
Range:	0 to 255
Description:	Specifies the identification (ID) for a SPI Hardware microcontroller peripheral (unit).

14.5.2.11 **Spi_AsyncModeType**

Table 14.15 Spi_AsyncModeType

Name:	Spi_AsyncModeType	
Type:	Enumeration	
Range:	SPI_POLLING_MODE	As the asynchronous mechanism is ensured by polling, the interrupts related to SPI buses handled asynchronously are disabled.
	SPI_INTERRUPT_MODE	As the asynchronous mechanism is ensured by interrupt, the interrupts related to SPI buses handled asynchronously are enabled.
Description:	Specifies the asynchronous mechanism mode for SPI buses handled asynchronously in LEVEL2.	

14.5.2.12 **Spi_CSType**

Table 14.16 Spi_CSType

Name:	Spi_CSType	
Type:	Enumeration	
Range:	SPI_DISABLE_CS	Master chip select handling is disabled
	SPI_GPIO_CS	Chip select is handled by general purpose IO
	SPI_PERIPHERAL_ENGINE_CS	Chip select is handled by peripheral HW engine
	SPI_SLAVE_CS	Slave mode
Description:	Specifies the chip select handling mode of job.	

14.5.3 Function Definitions

The **Table 14.17** shows list of APIs, and Renesas Original callback functions.

Table 14.17 List of APIs

Sl. No.	APIs
AUTOSAR API	
1	Spi_Init
2	Spi_DeInit
3	Spi_WriteIB
4	Spi_AsyncTransmit
5	Spi_ReadIB
6	Spi_SetupEB
7	Spi_GetStatus
8	Spi_GetJobResult
9	Spi_GetSequenceResult
10	Spi_GetVersionInfo
11	Spi_SyncTransmit
12	Spi_Cancel
13	Spi_SetAsyncMode
14	Spi_MainFunction_Handling
15	Spi_GetHWUnitStatus
Renesas Original API	
16	Spi_ForceCancel
Renesas Original callback function	
17	SpiSeqStartNotification *1

Note *1: The name of callback functions are configurable by user.

14.5.3.1 **Renesas Original API**

(1) Spi_ForceCancel

The **Table 14.18** shows the available RENESAS APIs.

Table 14.18 Spi_ForceCancel

Function Name:	Spi_ForceCancel		
Syntax:	FUNC(void, SPI_PUBLIC_CODE) Spi_ForceCancel (const Spi_SequenceType LucSequence)		
Service ID:	0xA0		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LucSequence	Sequence ID.	
		Range :	0..(SpiMaxSequence – 1)
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	None		
Description:	Service cancels the specified sequence being asynchronous transmission.		
Preconditions:	SPI Driver must be initialized. “SpiForceCancelApi” must be configured as true. This API must not be invoked in a context which has the higher priority than ISRs or Spi_MainFunction_Handling.		
Remarks:	None		
DET/DEM Error handled:	SPI_E_PARAM_SEQ	Sequence ID is invalid	
	SPI_E_UNINIT	SPI Driver is not initialized	
	SPI_E_SEQ_IN_PROCESS	The specified sequence is still being processed as a synchronous transmission.	

14.5.3.2 Renesas Original callback function

(1) SpiSeqStartNotification

The **Table 14.19** shows explanation of SpiSeqStartNotification function.

Table 14.19 SpiSeqStartNotification

Function Name:	SpiSeqStartNotification
Syntax:	FUNC(void, SPI_APPL_CODE) SpiSeqStartNotification(void)
Service ID:	None
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (In):	None
Parameters (In-Out):	None
Parameters (Out):	None
Return Value:	None
Description:	Callback routine provided by the user for each Sequence to notify the caller that a sequence start.
Preconditions:	This function should be enabled by parameter SpiEnableSeqStartNotification.
Remarks:	This function name can be defined SpiSeqStartNotification parameter by user.
DET/DEM Error handled:	None

14.5.4 Preemption of APIs

The following table specifies the preemption of each API that can be invoked at the same time as the API.

Table 14.20 Preemption Table of APIs of the SPI Driver

	Spi_Init	Spi_DeInit	Spi_WriteIB	Spi_AsyncTransmit	Spi_ReadIB	Spi_SetupEB	Spi_GetStatus	Spi_GetJobResult	Spi_GetSequenceResult	Spi_GetVersionInfo	Spi_SyncTransmit	Spi_GetHWUnitStatus	Spi_Cancel	Spi_SetAsyncMode	Spi_MainFunction_Handling	Spi_ForceCancel
Spi_Init	-															
Spi_DeInit	-	-														
Spi_WriteIB	-	-	*4													
Spi_AsyncTransmit	-	-	*5	√												
Spi_ReadIB	-	-	√	√	√											
Spi_SetupEB	-	-	√	*5	√	*4										
Spi_GetStatus	-	-	√	√	√	√	√									
Spi_GetJobResult	-	-	√	√	√	√	√	√								
Spi_GetSequenceResult	-	-	√	√	√	√	√	√	√							
Spi_GetVersionInfo	√	√	√	√	√	√	√	√	√	√						
Spi_SyncTransmit	-	-	*5	√	√	*5	√	√	√	√	√					
Spi_GetHWUnitStatus	-	-	√	√	√	√	√	√	√	√	√	√				
Spi_Cancel	-	-	√	√	√	√	√	√	√	√	√	√	√			
Spi_SetAsyncMode	-	-	√	-	√	√	√	√	√	√	√	√	√	-		
Spi_MainFunction_Handling	√	√	√	√	√	√	√	√	√	√	√	√	√	√	-	
Spi_ForceCancel	-	-	√	*2	√	√	√	√	√	√	√	√	√	-	*1	*3

-: cannot be invoked at the same time
√: can be invoked at the same time
Number (1, 2, 3, 4, 5): can be invoked at the same time with a condition. The detail is addressed in the corresponding note.
Note:

1 Spi_ForceCancel shall not preempt Spi_MainFunction_Handling, but it can preempt Spi_ForceCancel. In other words, Spi_ForceCancel should be invoked in the lower priority task than Spi_MainFunction_Handling or sequentially.

2 Spi_ForceCancel shall not preempt Spi_AsyncTransmit with the same sequence.

3 Spi_ForceCancel with the same sequence is not reentrant, but it with the different sequence is reentrant.

4 Spi_SetupEB and Spi_WriteIB with the same channel are not reentrant, but those with different channels are reentrant.

5 Spi_WriteIB / Spi_SetupEB and Spi_AsyncTransmit / Spi_SyncTransmit with the same channel in the sequence shall not preempt each other.

14.6 Development and Production Errors

In this section, the development errors reported by the SPI Driver Component are tabulated. The development errors will be reported only when the pre-compiler option “SpiDevErrorDetect” is enabled in the configuration. The production code errors are not supported by SPI Driver Component.

14.6.1 SPI Driver Component Development Errors

The **Table 14.21** and **Table 14.22** contain the DET errors reported by SPI Driver Component. These errors are reported to Default Error Tracer Module when the SPI Driver Component APIs are invoked with wrong input parameters or without initialization of the driver. If "SpiDevErrorDetect" is configured as false, no error check is performed.

Table 14.21 DET Errors of SPI Driver Component (1/2)

Sl. No.	**1**
Error Code	SPI_E_PARAM_CHANNEL
Value	0x0A
Related API(s)	Spi_WriteIB, Spi_ReadIB and Spi_SetupEB
Source of Error	When the API service is invoked with invalid channel ID and if incorrect type of channel (IB or EB) is used with services.
Sl. No.	**2**
Error Code	SPI_E_PARAM_JOB
Value	0x0B
Related API(s)	Spi_GetJobResult
Source of Error	When the API service is invoked with invalid job ID.
Sl. No.	**3**
Error Code	SPI_E_PARAM_SEQ
Value	0x0C
Related API(s)	Spi_AsyncTransmit, Spi_GetSequenceResult, Spi_SyncTransmit, Spi_Cancel and Spi_ForceCancel
Source of Error	When the API service is invoked with invalid sequence ID.
Sl. No.	**4**
Error Code	SPI_E_PARAM_LENGTH
Value	0x0D
Related API(s)	Spi_SetupEB
Source of Error	When the API service is invoked with length greater than the configured length.
Sl. No.	**5**
Error Code	SPI_E_PARAM_UNIT
Value	0x0E
Related API(s)	Spi_GetHWUnitStatus
Source of Error	When the API service is invoked with invalid hardware unit ID.

Table 14.22 DET Errors of SPI Driver Component (2/2)

Sl. No.	6
Error Code	SPI_E_ALREADY_INITIALIZED
Value	0x4A
Related API(s)	Spi_Init
Source of Error	When the API Spi_Init is invoked when the SPI Driver is already initialized.
Sl. No.	**7**
Error Code	SPI_E_INVALID_DATABASE
Value	0xEF
Related API(s)	Spi_Init
Source of Error	When the API service is invoked with invalid pointer.
Sl. No.	**8**
Error Code	SPI_E_UNINIT
Value	0x1A
Related API(s)	Spi_DeInit, Spi_AsyncTransmit, Spi_Cancel, Spi_ForceCancel, Spi_GetHWUnitStatus, Spi_GetJobResult, Spi_GetSequenceResult, Spi_WriteIB, Spi_ReadIB, Spi_SetupEB, Spi_SyncTransmit, Spi_SetAsyncMode and Spi_GetStatus
Source of Error	When the APIs are invoked without the initialization of SPI Driver Component.
Sl. No.	**9**
Error Code	SPI_E_PARAM_POINTER
Value	0x10
Related API(s)	Spi_Init, Spi_ReadIB and Spi_GetVersionInfo
Source of Error	When the API service is invoked with null pointer.

14.6.2 SPI Driver Component Production Errors

In this section, the DEM errors identified in the SPI Driver Component are listed. SPI Driver Component reports these errors to DEM by invoking the API Dem_SetEventStatus. This API is invoked when the processing of the given API request fails. The **Table 14.23** describes the list of DEM errors.

Table 14.23 DEM Errors of SPI Driver Component

Sl. No.	1
Error Code	SPI_E_HARDWARE_ERROR
Related API(s)	Spi_SyncTransmit, Spi_MainFunction_Handling, SPI_MSIOFn_ISR (n: 0..5), SPI_MSIOFn_CAT2_ISR (n: 0..5) SPI_DMAj_nn_ISR (j: 0..1, nn: 00..15), SPI_DMAj_nn_CAT2_ISR (j: 0..1, nn: 00..15)
Source of Error	When an overflow/underflow/frame sync error occurs. For more detail of each error, please refer the V4H hardware user's manual.
Origin	AUTOSAR
Sl. No.	**2**
Error Code	SPI_E_DATA_TX_TIMEOUT_FAILURE
Related API(s)	Spi_SyncTransmit
Source of Error	When a hardware data transmit timeout error is detected, this error should be reported. If the reference is not configured the error shall not be reported.
Origin	Renesas
Sl. No.	**3**
Error Code	SPI_E_INTERRUPT_CONTROLLER_FAILURE

Related API(s)	SPI_MSIOFn_ISR (n: 0..5), SPI_MSIOFn_CAT2_ISR (n: 0..5) SPI_DMAj_nn_ISR (j: 0..1, nn: 00..15), SPI_DMAj_nn_CAT2_ISR (j: 0..1, nn: 00..15)
Source of Error	An ISR is invoked even though the corresponding interrupts of MSIOF are masked or prohibited.
Origin	Renesas
Sl. No.	**4**
Error Code	SPI_E_WRITE_VERIFY_FAILURE
Related API(s)	Spi_Init, Spi_DeInit, Spi_AsyncTransmit, Spi_SyncTransmit, Spi_SetAsyncMode, and Spi_MainFunction_Handling SPI_MSIOFn_ISR (n: 0..5), SPI_MSIOFn_CAT2_ISR (n: 0..5) SPI_DMAj_nn_ISR (j: 0..1, nn: 00..15), SPI_DMAj_nn_CAT2_ISR (j: 0..1, nn: 00..15), Spi_Cancel
Source of Error	Written data to control registers of MSIOF and SYS-DMAC is not correct with the expected value by reading back.
Origin	Renesas

14.7 SPI Driver Component Runtime Errors

The following table contains the Runtime errors reported by SPI Driver Component. These errors are reported to Default Error Tracer Module when the SPI Driver Component APIs are invoked in time and the SPI Driver Component does not meet the requirement to execute the API.

Table 14.24 Runtime Errors of SPI Driver Component

Sl. No.	1
Error Code	SPI_E_SEQ_PENDING
Value (hex)	0x2A
Related API(s)	Spi_AsyncTransmit
Source of Error	Services called in a wrong sequence.
Sl. No.	**2**
Error Code	SPI_E_SEQ_IN_PROCESS
Value (hex)	0x3A
Related API(s)	Spi_SyncTransmit, Spi_Cancel, Spi_ForceCancel
Source of Error	Synchronous transmission service called at wrong time.

14.8 Memory Organization

Table 14.25 and **Table 14.26** describe the Memory Section Symbols used by the SPI module.
The Memory Section prefix is "SPI".

Table 14.25 ROM Sections of the SPI Driver

Section name	Alignment (*1)	Description
SPI_PUBLIC_CODE_ROM	-	API(s) of SPI Driver Component that can be in code memory.
SPI_PRIVATE_CODE_ROM	-	Internal functions of SPI Driver Component code that can be in code memory.
CODE_FAST	-	SPI Driver code related to ISR function are placed in this memory.
CONST_UNSPECIFIED	-	This section consists of SPI Driver Component constant structures used for function pointers in SPI Driver Component. It can be in code memory.
CONFIG_DBTOC_DATA_UNSPECIFIED	-	This section consists of SPI Driver Component database table of contents generated by the SPI Driver Component Generation Tool. It can be in code memory.
CONFIG_DATA_UNSPECIFIED	-	This section consists of SPI Driver Component constant configuration structures. It can be located in code memory.

Table 14.26 RAM Sections of the SPI Driver

Section Name	Alignment (*1)	Description
VAR_INIT_1	-	This section consists of the global RAM variables of 1-bit size initialized by startup code and used internally by SPI Driver Component. It can be in data memory.
VAR_NO_INIT_32	-	This section consists of the global RAM variables of 32-bit size used internally by SPI Driver Component. It can be in data memory.
VAR_NO_INIT_UNSPECIFIED	-	This section consists of the global RAM of unspecific size variables used internally by SPI Driver Component. It can be in data memory.
VAR_NO_INIT_PTR	-	This section consists of the global RAM pointer variables used internally by SPI Driver Component. It can be in data memory.

*1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

14.9 Device-Specific Information

V4H supports the following devices:
Refer to “R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 5.1 Product.

14.9.1 Multi-Core / Multi-Instantiation

SPI driver does not support multi-core and multi-instantiation for this device.

14.9.2 Interaction between the User and SPI Driver Component

The following sections detail the services supported by the SPI Driver Component to the upper layers users and the mapping of the channels to the hardware units:

14.9.2.1 Channel Mapping

The channel setting does not depend on H/W resources.

14.9.2.2 ISR Functions

The **Table 14.27** and **Table 14.28** provide the list of handlers corresponding to the hardware unit ISR(s) in SPI Driver Component. The user should configure the ISR functions mentioned below.

Table 14.27 Interrupt Vector Table for MSIOF (n: 0..5)

Interrupt Source	Name of the ISR Function
MSIOFn	SPI_MSIOFn_ ISR
	SPI_MSIOFn_ CAT2_ISR

Table 14.28 Interrupt Vector Table for SYS-DMAC (j: 0..1, nn:00..15)

Interrupt Source	Name of the ISR Function
SYS-DMAC	SPI_DMAj_nn_ISR
	SPI_DMAj_nn_CAT2_ISR

14.10 Non-AUTOSAR environment integration

The SPI Driver Components for R-Car Gen4 microcontrollers is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the SPI Driver Components to Non-AUTOSAR environment.

14.10.1 Stub modules handling

14.10.1.1 MCU

The MCU driver provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required by other MCAL software modules. The MCU stub files are located in the following folder:

\external\rel\V4H\common_family\config\V4H\19_11\

In AUTOSAR environment, SPI Component Driver uses McuMSOClk from MCU module for clock references. With Non-AUTOSAR environment, user needs to config MCU to provide clock for SPI Driver as reference:

*.arxml in \external\rel\V4H\common_family\config\V4H\19_11\

14.10.1.2 Det

The Det stub files are organized in the following folder:

\external\rel\common\generic\stubs\19_11\Det

In the AUTOSAR environment, SPI Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g., SPI Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:

Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

14.10.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.

The SchM (Basic Software Scheduler) stub files are organized in the following folder:

\external\rel\common\generic\stubs\19_11\Rte

SPI driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when SPI_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user MSN driver as following:

Write SchM functions with prototype as below:

void SchM_Enter_Spi_SPI_RAM_DATA_PROTECTION (void);

void SchM_Exit_Spi _SPI_RAM_DATA_PROTECTION (void);

void SchM_Enter_Spi _SPI_INTERRUPT_CONTROL_PROTECTION (void);

void SchM_Exit_Spi _SPI_INTERRUPT_CONTROL_PROTECTION (void);

14.10.1.4 Dem

The Dem stub files are organized in the following folder:

\external\rel\common\generic\stubs\19_11\Dem

In the AUTOSAR environment, SPI Driver Components uses Dem_ReportErrorStatus API provided by the DEM module to report a production error e.g. SPI Driver interrupt consistency check failed, … The API prototype is as of follow:

Dem_ReportErrorStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_ReportErrorStatus API with their current error handling strategy.

14.10.2 Callback function usage

Callback function was provided to notify caller that a sequence starts in each Sequence by the paramter:

SpiSeqStartNotification

The return type of this callback function must be void. And no parameter shall be passed to this callback function.
Example: void SpiSequence0StartNotification(void)

14.10.3 Scheduled function usage

SPI Driver Component scheduled function shall be invoked directly by the BSW scheduler. The SPI Driver Component scheduled functions are listed below:

Spi_MainFunction_Handling

Non AUTOSAR user can invoke above API using their own implementation.
E.g: To manage the asynchronous mode managed with polling.

14.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:

\external\rel\V4H\common_family\src\arm\Interrupt_VectoTable.c

15.WDG

15.1 Overview

The purpose of this chapter is to describe the information related to WDG Driver Component.

This chapter shall be used as reference by the users of WDG Driver Component. The system overview of complete AUTOSAR architecture is shown in the Figure 15-1.

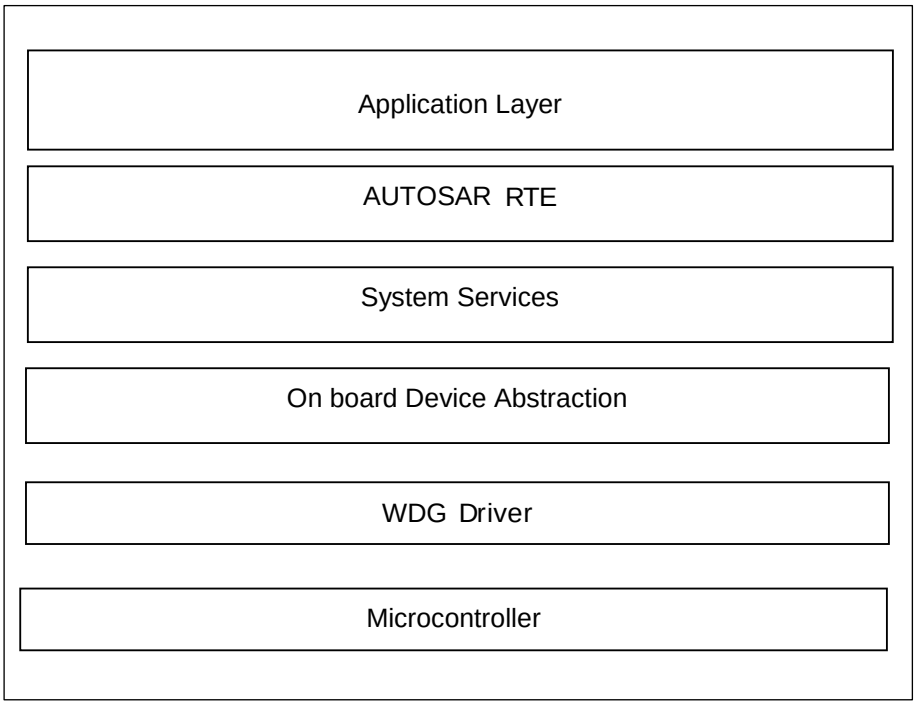


Figure 15-1 System Overview of AUTOSAR Architecture

The WDG Component comprises Embedded software and the Configuration Tool to achieve scalability and configurability.

The WDG Generation Tool is a command line tool that accepts ECU configuration description files as input and generates source and header files. The configuration description is an ARXML file that contains information about the configuration for Watchdog timer. The tool generates the Wdg_PBcfg.c and Wdg_Cfg.h for Watchdog.

The Figure 15-2 depicts the WDG Driver as part of layered AUTOSAR MCAL Layer:

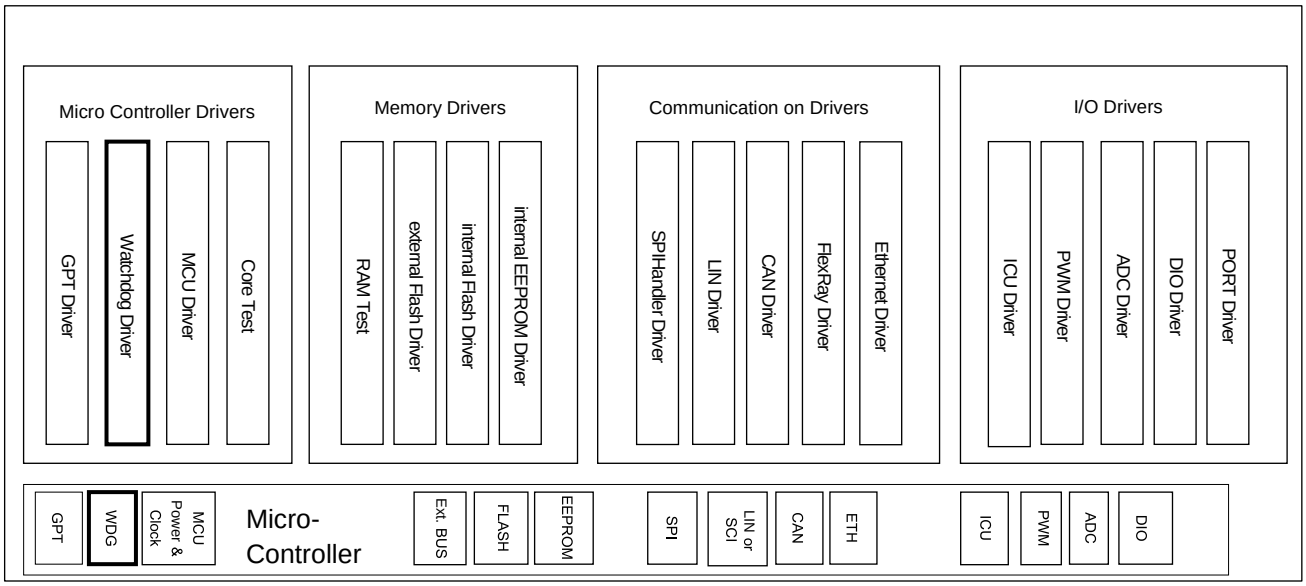


Figure 15-2 System Overview of the WDG Driver in AUTOSAR MCAL Layer

Watchdog Driver module provides the services for initializing, changing the operation mode and triggering the watchdog.

15.2 Forethoughts

15.2.1 General

The following information will help the user to use the WDG Driver Component software efficiently:

- The WDG Component does not implement any scheduled functions.
- WDG Component implements GPT Call Back Notification functions.
- Example code mentioned in this chapter shall be taken only as a reference for implementation.
- All development errors will be reported to Det by using the API Det_ReportError() provided by DET.
- All production errors will be reported to Dem by using the API Dem_SetEventStatus() provided by DEM.
- It should be ensured that the respective clock source is switched ON before Watchdog is set to Main Oscillator in Wdg_Init API.

- WDG MCAL driver can be used independently if the WDG is operated using WDG Overflow interrupt or internal reset request when WDG has overflowed. User should set the required configuration before using the WDG driver. Refer to the following information:
  - To enable or disable WDG Overflow interrupt, user shall set bit WOVFE of register RWTCSRA accordingly. Please refer to section "154.1.2.2 RCLK Watchdog Timer Control/Status Register A (RWTCSRA)" of R-Car V4H Series User's Manual.
  - To enable or disable reset request, user shall set bit RWDT_RSTMSK of register WDTRSTCR accordingly. Please refer to section "14.2.3 Watchdog Timer Reset Control Register (WDTRSTCR)" of R-Car V4H Series User's Manual.
  - When WDG Overflow interrupt is used, user shall implement the necessary interrupt handler and keep the WDG reset request disabled. As the result, the WDG reset is not generated and user has to ensure a reset is generated before using WDG driver again.

15.2.2 Preconditions

The following preconditions must be adhered by the user, for proper functioning of the WDG Driver Component:

- Validation of input parameters are performed only when the static configuration parameter 'WDG_DEV_ERROR_DETECT' is enabled. Application should ensure that the right parameters are passed while invoking the APIs when WDG_DEV_ERROR_DETECT is disabled.
- A mismatch in the version numbers results in compilation error. Ensure that the correct versions of the header and the source files are used.
- The files 'Wdg_Cfg.h' and 'Wdg_PBcfg.c' generated using the watchdog driver generation tool has to be linked with the WDG Component source files.
- The file 'Wdg_PBcfg.c' generated for single configuration set using Watchdog Driver Generation Tool can be compiled and linked independently.
- The WDG Component needs to be initialized before accepting any API requests. Wdg_Init should be called by the ECU State Manager Module to initialize WDG Component. It should not be called more than once.
- User should ensure that MCU Driver is supplies clock for RCLK to make WDG component works properly. Make sure that Mcu_Init execute before invoking Wdg_Init.
- The user should ensure that WDG Component API requests are invoked in the correct and expected sequence along with correct input arguments.

- The RWDT is a single-channel timer that uses the RCLK as an input and can be used as a watchdog timer. RCLK is a clock, which is generated by the clock pulse generator (CPG)
  Frequency of RCLK: - R-Car V4H: 32.8 kHz.
- The WDG using GPT call back function to trigger it. For more detail about using GPT, please refer section 15.2.5
- User must initialize some other hardware IPs as below before initializing WDG driver.
- All APIs of the Wdg Driver must not be invoked in contexts which have higher priority than the ISRs of the GPT module.

Table 15-1 List of hardware IP affect to WDG

Hardware IP	Module Name	Expected Setting	Description
RST	-	WDTRSTCR should be configured as Section 14.2.3 in HW UM for V4H	This bit is used to mask the detection of RWDT overflow. 0: The reset request will be raised whenever RWDT is overflowed. 1: There is no reset request will be raised when RWDT is overflowed. It should be configured by user before using this module.
CPG	MCU	Clock signal for WDG should be enabled based on configure for following registers: MSTPSR9, MSTPCR9, SRCR9, CPGWPCR.	MCU will handle to enable clock signal. Refer to MCU User Manual for more detail in setting.
AXI-bus	-	SDRAM address mapping in HW should be configured according to Section 18.3 in HW UM for V4H	WDG is designed assuming the setting of SDRAM address mapping is corrected, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since WDG operation is under controlled by DRAM for memory control.
DBSC5	-	The maximum use of SDRAM bus bandwidth will be enabled by DBSC5 according to Section 33.4.1 in HW UM for V4H	WDG is designed assuming the configuration of SDRAM bus bandwidth is suitable, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since WDG operation is under controlled by DRAM for memory control.

Life Cycle	-	Security/safety access protection for RWDT should be configured to be accessed by all resource.	WDG is designed assuming the setting of this protection mechanism allow register to be accessed by all resource, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed.

15.2.3 Data Consistency

To support the re-entrance and interrupt services, the AUTOSAR WDG component will ensure the data consistency while switching the watchdog mode and during the watchdog trigger routine. The WDG Driver component will use SchM_Enter_Wdg and SchM_Exit_Wdg functions. The SchM_Enter_Wdg function is called before the data needs to be protected, and the SchM_Exit_Wdg function after the data is accessed.
The following exclusive areas along with scheduler services are used to provide data integrity for shared resources:

- WDG_RAM_DATA_PROTECTION
- WDG_INTERRUPT_CONTROL_PROTECTION (This protection area depends on HW)

The protection area ‘WDG_RAM_DATA_PROTECTION’ is used to protect the RAM Data.

The protection area ‘WDG_INTERRUPT_CONTROL_PROTECTION’ is used to protect register accessible by other modules.

The functions ‘SchM_Enter_Wdg’ and SchM_Exit_Wdg can be disabled by disabling the configuration parameter ‘WdgCriticalSectionProtection’.

15.2.4 WDG State Diagram

The State diagram of WDG Driver is as shown below:

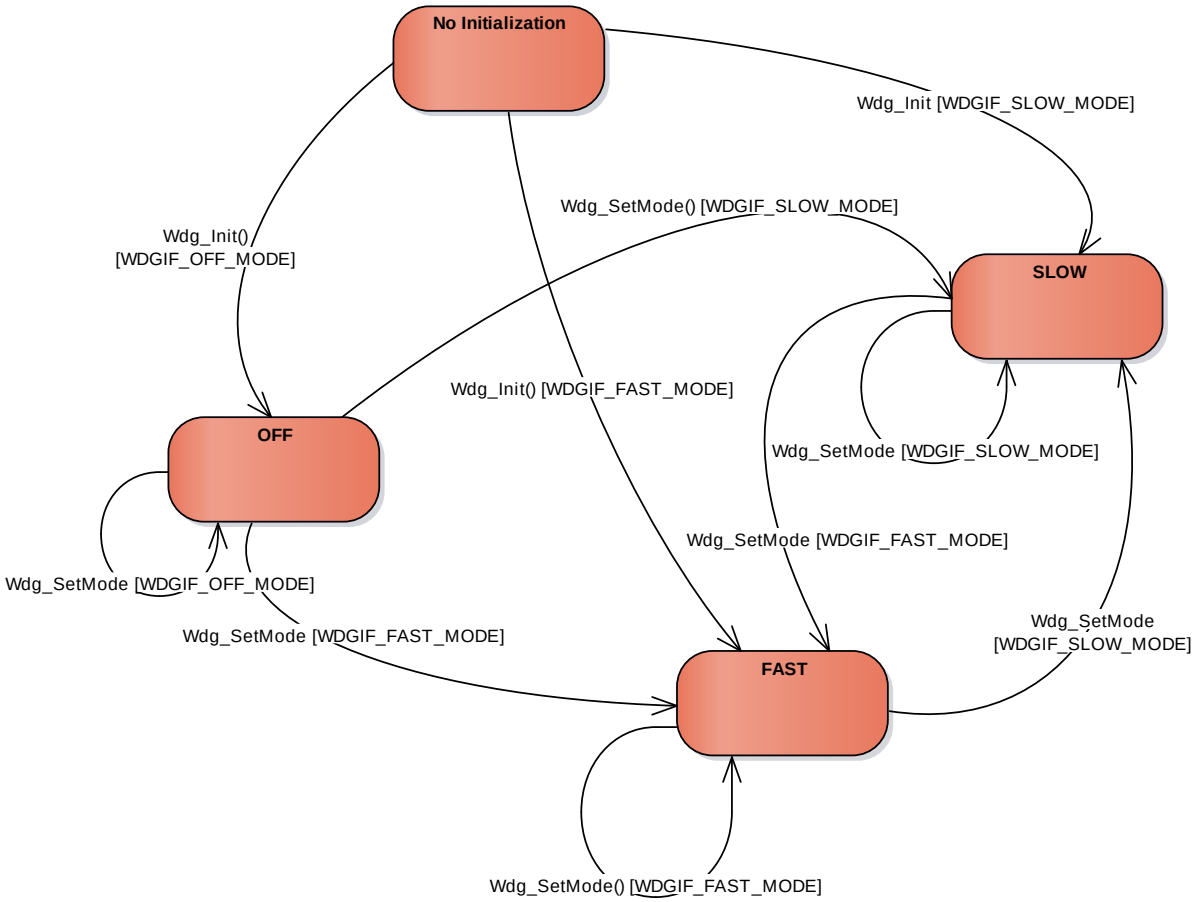


Figure 15-3 State Diagram of WDG when WdgDisableAllowed is true

WDG Driver supports the following modes when configuration parameter ‘WdgDisableAllowed’ is true.

1.WDGIF_OFF_MODE

2.WDGIF_SLOW_MODE

3.WDGIF_FAST_MODE

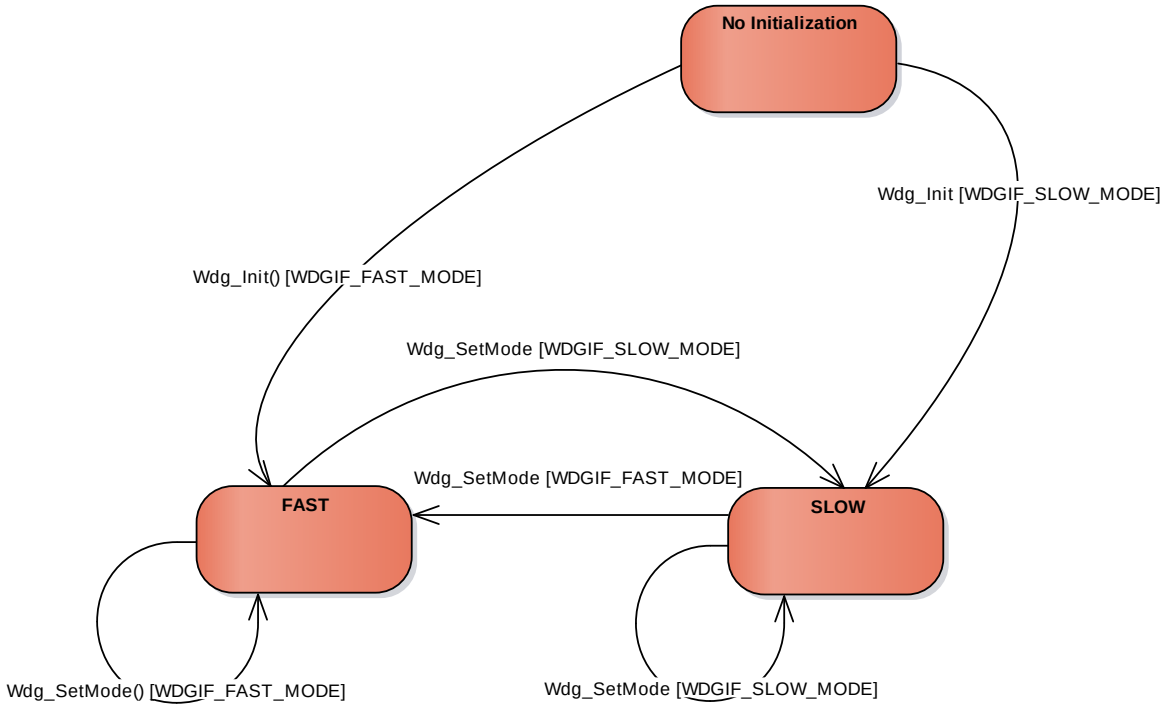


Figure 15-4 State Diagram of WDG when WdgDisableAllowed is false

WDG Driver supports the following modes when configuration parameter 'WdgDisableAllowed' is false.

1. WDGIF_SLOW_MODE

2. WDGIF_FAST_MODE

As shown in the **Figure 15-3** and **Figure 15-4**, when WDG Driver is initialized by the API Wdg_Init, the WDG Driver enters one of the modes based on the default value configured during configuration. Also, the modes can be changed by the API Wdg_SetMode only once after Wdg_Init if the current mode is WDGIF_OFF_MODE.

Note: User should re-init the external GPT timer when change mode based on the configured mode trigger timeout.

15.2.5 GPT Callback Notification for trigger WDG Driver

The callback entity is implemented in Watchdog module as Wdg_Cbk_GptNotification(). The interfaces to GPT are provided as mandatory interfaces to fulfill the triggering functionality of Watchdog module.

This is callback entry shall be invoked by GPT callback is used for servicing the hardware Watchdog. This service will be called by the Gpt Driver callback GptNotification().

If the software trigger counter (Wdg_GusTriggerCounter) is greater than zero, Wdg_ Cbk_GptNotification() shall decrease the software trigger counter and update the counter register of Watchdog.

If the software trigger counter has reached zero, the callback entry shall not trigger the hardware Watchdog

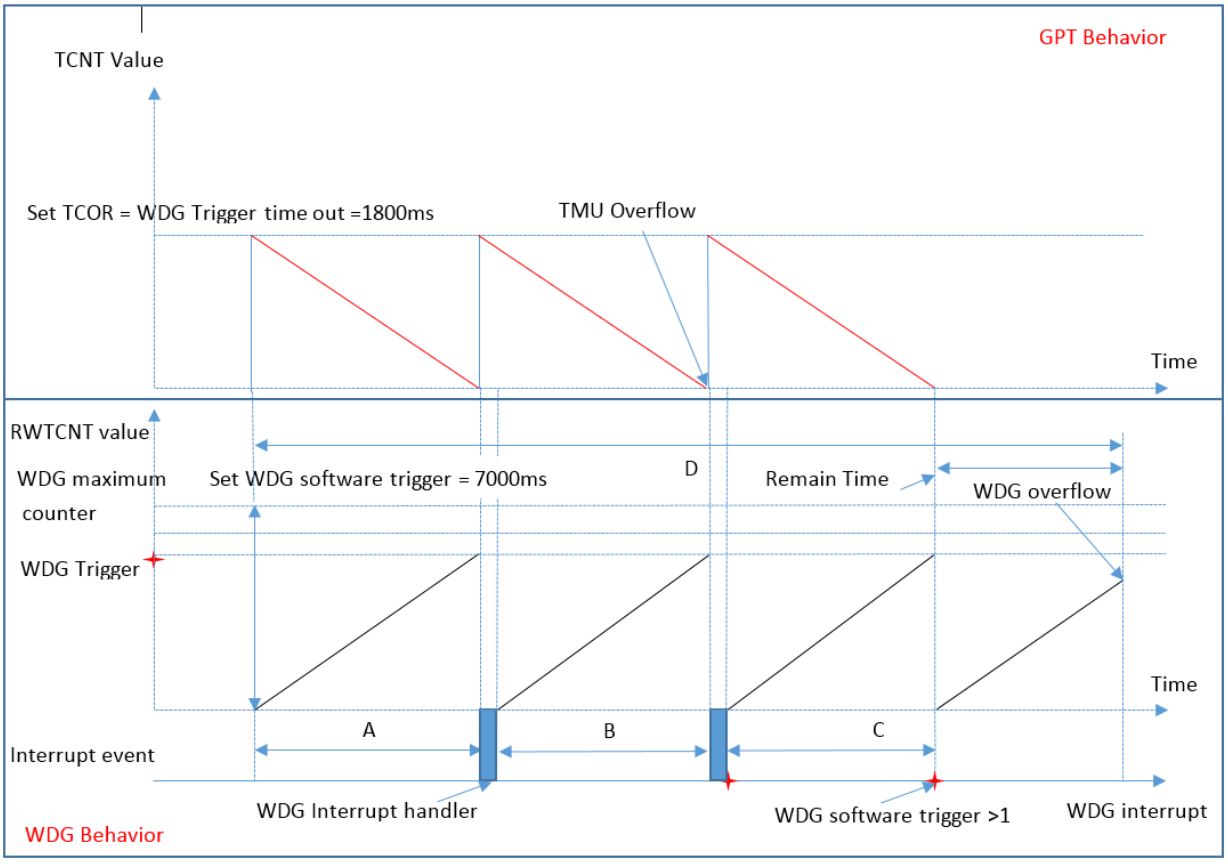
(i.e the Watchdog is not triggered and will therefore expire).

Rule for GPT configuration:

Since GPT channel is used to call WDG callback function, these parameters of GPT should configured as following:

The value configured for parameter 'GptChannelMode' in container 'GptChannelConfiguration' should be <GPT_CH_MODE_CONTINUOUS>.
The value configured for parameter 'GptNotification' in container 'GptChannelConfiguration' have not to be <NULL>.
The value configured for parameter 'GptChannelClkPrescaler' in container 'GptChannelConfiguration' shouldn't be <EXTERNAL_CLK>.
The GPT timeout of GPT channel (Refer GPT user's manual to configure GPT timeout), which used to call WDG callback function, must be configured equal trigger timeout parameter (WdgFastTriggerTimeout or WdgSlowTriggerTimeout) from WDG (fast or slow) setting configuration.
Figure 15-5 Below is the explanation for the operation of WDG with using GPT callback.

Assume that:
User configures WDG trigger time value (WdgFastTriggerTimeout) (default mode: FAST) = 1800ms
GPT trigger time is configured equal WDG trigger time value = 1800ms
Set trigger condition (By using Wdg_Set_TriggerCondition) = 7 seconds (7000ms)
The operation of WDG and GPT will be as below:



- Timeout value set by Wdg_SetTriggerCondition = D
- Red-line: GPT HW counter, Black-line: WDG HW counter
- WDG counter = WDG trigger timeout + 10% margin time

Figure 15-5 WDG and GPT behavior when Wdg_SetTriggerCondition is called.

Note:

SW trigger function of WDG driver is depended on GPT Callback Notification function. Hence, to ensure the timing of the SW trigger function user should consider using more than one channel of GPT module to implement the SW trigger function.
Besides that, user shall ensure that do not use GPT channel used by WDG for other modules at the same time.

If users want to reset timeout of WDG by using Wdg_SetTriggercondition, the timeout value should be updated before WDG counter expired. After this time, the WDG hardware can be reset.
WDG cannot control GPT full operation features (start or stop GPT). This operation should be managed by application site. To prevent the gap when user re-invokes the Wdg_SetTriggerCondition, user has to stop/start GPT to make sure WDG operations correctly.

User can set any value to WdgTimeMargin but setting to 0 is not recommended. WdgTimeMargin = 0 may lead the GPT cannot trigger WDG correctly.

15.3 Architecture Details

The WDG Driver architecture is shown in the **Figure 15-6** Watchdog Driver and Watchdog Interface Architecture. The WDG user shall directly use the APIs to configure and execute the WDG conversions:

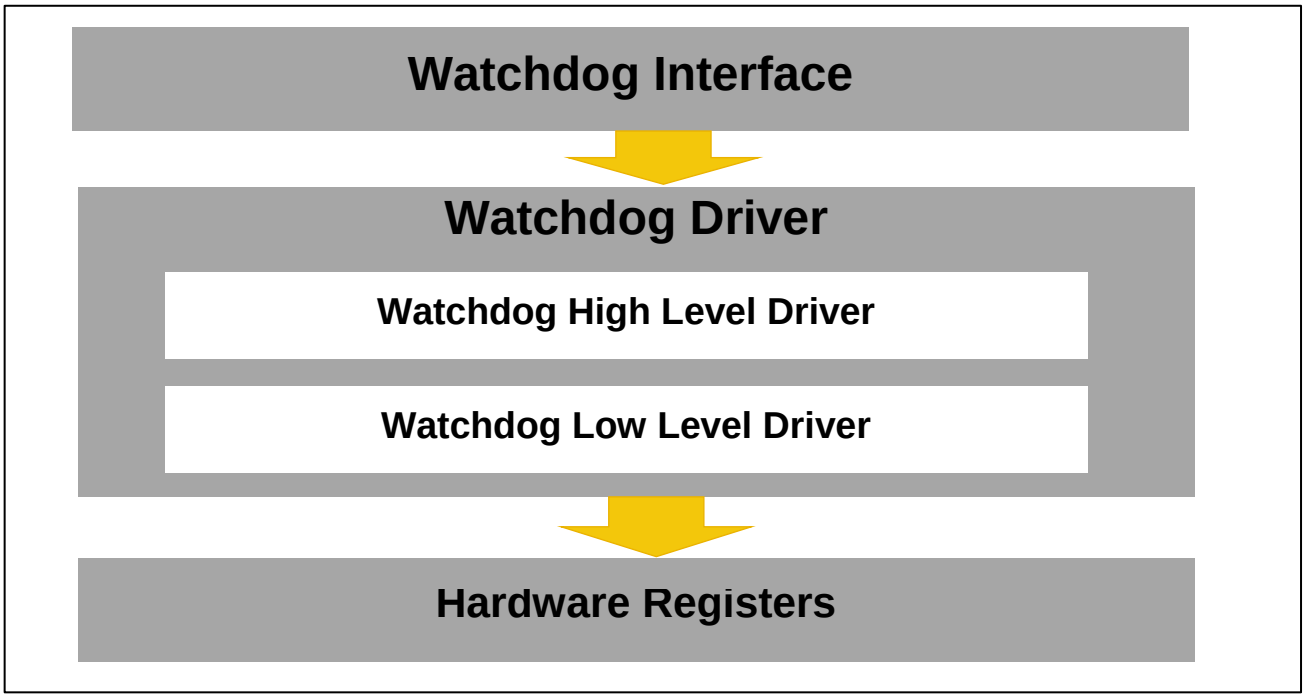


Figure 15-6 Watchdog Driver and Watchdog Interface Architecture

Watchdog Interface invokes the corresponding Driver. The Driver APIs will access the hardware register of the Watchdog Timers to change the mode and trigger the Watchdog Timer.

Watchdog Driver component:
The Watchdog Driver component is composed of the following modules:
Watchdog Driver Initialization module
Watchdog Driver SetMode module
Watchdog Driver SetTriggerCondition module
Watchdog Driver VersionInfo module

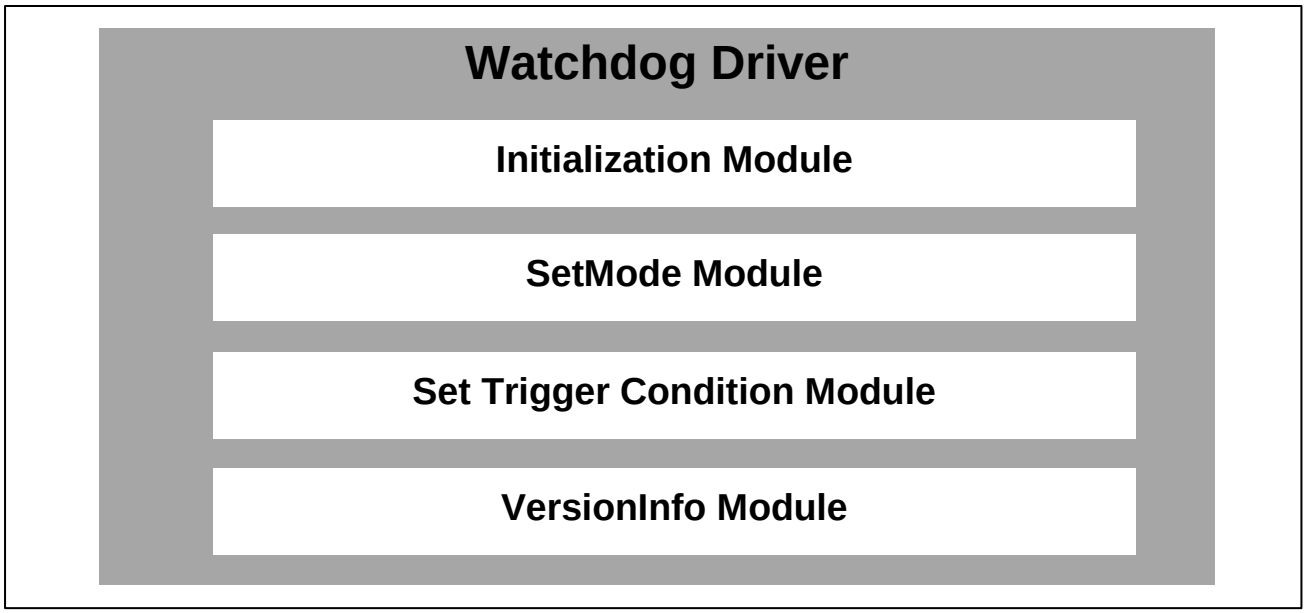


Figure 15-7 Basic Architecture of WDG Component

Watchdog Driver Initialization module:
This module initializes the watchdog driver and watchdog hardware. It provides the Wdg_Init API. This API should be invoked before the usage of any other APIs of Watchdog Driver Module.

Watchdog Driver SetMode module:
This module will handle the functionality to set the modes. It provides the Wdg_SetMode. The following mode settings are possible:

- WDGIF_SLOW_MODE
- WDGIF_FAST_MODE
- WDGIF_OFF_MODE

SetMode API will set the state of the module to WDG_BUSY during execution and reset the state to WDG_IDLE before return.

Watchdog Driver SetTriggerCondition module :
This module will handle the functionality to reset the watchdog timeout counter according to the timeout value passed. It provides the API Wdg_SetTriggerCondition.

Watchdog Driver VersionInfo module:
This module will provide the current version of the Watchdog Driver Module.
It contains the Wdg_GetVersionInfo API.

15.4 WDG Driver Component Header and Source File Description

This section explains the WDG Driver Component's C Source and C Header files. These files must be included in the project application while integrating with other modules.

The C header file generated by WDG Driver Generation Tool:

- Wdg_Cfg.h

The C source file generated by WDG Driver Generation Tool:

- Wdg_PBcfg.c

The WDG Driver Component C header files and Component source files:
Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.3.9.3 Folder Structure.

The Stub C header files:
Refer to R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" 3.2.9 Stubs File

15.5 Application Programming Interface

This section explains the Data types and APIs provided by the WDG Driver Component to the Upper layers.

15.5.1 Imported Types

This section explains the Data types imported by the WDG Driver Component and lists its dependency on other modules.

15.5.1.1 Standard Types

In this section, all types included in the Std_Types.h are listed:

- Std_ReturnType
- Std_VersionInfoType

15.5.1.2 Other Module Types

In this section, all types included in the WdgIf_Types.h and Dem.h are listed:

- WdgIf_ModeType
- Dem_EventIdType
- Dem_EventStatusType

15.5.2 Type Definitions

This section explains the type definitions of WDG Driver Component according to AUTOSAR Specification. Refer to “AUTOSAR_SWS_WatchdogDriver.pdf” for more type definitions.

15.5.2.1 Wdg_ConfigType

The Table **15-2** shows explanation of Wdg_ConfigType.

Table 15-2 Wdg_ConfigType

Name:	**Wdg_ConfigType**		
Type:	**Structure**		
Element:	**Type**	**Name**	**Explanation**
	unit32	ulStartOfDbToc	Database start value
	uint32	ulInitTimerCountVal	Trigger counter value
	uint32	ulSlowTimeValue	SLOW mode value of RWTCSRA
	uint32	ulFastTimeValue	FAST mode value of RWTCSRA register
	uint32	ulDefaultTimeValue	Value of interrupt time for Default mode in msec
	uint8	ucWdtctSlowValue	RWTCSRA register value for the Slow Mode.
	uint8	ucWdtctFastValue	RWTCSRA register value for the Fast Mode.
	uint32	ulDefaultTimeValue	Time value of either slow or fast mode in milliseconds
	uint8	ucWdtctDefaultValue	RWTCSRA register value for Watchdog default mode
	WdgIf_ModeType	ddWdtctDefaultMode	Default mode value configured by the user
	Wdg_HwFuncTableType	pHwDepFunc	Pointer to HW-dependent configuration
Description:	This is the type of the data structure required for initializing the Watchdog Hardware unit.		

15.5.3 Function Definitions

The Table 15-3 shows explanation of APIs provided by the WDG Driver Component.

Table 15-3 APIs Provided by the WDG Driver Component

Sl. No	API
AUTOSAR API	
1.	Wdg_Init
2.	Wdg_SetMode
3.	Wdg_SetTriggerCondition
4.	Wdg_GetVersionInfo
Renesas Original API	
1.	Wdg_Cbk_GptNotification

15.5.3.1 Renesas Original API

Sl. No	API
Renesas Original API	
1.	Wdg_Cbk_GptNotification

Table 15-4 Wdg_Cbk_GptNotification

Function Name:	Wdg_Cbk_GptNotification
Syntax:	FUNC (void, WDG_FAST_CODE) Wdg_Cbk_GptNotification (void)
Service ID:	0x05
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (In):	None
Parameters (In-Out):	None
Parameters (Out):	None
Return Value:	None
Description:	This is callback entry in case GPT callback is used for servicing the hardware watchdog. This service will be called by the Gpt Driver callback Gpt_CbkNotification(). If the trigger counter is greater than zero, this routine shall decrement the trigger counter and trigger the hardware watchdog. If the trigger counter has reached zero, this routine does not trigger the watchdog timer (i.e: the watchdog is not triggered and will therefore expire). This function has to set module's state to WDG_BUSY during execution and should reset the module's state to WDG_IDLE before return.

Preconditions:	The function is available if the configuration parameter WDT_RWDT_CALLBACK_API is STD_ON	
Remarks:	None	
DET/DEM Errors handled:	WDG_E_WRITE_REGISTER_FAILED	The failure of write value to registers

15.5.4 Preemption of APIs

The following table specifies the preemption of each API that can be invoked at the same time as the API.

Table 15-5 Preemption of APIs in the WDG Driver

	Wdg_Init	Wdg_SetMode	Wdg_SetTriggerCondition	Wdg_GetVersionInfo	Wdg_Cbk_GptNotification
Wdg_Init	-				
Wdg_SetMode	-	-			
Wdg_SetTriggerCondition	-	*	-		
Wdg_GetVersionInfo	√	√	√	√	
Wdg_Cbk_GptNotification	-	-	√	√	-

-: cannot be invoked at the same time

√: can be invoked at the same time

*: can be invoked at the same time with condition: Wdg_SetMode can preempt Wdg_SetTriggerCondition but Wdg_SetTriggerCondition must not preempt Wdg_SetMode.

15.6 Development and Production Errors

In this section the development errors reported by the WDG Driver Component are tabulated. The development errors will be reported only when the configuration parameter 'WdgDevErrorDetect' is enabled in the configuration.

15.6.1 WDG Driver Component Development Errors

The Table 15-6 contains the DET errors reported by WDG Driver Component. These errors are reported to Default Error Tracer Module when the WDG Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 15-6 DET Errors of WDG Driver Component

Sl. No.	**1**
Error Code	WDG_E_PARAM_MODE
Value	0x11
Related API(s)	Wdg_SetMode
Source of Error	When the API service is called the Driver is not possible to change the mode.
Sl. No.	**2**
Error Code	WDG_E_DRIVER_STATE
Value	0x10
Related API(s)	Wdg_SetMode, Wdg_SetTriggerCondition
Source of Error	If the API service is called when the driver state is not in idle state.
Sl. No.	**3**
Error Code	WDG_E_PARAM_POINTER
Value	0x14
Related API(s)	Wdg_GetVersionInfo, Wdg_Init
Source of Error	When the API service Wdg_GetVersionInfo is called with NULL pointer.
Sl. No.	**4**
Error Code	WDG_E_PARAM_TIMEOUT
Value (hex)	0x13
Related API(s)	Wdg_SetTriggerCondition
Source of Error	When the API service 'Wdg_SetTriggerCondition' is called with timeout value greater than the maximum timeout value (WdgMaxTimeout).
Sl. No.	**5**
Error Code	WDG_E_INVALID_DATABASE
Value (hex)	0xEF
Related API(s)	Wdg_Init
Source of Error	When the API service is called with wrong database.

15.6.2 WDG Driver Component Production Errors

The Table 15-7 DEM Errors of WDG Driver Component contains the DEM errors reported by the WDG Component

Table 15-7 DEM Errors of WDG Driver Component

Sl. No.	**1**
Error Code	WDG_E_DISABLE_REJECTED
Value	Depend on configuration
Related API(s)	Wdg_Init, Wdg_SetMode
Source of Error	If error during mode switch failed, the above error is reported to DEM
Sl. No.	**2**
Error Code	WDG_E_MODE_FAILED
Value	Depend on configuration
Related API(s)	Wdg_Init, Wdg_SetMode
Source of Error	When switching between the modes is failed above error is reported to DEM.
Sl. No.	**3**
Error Code	WDG_E_VALUE_COUNTER_FAILED
Value	Depend on configuration
Related API(s)	Wdg_Init, Wdg_SetMode, Wdg_SetTriggerCondition
Source of Error	When write counter register is failed above error is reported to DEM.
Sl. No.	**4**
Error Code	WDG_E_WRITE_REGISTER_FAILED
Value	Depend on configuration
Related API(s)	Wdg_Init, Wdg_SetMode, Wdg_SetTriggerCondition, Wdg_Cbk_GptNotification
Source of Error	When write to control register is failed above error is reported to DEM.

15.7 WDG Driver Component Runtime Errors

AUTOSAR does not define any runtime error code (see Chapter 7.11.2 of AUTOSAR Specification of WDG Driver (AUTOSAR_SWS_WatchdogDriver.pdf)) for WDG Driver.

15.8 Memory Organization

The following table show a typical memory organization, which must be met for proper functioning of WDG Component software.

Table 15-8 ROM Sections of the WDG Driver

Section Name	Alignment(*1)	Description
WDG_PUBLIC_CODE_ROM	-	This section contains codes which belong to the API functions of the WDG Driver.
WDG_PRIVATE_CODE_ROM	-	This section contains codes which belong to the internal functions of the WDG Driver.
CODE_FAST	-	This section contains codes which belong to the ISRs of the WDG Driver.
CONFIG_DBTOC_DATA_UNSPECIFIED	-	This section contains post-build config data table of the WDG Driver.
CONST_UNSPECIFIED	-	This section contains miscellaneous constants of the WDG Driver.

Table 15-9 RAM Sections of the WDG Driver

Section Name	Alignment(*1)	Description
VAR_NO_INIT_UNSPECIFIED	-	This section consists of the global RAM variables that are used internally by WDG Component and other software components.
VAR_INIT_UNSPECIFIED	-	This section consists of the global RAM variables of unspecified size initialized by start-up code and used internally by WDG Driver Component and other software components.
VAR_NO_INIT_PTR	-	This section consists of the global RAM pointer variable that are need not be initialized by WDG Driver. This can be located in data memory.
VAR_NO_INIT_32	-	This section consists of the global RAM variables of 32-bit size used internally by WDG Driver Component.

- All the sections written in the tables above pertain to the WDG Component only and do not include memory occupied by Wdg_PBCfg.c file generated by Watchdog Driver Generation Tool.
- User must ensure that none of the memory areas overlap with each other. Even ‘debug’ information should not overlap.

Note:
*1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the section's alignment.

15.9 Device-Specific Information

The device supports the following devices:
Refer to “R-Car Gen4 AUTOSAR R19-11 MCAL User’s Manual Modules Overview” 5.1 Product.

15.9.1 Interaction between the User and WDG Driver Component

15.9.1.1 Channel Mapping

None.

15.9.1.2 ISR Functions

None.

15.9.2 Multi-Core / Multi-Instantiation

WDG driver does not support multi-core and multi-instantiation for this device.

15.9.3 Interaction between the User and WDG Driver Component

The details of the services supported by the WDG Driver Component to the upper layer users and the mapping of the channels to the hardware units is provided in the following sections:

15.10 Non-AUTOSAR environment integration

The WDG Driver Components for Renesas R-Car Gen4 is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the WDG Driver Components to Non-AUTOSAR environment.

15.10.1 Stub modules handling

15.10.1.1 Det

The Det stub files are organized in the following folder:
\external\rel\common\generic\stubs\19-11\Det
In the AUTOSAR environment, WDG Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g WDG Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:
Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)
Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.
Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

15.10.1.2 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.
The SchM (Basic Software Scheduler) stub files are organized in the following folder:
\external\rel\common\generic\stubs\19-11\Rte\

WDG driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when WDG_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user WDG driver as following:
Write SchM functions with prototype as below:

```
void SchM_Enter_Wdg_WDG_RAM_DATA_PROTECTION(void);

void SchM_Exit_Wdg_WDG_RAM_DATA_PROTECTION(void);

void SchM_Enter_Wdg_WDG_INTERRUPT_CONTROL_PROTECTION(void);

void SchM_Exit_Wdg_WDG_INTERRUPT_CONTROL_PROTECTION(void);
```

15.10.1.3 Dem

The Dem stub files are organized in the following folder:
\external\rel\common\generic\stubs\19-11\Dem
In the AUTOSAR environment, WDG Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error e.g WDG Driver switching mode failure… The API prototype is as of follow:

Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)
Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.
Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

15.10.1.4 WdgIf

The Watchdog Interface provides uniform access to services of the underlying watchdog drivers like mode switching and setting trigger conditions.
WdgIf provide external declarations: Mode type (WdgIf_ModeType) and WDG status type (WdgIf_StatusType).
The WdgIf stub files are organized in the following folder:
\external\rel\common\generic\stubs\19-11\WdgIf\include

15.10.1.5 Mcu

The MCU driver provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required by other MCAL software modules.

In AUTOSAR environment, WDG Component Driver uses McuRCLKClk from MCU module for clock references. With Non-AUTOSAR environment, user needs to config MCU to provide clock for WDG Driver as refer: MCU_WDG_V4H. ARXML in \external\rel\V4H\common_family\config\V4H\19_11\

15.10.1.6 Gpt

In AUTOSAR environment, WDG Component Driver uses GPT channel to call WDG callback function. With Non-AUTOSAR environment, user needs to configure GPT to trigger WDG Driver as refer: GPT_WDG_V4H.arxml in \external\rel\V4H\common_family\config\V4H\19_11\

15.10.2 Callback function usage

The WDG Driver Component does not provide any callback functions.

15.10.3 Scheduled function usage

The WDG Driver Component does not provide any scheduled functions.

15.10.4 Interrupt handling usage

The WDG Driver Component does not provide any interrupt handling functions.

16.THS

16.1 Overview

The purpose of this document is to describe the information related to THS Complex Device Driver Component. This document is intended for the developers of ECU software on R-Car Series, 4th Generation SoC using Application Programming Interfaces provided by AUTOSAR specification for THS Complex Device Driver. The THS Complex Device Driver Component provides the following services:

- Initialization
- De-Initialization
- Enable interruption for a specific thermal channel
- Disable interruption for a specific thermal channel
- Configure interruption value and interruption type for the thermal channel
- Get current temperature inside LSI
- Get current voltage inside LSI
- Switch operation state of CDD THS module
- Version Information
- Clear temperature error status

The THS Complex Device Driver Component comprises of two parts: Embedded Software and Generation Tool to achieve scalability and configurability.

The purpose of this document is to describe the information related to Embedded Software part of the THS Complex Device Driver Component for R-Car Series, 4th Generation SoC. Please refer to "*THS Complex Device Driver Component Generation Tool User's Manual"* for the detail of Generation Tool part.
The below figure depicts the THS Complex Device Driver as part of layered AUTOSAR MCAL Layer:

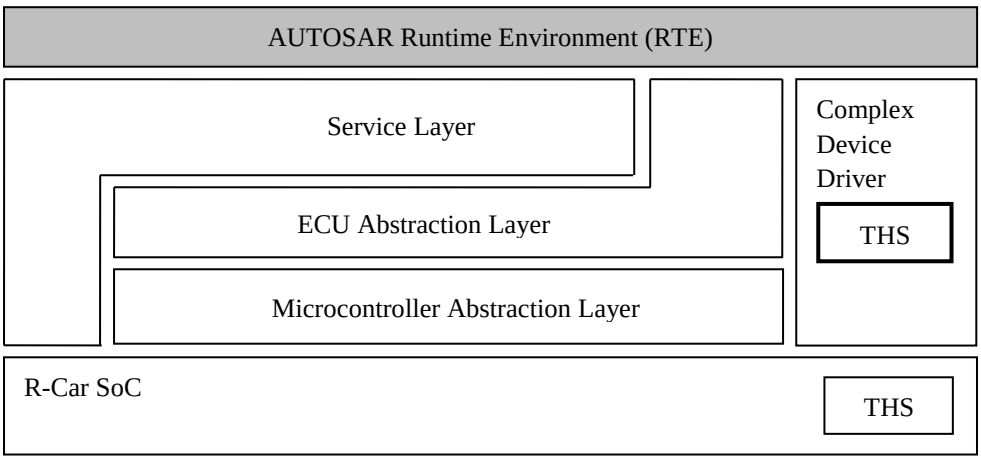


Figure 16-1 System Overview of the THS Complex Device Driver in AUTOSAR Layer Architecture

The following diagram shows the system overview of the AUTOSAR Architecture for THS Complex Device Driver:

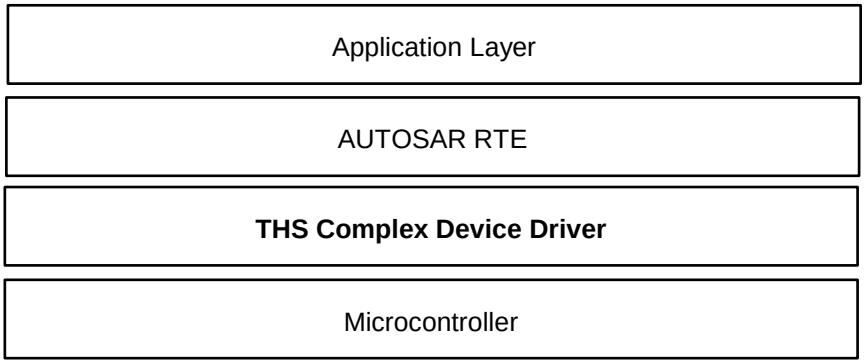


Figure 16-2 System Overview of AUTOSAR Architecture

The THS Complex Device Driver Component is Vendor Specific Module provided by Renesas to measure the temperature and voltage inside LSI.

Sample Application is available for user to understand and test THS Complex Device Driver without proper AUTOSAR upper layer, which is described in section *"Sample Application"* of this document.
Please refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* - section 3.7.2.2 How to build the Sample Application for more information on how to configure, compile and flash the THS Complex Device Driver to R-Car SoC.

16.2 Forethoughts

16.2.1 General

Following information will aid the user to use the THS Complex Device Driver Component software efficiently:

- Example code mentioned in this document shall be taken only as a reference for implementation.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- If any development errors are reported to DET, the normal flow of execution of driver will be aborted.
- All production errors will be reported to DEM by using the API Dem_SetEventStatus provided by DEM.
- The number of channels that are configured should be maintained in single post-build configuration set.
- THS Complex Driver supports one instance, therefore, configuration parameter CddInstanceId should be configured as 0. In the implementation, instance ID is always used as 0 without using CddInstanceId parameter in CDF setting.

16.2.2 Preconditions

Following preconditions must be adhered by the user, for proper functioning of the THS Complex Device Driver Component:

- The THS Complex Driver Component needs to be initialized before accepting any API requests except CddThs_GetVersionInfo() API. CddThs_Init() API should be called to initialize THS Complex Driver Component.
- The user should ensure that THS Component API requests are invoked in the correct and expected sequence along with correct input arguments.
- Validation of input parameters is done only when the static configuration parameter CddThsDevErrorDetection is enabled. Application should ensure that the right parameters are passed while invoking the APIs when CddThsDevErrorDetection is disabled.
- The files CDD_Ths_Cfg.h and CDD_Ths_Reg.h generated using THS Complex Driver Generation Tool must be linked along with THS Complex Driver Component source files.
- File CDD_Ths_PBcfg.c generated for single configuration set using THS Complex Driver Generation Tool can be compiled and linked independently.
- The application must be rebuilt, if there is any change in the CDD_Ths_Cfg.h, CDD_Ths_Reg.h files generated by the THS Complex Driver Generation Tool.
- THS Complex Driver Component support to Enable/Disable temperature error detection for specified channel by API CddThs_SetThermalInterruptionMode, config a thermal channel control values which will be used for generating error by API CddThs_ConfigureThermalInterruption. When the measured temperature exceeds the set threshold, the error signal is raised and be control by ECM module for the further action.
- User must initialize the following hardware IPs as below table before initializing THS module:

Table 16-1 List of hardware need to be initialized before initializing THS module

Hardware IP	Module Name	Expected Setting	Description
CPG	MCU	Supply of the clock signal to THS module should be enabled through the setting on the below register: - CPGWPR, PLLECR, PLL1CR0, PLL1CR1, MSTPSR9, MSTPCR9, SRCR9.	CDDTHS is designed assuming the setting of supply of the clock signal is enabled, so MCU will handle to enable clock signal. Please refer to User Manual of MCU for detail setting.
AXI-bus	-	SDRAM address mapping in HW should be configured according to Section 18.3.1 in V4H HW UM.	CDDTHS is designed assuming the setting of SDRAM address mapping is corrected, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since THS operation is under

Hardware IP	Module Name	Expected Setting	Description
			controlled by DRAM for memory control.
DBSC5	-	The maximum use of SDRAM bus bandwidth will be enabled by DBSC5 according to: - Section 33 in V4H HW UM.	CDDTHS is designed assuming the configuration of SDRAM bus bandwidth is suitable for V4H, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since THS operation is under controlled by DRAM for memory control.

16.2.3 Data Consistency

To support the re-entrance and interrupt services, the THS Complex Driver module will ensure the data consistency while accessing its own RAM storage or hardware registers. The THS Complex Driver module will use below macro.

- #define CDD_THS_ENTER_CRITICAL_SECTION(Exclusive_Area) SchM_Enter_CddThs(Exclusive_Area)
- #define CDD_THS_EXIT_CRITICAL_SECTION(Exclusive_Area) SchM_Exit_CddThs(Exclusive_Area)

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

- CDDTHS_RAM_DATA_PROTECTION
- CDDTHS_INTERRUPT_CONTROL_PROTECTION

These functions can be disabled by disabling the configuration parameter ‘CddThsCriticalSectionProtection’. The flowchart will indicate the flow with the precompile option CddThsCriticalSectionProtection.

16.3 Architecture Details

The THS Complex Driver architecture is shown in the following figure. The THS Complex Driver user shall directly use the APIs to configure and execute the THS Complex Driver operation.
The following diagram shows the THS Complex Driver Component as defined in AUTOSAR architecture.

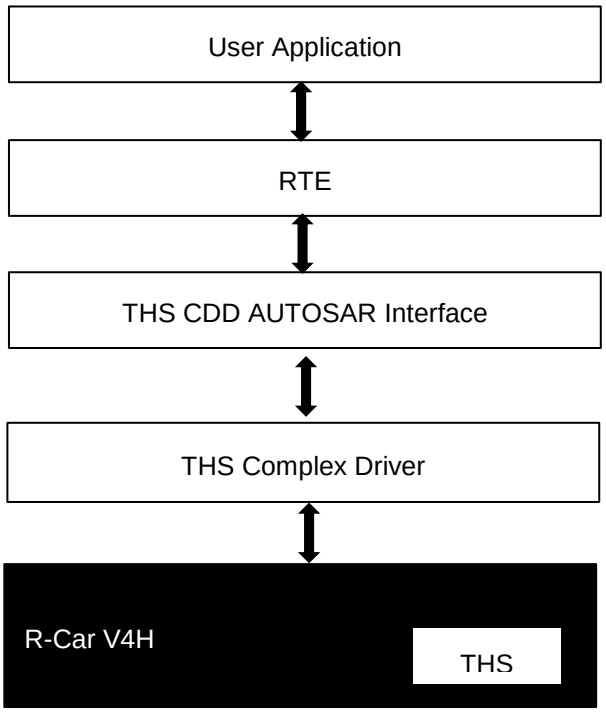


Figure 16-3 THS Complex Device Driver Component Architecture

The THS Complex Driver component should provide following services based on the functions performed by the THS Complex Driver:

- Initialization
- De-Initialization
- Set thermal interruption mode for a specific channel
- Configure a thermal channel control values which will be used for generating interruption
- Get current temperature inside the LSI
- Get current voltage inside the LSI
- Set operation state of the THS complex driver module
- Get current operation state of the THS complex driver module
- Get version information
- Clear temperature error status

Driver Initialization:

This sub module provides the structures and APIs for both global and controller specific initialization. It provides the services for initialization of the drivers before the THS Driver is put into operation. This sub module also checks if the database is flashed. The THS Complex Driver is initialized by invoking CddThs_Init() API. The THS Complex Driver module operation state after initialization depends on user's input "Cdd Ths Initial Operation State".

De-Initialization:

This sub module provides the service to de-initialize the THS Complex Driver module. This service includes setting the operation state into CDD_THS_IDLE state (HW "Standby mode", the module is off), and the driver state is UNINITIALZED. The THS Complex Driver Component is de-initialized by invoking CddThs_DeInit() API.

After De-initialization, all THS Complex Driver interrupts and notifications will be disabled.

Set thermal interruption mode for a specific channel:
This sub module provides the service to enable/disable interruption for a specific thermal channel. There are 3 thermal channels in total, and it depends on initial configuration structure, those channels are available or switched off. By default, the available channel will be enabled thermal interruption. The channel will be changed interruption mode by invoking CddThs_SetThermalInterruptionMode() API.

Configure a thermal channel control values which will be used for generating interruption:
This sub module provides the service to change the interruption control values such as type of checking interruption (upper/lower), interruption boundary value. The channel interruption control values can be changed by invoking CddThs_ConfigureThermalInterruption() API.

Get current temperature inside the LSI:
This sub module provides service to get current temperature inside the chip. The temperature is returned by invoking the CddThs_GetCurrentTemperature () API.

Get current voltage inside the LSI:
This sub module provides service to get current voltage inside the chip. The voltage is returned by invoking the CddThs_GetCurrentVoltage() API.

Set operation state of the THS complex driver module:
This sub module provides service to set the operation state of the module. The operation state is changed based on the module's features or functional. The operation state can be changed by invoking the CddThs_SetOperationState() API.

Get current operation state of the THS complex driver module:
This sub module provides service to get current operation state of the module. The current operation state is return by invoking the CddThs_GetOperationState() API.

Get Version Information:
This sub module provides the service to get the version information of THS Driver Component. The version information will be available by invoking the CddThs_GetVersionInfo() API (Implemented as macro). No HW setting is required for this function.

Clear temperature error status:
This sub module provides the service to clear the error status bit when an error occurs. The error status bit of each channel is cleared by invoking CddThs_ClearTemperatureErrorStatus() API.

16.4 THS Complex Device Driver Component Header and Source File Description

This chapter explains the THS Complex Device Driver Component's source and header files. These files must be included in the project application while integrating with other modules.
The C header file generated by THS Complex Device Driver Generation Tool:

- CDD_Ths_Cfg.h
- CDD_Ths_Reg.h

The C source file generated by THS Complex Device Driver Generation Tool:

- CDD_Ths_PBcfg.c

The THS Complex Device Driver Component C header files and Component source files: Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.4.5.3 Folder Structure.

The Stub C header files and source file: Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.2.9 Stubs File.

16.5 Application Programming Interface

This chapter explains the Data types and APIs provided by the THS Complex Device Driver Component to the Upper layers.

16.5.1 Imported Types

This chapter explains the Data types imported by the THS Complex Device Driver Component and lists its dependency on other modules.

16.5.1.1 Standard Types

In this chapter all types included from the Std_Types.h (according to "*AUTOSAR Specification of Standard Types"*) are listed:

- Std_ReturnType.
- Std_VersionInfoType.

16.5.1.2 Dem Types

In this chapter all types included from the Dem.h (according to "*AUTOSAR Specification of Diagnostic Event Manager"*) are listed:

- Dem_EventIdType.
- Dem_EventStatusType.

16.5.2 Type Definitions

This chapter explain the type definitions of THS Complex Device Driver Component according to AUTOSAR Specification.

16.5.2.1 **CddThs_ChannelConfigType**

Table 16-2 CddThs_ChannelConfigType

Name:	CddThs_ChannelConfigType		
Type:	Structure		
Elements:	ulStartOfDbToc	Contains version of configuration structure	
		Range:	Version of configuration structure
	ddInitialOperationState	Initial Operation State of the module	
		Range:	CDD_THS_NORMAL CDD_THS_IDLE
	pThermalInterruptionConfig	Pointer to thermal interruption channels configuration structure	
		Range:	&CddThs_GstThermalChannel[0]
Description:	Structure contains the initialization data for THS module.		

Note: CDD_THS_IDLE is “Standby mode” in section “13 Thermal Sensor/Chip Internal Voltage Monitor/Core Voltage Monitor (THS/CIVM/CVM)” of HW UM

16.5.2.2 **CddThs_ThermalChannel**

Table 16-3 CddThs_ThermalChannel

Name:	CddThs_ThermalChannel		
Type:	Structure		
Elements:	ddChannelId	Thermal channel ID	
		Range:	CDD_THS_THERMAL_CH0 CDD_THS_THERMAL_CH1 CDD_THS_THERMAL_CH2
	biChannelStatus	Thermal channel status	
		Range:	STD_OFF STD_ON
	ddInterruptionType	Interruption type for this particular channel	
		Range:	CDD_THS_LOWER_BOUND CDD_THS_UPPER_BOUND
	siInterruptionValue	The boundary value causes interruption	
		Range:	-40 to 125 degree Celcius
Description:	Structure contains the Thermal Channels data		

16.5.2.3 **CddThs_ThermalChannelId**

Table 16-4 CddThs_ThermalChannelId

Name:	CddThs_ThermalChannelId	
Type:	uint8	
Elements:	CDD_THS_THERMAL_CH0 = 0	Thermal Channel 0
	CDD_THS_THERMAL_CH1 = 1	Thermal Channel 1
	CDD_THS_THERMAL_CH2 = 2	Thermal Channel 2
	CDD_THS_MAX_CHANNEL_ID = 3	Number of thermal channels
Description:	This is the identification (ID) for different thermal channels.	

16.5.2.4 **CddThs_OperationState**

Table 16-5 CddThs_OperationState

Name:	CddThs_OperationState	
Type:	Enumeration	
Range:	CDD_THS_IDLE = 0	CDD THS module is off.
	CDD_THS_NORMAL = 1	CDD THS module is working.
Description:	This is the input data type of CddThs_SetOperationState, or output data from CddThs_GetOperationState	

Note: CDD_THS_IDLE is “Standby mode” in section “13 Thermal Sensor/Chip Internal Voltage Monitor/Core Voltage Monitor (THS/CIVM/CVM)” of HW UM

16.5.2.5 **CddThs_InterruptionType**

Table 16-6 CddThs_InterruptionType

Name:	CddThs_InterruptionType	
Type:	uint8	
Range:	CDD_THS_LOWER_BOUND = 0	Lower boundary interruption
	CDD_THS_UPPER_BOUND = 1	Upper boundary interruption
Description:	This type is used to identify the type of interruption value	

16.5.2.6 **CddThs_InterruptionModeType**

Table 16-7 CddThs_InterruptionModeType

Name:	CddThs_InterruptionModeType	
Type:	uint8	
Range:	CDD_THS_ENABLE_INTERRUPTION	Disable interruption for a channel
	CDD_THS_DISABLE_INTERRUPTION	Enable interruption for a channel
Description:	This is the input data type of CddThs_EnableThermalInterruption, CddThs_ConfigureThermalInterruption	

16.5.3 Function Definitions

Table 16-8 APIs provided by the THS Complex Device Driver Component

Sl. No.	APIs
AUTOSAR API	
1	CddThs_Init
2	CddThs_GetVersionInfo
3	CddThs_DeInit
RENESAS API	
4	CddThs_SetThermalInterruptionMode
5	CddThs_ConfigureThermalInterruption
6	CddThs_GetCurrentTemperature
7	CddThs_GetCurrentVoltage
8	CddThs_SetOperationState
9	CddThs_GetOperationState
10	CddThs_ClearTemperatureErrorStatus

16.5.3.1 **CddThs_SetThermalInterruptionMode**

Table 16-9 CddThs_SetThermalInterruptionMode

Name:	CddThs_SetThermalInterruptionMode		
Prototype:	FUNC (Std_ReturnType, CDD_THS_CODE_SLOW) CddThs_SetThermalInterruptionMode (CddThs_InterruptionModeType InterruptionMode CddThs_ThermalChannelId ChannelId)		
Service Id:	0x01		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	InterruptionMode	Enable/disable interruption for thermal monitor The validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	CDD_THS_ENABLE_INTERRUPTION : enable interruption CDD_THS_DISABLE_INTERRUPTION : disable interruption
	ChannelId	Thermal interruption output channel ID When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	CDD_THS_THERMAL_CH0: Thermal channel 0

			CDD_THS_THERMAL_CH1: Thermal channel 1 CDD_THS_THERMAL_CH2: Thermal channel 2
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	E_OK	Command has been accepted.	
	E_NOT_OK	Command has not been accepted for example due to initialization error.	
Description:	This function is used to enable thermal interruption for a specific thermal channel output.		
Configuration Dependency:	CddThsThermalChannelEnable CddThsThermalInterruptionType.		
Preconditions:	This function requires an execution of CddThs_Init() before it can be used.		

16.5.3.2 **CddThs_ConfigureThermalInterruption**

Table 16-10 CddThs_ConfigureThermalInterruption

Name:	CddThs_ConfigureThermalInterruption		
Prototype:	FUNC(Std_ReturnType, CDD_THS_CODE_SLOW) CddThs_ConfigureThermalInterruption (CddThs_ThermalChannelId ChannelId, CddThs_InterruptionType InterruptionType, sint16 InterruptionValue)		
Service Id:	0x02		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	ChannelId	Thermal interruption output channel ID When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	CDD_THS_THERMAL_CH0: Thermal channel 0 CDD_THS_THERMAL_CH1: Thermal channel 1 CDD_THS_THERMAL_CH2: Thermal channel 2
	InterruptionType	Boundary type to be set When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	CDD_THS_UPPER_BOUND: Upper boundary CDD_THS_LOWER_BOUND: Lower boundary

	InterruptionValue	Boundary value to be checked for generating interruption When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	- 40 to 125 unit "degree Celsius"
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	E_OK	Command has been accepted	
	E_NOT_OK	Command has not been accepted, for example due to parameter error	
Description:	This function will check for the thermal interruption mode on given channel. If the channel is enabled, then it will check for the boundary type and validating boundary value After passing all the checks, the limit value will be written to equivalent register		
Configuration Dependency:	CddThsThermalChannelEnable		
Preconditions:	This function requires an execution of CddThs_Init() before it can be used. Thermal interruption should be enabled before setting its configuration		

16.5.3.3 **CddThs_GetCurrentTemperature**

Table 16-11 CddThs_GetCurrentTemperature

Name:	CddThs_GetCurrentTemperature	
Prototype:	FUNC (Std_ReturnType, CDD_THS_CODE_SLOW) CddThs_GetCurrentTemperature (P2VAR(sint16 AUTOMATIC, CDD_THS_APPL_DATA) CurrentTemperature)	
Service Id:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	None	
Parameters (In-Out):	None	
Parameters (Out):	CurrentTemperature	Current temperature measures inside the LSI When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).

		Range:	- 40 to 125 unit "degree Celsius" (outside that range, the operation and accuracy cannot be guaranteed.)
Return Value:	E_OK	Command has been accepted	
	E_NOT_OK	Command has not been accepted. For example, module is not initialized	
Description:	Return the measured temperature inside the chip.		
Configuration Dependency:	No dependency.		
Preconditions:	This function requires an execution of CddThs_Init() before it can be used. THS in CDD_THS_NORMAL operating state		

16.5.3.4 **CddThs_GetCurrentVoltage**

Table 16-12 CddThs_GetCurrentVoltage

Name:	CddThs_GetCurrentVoltage		
Prototype:	FUNC(Std_ReturnType, CDD_THS_CODE_SLOW) CddThs_GetCurrentVoltage (P2VAR(uint16, AUTOMATIC, CDD_THS_APPL_DATA) CurrentVoltage)		
Service Id:	0x04		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	None		
Parameters (In-Out):	None		
Parameters (Out):	CurrentVoltage	Current monitored voltage inside the LSI When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	0 to 1170 mV (millivolts)
Return Value:	E_OK	Command has been accepted	
	E_NOT_OK	Command has not been accepted. For example, module is not initialized	
Description:	Return the monitored voltage inside the chip.		
Configuration Dependency:	No dependency.		
Preconditions:	This function requires an execution of CddThs_Init() before it can be used. THS in CDD_THS_NORMAL operating state		

16.5.3.5 **CddThs_SetOperationState**

Table 16-13 CddThs_SetOperationState

Name:	CddThs_SetOperationState		
Prototype:	FUNC (Std_ReturnType, CDD_THS_CODE_SLOW) CddThs_SetOperationState (CddThs_OperationState OperationState)		
Service Id:	0x05		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	OperationState	THS operating state When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	CDD_THS_NORMAL: the module is working as normal. CDD_THS_IDLE: the module is off, or not used.
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	E_OK	Command has been accepted	
	E_NOT_OK	Command has not been accepted, for example due to parameter error	
Description:	Set operating state for THS		
Configuration Dependency:	No dependency.		
Preconditions:	This function requires an execution of CddThs_Init() before it can be used.		

Note: CDD_THS_IDLE is “Standby mode” in section “13 Thermal Sensor/Chip Internal Voltage Monitor/Core Voltage Monitor (THS/CIVM/CVM)” of HW UM

16.5.3.6 **CddThs_GetOperationState**

Table 16-14 CddThs_GetOperationState

Name:	CddThs_GetOperationState
Prototype:	FUNC(void, CDD_THS_CODE_SLOW) CddThs_GetOperationState (P2VAR(CddThs_OperationState,AUTOMATIC,CDD_THS_APPL_DATA) CurrentOperationState)

Service Id:	0x06		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	None		
Parameters (In-Out):	None		
Parameters (Out):	CurrentOperationState	THS current operation state When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	CDD_THS_ NORMAL: the module is working as normal. CDD_THS_IDLE: the module is off, or not used.
Return Value:	None		
Description:	Return the operating mode from reading register value		
Configuration Dependency:	No dependency.		
Preconditions:	This function requires an execution of CddThs_Init() before it can be used.		

Note: CDD_THS_IDLE is “Standby mode” in section “13 Thermal Sensor/Chip Internal Voltage Monitor/Core Voltage Monitor (THS/CIVM/CVM)” of HW UM

16.5.3.7 **CddThs_ClearTemperatureErrorStatus**

Table 16-15 CddThs_ClearTemperatureErrorStatus

Name:	CddThs_ClearTemperatureErrorStatus		
Prototype:	FUNC(void, CDD_THS_CODE_SLOW) CddThs_ClearTemperatureErrorStatus (CddThs_ThermalChannelId ChannelId)		
Service ID:	0x0C		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	ChannelId	Thermal interruption output channel ID When CddThsDevErrorDetection parameter is true, API check whether input parameter is invalid or not. Otherwise, the validity of input parameter shall be guaranteed by user. This parameter is mandatory to be provided correctly (it cannot be omitted).	
		Range:	CDD_THS_THERMAL_CH0: Thermal channel 0 CDD_THS_THERMAL_CH1: Thermal channel 1 CDD_THS_THERMAL_CH2: Thermal channel 2

Parameters (In-Out):	None
Parameters (Out):	None
Return Value:	None
Description:	This function is used to clear the error status bit when an error occurs
Configuration Dependency:	CddThsThermalChannelEnable
Preconditions:	The CDD THS Driver must be initialized. Thermal interruption of corresponding channel should be enabled.

16.5.4 Preemption of APIs

The following table shows the list of Preemption Table of APIs of the THS Driver

Table 16-16 Preemption Table of APIs of the THS Driver

	CddThs_Init	CddThs_GetVersionInfo	CddThs_DeInit	CddThs_SetThermalInterruptionMode	CddThs_ConfigureThermalInterruption	CddThs_GetCurrentTemperature	CddThs_GetCurrentVoltage	CddThs_SetOperationState	CddThs_GetOperationState	CddThs_ClearTemperatureErrorStatus
CddThs_Init	-	√	-	-	-	-	-	-	-	-
CddThs_GetVersionInfo	√	√	√	√	√	√	√	√	√	√
CddThs_DeInit	-	√	-	-	-	-	-	-	-	-
CddThs_SetThermalInterruptionMode	-	√	-	-	-	√	√	-	√	-
CddThs_ConfigureThermalInterruption	-	√	-	-	-	√	√	-	√	-
CddThs_GetCurrentTemperature	-	√	-	√	√	√	√	-	√	√
CddThs_GetCurrentVoltage	-	√	-	√	√	√	√	-	√	√
CddThs_SetOperationState	-	√	-	-	-	-	-	-	-	-
CddThs_GetOperationState	-	√	-	√	√	√	√	-	√	√
CddThs_ClearTemperatureErrorStatus	-	√	-	-	-	√	√	-	√	√

-: cannot be invoked at the same time
√: can be invoked at the same time

16.6 Development and Production Errors

In this chapter, the development errors that are reported by the THS Complex Device Driver Component are tabulated. The development errors will be reported only when the pre-compiler option

CddThsDevErrorDetection is enabled in the configuration. The production code errors are not supported by THS Complex Device Driver Component.

16.6.1 THS Complex Device Driver Component Development Errors

The following table contains the DET errors that are reported by THS Complex Device Driver Component. These errors are reported to DET Module when the THS Complex Device Driver Component APIs is invoked with wrong input parameters or without initialization of the driver.

Table 16-17 DET Errors of THS Complex Device Driver Component

Sl. No.	**1**
Error Code	CDD_THS_E_UNINIT
Value (hex)	0x0A
Related API(s)	CddThs_SetThermalInterruptionMode, CddThs_ConfigureThermalInterruption, CddThs_GetCurrentTemperature, CddThs_GetCurrentVoltage, CddThs_SetOperationState, CddThs_GetOperationState, CddThs_DeInit,
Source of Error	When the APIs are invoked without the initialization of the THS Complex Driver Component.
Sl. No.	**2**
Error Code	CDD_THS_E_INVALID_VALUE
Value (hex)	0x0C
Related API(s)	CddThs_ConfigureThermalInterruption
Source of Error	Input boundary value is not in supported range.
Sl. No.	**3**
Error Code	CDD_THS_E_DISABLED_CHANNEL
Value (hex)	0x10
Related API(s)	CddThs_SetThermalInterruptionMode, CddThs_ConfigureThermalInterruption, CddThs_ClearTemperatureErrorStatus
Source of Error	Channel is disabled at initialization.
Sl. No.	**4**
Error Code	CDD_THS_E_INVALID_CHANNEL
Value (hex)	0x0E
Related API(s)	CddThs_SetThermalInterruptionMode, CddThs_ConfigureThermalInterruption, CddThs_ClearTemperatureErrorStatus
Source of Error	Channel is configured as unavailable at initialization.
Sl. No.	**5**
Error Code	CDD_THS_E_ALREADY_INITIALIZED
Value (hex)	0x0B
Related API(s)	CddThs_Init
Source of Error	The THS CDD is already initialized.
Sl. No.	**6**
Error Code	CDD_THS_E_PARAM_POINTER
Value (hex)	0x0D

Related API(s)	CddThs_Init, CddThs_SetThermalInterruptionMode, CddThs_ConfigureThermalInterruption, CddThs_GetCurrentTemperature, CddThs_GetCurrentVoltage, CddThs_GetOperationState, CddThs_GetVersionInfo,
Source of Error	Parameter (In) is a null pointer.
Sl. No.	**7**
Error Code	CDD_THS_E_INVALID_DATABASE
Value (hex)	0x0F
Related API(s)	CddThs_Init
Source of Error	Database is failed
Sl. No.	**8**
Error Code	CDD_THS_E_INVALID_PARAM
Value (hex)	0x12
Related API(s)	CddThs_SetThermalInterruptionMode, CddThs_ConfigureThermalInterruption
Source of Error	Input parameter is out of range
Sl. No.	**9**
Error Code	CDD_THS_E_STATE_NOT_ACTIVE
Value (hex)	0x11
Related API(s)	CddThs_GetCurrentTemperature, CddThs_GetCurrentVoltage
Source of Error	Operation state not active

16.6.2 THS Complex Device Driver Component Production Errors

The following table contains Renesas specific DEM errors that are reported by THS Complex Device Driver Component.

Table 16-18 DEM Errors of THS Complex Device Driver Component

Sl. No.	**1**
Error Code	CDD_THS_E_WRITEVERIFY_FAILURE
Related API(s)	CddThs_WriteVerifyCheck
Source of Error	CDD THS fails to set register value.

16.7 THS Driver Component Runtime Errors

AUTOSAR does not define any runtime error code for THS Driver.

16.8 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of THS Complex Device Driver Component software.

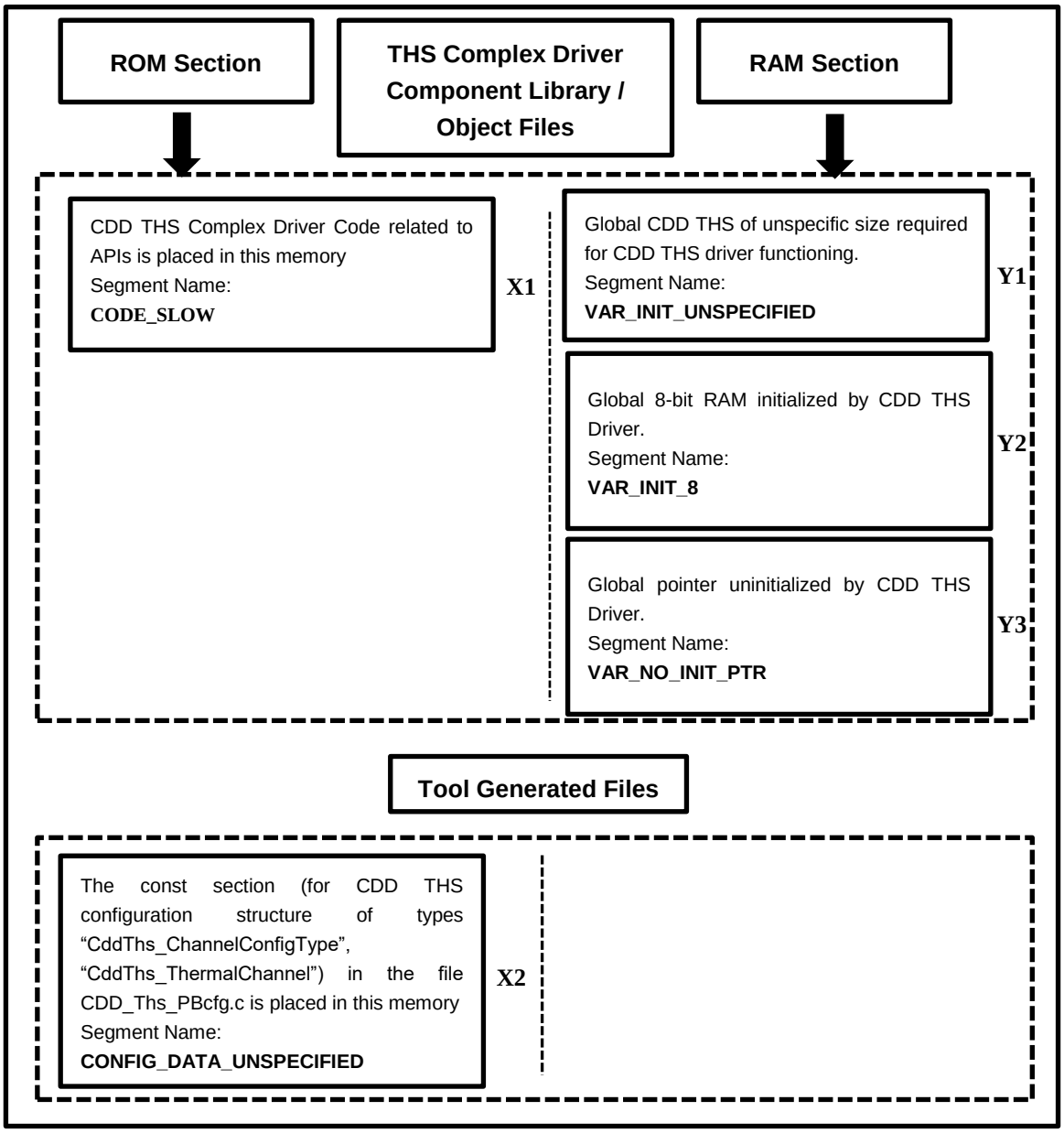


Figure 16-4 THS Complex Device Driver Component Memory Organization

ROM Section (X1, X2):

CODE_SLOW (X1): API(s) of THS Complex Driver Component, which located in code memory.

CONFIG_DATA_UNSPECIFIED (X2): This section consists of THS Complex Driver Component database table of contents generated by the THS Complex Driver Component Generation Tool. This located in code memory.

Note: ROM in R-Car is the Read-only Section in memory (DRAM).

RAM Section (Y1, Y2 and Y3):

VAR_INIT_UNSPECIFIED (Y1): This section consists of the global RAM variables of unspecified size that are used internally by THS Complex Driver Component. This located in data memory.

VAR_INIT_8 (Y2): This section consists of the global RAM variables of 8-bit size that are used internally by THS Complex Driver Component. This located in data memory.

VAR_NO_INIT_PTR (Y3): This section consists of the global pointer variables that are used internally by THS Complex Driver Component. This located in data memory.
Note: User must ensure that none of the memory areas overlap with each other. Even 'debug' information should not overlap.

16.9 Device Specific Information

The device supports the following devices:
Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 5.1 Product.

16.9.1 Interaction Between the User and THS Complex Device Driver Component

The detail of the services supported by the THS Complex Device Driver Component to the upper layers users is provided in the following chapter:

16.9.1.1 Channel Mapping

Table 16-19 Hardware CDDTHS channel mapping

No.	Hardware Channel
0	ThermalSensor.ch0
1	ThermalSensor.ch1
2	ThermalSensor.ch2

16.9.1.2 ISR Functions

None.

16.9.1.3 Translation header file

Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.2.4 Translation Header File.

16.9.1.4 Parameter Definition File

Refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.2.3 Parameter Definition File

16.9.2 Compiler, Linker and Assembler

The details of Compiler and Linker and Assembler used for building the THS Complex Device Driver Component is described to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* – chapter 5. Required Operation Conditions.

16.9.3 Multi-Core / Multi-Instantiation

THS driver does not support multi-core and multi-instantiation for this device.

16.10 Non-AUTOSAR environment integration

The THS Complex Driver Components for Renesas R-Car Gen4 SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the THS Complex Driver Components to Non-AUTOSAR environment.

16.10.1 Stub modules handling

16.10.1.1 Det

The Det stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, THS Complex Driver Components uses Det_ReportError API provided by the DET module to report a development error. E.g: THS Complex Driver has not been initialized, API is provided with invalid parameter. The API prototype is as of follow:
Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

16.10.1.2 Dem

The Dem stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, THS Complex Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error. The API prototype is as of follow:
Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

16.10.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.
The SchM (Basic Software Scheduler) stub files are organized in the following folder:
rel\common\generic\stubs\<Autosar version>\Rte

The THS Complex Driver Component needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when CddThsCriticalSectionProtection parameter is configured as true. With Non-AUTOSAR environment, user needs to prepare SchM stub to user THS Complex Driver Component as following.

Write SchM functions with prototype as below:

- void SchM_Enter_CddThs_CDDTHS_RAM_DATA_PROTECTION (void);
- void SchM_Exit_CddThs_CDDTHS_RAM_DATA_PROTECTION (void);
- void SchM_Enter_CddThs_CDDTHS_INTERRUPT_CONTROL_PROTECTION (void);
- void SchM_Exit_CddThs_CDDTHS_INTERRUPT_CONTROL_PROTECTION (void);

16.10.2 Callback function usage

The THS Complex Driver Component does not have callback function.

16.10.3 Scheduled function usage

The THS Complex Driver Component does not have Scheduled function.

16.10.4 Interrupt handling usage

The THS Complex Driver Component does not have Interrupt handling.

17.IPMMU

17.1 Overview

This document is intended for the developers of ECU software on R-Car Series, 4th Generation SoC using Application Programming Interfaces provided by AUTOSAR specification for IPMMU Complex Driver.

The IPMMU is a Memory Management Unit (MMU) which provides address translation and access protection functionalities to processing units and interconnect networks.

IPMMU includes the following main services:

- Initialization
- Address translation functionality with supported domains, include:
  - Enable/Disable designating MMU/PMB
  - Set the base address of translation table and attribute for designating MMU
  - Set operation mode for designating MMU
  - Flush the designating TLB
- Caching recently used page table entries in TLB
- Get version information

The IPMMU Complex Driver Component comprises of two parts: Embedded Software and the Generation Tool to achieve scalability and configurability.

The purpose of this document is to describe the information related to Embedded Software part of the IPMMU Complex Driver Component for R-Car Series, 4th Generation SoC.

The below figure depicts the IPMMU Complex Device Driver as part of layered AUTOSAR MCAL Layer:

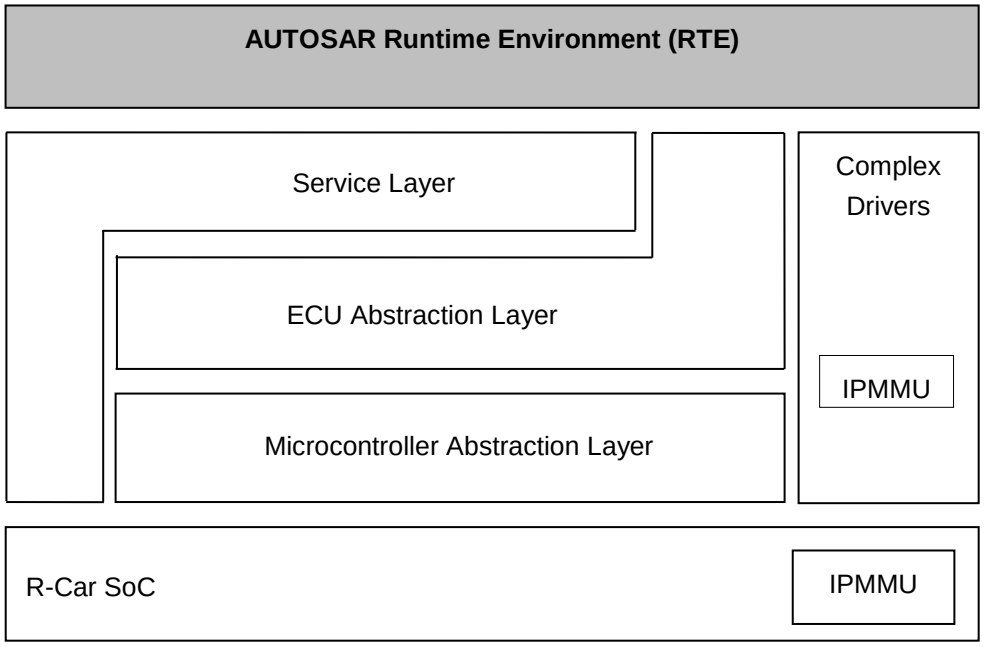


Figure 17-1 System Overview of the IPMMU Complex Driver in AUTOSAR Software Layer

The following diagram shows the system overview of the AUTOSAR Architecture.

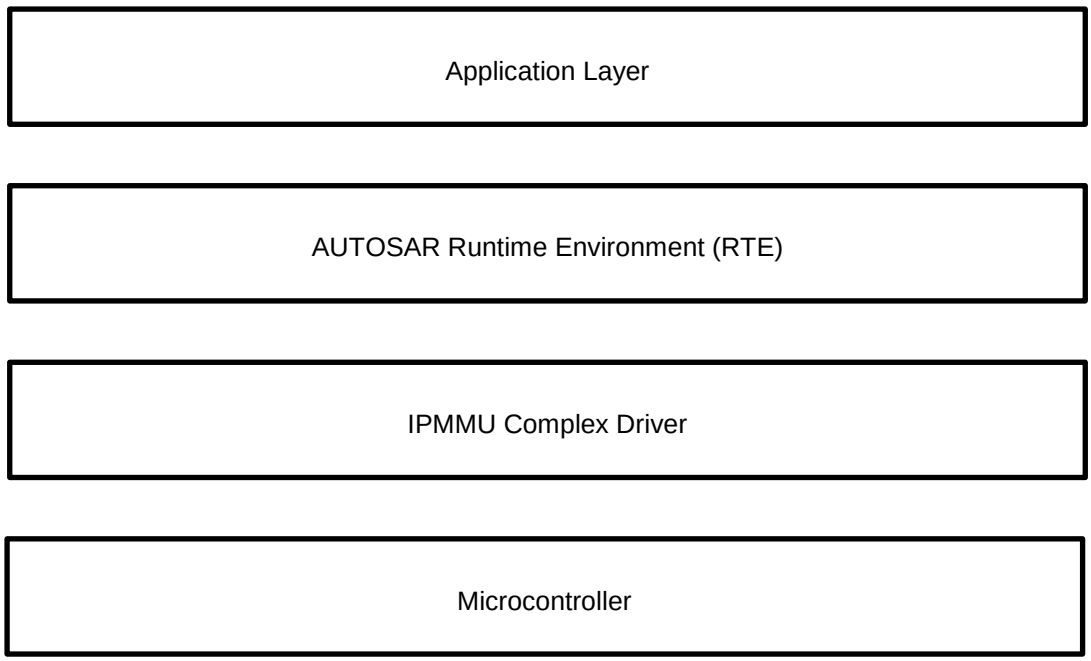


Figure 17-2 System Overview of AUTOSAR Architecture

The IPMMU CDD Component is Vendor Specific Module provided by Renesas to communicate with Non-AUTOSAR domain of AP-System Core of R-Car SoC.

Sample Application is available for user to understand and test IPMMU Complex Driver without proper AUTOSAR upper layer. Please refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" - section "*3.7 Sample Application*" for the more information.

17.2 Forethoughts

17.2.1 General

Following information will support the user to use the IPMMU Complex Driver Component software efficiently:

- Renesas assumes that software runs on Cortex-R52 is developed at the highest ASIL level, therefore, Renesas recommends the user that Cortex-R52 software should set all IPMMU registers (accessed by IPMMU Driver) shared between Cortex-R52 and Cortex-A76 and related to target HW-IPs. This recommendation is also applied in case Cortex-A76 Software and the Cortex-R52 Software have the same ASIL level. For example, the target Hardware IP Ether-AVB (Ether-AVB channel 0 and channel 1) is shared between Cortex-R52 Software and Cortex-A76 Software. The Cortex-A76 software wants to use MMU1 and PMB1 to translate the address to the target Hardware IP Ether-AVB channel 0 and 1 of the IPMMU-HC domain. Because the Ether-AVB channel 0 and channel 1 are also used by the Cortex-R52 software, then MMU1, PMB1, uTLB0 and uTLB1 of IPMMU-HC domain should be configured by Cortex-R52 software. Finally, both ETH-AVB channels 0 and 1 of software run on Cortex-R52 and Cortex-A76 can use IPMMU (after IPMMU is initialized). To configure MMU1, PMB1, uTLB0 and uTLB1 with the IPMMU-HC domain, the user should correspondingly set parameters inside CddIpmmuMmu, CddIpmmuMmuTranslationTable,

CddIpmmuPmb and CddIpmmuTlb containers. In the Cortex-A76 Software, the user should not set up the IPMMU driver again.

- In case the user wants to set up the IPMMU driver for 3 Cortex-R52 cores, Renesas recommends the user set up the IPMMU driver on the single Cortex-R52 core, which software is at the highest ASIL level. And the other Cortex-R52 cores don't need to set up the IPMMU driver again.
- Example code mentioned in this document shall be taken only as a reference for implementation.
- IPMMU Complex Driver is used to interface with non-AUTOSAR application domain. Usage of IPMMU Complex Driver requires proper IPMMU driver/application from non-AUTOSAR application domain.
- MCU specific initializations such as reset registers, one-time writable registers, interrupt stack pointer, user stack pointer, internal watchdog, specific features of internal memory and registers is not implemented by IPMMU Complex Driver. These initializations must be implemented by the start-up code.
- The ISR functions and the corresponding handler addresses are provided in section "*17.9.1.2 ISR Functions*". User should ensure that Interrupt Vector table configuration is done as per the information provided in the table.
- Porting specific information such as compiler, linker details are provided in section "*17.9 Device Specific Information*".
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- If any development errors are reported to DET, the normal flow of execution of driver will be aborted.
- All production errors will be reported to DEM by using the API Dem_SetEventStatus provided by DEM.
- The IPMMU Complex Driver only supports the 40-bit translation System with the ARMv8 Long-descriptor translation table format. Short-descriptor translation table format will not be supported.
- The IPMMU Complex Driver supports VMSAv8-32 mode.
- The IPMMU Complex Driver supports one instance, therefore, the configuration parameter CddInstanceId should be configured as 0. In the implementation, the instance ID is always used as 0 without using CddInstanceId parameter in CDF setting.
- The IPMMU Complex Driver does not support 2-stage translation. Only 1-stage translation is supported.
- The assignment of micro-TLB varies with the RCar-V4H products. Refer to the section "*19.3.10 micro-TLB Assignment*", in "*R-Car V4H Series User's Manual: Hardware*".
- All interrupt-related functions are mapped to CODE_FAST section (see "*Table 7.12*" in "*AUTOSAR CP R19-11 Specification of Memory Mapping*"). Other functions are mapped to CODE_SLOW section (see "*Table 7.13*" of "*AUTOSAR CP R19-11 Specification of Memory Mapping"*).
- The IPMMU Complex Driver is only supported and tested by non-secure mode.

17.2.2 Preconditions

Following preconditions must be adhered by the user, for proper functioning of the IPMMU Complex Driver Component:

- The CDD_Ipmmu_PBcfg.c, CDD_Ipmmu_Cbk.h, CDD_Ipmmu_Cfg.h, CDD_Ipmmu_Hardware.h files generated by the CDD IPMMU Driver Component Code Generation Tool must be compiled and linked along with CDD IPMMU Driver Component source files.
- The application must be rebuilt, if there is any change in the CDD_Ipmmu_PBcfg.c, CDD_Ipmmu_Cbk.h, CDD_Ipmmu_Cfg.h, CDD_Ipmmu_Hardware.h files generated by the CDD IPMMU Driver Component Code Generation Tool.
- File CDD_Ipmmu_PBcfg.c generated for single configuration set using CDD IPMMU Driver Component Code Generation Tool can be compiled and linked independently.
- The IPMMU Complex Driver Component needs to be initialized before accepting any API requests. CddIpmmu_Init API should be called to initialize IPMMU Complex Driver Component.

- The user should ensure that CDD IPMMU Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- ISR priority should be configured to be greater than priority of the thread/task which invoke the driver's APIs. Otherwise, the driver operation is not guaranteed.
- Validation of input parameters is done only when the static configuration parameter CDDIPMMU_DEV_ERROR_DETECT is enabled. Application should ensure that the right parameters are passed while invoking the APIs when CDDIPMMU_DEV_ERROR_DETECT is disabled.
- A mismatch in the version numbers will result in compilation error. Ensure that the correct versions of the header and the source files are used.
- When attribution of memory region contains MMU translation table is changed and cache of the memory region is enabled. The cache should be maintained to ensure that any cached copies of affected locations are removed from the caches, typically by cleaning and invalidating the locations from the levels of cache that might hold copies of the locations affected by the attribute change.
- User should use Generation Tool to generate the configuration of IPMMU Complex driver. If not please refer to "*R-Car V4H AUTOSAR R19-11 MCAL User's Manual: Driver Component Generation Tool User's Manual*" for more detail about constraints of parameter.
- User should only enable Micro TLB for used master domain after PMB or MMU which will translate address for used domain is configured correctly. About MMU setup, the translation table should be configured before MMU is enabled. For more detail about translation table, please refer to ARM architecture, VMSAv8.
- When use the IPMMU in the power domain IPMMU-IR which is not "Always-On", the corresponded power domain should be enabled before use CDD IPMMU.
- Besides, parameter CddIpmmuIRDomainSupport should be enabled when domain IPMMU-IR is used. In case of error interrupt, if IPMMU-IR domain is used and CddIpmmuIRDomainSupport is disabled, the error detection for this domain is not performed. On the contrary, user should not enable CddIpmmuIRDomainSupport when IPMMU-IR domain is not used.
- User should not configure or change the status for the same MMU to prevent the conflict in multiple tasks. Besides, CddIpmmu_MmuGetMemAttr should not be invoked concurrently with CddIpmmu_MmuSetMemAttr because incorrect data can be returned by CddIpmmu_MmuGetMemAttr. Additionally, CddIpmmu_MicroTlbSet and CddIpmmu_MicroTlbFlush should not be invoked concurrently since the conflict may occurs.
- Due to Hardware limitation, configuration of parameter CddIpmmuMmuTranslationSize should be less than 5. The virtual address space range of MMU translation table less than or equal 128MB is not supported. A translation fault error will be erroneously detected when TLB flushing if the virtual address space of MMU translation table is less than or equal 128MB. For VMSAv8-32 mode, the size offset of the TTBR0/1 addressed region should not be set as 5, 6 or 7.
- Due to Hardware limitation, the unused MMU context number should be set to IMUCTRn.TTSEL [2:0] for the master which is disabled address transfer by IPMMU. CDDIPMMU supports MMU1 to MMU15. MMU0 should not be used with enable setting because the default value of IMUCTRn.TTSEL [2:0] is MMU0, therefore CDD IPMMU defined MMU0 for unused MMU context number. User should set "TTSEL = MMU0, MMUEN = disable" for each unused uTLB by IMUCTRn. This limitation effects chapter "*17.5.5 Additional Error Handling and Restrictions*".
- Due to Hardware limitation, during the power shutting off the power domain which is not "Always-On", address translation by IPMMU in same power domain should not run. When the IPMMU in the power domain which is not "Always-On" is enabled by CDD IPMMU, user should check that the master-module in same power domain is inoperative state by software before power shut off the same power domain.
- Renesas does not test all possible supported IPMMU domains and masters. Adequate testing for domains

and masters (except RT-DMA, SYS-DMA on IPMMU_RT1 and IPMMU_DS0 domain) need to be performed by the user after MCAL/CDD integration.

- User must initialize the following hardware IPs as below before initializing IPMMU module.

Table 17-1 List of Hardware IP affect to IPMMU

Hardware IP	Module Name	Expected Setting	Description
SYSC	None	Power Domain (except Always-on domain) of IPMMU should be turned on followed power-up sequence according to Section "*11.3.3*" and "*11.3.4*" in "*R-Car V4H Series User's Manual_Global*".	IPMMU Complex Driver support IPMMU in different domains. When user use IPMMU-IR, domain A3IR needs to be turned on before initialization of IPMMU, so they should be configured like that by user in boot loader/ stub/ above layer before using this domain. Otherwise, operation of driver is not guaranteed
RST	None	Value of all registers of IPMMU will return to initialized state after reset sequence according to sheet "*RegisterDescription" in "R-CarV4H_UM_019_IPMMU.xlsx*"	User needs to ensure that IPMMU Complex Driver will be re-initialized after system is reset. Otherwise, operation of driver is not guaranteed.
AXI-bus	None	SDRAM address mapping in HW should be configured according to Section "*18.3.1*" in HW UM "*R-Car V4H Series User's Manual Global*".	IPMMU is designed assuming the setting of SDRAM address mapping is corrected, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since IPMMU operation is under controlled by DRAM for memory control.
DBSC5	None	The maximum use of SDRAM bus bandwidth will be enabled by DBSC5 according to Section "*33.4.1*" in HW UM "*R-Car V4H Series User's Manual_Global*".	IPMMU is designed assuming the configuration of SDRAM bus bandwidth is suitable for RCar-V4H, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since IPMMU operation is under controlled by DRAM for memory control.0

Life Cycle	None	Life Cycle setting for the assignment group of IPMMU must be accessed by CR52 master group.	IPMMU is designed assuming the setting of this protection mechanism allow register to be accessed by CR52 master group, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since IPMMU may not access to registers of IPMMU.

17.2.3 Data Consistency

To support the re-entrance and interrupt services, the IPMMU Complex Device Driver Component will ensure the data consistency while accessing its own hardware registers. The IPMMU Complex Component will use SchM_Enter_CddIpmmu_<*Exclusive Area*> and SchM_Exit_CddIpmmu_<*Exclusive Area*> functions. The SchM_Enter_CddIpmmu_<*Exclusive Area*> function is called before accessing the data needs to be protected and SchM_Exit_CddIpmmu_<*Exclusive Area*> function is called after the data is accessed.

The IPMMU Complex Component module will use the following macros:

```
#define CDDIPMMU_ENTER_CRITICAL_SECTION(Exclusive Area)
SchM_Enter_CddIpmmu_##Exclusive_Area()
#define CDDIPMMU_EXIT_CRITICAL_SECTION(Exclusive Area)
SchM_Exit_CddIpmmu_##Exclusive_Area()
```

The following exclusive area along with scheduler services is used to provide data integrity for shared resources: "*CDDIPMMU_INTERRUPT_CONTROL_PROTECTION*".

These functions can be disabled by disabling the configuration parameter '*CddIpmmuCriticalSectionProtection*'.

17.3 Architecture Details

The IPMMU Complex Driver architecture is shown in the following figure. The IPMMU Complex Driver user shall directly use the APIs to configure and execute the IPMMU Complex Driver operation.

The following diagram shows the IPMMU Complex Driver Component as defined in AUTOSAR architecture:

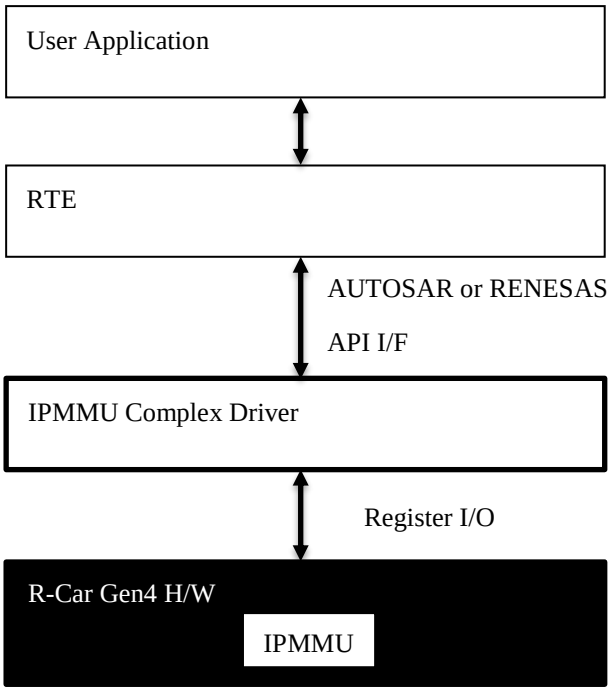


Figure 17-3 Driver Architecture

IPMMU Complex Driver Component:

The IPMMU Complex Driver provides services for controlling Memory Management Unit (MMU) which provides address translation and access protection functionalities to processing units and interconnect networks.

The IPMMU Complex Driver Component is divided into the following sub feature based on the functionality required:

- Initialization
- Address translation functionality include:
  - Enable/Disable designating MMU/PMB
  - Set the base address of translation table and attribute for designating MMU
  - Set operation mode for designating MMU
  - Flush the designating TLB
- Caching recently used page table entries in TLB
- Get version information

IPMMU Complex Driver Initialization:

The IPMMU Complex driver initialization sub-module internally stores the configuration data address to enable subsequent API calls to access the configuration data and initializes the MMU/PMB/TLB used by IPMMU Component. The API related to this sub-module is CddIpmmu_Init().

Address translation functionality:

This sub-module provides the service for setting which level target MMU will start to translate, base address of translation table and attribute of memory that MMU handles. User can also control MMU and PMB by calling APIs to enable/disable or flush MMU.

The API related to this module is CddIpmmu_MmuSetMode(), CddIpmmu_MmuSetMemAttr(), CddIpmmu_MmuSetTransTableBase(), CddIpmmu_MmuEnable(), CddIpmmu_MmuDisable(), CddIpmmu_MmuFlush(), CddIpmmu_PmbSet(), CddIpmmu_PmbEnable(), CddIpmmu_PmbDisable().

TLB is a memory cache that stores recently used page table entries. IPMMU Complex Driver provides CddIpmmu_MicroTlbSet() and CddIpmmu_MicroTlbFlush() APIs to control target TLB.

Module Version Information:
This module provides APIs for reading module Id, vendor Id and vendor specific version numbers.
The API related to this module is CddIpmmu_GetVersionInfo().

17.4 IPMMU Complex Driver Component Header and Source File Description

This section explains the IPMMU Complex Driver Component's C source and C header files. These files must be included in the project application while integrating with other modules:

- The C header file generated by IPMMU Complex Driver Generation Tool:
  - CDD_Ipmmu_Cfg.h
  - CDD_Ipmmu_Cbk.h
  - CDD_Ipmmu_Hardware.h
- The C source file generated by IPMMU Complex Driver Generation Tool: CDD_Ipmmu_PBcfg.c
- The IPMMU Complex Driver Component C header files and source files: Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" – section "*3.4.4.3 Folder Structure*".
- The Stub C header files and source file: Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" – section "*3.2.9 Stubs File*".

17.5 Application Programming Interface

This section explains the Data types and APIs provided by the IPMMU Complex Driver Component to the Upper layers.

17.5.1 Imported Types

This chapter explains the Data types imported by the IPMMU Complex Driver Component and lists its dependency on other modules.

17.5.1.1 Standard Types

In this chapter, all types included from the Std_Types.h (according to "*AUTOSAR CP R19-11 Specification of Standard Types*") are listed:

- Std_ReturnType
- Std_VersionInfoType

17.5.1.2 Os Types

In this chapter, all types included from the Os.h (according to "*AUTOSAR CP R19-11 Specification of Operating System*") are listed:

- CounterType
- TickType
- TickRefType

17.5.1.3 **Dem Types**

In this chapter, all types included from the Dem.h (according to "*AUTOSAR CP R19-11 Specification of Diagnostic Event Manager*") are listed:

- Dem_EventIdType
- Dem_EventStatusType

17.5.2 Type Definitions

This section explains the type definitions of IPMMU Complex Driver Component.

17.5.2.1 **CddIpmmu_TranslationLevelType**

Table 17-2 CddIpmmu_TranslationLevelType

Name:	CddIpmmu_TranslationLevelType	
Type:	Enumeration	
Range:	CDD_IPMMU_LEVEL_1ST	First level
	CDD_IPMMU_LEVEL_2ND	Second level
Description:	Represents the Starting translation level of an IPMMU context.	

17.5.2.2 **CddIpmmu_CacheAbilityType**

Table 17-3 CddIpmmu_CacheAbilityType

Name:	CmddIpmmu_CacheAbilityType	
Type:	Enumeration	
Range:	CDD_IPMMU_NON_CACHEABLE	Non-cacheable
	CDD_IPMMU_WRITE_BACK_WA	Write-Back Write-Allocate Cacheable
	CDD_IPMMU_WRITE_THROUGH	Write-Through Cacheable
	CDD_IPMMU_WRITE_BACK_NWA	Write-Back no Write-Allocate Cacheable
Description:	Identify the cache ability attribute.	

17.5.2.3 **CddIpmmu_ShareAbilityType**

Table 17-4 CddIpmmu_ShareAbilityType

Name:	CddIpmmu_ShareAbilityType	
Type:	Enumeration	
Range:	CDD_IPMMU_NON_SHAREABLE	Non-shareable
	CDD_IPMMU_OUTER_SHAREABLE	Outer shareable

	CDD_IPMMU_INNER_SHAREABLE	Inner shareable
Description:	Identify the memory share ability.	

17.5.2.4 **CddIpmmu_ErrorCodeType**

Table 17-5 CddIpmmu_ErrorCodeType

Name:	CddIpmmu_ErrorCodeType	
Type:	Enumeration	
Range:	CDD_IPMMU_ERRCODE_UNDEFINE	Undefined Fault
	CDD_IPMMU_ERRCODE_TLB_FORMAT_ERR	Translation Fault
	CDD_IPMMU_ERRCODE_ACCESS_PERMISSI	Access Permission Fault
	CDD_IPMMU_ERRCODE_SECURE_ACCESS	Secure Access Fault
Description:	Identify the MMU error code.	

17.5.2.5 **CddIpmmu_BusDomainType**

Table 17-6 CddIpmmu_BusDomainType

Name:	CddIpmmu_BusDomainType	
Type:	Enumeration	
Range:	CDD_IPMMU_VI0	IPMMU-VI0 domain
	CDD_IPMMU_VI1	IPMMU-VI1 domain
	CDD_IPMMU_VC0	IPMMU-VC domain
	CDD_IPMMU_IR	IPMMU-IR domain
	CDD_IPMMU_RT	IPMMU-RT0 domain
	CDD_IPMMU_RT1	IPMMU-RT1 domain
	CDD_IPMMU_DS0	IPMMU-DS0 domain
	CDD_IPMMU_HC	IPMMU-HC domain
	CDD_IPMMU_3DG	IPMMU-3DG domain
	CDD_IPMMU_VIP0	IPMMU-VIP0 domain
	CDD_IPMMU_VIP1	IPMMU-VIP1 domain
	CDD_IPMMU_MM	IPMMU-MM domain
	CDD_IPMMU_MAX	Maximum range
Description:	Identify the MMU supported bus domain.	

17.5.2.6 **CddIpmmu_PMBSizeType**

Table 17-7 CddIpmmu_PMBSizeType

Name:	CddIpmmu_PMBSizeType	
Type:	Enumeration	
Range:	CDD_IPMMU_PMB_SIZE_16MB	16-Mbyte page

	CDD_IPMMU_PMB_SIZE_64MB	64-Mbyte page
	CDD_IPMMU_PMB_SIZE_128MB	128-Mbyte page
	CDD_IPMMU_PMB_SIZE_512MB	512-Mbyte page
Description:	Identify the PMB size.	

17.5.2.7 **CddIpmmu_ConfigType**

Table 17-8 CddIpmmu_ConfigType

Name:	CddIpmmu_ConfigType	
Type:	Structure	
Elements:	uint32	ulStartOfDbToc
	P2CONST(CddIpmmu_MmuConfigType, TYPEDEF, CDDIPMMU_APPL_CONST)	pMmuConfig
	P2CONST(CddIpmmu_PmbConfigType, TYPEDEF, CDDIPMMU_APPL_CONST)	pPmbConfig
	P2CONST(CddIpmmu_MicroTlbConfigType, TYPEDEF, CDDIPMMU_APPL_CONST)	pMicroTlbConfig
	uint8	ucNumberOfMmuConfig
	uint8	ucNumberOfPmbConfig
	Uint32	ucNumberOfTlbConfig
Description:	A structure contains the configuration of IPMMU Complex Driver.	

17.5.2.8 **CddIpmmu_MmuConfigType**

Table 17-9 CddIpmmu_MmuConfigType

Name:	CddIpmmu_MmuConfigType	
Type:	Structure	
Elements:	P2CONST(CddIpmmu_TranslationTableConfigType, TYPEDEF, CDDIPMMU_APPL_CONST)	pMmuTranslationTable
	CddIpmmu_MmuCbkFunctionPtrType	pMmuCallbackFunction
	CddIpmmu_TranslationLevelType	enMmuStartLevel
	uint32	ulMmuMemoryAttributeIndirection0
	uint32	ulMmuMemoryAttributeIndirection1
	uint8	ucMmuId
	uint8	ucMmuSelection
	uint8	ucNumberTranslationTable
	uint8	ucMmuAArchSupport
	uint8	ucMmuTranslationTableFormat

	uint8	ucMmuAccessFlagEnable
Description:	A structure contains the MMU configuration.	

17.5.2.9 **CddIpmmu_TranslationTableConfigType**

Table 17-10 CddIpmmu_TranslationTableConfigType

Name:	CddIpmmu_TranslationTableConfigType	
Type:	Structure	
Elements:	CddIpmmu_ShareAbilityType	enMmuShareAbility
	CddIpmmu_CacheAbilityType	enMmuOuterCacheAbility
	CddIpmmu_CacheAbilityType	enMmuInnerCacheAbility
	uint32	ulMmuLowerBaseAddress
	uint8	ucMmuUpperBaseAddress
	uint8	ucTranslationTableId
	uint8	ucMmuId
	uint8	ucMmuTranslationSize
Description:	A structure contains the translation table configuration.	

17.5.2.10 **CddIpmmu_PmbConfigType**

Table 17-11 CddIpmmu_PmbConfigType

Name:	CddIpmmu_PmbConfigType	
Type:	Structure	
Elements:	CddIpmmu_PmbCbkFunctionPtrType	pPmbCallbackFunction
	CddIpmmu_BusDomainType	enPmbBusDomain
	CddIpmmu_PMBSizeType	enPmbPageSize
	uint8	ucPmbPhysicalPageNumber
	uint8	ucPmbUpperPhysicalPageNumber
	uint8	ucPmbVirtualPageNumber
	uint8	ucPmbId
	uint8	ucPmbSelection
	uint8	ucPmbCacheBit
Description:	A structure contains the PMB configuration.	

17.5.2.11 **CddIpmmu_MicroTlbConfigType**

Table 17-12 CddIpmmu_MicroTlbConfigType

Name:	CddIpmmu_MicroTlbConfigType	
Type:	Structure	
Elements:	CddIpmmu_BusDomainType	enTlbBusDomain
	uint32	ulTlbId
	uint8	ucTlbSelection
	uint8	ucTlbTranslationTable
	uint8	ucTlbASID
	uint8	ucTlbAddressTranslationEnable
Description:	A structure contains the micro-TLB configuration.	

17.5.2.12 CddIpmmu_TransTableSettingType

Table 17-13 CddIpmmu_TransTableSettingType

Name:	CddIpmmu_TransTableSettingType	
Type:	Structure	
Elements:	CddIpmmu_ShareAbilityType	enMmuShareAbility
	CddIpmmu_CacheAbilityType	enMmuOuterCacheAbility
	CddIpmmu_CacheAbilityType	enMmuInnerCacheAbility
	uint32	ulMmuLowerBaseAddress
	uint8	ucMmuUpperBaseAddress
	uint8	ucMmuTranslationSize
Description:	A structure contains the Translation table base setting.	

17.5.2.13 CddIpmmu_PmbSettingType

Table 17-14 CddIpmmu_PmbSettingType

Name:	CddIpmmu_PmbSettingType	
Type:	Structure	
Elements:	CddIpmmu_PMBSizeType	enPmbPageSize
	uint8	ucPmbPhysicalPageNumber
	uint8	ucPmbUpperPhysicalPageNumber
	uint8	ucPmbVirtualPageNumber
	uint8	ucPmbCacheBit
Description:	A structure contains the Pmb setting.	

17.5.2.14 CddIpmmu_MicroTlbSettingType

Table 17-15 CddIpmmu_MicroTlbSettingType

Name:	CddIpmmu_MicroTlbSettingType	
Type:	Structure	
Elements:	uint8	ucTlbTranslationTable

	uint8	ucTlbASID
	uint8	ucTlbAddressTranslationEnable
Description:	A structure contains the micro-TLB setting.	

17.5.2.15 **CddIpmmu_MmuModeType**

Table 17-16 CddIpmmu_MmuModeType

Name:	CddIpmmu_MmuModeType	
Type:	Structure	
Elements:	CddIpmmu_TranslationLevelType	enStartLevel
	uint8	ucAccessFlagEnable
Description:	A structure contains the Mode of target MMU.	

17.5.2.16 **CddIpmmu_IMCTRn**

Table 17-17 CddIpmmu_IMCTRn

Name:	CddIpmmu_IMCTRn		
Type:	Union		
Element:	struct BIT	uint32 MMUEN: 1	MMU Enable
		uint32 FLUSH: 1	TLB Invalidate
		uint32 INTEN: 1	Interrupt Enable
		uint32 TREN: 1	Re-translation Enable
		uint32 RTSEL: 4	Re-translation Table Select
		uint32 reserve_1: 7	Reserved
		uint32 SEC: 1	Copy of IMSEC.SEC[n]
		uint32 AFE: 1	Access Flag Enable
		uint32 TRE: 1	TEX Remap Enable
		uint32 reserve_2: 11	Reserved
		uint32 VA64: 1	AArch64 support
		uint32 reserve_3: 2	Reserved
	uint32 INT		This is allowed access of the whole register which controls the behavior of the MMU
Description:	Structure of IMCTRn		

17.5.2.17 **CddIpmmu_IMELARn**

Table 17-18 CddIpmmu_IMELARn

Name:	CddIpmmu_IMELARn		
Type:	Union		
Element:	struct BIT	uint32 EAR_1: 16	The faulting virtual address is set when an address translation error occurred
		uint32 EAR_2: 16	The faulting virtual address is set when an address translation error occurred (upper 16 bits)
	uint32 INT		This is allowed access of the whole register which indicates the address which an address translation error occurred.
Description:	Structure of IMELARn		

17.5.2.18 **CddIpmmu_IMEUARn**

Table 17-19 CddIpmmu_IMEUARn

Name:	CddIpmmu_IMEUARn		
Type:	Union		
Element:	struct BIT	uint32 EAR: 8	The faulting virtual address is set when an address translation error occurred.
		uint32 reserve_1: 16	Reserved. These bits are always read as 0. The write value should always be 0.
		uint32 reserve_2: 8	Reserved. These bits are always read as 0. The write value should always be 0.
	uint32 INT		This is allowed access of the whole register which indicates the address which an address translation error occurred.
Description:	Structure of IMEUARn		

17.5.2.19 **CddIpmmu_IMMAIR0n**

Table 17-20 CddIpmmu_IMMAIR0n

Name:	CddIpmmu_IMMAIR0n		
Type:	Union		
Element:	struct BIT_LD	uint32 ATTR0: 8	The memory attribute encoding for an AttrIndx[2:0] entry
		uint32 ATTR1: 8	The memory attribute encoding for an AttrIndx[2:0] entry
		uint32 ATTR2: 8	The memory attribute encoding for an AttrIndx[2:0] entry

		uint32 ATTR3: 8	The memory attribute encoding for an AttrIndx[2:0] entry
	struct BIT_SD	uint32 TRn: 16	Primary TEX mapping for memory attributes “n.n” is the value of TEX [0], C and B bits.
		uint32 DS0: 1	Mapping of S = 0 attribute for Device memory
		uint32 DS1: 1	Mapping of S = 1 attribute for Device memory
		uint32 NS0: 1	Mapping of S = 0 attribute for Normal memory
		uint32 NS1: 1	Mapping of S = 1 attribute for Normal memory
		uint32 reserve_1: 4	Reserved. These bits are always read as 0. The write value should always be 0.
		uint32 NOSn: 8	Outer Shareable property mapping for memory attributes n
	uint32 INT		This is allowed access of the whole register which indicates long/short-descriptor translation format
Description:	Structure of IMMAIR0n (When using the Long-descriptor translation format, this register works as MAIR0, Memory Attribute Indirection Register in Armv8 VMSAv8-64 and VMSAv8-32. When using the Short-descriptor translation format, this register works as PRRR, Primary Region Remap Register in Armv8 VMSAv8-32)		

17.5.2.20 **CddIpmmu_IMMAIR1n**

Table 17-21CddIpmmu_IMMAIR1n

Name:	CddIpmmu_IMMAIR1n		
Type:	Union		
Element:	struct BIT_LD	uint32 ATTR4: 8	The memory attribute encoding for an AttrIndx[2:0] entry
		uint32 ATTR5: 8	The memory attribute encoding for an AttrIndx[2:0] entry
		uint32 ATTR6: 8	The memory attribute encoding for an AttrIndx[2:0] entry
		uint32 ATTR7: 8	The memory attribute encoding for an AttrIndx[2:0] entry
	struct BIT_SD	uint32 IRn: 16	Inner Cacheable property mapping for memory attributes n
		uint32 ORn: 16	Outer Cacheable property mapping for memory attributes n
	uint32 INT		This is allowed access of the whole register which indicates long/short descriptor translation format

Description:	Structure of IMMAIR1n (When using the Long-descriptor translation format, this register works as MAIR1, Memory Attribute Indirection Register in Armv8 VMSAv8-64 and VMSAv8-32. When using the Short-descriptor translation format, this register works as NMRR, Normal Memory Remap Register in Armv8 VMSAv8-32)

17.5.2.21 **CddIpmmu_IMPCTR**

Table 17-22CddIpmmu_IMPCTR

Name:	CddIpmmu_IMPCTR		
Type:	Union		
Element:	struct BIT	uint32 PMBEN: 1	PMB Enable
		uint32 reserve_1: 1	Reserved
		uint32 INTEN: 1	Interrupt Enable
		uint32 TTEN: 1	TLB Translation Enable
		uint32 TTSEL: 4	Translation Table Select
		uint32 reserve_2: 16	Reserved
		uint32 reserve_3: 8	Reserved
	uint32 INT		This is allowed access of the whole register which controls the behavior of the PMB function.
Description:	Structure of IMPCTR		

17.5.2.22 **CddIpmmu_IMPEAR**

Table 17-23CddIpmmu_IMPEAR

Name:	CddIpmmu_IMPEAR		
Type:	Union		
Element:	struct BIT	uint32 EAR_1: 16	Address translation error occurred
		uint32 EAR_2: 16	Address translation error occurred
	uint32 INT		This is allowed access of the whole register which indicates the error status of the address translation by PMB.
Description:	Structure of IMPEAR		

17.5.2.23 **CddIpmmu_IMPMBAn**

Table 17-24 CddIpmmu_IMPMBAn

Name:	CddIpmmu_IMPMBAn		
Type:	Union		
Element:	struct BIT	uint32 reserve_1: 8	Reserved
		uint32 V: 1	Enable this page translation
		uint32 reserve_2: 15	Reserved
		uint32 VPN: 8	Virtual Page Number
	uint32 INT		This is allowed access of the whole register which sets the virtual page number.
Description:	Structure of IMPMBAn		

17.5.2.24 **CddIpmmu_IMPMBDn**

Table 17-25 CddIpmmu_IMPMBDn

Name:	CddIpmmu_IMPMBDn		
Type:	Union		
Element:	struct BIT	uint32 reserve_1: 3	Reserved
		uint32 C: 1	Cache Bit
		uint32 SZ_0: 1	Page Size
		uint32 reserve_2: 2	Reserved
		uint32 SZ_1: 1	This bit and SZ[0] (bit 4) specify the page size
		uint32 V: 1	Enable this page translation
		uint32 reserve_3: 7	Reserved
		uint32 PPN_U: 8	Upper Physical Page Number
		uint32 PPN_D: 8	Physical Page Number
	uint32 INT		This is allowed access of the whole register which sets the physical page number and the memory size which PMB managed.
Description:	Structure of IMPMBDn		

17.5.2.25 **CddIpmmu_IMPSTR**

Table 17-26 CddIpmmu_IMPSTR

Name:	CddIpmmu_IMPSTR
Type:	Union

Element:	struct BIT	uint32 TF: 1	Translation Fault. This bit is set to 1 when a translation fault occurred during a PMB translation
		uint32 reserve_1: 3	Reserved
		uint32 MHIT: 1	Multiple hit which indicates that multiple PMB hits occurred
		uint32 reserve_2:16	Reserved
		uint32 reserve_3:11	Reserved
	uint32 INT		This is allowed access of the whole register which indicates the error status of the address translation by PMB
Description:	Structure of IMPSTR		

17.5.2.26 **CddIpmmu_IMSCTLR**

Table 17-27 CddIpmmu_IMSCTLR

Name:	CddIpmmu_IMSCTLR		
Type:	Union		
Element:	struct BIT	uint32 reserve_1: 16	Reserved
		uint32 reserve_2: 12	Reserved
		uint32 use_secgrp:1	Use security group to judge Secure/Non-secure access.
		uint32 dismmu: 1	Non-secure access enable for MMU System Control Register
		uint32 nsaccen: 1	Non-secure access enable for MMU System Control Register
		uint32 diswprot: 1	Write protection of MMU System Control Register and auxiliary
	uint32 INT		This is allowed access of the whole register which controls the behavior of IPMMU function
Description:	Structure of IMSCTLR		

17.5.2.27 **CddIpmmu_IMSTRn**

Table 17-28 CddIpmmu_IMSTRn

Name:	CddIpmmu_IMSTRn		
Type:	Union		
Element:		uint32 TF: 1	Translation Fault

	struct BIT	uint32 PF: 1	Page Fault
		uint32 ABORT: 1	This bit is set to 1 when the IPMMU received an error response
		uint32 reserve_1: 1	Reserved
		uint32 MHIT: 1	Indicate that multiple TLB hits occurred
		uint32 reserve_2: 3	Reserved
		uint32 ERRCODE: 3	Indicate error type
		uint32 reserve_3: 1	Reserved
		uint32 ERRLVL: 2	Indicate which level of page table walk caused the error
		uint32 reserve_4: 16	Reserved
		uint32 reserve_5: 2	Reserved
	uint32 INT		This is allowed access of the whole register which indicates the error status of during address translation
Description:	Structure of IMSTRn		

17.5.2.28 **CddIpmmu_IMTTBCRn**

Table 17-29 CddIpmmu_IMTTBCRn

Name:	CddIpmmu_IMTTBCRn		
Type:	Union		
Element:	struct BIT	uint32 TSZ0_32: 3	The size offset of TTBR0n addressed region, encoded as a 3-bit
		uint32 TSZ0_64: 3	VMSA64 mode
		uint32 SL: 2	Starting level for translation table walks
		uint32 IRGN0: 2	Inner Cache ability attributes for the memory
		uint32 ORGN0: 2	Outer Cache ability attributes for the memory
		uint32 SH0: 2	Share ability attributes for the memory
		uint32 reserve_1: 2	Reserved
		uint32 TSZ1_32: 3	The size offset of TTBR1n addressed region, encoded as a 3-bit
		uint32 TSZ1_64: 3	VMSA64 Mode
		uint32 SCSZ: 1	1st level page size select (VMSA32 no LPAE only)
		uint32 PGSZ: 1	(VMSA32 no LPAE)
		uint32 IRGN1: 2	Inner Cache ability attributes for the memory
		uint32 ORGN1: 2	Outer Cache ability attributes for the memory

		uint32 SH1: 2	Share ability attributes for the memory
		uint32 PMB: 1	PMB Enable
		uint32 EAE: 1	Extended Address Enable
	uint32 INT		This is allowed access of the whole register which controls the attribute of TLB managed by each MMU
Description:	Structure of IMTTBCRn		

17.5.2.29 **CddIpmmu_IMTTLBR0_1n**

Table 17-30CddIpmmu_IMTTLBR0_1n

Name:	CddIpmmu_IMTTLBR0_1n		
Type:	Union		
Element:	struct BIT	uint32 reserve_1: 12	Reserved
		uint32 TTBR_1: 4	Bits [15:12] of translation table base address
		uint32 TTBR_2: 16	Bits [31:16] of translation table base address
	uint32 INT		This is allowed access of the whole register which indicates the base address of the TLB managed by each MMU
Description:	Structure of IMTTLBR0n and IMTTLBR1n		

17.5.2.30 **CddIpmmu_IMTTUBR0_1n**

Table 17-31CddIpmmu_IMTTUBR0_1n

Name:	CddIpmmu_IMTTUBR0_1n		
Type:	Union		
Element:	struct BIT	uint32 TTBR: 8	Bits [39:32] of translation table base address
		uint32 reserve_1: 16	Reserved
		uint32 reserve_2: 8	Reserved
	uint32 INT		This is allowed access of the whole register which indicates the base address of the TLB managed by each MMU
Description:	Structure of IMTTUBR0/1n		

17.5.2.31 **CddIpmmu_IMUASIDn**

Table 17-32CddIpmmu_IMUASIDn

Name:	CddIpmmu_IMUASIDn
Type:	Union

Element:	struct BIT	uint32 ASID0: 8	ASID0 which indicates the ASID which the micro-TLB uses in the stage 1 translation
		uint32 ASID1: 8	ASID1 which indicates the ASID which the micro-TLB uses in the stage 2 translation
		uint32 reserve_1: 16	Reserved
	uint32 INT		This is allowed access of the whole register which is used for setting ASID of each uTLB.
Description:	Structure of IMUASIDn		

17.5.2.32 **CddIpmmu_IMUCTRn**

Table 17-33 CddIpmmu_IMUCTRn

Name:	CddIpmmu_IMUCTRn		
Type:	Union		
Element:	struct BIT	uint32 MMUEN: 1	Address Translation Enable
		uint32 FLUSH: 1	Micro-TLB Invalidate
		uint32 reserve_1: 2	Reserved
		uint32 TTSEL: 5	Translation Table
		uint32 reserve_2: 7	Reserved
		uint32 FIXADD: 8	When FIXADDEN is 1, the upper 8bit of physical address is FIXADD [39:32]. This bit must be used only when IMTTBCRn.EAE is 0
		uint32 reserve_3: 7	Reserved
		uint32 FIXADDEN: 1	Fix the upper 8 bits of physical address
	uint32 INT		This is allowed access of the whole register which controls the attribute of TLB managed by each MMU
Description:	Structure of IMUCTRn		

17.5.3 Function Definitions

Below table list all the AUTOSAR APIs and Renesas APIs supported by the IPMMU Complex Driver Component

Table 17-34 APIs provided by the IPMMU Complex Device Driver Component

Sl.No	API's Name	Description

AUTOSAR API		
1	CddIpmmu_Init	This function initializes the module.
2	CddIpmmu_GetVersionInfo	This function provides the version information of this Driver Component.
RENESAS API		
3	CddIpmmu_MmuSetMode	This API sets the mode for each MMU.
4	CddIpmmu_MmuSetMemAttr	This API sets the memory attribute for each MMU.
5	CddIpmmu_MmuSetTransTableBase	This API sets the base address of translation table and attribute.
6	CddIpmmu_MmuEnable	This API enables for the translation based on designating MMU.
7	CddIpmmu_MmuDisable	This API disables for the translation based on designating MMU.
8	CddIpmmu_MmuFlush	This API flushes TLB for the designating MMU.
9	CddIpmmu_PmbSet	This API sets the translation for PMB address.
10	CddIpmmu_PmbEnable	This API enables for PMB translation for the designating domain.
11	CddIpmmu_PmbDisable	This API disables for PMB translation for the designating domain.
12	CddIpmmu_MicroTlbSet	This API sets the address translation for target micro-TLB.
13	CddIpmmu_MicroTlbFlush	This API flushes the target micro-TLB.
14	CddIpmmu_MmuIsEnable	This API support getting status of target MMU.
15	CddIpmmu_MmuGetMode	This API support getting mode of target MMU.
16	CddIpmmu_MmuGetMemAttr	This API support getting memory attribution of target.
17	CddIpmmu_MmuGetTransTableBase	This API support getting translation table of target MMU.
18	CddIpmmu_PmbGet	This API support getting the information of target PMB on target domain
19	CddIpmmu_PmbIsEnable	This API support getting the status of target PMB on target Domain.
20	CddIpmmu_MicroTlbGet	This API support getting the information of target uTLB on target Domain.

17.5.3.1 **CddIpmmu_MmuSetMode**

Table 17-35 CddIpmmu_MmuSetMode

Function Name:	CddIpmmu_MmuSetMode		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuSetMode (uint8 Mmu, CddIpmmu_TranslationLevelType Level)		
Service Id:	0x02		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	Mmu	MMU index:
			00.. 15
	CddIpmmu_TranslationLevelType	Level	Starting level for translation table walks: CDD_IPMMU_LEVEL_1ST CDD_IPMMU_LEVEL_2ND
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API sets the mode for each MMU.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.2 **CddIpmmu_MmuSetMemAttr**

Table 17-36 CddIpmmu_MmuSetMemAttr

Function Name:	CddIpmmu_MmuSetMemAttr
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuSetMemAttr (uint8 Mmu, const uint8 Attr[8])
Service Id:	0x03
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant		
Parameters In:	**Type**	**Parameter**	**MMU index:**
	uint8	Mmu	MMU index:
			00.. 15
	const uint8	Attr	Attribution array: Compatible for ARMv8 MAIR_EL1
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API sets the memory attribute for each MMU.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.3 **CddIpmmu_MmuSetTransTableBase**

Table 17-37 CddIpmmu_MmuSetTransTableBase

Function Name:	CddIpmmu_MmuSetTransTableBase		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuSetTransTableBase (uint8 Mmu, uint8 TableIndex, uint32 BaseAddr, uint8 TransSize, CddIpmmu_ShareAbilityType ShareAbility, CddIpmmu_CacheAbilityType OuterCacheAbility, CddIpmmu_CacheAbilityType InnerCacheAbility)		
Service Id:	0x04		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	Mmu	MMU index:
			00.. 15

	uint8	TableIndex	Translation table index: 0.. 1
	uint32	BaseAddr	Translation table base address: Valid base address.
	uint8	TransSize	Translation table size: 00.. 07
	CddIpmmu_ShareAbility Type	ShareAbility	Share ability attribution: CDD_IPMMU_NON_SHAREABLE CDD_IPMMU_OUTER_SHAREABL E CDD_IPMMU_INNER_SHAREABLE
	CddIpmmu_CacheAbility Type	OuterCacheAbility	Outer Cache ability attribution: CDD_IPMMU_NON_CACHEABLE CDD_IPMMU_WRITE_BACK_WA CDD_IPMMU_WRITE_THROUGH CDD_IPMMU_WRITE_BACK_NWA
	CddIpmmu_CacheAbility Type	InnerCacheAbility	Inner Cache ability attribution: CDD_IPMMU_NON_CACHEABLE CDD_IPMMU_WRITE_BACK_WA CDD_IPMMU_WRITE_THROUGH CDD_IPMMU_WRITE_BACK_NWA
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API set the base address of translation table and attribute.		
Configuration Dependency:	None		
Preconditions:	Driver is already initialized.		

17.5.3.4 **CddIpmmu_MmuEnable**

Table 17-38 CddIpmmu_MmuEnable

Function Name:	CddIpmmu_MmuEnable
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuEnable (uint8 Mmu, Boolean AccessFlagUse)
Service Id:	0x05

Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	Mmu	MMU index:
			00.. 15
	boolean	AccessFlagUse	Access flag use: True/False
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API enables for the translation based on designating MMU.		
Configuration Dependency:	None		
Preconditions:	Driver is already initialized.		

17.5.3.5 **CddIpmmu_MmuDisable**

Table 17-39 CddIpmmu_MmuDisable

Function Name:	CddIpmmu_MmuDisable		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuDisable (uint8 Mmu)		
Service Id:	0x06		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	Mmu	MMU index:
			00.. 15
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	

	Std_ReturnType	E_OK E_NOT_OK
Description:	This API disables for the translation based on designating MMU.	
Configuration Dependency:	None	
Preconditions:	Driver is already initialized.	

17.5.3.6 **CddIpmmu_MmuFlush**

Table 17-40 CddIpmmu_MmuFlush

Function Name:	CddIpmmu_MmuFlush		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuFlush (uint8 Mmu, CddIpmmu_BusDomainType BusDomain)		
Service Id:	0x07		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	CddIpmmu_BusDomainType	BusDomain	Bus domain:
			CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1 CDD_IPMMU_MM
	uint8	Mmu	MMU index:
			00.. 15
Parameters InOut:	None	-	-
Parameters Out:	None	-	-

Return Value:	**Type**	**Possible Return values**
	Std_ReturnType	E_OK E_NOT_OK
Description:	This API flushes TLB for the designating MMU.	
Configuration Dependency:	None	
Preconditions:	Driver must already be initialized.	

17.5.3.7 **CddIpmmu_PmbSet**

Table 17-41 CddIpmmu_PmbSet

Function Name:	CddIpmmu_PmbSet		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_PmbSet (CddIpmmu_BusDomainType BusDomain, uint8 Pmb, Boolean EnableFlag, uint32 VirtualPageNumber, uint32 PhysicalPageNumber, CddIpmmu_PMBSizeType PmbSize)		
Service Id:	0x08		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Parameters In:	Type	Parameter	Value/Range
	CddIpmmu_BusDomainType	BusDomain	Bus domain: CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1
	uint8	Pmb	PMB index: 00.. 15

	boolean	EnableFlag	PMB enable flag: True/False
	uint32	VirtualPageNumber	Virtual page number: 0.. 255
	uint32	PhysicalPageNumber	Physical page number: 0.. 65535
	CddIpmmu_PMBSizeType	PmbSize	PMB size: CDD_IPMMU_PMB_SIZE_16MB CDD_IPMMU_PMB_SIZE_64MB CDD_IPMMU_PMB_SIZE_128MB CDD_IPMMU_PMB_SIZE_512MB
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API sets the translation for PMB's address.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.8 **CddIpmmu_PmbEnable**

Table 17-42 CddIpmmu_PmbEnable

Function Name:	CddIpmmu_PmbEnable		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_PmbEnable (CddIpmmu_BusDomainType BusDomain)		
Service Id:	0x09		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	CddIpmmu_BusDomainType	BusDomain	Bus domain:

			CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API enables for PMB’s translation for the designating domain.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.9 **CddIpmmu_PmbDisable**

Table 17-43CddIpmmu_PmbDisable

Function Name:	CddIpmmu_PmbDisable		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_PmbDisable (CddIpmmu_BusDomainType BusDomain)		
Service Id:	0x0A		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	CddIpmmu_BusDomainType	BusDomain	Bus domain:
			CDD_IPMMU_VI0 CDD_IPMMU_VI1

			CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	Possible Return values	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API disables for PMB's translation for the designating domain.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.10 **CddIpmmu_MicroTlbSet**

Table 17-44CddIpmmu_MicroTlbSet

Function Name:	CddIpmmu_MicroTlbSet		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MicroTlbSet (CddIpmmu_BusDomainType BusDomain, uint8 uTlb, boolean EnableFlag, boolean UsePmb, uint8 Mmu, uint8 ASID)		
Service Id:	0x0B		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	CddIpmmu_BusDomainType	BusDomain	Bus domain:

			CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1
	uint8	uTlb	Micro-TLB index:
			00.. 63
	boolean	EnableFlag	Enable the micro-TLB flag: True/False
	boolean	UsePmb	Using PMB translation flag: True/False
	uint8	Mmu	MMU index:
			00.. 15
	uint8	ASID	Valid ASID:
			00.. 255
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API sets the address translation for target micro-TLB.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.11 **CddIpmmu_MicroTlbFlush**

Table 17-45 CddIpmmu_MicroTlbFlush

Function Name:	CddIpmmu_MicroTlbFlush

Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MicroTlbFlush (CddIpmmu_BusDomainType BusDomain, uint8 uTlb)		
Service Id:	0x0C		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	CddIpmmu_BusDomainType	BusDomain	Bus domain:
			CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1
	uint8	uTlb	Micro-TLB index:
			00.. 63
Parameters InOut:	None	-	-
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API flushes the target micro-TLB.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.12 **CddIpmmu_MmuIsEnable**

Table 17-46 CddIpmmu_MmuIsEnable

Function Name:	CddIpmmu_MmuIsEnable

Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuIsEnable (uint8 Mmu, P2VAR(uint8, AUTOMATIC, CDDIPMMU_APPL_DATA) MmuStatusPtr)		
Service Id:	0x0D		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	Mmu	MMU index:
			00.. 15
Parameters InOut:	uint8*	MmuStatusPtr	Pointer to where to store the status information of target MMU: Address of memory region to store MMU status pointer
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API support getting status of target MMU.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.13 **CddIpmmu_MmuGetMode**

Table 17-47 CddIpmmu_MmuGetMode

Function Name:	CddIpmmu_MmuGetMode
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuGetMode (uint8 Mmu, P2VAR(CddIpmmu_MmuModeType, AUTOMATIC, CDDIPMMU_APPL_DATA) MmuModePtr)
Service Id:	0x0E

Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	Mmu	MMU index:
			00.. 15
Parameters InOut:	uint8*	MmuModePtr	Pointer to where to store the level information of target MMU: Address of memory region to store MMU level pointer
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API support getting mode of target MMU.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.14 **CddIpmmu_MmuGetMemAttr**

Table 17-48 CddIpmmu_MmuGetMemAttr

Function Name:	CddIpmmu_MmuGetMemAttr		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuGetMemAttr (uint8 Mmu, P2VAR(uint8, AUTOMATIC, CDDIPMMU_APPL_DATA) AttrPtr)		
Service Id:	0x0F		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	uint8	Mmu	MMU index:
			00.. 15

Parameters InOut:	uint8*	AttrPtr	Pointer to where to store the memory attribute information of target: Address of memory region to store memory attribute information pointer (size of memory which is pointed by this pointer should be at least 8 byte)
Parameters Out:	None	-	-
Return Value:	**Type**	**Possible Return values**	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API support getting memory attribution of target MMU.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.15 **CddIpmmu_MmuGetTransTableBase**

Table 17-49 CddIpmmu_MmuGetTransTableBase

Function Name:	CddIpmmu_MmuGetTransTableBase		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MmuGetTransTableBase (uint8 Mmu, uint8 TableIndex, P2VAR(CddIpmmu_TransTableSettingType, AUTOMATIC, CDDIPMMU_APPL_DATA) TransTableConfigPtr)		
Service Id:	0x10		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	Type	Parameter	Value/Range
	uint8	Mmu	MMU index: 00.. 15
	uint8	TableIndex	Translation table index: 00.. 01
Parameters InOut:	CddIpmmu_TransTableSettingType*	TransTableConfigPtr	Pointer to where to store the translation table configuration information of target MMU: Address of memory region to store translation table configuration pointer

Parameters Out:	None	-	-
Return Value:	Type	Possible Return values	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API support getting translation table of target MMU.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.16 **CddIpmmu_PmbGet**

Table 17-50 CddIpmmu_PmbGet

Function Name:	CddIpmmu_PmbGet		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_PmbGet (uint8 Pmb, CddIpmmu_BusDomainType BusDomain, P2VAR(CddIpmmu_PmbSettingType, AUTOMATIC, CDDIPMMU_APPL_DATA) PmbSettingPtr)		
Service Id:	0x11		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	Type	Parameter	Value/Range
	uint8	Pmb	Pmb index: 0.. 15
	CddIpmmu_BusDomainType	BusDomain	Bus domain: CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1

Parameters InOut:	CddIpmmu_PmbSettingType*	PmbSettingPtr	Pointer to where to store the configuration information of target PMB: Address of memory region to store PMB configuration pointer
Parameters Out:	None	-	-
Return Value:	Type	Possible Return values	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API support getting the information of target PMB on target domain.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.17 **CddIpmmu_PmbIsEnable**

Table 17-51 CddIpmmu_PmbIsEnable

Function Name:	CddIpmmu_PmbIsEnable		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_PmbIsEnable (CddIpmmu_BusDomainType Busdomain, P2VAR(uint8, AUTOMATIC, CDDIPMMU_APPL_DATA) PmbStatusPtr)		
Service Id:	0x12		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	Type	Parameter	Value/Range
	CddIpmmu_BusDomainType	Busdomain	Bus domain:

			CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1
Parameters InOut:	uint8*	PmbStatusPtr	Pointer to where to store the status information of target PMB: Address of memory region to store PMB status pointer
Parameters Out:	None	-	-
Return Value:	Type	Possible Return values	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API support getting the status of target PMB on target Domain.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.3.18 **CddIpmmu_MicroTlbGet**

Table 17-52CddIpmmu_MicroTlbGet

Function Name:	CddIpmmu_MicroTlbGet		
Prototype:	FUNC(Std_ReturnType, CDDIPMMU_CODE_SLOW) CddIpmmu_MicroTlbGet (uint32 uTlb, CddIpmmu_BusDomainType BusDomain, P2VAR(uint8, AUTOMATIC, CDDIPMMU_APPL_DATA) TlbSettingPtr)		
Service Id:	0x13		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	Type	Parameter	Value/Range
	uint32	uTlb	Tlb index:
			00.. 63

	CddIpmmu_BusDomainType	BusDomain	Bus domain: CDD_IPMMU_VI0 CDD_IPMMU_VI1 CDD_IPMMU_VC0 CDD_IPMMU_IR CDD_IPMMU_RT CDD_IPMMU_RT1 CDD_IPMMU_DS0 CDD_IPMMU_HC CDD_IPMMU_3DG CDD_IPMMU_VIP0 CDD_IPMMU_VIP1
Parameters InOut:	uint8*	PmbStatusPtr	Pointer to where to store the configuration information of target uTlb: Address of memory region to store uTlb setting pointer
Parameters Out:	None	-	-
Return Value:	Type	Possible Return values	
	Std_ReturnType	E_OK E_NOT_OK	
Description:	This API support getting the information of target uTlb on target Domain.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

17.5.4 Preemption of APIs

The table below specifies the preemption of each API that can be invoked at the same time with the API.

Table 17-53Preemption Table of APIs of the IPMMU Driver

	CddIpmmu_Init	CddIpmmu_GetVersionInfo	CddIpmmu_MmuSetMode	CddIpmmu_MmuSetMemAttr	CddIpmmu_MmuSetTransTableBase	CddIpmmu_MmuEnable	CddIpmmu_MmuDisable	CddIpmmu_MmuFlush	CddIpmmu_PmbSet	CddIpmmu_PmbEnable	CddIpmmu_PmbDisable	CddIpmmu_MicroTlbSet	CddIpmmu_MicroTlbFlush	CddIpmmu_MmuIsEnable	CddIpmmu_MmuGetMode	CddIpmmu_MmuGetMemAttr	CddIpmmu_MmuGetTransTableBase	CddIpmmu_PmbGet	CddIpmmu_PmbIsEnable	CddIpmmu_MicroTlbGet
CddIpmmu_Init	-																			
CddIpmmu_GetVersionInfo	√	√																		

CddIpmmu_MmuSetMode	-	√	√																		
CddIpmmu_MmuSetMemAttr	-	√	√	-																	
CddIpmmu_MmuSetTransTableBase	-	√	-	√	-																
CddIpmmu_MmuEnable	-	√	√	√	√	√															
CddIpmmu_MmuDisable	-	√	√	√	√	√	√														
CddIpmmu_MmuFlush	-	√	√	√	√	√	√	√													
CddIpmmu_PmbSet	-	√	√	√	√	√	√	√	-												
CddIpmmu_PmbEnable	-	√	√	√	√	√	√	√	√	√											
CddIpmmu_PmbDisable	-	√	√	√	√	√	√	√	√	√	√										
CddIpmmu_MicroTlbSet	-	√	√	√	√	√	√	√	√	√	√	-									
CddIpmmu_MicroTlbFlush	-	√	√	√	√	√	√	√	√	√	√	-	√								
CddIpmmu_MmuIsEnable	-	√	√	√	√	√	√	√	√	√	√	√	√	√							
CddIpmmu_MmuGetMode	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√						
CddIpmmu_MmuGetMemAttr	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√					
CddIpmmu_MmuGetTransTableBase	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√				
CddIpmmu_PmbGet	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√			
CddIpmmu_PmbIsEnable	-	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√		
CddIpmmu_MicroTlbGet	-	√	√	√	√	√	√	√	√	√	√	-	√	√	√	√	√	√	√	√	

-: cannot be invoked at the same time

√: can be invoked at the same time

17.5.5 Additional Error Handling and Restrictions

Following is additional error handling and API specific limitations:

- The CddIpmmu_MicroTlbSet should be called after MMU (CddIpmmu_MmuSetMode, CddIpmmu_MmuSetMemAttr, CddIpmmu_MmuSetTransTableBase, CddIpmmu_MmuDisable, CddIpmmu_MmuFlush) or PMB (CddIpmmu_PmbSet, CddIpmmu_PmbEnable) which is used for this uTLB is setup correctly.
- Due to Hardware limitation, input parameter TransSize of CddIpmmu_MmuSetTransTableBase API should be less than 5. Detail of the Hardware limitation is described in section "*17.2.2 Preconditions*".
- Due to Hardware limitation, input parameter Mmu of CddIpmmu_MmuSetMode, CddIpmmu_MmuSetMemAttr, CddIpmmu_MmuSetTransTableBase, CddIpmmu_MmuEnable, CddIpmmu_MicroTlbSet should not be 0. Detail of the Hardware limitation is described in section "*17.2.2 Preconditions*".
- Due to Hardware limitation, input parameter Mmu of CddIpmmu_MmuFlush should not be 0 or the number of disabled MMU by CddIpmmu_MmuDisable. Detail of the Hardware limitation is described in section "*17.2.2 Preconditions*".
- Due to Hardware limitation, User should not call API CddIpmmu_Mmuflush to flush MMUn when TLB which use MMUn is disabled. Detail of Hardware limitation is described in section "*17.2.2 Preconditions*".

17.6 Development and Production Errors

This section describes all the development errors and production error code that are reported by the IPMMU Complex Driver Component.

17.6.1 IPMMU Complex Driver Component Development Errors

The following table contains the DET errors that are reported by IPMMU Complex Driver Component. These errors are reported to Development Error Tracer Module when the IPMMU Complex Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 17-54 DET Errors of IPMMU Complex Driver Component

Sl. No.	**1**
Error Code	CDDIPMMU_E_UNINIT
Related API(s)	CddIpmmu_MmuSetMode CddIpmmu_MmuSetMemAttr CddIpmmu_MmuSetTransTableBase CddIpmmu_MmuEnable CddIpmmu_MmuDisable CddIpmmu_MmuFlush CddIpmmu_PmbSet CddIpmmu_PmbEnable CddIpmmu_PmbDisable CddIpmmu_MicroTlbSet CddIpmmu_MicroTlbFlush CddIpmmu_MmuIsEnable CddIpmmu_MmuGetMode CddIpmmu_MmuGetMemAttr CddIpmmu_MmuGetTransTableBase CddIpmmu_PmbGet CddIpmmu_PmbIsEnable CddIpmmu_MicroTlbGet
Source of Error	When the APIs are invoked without the initialization of IPMMU Complex Driver Component.
Origin	Renesas
Sl. No.	**2**
Error Code	CDDIPMMU_E_PARAM_VALUE

Related API(s)	CddIpmmu_MmuSetMode CddIpmmu_MmuSetMemAttr CddIpmmu_MmuSetTransTableBase CddIpmmu_MmuEnable CddIpmmu_MmuDisable CddIpmmu_MmuFlush CddIpmmu_MmuIsEnable CddIpmmu_MmuGetMode CddIpmmu_MmuGetMemAttr CddIpmmu_MmuGetTransTableBase CddIpmmu_PmbSet CddIpmmu_PmbEnable CddIpmmu_PmbDisable CddIpmmu_PmbGet CddIpmmu_PmbIsEnable CddIpmmu_MicroTlbSet CddIpmmu_MicroTlbFlush CddIpmmu_MicroTlbGet
Source of Error	When the API is invoked with the invalid input values.
Origin	Renesas
Sl. No.	**3**
Error Code	CDDIPMMU_E_ALREADY_INITIALIZED
Related API(s)	CddIpmmu_Init
Source of Error	When the API CddIpmmu_Init is invoked twice.
Origin	Renesas
Sl. No.	**4**
Error Code	CDDIPMMU_E_PARAM_POINTER
Related API(s)	CddIpmmu_Init CddIpmmu_MmuSetMemAttr CddIpmmu_MmuEnable CddIpmmu_GetVersionInfo CddIpmmu_MmuIsEnable CddIpmmu_MmuGetMode CddIpmmu_MmuGetMemAttr CddIpmmu_MmuGetTransTableBase CddIpmmu_PmbGet CddIpmmu_PmbIsEnable CddIpmmu_MicroTlbGet
Source of Error	When API invoked with message pointer is NULL_PTR.
Origin	Renesas
Sl. No.	**5**
Error Code	CDDIPMMU_E_INVALID_DATABASE
Related API(s)	CddIpmmu_Init

Source of Error	DET error indicates that database is invalid or the pointer to the database is NULL
Origin	Renesas

17.6.2 IPMMU Complex Driver Component Production Errors

In this section the DEM errors identified in the IPMMU Complex Driver Component are listed. IPMMU Driver Component reports these errors to DEM by invoking Dem_SetEventStatus API. This API is invoked when the processing of the given API request fails.

Table 17-55 DEM Errors of IPMMU Complex Driver Component

Sl. No.	**1**
Error Code	CDDIPMMU_E_WRITE_VERIFY_FAILURE
Related API(s)	CddIpmmu_Init CddIpmmu_MmuSetMode CddIpmmu_MmuSetMemAttr CddIpmmu_MmuSetTransTableBase CddIpmmu_MmuEnable CddIpmmu_MmuDisable CddIpmmu_PmbSet CddIpmmu_PmbEnable CddIpmmu_PmbDisable CddIpmmu_MicroTlbSet CDDIPMMU_ERROR_ISR
Source of Error	When the written value to IPMMU registers is incorrect. Detection mechanism is read-back registers value and compare with written value.
Origin	Renesas
Sl. No.	**2**
Error Code	CDDIPMMU_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	CDDIPMMU_ERROR_ISR
Source of Error	DEM error is raised when unintended interrupt occurred
Origin	Renesas

17.7 IPMMU Driver Component Runtime Errors

AUTOSAR does not define any runtime error code for CDD IPMMU Driver.

17.8 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of IPMMU Complex Driver Component software.

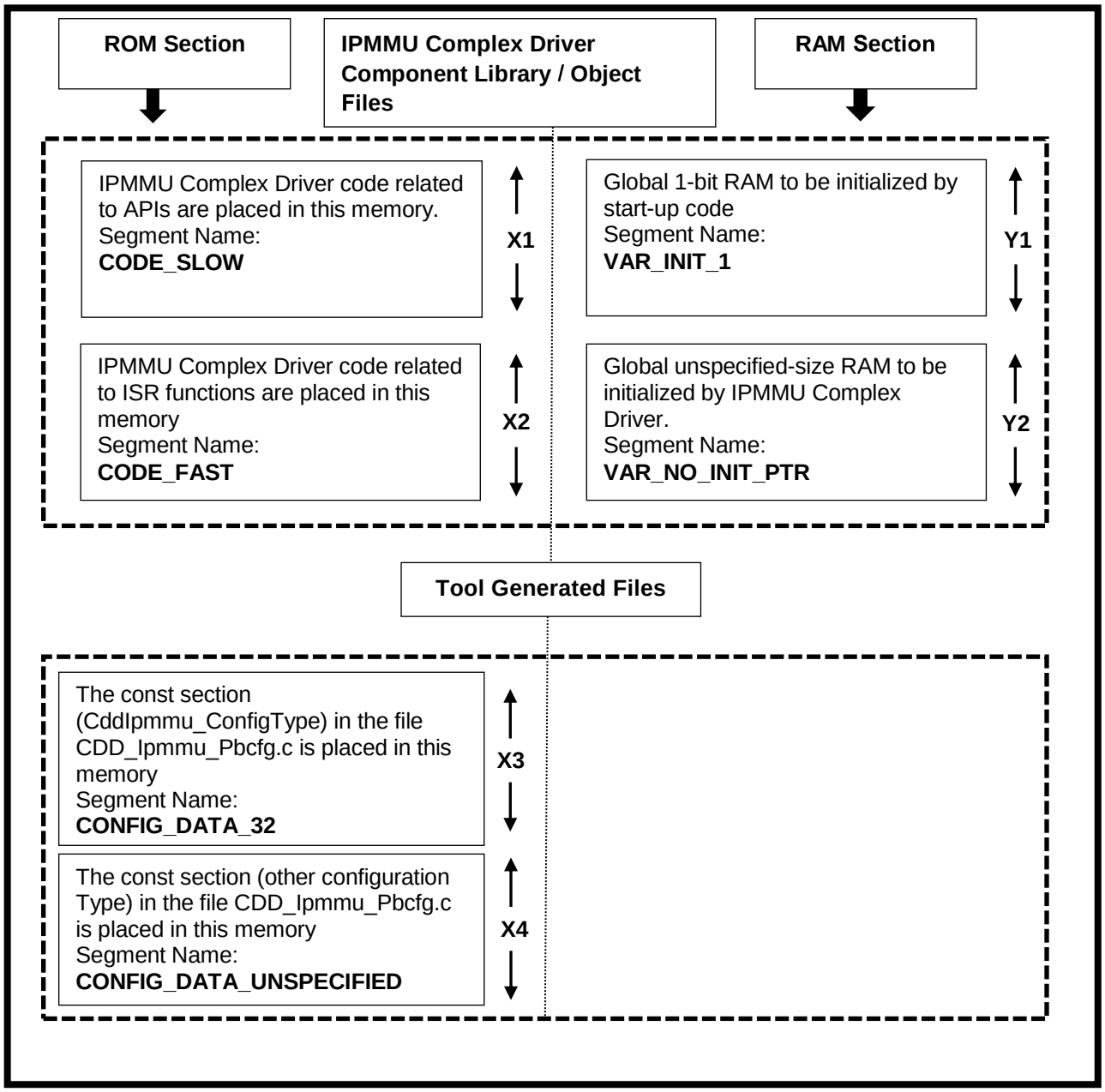


Figure 17-4 IPMMU Complex Driver Component Memory Organization

ROM Section (X1, X2, X3, X4):

- **CODE_SLOW (X1):** API(s) of IPMMU Complex Driver Component except for ISR functions.
- **CODE_FAST (X2):** Interrupt functions of IPMMU Complex Driver Component code that can be located in code memory.
- **CONFIG_DATA_32 (X3):** This section consists of IPMMU Complex Driver Component controller CddConfigSet constant structures generated by IPMMU Complex Driver Component generation tool. This can be located in the code memory
- **CONFIG_DATA_UNSPECIFIED (X4):** This section consists of IPMMU Complex Driver Component controller Mmu, Pmb, Translation table and micro TLB configuration constant structures generated by IPMMU Complex Driver Component generation tool. This can be located in the code memory.

RAM Section (Y1, Y2):

- **VAR_INIT_1 (Y1):** This section consists of the global RAM variables of 1-bit size that are initialized by start-up code and used internally by IPMMU Complex Driver Component. This can be located in data memory.
- **VAR_NO_INIT_PTR (Y2):** This section consists of the global RAM variables generated by IPMMU Complex Driver Component Generation Tool. This can be located in data memory.

Remark: X1, X2, X3, X4, X5, Y1 and Y2 pertain to only IPMMU Complex Driver Component. User must ensure that none of the memory areas overlap with each other. Even 'debug' information should not overlap

17.9 Device Specific Information

17.9.1 Interaction Between the User and IPMMU Complex Driver Component

The detail of the services supported by the IPMMU Complex Driver Component to the upper layers users is provided in this section.

17.9.1.1 Domain Mapping

The hardware domain available for RCar-V4H is as given in the below table:

Table 17-56 Hardware domain mapping

IPMMU	Bus Domain	Power Domain	CddIpmmu_BusDomainType	RCar-V4H
IPMMU-VI0	VIO (Video IO) domain AXI	Always-On	CDD_IPMMU_VI0	√
IPMMU-VI1	VIO (Video IO) domain AXI	Always-On	CDD_IPMMU_VI1	√
IPMMU-VC	VC (Video Codec) domain AXI	Always-On	CDD_IPMMU_VC0	√
IPMMU-IR	IMP domain AXI	A3IR	CDD_IPMMU_IR	√
IPMMU-RT0	RT (Real Time) domain AXI	Always-On	CDD_IPMMU_RT	√
IPMMU-RT1	RT (Real Time) domain AXI	Always-On	CDD_IPMMU_RT1	√
IPMMU-DS0	Peripheral domain AXI	Always-On	CDD_IPMMU_DS0	√
IPMMU-HC	HC (High communication) domain AXI	Always-On	CDD_IPMMU_HC	√
IPMMU-3DG	3DG (3D-Graphics) domain AXI	Always-On	CDD_IPMMU_3DG	√
IPMMU-VIP0	VIP (Vision IP) domain AXI	Always-On	CDD_IPMMU_VIP0	√
IPMMU-VIP1	VIP (Vision IP) domain AXI	Always-On	CDD_IPMMU_VIP1	√
IPMMU-MM	Main Memory domain AXI	Always-On	CDD_IPMMU_MM	√

17.9.1.2 ISR Functions

The table below provides the list of handlers corresponding to the hardware unit ISR(s) in IPMMU Complex Device Driver Component. The user should configure the ISR functions mentioned below:

Table 17-57Interrupt Vector Table

Name of interrupt	Interrupt Source	Name of the ISR Function
IPMMU	- MMU Error Interrupt - PMB Error interrupt	CDDIPMMU_ERROR_CAT2_ISR
		CDDIPMMU_ERROR_ISR

17.9.2 Multi-Core / Multi-Instantiation

IPMMU driver does not support multi-core and multi-instantiation for this device.

17.10 Non-AUTOSAR environment integration

The IPMMU Complex Driver Component for Renesas R-Car Gen4 SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the IPMMU Complex Driver Component to Non-AUTOSAR environment.

17.10.1 Stub modules handling

17.10.1.1 Os

The Os stub files are organized in the following folder: rel\common\generic\stubs\<Autosar version>\Os

In the AUTOSAR environment, IPMMU Complex Driver Component must include "*Os.h*" header file to obtain the interrupt category information configured in the OS.

17.10.1.2 Det

The Det stub files are organized in the following folder: \rel\common\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, IPMMU Complex Driver Component uses Det_ReportError API provided by the DET module to report a development error when CDDIPMMU_DEV_ERROR_DETECT configured as STD_ON. E.g. IPMMU Complex Driver has not been initialized, API is provided with invalid parameter, etc. The API prototype is as of follow: Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

17.10.1.3 Dem

The Dem stub files are organized in the following folder: \rel\common\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, IPMMU Complex Driver Component uses Dem_SetEventStatus API provided by the DEM module to report a production error. The API prototype is as of follow:
Std_ReturnType Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

17.10.2 Callback function usage

The IPMMU Complex Driver Component has a notification callback function which will be configured in parameters "*CddIpmmuMmuCallbackFunction*" and "*CddIpmmuPmbCallbackFunction*" being used to notify address translation error to application SWC. The configured value of callback function parameters should follow the following naming convention:

- CddIpmmu_MmuCallBackFunction<n>
- CddIpmmu_PmbCallBackFunction<n><domain>

Note: <n> is MMU or PMB number, and <domain> is domain name configured in the same configuration container.

17.10.3 Scheduled function usage

IPMMU Complex Driver Component does not provide scheduled functions.

17.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:

- \rel\V4H\common_family\include\arm\Interrupt_VectorTable.h
- \rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

Non-AUTOSAR users shall use CDDIPMMU_ERROR_ISR ISR as specified in section "*17.9.1.2 ISR Functions*".

18. EMM

18.1 Overview

The purpose of this document is to describe the information related to EMM Complex Device Driver Component.

This document is intended for the developers of ECU software on R-Car Series, 4th Generation SoC using Application Programming Interfaces provided by AUTOSAR specification for EMM Complex Driver. The EMM Complex Driver Component provides the following services:

- EMM Complex Driver Component initialization.
- Read status of safety-related errors.
- Select a destination of error signal to be notified, either the system interrupt controller (INTC) or external pin.
- Enable/Disable a pseudo error functionality.
- Set/Clear a specific pseudo error signal.
- To which error signals with target configured as the system interrupt controller (INTC), the corresponding error status will be stored at a specific address provided by user via Generation Tool.
- Clear all status of safety-related errors.
- Get all current count-up value.
- Support to control external error request
- Get version information.

The EMM Complex Driver Component comprises of two parts: Embedded Software and the Generation Tool to achieve scalability and configurability.

The purpose of this document is to describe the information related to Embedded Software part of the EMM Complex Driver Component for Renesas R-Car Series, 4th Generation SoC. Device V4H, please refer to *"EMM Complex Driver Component Generation Tool User's Manual"* for the detail of Generation Tool part.

The below figure depicts the EMM Complex Device Driver as part of layered AUTOSAR MCAL Layer:

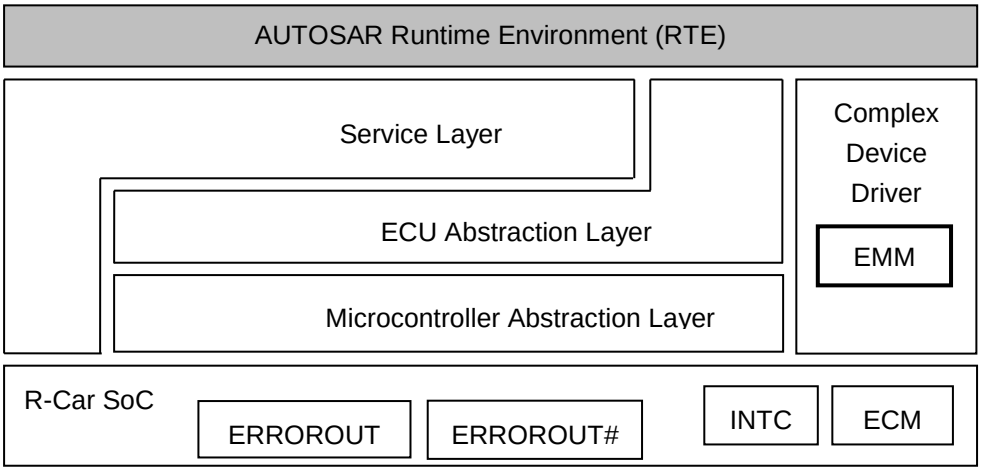


Figure 18-1 System Overview of the EMM Complex Device Driver in AUTOSAR Layer Architecture

The following diagram shows the system overview of the AUTOSAR Architecture for EMM Complex Device Driver:

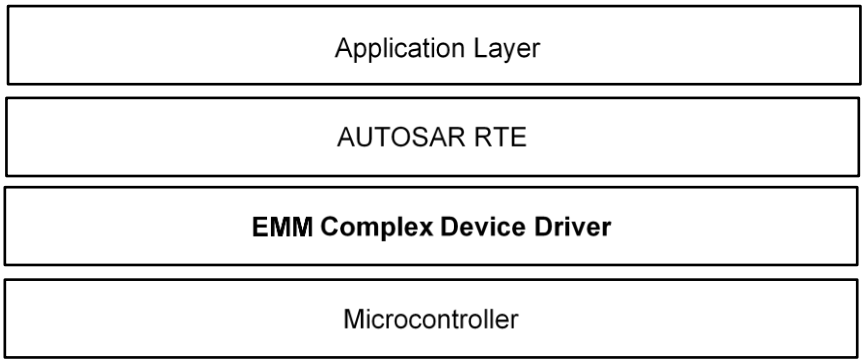


Figure 18-2 System Overview of AUTOSAR Architecture

The EMM Complex Device Driver Component is Vendor Specific Module provided by Renesas to manage the error signals received from each module.

Sample Application is available for user to understand and test EMM Complex Device Driver without proper AUTOSAR upper layer, which is described in *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.7 Sample Application*. Please refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" - section 3.7.2.2 How to build the Sample Application* for more information on how to configure, compile and flash the EMM Complex Device Driver to R-Car SoC.

18.2 Forethoughts

18.2.1 General

Following information will aid the user to use the EMM Complex Driver Component software efficiently:

- EMM Complex Driver Component only supports to manage safety-related errors on R-Car Series, 4th Generation SoC.
- MCU specific initializations such as reset registers, one-time writable registers, interrupt stack pointer, user stack pointer, internal watchdog, specific features of internal memory and registers are not implemented by EMM Complex Driver. These initializations must be implemented by the start-up code.
- The ISR functions and the corresponding handler addresses are provided in *section 18.9.1.2 "ISR Functions"*. User must ensure that Interrupt Vector Table configuration is done as per the information provided in the table.
- Porting specific information such as compiler details, timing details and memory consumption are provided in *section 18.9 "Device Specific Information"*.
- All development errors will be reported to DET by using the Det_ReportError API provided by DET.
- If any development errors are reported to DET, the normal flow of execution of driver will be aborted.
- All production errors will be reported to DEM by using the Dem_SetEventStatus API provided by DEM.
- All interrupt-related functions are mapped to CODE_FAST section (see *Table 7.12 of "AUTOSAR Specification of Memory Mapping"*). Other functions are mapped to CODE_SLOW section (see *Table 7.13 of "AUTOSAR Specification of Memory Mapping").*
- Pseudo-error related APIs are enabled by default. If no use pseudo error functionality, it can be disabled through CddEmmPseudoErrorApi configuration parameter under CddGeneral container to reduce memory

usage. Base on this, three following APIs CddEmm_SupportPseudoError, CddEmm_SetPseudoErrorSignal and CddEmm_ClearPseudoErrorSignal will be excluded from compiling.

- EMM Complex Driver provides Register Write-Verify feature to verify the execution of control register's write operation. After writing to the control register, value of that register shall be read back to make sure that register has been written correctly.
- EMM Complex Driver supports one instance. Therefore, configuration parameter CddInstanceId should be configured as 0. In the implementation, instance ID is always used as 0 without using CddInstanceId parameter in CDF setting.

18.2.2 Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the EMM Complex Driver Component:

- ECM registers which are handled by EMM driver shall be accessed by CR52 EMM CDD only.
- User-address shall be configured with valid range before using (User-address configuration parameter name is CddEmmAddressToSaveErrorStatus). This valid range is not only in range of a 32-bit number (from 0x40000000 to 0xBFFFFFFF) but also it must be belonged to the User RAM (User RAM is the memory that is free for user to develop their application, this memory will be allocated by Boot Loader. Therefore, to avoid the access to the prohibit memory (the memory area is predefined to use for system operation), the configurable of user-address must be considered clearly and carefully. Otherwise, the whole system can be broken and lead to the serious consequences. Finally, user-address is the address that must be accessed (Read/Write) and that address mustn't be used with multiple purpose.

Note: Each error domain has only one user-address (CDDEMM_DOMAINn_USER_ADDR, n = 0..13, 16..36, 38..42 for V4H) and that error domain must be enable before its user-address is used. Shouldn't use user-address if that error domain is disabled or not use. Besides that, to avoid the concurrent access to user-addresses, user-addresses must be use with only one purpose that store value of ECMERRSTSRn register (n = 0..13, 16..36, 38..42 for V4H) when the target of error signal is set to Interrupt Controller.

- At least one error signal should be configured as true to use fully EMM Driver's API. If all error signals are configured as false, only CddEmm_Init is available for use.
- Register Write-Verify mechanism should be enabled to avoid and prevent the issue that EMM CDD driver cannot access to set HW registers in Initialization service since the clock is stop supplying unintendedly.
- When in pseudo error mode, to clear completely an error status which is inserted by CddEmm_SetPseudoErrorSignal API, the user must call CddEmm_ClearPseudoErrorSignal and CddEmm_ClearErrorStatus API in turn.
- ISR priority should be configured to be greater than priority of the thread/task which invoke the driver's APIs. Otherwise, the driver operation is not guaranteed.
- The CDD_Emm_Cfg.h file which is generated by the EMM Complex Driver Component Code Generation Tool must be compiled and linked along with EMM Complex Driver Component source files.
- The application has to be rebuilted, if there is any change in the CDD_Emm_Cfg.h file which is generated by the EMM Complex Driver Component Generation Tool.

- CDD_Emm_PBcfg.c which is generated for single configuration using EMM Complex Driver Component Generation Tool can be compiled and linked independently.
- The EMM Complex Driver Component needs to be initialized before accepting any request. The CddEmm_Init API should be invoked to initialize EMM Complex Driver Component before calling any other EMM Complex Driver APIs.
- The user should ensure that EMM Complex Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated only when the static configuration parameter CddEmmDevErrorDetect is enabled. Application should ensure that the right parameters are passed while invoking the APIs when CddEmmDevErrorDetect is disabled.
- The EMM Complex Driver states are checked only when the static configuration parameter CddEmmDevErrorDetect is enabled. Application should ensure the recommended sequence of invoking APIs is satisfied when CddEmmDevErrorDetect is disabled.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used.
- Error Output Control for ECM should be enabled through the setting on ERROUT bit of ECMERROUTCTLR register (ERROUT = 1'b1). EMM is designed assuming the setting of Error Output Control mechanism is enabled, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, the operation of driver is not guaranteed.
- Interrupt service routine CDDEMM_DOMAINn_ISR (n = 0..13, 16..36, 38..42) is non-reentrant functions. Otherwise, the driver operation is not guaranteed.
- User must initialize the following hardware IPs as below *Table 18-1* before initializing EMM module.

Table 18-1 List of Hardware IP affect to EMM

Hardware IP	Module Name	Expected Setting	Description
CPG	MCU	The user has to check to confirm that the clock is supplied for CDDEMM module (ECM HW IP).	MCU handle the clock supply control of CDDEMM module, please refer to User's Manual of MCU module for detail setting.
ECM	N/A	Write Access Protection for ECM is disable through the setting on WPD bit of ECMWPCNTR register. WPD = 1'b1.	CDDEMM for the R-Car Gen4 is designed assuming the setting of Write Access Protection mechanism is disable, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, the operation of driver is not guaranteed since it may not write to registers of ECM.

AXI-bus	N/A	SDRAM address mapping in HW should be configured according to *chapter 18* in *R-Car V4M Series User's Manual: Hardware*.	CDDEMM is designed assuming the setting of SDRAM address mapping is corrected for V4H device, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, the operation of driver is not guaranteed, since EMM operation is under controlled by DRAM for memory control.
DBSC5	N/A	The maximum use of SDRAM bus bandwidth will be enabled by DBSC5, according to *chapter 33* in *R-Car V4M Series User's Manual: Hardware*.	CDDEMM is designed assuming the configuration of SDRAM bus bandwidth is suitable for V4H device, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, the operation of driver is not guaranteed, since CDDEMM operation is under controlled by DRAM for memory control.
Life Cycle	N/A	Security/safety access protection for ECM should be configured to be accessed by all resource.	CDDEMM is designed assuming the setting of this protection mechanism allow register to be accessed by all resource for V4H so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, the operation of driver is not guaranteed since it may not access to registers of ECM.

18.2.3 Data Consistency

To support the re-entrance and interrupt services, the EMM Complex Driver Component will ensure the data consistency while accessing its hardware registers. The EMM Complex Driver Component will use SchM_Enter_CddEmm_<Exclusive Area> and SchM_Exit_CddEmm_<Exclusive Area> functions. The SchM_Enter_CddEmm_<Exclusive Area> function is called before accessing the data needs to be protected and SchM_Exit_CddEmm_<Exclusive Area> function is called after the data is accessed.

The following exclusive area along with scheduler services is used to provide data integrity for shared resources: CDDEMM_INTERRUPT_CONTROL_PROTECTION.

These functions can be disabled by disabling the configuration parameter 'CddEmmCriticalSectionProtection'.

18.3 Architecture Details

The EMM Complex Driver architecture is shown in the following figure:

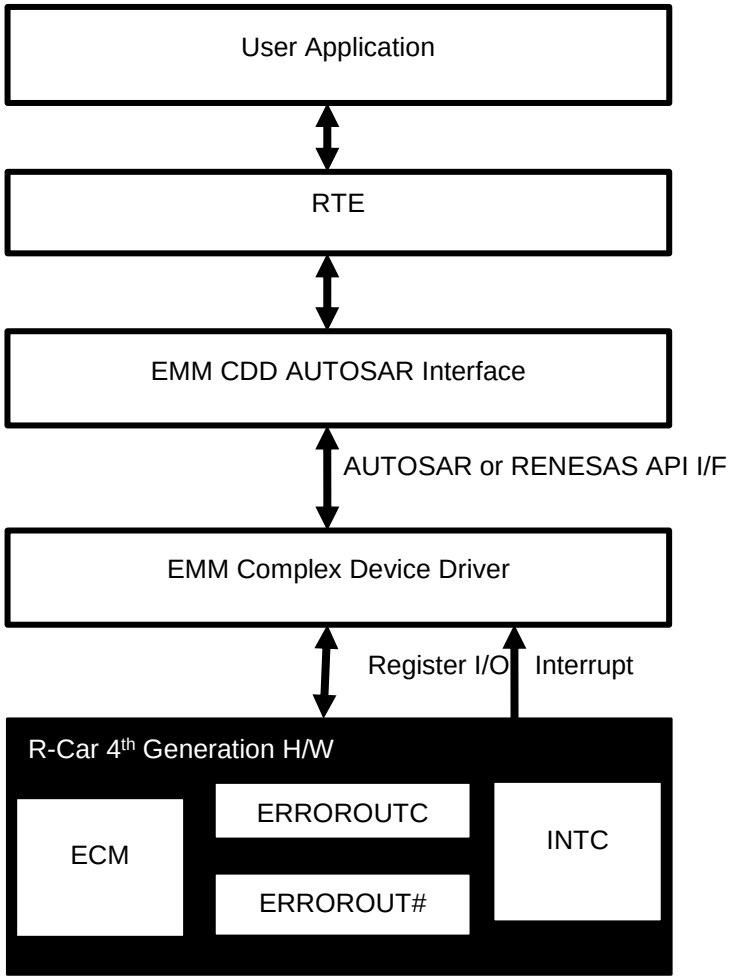


Figure 18-3 EMM Complex Device Driver Component Architecture

The EMM Complex Driver Component can be divided into following sub-components based on the functions performed by the EMM Complex Driver:

- Driver initialization.
- Read status of safety-related errors.
- Select the destination of error signal to be notified, either the system interrupt controller (INTC) or external pin (ERROROUTC, ERROROUT#).
- Support pseudo error functionality.
- Clear all status of safety-related errors.
- Get all current error count-up value.
- Get Version Information.
- Support external request control functionality

Driver Initialization:

The driver initialization internally stores the configuration data address to enable subsequent API calls to access the configuration data and initializes the global variables used by EMM Complex Driver Component.

The API related to this sub-module is CddEmm_Init.

Read Status of Safety-Related Errors:

The EMM Complex Driver provides a service to read status of safety-related errors from each module in R-Car Series, 4rd Generation SoC.

The API related to this sub-module is CddEmm_ReadErrorStatus.

Select the Destination of Error Signal to be notified:

The EMM Complex Driver provides a service to select either the system interrupt controller (INTC) or external pin (ERROROUTC, ERROROUT#) as target to be notified in case of errors occurred.

The API related to this sub-module is CddEmm_SetTarget.

Support Pseudo Error Functionality:

The EMM Complex Driver provides services to enable/disable the pseudo error functionality, also set/clear a specific pseudo error signal.

The API related to enable/disable the pseudo error functionality is CddEmm_SupportPseudoError.

The API related to set the specific pseudo error signal is CddEmm_SetPseudoErrorSignal.

The API related to clear the specific pseudo error signal is CddEmm_ClearPseudoErrorSignal.

Clear all Status of Safety-Related Errors:

The EMM Complex Driver provides a service to clear the status of safety-related errors.

The API related to this sub-module is CddEmm_ClearErrorStatus.

Get all current error count-up value:

The EMM Complex Driver provides a service to get current error 1-bit value and current error multi-bit value of all ECM Error Count Registers.

The API related to this sub-module is CddEmm_GetCurrentErrorCountUpValue.

Get Version Information:

The EMM Complex Driver provides a service to return version information.

The API related to this sub-module is CddEmm_GetVersionInfo.

Support External Error Request Control Functionality:

The EMM Complex Driver provides services to enable/disable hold/mask counter for the external error request control functionality, also set Count Value for hold counter and mask counter. The API related to enable/disable hold/mask counter of the external error request control is CddEmm_SupportControlExternalErrorRequest.

The API related to set hold/mask counter of the external error request control is CddEmm_SetHoldMaskCounter.

Remark: in External Error Request Control functionality:

- The Count Value of hold counter or mask counter shall be set before hold counter or mask counter is enabled.
- In application, when change the current Count Value of hold counter or mask counter to another value, hold counter or mask counter must be disabled as firstly when it's enabling. If not a development error is reported to DET, refer to section *18.6.1 EMM Complex Device Driver Component Development Errors*.

18.4 EMM Complex Device Driver Component Header And Source File Description

This chapter explains the EMM Complex Device Driver Component's source and header files. Those files have to be included in the project application while integrating with other modules.

The C header file generated by EMM Complex Device Driver Generation Tool: CDD_Emm_Cfg.h
The C source file generated by EMM Complex Device Driver Generation Tool: CDD_Emm_PBcfg.c

The EMM Complex Device Driver Component C header files and Component source files: Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.4.6.3 Folder Structure*.
The Stub C header files and source file: Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.2.9 Stubs File*.
S

18.5 Application Programming Interface

This chapter explains the Data types and APIs provided by the EMM Complex Device Driver Component to the Upper layers.

18.5.1 Imported Types

This chapter explains the Data types imported by the EMM Complex Device Driver Component and lists its dependency on other modules.

18.5.1.1 Standard Types

In this chapter all types included from the Std_Types.h (according to *"AUTOSAR Specification of Standard Types"*) are listed:

- Std_ReturnType.
- Std_VersionInfoType.

18.5.1.2 OS Types

None

18.5.1.3 Dem Types

In this chapter all types included from the Dem.h (according to *"AUTOSAR Specification of Diagnostic Event Manager"*) are listed:

- Dem_EventIdType.
- Dem_EventStatusType.

18.5.1.4 Platform Types

In this chapter all types included from the Platform_Types.h (according to *"AUTOSAR_SWS_PlatformTypes"*) are listed:

- Boolean
- Char

- Int
- Uint/uint8/uint16/ uint32/ uint64
- Void

18.5.2 Type Definitions

This chapter explain the type definitions of EMM Complex Device Driver Component according to AUTOSAR Specification.

18.5.2.1 **CddEmm_ErrorIDType**

Table 18-2 CddEmm_ErrorIDType

Name:	CddEmm_ErrorIDType
Type:	uint32
Elements:	0..<Maximum number of error signal enabled in Generation Tool>
Description:	Represents an error signal identifier.

18.5.2.2 **CddEmm_ErrorCountType**

Table 18-3 CddEmm_ErrorCountType

Name:	CddEmm_ErrorCountType	
Type:	Enumeration	
Elements:	CDDEMM_ERROR_1_BIT	1-bit error count
	CDDEMM_ERROR_MULTI_BIT	Multi-bit error count
Description:	Enumeration for error count type	

18.5.2.3 **CddEmm_PseudoErrorModeType**

Table 18-4 CddEmm_PseudoErrorModeType

Name:	CddEmm_PseudoErrorModeType	
Type:	enumeration	
Elements:	CDDEMM_PSEUDO_ERROR_DISABLE	Disable pseudo error functionality
	CDDEMM_PSEUDO_ERROR_ENABLE	Enable pseudo error functionality
Description:	Enumeration for pseudo error mode type.	

18.5.2.4 **CddEmm_TargetType**

Table 18-5 CddEmm_TargetType

Name:	CddEmm_TargetType
Type:	Enumeration

Elements:	CDDEMM_TARGET_ERROROUT	External Pin
	CDDEMM_TARGET_INTC	Interrupt Controller
Description:	Enumeration for reporting target type	

18.5.2.5 **CddEmm_ErrorSignalConfigType**

Table 18-6 CddEmm_ErrorSignalConfigType

Name:	CddEmm_ErrorSignalConfigType		
Type:	Structure		
Elements:	**Type**	**Name**	**Explanation**
	P2VAR(uint32, TYPEDEF, REGSPACE)	pECMERRTGTRnReg	Pointer points to ECMERRTGTR[n] register
	VAR(uint8, TYPEDEF)	ucErrorTargetBitNo	Bit index assigned to the error signal
	VAR(uint8, TYPEDEF)	ucErrorStatusRegNo	Status register number to which a specific error signal belongs
Description:	A structure contains the individual information for each error signal.		

18.5.2.6 **CddEmm_ErrorCountInitialSettingType**

Table 18-7 CddEmm_ErrorCountInitialSettingType

Name:	CddEmm_ErrorCountInitialSettingType		
Type:	structure		
Elements:	**Type**	**Name**	**Explanation**
	P2VAR(uint32, TYPEDEF, REGSPACE)	pECMERRCNTRnReg	Pointer points to ECMERRCNTRn register.
	VAR(uint8, TYPEDEF)	ucCountVal	Count value of error signal
	VAR(CddEmm_ErrorCountType, TYPEDEF)	enErrorCountType	Type of error count function (1-bit error or multi-bit error)
Description:	A structure contains the setting information for error count function.		

18.5.2.7 **CddEmm_ConfigType**

Table 18-8 CddEmm_ConfigType

Name:	CddEmm_ConfigType	
Type:	Structure	
Elements:	VAR(uint32, TYPEDEF)	ulStartOfDbToc

	P2CONST(void, TYPEDEF, CDDEMM_APPL_CONST)	pErrorRegisterInitialSetting
	P2CONST(void, TYPEDEF, CDDEMM_APPL_CONST)	pErrorSignalConfig
	P2CONST(void, TYPEDEF, CDDEMM_APPL_CONST)	pErrorCountInitialSetting
	VAR(uint8, TYPEDEF)	ucNoOfErrorCount
Description:	Contains the desired configuration set for the EMM	

18.5.2.8 **CddEmm_GetErrorCountType**

Table 18-9 CddEmm_GetErrorCountType

Name:	CddEmm_GetErrorCountType	
Type:	structure	
Elements:	VAR(uint8, TYPEDEF)	ucError1Bit
	VAR(uint8, TYPEDEF)	ucErrorMultiBit
Descriptions	This structure contains the value of error count.	

18.5.2.9 **CddEmm_ErrorRegisterInitialSettingType**

Table 18-10 CddEmm_ErrorRegisterInitialSettingType

Name:	CddEmm_ErrorRegisterInitialSettingType	
Type:	structure	
Elements:	P2VAR(uint32, TYPEDEF, REGSPACE)	pErrStatusRegAddr
	P2VAR(uint32, TYPEDEF, REGSPACE)	pErrControlRegAddr
	P2VAR(uint32, TYPEDEF, REGSPACE)	pErrTargetRegAddr
	VAR(uint32, TYPEDEF)	ulErrControlVal
	VAR(uint32, TYPEDEF)	ulErrTargetVal
	VAR(uint32, TYPEDEF)	ulErrControlMask
Description:	A structure contains the information ECM Error Registers for initialization include (Error Status Address, Error Control Address, Error Target Address, Initial vaule of Error Control, Initial value of Error Target, Value of Mask for Error Control initial).	

18.5.2.10 **CddEmm_ControlCounterType**

Table 18-11 CddEmm_ControlCounter

Location:	CDD_Emm_PBTypes.h	
Type:	Enumeration	
Elements:	CDDEMM_COUNTER_DISABLE	Disable error counter
	CDDEMM_COUNTER_ENABLE	Enable error counter
Description:	This type is used to control the counter.	

18.5.2.11 **CddEmm_SelectCounterType**

Table 18-12 CddEmm_SelectCounterType

Location:	CddEmm_SelectCounterType	
Type:	Enumeration	
Elements:	CDDEMM_HOLD_COUNTER	Select Hold counter
	CDDEMM_MASK_COUNTER	Select Mask counter
Description:	This type is used to select the counter.	

18.5.3 Function Definitions

Table 18-13 APIs provided by the EMM Complex Device Driver Component

Sl. No.	APIs
AUTOSAR API	
1	CddEMM_Init
2	CddEMM_GetVersionInfo
RENESAS API	
3	CddEmm_ReadErrorStatus
4	CddEmm_ClearErrorStatus
5	CddEmm_SetTarget
6	CddEmm_SupportPseudoError
7	CddEmm_SetPseudoErrorSignal
8	CddEmm_ClearPseudoErrorSignal
9	CddEmm_GetCurrentErrorCountUpValue
10	CddEmm_SetHoldMaskCounter
11	CddEmm_SupportControlExternalErrorRequest

18.5.3.1 **CddEmm_ReadErrorStatus**

Table 18-14 CddEmm_ReadErrorStatus

Name:	CddEmm_ReadErrorStatus		
Prototype:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_ReadErrorStatus (P2VAR(uint32, AUTOMATIC, CDDEMM_APPL_DATA) LpData, P2VAR(uint8, AUTOMATIC, CDDEMM_APPL_DATA) LpErrorStatusCount)		
Service Id:	0x02		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Parameters (In):	None		
Parameters (In-Out):	**Type**	**Parameter**	**Value/Range**
	uint32 *	LpData	- Valid pointer to store error status. - LpData != NULL_PTR **Note**:

			To avoid the conflict memory that's accessed by LpData pointer. The input parameter shall be address of an array with identified total of Elements. Recommend that the usage way of CddEmm_ReadErrorStatus API is presented in Sample Application.
	uint8 *	LpErrorStatusCount	- Valid pointer to store the number of error status domain. - LpErrorStatusCount != NULL_PTR
Parameters (Out):	None		
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: Read error status successfully E_NOT_OK: Development error occurred	
Description:	This function is called to read safety-related error status from each module in R-Car Series, 4rd Generation SoC. The number of error status domain is specified by value stored in LpErrorStatusCount pointer. R-Car V4H supports to handle 40 error domains, so the returned value stored in LpErrorStatusCount pointer is 40 and data buffer (LpData) shall be extracted as below: LpData[0]: Indicates error status of ECMERRSTSR0 register (Error Domain 0) LpData[1]: Indicates error status of ECMERRSTSR1 register (Error Domain 1) LpData[2]: Indicates error status of ECMERRSTSR2 register (Error Domain 2) LpData[3]: Indicates error status of ECMERRSTSR3 register (Error Domain 3) LpData[4]: Indicates error status of ECMERRSTSR4 register (Error Domain 4) LpData[5]: Indicates error status of ECMERRSTSR5 register (Error Domain 5) LpData[6]: Indicates error status of ECMERRSTSR6 register (Error Domain 6) LpData[7]: Indicates error status of ECMERRSTSR7 register (Error Domain 7) LpData[8]: Indicates error status of ECMERRSTSR8 register (Error Domain 8) LpData[9]: Indicates error status of ECMERRSTSR9 register (Error Domain 9) LpData[10]: Indicates error status of ECMERRSTSR10 register (Error Domain 10) LpData[11]: Indicates error status of ECMERRSTSR11 register (Error Domain 11) LpData[12]: Indicates error status of ECMERRSTSR12 register (Error Domain 12) LpData[13]: Indicates error status of ECMERRSTSR13 register (Error Domain 13) LpData[14]: Indicates error status of ECMERRSTSR16 register (Error Domain 16) LpData[15]: Indicates error status of ECMERRSTSR17 register (Error Domain 17) LpData[16]: Indicates error status of ECMERRSTSR18 register (Error Domain 18) LpData[17]: Indicates error status of ECMERRSTSR19 register (Error Domain 19) LpData[18]: Indicates error status of ECMERRSTSR20 register (Error Domain 20) LpData[19]: Indicates error status of ECMERRSTSR21 register (Error Domain 21) LpData[20]: Indicates error status of ECMERRSTSR22 register (Error Domain 22) LpData[21]: Indicates error status of ECMERRSTSR23 register (Error Domain 23) LpData[22]: Indicates error status of ECMERRSTSR24 register (Error Domain 24) LpData[23]: Indicates error status of ECMERRSTSR25 register (Error Domain 25) LpData[24]: Indicates error status of ECMERRSTSR26 register (Error Domain 26) LpData[25]: Indicates error status of ECMERRSTSR27 register (Error Domain 27)		

	LpData[26]: Indicates error status of ECMERRSTSR28 register (Error Domain 28) LpData[27]: Indicates error status of ECMERRSTSR29 register (Error Domain 29) LpData[28]: Indicates error status of ECMERRSTSR30 register (Error Domain 30) LpData[29]: Indicates error status of ECMERRSTSR31 register (Error Domain 31) LpData[30]: Indicates error status of ECMERRSTSR32 register (Error Domain 32) LpData[31]: Indicates error status of ECMERRSTSR33 register (Error Domain 33) LpData[32]: Indicates error status of ECMERRSTSR34 register (Error Domain 34) LpData[33]: Indicates error status of ECMERRSTSR35 register (Error Domain 35) LpData[34]: Indicates error status of ECMERRSTSR36 register (Error Domain 36) LpData[35]: Indicates error status of ECMERRSTSR38 register (Error Domain 38) LpData[36]: Indicates error status of ECMERRSTSR39 register (Error Domain 39) LpData[37]: Indicates error status of ECMERRSTSR40 register (Error Domain 40) LpData[38]: Indicates error status of ECMERRSTSR41 register (Error Domain 41) LpData[39]: Indicates error status of ECMERRSTSR42 register (Error Domain 42)
Dependency:	None
Preconditions:	The EMM Complex Driver must be initialized.

18.5.3.2 **CddEmm_ClearErrorStatus**

Table 18-15 CddEmm_ClearErrorStatus

Name:	CddEmm_ClearErrorStatus	
Prototype:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_ClearErrorStatus (void)	
Service Id:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	None	
Parameters (In-Out):	None	
Parameters (Out):	None	
Return Value:	**Type**	**Possible Return Values**
	Std_ReturnType	E_OK: Clear error status successfully E_NOT_OK: Development error occurred
Description:	This function is to clear status of all safety-related errors.	
Configuration Dependency:	None	
Preconditions:	The EMM Complex Driver must be initialized.	

18.5.3.3 **CddEmm_SetTarget**

Table 18-16 CddEmm_SetTarget

Name:	CddEmm_SetTarget
Prototype:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_SetTarget (VAR(CddEmm_ErrorIDType, AUTOMATIC) LulErrorSignalId,

	VAR(CddEmm_TargetType, AUTOMATIC) LenTarget)		
Service Id:	0x03		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	CddEmm_ErrorIDType	LulErrorSignalId	0..<Maximum number of error signal enabled in Generation Tool> Each error has a unique ID which generated in CDD_Emm_Cfg.h
	CddEmm_TargetType	LenTarget	CDDEMM_TARGET_ERROROUT CDDEMM_TARGET_INTC
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: Set target successfully E_NOT_OK: Development error occurred	
Description:	This function selects the destination of error as a target to be notified in case of error occurred, select either the system interrupt controller (CDDEMM_TARGET_INTC) or external pin (CDDEMM_TARGET_ERROROUT).		
Configuration Dependency:	The ID of error signal is only generated in CDD_Emm_Cfg.h if the corresponding 'CddEmmErrorSignalEnable' configuration parameter of that error is enabled.		
Preconditions:	The EMM Complex Driver must be initialized.		

18.5.3.4 **CddEmm_SupportPseudoError**

Table 18-17 CddEmm_SupportPseudoError

Name:	CddEmm_SupportPseudoError		
Prototype:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_SupportPseudoError (VAR(CddEmm_PseudoErrorModeType, AUTOMATIC) LenMode)		
Service Id:	0x04		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	CddEmm_PseudoErrorModeType	LenMode	CDDEMM_PSEUDO_ERROR_DISABLE CDDEMM_PSEUDO_ERROR_ENABLE
Parameters (In-Out):	None		

Parameters (Out):	None	
Return Value:	**Type**	**Possible Return Values**
	Std_ReturnType	E_OK: Enable/Disable pseudo error functionality successfully. E_NOT_OK: Development error occurred
Description:	This function enables/disables a pseudo error functionality.	
Configuration Dependency:	'CddEmmPseudoErrorApi' configuration parameter under 'CddGeneral' container shall be enabled if this API is used. The default value is enabled.	
Preconditions:	The EMM Complex Driver must be initialized.	

18.5.3.5 **CddEmm_SetPseudoErrorSignal**

Table 18-18 CddEmm_SetPseudoErrorSignal

Name:	CddEmm_SetPseudoErrorSignal		
Prototype:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_SetPseudoErrorSignal (VAR(CddEmm_ErrorIDType, AUTOMATIC) LulErrorSignalId)		
Service Id:	0x05		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	CddEmm_ErrorIDType	LulErrorSignalId	0..<Maximum number of error signal enabled in Generation Tool> Each error has a unique ID which generated in CDD_Emm_Cfg.h
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: Set a specific pseudo error successfully E_NOT_OK: Development error occurred	
Description:	This function sets a specific pseudo error signal.		
Configuration Dependency:	'CddEmmPseudoErrorApi' configuration parameter under 'CddGeneral' container shall be enabled if this API is used. The default value is enabled.		
Preconditions:	Following APIs must be called prior to this API: 1. EMM Complex Driver must be initialized (CddEmm_Init) 2. Pseudo error functionality must be enabled (CddEmm_SupportPseudoError)		

18.5.3.6 **CddEmm_ClearPseudoErrorSignal**

Table 18-19 CddEmm_ClearPseudoErrorSignal

Name:	CddEmm_ClearPseudoErrorSignal		
Prototype:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_ClearPseudoErrorSignal (VAR(CddEmm_ErrorIDType, AUTOMATIC) LulErrorSignalId)		
Service Id:	0x06		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	CddEmm_ErrorIDType	LulErrorSignalId	0..<Maximum number of error signal enabled in Generation Tool> Each error has a unique ID which generated in CDD_Emm_Cfg.h
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: Clear a specific pseudo error successfully. E_NOT_OK: Development error occurred	
Description:	This function clears a specific pseudo error.		
Configuration Dependency:	'CddEmmPseudoErrorApi' configuration parameter under 'CddGeneral' container shall be enabled if this API is used. The default value is enabled.		
Preconditions:	Following APIs must be called prior to this API: 1. EMM Complex Driver must be initialized (CddEmm_Init) 2. Pseudo error functionality must be enabled (CddEmm_SupportPseudoError)		

18.5.3.7 **CddEmm_GetCurrentErrorCountUpValue**

Table 18-20 CddEmm_GetCurrentErrorCountUpValue

Name:	CddEmm_GetCurrentErrorCountUpValue
Prototype:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_GetCurrentErrorCountUpValue (P2VAR(CddEmm_GetErrorCountType, AUTOMATIC, CDDEMM_APPL_DATA) LpDataErrorCount, P2VAR(uint8, AUTOMATIC, CDDEMM_APPL_DATA) LpNumErrorCount)
Service Id:	0x0C

<table>
<tr><td>Sync/Async:</td><td colspan="3">Synchronous</td></tr>
<tr><td>Reentrancy:</td><td colspan="3">Non Reentrant</td></tr>
<tr><td>Parameters (In):</td><td colspan="3">None</td></tr>
<tr><td rowspan="3">Parameters (In-Out):</td><td>Type</td><td>Parameter</td><td>Value/Range</td></tr>
<tr><td>CddEmm_GetErrorCountType *</td><td>LpDataErrorCount</td><td>- Valid pointer to store current 1-bit error count value and multi-bit error count value on error count registers.
- LpDataErrorCount !=NULL_PTR
Note: To avoid the conflict memory that accessed by LpDataErrorCount pointer, the input parameter shall be address of an array with identified total of Elements. Recommend that the usage way of CddEmm_GetCurrentErrorCountUpValue API is presented in Sample Application.</td></tr>
<tr><td>Uint8*</td><td>LpNumErrorCount</td><td>- Valid pointer to store a number of error count register.
- LpNumErrorCount ! = NULL_PTR</td></tr>
<tr><td>Parameters (Out):</td><td colspan="3">None</td></tr>
<tr><td rowspan="2">Return Value:</td><td>Type</td><td colspan="2">Possible Return Values</td></tr>
<tr><td>Std_ReturnType</td><td colspan="2">E_OK: Read error count value successfully.
E_NOT_OK: Development error occurred</td></tr>
<tr><td>Description:</td><td colspan="3">This function is called to read 1-bit error count value and multi-bit error count value of all ECM Error Count Registers. The number of ECM Error Count Registers is specified by value stored in LpNumErrorCount pointer.

[R-Car V4H]
R-Car V4H supports to read 49 error count registers, so data buffer shall be extracted as below:
LpDataErrorCount[0] – Value of error count on ECMERRCNTR0 register
LpDataErrorCount[1] – Value of error count on ECMERRCNTR1 register
LpDataErrorCount[2] – Value of error count on ECMERRCNTR3 register
LpDataErrorCount[3] – Value of error count on ECMERRCNTR4 register
LpDataErrorCount[4] – Value of error count on ECMERRCNTR5 register
LpDataErrorCount[5] – Value of error count on ECMERRCNTR6 register
LpDataErrorCount[6] – Value of error count on ECMERRCNTR7 register
LpDataErrorCount[7] – Value of error count on ECMERRCNTR8 register
LpDataErrorCount[8] – Value of error count on ECMERRCNTR9 register
LpDataErrorCount[9] – Value of error count on ECMERRCNTR10 register</td></tr>
</table>

	LpDataErrorCount[10] – Value of error count on ECMERRCNTR11 register LpDataErrorCount[11] – Value of error count on ECMERRCNTR12 register LpDataErrorCount[12] – Value of error count on ECMERRCNTR13 register LpDataErrorCount[13] – Value of error count on ECMERRCNTR14 register LpDataErrorCount[14] – Value of error count on ECMERRCNTR15 register LpDataErrorCount[15] – Value of error count on ECMERRCNTR16 register LpDataErrorCount[16] – Value of error count on ECMERRCNTR17 register LpDataErrorCount[17] – Value of error count on ECMERRCNTR18 register LpDataErrorCount[18] – Value of error count on ECMERRCNTR19 register LpDataErrorCount[19] – Value of error count on ECMERRCNTR20 register LpDataErrorCount[20] – Value of error count on ECMERRCNTR21 register LpDataErrorCount[21] – Value of error count on ECMERRCNTR22 register LpDataErrorCount[22] – Value of error count on ECMERRCNTR25 register LpDataErrorCount[23] – Value of error count on ECMERRCNTR26 register LpDataErrorCount[24] – Value of error count on ECMERRCNTR27 register LpDataErrorCount[25] – Value of error count on ECMERRCNTR28 register LpDataErrorCount[26] – Value of error count on ECMERRCNTR29 register LpDataErrorCount[27] – Value of error count on ECMERRCNTR30 register LpDataErrorCount[28] – Value of error count on ECMERRCNTR31 register LpDataErrorCount[29] – Value of error count on ECMERRCNTR32 register LpDataErrorCount[30] – Value of error count on ECMERRCNTR33 register LpDataErrorCount[31] – Value of error count on ECMERRCNTR34 register LpDataErrorCount[32] – Value of error count on ECMERRCNTR35 register LpDataErrorCount[33] – Value of error count on ECMERRCNTR36 register LpDataErrorCount[34] – Value of error count on ECMERRCNTR37 register LpDataErrorCount[35] – Value of error count on ECMERRCNTR38 register LpDataErrorCount[36] – Value of error count on ECMERRCNTR39 register LpDataErrorCount[37] – Value of error count on ECMERRCNTR40 register LpDataErrorCount[38] – Value of error count on ECMERRCNTR41 register LpDataErrorCount[39] – Value of error count on ECMERRCNTR42 register LpDataErrorCount[40] – Value of error count on ECMERRCNTR43 register LpDataErrorCount[41] – Value of error count on ECMERRCNTR44 register LpDataErrorCount[42] – Value of error count on ECMERRCNTR45 register LpDataErrorCount[43] – Value of error count on ECMERRCNTR46 register LpDataErrorCount[44] – Value of error count on ECMERRCNTR47 register LpDataErrorCount[45] – Value of error count on ECMERRCNTR48 register LpDataErrorCount[46] – Value of error count on ECMERRCNTR49 register LpDataErrorCount[47] – Value of error count on ECMERRCNTR50 register LpDataErrorCount[48] – Value of error count on ECMERRCNTR51 register And the value of LpNumErrorCount pointer shall be 49
Configuration Dependency:	None
Preconditions:	1. The EMM Complex Driver must be initialized. 2. At least one error signal of CddEmmDomain*n* (n=10, 11, 12, 13) should be configured as true to use this API.

18.5.3.8 **CddEmm_SetHoldMaskCounter**

Table 18-21 CddEmm_SetHoldMaskCounter

Function Name:	CddEmm_SetHoldMaskCounter		
Syntax:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_SetHoldMaskCounter (VAR(uint16, AUTOMATIC) LuiTimeValue, VAR(CddEmm_SelectCounterType, AUTOMATIC) LenCounterSel)		
Service Id:	0x0E		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LuiTimeValue	Hold/Mask the error status in a period.	
		Range:	1…2047
	LenCounterSel	Select hold counter or mask counter for setting.	
		Range:	CDDEMM_HOLD_COUNTER: Hold counter CDDEMM_MASK_COUNTER: Mask counter
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	E_OK	Hold/Mask counter is set successfully
		E_NOT_OK	A development error occurred (see DET Errors handled)
Description:	This service sets hold/mask counter of the external error request control.		
Preconditions:	Driver must already be initialized.		
Remarks:	User should not call this API directly. This API should be invoked by Runnable Entity in Basic Software Component.		
DET/DEM Errors handled:	CDDEMM_E_UNINIT		DET error indicates that this driver is not initialized
	CDDEMM_E_PARAM_VALUE		DET error indicates that invalid input parameter.
	CDDEMM_E_COUNTER_MODE		DET error indicates that the hold/mask counter is enabled.

18.5.3.9 **CddEmm_SupportControlExternalErrorRequest**

Table 18-22 CddEmm_SupportControlExternalErrorRequest

Function Name:	CddEmm_SupportControlExternalErrorRequest		
Syntax:	FUNC(Std_ReturnType, CDDEMM_CODE_SLOW) CddEmm_SupportControlExternalErrorRequest (VAR(CddEmm_ControlCounterType, AUTOMATIC) LenCounterMode VAR(CddEmm_SelectCounterType, AUTOMATIC) LenCounterSel)		
Service Id:	0x0F		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LenCounterMode	Enable/Disable hold/mask counter of the external error request control.	
		Range:	CDDEMM_COUNTER_ENABLE: Enable counter CDDEMM_COUNTER_DISABLE: Disable counter
	LenCounterSel	Select hold counter or mask counter for setting.	
		Range:	CDDEMM_HOLD_COUNTER: Hold counter CDDEMM_MASK_COUNTER: Mask counter
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	E_OK	Enable/Disable counter of the external error request control successfully.
		E_NOT_OK	A development error occurred (see DET Errors handled)
Description:	This service enable/disable counter of the external error request control.		
Preconditions:	1. Driver must already be initialized. 2. CddEmm_SetHoldMaskCounter shall be invoked to set the value of hold/mask counter before these counters are enabled. Note: In case that user wants to change the value of hold/mask counter, the counter shall be disabled first. After that, invoke CddEmm_SetHoldMaskCounter to set new value and then hold/mask counter is enabled again.		
Remarks:	User should not call this API directly. This API should be invoked by Runnable Entity in Basic Software Component.		

DET/DEM Errors handled:	CDDEMM_E_UNINIT	DET error indicates that this driver is not initialized
	CDDEMM_E_PARAM_VALUE	DET error indicates that invalid input parameter.

18.5.4 Preemption of APIs

The following table specifies the preemption of each API that can be invoked at the same time as the API.

Table 18-23 Preemption of APIs of the EMM Driver

	CddEmm_Init	CddEmm_GetversionInfo	CddEmm_SupportPseudoError	CddEmm_SetPseudoErrorSignal	CddEmm_SetTarget	CddEmm_ReadErrorStatus	CddEmm_GetCurrentErrorCountUpValue	CddEmm_ClearPseudoErrorSignal	CddEmm_ClearErrorStatus	CddEmm_SupportControlExternalErrorRequest	CddEmm_SetHoldMaskCounter
CddEmm_Init	-										
CddEmm_GetversionInfo	✓	-									
CddEmm_SupportPseudoError	-	✓	-								
CddEmm_SetPseudoErrorSignal	-	✓	✓	-							
CddEmm_SetTarget	-	✓	✓	✓	-						
CddEmm_ReadErrorStatus	-	✓	✓	✓	✓	-					
CddEmm_GetCurrentErrorCountUpValue	-	✓	✓	✓	✓	✓	-				
CddEmm_ClearPseudoErrorSignal	-	✓	✓	✓	✓	✓	✓	-			
CddEmm_ClearErrorStatus	-	✓	✓	✓	✓	✓	✓	✓	-		
CddEmm_SupportControlExternalErrorRequest	-	✓	✓	✓	✓	✓	✓	✓	✓	-	
CddEmm_SetHoldMaskCounter	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	-

-: cannot be invoked at the same time

✓: can be invoked at the same time

18.6 Development and Production Errors

In this chapter the development errors that are reported by the EMM Complex Device Driver Component are tabulated. The development errors will be reported only when the pre-compiler option CddEmmDevErrorDetect is enabled in the configuration. The production code errors are not supported by EMM Complex Device Driver Component.

18.6.1 EMM Complex Device Driver Component Development Errors

The development errors will be reported only when the pre-compile option 'CddEmmDevErrorDetect' is enabled in the configuration. The following table contains the DET errors that are reported by EMM Complex Driver Component. These errors are reported to DET Module when the EMM Complex Driver Component APIs is invoked with wrong input parameters or without initialization of the driver.

Table 18-24 DET Errors of EMM Complex Device Driver Component

Sl. No.	**1**
Error Code	CDDEMM_E_UNINIT
Related API(s)	CddEmm_ReadErrorStatus, CddEmm_SetTarget, CddEmm_SupportPseudoError, CddEmm_SetPseudoErrorSignal, CddEmm_ClearPseudoErrorSignal, CddEmm_ClearErrorStatus, CddEmm_GetCurrentErrorCountUpValue.
Source of Error	When the API service is invoked before initialization.
Origin	Renesas
Sl. No.	**2**
Error Code	CDDEMM_E_ALREADY_INITIALIZED
Related API(s)	CddEmm_Init
Source of Error	When CddEmm_Init is invoked while EMM Complex Driver module is already initialized.
Origin	Renesas
Sl. No.	**3**
Error Code	CDDEMM_E_PARAM_POINTER
Related API(s)	CddEmm_Init, CddEmm_ReadErrorStatus, CddEmm_GetVersionInfo, CddEmm_GetCurrentErrorCountUpValue
Source of Error	When the API service is invoked with invalid pointer.
Origin	Renesas
Sl. No.	**4**
Error Code	CDDEMM_E_INVALID_DATABASE
Related API(s)	CddEmm_Init
Source of Error	When the API service is invoked with invalid database address.
Origin	Renesas
Sl. No.	**5**
Error Code	CDDEMM_E_PARAM_VALUE
Related API(s)	CddEmm_SetTarget, CddEmm_SupportPseudoError, CddEmm_SetPseudoErrorSignal, CddEmm_ClearPseudoErrorSignal.
Source of Error	When the API service is invoked with invalid value parameter.
Origin	Renesas
Sl. No.	**6**
Error Code	CDDEMM_E_PSEUDO_ERROR_MODE
Related API(s)	CddEmm_SetPseudoErrorSignal, CddEmm_ClearPseudoErrorSignal.
Source of Error	When set/clear a pseudo error signal before enabling the pseudo-error functionality.
Origin	Renesas
Sl. No.	**7**

Error Code	CDDEMM_E_PSEUDO_ERROR_GEN
Related API(s)	CddEmm_SetPseudoErrorSignal
Source of Error	When a specific pseudo error signal is not generated after invoking CddEmm_SetPseudoErrorSignal API.
Origin	Renesas
Sl. No.	**8**
Error Code	CDDEMM_E_COUNTER_MODE
Related API(s)	CddEmm_SetHoldMaskCounter
Source of Error	When CddEmm_SetHoldMaskCounter with valid input parameters is invoked during CddEmm_GblHoldCounterRequestMode or CddEmm_GblMaskCounterRequestMode is CDDEMM_ENABLE.
Origin	Renesas

18.6.2 EMM Complex Device Driver Component Production Errors

The following table contains Renesas specific DEM errors that are reported by EMM Complex Driver Component.

Table 18-25 DEM Errors of EMM Complex Device Driver Component

Sl. No.	**1**
Error Code	CDDEMM_E_WRITE_VERIFY_FAILURE
Related API(s)	CddEmm_Init, CddEmm_SetTarget, CddEmm_SupportPseudoError, CddEmm_SetPseudoErrorSignal
Source of Error	Monitors register write failure. DEM_EVENT_STATUS_FAILED: When register read-back value does not match with expected value.
Origin	Renesas
Sl. No.	**2**
Error Code	CDDEMM_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	CDDEMM_DOMAINn_ISR/ ISR (CDDEMM_DOMAINn_CAT2_ISR) (n=0..13, 16..36, 38..42)
Source of Error	When the Shared Peripheral Interrupt (SPI) received via INTC-AP in INTC is not really initiated via ECM before processing the software operation for interrupt handling.
Origin	Renesas

18.7 EMM Complex Device Driver Component Runtime Errors

AUTOSAR does not define any runtime error code for EMM Driver.

18.8 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of EMM Complex Driver Component software.

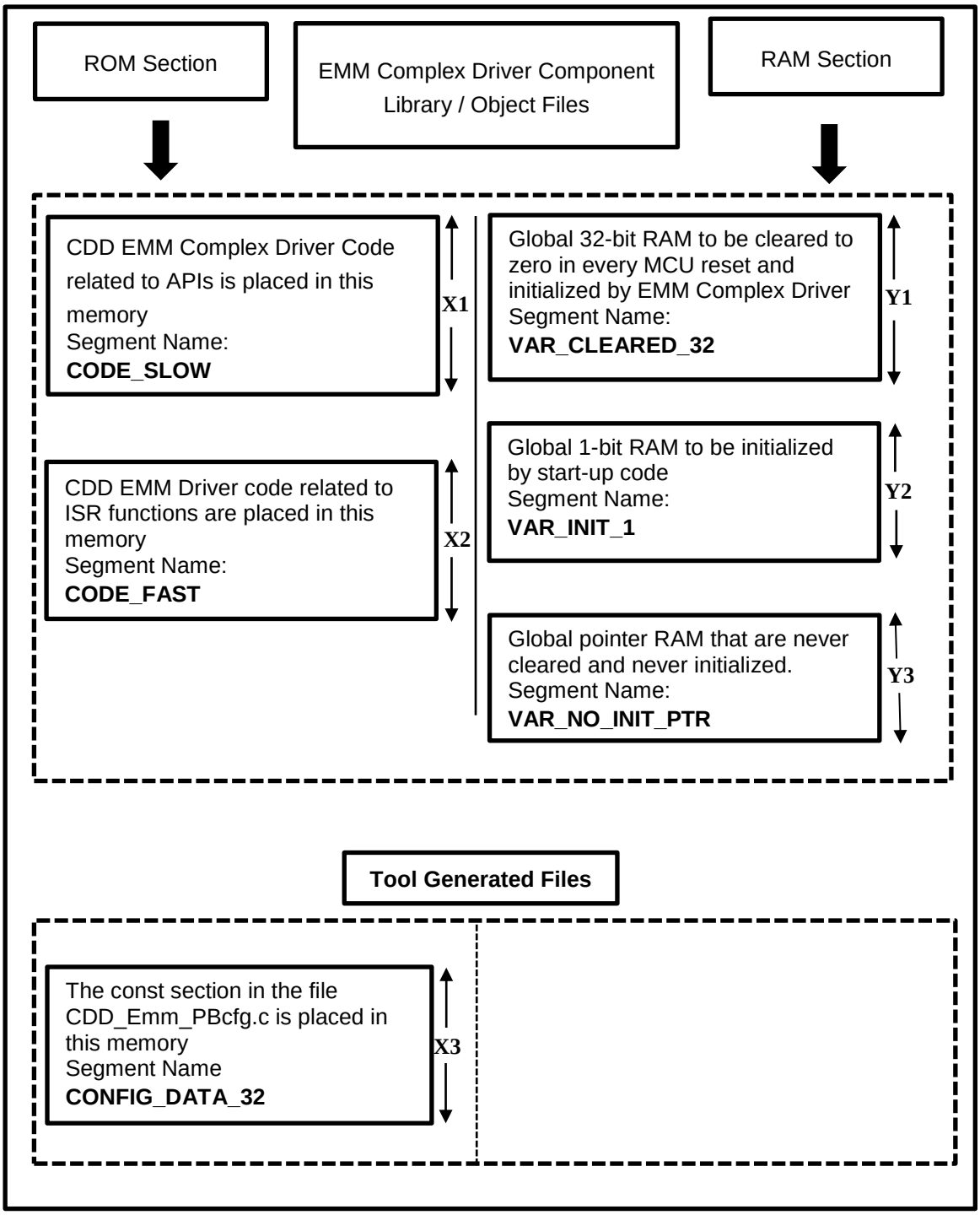


Figure 18-4 EMM Complex Device Driver Component Memory Organization

ROM Section (X1 to X3):

CODE_SLOW (X1): API(s) of EMM Complex Driver Component except for ISR functions.

CODE_FAST (X2): This section consists of ECM error ISR functions that can be located in code memory.

CONFIG_DATA_32 (X3): This section consists of EMM Complex Driver Component specific configuration CONST structures (32-bit maximum data size) generated by EMM Complex Driver component generation tool. This can be located in the code memory.

RAM Section (Y1 to Y3):

VAR_CLEARED_32 (Y1): This section consists of the global RAM variables of 32-bit maximum data size that are cleared to zero after every MCU reset and initialized internally by EMM Complex Driver Component. This can be located in data memory.

VAR_INIT_1 (Y2): This section consists of the global RAM variables of 1-bit size that are initialized by start-up code and used internally by EMM Complex Driver Component. This can be located in data memory.

VAR_NO_INIT_PTR (Y3): This section consists of the global RAM pointer variable that are never cleared and never initialized by EMM Complex Driver Component. This can be located in data memory.

Note: User must ensure that none of the memory areas overlap with each other. Even 'debug' information should not overlap.

18.9 Device Specific Information

The device supports the following devices:
Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 5.1 Product*.

18.9.1 Interaction Between The User And EMM Complex Device Driver Component

The detail of the services supported by the EMM Complex Device Driver Component to the upper layers users is provided in the following chapter.

18.9.1.1 Channel Mapping

None.

18.9.1.2 ISR Function

The table below provides the list of handlers corresponding to the hardware unit ISR(s) in EMM Complex Device Driver Component.

Table 18-26 ISR Handler Addresses

Interrupt Source (<n>=0..13, 16..36, 38..42)	**Name of the ISR Function (<n>= 0..13, 16..36, 38..42)**
ECM error<n>	CDDEMM_DOMAIN<n>_ISR
	CDDEMM_DOMAIN<n>_CAT2_ISR

18.9.2 Multi-Core / Multi-Instantiation

EMM driver does not support multi-core and multi-instantiation for this device.

18.10 Non-AUTOSAR environment integration

The EMM Complex Driver Components for Renesas R-Car Series, 4th Generation SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the EMM Complex Driver Components to Non-AUTOSAR environment.

18.10.1 Stub modules handling

18.10.1.1 Os

The Os stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar version>\Os

In the AUTOSAR environment, EMM Complex Driver Components must include “Os.h” header file to obtain the interrupt category information configured in the OS.

18.10.1.2 Det

The Det stub files are organized in the following folder: \rel\common\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, EMM Complex Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g. EMM Complex Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:
Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

18.10.1.3 Dem

The Dem stub files are organized in the following folder: \rel\common\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, EMM Complex Driver Components uses Dem_ReportErrorStatus API provided by the DEM module to report a production error. The API prototype is as of follow:
Dem_ReportErrorStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_ReportErrorStatus API with their current error handling strategy.

18.10.1.4 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.
The SchM (Basic Software Scheduler) stub files are organized in the following folder:
rel\common\generic\stubs\19_11\Rte\

EMM Complex Device Driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when CDDEMM_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user EMM Complex Device Driver as following:
Write SchM functions with prototype as below:
void SchM_Enter_CDDEMM_INTERRUPT_CONTROL_PROTECTION (void);
void SchM_Exit_CDDEMM_INTERRUPT_CONTROL_PROTECTION (void);

18.10.2 Callback function usage

The EMM Complex Driver Component does not use any callback functions.

18.10.3 Scheduled function usage

The EMM Complex Driver Component does not use any scheduled functions.

18.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:
\rel\V4H\common_family\include\arm\Interrupt_VectorTable.h
\rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

Non-AUTOSAR users shall use CDDEMM_DOMAIN<n>_ISR as specified in chapter *18.9.1.2 ISR Function*.

19. RFSO

19.1 Overview

The purpose of this document is to describe the information related to RFSO Complex Driver Component.

This document is intended for the developers of ECU software on R-Car Series, 4th Generation SoC using Application Programming Interfaces provided by AUTOSAR specification for RFSO Complex Driver.

The users of RFSO Complex Driver Component shall use this document as reference. This document describes the common features of RFSO Complex Driver Component.

The RFSO (Failure Self-Detection Output) is a safety module which consists Interval and Time-out Timers and has output signals to the outside of the SoC. This module can be used as a trigger and time-out detection for Runtime Test, trigger for Periodical Checks, internal watchdog timer, and for fault notification to ECM module.

The RFSO Complex Driver Component provides the following services:

- RFSO Complex Driver Component initialization.
- Interval Timer and Time-out Timer control functionality:
  - Start/Stop timer
  - Timer configuration
- Get Interval Timer and Time-out Timer counter value
- Clock division of RFSO channel configuration
- Interval timer Interrupt control
- Get Interval timer Interrupt status
- Output to ECM module with related bits control and examination
- Get version information

The RFSO Complex Driver Component comprises of two parts: Embedded Software and Generation Tool to achieve scalability and configurability.

The purpose of this document is to describe the information related to Embedded Software part of the RFSO Complex Driver Component for R-Car Series, 4th Generation SoC. Please refer to "*RFSO Complex Driver Component Generation Tool User's Manual"* for the detail of Generation Tool part.
The below figure depicts the RFSO Complex Device Driver as part of layered AUTOSAR MCAL Layer:

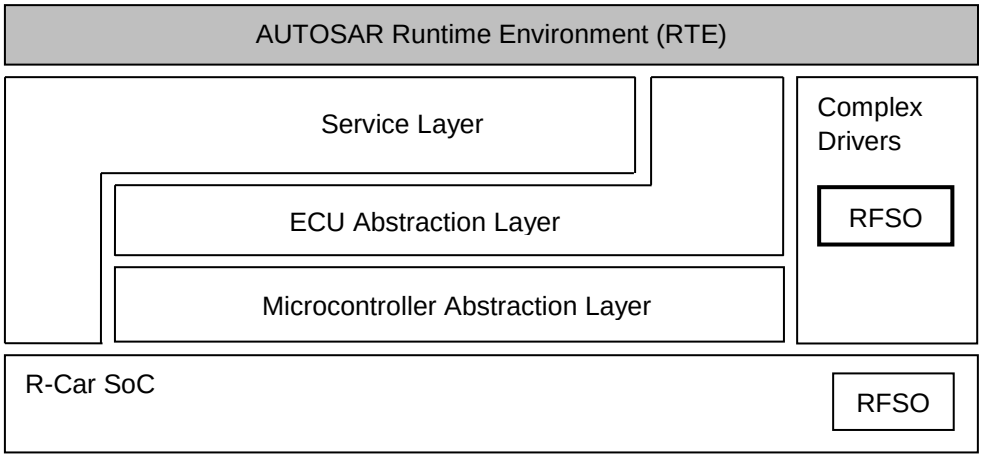


Figure 19.1 System Overview of the RFSO Complex Device Driver in AUTOSAR Layer Architecture

The following diagram shows the system overview of the AUTOSAR Architecture for RFSO Complex Driver :

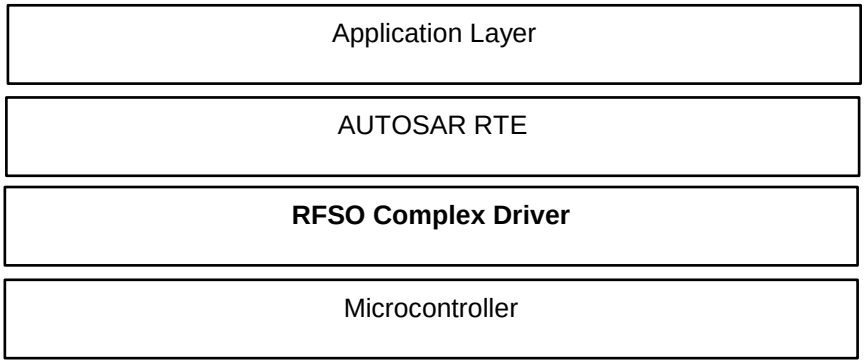


Figure 19.2 System Overview of AUTOSAR Architecture

Sample Application is available for user to understand and test RFSO Complex Device Driver without proper AUTOSAR upper layer, which is described in section *"Sample Application"* of this document.

Please refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" - section 3.7.2.2* How to build the Sample Application for more information on how to configure, compile and flash the RFSO Complex Driver to R-Car SoC.

19.2 Forethoughts

19.2.1 General

Following information will aid the user to use the RFSO Complex Driver Component software efficiently:

- Example code mentioned in this document shall be taken only as a reference for implementation.
- MCU specific initializations such as reset registers, once writable registers, interrupt stack pointer, user stack pointer, internal watchdog, specific features of internal memory and registers is not implemented

by RFSO Complex Driver. These initializations must be implemented by the start-up code.

- The ISR functions and the corresponding handler addresses are provided in *19.9.1.2 ISR Functions*. User should ensure that Interrupt Vector table configuration is done as per the information provided in the table.
- Porting specific information such as compiler details, timing details and memory consumption are provided in *11.9 Device Specific Information*.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- If any development errors are reported to DET, the normal flow of execution of driver will be aborted.
- All production errors will be reported to DEM by using the API Dem_SetEventStatus provided by DEM.
- All interrupt-related functions are mapped to CODE_FAST section (see *Table 7.12* of *AUTOSAR Specification of Memory Mapping*). Other functions are mapped to CODE_SLOW section (see *Table 7.13* of *AUTOSAR Specification of Memory Mapping*).

19.2.2 Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the RFSO Complex Driver Component:

- The CddRfso_PBcfg.c, CddRfso_Cbk.h and CddRfso_Cfg.h files generated by the RFSO Complex Driver Component Code Generation Tool must be compiled and linked along with RFSO Complex Driver Component source files.
- The application has to be rebuilt, if there is any change in the CddRfso_PBcfg.c, CddRfso_Cbk.h and CddRfso_Cfg.h files generated by the RFSO Complex Driver Component Code Generation Tool.
- File CddRfso_PBcfg.c generated for single configuration set using RFSO Complex Driver Component Code Generation Tool can be compiled and linked independently.
- The RFSO Complex Driver Component needs to be initialized before accepting any API requests. CddRfso_Init API should be called to initialize RFSO Complex Driver Component.
- ISR priority should be configured to be greater than priority of the thread/task which invoke the driver's APIs. Otherwise, the driver operation is not guaranteed.
- The user should ensure that CDD RFSO Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Validation of input parameters is done only when the static configuration parameter CDDRFSO_DEV_ERROR_DETECT is enabled. Application should ensure that the right parameters are passed while invoking the APIs when CDDRFSO_DEV_ERROR_DETECT is disabled.
- A mismatch in the version numbers will result in compilation error. Ensure that the correct versions of the header and the source files are used.
- User should use Generation Tool to generate the configuration of RFSO Complex driver. If not, please refer to Tool User Manual for more detail about constraints of parameter.
- APIs of RFSO Complex Driver just supports RFSO channel that already has been configured in CddRfso_PBcfg.c file.
- Due to Hardware description, RFSO Complex Driver does not managed interrupt service routine of Time-out Timer, RFSO Complex Driver just supports to enable time-out interrupt, clear time-out interrupt flag, get current value of timer counter and get status of bit that indicated the raising of interrupt. Raising signal from Time-out Timer will notify to ECM module and be reported to the external systems through ECM external pin (ERROROUT#).
- During the time Time-out Timer or Interval Timer is operating, setting action is prohibited. Therefore, timer must be stopped before re-configuring.

- In One shot mode, Interval Timer interrupt shall be enabled. Otherwise, the corresponding Interval Timer will not work as expected.
- User must initialize the following hardware IPs as below before initializing RFSO module:

Table 19-1 List of hardware IP affect to CDDRFSO

Hardware IP	Module Name	Expected Setting	Description
AXI-bus	-	SDRAM address mapping in HW should be configured according to Section 18.3.1 in HW UM (*R-Car V4H Hardware User Manual*)	RFSO is designed assuming the setting of SDRAM address mapping is corrected (for V4H), so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since RFSO operation is under controlled by DRAM for memory control.
DBSC5	-	The maximum use of SDRAM bus bandwidth will be enabled by DBSC5 according to *Section 33.3* in HW UM(*R-Car V4H Hardware User Manual*).	RFSO is designed assuming the configuration of SDRAM bus bandwidth is suitable for V4H, so they should be configured like that by user in boot loader/ stub/ above layer before using this module. Otherwise, operation of driver is not guaranteed, since RFSO operation is under controlled by DRAM for memory control.
Life Cycle	-	Life Cycle setting for the assignment group of RFSO must be accessed by CR52 master group (for V4H).	RFSO is designed assuming the setting of this protection mechanism allow register to be accessed by CR52 master group (for V4H) , so they should be configured like that by user in boot loader/ stub/ above layer before using this module.

			Otherwise, operation of driver is not guaranteed, since RFSO may not access to registers of RFSO.

19.2.3 Data Consistency

To support the re-entrance and interrupt services, the RFSO Complex Device Driver Component will ensure the data consistency while accessing its own RAM storage or hardware registers.

```
#define CDDRFSO_ENTER_CRITICAL_SECTION(Exclusive Area)
SchM_Enter_CddRfso_##Exclusive_Area()

#define CDDRFSO_EXIT_CRITICAL_SECTION(Exclusive Area)
SchM_Exit_CddRfso_##Exclusive_Area()
```

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

- CDDRFSO_RAM_DATA_PROTECTION
- CDDRFSO_INTERRUPT_CONTROL_PROTECTION

These functions can be disabled by disabling the configuration parameter ‘CddRfsoCriticalSectionProtection’.

19.3 Architecture Details

The RFSO Complex Driver architecture is shown in the following figure. The RFSO Complex Driver user shall directly use the APIs to configure and execute the RFSO Complex Driver operation.

The following diagram shows the RFSO Complex Driver Component as defined in AUTOSAR architecture:

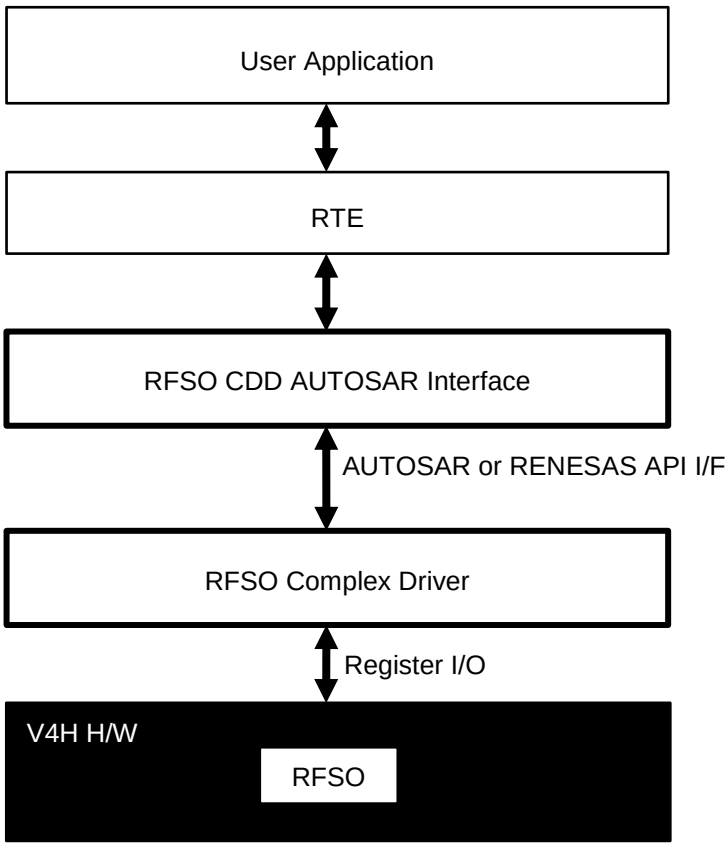


Figure 19.3 RFSO Complex Device Driver Component Architecture

RFSO Complex Driver component

The RFSO Complex Driver provides services for controlling RFSO channel. Each channel contains Interval Timer and Time-out Timer can be used dependently and vice versa. Moreover, RFSO HW provides functionality to control external pins to assert when error happened.

The RFSO Complex Driver component is divided into the following sub feature based on the functionality required:

- Initialization
- Interval Timer and Time-out Timer control functionality:
  - Start/Stop timer
  - Timer configuration
- Get Interval Timer and Time-out Timer counter value
- Clock division of RFSO channel configuration
- Interval timer Interrupt control
- Get Interval timer interrupt status
- Output to ECM module with related bits control and examination
- Get version information

Driver Initialization

The RFSO Complex Driver initialization sub-module internally stores the configuration data address to enable subsequent API calls to access the configuration data and initializes the RFSO channel used by RFSO Component.

The API related to this sub-module is CddRfso_Init().

Interval and Time-out Timer control functionality

RFSO Complex Driver supports to change the operation of timers from operating to non-operating state and vice versa. Driver also provides API to re-configure RFSO channel data when user ensures target timers are not operating.

The API related to this sub-module is CddRfso_IntervalTimeConfigure(), CddRfso_IntervalCycleConfigure(), CddRfso_StartIntervalTimer(), CddRfso_StopIntervalTimer(), CddRfso_TimeoutCycleConfigure(), CddRfso_TimeoutTimeConfigure(), CddRfso_StartTimeoutTimer CddRfso_StopTimeoutTimer().

Get Interval and Time-out Timer counter value

RFSO Complex Driver supports to get current value of timer counter.
The API related to this sub-module is CddRfso_GetTimeoutTimerValue(), CddRfso_GetIntervalTimerValue().

Clock division of RFSO channel configuration

Clock division of each RFSO channel can be re-configured to change overflow time of timers.
The API related to this sub-module is CddRfso_ChannelClockSet().

Time-out Interrupt control

Interrupt sources of Time-out Timer is controlled by MFIS module. Therefore, RFSO Complex Driver just provides functionality to clear interrupt flag of Time-out Timer.
The API related to this sub-module is CddRfso_ClearTimeoutInterrupt().

Interval timer Interrupt control

RFSO Complex Driver provides functionality to enable or disable interrupt of Interval Timer.
The API related to this sub-module is CddRfso_CtrlIntervalTimerInterrupt().

Get Interval timer Interrupt status

RFSO Complex Driver supports to read the status flag of the interrupt of Interval Timer.
The API related to this sub-module is CddRfso_IntervalTimerInterruptStatus ().

External output of RFSO and related bits control and examination

Hardware of RFSO module has external output for fault notification to the external device by setting bits FSO_CTL.CFEO_1 and FSO_CTL.CFEO_0 to 1. CddRfso_ExternalPinControl() API supports to control output signal to ECM Module.

RFSO Complex Driver provides API CddRfso_GetTOESPinStatus() to gets status of TOES Indicates that time-out error status is appropriately output and TOI bit that indicates whether there is a Time-out detection Timer interrupt request or whether there is a Time-out detection Timer clear in the underflow. Besides, CddRfso_GetCFEPinStatus() gets status of CFES_0/CFES_1 bit that indicates feedback value from ECM module for each of 11 channels and also gets status of CFEO_0/CFEO_1 that sets output to external device.

Get version information
This module provides APIs for reading module Id, vendor Id and vendor specific version numbers.
The API related to this module is CddRfso_GetVersionInfo().

19.4 RFSO Complex Driver Component Header and Source File Description

This section explains the RFSO Complex Driver Component's C Source and C Header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by RFSO Complex Driver Generation Tool:
- CDD_Rfso_Cfg.h
- CDD_Rfso_Cbk.h

The C source file generated by RFSO Complex Driver Generation Tool:
CDD_Rfso_PBcfg.c

The RFSO Complex Device Driver Component C header files and Component source files: Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" – *section.3.4.2.3 Folder Strucure*

The Stub C header files and source file: Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 3.2.9 Stubs File*.

19.5 Application Programming Interface

This section explains the Data types and APIs provided by the RFSO Complex Driver Component to the Upper layers.

19.5.1 Imported Types

This chapter explains the Data types imported by the RFSO Complex Driver Component and lists its dependency on other modules.

19.5.1.1 Standard Types

In this chapter all types included from the Std_Types.h (according to "*AUTOSAR Specification of Standard Types"*) are listed:
- Std_ReturnType.
- Std_VersionInfoType.

19.5.1.2 OS Types

In this chapter, all types included from the Os.h (according to AUTOSAR Specification of Operating System)

are listed:

- CounterType
- TickType
- TickRefType

19.5.1.3 Dem Types

In this chapter all types included from the Dem.h (according to "*AUTOSAR Specification of Diagnostic Event Manager*") are listed:

- Dem_EventIdType.
- Dem_EventStatusType.

19.5.2 Type Definitions

This chapter explains the type definitions of RFSO Complex Driver Component.

19.5.2.1 CddRfso_TOESStatusType

Table 19-2 CddRfso_TOESStatusType

Name:	CddRfso_TOESStatusType	
Type:	Structure	
Range:	VAR(boolean, TYPEDEF)	blTOES
	VAR(boolean, TYPEDEF)	blTOI
	VAR(boolean, TYPEDEF)	blTOCUNF
Description:	Structure status of TOES, and TOI and TOCUNF bit.	

19.5.2.2 CddRfso_CFEStatusType

Table 19-3 CddRfso_CFEStatusType

Name:	CddRfso_CFEStatusType	
Type:	Structure	
Elements:	VAR(boolean, TYPEDEF)	blCFEO0
	VAR(boolean, TYPEDEF)	blCFEO1

	VAR(boolean, TYPEDEF)	blCFES0
	VAR(boolean, TYPEDEF)	blCFES1
Description:	Structure status of CFEO_0/CFEO_1/CFES_0/CFES_1 bit.	

19.5.2.3 CddRfso_ConfigType

Table 19-4 CddRfso_ConfigType

Name:	CddRfso_ConfigType	
Type:	Structure	
Elements:	VAR(uint32, TYPEDEF)	ulStartOfDbToc
	P2CONST(CddRfso_ChannelConfigType, TYPEDEF, CDDRFSO_APPL_CONST)	pChannelConfig
	VAR(uint8, TYPEDEF)	ucNumberOfChannelConfig
Description:	Structure contains the configuration of RFSO Complex Driver.	

19.5.2.4 CddRfso_ChannelConfigType

Table 19-5 CddRfso_ChannelConfigType

Name:	CddRfso_ChannelConfigType	
Type:	Structure	
Elements:	VAR(CddRfso_IntervalCbkFuncPtrType, TYPEDEF)	pIntervalTimerCallbackFunction
	VAR(CddRfso_TimerUnitType, TYPEDEF)	enIntervalUnit
	VAR(CddRfso_TimerUnitType, TYPEDEF)	enTimeoutUnit
	VAR(uint32, TYPEDEF)	ulFrequencyDivision
	VAR(uint32, TYPEDEF)	ulIntervalTimerDuration
	VAR(uint32, TYPEDEF)	ulTimeoutTimerMaxDuration
	VAR(uint32, TYPEDEF)	ulTimeoutTimerMinDuration
	VAR(uint8, TYPEDEF)	ucChannelSelection
	VAR(uint8, TYPEDEF)	ucChannelId
	VAR(uint8, TYPEDEF)	ucIntervalTimerOneShot
	VAR(uint8, TYPEDEF)	ucIntervalTimerInterruptEnable
Description:	Structure contains the Channel configuration.	

19.5.2.5 CddRfso_ChannelStatusType

Table 19-6 CddRfso_ChannelStatusType

Name:	CddRfso_ChannelStatusType	
Type:	Structure	
Elements:	VAR(uint8, TYPEDEF)	ucChannelSelection
	VAR(boolean, TYPEDEF)	blIntervalTimerStarted
	VAR(boolean, TYPEDEF)	blIntervalTimerOneShot
	VAR(boolean, TYPEDEF)	blTimeoutTimerStated
	VAR(boolean, TYPEDEF)	blChannelIsBusy
	VAR(boolean, TYPEDEF)	blIntervalTimerInterruptEnable
Description:	Structure contains Channel configuration.	

19.5.2.6 CddRfso_TimerUnitType

Table 19-7 CddRfso_TimerUnitType

Name:	CddRfso_TimerUnitType	
Type:	Enumeration	
Range:	CDDRFSO_MICROSECOND	Microsecond is unit
	CDDRFSO_CYCLE	Cycle is unit
Description:	Represents the unit of channel timer.	

19.5.2.7 CddRfso_CycleType

Table 19-8 CddRfso_CycleType

Name:	CddRfso_CycleType	
Type:	Enumeration	
Range:	CDDRFSO_INTERVAL_CYCLE	Indicate interval timer cycle
	CDDRFSO_TIMEOUT_MAX_CYCLE	Indicate time-out maximum cycle

	CDDRFSO_TIMEOUT_MIN_CYCLE	Indicate time-out minimum cycle
Description:	Represents the cycle type of CDD RFSO.	

19.5.2.8 CddRfso_ReturnType

Table 19-9 CddRfso_ReturnType

Name:	CddRfso_ReturnType	
Type:	Enumeration	
Range:	CDDRFSO_E_OK	Indicate the operation is successful
	CDDRFSO_E_NOT_OK	Indicate the operations is unsuccessful
	CDDRFSO_BUSY	Indicate the operation is already in busy state
Description:	Represents return type of CDD RFSO.	

19.5.2.9 CddRfso_IntervalCbkFuncPtrType

Table 19-10 CddRfso_IntervalCbkFuncPtrType

Name:	CddRfso_IntervalCbkFuncPtrType
Type:	CddRfso_IntervalCbkFuncPtrType
Range:	-
Description:	Interval Callback function pointer.

19.5.3 Function Definitions

Table 19-11 APIs provided by the RFSO Complex Driver Component

Sl.No	**API's name**	**Description**
AUTOSAR API		
1	CddRfso_Init	This function initializes the module.
2	CddRfso_GetVersionInfo	This function provides the version information of this Driver Component.
RENESAS API		
1	CddRfso_ChannelClockSet	This API set frequency clock division for timer of selected RFSO channel.

2	CddRfso_IntervalTimeConfigure	This API sets interval cycle based on unit is time for interval timer and set behavior of Interval Timer one time running or free running.
3	CddRfso_IntervalCycleConfigure	This API sets interval cycle based on unit is cycle for interval timer and set behavior of Interval Timer one time running or free running.
4	CddRfso_StartIntervalTimer	This API starts Interval Timer of selected RFSO channel and also stop Time-out Timer if Time-out Timer is running.
5	CddRfso_StopIntervalTimer	This API stops Interval Timer of selected RFSO channel and also stop Time-out Timer if Time-out Timer is running.
6	CddRfso_TimeoutTimeConfigure	This API is used to configure time-out timer of selected channel with default unit is time (microsecond).
7	CddRfso_TimeoutCycleConfigure	This API is used to configure Time-out Timer of selected RFSO channel with default unit is cycle.
8	CddRfso_StartTimeoutTimer	This API starts Time-out Timer of selected RFSO channel.
9	CddRfso_StopTimeoutTimer	This API stops Time-out timer of selected RFSO channel.
10	CddRfso_ClearTimeoutInterrupt	This API clears time-out interrupt flag of selected RFSO channel.
11	CddRfso_GetTOESPinStatus	This API gets status of TOES bit, TOI bit, TOCUNF bit of selected RFSO channel.
12	CddRfso_GetCFEPinStatus	This API gets status of FSO_CFE0, FSO_CFE1, CFEO0 and CFEO1 bits of selected RFSO channel.
13	CddRfso_ExternalPinControl	This API supports to control external output of RFSO
14	CddRfso_GetTimeoutTimerValue	This API to support read the current counter value of timeout detection timer of selected RFSO channel
15	CddRfso_GetIntervalTimerValue	This API to support read the current counter value of interval timer of selected RFSO channel
16	CddRfso_CtrlIntervalTimerInterrupt	This API to support enable/disable interval timer interrupt of selected RFSO channel
17	CddRfso_IntervalTimerInterruptStatus	This API to support read the status flag of the interrupt interval timer of selected RFSO channel

19.5.3.1 CddRfso_ChannelClockSet

Table 19-12 CddRfso_ChannelClockSet

Function Name:	CddRfso_ChannelClockSet
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_ChannelClockSet (uint8 ChannelId,

	uint32 FrequencyDiv,)		
Service ID:	0x02		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
	uint32	FrequencyDiv	Clock frequency division: 0x00000001.. 0xFFFFFFFF
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Set clock successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API set frequency clock division for timer of selected RFSO channel.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.2 CddRfso_IntervalTimeConfigure

Table 19-13 CddRfso_IntervalTimeConfigure

Function Name:	CddRfso_IntervalTimeConfigure
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_IntervalTimeConfigure (

	uint8 ChannelId, uint64 IntervalTime, boolean OneShot)		
Service ID:	0x03		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
	uint64	IntervalTime	Periodical interval cycle for Interval timer, with unit is Microsecond: 0x00000001.. (0.0075 * FrequencyDivision) *0xFFFFFFFF **Note:** (FrequencyDivision = [0x00000001.. 0xFFFFFFFF])
	boolean	OneShot	Interval timer run in one time: CDDRFSO_TRUE CDDRFSO_FALSE
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Configure Interval timer successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy

Description:	This API sets interval cycle based on unit is time for interval timer and set behavior of Interval Timer one time running or free running.
Configuration Dependency:	None
Preconditions:	Driver must already be initialized.

19.5.3.3 CddRfso_IntervalCycleConfigure

Table 19-14 CddRfso_IntervalCycleConfigure

Function Name:	CddRfso_IntervalCycleConfigure		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_IntervalCycleConfigure (uint8 ChannelId, uint64 IntervalCycle, boolean OneShot)		
Service ID:	0x04		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
	uint64	IntervalCycle	Interval cycle for interval timer with unit is cycle: 0x00000001.. 0xFFFFFFFF
	boolean	OneShot	Interval timer run in one time: CDDRFSO_TRUE CDDRFSO_FALSE
Parameters (In-Out):	None	-	-

Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Configure Interval timer successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API sets interval cycle based on unit is cycle for interval timer and set behavior of Interval Timer one time running or free running.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.4 CddRfso_StartIntervalTimer

Table 19-15 CddRfso_StartIntervalTimer

Function Name:	CddRfso_StartIntervalTimer		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_StartIntervalTimer (uint8 ChannelId)		
Service ID:	0x05		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant for the same ChannelId, reentrant for different ChannelId		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10

Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Interval timer starts successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API starts Interval Timer of selected RFSO channel and also stop Time-out Timer if Time-out Timer is running.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.5 CddRfso_StopIntervalTimer

Table 19-16 CddRfso_StopIntervalTimer

Function Name:	CddRfso_StopIntervalTimer		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_StopIntervalTimer (uint8 ChannelId)		
Service ID:	0x06		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant for the same ChannelId, reentrant for different ChannelId		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10

Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Interval timer stops successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API stops Interval Timer of selected RFSO channel and also stop Time-out Timer if Time-out Timer is running.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.6 CddRfso_TimeoutTimeConfigure

Table 19-17 CddRfso_TimeoutTimeConfigure

Function Name:	CddRfso_TimeoutTimeConfigure		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_TimeoutTimeConfigure (uint8 ChannelId, uint64 TimeoutMaxTime, uint64 TimeoutMinTime)		
Service ID:	0x07		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration:

			0..10
	uint64	TimeoutMaxTime	Maximum time-out cycle with unit is microsecond: 0x00000001.. (0.0075 * FrequencyDivision) *0x100000000 **Note:** (FrequencyDivision = [0x00000001.. 0xFFFFFFFF])
	uint64	TimeoutMinTime	0x00000000.. (0.0075 * FrequencyDivision) *0xFFFFFFFF **Note:** (FrequencyDivision = [0x00000001.. 0xFFFFFFFF])
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Configure Time-out timer successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API configures Time-out Timer of selected RFSO channel, with default unit is time (microsecond).		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.7 CddRfso_TimeoutCycleConfigure

Table 19-18 CddRfso_TimeoutCycleConfigure

Function Name:	CddRfso_TimeoutCycleConfigure
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_TimeoutCycleConfigure

	(uint8 ChannelId, uint64 TimeoutMaxCycle, uint64 TimeoutMinCycle)		
Service ID:	0x08		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
	uint64	TimeoutMaxCycle	Maximum time-out cycle with unit is cycle: 0x00000001.. 0x100000000
	uint64	TimeoutMinCycle	Minimum time-out cycle with unit is cycle: 0x00000000.. 0xFFFFFFFF
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Configure Time-out timer successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API is used to configure Time-out Timer of selected RFSO channel with default unit is cycle.		
Configuration Dependency:	None		

Preconditions:	Driver must already be initialized.

19.5.3.8 CddRfso_StartTimeoutTimer

Table 19-19 CddRfso_StartTimeoutTimer

Function Name:	CddRfso_StartTimeoutTimer		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_StartTimeoutTimer (uint8 ChannelId)		
Service ID:	0x09		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant for the same ChannelId, reentrant for different ChannelId		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Time-out timer starts successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API starts Time-out Timer of selected RFSO channel.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.9 CddRfso_StopTimeoutTimer

Table 19-20 CddRfso_StopTimeoutTimer

Function Name:	CddRfso_StopTimeoutTimer		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_StopTimeoutTimer (uint8 ChannelId)		
Service ID:	0x0A		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant for the same ChannelId, reentrant for different ChannelId		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Time-out timer stops successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API stops Time-out timer of selected RFSO channel.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.10 CddRfso_ClearTimeoutInterrupt

Table 19-21 CddRfso_ClearTimeoutInterrupt

Function Name:	CddRfso_ClearTimeoutInterrupt		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_ClearTimeoutInterrupt (uint8 ChannelId)		
Service ID:	0x0B		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Clear Time-out interrupt successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API clears time-out interrupt flag of selected RFSO channel.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.11 CddRfso_GetTOESPinStatus

Table 19-22 CddRfso_GetTOESPinStatus

Function Name:	CddRfso_GetTOESPinStatus

Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_GetToesPinStatus (uint8 ChannelId, P2VAR(CddRfso_TOESStatusType, AUTOMATIC, CDDRFSO_APPL_DATA) ToesPinStatusPtr)		
Service ID:	0x0C		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
Parameters (In-Out):	P2VAR(CddRfso_ToesStatus, AUTOMATIC, CDDRFSO_APPL_DATA)	ToesPinStatusPtr	Address of memory region to store TOES bit, TOI bit, TOCUNF bit status
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Bit status report successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
Description:	This API gets status of TOES bit, TOI bit, TOCUNF bit of selected RFSO channel.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.12 CddRfso_GetCFEPinStatus

Table 19-23 CddRfso_GetCFEPinStatus

Function Name:	CddRfso_GetCFEPinStatus		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_GetCFEPinStatus (uint8 ChannelId, P2VAR(CddRfso_CFEStatusType, AUTOMATIC, CDDRFSO_APPL_DATA) CfePinStatusPtr)		
Service ID:	0x0D		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	Channel Id configuration: 0..10
Parameters (In-Out):	P2VAR(CddRfso_CFEStatusType, AUTOMATIC, CDDRFSO_APPL_DATA)	CfePinStatusPtr	Address of memory region to store FSO_CFE0/FSO_CFE1 pin and CFEO0/CFEO1 status.
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	External pin report successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
Description:	This API gets status of FSO_CFE0, FSO_CFE1, CFEO0 and CFEO1 bits of selected RFSO channel.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.13 CddRfso_ExternalPinControl

Table 19-24 CddRfso_ExternalPinControl

Function Name:	CddRfso_ExternalPinControl		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_ExternalPinControl (uint8 Channel, boolean EnableCfe0, boolean EnableCfe1)		
Service ID:	0x0E		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	Channel	channel of RFSO: 0..10
	boolean	EnableCfe0	Set FSO_CFE0 bit: CDDRFSO_TRUE CDDRFSO_FALSE
	boolean	EnableCfe1	Set FSO_CFE1 bit: CDDRFSO_TRUE CDDRFSO_FALSE
Parameters (In-Out):	None	-	-
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	External Pin control successfully

		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
Description:	This API supports to control CFEO0 and CFEO1 bit.		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized		

19.5.3.14 CddRfso_GetTimeoutTimerValue

Table 19-25 CddRfso_GetTimeoutTimerValue

Function Name:	CddRfso_GetTimeoutTimerValue		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_GetTimeoutTimerValue (uint8 ChannelId, P2VAR(uint32, AUTOMATIC, CDDRFSO_APPL_DATA) TimeoutValuePtr)		
Service ID:	0x0F		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	channel of RFSO: 0..10
Parameters (In-Out):	P2VAR (uint32, AUTOMATIC, CDDRFSO_APPL_DATA)	TimeoutValuePtr	Address of memory region to store current counter value of timeout detection timer
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Counter value report successfully

		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
Description:	This API to support read the current counter value of timeout detection timer		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.15 CddRfso_GetIntervalTimerValue

Table 19-26 CddRfso_GetIntervalTimerValue

Function Name:	CddRfso_GetIntervalTimerValue		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_GetIntervalTimerValue (uint8 ChannelId, P2VAR(uint32, AUTOMATIC, CDDRFSO_APPL_DATA) IntervalTimerValuePtr)		
Service ID:	0x10		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	channel of RFSO: 0..10
Parameters (In-Out):	P2VAR (uint32, AUTOMATIC, CDDRFSO_APPL_DATA)	IntervalTimerValuePtr	Address of memory region to store current counter value of interval timer
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	

	CddRfso_ReturnType	CDDRFSO_E_OK	Counter value report successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
Description:	This API to support read the current counter value of interval timer		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.3.16 CddRfso_CtrlIntervalTimerInterrupt

Table 19-27 CddRfso_CtrlIntervalTimerInterrupt

Function Name:	CddRfso_CtrlIntervalTimerInterrupt		
Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_CtrlIntervalTimerInterrupt (uint8 ChannelId, boolean IntervalTimerIrqState)		
Service ID:	0x11		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant for the same ChannelId, reentrant for the same ChannelId		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	channel of RFSO: 0..10

	boolean	IntervalTimerIrqState	Enable/disable interval timer interrupt: CDDRFSO_TRUE CDDRFSO_FALSE
Parameters (In-Out):	None	None	None
Parameters (Out):	None	None	None
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Control interval timer interrupt successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
		CDDRFSO_BUSY	Channel is busy
Description:	This API to support enable/disable interval timer interrupt.		
Configuration Dependency:	None		
Preconditions:	- Driver must already be initialized. - Interval timer should not be running when enable/disable interval timer interrupt. - Input parameter IntervalTimerIrqState shall be different from the current state of interval timer interrupt.		

19.5.3.17 CddRfso_IntervalTimerInterruptStatus

Table 19-28 CddRfso_IntervalTimerInterruptStatus

Function Name:	CddRfso_IntervalTimerInterruptStatus

Syntax:	FUNC(CddRfso_ReturnType, CDDRFSO_CODE_SLOW) CddRfso_IntervalTimerInterruptStatus (uint8 ChannelId, P2VAR(uint8, AUTOMATIC, CDDRFSO_APPL_DATA) IntervalTimerIrqSttPtr)		
Service ID:	0x12		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	**Type**	**Parameter**	**Value/Range**
	uint8	ChannelId	channel of RFSO: 0..10
Parameters (In-Out):	P2VAR (uint8, AUTOMATIC, CDDRFSO_APPL_DATA)	IntervalTimerIrqSttPtr	Address of memory region to store status flag of the interrupt of interval timer
Parameters (Out):	None	-	-
Return Value:	**Type**	**Possible Return values**	
	CddRfso_ReturnType	CDDRFSO_E_OK	Interval Timer interrupt status report successfully
		CDDRFSO_E_NOT_OK	A development error occurred (see DET/DEM Errors handled)
Description:	This API to support read the status flags of the interrupt of interval timer		
Configuration Dependency:	None		
Preconditions:	Driver must already be initialized.		

19.5.4 Preemption of APIs

The table 1.32 specifies the preemption of each API that can be invoked at the same time with the API.

Table 19-29 Preemption Table of APIs of the RFSO Driver

	CddRfso_Init	CddRfso_GetVersionInfo	CddRfso_ChannelClockSet	CddRfso_IntervalTimeConfigure	CddRfso_IntervalCycleConfigure	CddRfso_StartIntervalTimer	CddRfso_StopIntervalTimer	CddRfso_TimeoutTimeConfigure	CddRfso_TimeoutCycleConfigure	CddRfso_StartTimeoutTimer	CddRfso_StopTimeoutTimer	CddRfso_ClearTimeoutInterrupt	CddRfso_GetTOESPinStatus	CddRfso_GetCFEPinStatus	CddRfso_ExternalPinControl	CddRfso_GetTimeoutTimerValue	CddRfso_GetIntervalTimerValue	CddRfso_CtrlIntervalTimerInterrupt	CddRfso_IntervalTimerInterruptStatus
CddRfso_Init	-																		
CddRfso_GetVersionInfo	√	√																	
CddRfso_ChannelClockSet	-	√	-																
CddRfso_IntervalTimeConfigure	-	√	-	-															
CddRfso_IntervalCycleConfigure	-	√	-	-	-														
CddRfso_StartIntervalTimer	-	√	-	-	-	√													
CddRfso_StopIntervalTimer	-	√	-	-	-	√	√												
CddRfso_TimeoutTimeConfigure	-	√	-	-	-	-	-	-											
CddRfso_TimeoutCycleConfigure	-	√	-	-	-	-	-	-	-										
CddRfso_StartTimeoutTimer	-	√	-	-	-	√	√	-	-	√									
CddRfso_StopTimeoutTimer	-	√	-	-	-	√	√	-	-	√	√								
CddRfso_ClearTimeoutInterrupt	-	√	-	-	-	√	√	-	-	√	√	√							
CddRfso_GetTOESPinStatus	-	√	-	-	-	√	√	-	-	√	√	√	√						
CddRfso_GetCFEPinStatus	-	√	-	-	-	√	√	-	-	√	√	√	√	√					
CddRfso_ExternalPinControl	-	√	-	-	-	√	√	-	-	√	√	√	√	√	√				
CddRfso_GetTimeoutTimerValue	-	√	-	-	-	√	√	-	-	√	√	√	√	√	√	√			
CddRfso_GetIntervalTimerValue	-	√	-	-	-	√	√	-	-	√	√	√	√	√	√	√	√		
CddRfso_CtrlIntervalTimerInterrupt	-	√	-	-	-	√	√	-	-	√	√	√	√	√	√	√	√	√	
CddRfso_IntervalTimerInterruptStatus	-	√	-	-	-	√	√	-	-	√	√	√	√	√	√	√	√	√	√

-: cannot be invoked at the same time

√: can be invoked at the same time

19.5.5 Additional Error Handling and Restriction

Following is additional error handling and API specific limitations:

- After calling CddRfso_ExternalPinControl, there should be a delay time before invoking CddRfso_GetCFEPinStatus for getting status of FSO_CFE0 and FSO_CFE1 bits. Otherwise, the status return from CddRfso_GetCFEPinStatus may not be correct.
- After calling CddRfso_IntervalTimeConfigure or CddRfso_TimeoutCycleConfigure, user should not call CddRfso_StartTimeoutTimer immediately. Otherwise, the running time of timeout timer will be not guaranteed.
- After calling CddRfso_TimeoutTimeConfigure or CddRfso_TimeoutCycleConfigure, use should not call immediately CddRfso_StartTimeoutTimer. OTherwise, the operation of Timeout Timer will be not guaranteed.
- Due to the rounding operation of integer number, the input interval time and timeout time of CddRfso_IntervalTimeConfigure and CddRfso_TimeoutTimeConfigure should be used with large channel clock division (approximately above 100 for clock division) and reasonable input time to avoid the time (cycle) error when converting from microsecond unit to cycle unit. Otherwise, the timing of timer will be not guaranteed.
- CddRfso_StartIntervalTimer and CddRfso_StopIntervalTimer API as described in *19.5.3.5* will stop Timeout Timer. Therefore, user should not call CddRfso_StartIntervalTimer and CddRfso_StopIntervalTimer after timeout timer has already been running except with intended purpose. Otherwise, the operation of Timeout Timer will not be guaranteed.
- When ISR of interval timer is invoked, interval ISR will also stop Timeout if Time-out Timer is running in the same RFSO channel.
- CddRfso_ChannelClockSet, CddRfso_IntervalTimeConfigure, CddRfso_IntervalCycleConfigure, CddRfso_StartIntervalTimer, CddRfso_TimeoutTimeConfigure, CddRfso_StartTimeoutTimer, CddRfso_StopTimeoutTimer, CddRfso_ClearTimeoutInterrupt, CddRfso_CtrlIntervalTimerInterrupt cannot disturb each other and itself while they have not returned value. Otherwise, CDDRFSO_BUSY value will be returned.
- In hardware specification, the interval time between start and first interval interrupt will have error as ±(Timer counter clock interval). Due to this specification, ± 1 cycle error (1 cycle = 7.5ns x CddRfsoFrequencyDivision configuration) will be occurred for the duration after called CddRfso_StartIntervalTimer API and first interval interrupt. This means One-shot mode always have this error.
- In hardware specification, the time-out time between start and first time-out interrupt will have error as ±(Timer counter clock interval). Due to this specification, ± 1 cycle error (1 cycle = 7.5ns x CddRfsoFrequencyDivision configuration) will be occurred for the duration after called CddRfso_StartTimeoutTimer API and first time-out interrupt.

19.6 Development and Production Errors

This chapter describes all the development errors (see AUTOSAR Specification of Default Error Tracer) and production error code (see AUTOSAR Specification of Diagnostic Event Manager) that are reported by the RFSO Complex Driver Component.

19.6.1 RFSO Complex Device Driver Component Development Errors

The following table contains the DET errors that are reported by RFSO Complex Driver Component. These errors are reported to Development Error Tracer Module when the RFSO Complex Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 19-30 DET Errors of RFSO Complex Driver Component

Sl. No.	**1**
Error Code	CDDRFSO_E_UNINIT
Related API(s)	CddRfso_ChannelClockSet CddRfso_IntervalTimeConfigure CddRfso_IntervalCycleConfigure CddRfso_StartIntervalTimer CddRfso_StopIntervalTimer CddRfso_TimeoutTimeConfigure CddRfso_TimeoutCycleConfigure CddRfso_StartTimeoutTimer CddRfso_StopTimeoutTimer CddRfso_ClearTimeoutInterrupt CddRfso_GetTOESPinStatus CddRfso_GetCFEPinStatus CddRfso_ExternalPinControl CddRfso_GetTimeoutTimerValue CddRfso_GetIntervalTimerValue CddRfso_CtrlIntervalTimerInterrupt CddRfso_IntervalTimerInterruptStatus
Source of Error	When the APIs are invoked without the initialization of RFSO Complex Driver
Origin	Renesas
Sl. No.	**2**
Error Code	CDDRFSO_E_PARAM_VALUE

Related API(s)	CddRfso_ChannelClockSet CddRfso_IntervalTimeConfigure CddRfso_IntervalCycleConfigure CddRfso_StartIntervalTimer CddRfso_StopIntervalTimer CddRfso_TimeoutTimeConfigure CddRfso_TimeoutCycleConfigure CddRfso_StartTimeoutTimer CddRfso_StopTimeoutTimer CddRfso_ClearTimeoutInterrupt CddRfso_GetTOESPinStatus CddRfso_GetCFEPinStatus CddRfso_ExternalPinControl CddRfso_GetTimeoutTimerValue CddRfso_GetIntervalTimerValue CddRfso_CtrlIntervalTimerInterrupt CddRfso_IntervalTimerInterruptStatus
Source of Error	When the API is invoked with the invalid input values.
Origin	Renesas
Sl. No.	**3**
Error Code	CDDRFSO_E_ALREADY_INITIALIZED
Related API(s)	CddRfso_Init
Source of Error	When the API CddRfso_Init is invoked twice.
Origin	Renesas
Sl. No.	**4**
Error Code	CDDRFSO_E_PARAM_POINTER
Related API(s)	CddRfso_Init CddRfso_GetTOESPinStatus CddRfso_GetCFEPinStatus CddRfso_GetTimeoutTimerValue CddRfso_GetIntervalTimerValue CddRfso_IntervalTimerInterruptStatus CddRfso_GetVersionInfo
Source of Error	When API invoked with message pointer is NULL_PTR.
Origin	Renesas
Sl. No.	**5**
Error Code	CDDRFSO_E_INVALID_DATABASE
Related API(s)	CddRfso_Init
Source of Error	When the API service is invoked with invalid database address.
Origin	Renesas
Sl. No.	**6**
Error Code	CDDRFSO_E_TIMER_MODE

Related API(s)	CddRfso_ChannelClockSet CddRfso_IntervalTimeConfigure CddRfso_IntervalCycleConfigure CddRfso_StartIntervalTimer CddRfso_StopIntervalTimer CddRfso_TimeoutTimeConfigure CddRfso_TimeoutCycleConfigure CddRfso_StartTimeoutTimer CddRfso_StopTimeoutTimer CddRfso_ClearTimeoutInterrupt CddRfso_CtrlIntervalTimerInterrupt
Source of Error	When the API service is invoked in invalid timer mode.
Origin	Renesas

19.6.2 RFSO Complex Device Driver Component Production Errors

The following table contains Renesas specific DEM errors that are reported by RFSO Complex Driver Component.

Table 19-31 DEM Errors of RFSO Device Driver Component

Sl. No.	**1**
Error Code	CDDRFSO_E_WRITE_VERIFY
Related API(s)	CddRfso_Init, CddRfso_ChannelClockSet, CddRfso_IntervalTimeConfigure, CddRfso_IntervalCycleConfigure, CddRfso_TimeoutTimeConfigure, CddRfso_TimeoutCycleConfigure, CddRfso_ExternalPinControl.
Source of Error	Monitors register write failure. • DEM_EVENT_STATUS_FAILED: When register read-back value does not match with expected value.
Origin	Renesas
Sl. No.	**2**
Error Code	CDDRFSO_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	CDDRFSO_CHANNELn_ISR / ISR (CDDRFSO_CHANNELn_CAT2_ISR) (n=0..10)
Source of Error	When the interrupt is issued via INTC-AP or GIC in CPU (Arm Realtime Core), but the related interrupt control and status registers of RFSO are not raised before processing the software operation for interrupt handling.
Origin	Renesas

19.7 RFSO Driver Component Runtime Errors

AUTOSAR does not define any runtime error code for RFSO Driver.

19.8 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of RFSO Complex Driver Component software.

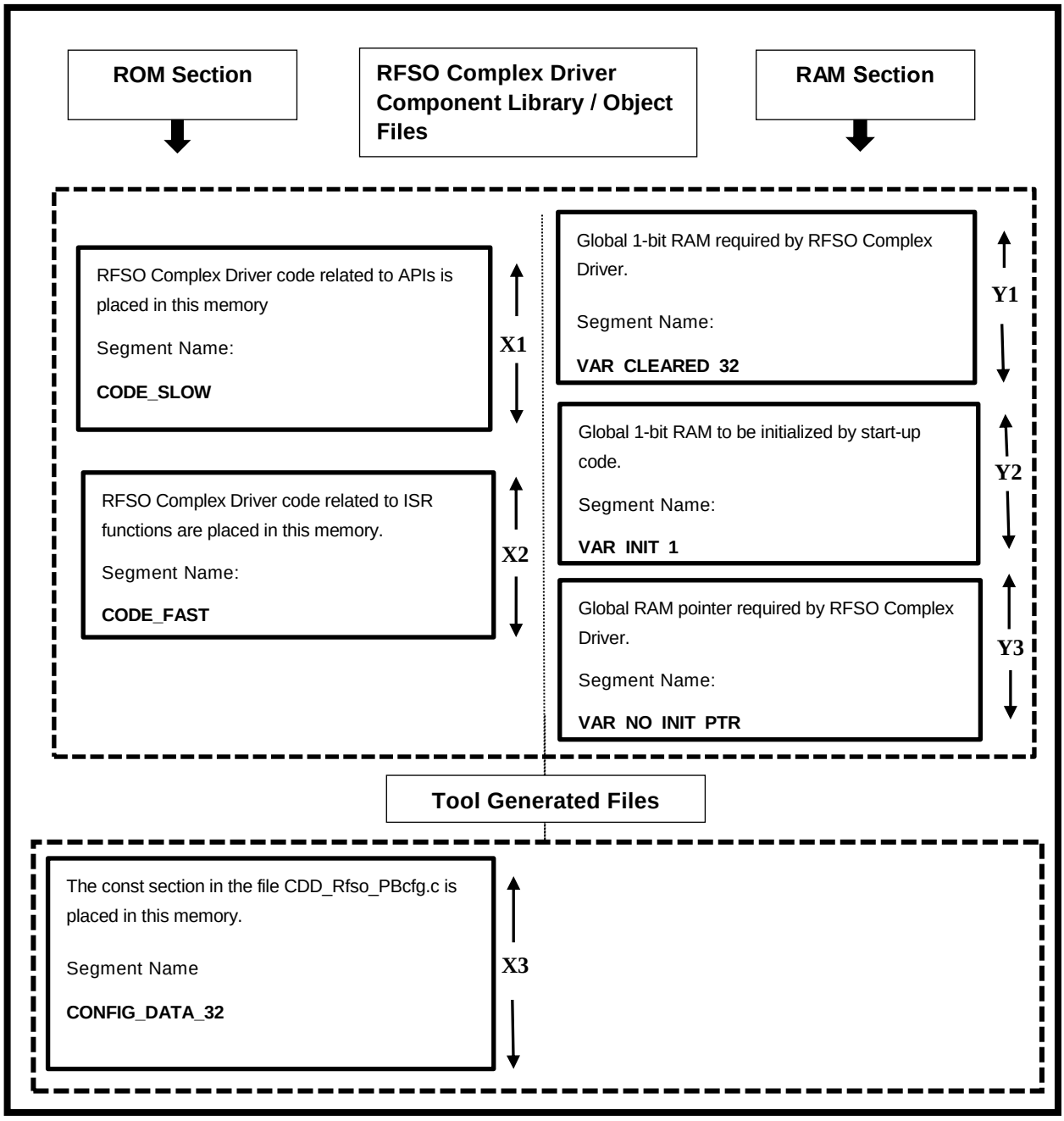


Figure 19.4 RFSO Complex Driver Component Memory Organization

ROM Section (X1, X2 and X3):

CODE_SLOW (X1): API(s) of RFSO Complex Device Driver Component except for ISR functions.

CODE_FAST (X2): This section consists of MFIS ISR functions that can be located in code memory.

CONFIG_DATA_32 (X3): This section consists of RFSO Complex Driver Component controller specific configuration constant structures generated by RFSO Complex Driver Component Code Generation Tool. This can be located in the code memory.

Note**:** ROM in R-Car is the Read-only Section in memory (DRAM).

RAM Section (Y1, Y2 and Y3):

VAR_CLEARED_32 (Y1): This section consists of the global RAM variables of 32-bit size that are used internally by RFSO Complex Driver Component. This can be located in data memory.

VAR_INIT_1 (Y2): This section consists of the global RAM variables of 1-bit size that are used internally by RFSO Complex Driver Component. This can be located in data memory.

VAR_NO_INIT_PTR (Y3): This section consists of the global RAM pointer variables used internally by RFSO Complex Driver Component. It can be located in data memory.

Note**:** User must ensure that none of the memory areas overlap with each other. Even 'debug' information should not overlap.

19.9 Device Specific Information

The device supports the following devices:
Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview" – section 5.1* Product.

19.9.1 Interaction between the User and RFSO Complex Device Driver Component

The detail of the services supported by the RFSO Complex Driver Component to the upper layers users is provided in the following chapter:

19.9.1.1 Channel Mapping

Table 19-32 Hardware CDDRFSO channel mapping

Channel	Hardware channel
0	RFSO 0
1	RFSO 1
2	RFSO 2
3	RFSO 3

4	RFSO 4
5	RFSO 5
6	RFSO 6
7	RFSO 7
8	RFSO 8
9	RFSO 9
10	RFSO 10

19.9.1.2 ISR Functions

The table below provides the list of handlers corresponding to the hardware unit ISR(s) in RFSO Complex Device Driver Component. The user should configure the ISR functions mentioned below:

Table 19-33 Interrupt Vector Table

Name of interrupt	Interrupt Source	Name of the ISR Function
RFSO_ch0	SPI227	CDDRFSO_ CHANNEL0 _CAT2_ISR
		CDDRFSO_ CHANNEL0 _ISR
RFSO_ch1	SPI228	CDDRFSO_ CHANNEL1 _CAT2_ISR
		CDDRFSO_ CHANNEL1 _ISR
RFSO_ch2	SPI229	CDDRFSO_ CHANNEL2 _CAT2_ISR
		CDDRFSO_ CHANNEL2 _ISR
RFSO_ch3	SPI230	CDDRFSO_ CHANNEL3 _CAT2_ISR
		CDDRFSO_ CHANNEL3 _ISR
RFSO_ch4	SPI231	CDDRFSO_ CHANNEL4 _CAT2_ISR
		CDDRFSO_ CHANNEL4 _ISR
RFSO_ch5	SPI232	CDDRFSO_ CHANNEL5 _CAT2_ISR
		CDDRFSO_ CHANNEL5 _ISR
RFSO_ch6	SPI233	CDDRFSO_ CHANNEL6 _CAT2_ISR

Name of interrupt	Interrupt Source	Name of the ISR Function
		CDDRFSO_ CHANNEL6 _ISR
RFSO_ch7	SP234	CDDRFSO_ CHANNEL7 _CAT2_ISR
		CDDRFSO_ CHANNEL7 _ISR
RFSO_ch8	SPI235	CDDRFSO_ CHANNEL8 _CAT2_ISR
		CDDRFSO_ CHANNEL8 _ISR
RFSO_ch9	SPI236	CDDRFSO_ CHANNEL9 _CAT2_ISR
		CDDRFSO_ CHANNEL9 _ISR
RFSO_ch10	SPI237	CDDRFSO_ CHANNEL10 _CAT2_ISR
		CDDRFSO_ CHANNEL10 _ISR

19.9.2 Multi-Core / Multi-Instantiation

RFSO driver does not support multi-core and multi-instantiation for this device.

19.10 Non-AUTOSAR environment integration

The RFSO Complex Driver Components for Renesas R-Car Gen4 SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the RFSO Complex Driver Components to Non-AUTOSAR environment.

19.10.1Stub modules handling

19.10.1.1 Det

The Det stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, RFSO Complex Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g. RFSO Complex Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:
Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)
Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

19.10.1.2 Dem

The Dem stub files are organized in the following folder:
\rel\common\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, RFSO Complex Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error. The API prototype is as of follow:
Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus).

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

19.10.1.3 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.
The SchM (Basic Software Scheduler) stub files are organized in the following folder:
rel\common\generic\stubs\<Autosar version>\Rte

The RFSO Complex Driver Component needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when CddRfsoCriticalSectionProtection parameter is configured as

true. With Non-AUTOSAR environment, user needs to prepare SchM stub to user RFSO Complex Driver Component as following:
Write SchM functions with prototype as below:
void SchM_Enter_CddRfso_CDDRFSO_RAM_DATA_PROTECTION (void);
void SchM_Exit_CddRfso_CDDRFSO_RAM_DATA_PROTECTION (void);
void SchM_Enter_CddRfso_CDDRFSO_INTERRUPT_CONTROL_PROTECTION (void);
void SchM_Exit_CddRfso_CDDRFSO_INTERRUPT_CONTROL_PROTECTION (void);

19.10.2Callback function usage

The RFSO Complex Driver Component has a callback function being used to notify that there is an Interval Timer interrupt.

The callback function is configured for each channel by parameter “CddRfsoIntervalTimerCallbackFunction” in each channel.

19.10.3Scheduled function usage

The RFSO Complex Driver Component does not have Scheduled function.

19.10.4Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:
\rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

Non-AUTOSAR users shall use CDDRFSO_CHANNEL<Channel Number>_ISR ISR as specified in *section 19.9.1.2 ISR Functions* .

20.CRC

20.1 Overview

The purpose of this document is to describe the information related to CRC Complex Driver Component for Renesas RCar microcontrollers.

User of CRC Complex Driver Component shall use this document as reference. This document describes common features of CRC Complex Driver Component.

This document is intended for the developers of ECU software using Application Programming Interfaces provided by AUTOSAR. The CRC Complex Driver Component provides the following services:

- CRC Complex Driver Component initialization
- Create CRC from input data for 9 polynomials
- Check a data packet for a CRC error
- Get version information
- Select channel mode
- Deliver data packet to CRC, KCRC module via DMA and operate on each module
- Set value for registers of WCRC, CRC, and KCRC by command function
- Stop data transfer
- Compare CRC results with expected CRC codes
- Unintended module stop check

The following diagram shows the system overview of the AUTOSAR Architecture.

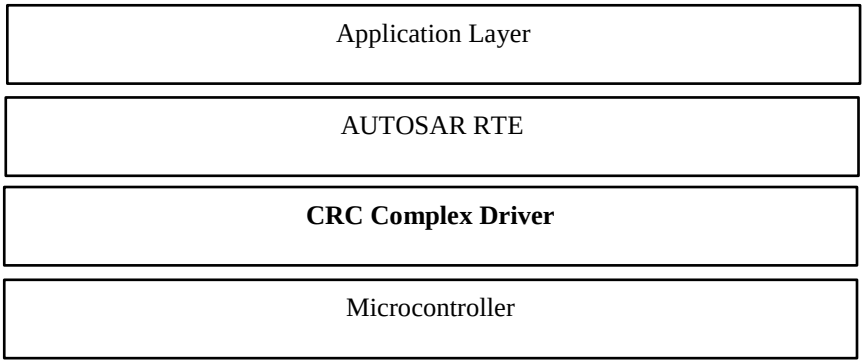


Figure 20-1 System Overview of AUTOSAR Architecture

The CRC Complex Driver Component comprises of two sections that is, embedded software and the configuration tool to achieve scalability and configurability.

The CRC Complex Driver Component Code Generation Tool is a command line tool that accepts ECU configuration description files as input and generates C Header files. The configuration description is an ARXML file that contains information about the configuration for CRC channels. The tool generates CDD_Crc_Cfg.h, CDD_Crc_PBcfg.c, CDD_Crc_Cbk.h file.

The following depicts the CRC Complex Driver as part of AUTOSAR Software Layers.

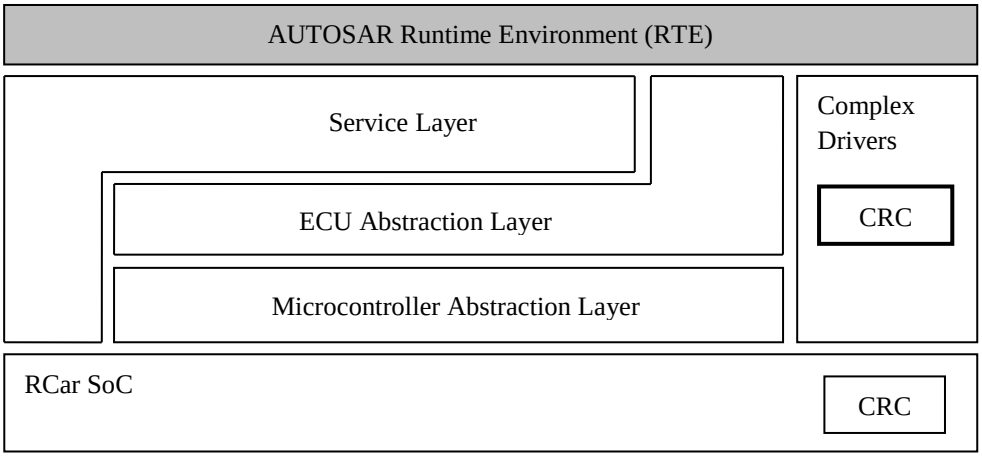


Figure 20-2 System Overview of the CRC Complex Driver in AUTOSAR Software Layer

20.2 Forethoughts

20.2.1 General

Following information will aid the user to use the CRC Complex Driver Component software efficiently:

- CRC result from CDD CRC is created based on R-Car Series. This CRC result complies with AutoSar Specification for CRC code.
- Example code mentioned in this document shall be taken only as a reference for implementation.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- If any development errors are reported to DET, the normal flow of execution of driver will be aborted.
- All production errors will be reported to DEM by using the API Dem_SetEventStatus provided by DEM.
- CRC Driver supports to set mode for each channel independently.
  These below modes are currently supported

  - Independent mode
  - E2E mode
  - Data through mode
  - E2E and data through mode
  - AES-ACC mode (*)

Note: (*) To use AES-ACC mode, user needs to enable AES-ACC following information from HWUM which is only disclosed to authorized customers.

- The CRC Driver does not handle DMA errors on the RT-DMAC hardware unit. If any DMA error occurs during processing, the channel may stall.
  To recover from the busy state, users shall invoke the API CddCrc_Stop to stop the channel, and then invoke CddCrc_SetMode before the next writing or command.

- The argument “LenCompFreq” of CddCrc_Compare shall be the number of expected CRC codes which shall be compared
  - FREQ_16_BYTE: 4 4-byte CRC codes shall be compared
  - FREQ_32_BYTE: 8 4-byte CRC codes shall be compared
  - FREQ_64_BYTE: 16 4-byte CRC codes shall be compared
- CRC Complex Driver supports one instance, therefore, configuration parameter CddInstanceId should be configured as 0. In the implementation, instance ID is always used as 0 without using CddInstanceId parameter in CDF setting.

20.2.2 Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the CRC Complex Driver Component:

- The CDD_Crc_Cfg.h, CDD_Crc_PBcfg.c file generated by the CRC Complex Driver Component Code Generation Tool must be compiled and linked along with CRC Complex Driver Component source files.

- The application has to be rebuilt, if there is any change in the CDD_Crc_Cfg.h, CDD_Crc_PBcfg.c, CDD_Crc_Cbk.h files generated by the CRC Complex Driver Component Generation Tool.
- The CRC Complex Driver Component needs to be initialized before accepting any request. The API CddCrc_Init should be invoked to initialize CRC Complex Driver Component before calling any other CRC Complex Driver API.
- The user should ensure that CRC Complex Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated only when the static configuration parameter CddCrcDevErrorDetect is enabled. Application should ensure that the right parameters are passed while invoking the APIs when CddCrcDevErrorDetect is disabled.
- The CRC Complex Driver states are checked only when the static configuration parameter CddCrcDevErrorDetect is enabled. Application should ensure the recommended sequence of invoking APIs is satisfied when CddCrcDevErrorDetect is disabled.
- CRC Driver provides Register Write-Verify feature to verify the execution of control registers' write operation. After writing to a control register, value of that register is read back to make sure that register has been written correctly.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used.
- The data size parameter shall be matched with the range of input data to ensure the correctness of creating/checking CRC.
- CRC Driver supports to calculate CRC code for only 8-bit data width.
- The pointer passed to the API CddCrc_SetupEB shall not be NULL and shall be allocated into non-cache memory. Otherwise, cache maintenance shall be implemented by users to ensure that CRC code can be read out from result port, or the written data to the data port in FIFO can be read out.
- The pointer passed to the API CddCrc_SetupEB, when reading from RESULT_PORT, shall be prepared enough space for data receiving, the data size required base on conversion size and data length passed to CddCrc_Write. Buffer size should comply with 4*(data length)/(conversion size).
- The pointer passed to the API CddCrc_SetupEB, when reading from DATA_PORT, shall be prepared enough space for data receiving, the data size required bases on data length passed to CddCrc_Write. Buffer size should comply with 4*(data length).
- The pointer passed to the API CddCrc_Write shall be prepared enough space for data transferring, the data size required bases on data length passed to CddCrc_Write. Buffer size should by a multiple of 4*(conversion size) (bytes).
- The CRC Driver provides a memory section for uninitialized data to prepare a non-cache buffer on DRAM. With macros CDDCRC_START_SEC_VAR_USER_RAM, and CDDCRC_STOP_SEC_VAR_USER_RAM, the buffer shall be placed at section VAR_CLEARED_USER_RAM_UNSPECIFIED defined in scatter file. The example of this section can be referred in below files:
  \rel\modules\cddcrc\sample_application\src\App_CDD_CRC_Common_Sample.c

  \rel\V4H\common_family\make\arm\App_Sample.scat

- The argument "LulConvSize" of CddCrc_Write is the number of bytes which shall be calculated for a CRC code, while the argument "LulDataLength" of CddCrc_Write is the total bytes of data block, it shall be broken into smaller block with size of "LulConvSize" bytes. Therefore, the number of CRC code shall be the result of dividing "LulDataLength" by "LulConvSize, and it should be a multiplication of 4 to ensure that the CRC results exist at result port.
- The pointer passed to CddCrc_Write shall be 4-byte aligned.
- The argument "LulDataLength" of CddCrc_Command is the number of 4-byte values, and it shall be an even number.
- The API CddCrc_Command shall not check the value written to registers, users need to make sure that the value does not violate Hardware User Manual.
- User shall not invoke CddCrc_Command with arguments of registers WCRC_XXXX_EN and WCRC_XXXX_CMDEN of the channel itself to change their value at any time. Since WCRC_XXXX_EN and WCRC_XXXX_CMDEN control operations, so the modification of them could lead to malfunctions warned in Hardware User Manual that are prohibition of simultaneous operation of Data through mode, Comparing CRC result, and Register access by command function. In case users modify WCRC_XXXX_EN and WCRC_XXXX_CMDEN intentionally, users must invoke API CddCrc_SetMode with a corresponding mode.

- In mode E2E_PLUS_DATA_THROUGH_MODE, users shall invoke the API CddCrc_SetupEB twice with data port and result port to prepare 2 locations on external memory to data, and CRC code fully read out.
- User must initialize the following hardware IPs as below before initializing CRC module.

Table 20-1 List of HW IP affect to CRC

Hardware IP	Module Name	Expected Setting	Description
CPG	MCU	**CPGWPCR:** Clear bit0 of this register to disable write protection of register in CPG. **MSTPCR9:** Bit 3, 4, 5, 6 of the register is set to 0 to enable supply of the clock signal. **MSTPCR12:** Bit 25, 26, 27, 28, 31 of the register is set to 0 to enable supply of the clock signal. **MSTPCR13:** Bit 0, 1, 2 of the register is set to 0 to enable supply of the clock signal.	This IP enable/disable clock signal of CRC modules. The setting of these register is provided in MCU Driver. For more detail setting, please refer to User Manual of MCU.

20.2.3 Data Consistency

None

20.3 Architecture Details

The CRC Complex Driver architecture is shown in the following figure:

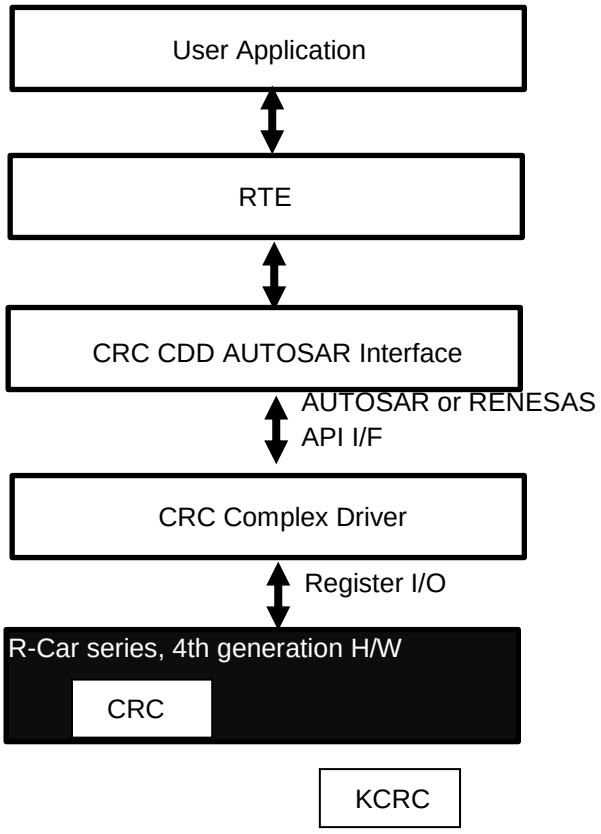


Figure 20-3 CRC Complex Driver Component Architecture

The CRC Complex Driver component can be divided into following sub components based on the functions performed by the CRC Complex Driver:

- Driver initialization
- Creating/Checking CRC with nine polynomials
- Getting version information
- Channel mode selection
- WCRC operations via DMA
- Register access by command function
- Status information
- Stop of data transferring
- Comparison of CRC results and expected CRC codes
- Unintended module stop check

Driver initialization

The driver initialization sub-module internally stores the configuration data address to enable subsequent API calls to access the configuration data and initializes the global variables used by CRC Complex Driver Component.

The API related to this sub-module is CddCrc_Init.

Creating/Checking CRC with nine polynomials

The CRC complex driver provides a service to calculate CRC code from the data input.

The API related to this sub-module is CddCrc_Process.

Getting version information

The CRC Complex Driver provides a service to return version information details to the User.

The API related to this sub-module is CddCrc_GetVersionInfo.

Channel mode selection

The CRC Complex Driver provides a service to select operation mode of each channel.

The API related to this sub-module is CddCrc_SetMode.

WCRC operations via DMA

The CRC Complex Driver provides the services to communicate with each module within WCRC via DMA to operate the data packet in compliance with the corresponding WCRC mode.

The APIs related to this sub-module are CddCrc_Write, CddCrc_SetupEB, CddCrc_Command.

Register access by command function

The CRC Complex Driver provides the service to access registers of WCRC, CRC, and KCRC via DMA.

The API related to this sub-module is CddCrc_Command.

Status information

The CRC Complex Driver provides the services to get status of each channel.

The API related to this sub-module is CddCrc_ReadStatus.

Stop of data transferring

The CRC Complex Driver provides the services to stop each channel.

The API related to this sub-module is CddCrc_Stop.

Comparison of CRC results and expected CRC codes

The CRC Complex Driver provides the services to compare CRC results with expected CRC codes via DMA.

The API related to this sub-module is CddCrc_Compare, CddCrc_ReadCompareResult.

Unintended module stop check

The CRC Complex Driver provides the services for periodically checking of unintended stop and notifying to DEM in case error is detected.

The API related to this sub-module is CddCrc_UnintendedModuleStopCheck.

20.4 CRC Driver Component Header and Source File Description

This chapter explains the CRC Complex Driver Component's source and header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by CRC Complex Driver Generation Tool:

- CDD_Crc_Cfg.h
- CDD_Crc_PBcfg.c
- CDD_Crc_Cbk.h

The CRC Complex Device Driver Component C header files and Component source files:

Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* – section *"3.4.7.3 Folder Structure"*.

The Stub C header files and source file:

Refer to *"R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview"* – section *"3.2.9 Stubs File"*.

20.5 Application Programming Interface

This section explains the Data types and APIs provided by the CRC Complex Driver

20.5.1 Imported Types

This chapter explains the data types imported by the CRC Complex Device Driver Component and lists its dependency on other modules.

20.5.1.1 Standard Types

In this chapter all types included from the Std_Types.h (according to "*AUTOSAR Specification of Standard Types"*) are listed:

- Std_ReturnType.
- Std_VersionInfoType.

20.5.1.2 OS Types

None

20.5.1.3 Dem Types

In this chapter all types included from the Dem.h (according to "*AUTOSAR Specification of Diagnostic Event Manager"*) are listed:

- Dem_EventIdType.
- Dem_EventStatusType.

20.5.2 Type Definitions

This chapter explains the type definitions of CRC Complex Driver Component.

20.5.2.1 CddCrc_ChannelType

Table 20-2 CddCrc_ChannelType

Name:	CddCrc_ChannelType
Type:	uint8
Description:	Represents the identifier of an CRC channel.

20.5.2.2 CddCrc_ChannelSttType

Table 20-3 CddCrc_ChannelSttType

Name:	CddCrc_ChannelSttType
Type:	Enum
Elements:	CDDCRC_CH_OK
	CDDCRC_CH_BUSY
	CDDCRC_CH_STOPPED
Description:	Identify channel status

20.5.2.3 CddCrc_ReturnType

Table 20-4 CddCrc_ReturnType

Name:	CddCrc_ReturnType
Type:	Enum
Elements:	CDDCRC_OK
	CDDCRC_NOT_OK
	CDDCRC_BUSY
Description:	Identify return type of Crc

20.5.2.4 **CddCrc_PolyType**

Table 20-5 CddCrc_PolyType

Name:	CddCrc_PolyType
Type:	Enum
Elements:	CDDCRC_POLY_32_ETH
	CDDCRC_POLY_16_CCITT
	CDDCRC_POLY_8_SAE_J1850
	CDDCRC_POLY_8_0x2F
	CDDCRC_POLY_32_F4ACFB13
	CDDCRC_POLY_32_1EDC6F41
	CDDCRC_POLY_21_102899
	CDDCRC_POLY_17_1685B
	CDDCRC_POLY_15_4599
Description:	Polynomial type

20.5.2.5 **CddCrc_ConfigType**

Table 20-6 CddCrc_ConfigType

Name:	CddCrc_ConfigType	
Type:	Structure	
Elements:	uint32	ulStartOfDbToc
	P2CONST(CddCrc_ChannelConfigType, TYPEDEF, CDDCRC_CONFIG_DATA)	pChannelConfig
	P2CONST(CddCrc_WcrcChannelConfigType, TYPEDEF, CDDCRC_CONFIG_DATA)	pWcrcChannelConfig

	P2CONST(CddCrc_DmaConfigType, TYPEDEF, CDDCRC_CONFIG_DATA)	pDmaChannelConfig
	P2CONST(CddCrc_DmaUnitConfigType, TYPEDEF, CDDCRC_CONFIG_DATA)	pDmaUnitConfig
Description:	Identify the desired configuration for the CRC CDD.	

20.5.2.6 **CddCrc_WcrcChannelConfigType**

Table 20-7 CddCrc_WcrcChannelConfigType

Name:	CddCrc_WcrcChannelConfigType	
Type:	Structure	
Elements:	P2VAR(volatile uint32, TYPEDEF, REGSPACE)	pECMEN
Description:	Identify the desired configuration for the WCRC.	

20.5.2.7 **CddCrc_CrcModuleConfigType**

Table 20-8 CddCrc_CrcModuleConfigType

Name:	CddCrc_CrcModuleConfigType	
Type:	Structure	
Elements:	P2VAR(uint32, TYPEDEF, REGSPACE)	pDCRAnCINReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pDCRAnCOUTReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pDCRAnCTLReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pDCRAnCTL2Reg
	VAR(CddCrc_PolyType, TYPEDEF)	enPoly
	VAR(uint32, TYPEDEF)	ulDCRAnCTL2
Description:	Identify the desired configuration for CRC channel.	

20.5.2.8 **CddCrc_KcrcModuleConfigType**

Table 20-9 CddCrc_KcrcModuleConfigType

Name:	CddCrc_KcrcModuleConfigType	
Type:	Structure	
Elements:	P2VAR(uint32, TYPEDEF, REGSPACE)	pKCRCnDINReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pKCRCnDOUTReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pKCRCnCTLReg

	P2VAR(uint32, TYPEDEF, REGSPACE)	pKCRCnPOLYReg
	P2VAR(uint32, TYPEDEF, REGSPACE)	pKCRCnXORReg
	VAR(CddCrc_PolyType, TYPEDEF)	enPoly
	VAR(uint32, TYPEDEF)	ulCMDValue
	VAR(uint32, TYPEDEF)	ulKCRCmXOR
Description:	Identify the desired configuration for KCRC channel.	

20.5.2.9 **CddCrc_ChannelConfigType**

Table 20-10 CddCrc_ChannelConfigType

Name:	CddCrc_ChannelConfigType	
Type:	Structure	
Elements:	VAR(uint8, TYPEDEF)	ucHWIPType
	P2CONST(void, TYPEDEF, CDDCRC_CONFIG_DATA)	pChannelConfig
	VAR(uint8, TYPEDEF)	ucWcrcIndex
	P2VAR(volatile CddCrc_CommonRegType, TYPEDEF, REGSPACE)	pWcrcRegs
	P2CONST(volatile CddCrc_FifoRegType, TYPEDEF, REGSPACE)	pFifoRegs
	VAR(uint8, TYPEDEF)	ucInDmaIndex
	VAR(uint8, TYPEDEF)	ucOutDmaIndex
	VAR(uint8, TYPEDEF)	ucResDmaIndex
	VAR(uint8, TYPEDEF)	ucCmdDmaIndex
	P2FUNC(void, CDDCRC_APPL_CODE, pCompareEndNotification) (CddCrc_CompareResultType LddResult);	
Description:	Identify the desired configuration for each channel.	

20.5.2.10 **CddCrc_ModeType**

Table 20-11 CddCrc_ModeType

Name:	CddCrc_ModeType

Type:	Enum
Elements:	INDEPENDENT_MODE
	E2E_MODE
	AES_ACC_MODE
	DATA_THROUGH_MODE
	E2E_PLUS_DATA_THROUGH_MODE
Description:	Identify mode type of WCRC.

20.5.2.11 **CddCrc_PortType**

Table 20-12 CddCrc_PortType

Name:	CddCrc_PortType
Type:	Enum
Elements:	DATA_PORT
	RESULT_PORT
Description:	Identify port type to be read out.

20.5.2.12 **CddCrc_DmaConfigType**

Table 20-13 CddCrc_DmaConfigType

Name:	CddCrc_DmaConfigType	
Type:	Structure	
Elements:	VAR(uint8, TYPEDEF)	ucChannelIdx
	P2VAR(volatile CddCrc_DmaRegType, TYPEDEF, REGSPACE)	pDmaRegs
	VAR(uint16, TYPEDEF)	usDMRS
Description:	Identify the desired configuration for DMA channel.	

20.5.2.13 **CddCrc_DmaUnitConfigType**

Table 20-14 CddCrc_DmaUnitConfigType

Name:	CddCrc_DmaUnitConfigType	
Type:	Structure	
Elements:	P2VAR(volatile uint16, TYPEDEF, REGSPACE)	pRDMOR
Description:	Identify the desired configuration for DMA unit.	

20.5.2.14 **CddCrc_ChannelStateType**

Table 20-15 CddCrc_ChannelStateType

Name:	CddCrc_ChannelStateType	
Type:	Structure	
Elements:	VAR(CddCrc_ChannelSttType, TYPEDEF)	enStatus
	VAR(CddCrc_ModeType, TYPEDEF)	enMode
Description:	Identify state for each channel.	

20.5.2.15 **CddCrc_CompareFreqType**

Table 20-16 CddCrc_CompareFreqType

Name:	CddCrc_CompareFreqType
Type:	Enum
Elements:	FREQ_16_BYTE
	FREQ_32_BYTE
	FREQ_64_BYTE
Description:	Identify comparison frequency

20.5.2.16 **CddCrc_CompareResultType**

Table 20-17 CddCrc_CompareResultType

Name:	CddCrc_CompareResultType
Type:	uint16
Description:	Identify comparison result type

20.5.2.17 **CrcInDataPtr**

Table 20-18 CrcInDataPtr

Name:	CrcInDataPtr
Type:	const uint8*
Description:	Identify input buffer for CRC independent mode

20.5.2.18 **CrcOutDataPtr**

Table 20-19 CrcOutDataPtr

Name:	CrcOutDataPtr
Type:	uint8*
Description:	Identify output buffer for CRC independent mode

20.5.3 Function Definitions

The Table 20-20 describes the API provided by the CRC Complex Driver Component.

Table 20-20 API provided by the CRC Complex Driver Component

SI.No	AUTOSAR API
1	CddCrc_Init
2	CddCrc_GetVersionInfo

	RENESAS API
3	CddCrc_Process
4	CddCrc_SetMode
5	CddCrc_Write
6	CddCrc_SetupEB
7	CddCrc_ReadStatus
8	CddCrc_Command
9	CddCrc_Stop
10	CddCrc_Compare
11	CddCrc_ReadCompareResult
12	CddCrc_UnintendedModuleStopCheck

20.5.3.1 **CddCrc_Process**

Table 20-21 CddCrc_Process

Function Name:	CddCrc_Process		
Syntax:	FUNC(CddCrc_ReturnType, CDDCRC_CODE_SLOW) CddCrc_Process (CddCrc_ChannelType LddChannelID, uint32 LulDataSize, CrcInDataPtr LpDataIn, CrcOutDataPtr LpDataOut)		
Service ID:	0x02		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelID	Channel ID	
		Range:	0..15
	LulDataSize	Size of data block	
		Range:	1.. 4294967295

	LpDataIn	Pointer to start address of data block	
		Range:	Valid address
Parameters (In-Out):	None		
Parameters (Out):	LpDataOut	Pointer to address of output data	
		Range:	Valid address

Return Value:	CddCrc_ReturnType
	{
	CDDCRC_OK,
	CDDCRC_NOT_OK,
	CDDCRC_BUSY
	}		

Description:	This Runnable Entity serves as ServerCallPoint for application SWC to get CRC value from CRC Driver.		
Preconditions:	The CRC Complex Driver must be initialized.		
Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.		
DET/DEM Errors handled:	CDDCRC_E_UNINIT	CddCrc Complex Driver is not initialized	
	CDDCRC_E_PARAM_POINTER	LpDataIn or LpDataOut is NULL_PTR	
	CDDCRC_E_PARAM_LENGTH	LulDataSize is equal to zero	
	CDDCRC_E_WRITE_VERIFY	Read back value does not same as written value of register	

20.5.3.2 **CddCrc_SetMode**

Table 20-22 CddCrc_SetMode

Function Name:	CddCrc_SetMode
Syntax:	FUNC(Std_ReturnType, CDDCRC_CODE_SLOW) CddCrc_SetMode (CddCrc_ChannelType LddChannelId, CddCrc_ModeType LenMode)
Service ID:	0x03
Sync/Async:	Synchronous

Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	Channel ID	
		Range:	0..15
	CddCrc_ModeType	Operation Mode	
		Range:	INDEPENDENT_MODE, E2E_MODE, AES_ACC_MODE, DATA_THROUGH_MODE, E2E_PLUS_DATA_THROUGH_MODE
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)
Description:	This service to select operation mode for WCRC HW module.		
Preconditions:	The CRC Complex Driver must be initialized.		
Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.		
DET/DEM Errors handled:	CDDCRC_E_UNINIT		CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_MODE		The mode is out of range
	CDDCRC_E_PARAM_CHANNEL		The channel ID is out of range
	CDDCRC_E_INCOMPATIBLE_HWIP		Mode is not compatible with HWIP
	CDDCRC_E_WRITE_VERIFY		When CddCrcRegisterWriteVerify parameter is configured as TRUE and the verification of written value to register failed.

20.5.3.3 **CddCrc_Write**

Table 20-23 CddCrc_Write

Function Name:	CddCrc_Write
Syntax:	FUNC(Std_ReturnType, CDDCRC_PUBLIC_CODE) CddCrc_Write (

	CddCrc_ChannelType LddChannelId, P2CONST(uint8, AUTOMATIC, CDDCRC_APPL_DATA) LpSrcAddress, uint32 LulDataLength, uint32 LulConvSize)		
Service ID:	0x04		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelID	Channel ID	
		Range:	0..15
	LpSrcAddress	Pointer to start address of data block	
		Range:	Valid address
	LulDataLength	Size of data block	
		Range:	1.. 4294967295
	LulCovSize	Conversion size	
		Range:	1..1048576
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)
Description:	This service supports to transfer input data to data port of FIFO by DMAC.		
Preconditions:	The CRC Complex Driver must be initialized.		
Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.		
DET/DEM Errors handled:	CDDCRC_E_UNINIT		CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_POINTER		Invalid pointer
	CDDCRC_E_INCOMPATIBLE_MODE		Incompatible mode
	CDDCRC_E_PARAM_CHANNEL		Invalid channel ID
	CDDCRC_E_PARAM_LENGTH		Invalid data length

20.5.3.4 CddCrc_SetupEB

Table 20-24 CddCrc_SetupEB

Function Name:	CddCrc_SetupEB		
Syntax:	FUNC(Std_ReturnType, CDDCRC_PUBLIC_CODE) CddCrc_SetupEB (CddCrc_ChannelType LddChannelId, P2CONST(uint32, AUTOMATIC, CDDCRC_APPL_DATA) LpDesAddress, uint32 LulDataLength, CddCrc_PortType LddPort)		
Service ID:	0x05		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	Channel ID	
		Range:	0..15
	LpDesAddress	Pointer to start address of result block	
		Range:	Valid address
	LulDataLength	Size of data block	
		Range:	1.. 4294967295
	LddPort	Port type from which data is read out	
		Range:	DATA_PORT, RESULT_PORT
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)
Description:	This service is to read out data from data/result port of FIFO by DMAC		
Preconditions:	The CRC Complex Driver must be initialized.		
Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.		

DET/DEM Errors handled:	CDDCRC_E_UNINIT	CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_POINTER	Invalid pointer
	CDDCRC_E_INCOMPATIBLE_MODE	Incompatible mode
	CDDCRC_E_PARAM_CHANNEL	Invalid channel ID
	CDDCRC_E_PARAM_LENGTH	Invalid data length
	CDDCRC_E_PARAM_PORT	Invalid port

20.5.3.5 **CddCrc_ReadStatus**

Table 20-25 CddCrc_ReadStatus

Function Name:	CddCrc_ReadStatus		
Syntax:	FUNC(Std_ReturnType, CDDCRC_PUBLIC_CODE) CddCrc_ReadStatus (CddCrc_ChannelType LddChannelId, P2VAR(CddCrc_ChannelSttType, AUTOMATIC, CDDCRC_APPL_DATA) LpChannelStatus)		
Service ID:	0x06		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	Channel ID	
		Range:	0..15
Parameters (In-Out):	None		
Parameters (Out):	LpChannelStatus	The status of the channel	
		Range:	CDDCRC_CH_OK, CDDCRC_CH_BUSY, CDDCRC_CH_STOPPED
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)
Description:	This service return the status of the channel.		
Preconditions:	The CRC Complex Driver must be initialized.		

Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.	
DET/DEM Errors handled:	CDDCRC_E_UNINIT	CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_POINTER	Invalid pointer
	CDDCRC_E_PARAM_CHANNEL	Invalid channel ID

20.5.3.6 CddCrc_Command

Table 20-26 CddCrc_Command

Function Name:	CddCrc_Command		
Syntax:	FUNC(Std_ReturnType, CDDCRC_PUBLIC_CODE) CddCrc_Command (CddCrc_ChannelType LddChannelId, P2CONST(uint32, AUTOMATIC, CDDCRC_APPL_DATA) LpSrcAddress, uint32 LulDataLength)		
Service ID:	0x07		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	Channel Id	
		Range:	0..15
	LpSrcAddress	Pointer to start address of data block	
		Range:	Valid address
	LulDataLength	Size of data block	
		Range:	1.. 4294967295
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)
Description:	This service supports access register by command function via DMAC		
Preconditions:	The CRC Complex Driver must be initialized.		

Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.	
DET/DEM Errors handled:	CDDCRC_E_UNINIT	CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_POINTER	Invalid pointer
	CDDCRC_E_INCOMPATIBLE_MODE	Incompatible mode
	CDDCRC_E_PARAM_CHANNEL	Invalid channel ID
	CDDCRC_E_PARAM_LENGTH	Invalid data length

20.5.3.7 **CddCrc_Stop**

Table 20-27 CddCrc_Stop

Function Name:	CddCrc_Stop		
Syntax:	FUNC(Std_ReturnType, CDDCRC_PUBLIC_CODE) CddCrc_Stop (CddCrc_ChannelType LddChannelId)		
Service ID:	0x0A		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	Channel index	
		Range:	0..15
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)
Description:	This service supports to stop a channel		
Preconditions:	The CRC Complex Driver must be initialized.		
Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.		
DET/DEM Errors handled:	CDDCRC_E_UNINIT		CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_CHANNEL		Invalid pointer

20.5.3.8 **CddCrc_Compare**

Table 20-28 CddCrc_Compare

Function Name:	CddCrc_Compare		
Syntax:	FUNC(Std_ReturnType, CDDCRC_PUBLIC_CODE) CddCrc_Compare (CddCrc_ChannelType LddChannelId, P2CONST(uint8, AUTOMATIC, CDDCRC_APPL_DATA) LpDataIn, P2CONST(uint32, AUTOMATIC, CDDCRC_APPL_DATA) LpExpData, CddCrc_CompareFreqType LenCompFreq, uint32 LulConvSize)		
Service ID:	0x0B		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	Channel index	
		Range:	0..15
	LpDataIn	Pointer to start address of data block	
		Range:	Valid address
	LulExpData	Pointer to start address of expected CRC block	
		Range:	Valid address
	LenCompFreq	Compare frequency	
		Range:	FREQ_16_BYTE, FREQ_32_BYTE, FREQ_64_BYTE
	LulConvSize	Conversion size	
		Range:	1..1048576
Parameters (In-Out):	None		
Parameters (Out):	None		
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)

Description:	This service supports to compare CRC result with expected CRC code.	
Preconditions:	The CRC Complex Driver must be initialized.	
Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.	
DET/DEM Errors handled:	CDDCRC_E_UNINIT	CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_POINTER	Invalid pointer
	CDDCRC_E_INCOMPATIBLE_HWIP	HWIP is not compatible with this function
	CDDCRC_E_PARAM_CHANNEL	Invalid channel ID
	CDDCRC_E_PARAM_COMPARE_FREQ	Invalid comparison frequency
	CDDCRC_E_PARAM_CONVERSION_SIZE	Invalid conversion size

20.5.3.9 **CddCrc_ReadCompareResult**

Table 20-29 CddCrc_ReadCompareResult

Function Name:	CddCrc_ReadCompareResult		
Syntax:	FUNC(Std_ReturnType, CDDCRC_PUBLIC_CODE) CddCrc_ReadCompareResult (CddCrc_ChannelType LddChannelId, P2VAR(uint16, AUTOMATIC, CDDCRC_APPL_DATA) LpCompareResult)		
Service ID:	0x0C		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (In):	LddChannelId	Channel index	
		Range:	0..15
Parameters (In-Out):	None		
Parameters (Out):	LpCompareResult	Comparison result	
		Range:	0..65535
Return Value:	Std_ReturnType	**Range:**	E_OK: Request accepted E_NOT_OK: A development error occurred (see DET/DEM Errors handled)

Description:	This service return comparison result	
Preconditions:	The CRC Complex Driver must be initialized.	
Remarks:	User should not call this API directly. It should be configured as part of AUTOSAR interface.	
DET/DEM Errors handled:	CDDCRC_E_UNINIT	CddCrc Complex Driver is not initialized
	CDDCRC_E_PARAM_CHANNEL	Invalid channel ID

20.5.3.10 **CddCrc_UnintendedModuleStopCheck**

Table 20-30 CddCrc_UnintendedModuleStopCheck

Function Name:	CddCrc_UnintendedModuleStopCheck	
Syntax:	FUNC(void, CDDCRC_PUBLIC_CODE) CddCrc_UnintendedModuleStopCheck(void)	
Service ID:	0x0D	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	None	
Parameters (In-Out):	None	
Parameters (Out):	None	
Return Value:	None	
Description:	This service supports for unintended module stop check	
Preconditions:	The CRC Complex Driver must be initialized. Enable SM via parameter CddCrcUnintendedModuleStopCheck	
Remarks:	-	
DET/DEM Errors handled:	CDDCRC_E_UNINIT	CddCrc Complex Driver is not initialized
	CDDCRC_E_UNINTENDED_MODULE_STOP_FAILURE	An unintended module stop is detected

20.5.4 Preemption of APIs

The Table 20-31 shows list of Preemption Table of APIs of the CRC Driver

-: cannot be invoked at the same time

√: can be invoked at the same time

Table 20-31 Preemption Table of APIs of the CRC Driver

	CddCrc_Init	CddCrc_ GetVersionInfo	CddCrc_Process	CddCrc_SetMode	CddCrc_Write	CddCrc_SetupEB	CddCrc_ReadStatus	CddCrc_Command	CddCrc_Stop	CddCrc_Compare	CddCrc_ReadCompareResult	CddCrc_UnintendedModuleStopCheck
CddCrc_Init	-											
Cdd_GetVersionInfo	√	√										
CddCrc_Process	-	√	√									
CddCrc_SetMode	-	√	√	√								
CddCrc_Write	-	√	√	√	√							
CddCrc_SetupEB	-	√	√	√	√	√						
CddCrc_ReadStatus	-	√	√	√	√	√	√					
CddCrc_Command	-	√	√	√	√	√	√	√				
CddCrc_Stop	-	√	√	√	√	√	√	√	√			
CddCrc_Compare	-	√	√	√	√	√	√	√	√	√		
CddCrc_ReadCompareResult	-	√	√	√	√	√	√	√	√	√	√	
CddCrc_UnintendedModuleStopCheck	-	√	√	√	√	√	√	√	√	√	√	√

20.6 Development and Production Errors

In this chapter the development errors that are reported by the CRC Complex Device Driver Component are tabulated. The development errors will be reported only when the pre-compiler option CddCrcDevErrorDetect is enabled in the configuration. The production code errors are not supported by CRC Complex Device Driver Component.

20.6.1 CRC Driver Component Development Errors

The following table contains the DET errors that are reported by CRC Complex Driver Component. These errors are reported to DET Module when the CRC Complex Driver Component APIs is invoked with wrong input parameters or without initialization of the driver.

Table 20-32 DET Errors of CRC Driver Component (1/3)

Sl. No.	1
Error Code	CDDCRC_E_UNINIT
Value	0x01
Related API(s)	CddCrc_Process(), CddCrc_SetMode(), CddCrc_Write(), CddCrc_SetupEB(), CddCrc_ReadStatus(), CddCrc_Command(), CddCrc_Stop, CddCrc_Compare(), CddCrc_ReadCompareResult(), CddCrc_UnintendedModuleStopCheck
Source of Error	When the APIs are invoked without the initialization of CRC Complex Driver Component.
Sl. No.	**2**
Error Code	CDDCRC_E_ALREADY_INITIALIZED
Value	0x02
Related API(s)	CddCrc_Init()
Source of Error	When the API CddCrc_Init is invoked more than once.

Table 20-33 DET Errors of CRC Driver Component (2/3)

Sl. No.	3
Error Code	CDDCRC_E_PARAM_CHANNEL
Value	0x03
Related API(s)	CddCrc_Process(), CddCrc_SetMode(), CddCrc_Write(), CddCrc_SetupEB(), CddCrc_ReadStatus(), CddCrc_Command(), CddCrc_Stop(), CddCrc_Compare(), CddCrc_ReadCompareResult()
Source of Error	When the API is invoked with the arguments of invalid channel ID.
Sl. No.	**4**
Error Code	CDDCRC_E_PARAM_POINTER
Value	0x04
Related API(s)	CddCrc_GetVersionInfo(), CddCrc_Init(), CddCrc_Process(), CddCrc_Write(), CddCrc_SetupEB(), CddCrc_ReadStatus(), CddCrc_Command(), CddCrc_Compare()
Source of Error	When the API is invoked with the arguments of NULL pointer.
Sl. No.	**5**
Error Code	CDDCRC_E_INVALID_DATABASE
Value	0x05
Related API(s)	CddCrc_Init()
Source of Error	When the above-mentioned API is invoked with invalid database address.
Sl. No.	**6**
Error Code	CDDCRC_E_PARAM_MODE
Value	0x06
Related API(s)	CddCrc_SetMode()
Source of Error	When the above-mentioned API is invoked with invalid mode.
Sl. No.	**7**
Error Code	CDDCRC_E_PARAM_PORT
Value	0x08
Related API(s)	CddCrc_SetupEB()
Source of Error	When the above-mentioned API is invoked with invalid port.
Sl. No.	**8**
Error Code	CDDCRC_E_PARAM_LENGTH
Value	0x09
Related API(s)	CddCrc_Process(), CddCrc_Write(), CddCrc_SetupEB(), CddCrc_Command()
Source of Error	When the above-mentioned API is invoked with data length.
Sl. No.	**9**
Error Code	CDDCRC_E_INCOMPATIBLE_MODE
Value	0x0A
Related API(s)	CddCrc_Write(), CddCrc_SetupEB(), CddCrc_Command()
Source of Error	When the above-mentioned API is invoked when the WCRC mode is incompatible for setting.
Sl. No.	**10**
Error Code	CDDCRC_E_PARAM_CONVERSION_SIZE
Value	0x0B
Related API(s)	CddCrc_Write(), CddCrc_Compare()
Source of Error	When the above-mentioned API is invoked with invalid conversion size

Table 20-34 DET Errors of CRC Driver Component (3/3)

Sl. No.	11
Error Code	CDDCRC_E_INCOMPATIBLE_HWIP
Value	0x0C
Related API(s)	CddCrc_Compare(), CddCrc_SetMode()
Source of Error	When the above-mentioned API is invoked with incompatible HWIP
Sl. No.	**12**
Error Code	CDDCRC_E_PARAM_COMPARE_FREQ
Value	0x0D
Related API(s)	CddCrc_Compare()
Source of Error	When the above-mentioned API is invoked with invalid compare frequency

20.6.2 CRC Driver Component Production Errors

The following table contains Renesas specific DEM errors that are reported by CRC Complex Driver Component.

Table 20-35 DEM Errors of CRC Driver Component

Sl. No.	1
Error Code	CDDCRC_E_WRITE_VERIFY
Related API(s)	CddCrc_Process()
Source of Error	When the written value to CRC registers is incorrect. Detection mechanism is read-back registers value and compare with written.
Sl. No.	**2**
Error Code	CDDCRC_E_HARDWARE_ERROR
Related API(s)	CDDCRC_DMA<00..63>_ISR
Source of Error	When the HW registers work incorrectly, the status bits is not set when transfering is completed.
Sl. No.	**3**
Error Code	CDDCRC_E_INTERRUPT_CONTROLLER_FAILURE
Related API(s)	CDDCRC_DMA<00..63>_ISR
Source of Error	When DMA’s Interrupt Service Routine (ISR) is triggered, interrupt status bit of that interrupt source is not set
Sl. No.	**4**
Error Code	CDDCRC_E_UNINTENDED_MODULE_STOP_FAILURE
Related API(s)	CddCrc_UnintendedModuleStopCheck
Source of Error	The error in “Module Standby, Software Reset” that causes unintended module stop is detected

20.7 CRC Driver Component Runtime Errors

None

20.8 Memory Organization

The following tables depict a typical memory organization, which must be met for proper functioning of CRC Driver Component software.

Table 20-36 ROM sections of the CRC Driver

Section Name	Alignment (*1)	Description
CDDCRC_PRIVATE_CODE_ROM	-	Internal functions of CRC Complex Driver Component that can be located in code memory.
CDDCRC_PUBLIC_CODE_ROM	-	API(s) of CRC Complex Driver Component that can be located in code memory.
CODE_FAST	-	Interrupt functions CRC Complex Driver Component that can be located in code memory.
CONFIG_DATA_UNSPECIFIED	-	This section is used for global constants aligned to 32 bits and table of contents generated by the CRC Complex Driver Component Generation Tool.
CONFIG_DBTOC_DATA_UNSPECIFIED	-	This section consists of CRC Complex Driver Component database table of contents generated by the CRC Complex Driver Component Generation Tool. It can be located in code memory.
CONST_UNSPECIFIED	-	This section consists of the global constants of unspecified size that are used internally by CRC Complex Driver Component. This can be located in data memory.

Note:
*1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

Table 20-37 RAM sections of the CRC Driver

Section Name	Alignment (*1)	Description
VAR_INIT_8	-	This section consists of the global RAM variables of 8-bit size that are initialized by start-up code and used internally by CRC Complex Driver Component. This can be located in data memory.
VAR_NO_INIT_PTR	-	This section consists of the global RAM variables of 32-bit size that are not initialized by start-up code and used internally by CRC Complex Driver Component. This can be located in data memory.
VAR_CLEARED_UNSPECIFIED	-	This section consists of the global RAM variables of unspecified size that are used internally by CRC Complex Driver Component. This can be located in data memory.
VAR_CLEARED_DESC_UNSPECIFIED	16 bytes	This section consists of the global RAM variables aligned to 16 bytes that are used internally by CRC Complex Driver Component. This can be located in DRAM.

Note:
*1: "-" means that the alignment for this section can be set with any value. When the alignment is set, the section's address needs to be adjusted along with the alignment.

20.9 Device-Specific Information

The device supports the following devices:

Refer to "*R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview*" – chapter "*5.1 Product*".

20.9.1 Interaction Between the User and CRC Driver Component

The details of the services supported by the CRC Driver Component to the upper layers users and the mapping of the channels to the hardware units is provided in the following sections:

20.9.1.1 Channel Mapping

The channel setting does not depend on H/W resources.

Table 20-38 Hardware CDDCRC channel mapping

SI. No	Hardware channel
1	CRC<0..3> KCRC<0..3>
2	WCRC<0..3>
3	RT-DMAC<00..63>

20.9.1.2 ISR Functions

The provide the list of handlers corresponding to the hardware unit ISR(s) in CRC Driver Component. The user should configure the ISR functions mentioned below:

Table 20-39 ISR Handler Addresses

Interrupt Source	Name of the ISR Function (<n>=00..63)
RT-DMAC	CDDCRC_DMA<n>_ISR
	CDDCRC_DMA<n>_CAT2_ISR

20.9.2 Multi-Core / Multi-Instantiation

CRC driver does not support multi-core and multi-instantiation for this device.

20.10 Non-AUTOSAR environment integration

The CRC Complex Driver Components for Renesas R-Car Series, 4th Generation SoC is assumed to be integrated in the AUTOSAR BSW environment. However, in special case where such environment is not available, additional steps need to be taken. This chapter explains the application notice to integrate the CRC Complex Driver Components to Non-AUTOSAR environment.

20.10.1 Stub modules handling

20.10.1.1 Os

None

20.10.1.2 Det

The Det stub files are organized in the following folder:

\common\generic\stubs\<Autosar version>\Det

In the AUTOSAR environment, CRC Complex Driver Components uses Det_ReportError API provided by the DET module to report a development error e.g., CRC Complex Driver has not been initialized, API is provided with invalid parameter… The API prototype is as of follow:

Std_ReturnType Det_ReportError (uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

Current Det stub implementation simply stored all the reported DET errors to global array GstDetErrBuffer[] which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Det_ReportError API with their current error handling strategy.

20.10.1.3 Dem

The Dem stub files are organized in the following folder:

\common\generic\stubs\<Autosar version>\Dem

In the AUTOSAR environment, CRC Complex Driver Components uses Dem_SetEventStatus API provided by the DEM module to report a production error. The API prototype is as of follow:

Dem_SetEventStatus (Dem_EventIdType EventId, Dem_EventStatusType EventStatus)

Current Dem stub implementation simply stored all the reported DEM errors to global variables Dem_EventId and Dem_EventStatus which can be used in debugging the Sample application.

Non-AUTOSAR users can modify the provided Dem_SetEventStatus API with their current error handling strategy.

20.10.1.4 Basic Software Scheduler

SchM (Basic Software Scheduler) is a part of RTE (Run-time Environment) in AUTOSAR ECU Architecture.

The SchM (Basic Software Scheduler) stub files are organized in the following folder:

\common\generic\stubs\<Autosar version>\Rte\

CRC driver needs SchM module to access global resources or registers when it needs to access. SchM module is enabled when CDDCRC_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON. With Non-AUTOSAR environment, user needs to prepare SchM stub to user CRC driver as following:

Write SchM functions with prototype as below:

void SchM_Enter_CddCrc_CDDCRC_RAM_DATA_PROTECTION (void);

void SchM_Exit_CddCrc_CDDCRC_RAM_DATA_PROTECTION (void);

void SchM_Enter_CddCrc_CDDCRC_INTERRUPT_CONTROL_PROTECTION (void);

void SchM_Exit_CddCrc_CDDCRC_INTERRUPT_CONTROL_PROTECTION (void);

20.10.2 Callback function usage

The CRC Complex Driver Component has a notification callback function being used to notify completion of comparing CRC result to application SWC.

The notification callback function which will be configured in the parameter “CddCrcCompareEndNotification”. The example can be referred in below files:

\rel\modules\cddcrc\sample_application\src\App_CDD_CRC_Common_Sample.c

\rel\modules\cddcrc\sample_application\V4H\19_11\config\ App_CDD_CRC_V4H_Sample.arxml

20.10.3 Scheduled function usage

The CRC Complex Driver Component provides CddCrc_Process as a scheduled function.

20.10.4 Interrupt handling usage

The sample Interrupt Vector Table files are organized in the following folder:

\rel\V4H\common_family\src\arm\Interrupt_VectorTable.c

Revision History		R-Car V4H AUTOSAR R19-11 MCAL User's Manual	
Rev.	**Date**	**Description**	
		Page	**Summary**
3.00	Mar 27, 2025	-	Cover, footer and colophon: - Update Rev and issue date. 1.Introduction: - Update Release Version in Table 1.1 2.Reference Documents - Update references version at Table 2.1 Reference Documents(1/2).
2.02	Feb 28, 2025	-	Cover, footer and colophon: - Update Rev and issue date. 1.Introduction: - Update Release Version in Table 1.1 2.Reference Documents - Update references version at Table 2.1 Reference Documents(1/2).
2.01	Jan 22, 2025	-	Cover, footer and colophon: - Update Rev and issue date. 1.Introduction: - Update Release Version in Table 1.1 5.DIO - At section 1.2.1 General: Add information support the parameter "DioDomainId". 7.PORT: - At section 1.2.1 General: Add information support the parameter "PortDomainId". 9.FLS - Section 1.2.2: Add precondition information related to MCU Bus Domain setting. 12.MCU: - Section 12.2.2: Add information related to parameter "McuDomainId" and Bus Domain protection setting. 14.SPI: - Section 14.2.1 General: Add new information about domain protection of regionID support port pin chip select.
2.00	Nov 27, 2024	-	Cover, footer and colophon: - Update Rev and issue date. 1.Introduction: - Update Release Version in Table 1.1 2.Reference Documents - Update references version at Table 2.1 Reference Documents(Update 1/2/3 and Add 4). 7.PORT: - Section 7.2.1: Add information support the parameter "PortPinInitialMode" is configured as CANFD5, the CANFD7 functionality will be reflected in these PortPin. 12.MCU: - Section 12.2.1: Add restriction related to parameters for the Module Stop Control Register must be always configured as False.

1.01	Oct 29, 2024	-	Cover, footer and colophon: - Update Rev and issue date. 5.CAN: - Section 5.9.2: Add “Multi-Core / Multi-Instantiation” section. 6.DIO: - Section 6.9.1: Update “Multi-Core / Multi-Instantiation” information 7.PORT: - Section 7.9.2: Add “Multi-Core / Multi-Instantiation” section. 8.ETH: - Section 8.9.2: Add “Multi-Core / Multi-Instantiation” section. 9.FLS: - Section 9.10.2: Add “Multi-Core / Multi-Instantiation” section. 10.GPT: - Section 10.9.1: Update “Multi-Core / Multi-Instantiation” section. 11.ICCOM: - Section 11.9.2: Add “Multi-Core / Multi-Instantiation” section. 12.MCU: - Section 12.9.2: Add “Multi-Core / Multi-Instantiation” section. 13.IIC: - Section 13.9.2: Add “Multi-Core / Multi-Instantiation” section. 14.SPI: - Section 14.9.1: Add “Multi-Core / Multi-Instantiation” section (Remove “Inter Core Exclusive Control” section) 15.WDG: - Section 15.9.2: Add “Multi-Core / Multi-Instantiation” section. 16.THS: - Section 16.9.3: Add “Multi-Core / Multi-Instantiation” section. 17.IPMMU: - Section 17.9.2: Add “Multi-Core / Multi-Instantiation” section. 18.EMM: - Section 18.9.2: Add “Multi-Core / Multi-Instantiation” section. 19.RFSO: - Section 19.9.2: Add “Multi-Core / Multi-Instantiation” section. 20.CRC: - Section 20.9.2: Add “Multi-Core / Multi-Instantiation” section.
1.00	Nov 28, 2023	-	Cover, footer and colophon: - Update Rev and issue date. 1.Introduction: - Update Release Version in Table 1.1 2.Reference Documents - Update references version at Table 2.1 Reference Documents(1/2). 5.CAN: - Section 5.2.1 General: Update HW UM references information. 11. ICCOM: - Table 11-1: Update HW UM references information. 13. IIC: - Table 13-1: Update HW UM references information.

			18. EMM: - Table 18-1: Update HW UM references information.
0.22	Oct 18, 2023	-	Cover, footer and colophon: - Update Rev and issue date. 1.Introduction: - Update Release Version and Modules version in Table 1.1 2.Reference Documents - Update references version at Table 2.1 Reference Documents(1/2). 12. MCU: - Section 12.2.1 General: Update mandatory configuration for PLL5 Control Register and Module Stop Control Register. 14.SPI: - Section 14.2.2 Preconditions: Add limitation inconsistent name of SYS-DMAC channels instance and register name.
0.21	Aug 30, 2023	-	Cover, footer and colophon: - Update Rev and issue date. 1.Introduction: - Update Release Version and Modules version in Table 1.1 2.Reference Documents - Update references version at Table 2.1 Reference Documents(1/2). 5.CAN - Section 5.2.1 General: Not supported DEEPSTOP functionality. 6.DIO - Section 6.5.2.9: Change Dio_HwFuncTableType: Row "Name": Dio_HwFuncTableType - Section 6.5.2.10: Change Dio_HwConfigType: Row "Name": Dio_HwConfigType - Section 6.9.1 Multi-core: Not support multi-core 7.PORT: - Section 7.10.1 Stub modules handling: remove information of OS stub function 8. ETH - Delete Eth_GetEgressTimeStamp API from table Table 8.4 ETH Driver Protected Resource List - Delete ETH_E_INV_MODE for Eth_EnableEgressTimeStamp from Table 8.6 ETH Driver Error List (2/4) - Add ETH_E_INV_PARAM for DET/DEM Errors handled field in Table 8.16 Eth_SetIncrementTimeForGptp - Add Eth_ProvideTxBuffer, Eth_GetCurrentTime, Eth_GetEgressTimeStamp, Eth_GetIngressTimeStamp for ETH_E_INV_MODE and Eth_SetIncrementTimeForGptp for ETH_E_INV_PARAM in Table 8.21 DET Errors of ETH Driver Component (2/2) - Add reference information in section 8.9.1.4 Parameter Definition File 10. GPT: - Section 10.2.1 General: Add information related to SASYNC/SASYNCD2 - Section 10.9.1 Multi-core / Multi-instantiation: Update the content to “GPT driver is not support Multi-core/Multi-instantiation.” 12. MCU:

			- Section 12.3 Architecture Details: Update information of Clock Initialization 14.SPI: - Section 14.6.2: Remove Spi_Init, Spi_DeInit, Spi_MainFunction_Handling, Spi_AsyncTransmit, Spi_Cancel from the Related API(s) of SPI_E_DATA_TX_TIMEOUT_FAILURE. 15. WDG: - Section 15.2.2 Preconditions: + Correct Frequency of RCLK to 32.8 kHz. + Correct name of name of Hardware IP from DBSC4 to DBSC5. - Section 15.3 Architecture Details + Correct number of section number to 15.2.4 WDG State Diagram. - Section 15.4 WDG Driver Component Header and Source File Description: + Correct name of Module Overiew to R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview. - Add section 15.9 Device-Specific Information. 16. THS: - Section 16.6.1 THS Complex Device Driver Component Development Errors: + Update the Related API(s) of Error Code CDD_THS_E_INVALID_VALUE to CddThs_ConfigureThermalInterruption. 18. EMM: - Section 18.2.2 Preconditions: + Correct name of bit EOE in ECMERROUTCTLR register to ERROUT - Section 18.10. Non-AUTOSAR environment integration: + Update information of Generation SoC from 3rd to 4th 19. RFSO: - Section 19.6.1: Add CddRfso_IntervalTimerInterruptStatus, CddRfso_GetVersionInfo to the Related API(s) of CDDRFSO_E_PARAM_POINTER

0.20	Jul 25, 2023	-	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references version at Table 2.1 Reference Documents(1/2). 7.PORT - Section 7.2.2. Preconditions: + In precondition related to MODSEL registers, remove information of TSN0 and AVBn 14.SPI - Section 14.2.2 Preconditions: Add limitation and workaround related to SpiSupportConcurrentSyncTransmit
0.19	Jul 08, 2023	-	1.Introduction: Update Release Version in Table 1.1 2.Reference Documents Update references version at Table 2.1 Reference Documents(1/2). 7.PORT - Section 7.2.2. Preconditions: + In precondition for Port Pin container configuration: change symbolic name from Port_Config to PortConf 12. MCU - Section 12.2.2. Preconditions: Remove the precondition for McuResetReasonConf container.
0.18	May 31, 2023	-	1.Introduction: Update Release Version in Table 1.1 2.Reference Documents Update references version at Table 2.1 Reference Documents(1/2). 7.PORT - Section 7.2.2. Preconditions: Update the precondition for Port Pin container configuration 12. MCU - Section 12.2.2. Preconditions: Update the precondition for McuResetReasonConf container.
0.17	May 05, 2023	-	1.Introduction: Update Release Version in Table 1.1 2.Reference Documents Update references version at Table 2.1 Reference Documents(1/2). 8. ETH - Section 8.2.5, Table ETH Driver Error List (3/4), in API Eth_MainFunction: remove row ETH_E_LATECOLLISION - Section 8.6.2, table DEM Errors of ETH Driver Component + Remove descriptions related to error code ETH_E_LATECOLLISION + In descriptions of error code ETH_E_REGISTER_CORRUPTION: Remove Eth_SetControllerMode - Section 8.9.1.2, table "ISR Function for the Device" ISR name is update as below: + ETH_AVB0ERRORISR becomes ETH_AVB0ERRISR + ETH_AVB0ERRORISR_CAT2 becomes ETH_AVB0ERRISR_CAT2 + ETH_AVB1ERRORISR becomes ETH_AVB1ERRISR + ETH_AVB1ERRORISR_CAT2 becomes ETH_AVB1ERRISR_CAT2

			+ ETH_AVB2ERRORISR becomes ETH_AVB2ERRISR + ETH_AVB2ERRORISR_CAT2 becomes ETH_AVB2ERRISR_CAT2 14. SPI - Section 14.6.2: Add API Spi_Cancel to “Related API(s)” of SPI_E_DATA_TX_TIMEOUT_FAILURE and SPI_E_WRITE_VERIFY_FAILURE 16. THS: - Section 16.3 Architecture Details, description of De-Initialization: change IDLE to CDD_THS_IDLE and add information “HW “Standby mode”” - Sections 16.5.2.1, 16.5.2.4, 16.5.3.5, 16.5.3.6: Add note related to CDD_THS_IDLE - Section 16.5.3.2. CddThs_ConfigureThermalInterruption, row "Prototype": Update type of parameter InterruptionValue to “sint16" 19. RFSO - Section 19.5.3: Correct the description of API "CddRfso_GetCFEPinStatus" in table "APIs provided by the RFSO Complex Driver Component" as "This API gets status of FSO_CFE0, FSO_CFE1, CFEO0 and CFEO1 bits of selected RFSO channel." - Section 19.5.3.2. Correct “IntervalTimerTime” as "IntervalTime" in Row "Syntax" and "Parameters (In):" - Section 19.5.3.3. Correct “IntervalTimerCycle” as " IntervalCycle" in Row "Syntax" and "Parameters (In):"
0.16	Mar 24, 2023	-	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references version at Table 2.1 Reference Documents(1/2). 5.CAN - Section 5.8 Memory Organization: Update section name for RAM and ROM 8.ETH - Section 8.2.2 Preconditions: Update precondition for feature EthSwitchManagementSupport - Section 8.4 ETH Driver Component Header and Source File Description: remove Eth_Rsw2_PBcfg.c information, add information of ETH Driver Component C header files and Component source files. - Section 8.8 Memory Organization Update section name for RAM and ROM 9.FLS: - Section 9.9 Memory Organization: Update section name for RAM and ROM. - Figure 9.5 FLS Driver Component Overview Memory Organization: Update section name for RAM and ROM. 10. GPT - Section 10.8 Memory Organization: + Update the name and description of ROM / RAM sections 12.MCU - Update number whole table in section MCU. - Section 12.2.1 General: Add notice configure for ZG clock. Add notice configure for PLL5 Control Register by table 12.1. - Section 12.8 Memory Organization:

			Rename section PUBLIC_CODE to MCU_PUBLIC_CODE_ROM, PRIVATE_CODE to MCU_PRIVATE_CODE_ROM, VAR_INIT_BOOLEAN to VAR_INIT_1 in Table 12.16. Remove section CODE_FAST, VAR_INIT_UNSPECIFIED, VAR_INIT_32 in Table 12.16. - Section 12.5.2.6 Mcu_RamStateType, update info for enum macros. - Section 12.4 MCU Driver Component Header and Source File Description, update number sections. 11. ICCOM - Section 11.8 Memory Organization: + Update memory section name for RAM and ROM section 13. IIC - Section13.2.2: + Add note 3 for description about Fix duty node and remove note for Variable duty description. - Section 13.4 Update references location from section 3.4.1.3 to section 3.4.3.3 - Section 13.5.3: + Add CddIic_Ch<n>NoticeCallback (<n> = 0..5) in number 7 of table 13 -20 APIs provided by the IIC Complex Driver Component. + Add section 13.5.3.5 CddIic_Ch<n>NoticeCallback - Section 13.6.1: update for table 13-22 DET Errors of IIC Complex Driver Component for “Related API(s)” of SI. No 5. - Section 13.6.2: update for table 13-23 DEM Errors of IIC Complex Driver Component for “Related API(s)” of SI. No 2. - Section 13.8: + Rename of section name: from” CDDIIC_CODE_SLOW” to “CODE_SLOW”, from “CDDIIC_CODE_FAST” to “CODE_FAST”, from “CDDIIC_VAR_CLEARED_32” to “VAR_CLEARED_32” , from “CDDIIC_VAR_INIT_BOOLEAN” to “VAR_INIT_1”, from “CDDIIC_VAR_NO_INIT_PTR” to “VAR_NO_INIT_PTR”, from “CDDIIC_CONFIG_DATA_32” to “CONFIG_DATA_32”. + Add section “CODE” to X1 of figure 13-4 and Rom Section (X1,X2 and X3) 14. SPI: - Section 14.8 Memory Organization: Add memory section VAR_NO_INIT_PTR, CONST_UNSPECIFIED, CONFIG_DBTOC_DATA_UNSPECIFIED, CONFIG_DATA_UNSPECIFIED - Section 14.9.1 Multi-core / Multi Instantiation: remove section Multi-core / Multi Instantiation - Section 14.4 SPI Driver Component Header and Source File Description: update number of section containing information of SPI Folder Structure in Module Overview to 3.3.8.3 15. WDG - Update format of font, table, figure, words, line & spacing and chapter content. 16. THS: - Section 16.3 Architecture Details: Update the description for "Get current temperature inside the LSI" - Section 16.8. Memory Organization: Change section name CONFIG_DATA to CONFIG_DATA_UNSPECIFIED, VAR_NO_INIT to VAR_NO_INIT_PTR 17. IPMMU - Section “17.3. Architecture Details”: Add CddIpmmu_MmuSetTransTableBase into “Address translation functionality”. - Section “17.8. Memory Organization”: Change section name

			VAR_INIT_BOOLEAN to VAR_INIT_1 18. EMM Section 18.3. Architecture Details: Add information “Support external request control functionality” Section 18.8. Memory Organization: Change name of VAR_INIT_BOOLEAN to VAR_INIT_1 19. RFSO - Section 19.4 Update references from section 3.4.2 to section 3.4.2.3 - Section 19.8 Memory Organization: Change CDDRFSO_CODE_SLOW, CDDRFSO_CODE_FAST , CDDRFSO_CONFIG_DATA_32 , CDDRFSO_VAR_CLEARED_32 , CDDRFSO_VAR_INIT_BOOLEAN, CDDRFSO_VAR_NO_INIT_PTR to CODE_SLOW, CODE_FAST , CONFIG_DATA_32 , VAR_CLEARED_32 , VAR_INIT_1, VAR_NO_INIT_PTR 20. CRC - Section 20.8: + Change PRIVATE_CODE_ROM to CDDCRC_PRIVATE_CODE_ROM + Change PUBLIC_CODE_ROM to CDDCRC_PUBLIC_CODE_ROM
0.15	Feb 23, 2023	-	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references version at Table 2.1 Reference Documents(1/2). 5. CAN - Update version of Hardware User Manual to R19UH0186EJ0070 at section 5.2.1 General. - Section 5.2.2 Preconditions: Remove the information about interrupt mode when transmission is completed. 6.DIO - Section 6.8 Memory Organization: change section name from PUBLIC_CODE to DIO_PUBLIC_CODE_ROM, PRIVATE_CODE to DIO_PRIVATE_CODE_ROM, CONFIG_DATA_UNSPECIFIED to DIO_CFG_DATA_UNSPECIFIED 7. PORT - Section 7.2.2 Add information about the configuration should only activate for one single pin to one given peripheral input function. 8. ETH: - Section 8.2.2 Add information about functions CR7_Invalidate_DCache_By_Addr, CR7_Flush_DCache_By_Addr - Section 8.10.4: remove information S4_CR52, S4_G4MH in this device. - Section 8.8 Memory Organization: updated memory section VAR_PORT_BUFFER_0, VAR_PORT_BUFFER_1, VAR_PORT_BUFFER_2, add descriptions for new memory sections: VAR_PORT_DESCRIPTOR_0, VAR_PORT_DESCRIPTOR_1, VAR_PORT_DESCRIPTOR_2. 9.FLS: - Section 9.2.1 General and Section 9.2.2 Preconditions: Remove “not support” from information related to features that supported in module function. 11. ICCOM: - Section 11.2.2 Preconditions: Update hardware filename from “r19uh0186ej0055-rcarv4h.pdf” to “r19uh0186ej0070-r-carv4h.pdf” in Table 11-1 12.MCU - Section 12.2.1 General : Remove note to describe using parameters in McuPllClockSetting container

			13. IIC - Section 13.2.2: in Table 13-1 replace r19uh0186ej0055-r-carv4h.pdf to r19uh0186ej0070-r-carv4h.pdf - Section 13.5.2.10 update for Elements pNotification, pEnterRegProtect, pExitRegProtect, pEnterGlbProtect, pExitGlbProtect - Section 13.5.2.16 update for all element in type CddIic_DmaConfigType - Section 13.5.2.18: + Replace ucChannelSlaveMapping to ucFisrtBitSetupCycle and add element ucSclClkGenDiv. + Rename for ucSlaveAddress to usSlaveAddress and update its type. 14. SPI - Section 14.8: Correct memory section name as SPI_PUBLIC_CODE_ROM SPI_PRIVATE_CODE_ROM CODE FAST VAR_INIT_1 VAR_NO_INIT_32 VAR_NO_INIT_UNSPECIFIED - Section 14: Update format of tables and characters 15. WDG - Section 15.8 Memory Organization In table 15-8 ROM Sections of the WDG Driver: + Correct name of ROM section from PUBLIC_CODE, PRIVATE_CODE, DBTOC_DATA_UNSPECIFIED to WDG_PUBLIC_CODE_ROM, WDG_PRIVATE_CODE_ROM, CONFIG_DBTOC_DATA_UNSPECIFIED + Add ROM section CONST_UNSPECIFIED In table 15-9 RAM Sections of the WDG Driver: + Remove RAM section NOINIT_RAM_16BIT + Add RAM section VAR_NO_INIT_PTR 17. IPMMU - Section 17.2.2: Add precondition to notify user about the configuration of parameter "CddIpmmuIRDomainSupport" before using IR domain - Remove section "17.5.2.27. CddIpmmu_IMSSTR" 18. EMM Section 18.1 Overview: - Add new information for supporting feature Control External Error Request Section 18.3 Architecture Detail: - Add new information for Support External Error Request Control Functionality Section 18.5.2 Type Definition: - Add CddEmm_ControlCounterType, CddEmm_SelectCounterType Section 18.5.3 Function Definitions: - Add function CddEmm_SetHoldMaskCounter and CddEmm_SupportControlExternalErrorRequest Section 18.5.3.1 CddEmm_ReadErrorStatus - Add ECMERRSTSR22 register Section 18.5.4 Preemption of APIs: - Add function CddEmm_SetHoldMaskCounter and CddEmm_SupportControlExternalErrorRequest in table 18-23 Preemption of APIs of the EMM Driver Section 18.6.1 EMM Complex Device Driver Component Development Errors: - Add new DET error CDDEMM_E_COUNTER_MODE in table 18-24 DET

			Errors of EMM Complex Device Driver Component Section 18.6.2 EMM Complex Device Driver Component Production Errors: - Update number of domains to (n=0..13, 16..36, 38..42) 19. RFSO - Section 19.5.3 Update Range of IntervalTimerTime in CddRfso_IntervalTimeConfigure, IntervalTimerCycle in CddRfso_IntervalCycleConfigure
0.14	Jan 30, 2023	-	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references version at Table 2.1 Reference Documents(1/2). 5. CAN - Section 5.2.2 Preconditions: Add the information about interrupt mode. - Update the name of the V4H Hardware User Manual at section 5.2.1 General. 7. PORT: Section 7.6.2 PORT Driver Component Production Errors: - Add information of “Source of Error” for PORT_E_FUSE_MONITORING_FAILURE, PORT_E_UNINTENDED_MODULE_STOP_FAILURE 8. ETH: - Section 8.2 Forethoughts: Remove Throughput Measurement List information. 11. ICCOM: - Section 11.2.2 Preconditions: Update hardware filename from “[R19UH0179EJ0045]R-Car V4H Series User’s Manual_Global.pdf” to “r19uh0186ej0055-rcarv4h.pdf” in Table 11-1 12.MCU - Section 12.2.1 General : Add note to describe using parameters in McuPllClockSetting container 13.IIC - Section 13.2.2: in Table 13-1 replace from [R19UH0186EJ0050]R-Car V4H Series User’s Manual_Global.pdf to r19uh0186ej0055-r-carv4h.pdf 16. THS - Remove parameters CddThsPtat1, CddThsPtat2, CddThsPtat3, CddThsThcode1, CddThsThcode2, CddThsThcode3 from Configuration Dependency of API CddThs_ConfigureThermalInterruption, CddThs_GetCurrentTemperature, CddThs_GetCurrentVoltage in section 16.5.3,2, 16.5.3.3, 16.5.3.4

0.13	Dec 21, 2022	-	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references name and version for No.[1],version for [2] at Table 2.1 Reference Documents(1/2). 5. CAN - Correct the information about Transmit history interrupt occurs when 24 messages are stored in transmit history buffer at section 5.2.1 General. 7. PORT: Section 7.2.2 Preconditions: - Add information to notify user about the using available port pins in RCAR V4H 8. ETH + Section 8.5.3.1 Renesas Original API: Update Cross-Reference for Table 8.17 to Table 8.19. + Section 8.2.3 Data Consistency: Update format for table Table 8.6 ETH Driver Protected Resource List + Section 8.5.4 Preemption of APIs Table 8.20 Preemption Table of APIs of the ETH Driver (1/2), Table 8.21 Preemption Table of APIs of the ETH Driver (2/2) : Set “√” and “-“ for able/unable instead of “O” and “X”. + 8.6.2 ETH Driver Component Production Errors: Add add description before Table 8.24 DEM Errors of ETH Driver Component (1/2). 11. ICCOM - Section 11.2.1: Add new information about supporting Instance ID. 16. THS - Section 16.2.1: Add new information about supporting Instance ID. 17. IPMMU: - Section “17.2.1. General”: Add new information about supporting Instance ID - Change “Description” in + Section “17.5.2.13. CddIpmmu_PmbSettingType”, table 17-14 + Section “17.5.2.14. CddIpmmu_MicroTlbSettingType”, table 17-15 - Section “17.5.2.15. CddIpmmu_MmuModeType”, table 17-16: Change structure name and table name. - Section “17.5.3.14. CddIpmmu_MmuGetMemAttr”, table 17-49: Change Service ID of table from "0x0E" to "0x0F". 18. EMM - Section 18.2.1 General: Add information for supporting only one instance by configing CddInstanceId as 0. - Section 18.2.2 Preconditions: Update the information relating supported range of parameter CddEmmAddressToSaveErrorStatus. - Section 18.6.1: Update Table 1-20 DET Errors of EMM Complex Device Driver Component: + Add CddEmm_GetCurrentErrorCountUpValue into CDDEMM_E_UNINIT + Add CddEmm_SetPseudoErrorSignal, CddEmm_ClearPseudoErrorSignal into CDDEMM_E_PARAM_VALUE + Add CddEmm_ClearPseudoErrorSignal into CDDEMM_E_PSEUDO_ERROR_MODE 19. RFSO - Section 19.5.3: Update Service ID of API to match with service ID of APIs in source code 20. CRC - Section 20.2.1: Add new information about supporting Instance ID.

0.12	Nov 24, 2022	—	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references version No.[1], [2] at Table 2.1 Reference Documents(1/2). 5. CAN - Section 5.5.4. Preemption of APIs: Fixed remain TBD for the note Preemption table of APIs of the CAN Driver. - Re-entrant for different Controllers. Non Re-entrant for the same Controller. - If the state transition START to STOP has been started by Can_SetControllerMode, just clear error flags and do not perform further operations. - Section 5.2.1. General: Update description and mapping refer for Relation of “CanIntervalTimerPrescalerSet” parameter and “CanTxRxFIFOTransmissionInterval” parameter. - Section 5.2.1. General: Update information for "3 common FIFO, 4 Queue and 8 Rx FIFO can be configured for a controller on Max". - Section 5.2.1. General: Update information for "In interrupt mode, the message standard buffers" - Section 5.10.1.2. Det: Add Det_ReportRuntimeError - Section 5.10.1.4. Dem: Change interface Dem_ReportErrorStatus to Dem_SetEventStatus - Section 5.2.2: add a precondition information for CanHwFilter container’s usage. 6. DIO - Update format of font, table, figure, words, chapter content. 8.ETH - Section 8.2.3 Data Consistency, Table 8.6 ETH Driver Protected Resource List: Update ETH_AVBn_DATA_ISR to ETH_AVBnDATAISR - Section 8.2.5 ETH Driver Error List, Table 8.10 ETH Driver Error List (4/4): Changed name function ISR to ETH_AVBnDATAISR, ETH_AVBnERRISR, ETH_AVBnMACISR (n=0..2). 9.FLS: - Section 9.5.4 Preemption of APIs:: Remove “Not supported” in preemption table. 10. GPT: - Section 10.10.2 Callback function usage: Add the example for the callback function. 11. ICCOM - Section 11.2.3 Data Consistency: change the functions name from REGISTER_PROTECTION<n>, GLOBAL_PROTECT<n> to CDDICCOM_INTERRUPT_CONTROL_PROTECTION<n>, CDDICCOM_RAM_DATA_PROTECTION<n>. - Section 11.8 Memory Organization: + Figure 11.9: add CDDICCOM_CODE section. + ROM Section: Add CDDICCOM_CODE section to (X1). 12.MCU: Section 12.10.1 Stub Modules Handling: correct content for Det, Basic Software Scheduler and Dem. 13.IIC - Section 13.2.3.Data Consistency: Add description for critical section protection function. - Remove Section 13.2.4.User mode and supervisor mode, 13.2.5. Deviation

			List. - Remove section 13.10.2. Compiler, Linker And Assembler, 13.10.1.3. Translation header file, 13.10.1.4. Parameter Definition File, 13.4. Relationship with register, 13.5. Interaction Between The User And IIC Complex Device Driver Component. - Move section 13.6 to 13.4. - Move section 13.7 to section 13.5. - Move section 13.8 to section 13.6. - Add section 13.7 IIC Driver Component Runtime Errors. - Move section 13.9 to 13.8. - Move section 13.10 to 13.9. - Add new section 13.10 Non-AUTOSAR environment integration. - Update section 13.4. IIC Complex Device Driver Component Header And Source File Description: to refer to "R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Modules Overview". - Section 13.6.1: Update table 13.22 "DET Errors of IIC Complex Driver Component" to remove row "Origin", and add row "Value (hex)" for each error code. - Section 13.6.2: Update table 13.23 "DEM Errors of IIC Complex Driver Component" to remove row "Origin" for all error code. 14.SPI: - Section 14.2: Forethoughts + Section 14.2.1: Add usage note on new parameter SpiClk2CsCount 15. WDG - Section 15.2. Remove Deviation List. 16. THS - Section 16.2: Forethoughts + Remove section 16.2.4: User mode and supervisor mode + Remove section 16.2.5: Deviation List - Remove section 16.4: Relationship with register - Remove section 16.5: Interaction Between the User and THS Complex Device Driver Component - Add section 16.7: THS Driver Component Runtime Errors - Add section 16.10: Non-AUTOSAR environment integration - Section 16.2.3: + Update VARIABLE_PROTECTION, DRIVERSTATE_PROTECTION to CDDTHS_RAM_DATA_PROTECTION + Update REGISTER_PROTECTION to CDDTHS_INTERRUPT_CONTROL_PROTECTION 17. IPMMU: - Update font and format of table, figure and bulleted list. - Add information for FFI use cases in section “17.2.1 General”. - Remove section: + "17.2.4 User Mode and Supervisor Mode" + "17.2.5 Deviation List" + "17.2.6 Limitations" + "17.4 Relationship with Register" + "17.5. Interaction Between The User And IPMMU Complex Device Driver Component". - Change section:

			+ "17.6. IPMMU Complex Driver Component Header and Source File Description" to "17.4. IPMMU Complex Driver Component Header and Source File Description". + "17.7. Application Programming Interface" to "17.5. Application Programming Interface". + "17.8. Development, Production and Runtime Errors" to "17.6. Development, Production and Runtime Errors". + "17.9. Memory Organization" to "17.8. Memory Organization". + "17.10. Device Specific Information" to "17.9. Device Specific Information". - Section "17.2.3. Data Consistency": change exclusive area from "REGISTER_PROTECTION" to “CDDIPMMU_INTERRUPT_CONTROL_PROTECTION”. - Section "17.5.4. Preemption of APIs": change the explanation "x" and "o" to "√" and "-" respectively. - Section "17.8 Memory Organization" change memory section name from CDDIPMMU_CODE_SLOW, CDDIPMMU_CODE_FAST, CDDIPMMU_VAR_INIT_BOOLEAN, CDDIPMMU_VAR_NO_INIT_PTR, CDDIPMMU_CONFIG_DATA_32 and CDDIPMMU_CONFIG_DATA_UNSPECIFIED to CODE_SLOW, CODE_FAST, VAR_INIT_BOOLEAN, VAR_NO_INIT_PTR, CONFIG_DATA_32 and CONFIG_DATA_UNSPECIFIED respectively. - Add section: 17.5.2.16 CddIpmmu_IMCTRn to 17.5.2.33 CddIpmmu_IMUCTRn - Add section: "17.10 Non-AUTOSAR environment integration". 18. EMM: - Update format of font, table, figure, words, line & paragraph spacing and chapter content. - Remove below sections: + 18.2.4 User Mode and Supervisor Mode + 18.2.5 Deviation List (moved to Generic/Appendix) + 18.4 Relationship with Register + 18.5 Interaction Between the User and EMM Complex Device Driver Component + 18.10.1.3 Translation Header File + 18.10.1.4 Parameter Definition File + 18.10.2 Compiler, Linker, and Assembler - Move section below: + Section "18.6 EMM Complex Device Driver Component Header And Source File Description to 18.4 + Section '18.7 Application Programming Interface' to 18.5 + Section '18.8 Development and Production Errors' to 18.6 + Section '18.9 Memory Organization' to 18.8 - Add new section '18.7 EMM Complex Driver Component Runtime Errors - Add new section 18.10. Non-AUTOSAR environment integration - In section 18.1. Overview Update in formation of reference documnent for Sample Application - In section 18.2.1. General Update information of reference documnent for ISR Functions and Device Specific Information. - In section 18.2.3: Data Consistency + Update exclusive area from REGISTER_PROTECTION, GLOBALVAR_PROTECTION to only

			CDDEMM_INTERRUPT_CONTROL_PROTECTION. - In section 18.8 Memory Organization: Replace CDDEMM_CODE_FAST, CDDEMM_CODE_SLOW, CDDEMM_CONFIG_DATA_32, CDDEMM_VAR_CLEARED_32, CDDEMM_VAR_NO_INIT_PTR by CODE_FAST, CODE_SLOW, CONFIG_DATA_32, VAR_CLEARED_32, VAR_NO_INIT_PTR. 19. RFSO: - Section 19.2: Forethoughts + Remove section 19.2.4: User mode and supervisor mode + Remove section 19.2.5: Deviation List - Section 19.2.3: Data Consistency + Update exclusive area to CDDRFSO_RAM_DATA_PROTECTION and CDDRFSO_INTERRUPT_CONTROL_PROTECTION - Section 19.10.1.3: Basic Software Scheduler + Correct SchM stub files folder as “rel\common\generic\stubs\<Autosar version>\Rte” + Update "SchM module is enabled when CDDRFSO_CRITICAL_SECTION_PROTECTION parameter is configured as STD_ON" to “SchM module is enabled when CddRfsoCriticalSectionProtection parameter is configured as true” + Update SchM functions with exclusive areas as below: void SchM_Enter_CddRfso_CDDRFSO_RAM_DATA_PROTECTION (void); void SchM_Exit_CddRfso_CDDRFSO_RAM_DATA_PROTECTION (void); void SchM_Enter_CddRfso_CDDRFSO_INTERRUPT_CONTROL_PROTECTION (void); void SchM_Exit_CddRfso_CDDRFSO_INTERRUPT_CONTROL_PROTECTION (void); 20. CRC - Section 20.8: + Table 20-36: add CONFIG_DATA_UNSPECIFIED, CONST_UNSPECIFIED, CONFIG_DBTOC_DATA_UNSPECIFIED + Table 20-37: remove CONFIG_DATA_UNSPECIFIED, CONST_UNSPECIFIED - Section 20.10.2: + Add an example for the callback function
0.11	Oct 24, 2022	—	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references version No.[1], [2] at Table 2.1 Reference Documents(1/2). 6.DIO - 6.8 Memory Organization: + ROM: Remove CONST_UNSPECIFIED, CODE_SLOW, CONST_32 Section + RAM: Add DIO_CFG_RAM_UNSPECIFIED Section 7. PORT: - Section 7.2.2 Precondition: + Add 3 information for direction, API Port_SetPinMode and API Port_SetToAlternateMode. - Section 7.8 Memory Organization:

			+ Change “NOINIT_RAM_PTR” to “VAR_NO_INIT_PTR”. - Section 7.5.2.4 Port_PinModeType: + Add new mode "ALT4". - Section 7.5.3.1.4 Port_FUSEMonitoring: + Update Reentrancy field from "Non-Reentrant" to "Reentrant". 8.ETH - Section 8.2.2 Preconditions: Update information for the transmission time stamp. - Section 8.2.3 Data Consistency, Table 8.6 ETH Driver Protected Resource List: Changed "ETH_AVBn_CH00_ISR to ETH_AVBn_CH24_ISR, with n equal 0..2" to "ETH_AVBn_DATA_ISR with n equal 0..2". - Section 8.2.5 ETH Driver Error List, Table 8.10 ETH Driver Error List (4/4): Changed "ETH_AVBn_CH00_ISR to ETH_AVBn_CH24_ISR, with n equal 0..2" to "ETH_AVBn_DATA_ISR, ETH_AVBn_ERROR_ISR, ETH_AVBn_MAC_ISR with n equal 0..2". Changed "ETH_AVBn_CH22_ISR, with n equal 0..2" to "ETH_AVBn_ERROR_ISR, with n equal 0..2" - Section 8.5.3 Function Definitions, Table 8.18 Eth_SetIncrementTimeForGptp: Removed ETH_E_INV_PARAM from DEM/DET error handled - Section 8.6.2 ETH Driver Component Production Errors, Table 8.25 DEM Errors of ETH Driver Component (2/2): Changed "ETH_AVBn_CH00_ISR -> ETH_AVBn_CH24_ISR, ETH_AVBn_CH00_ISR_CAT2 -> ETH_AVBn_CH24_ISR_CAT2 n=0..2" to "ETH_AVBn_DATA_ISR, ETH_AVBn_ERROR_ISR, ETH_AVBn_MAC_ISR, ETH_AVBn_DATA_ISR_CAT2, ETH_AVBn_ERROR_ISR_CAT2, ETH_AVBn_MAC_ISR_CAT2 (n =0..2)". Changed "ETH_AVBn_CH22_ISR_CAT2 n=0..2" to "ETH_AVBn_ERROR_ISR, ETH_AVBn_ERROR_ISR_CAT2 (n =0..2)" - Section 8.9.1.2 ISR Functions, Table 8.29 ISR Function for the Device: Removed ETH_AVBn_CH00_ISR to ETH_AVBn_CH24_ISR, ETH_AVBn_CH00_ISR_CAT2 to ETH_AVBn_CH24_ISR_CAT2 (n=0..2). Added ETH_AVBn_DATA_ISR, ETH_AVBn_ERROR_ISR, ETH_AVBn_MAC_ISR, ETH_AVBn_DATA_ISR_CAT2, ETH_AVBn_ERROR_ISR_CAT2, ETH_AVBn_MAC_ISR_CAT2 (n =0..2) 9. FLS: - Update name of critical section in 9.11.1.3 Basic Software Scheduler, 9.2.3 Data Consistency. - Remove ”not supported” in 9.2.3 Data Consistency. 12.MCU: - Section 12.8: Add new section name VAR_INIT_32, VAR_INIT_UNSPECIFIED, CONFIG_DATA_8 and CONFIG_DBTOC_DATA_UNSPECIFIED. Remove section name VAR_INIT_8 and CONST_32. 13. IIC: - Section 13.7.2.6. CddIic_BufferAddressType: Update memclass of pSndBuffer,pRcvBuffer to TYPEDEF. - Section 13.7.2.10. CddIic_ChannelConfigType: + Update memclass of pICMCRnReg, pICMSRnReg, pICMIERnReg, pICMARnReg, pICTXRXDnReg, pICSCRnReg , pICSSRnReg, pICSIERnReg, pICSARnReg, pICCCRnReg, pICCCR2nReg, pICMPRnReg, pICHPRnReg, pICLPRnReg, pICDMAERnReg, pICFBSCRnReg, pDmaConfiguration to TYPEDEF. - Section 13.7.2.15. CddIic_SlaveInterfaceType: Update memclass of pTxBuff,

			pRxBuff to TYPEDEF. - Section 13.7.2.16. CddIic_DmaConfigType: Update memclass of pMasterTxDmaBaseAdress, pMasterRxDmaBaseAdress, pSlaveTxDmaBaseAdress, pSlaveRxDmaBaseAdress to TYPEDEF. - Section 13.7.3.1. CddIic_Ch<n>Write, 13.7.3.3. CddIic_Ch<n>WriteRead: Update Possible Return Values to RTE. - Section 13.9. Memory Organization: Add RAM section Y3 CDDIIC_VAR_NO_INIT_PTR. 16. THS: - Change VAR_NO_INIT to VAR_NO_INIT_PTR in section 16.9 17. IPMMU: - Section "17.9. Memory Organization": Change CDDIPMMU_VAR_CLEARED_32 to CDDIPMMU_VAR_NO_INIT_PTR. - Section "17.7.2.7. CddIpmmu_ConfigType" and "17.7.2.8. CddIpmmu_MmuConfigType": Change CDDIPMMU_VAR_INIT to TYPEDEF. - In 17.2.1 General, remove the information "timing details and memory consumption ". 18. EMM: - Section "18.9. Memory Organization": Add RAM section Y3 CDDEMM_VAR_NO_INIT_PTR. 19. RFSO: - Section 19.5.2 Type Definitions + Add compiler abstraction macro VAR for parameters in definition types CddRfso_TOESStatus, CddRfso_CFEStatusType, CddRfso_ConfigType, CddRfso_ChannelConfigType, CddRfso_ChannnelStatusType - Section 19.5.2.3 CddRfso_ConfigType: + Update memclass of pChannelConfig from CDDRFSO_VAR_INIT to TYPEDEF - Section 19.5.2.4 CddRfso_ChannelConfigType: + Update name from CddRfso_TimeOutModeType to CddRfso_ChannelConfigType - Section 19.5.2.5 CddRfso_ChannelStatusType: + Update name of CddRfso_ChannelConfigType to CddRfso_ChannelStatusType + Update ucChannelId to blChannelIsBusy - Section 19.5.2.7 CddRfso_CycleType: + Update name from CddRfso_ReturnType to CddRfso_CycleType - Section 19.5.2.9 CddRfso_IntervalCbkFuncPtrType + Update “Function pointer” to “CddRfso_IntervalCbkFuncPtrType” - Section 19.6.1 RFSO Complex Device Driver Component Development Errors + Update “No.” to “SI. No.” - Section 19.8 Memory Organization + Add memory section CDDRFSO_VAR_NO_INIT_PTR 20. CRC: - Section 20.2.2: Preconditions: add usage note for API CddCrc_Command.
0.10	Sep 23, 2022	—	1.Introduction: Update Release Version and Modules version in Table 1.1 2.Reference Documents Update references version No.[1], [2] at Table 2.1 Reference Documents(1/2). 5.CAN: - Remove Can_CheckWakeup, CanIf_CheckWakeup, EcuM_CheckWakeup, Can_MainFunction_Wakeup, Wakeup processing at Section 5.3 Architecture Details in Figure 5.3 Driver Architecture - Remove Can_MainFunction_Wakeup, Can_CheckWakeup APIs out of “Table 5-14 Preemption table of APIs of the CAN Driver”, “Table 5-11 API provided by

			the CAN Driver Component” and “Table 5-15 DET Errors of CAN Driver Component”. - Remove Can_MainFunction_Wakeup API out of section 5.10.3 Scheduled function usage. 11. ICCOM: - Section 11.5.2.9. CddIccom_ChannelConfigType: Update memclass for elements pCtaRxUpper and pCtaRxBottom. - Section 11.8. Memory Organization: Add memory section CDDICCOM_VAR_NO_INIT_PTR, CDDICCOM_VAR_INIT_32 and remove memory section CDDICCOM_VAR_INIT_BOOLEAN. 12.MCU: - Section 12.2.1: Add new description about the ZB3 clock range. 13. IIC: - Update memory type for variable from AUTOMATIC to TYPEDEF in section: + 13.7.2.12. CddIic_WriteType: ulWriteNumber + 13.7.2.13. CddIic_ReadType: ulReadNumber 15.WDG: - Section 15.2.5. Add note for WDG module uses the resource of GPT module 16. THS: - Remove section 16.7.1.2. Os Types 17. IPMMU: - Section 17.2.2. Preconditions: Add preconditions to prevent conflict in IPMMU operation. - Section 17.2.3. Data Consistency: Add information about data consistency of IPMMU. 18. EMM: - Section 18.2.2 Preconditions: + Add one new precondition for user when clearing pseudo error status. 19. RFSO: - Section 19.4: Remove Section 19.4 Relationship with register - Section 19.5: Remove Section 19.5 Interaction Between The User And RFSO Complex Device Driver Component - Section 19.10: Device Specific Information + Section 19.10.1.2: Remove section 19.10.1.2 Translation header file + Section 19.10.1.3: Remove section 19.10.1.3 Parameter Definition File + Section 19.10.1.4: Remove section 19.10.1.4 Channel using recommendation + Section 19.10.1.1: Move section 19.10.1.1 ISR Functions to section 19.10.1.2 + Add new section 19.10.1.1 Channel Mapping + Section 19.10.2: Remove section 19.10.2 Compiler, Linker And Assembler + Section 19.10.3: Remove section 19.10.3 Memory Usage And Execution Time for R-Car V4H - Move section 19.6 RFSO Complex Component Header And Source File Description to section 19.4 - Move section 19.7 Application Programming Interface to section 19.5 - Move section 19.8 Development And Production Errors to section 19.6 - Add new section 19.7 RFSO Driver Component Runtime Errors - Move section 19.9 Memory Organization to section 19.8 - Move section 19.10 Device Specific Information to section 19.9 - Add new section 19.10 Non-AUTOSAR environment integration - Section 19.5.3 Function Definition + Section 19.5.3.4 CddRfso_StartIntervalTimer: update Reentrancy property + Section 19.5.3.5 CddRfso_StopIntervalTimer: update Reentrancy property + Section 19.5.3.8 CddRfso_StartTimeoutTimer: update Reentrancy property + Section 19.5.3.9 CddRfso_StopTimeoutTimer: update Reentrancy property + Section 19.5.3.16 CddRfso_CtrlIntervalTimerInterrupt: update Reentrancy property

			- Section 19.6.1 : fix typo at No. 6 20. CRC: - Update requirement for pointers passed to CddCrc_SetupEB and CddCrc_Write in 20.2.2 Precondition. - Add CrcInDataPtr and CrcOutDataPtr in 20.5.2 Type Definitions. - Update description for output CRC codes generated by CRC module in 20.2.1 General
0.09	Aug 26, 2022	—	1.Introduction: Update Release Version and Modules version in Table 1.1 Update references version No.[1], [2], [3] at Table 2.1 Reference Documents(1/2). 5.CAN: - Add information related to pretended network at Section 5.2.1 - Add precondition for pretended network Section 5.2.2 - Add Can_RAMTest to “Figure 5.3 Driver Architecture” at section 5.3 - Add Can_SetIcomConfiguration to "Figure 5.4" at Section 5.3: - Add new type definition Can_RamTestWalkType and Can_RamTestFillType to support for Can_RAMTest at Section 5.5.2 - Add new API Can_SetIcomConfiguration at Section 5.5.3: - Add Can_RAMTest in table 5-11 API provided by the CAN Driver Component - Add Can_RAMTest to section 5.5.4 Preemption of APIs and section 5.5.3.1 Renesas Original API - Change name of variable: Test_Transition to TestTransition in Can_SelfTestChannel function at section 5.5.3.1 Renesas Original API - Add new error check for Can_RAMTest at section 5.6.1 - Add new RAM section VAR_NO_INIT_8 at section 5.8 Memory Organization. - Update information of CAN Global interrupt at table “ISR Handler Addresses” in section “ISR functions”. 6.DIO - Remove section 6.9.3 Memory Usage and Execution Time 7. PORT: - Section 7.8 Memory Organization: Add VAR_NO_INIT_32 - Remove section 7.9.2 Memory Usage and Execution Time for V4H 8.ETH: - Section 8.2.5 ETH Driver Error List: Remove ETH_E_TIMERINC_FAILED, ETH_E_TIMEROFFSET_FAILED - Section 8.3 Architecture Details Remove "Not Support" for timestamp/ adjust gPTP Timer. - Section 8.5.3 Function Definition: + Remove "Not supported" in Table 8.15 + Remove ETH_E_TIMERINC_FAILED, update range for IncVal in table 8.18 Eth_SetIncrementTimeForGptp. + Remove ETH_E_TIMEROFFSET_FAILED in table 8.19 Eth_SetOffsetTimeForGptp. - Section 8.6.2 ETH Driver Component Production Errors: Remove ETH_E_TIMERINC_FAILED, ETH_E_TIMEROFFSET_FAILED in table 8.25 DEM Errors of ETH Driver Component (2/2). - Update section 8.2.2 Preconditions for timestamp - Added information when access gPTP:in section 8.2.2 Precondition: “The gPTP module only access in secure mode. Therefore, CR52 must change to Secure mode before initialize Ethernet Driver when EthGlobalTimeSupport parameter configured true.”

			9.FLS: - Section 9.2.1: + Add description for new configuration parameters FlsSerialFlashDDRPattern, FlsSerialFlashDDRVerify. + Add description for new APIs Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate - Section 9.2.2: Add preconditions for Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate - Section 9.3: Update Figure 9.4 to add Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate and related description for these functions - Section 9.5.2: Add type definition for Fls_SfSpecificConfigType, Fls_SpecificConfigType, Fls_HfSpecificConfigType to new section 9.5.2.4, 9.5.2.5, 9.5.2.6. - Section 9.5.3: Add Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate APIs to table 9.6 and section 9.5.3.1 Renesas Original API. - Section 9.5.4: Add Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate APIs to table “Preemption Table of APIs of the FLS Driver”. - Section 9.6.1, 9.6.2, 9.7: Add Fls_SendSpecificConfig, Fls_DDRWritePattern, Fls_DDRVerifyPattern, Fls_DDRCalibrate APIs to related DET/DEM errors. - Remove section 9.10.1.3 Execution Time Details 12.MCU: - Section 12.2.1: Add new description of High Performance mode. - Section 12.2.6: Remove the description of Error Control Module(ECM) includes the WDT Error Interrupt. - Remove section 12.9.2 Memory Usage and Execution Time for V4H 13. IIC - Section 13.7.2.2 IicMessageType: Add IIC_NOTICE_IDLE in table 13.13. - Section 13.7.2.5 CddIic_ChannelStatusType: Add variable enSlaveAddressMode, enDmaStatus, enRepeatStartSta in table 13.16. - Section 13.7.2.9 CddIic_OperationInterfaceType: Update the range in table 13.20. - Section 13.7.2.9 CddIic_OperationInterfaceType, Section 13.7.2.16 CddIic_DmaConfigType, Section 13.7.2.17 CddIic_DmaModeType: Correct the description in table 13.20, 13.27, 13.28. - Section 13.7.2.10 CddIic_ChannelConfigType: + Add variables: pEnterGlbProtect, pExitGlbProtect. + Update variables pICMCRnReg, pICMSRnReg, pICMIERnReg, pICMARnReg to remove volatile. - Section 13.7.3.3 update Function prototype of CddIic_ChnWriteRead. - Section 13.7.3.4 update type of Parameter In for parameter LucSlaveOwnAdress, LpSlaveInterface and Range of LucSlaveOwnAdress, update for Function prototype in function CddIic_ChnSlaveInit. - Section 13.7.4 Preemption of APIs: + Update table 13.31 for function CddIic_Ch<n>Write, CddIic_Ch<n>Read, CddIic_Ch<n>WriteRead. - Section 13.10.1.2 ISR Functions: + Add CDDIIC_DMA<m>_ISR, CDDIIC_DMA<m>_CAT2_ISR to table 13.35 ISR Handler Addresses. - Remove Section 13.10.3 Memory Usage and Execution Time for V4H. 14.SPI: - Section 14.6.2 SPI Driver Component Production Errors: Add SPI_E_INTERRUPT_CONTROLLER_FAILURE and

			SPI_E_WRITE_VERIFY_FAILURE into the “DEM Errors of SPI Driver Component” Table 14.23. - Remove section 14.9.3 Memory Usage and Execution Time for V4H. 15.WDG: - Remove section 15.9.3. Remove Memory Usage and Execution Time for V4H 16. THS - Remove section 16.10.3 Memory Usage and Execution Time for V4H 17. IPMMU: - Remove section “17.10.3. Memory Usage And Execution Time for R-Car V4H” 18. EMM: - Remove section 18.10.3 Memory Usage And Execution Time for R-Car V4H 19. RFSO - Section 19.7.3.13: Update description of support for API CddRfso_ExternalPinControl. 20. CRC: - Remove section “20.9.2. Memory Usage And Execution Time for R-Car V4H”
0.08	July 28, 2022	—	1.Introduction: Update Release Version and Modules version in Table 1.1 Update references version No.[1], [2] at Table 2.1 Reference Documents(1/2) 5.CAN: - Add a new data type definition “Can_SelfTestType” at section “Type Definitions Table 5.8 Can_SelfTestType - Support new Renesas API Can_SelfTestChannel() at section “Function Definitions”, “ Preemptions of API” and “Architecture Details” from Table 5.9 to 5.11 and Figure 1.4 Driver Architecture - Add Can_SelfTestChannel() to related API field at DET/DEM errors “CAN_E_PARAM_CONTROLLER”, “CAN_E_UNINIT”, “CAN_E_TRANSITION” and “CAN_E_TIMEOUT_FAILURE” from Table 5.12 to 5.14. - Add a new DEM error “CAN_E_INTERRUPT_CONTROLLER_FAILURE” to support Safety mechanism Unintended Interrupt check at section “CAN Driver Component Production Errors” Table 5.14. - Remove DEM error “CAN_E_INT_INCONSISTENT” at section “CAN Driver Component Production Errors” Table 5.14. - Add error CAN_E_DATALOST at section “CAN Driver Component Runtime Errors” Table 5.15. - Add section 5.7 CAN Driver Component Runtime Errors. 6.DIO: Remove safety function: + Section 6.5.3.1.3, 6.5.4, 6.6.2: remove Dio_UnintendedModuleStopCheck. + Section 6.2.2 Update Precondition for Exclusive Control. + Table 6.13: remove Dio_UntendedModuleStopCheckApi. 7. PORT: - Add section 7.5.3.1.5 Port_UnintendedModuleStopCheck. - Sections 7.5.4 Preemption of APIs: + Add Port_UnintendedModuleStopCheck. - Sections 7.6.2 PORT Driver Component Production Errors: + Add Port_UnintendedModuleStopCheck. + Add DEM error PORT_E_UNINTENDED_MODULE_STOP_FAILURE - Section 7.6.1 PORT Driver Component Development Errors: + Add Port_SetToAlternateMode, Port_SetToDioMode, Port_SetPinMode, Port_UnintendedModuleStopCheck.

			- Section 7.3 Architecture Details: + Add new feature Port Unintended Module Stop Check. 8.ETH - Update Section 8.3 Architecture Details: Removed “Not Supported” for feature Getting Dropped Packets, IEEE1722 packet filtering functionality. - Update table 8.25 DEM Errors of ETH Driver Component (2/2) : Removed Error Code of SI.No.8 from ETH_E_INT_INCONSISTENT . Add new ETH_E_INTERRUPT_CONTROLLER_FAILURE - Section 8.2.4Table 8.10 ETH Driver Error List (4/4): Removed ETH_E_INT_INCONSISTENT. Added new ETH_E_INTERRUPT_CONTROLLER_FAILURE - Section 8.5.3 Table 8.15 API Provided by ETH Driver Component: Removed "Not Supported" for Eth_GetTxErrorCounterValues - Section 8.5.4 Table 8.20: Preemption of APIs: Removed "Not Supported" for Eth_GetTxErrorCounterValues 9.FLS: - Add Fls_Suspend/Fls_Resume in Figure 9.5, Table 9.12, Table 9.15 to Table 9.19. - Add the information of Fls_Resume/Fls_Suspend in section 9.2.1 General. 13. IIC - Section 13.4.1 In Table 13.3: Replace Cddlic_Dma<n>ISR by CDDIIC_DMA<n>_ISR. - Section 13.4.2: + In Table 13.4, Table 13.6: Replace Cddlic_Dma<n>ISR by CDDIIC_DMA<n>_ISR. + In Table 13.10: Replace DEM error CDDIIC_E_UNINTENDED_INTERRUPT_CHECK by CDDIIC_E_INTERRUPT_CONTROLLER_FAILURE. Replace Cddlic_Dma<n>ISR by CDDIIC_DMA<n>_ISR. Add IIC_HW_CH<n>_ISR to column "Related API/ISR name" of Configuration Parameter CDDIIC_E_INTERRUPT_CONTROLLER_FAILURE. - Section 13.8.2 In Table 13.33: + Replace DEM error CDDIIC_E_UNINTENDED_INTERRUPT_CHECK by CDDIIC_E_INTERRUPT_CONTROLLER_FAILURE. + Add CDDIIC_DMA<n>_ISR to field "Related API(s)" of error code CDDIIC_E_INTERRUPT_CONTROLLER_FAILURE. 14.SPI: - Remove “not supported” for DMA transmission feature of V4H. - Add note for usage with DMA. 17. IPMMU: - Change Dem_ReportErrorStatus to Dem_SetEventStatus in 17.2.1. General and 17.8.2 IPMMU Complex Driver Component Production Errors. - Change CDDIPMMU_E_WRITE_VERIFY to CDDIPMMU_E_WRITE_VERIFY_FAILURE in Table 17.8 (DEM Errors of IPMMU Complex Driver Component) of 17.8.2. - Change the name of the parameter CddIpmmuRegisterWriteVerify to CddIpmmuWriteVerifyCheck in table 17.4 (Relationship with configuration parameter). - Correct the value of CddIpmmu_ErrorCodeType in section 17.7.2.4. CddIpmmu_ErrorCodeType. - Correct type of ucNumberOfTlbConfig in section 17.7.2.7. CddIpmmu_ConfigType. - Add related register IMPCTR for API CDDIPMMU_ERROR_ISR in table Table

			17.3: Relationship between API/ISR and related registers (2/2). - Add related registers for parameter CddIpmmuWriteVerifyCheck and CddIpmmuUnintendedInterruptCheck in table 17.4 (Relationship with configuration parameter). 18. EMM: Section 18.2.2 Preconditions: - Add information for CDD EMM driver when all error signals are configured as false. Section 18.7.3.7 CddEmm_GetCurrentErrorCountUpValue: - Update the “Preconditions” of Table 18.20 CddEmm_GetCurrentErrorCountUpValue. 19. RFSO: - Section 19.4.2: add configuration parameters CddRfsoRegisterWriteVerify and CddRfsoUnintendedInterruptCheck to Table 1-3 20. CRC -Add Unintended Module Stop Check to 1.1 Overview section and Figure 1 3 Architecture Details section. -Add CddCrc_UnintendedModuleStopCheck to 1.5.3 Function Definitions, and 1.5.4 Preemption table. -Add new Dem errors CDDCRC_E_INTERRUPT_CONTROLLER_FAILURE, and CDDCRC_E_UNINTENDED_MODULE_STOP_FAILURE to 1.6.2 CRC Driver Component Production Errors. -In 1.5.2.9 CddCrc_ChannelConfigType, change prototype of function pointer pCompareEndNotification from pass-by-reference to pass-by-value. -Add 1.5.2.16 CddCrc_CompareResultType. -In 1.5.2.6 CddCrc_WcrcChannelConfigType, + Add element pECMEN. + Remove unused elements ucCompareFunction, ucECMErrorOutput, ucWcrcCommonInterrupt, ucCompletedComparingModeInterrupt, ucErrorCompareOutputInterrupt
0.07	Jul 27, 2022	—	1.Introduction: Update Release Version and Modules version in Table 1.1 Update references version No.[1], [2] at Table 2.1 Reference Documents(1/2) 5.CAN: -Add bellow preconditions at section “Preconditions”: + A precondition to mention Port pin GP1_3 need to be configured when using CAN channel 0. + A precondition to mention that Can_EnableControllerInterrupts and Can_DisableControllerInterrupts APIs should not be invoked immediately after invoking Can_Write and message is not confirmed to user. 6.DIO: Support Exclusive Control feature: - Section 6.5.2: Add Section 6.5.2.11. Dio_ExclusiveType and Table 6.12. Dio_ExclusiveType - Section 6.6.2: Add DEM error DIO_E_GET_CONTROL_FAILURE 7. PORT: - Add section 7.5.3.1.4 Port_FUSEMonitoring. - Sections 7.5.4 Preemption of APIs: + Add Port_FUSEMonitoring - Sections 7.6.2 PORT Driver Component Production Errors: + Add Port_FUSEMonitoring

			+ Add DEM error PORT_E_FUSE_MONITORING_FAILURE to. - Update Table 7.2 Port_ConfigType. 8.ETH - Updated the Description content to support the 1522 data size for OversizePkt in Table Dropped Packet List, RxStatsOversizePkts, RxStatsPkts1024to1518Octets in Table ETH Rx Statistics Counter List. - Update Section 8.3 Architecture Details: Removed “Not Supported” for feature Getting Ethernet Statistics. - Update table API Provided by ETH Driver Component in section 8.5.3 Function Definitions: Removed “Not Supported” for Eth_GetRxStats, Eth_GetTxStats. 10. GPT: - Add Table 10.10 DEM Errors Of GPT Driver Component for the error code GPT_E_INTERRUPT_CONTROLLER_FAILURE - Add the information of Dem_SetEventStatus() in Chapter 10.10.1.4 Dem 11. ICCOM: Section 11.2.2: - Change name of parameter "CddIccomRegisterWriteVerify" to "CddIccomWriteVerifyCheck" Section 11.6.2: - Change name of parameter "CDDICCOM_E_WRITE_VERIFY" to "CDDICCOM_E_WRITE_VERIFY_FAILURE" 13. IIC: - Update definition type of “ucSlaveID” in Table 13.29 CddIic_SlaveConfigType. 16. THS: - Table 16.3: Remove Elements pThermalInterruptionNotification - Figure 16.4: Add CddThs_ThermalChannel to Segment CONFIG_DATA_32 18. EMM: Section General: - Change name of Dem_ReportErrorStatus to Dem_SetEventStatus Section Relationship between configuration parameter and related registers and section Section EMM Complex Device Driver Component Production Errors: - Correct name of parameter CDDEMM_E_UNINTENDED_INTERRUPT_CHECK to CDDEMM_E_INTERRUPT_CONTROLLER_FAILURE. - Correct name of parameter CDDEMM_E_REG_WRITE_VERIFY to CDDEMM_E_WRITE_VERIFY_FAILURE - Correct name of parameter CddEmmRegisterWriteVerify to CddEmmWriteVerifyCheck 19. RFSO: - Chapter 19.3: + Unify font format of "FSO_CTL.CFEO_1 and FSO_CTL.CFEO_0 to 1" + In description of "Time-out Interrupt control", unify font format of "CddRfso_ClearTimeoutInterrupt()" + In "External output of RFSO and related bits control and examination", update "indicates logic sum made feedback value from ECM module of 11 channels" to "indicates feedback value from ECM module for each of 11 channels" - Chapter 19.6: unify format related to spacing line between paragraphs - Chapter 19.7.3.4 -> 19.7.3.10: Correct description for return value CDDRFSO_E_OK corresponding to the API. - Chapter 19.2.1: update "one time" to "once" - Chapter 19.2.2: Table 19-1: update "DBSC4" to "DBSC5" - Chapter 19.7.2.6, 19.7.3, 19.7.3.2, 19.7.3.6, 19.7.5: update "micro second" to "microsecond" - Chapter 19.7.5: update reference link of "19.7.3.5"

			- Chapter 19.2.1: updated reference link of “19.10” - Chapter 19.2.3: Remove (not supported) at title of this chapter
0.06	Jun 27, 2022	—	1.Introduction: Update Release Version and Modules version in Table 1.1 5.CAN: - Remove CAN_CONTROLLERm_RX_ISR, CAN_CONTROLLERm_RX_CAT2_ISR, CAN_RSCANn_RXFIFO_ISR(), CAN_RSCANn_RXFIFO_CAT2_ISR(),CAN_CONTROLLERm_BUSOFF_ISR(), CAN_CONTROLLERm_BUSOFF_CAT2_ISR() out of related APIs of DEM error CAN_E_INT_INCONSISTENT at section “CAN Driver Component Production Errors” - Replace CAN_RSCANn_RXFIFO_ISR(), CAN_RSCANn_RXFIFO_CAT2_ISR(), CAN_CONTROLLERm_RX_ISR() and CAN_CONTROLLERm_RX_CAT2_ISR() by CAN_GLOBAL_ISR(),CAN_GLOBAL_CAT2_ISR (), CAN_CHANNEL_ISR() and CAN_CHANNEL_CAT2_ISR() at related APIs field of DET error CAN_E_DATALOST at section “CAN Driver Component Development Errors”. - Add CAN_GLOBAL_ISR, CAN_GLOBAL_CAT2_ISR and update information of CAN Channel interrupt at table “ISR Handler Addresses” in section “ISR functions”. 6.DIO: 6.5.3: add Dio_UnintendedModuleStopCheckApi 6.5.3.1: add table explain API Dio_ UnintendedModuleStopCheck 6.6.2: add new table DEM Errors of DIO Driver Component to describe DEM API 6.8: add section CODE SLOW 7. PORT: - Supported API: Port_SetPinMode 8. ETH: - Updated Section Architecture Details: Removed “Not Supported” from Adding/ Removing Mac Address to/from the Rx Hardware Filter Table. - Updated table API Provided by ETH Driver Component for Eth_GetCounterValues and Eth_UpdatePhysAddrFilter. - Updated Table Preemption Table of APIs of the ETH Driver (1/2). - Added VAR_PORT_BUFFER_2 Table RAM sections of the ETH Driver. 9. FLS: - In “9.9”: Remove information of “FLS_CALLOUT_CODE”. - In “9.2.2”, “9.4”: Remove information of Fls_Cbk.h. - In “9.11.1.2”: Add information about Det_ReportTransientFault. - In “9.11.2”: Remove information of Fee_JobEndNotification(void) and Fee_JobErrorNotification(void). - Add section 9.8 FLS Driver Component Transient Faults. - Add Fls_Compare in Table 9.7 Preemption Table of APIs of The FLS Driver. - Add Fls_Compare in Table 9.8 DET Errors of FLS Driver Component. - Add Fls_Compare in Table 9.9 DEM Error of FLS Driver Component. - Add Fls_Compare in Figure 9.4 Component Overview of FLS Driver Component. - Add Fls_Compare in Table 9.6 Function Definitions.

			- Change caption of Figure 9.5. 10. GPT: - Added "Read Predef timer values" under section 10.3 Architecture Detail. - Added "Read Predef timer values" for table 10.1 Supported Service List of GPT Module. - Update Gpt_ConfigType at section 10.5.2 Type Definition. - Added Gpt_GetPredefTimerValue API in section 10.5.3 Function Definition. - Added Gpt_GetPredefTimerValue API in section 10.5.4 Preemption of APIs. - Added Gpt_GetPredefTimerValue API in section 10.6.1 GPT Driver Component Development Errors. - At section 10.6.1 GPT Driver Component Development Errors, added new error message GPT_E_PARAM_PREDEF_TIMER. - Update Table 10.11 Runtime Errors of GPT Driver Component at section 10.7 GPT Driver Component Runtime Errors to support GPT_E_MODE error message. 11. ICCOM: Section 11.6.2: Add new DEM Errors CDDICCOM_E_INTERRUPT_CONTROLLER_FAILURE 13. IIC: - Replace “Dem_ReportErrorStatus” by “Dem_SetEventStatus”. - Replace “CddIicHWChannelSelection” by “CddIicHWChannelSelect”. - Replace “CddIicRasingTime” by “CddIicRisingTime”. - Replace “CddIicClockfrequency” by “CddIicClockFrequency”. 14. SPI: - Remove “Not Supported” at Spi_SetAsyncMode feature - Add V4H interrupt handling usage at chapter 14.10.4 16.THS: Chapter 16.8.1, table 16.18: Add CddThs_ConfigureThermalInterruption to Related API of Error Code CDD_THS_E_INVALID_PARAM 17. IPMMU: - Add paramter CddIpmmuUnintendedInterruptCheck in section 17.4.2 - Add Dem error ID CDDIPMMU_E_INTERRUPT_CONTROLLER_FAILURE in section 17.8.2 - Add the relationship information between the registers IMRAM0ERRCTR0, IMRAM0ERRCTR1 and the IPMMU API in section 17.4.1 - Add the relationship information between the registers IMRAM0ERRCTR0, IMRAM0ERRCTR1 and the configuration paramter CddIpmmuEdcErrorDetect in section 17.4.2 19. RFSO: - In “19.2.1”: Dem_ReportErrorStatus is updated to Dem_SetEventStatus. - In “19.8.2”: Add information of DEM error for RFSO Driver Device Component. 20. CRC: - Update overview, general, and preconditions for WCRC, and command function

			- Add CDDCRC_CH_STOPPED to CddCrc_ChannelSttType - Update struct CddCrc_ConfigType, CddCrc_WcrcChannelConfigType, CddCrc_ChannelConfigType - Add CddCrc_ModeType, CddCrc_PortType, CddCrc_DmaConfigType, CddCrc_DmaUnitConfigType - Add CddCrc_Write, CddCrc_ReadStatus, and CddCrc_Command to RENESAS API - Update range of LddChannelID from 0..7 to 0..15 - Add function definitions of CddCrc_Write, CddCrc_ReadStatus, and CddCrc_Command - Update preemption table for new API - Add new Dem error code CDDCRC_E_HARDWARE_ERROR - Update HW channel mapping - Add ISR functions - Section 20.2.1 Add a note to indicate that AES-ACC mode needs to be enabled by user. - Update overview for stop, and compare function - Add CDD_Crc_Cbk.h - Add ucCmdDmaIndex, and pCompareEndNotification to struct CddCrc_ChannelConfigType - Add struct CddCrc_ChannelStateType - Add information for CddCrc_Stop, CddCrc_Compare, and CddCrc_ReadCompareResult - Add new error CDDCRC_E_INCOMPATIBLE_HWIP, CDDCRC_E_PARAM_COMPARE_FREQ - Change return type of CddCrc_Write, and CddCrc_Command from CddCrc_ReturnType tp Std_ReturnType
0.05	May 27, 2022	—	5.CAN: - Remove CAN_CONTROLLERm_TX_ISR, CAN_CONTROLLERm_TX_CAT2_ISR out of related APIs of DEM error CAN_E_INT_INCONSISTENT at section “CAN Driver Component Production Errors” - Update ISR handler Address to add CAN_CHANNEL_ISR and CAN_CHANNEL_CAT2_ISR at section ISR Functions 7. PORT: - Section 7.8 Memory Organization: Correct memory section name. - Supported API: Port_SetToAlternateMode and Port_SetToDioMode 8.ETH: - Added information for AVB2 in CUM. - Added ETH_AVB2_CH00_ISR to ETH AVB2_CH24_ISR for HW ETH Channel Mapping. - Added AVB2 to table ETH Driver Protected Resource List. - Added AVB2 to table Limitation of Communication List. 9.FLS: - In "9.4 FLS Driver Component Header and Source File Description", T.B.D. are updated. - In "9.10.1.2 Det", "Std_ReturnType Det_ReportRuntimeError" is added. - In "9.10.1.4 Dem", Dem_ReportErrorStatus is updated to Dem_SetEventStatus. 10.GPT: - At section GPT Driver Component Production Errors, Function Definitions, Preemption of APIs: Update information support new APIs: Gpt_CheckWakeup,

			Gpt_EnableWakeup and Gpt_DisableWakeup. 12.MCU: - Section: Architecture Details Add MCU Reduced Power Modes Activated for V4H device - Section: Type Defenition. Add Pointer to MCU Mode Setting configuration for V4H device - Table: APIs Provided by the MCU Driver Component Add Mcu_SetMode function. - Table: Preemption Table of APIs of the MCU Driver: Add Mcu_SetMode function. 15.WDG: - Supported new function Wdg_Cbk_GptNotification. 18.EMM: - At section Type Definitions: Remove type CddEmm_ControlCounterType 17. IPMMU: - Supported APIs: CddIpmmu_MmuSetMode , CddIpmmu_MmuSetMemAttr, CddIpmmu_MmuSetTransTableBase, CddIpmmu_MmuEnable, CddIpmmu_MmuDisable, CddIpmmu_MmuFlush, CddIpmmu_PmbSet, CddIpmmu_PmbEnable, CddIpmmu_PmbDisable, CddIpmmu_MmuIsEnable, CddIpmmu_MmuGetMode, CddIpmmu_MmuGetMemAttr, CddIpmmu_MmuGetTransTableBase. 18. EMM: - In section EMM Complex Device Driver Component Production Errors: Add new error code CDDEMM_E_UNINTENDED_INTERRUPT_CHECK in table DEM Errors of EMM Complex Device Driver Component 20.CRC: - Change critical section name from GLOBALVAR_PROTECTION, REGISTER_PROTECTION to RAM_DATA_PROTECTION, INTERRUPT_CONTROL_PROTECTION. - Update struct CddCrc_ConfigType: add element pChannelConfig, remove elements pCrcChannelConfig, and pKcrcChannelConfig. - Update elements of struct CddCrc_CrcModuleConfigType, CddCrc_KcrcModuleConfigType - Add new struct CddCrc_ChannelConfigType.
0.04	Apr 25, 2022	—	5.CAN Remove below sections: - 1.2.4 User Mode and Supervisor Mode - 1.2.5 Deviation List (moved to Generic/Appendix) - 1.2.6 Limitations - 1.4 Relationship with Register - 1.5 Interaction Between the User and CAN Driver Component - 1.10.1.3 Translation Header File - 1.10.1.4 Parameter Definition File - 1.10.2 Compiler, Linker, and Assembler - 1.10.3 Sample Application - 1.10.4 Memory and Execution Time for S4(G4MH, CR52) Move section 1.6 CAN Driver Component Header and Source File Description to 1.4 Move Section 1.7 Application Programming Interface to 1.5 Move Section 1.8 Development and Production Errors to 1.6

			Add new section 1.7 CAN Driver Component Runtime Errors Move section 1.9 Memory Organization to 1.8 Move section 1.10 Device-Specific Information up to 1.9 Add new section 1.10 Non-AUTOSAR environment integration 6.DIO: - Supported APIs: Dio_ReadChannelGroup, Dio_WriteChannelGroup, Dio_MaskedWritePort, Dio_ReadChannelOutputValue and Dio_ReadChannelGroupOutputValue 7. PORT: - Supported API: Port_RefreshPortDirection 8.ETH: - In ETH Driver Error List table: modify "ETH_AVBnCH00ISR -> ETH_AVBnCh24ISR" to "ETH_AVBn_CH00_ISR -> ETH_AVBn_CH24_ISR". - Added subcomponents information of ETH driver in section Architecture Details: - Updated “Writing MII Interface Register” feature to correct mode support. - Supported “Frame Transmission Interrupt handling” function. - Supported “Frame Reception Interrupt handling” function. - Supported “Providing Transmit Buffer Access” function. - In DEM Errors of ETH Driver Component table: modify "ETH_AVBnDATAISR, ETH_AVBnDATAISR_CAT2, ETH_AVBnERRISR, ETH_AVBnERRISR_CAT2, ETH_AVBnMNGISR, ETH_AVBnMNGISR_CAT2, ETH_AVBnSTAISR, ETH_AVBnSTAISR_CAT2" to "ETH_AVBn_CH00_ISR -> ETH_AVBn_CH24_ISR", "ETH_AVBn_CH00_ISR_CAT2 -> ETH_AVBn_CH24_ISR_CAT2". - Updated “Name of the ISR Function” table to provide full interrupt function. - Update all table number format for ETH AVB. 10.GPT: - Supported APIs: Gpt_GetTimeElapsed and Gpt_GetTimeRemaining. - Remove redundant the critical section GPT_WRITE_PROTECTION 11.ICCOM: Remove below sections: - User Mode and Supervisor Mode - Deviation List - Relationship with register - Interaction Between the User and ICCOM Complex Device Driver Component - Translation Header File - Parameter Definition File - Compiler, Linker, and Assembler - Sample Application - Memory and Execution Time for V4H Change section numbers of the following sections: - ICCOM Complex Device Driver Component Header and Source File Description - Application Programming Interface - Development and Production Errors - Memory Organization - Device-Specific Information Add new section ‘ICCOM Driver Component Runtime Errors’

			Add new section ‘Non-AUTOSAR environment integration’ 15.WDG: - Supported API: Wdg_SetTriggerCondition 16.THS: - Supported APIs: CddThs_SetThermalInterruptionMode, CddThs_ConfigureThermalInterruption, CddThs_GetCurrentTemperature, CddThs_GetCurrentVoltage, CddThs_ClearTemperatureErrorStatus - Update all table number and title 17.IPMMU: - Supported APIs: CddIpmmu_MicroTlbSet, CddIpmmu_MicroTlbFlush, CddIpmmu_PmbIsEnable, CddIpmmu_MicroTlbGet and CddIpmmu_PmbGet 18.EMM: - Supported APIs: CddEmm_Init, CddEmm_GetVersionInfo, CddEmm_ReadErrorStatus, CddEmm_SetTarget, CddEmm_SupportPseudoError, CddEmm_SetPseudoErrorSignal, CddEmm_ClearPseudoErrorSignal, CddEmm_ClearErrorStatus, CddEmm_GetCurrentErrorCountUpValue
0.03	Mar 28, 2022	—	Update all features information of all V4H modules.
0.02	Jan 28, 2022	—	Update all features information of DIO, PORT, SPI, IIC, ICCOM.
0.01	Nov 24, 2021	—	First Edition issued

R-Car V4H AUTOSAR R19-11 MCAL User's Manual
Driver Component
Embedded User's Manual

Publication Date: Rev.3.00 Mar 27, 2025

Published by: Renesas Electronics Corporation

R-Car V4H AUTOSAR R19-11 MCAL
User's Manual

Renesas Electronics Corporation